The functions erf and erfc computed with arbitrary precision and explicit error bounds

Sylvain Chevillard

Laboratoire de l'Informatique du Parallélisme

École Normale Supérieure de Lyon
Unité Mixte de Recherche CNRS-INRIA-ENS LYON-UCBL n° 5668

The functions erf *and* erfc *computed with arbitrary precision*

Sylvain Chevillard

January 2009 - May 2010

Research Report N° RR2009-04

École Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, France
Téléphone : +33(0)4.72.72.80.37
Télécopieur : +33(0)4.72.72.80.80
Adresse électronique : `lip@ens-lyon.fr`

The functions erf and erfc computed with arbitrary precision

Sylvain Chevillard

January 2009 - May 2010

Abstract

The error function erf is a special function. It is widely used in statistical computations for instance, where it is also known as the standard normal cumulative probability. The complementary error function is defined as $\operatorname{erfc}(x) = 1 - \operatorname{erf}(x)$.

In this paper, the computation of $\operatorname{erf}(x)$ and $\operatorname{erfc}(x)$ in arbitrary precision is detailed: our algorithms take as input a target precision t' and deliver approximate values of $\operatorname{erf}(x)$ or $\operatorname{erfc}(x)$ with a relative error guaranteed to be bounded by $2^{-t'}$.

We study three different algorithms for evaluating erf and erfc. These algorithms are completely detailed. In particular, the determination of the order of truncation, the analysis of roundoff errors and the way of choosing the working precision are presented. The scheme used for implementing erf and erfc and the proofs are expressed in a general setting, so they can directly be reused for the implementation of other functions.

We have implemented the three algorithms and studied experimentally what is the best algorithm to use in function of the point x and the target precision t'.

Keywords: Error function, complementary error function, erf, erfc, floating-point arithmetic, arbitrary precision, multiple precision.

Résumé

La fonction d'erreur erf est une fonction spéciale. Elle est couramment utilisée en statistiques par exemple. La fonction d'erreur complémentaire est définie comme $\mathrm{erfc}(x) = \mathrm{erf}(x) - 1$.

Dans cet article, nous détaillons l'évaluation de $\mathrm{erf}(x)$ et $\mathrm{erfc}(x)$ en précision arbitraire : nos algorithmes prennent en entrée une précision cible t' et fournissent en retour une valeur approchée de $\mathrm{erf}(x)$ ou $\mathrm{erfc}(x)$ avec une erreur relative bornée par $2^{-t'}$.

Nous étudions trois algorithmes différents pour évaluer erf et erfc. Ces algorithmes sont expliqués en détails. En particulier, nous décrivons comment déterminer l'ordre de troncature, comment analyser les erreurs d'arrondi et comment choisir la précision intermédiaire de travail. Le schéma employé pour l'implantation de erf et erfc est expliqué en des termes généraux et peut donc être directement réutilisé pour l'implantation d'autres fonctions.

Nous avons implémenté les trois algorithmes ; nous concluons par une étude expérimentale qui montre quel algorithme est le plus rapide en fonction du point x et de la précision cible t'.

Mots-clés: fonctions d'erreur, erf, erfc, arithmétique flottante, précision arbitraire, multiprécision

1 Introduction

The error function, generally denoted by erf is defined as

$$\operatorname{erf} : x \mapsto \frac{2}{\sqrt{\pi}} \int_0^x e^{-v^2} \, \mathrm{d}v.$$

Sometimes it is called the *probability integral* [12], in which case, erf denotes the integral itself without the normalization factor $2/\sqrt{\pi}$. The complementary error function denoted by erfc is defined as $\operatorname{erfc} = 1 - \operatorname{erf}$. These two functions are defined and analytic on the whole complex plane. Nevertheless we will consider them only on the real line herein.

These functions are important because they are encountered in many branches of applied mathematics, in particular probability theory. Namely, if X is a Gaussian random variable with mean 0 and standard deviation $1/\sqrt{2}$, the probability $P(-x \leq X \leq x)$ is equal to $\operatorname{erf}(x)$ (Figure 1). See [12] for instance for other applications.

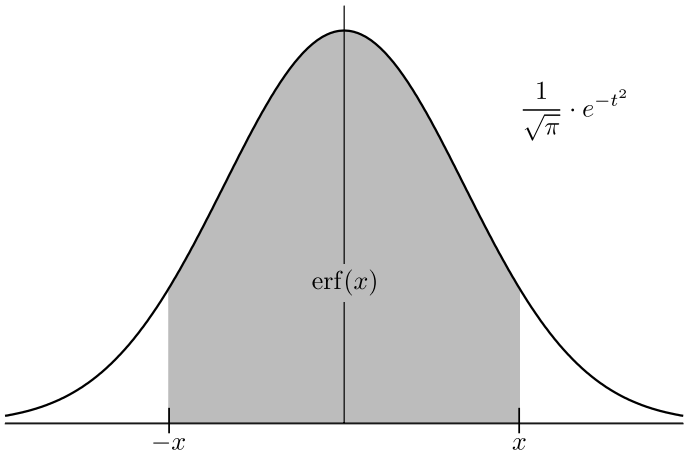


Figure 1: $\operatorname{erf}(x)$ is the probability that a certain Gaussian random variable lies in $[-x, x]$

In this article we describe the numerical implementation of erf and erfc in floating-point arithmetic with arbitrary precision. Such an arbitrary precision implementation is useful in several cases including:

- when highly accurate values of the functions are needed;
- for testing the quality of lower precision libraries;
- for building good approximating polynomials with a given accuracy.

A good overview of the applications of arbitrary precision is given in the introduction of [5].

We approximate the real numbers by floating-point numbers with arbitrary precision with radix 2; more precisely: let $t \in \mathbb{N}^\star$, the set of floating-point numbers with precision t is the set

$$\mathcal{F}_t = \{0\} \cup \left\{ \pm \frac{m}{2^t} \, 2^e,\ e \in \mathbb{Z},\ m \in [\![2^{t-1},\ 2^t - 1]\!] \right\} \quad \text{where} \quad [\![2^{t-1},\ 2^t - 1]\!] = [2^{t-1},\ 2^t - 1] \cap \mathbb{Z}.$$

The integer e is called the exponent of the floating-point number. For practical reasons, it is usually bounded in the implementation. However, in general, in multiple precision libraries,

its range is extremely large (typically $e \in [-2^{32}; 2^{32}]$) and may be considered as practically almost unbounded. We will assume in this paper that e is unbounded. The number $m/2^t$ is called the mantissa and is more conveniently written as $0.1b_2 \dots b_t$ where $b_2, \dots, b_t$ are the bits of its binary representation. The mantissa lies in the interval $[1/2, 1)$: this is the convention used by the `MPFR` library and we will adopt it here. Note that the IEEE-754 standard takes another convention and suppose that the mantissa lies in $[1, 2)$.

Our goal is the following: given t' in $\mathbb{N}^\star$ and x a floating-point number, compute a value y approximating $\operatorname{erf}(x)$ with a relative error less than $2^{-t'}$. More formally, we would like to compute y such that

$$\exists \delta \in \mathbb{R},\ y = \operatorname{erf}(x)(1+\delta) \quad \text{where} \quad |\delta| \leq 2^{-t'}.$$

Arbitrary precision floating-point arithmetic is already implemented in several software tools and libraries. Let us cite Brent's historical Fortran `MP` package [4], Bailey's `Arprec` C++ library [2], the `MPFR` library [9], and the famous Mathematica and Maple tools. `MPFR` provides correct rounding of the functions: four rounding-modes are provided (rounding upwards, downwards, towards zeros, and to the nearest) and the returned value y is the rounding of the exact value $\operatorname{erf}(x)$ to the target precision t'. Other libraries and tools usually only ensure that the relative error between y and the exact value is $\mathcal{O}(2^{-t'})$.

Our implementation guarantees that the final relative error be smaller than $2^{-t'}$. This is stronger than $\mathcal{O}(2^{-t'})$, since the error is explicitly bounded.

Since we provide an explicit bound on the error, our implementation can also be used to provide correct rounding, through an algorithm called Ziv's onion strategy [15]: in order to obtain the correct rounding in precision t', a first approximate value is computed with a few guard bits (for instance 10 bits). Hence $|\operatorname{erf}(x) - y| \leq 2^{-t'-10} \operatorname{erf}(x)$. Most of the time, it suffices to round y to t' bits in order to obtain the correct rounding of $\operatorname{erf}(x)$. But in rare cases, this is not true and another value y must be computed with more guard bits. The precision is increased until the correct rounding can be decided. This is the strategy used by `MPFR` for instance. Providing correct rounding does not cost more in average than guaranteeing an error smaller than $2^{-t'}$: the average complexity of Ziv' strategy is the complexity of the first step, i.e. when only 10 guard bits are used.

For our implementation, we use classical formulas for approximating erf and erfc and we evaluate them with a summation technique proposed by Smith [14] in 1989. As we already mentioned, some implementations of erf and erfc are already available. However:

- None of them uses Smith's technique. This technique allows us to have the most efficient currently available implementation.

- In general they use only one approximating method. In this article, we study three different approximating formulas and show how to heuristically choose the most efficient one, depending on the target precision and the evaluation point.

- Most of the available implementations do not guarantee an explicit error bound. The `MPFR` library actually does give an explicit error bound. However, only few details are given on the implementation of functions in `MPFR`. The only documentation on `MPFR` is given by [9] and by the file `algorithm.pdf` provided with the library. In general, there is no more than one or two paragraphs of documentation per function.

- More generally, to our best knowledge, there does not exist any reference article describing in detail how to implement a function in arbitrary precision, with guaranteed error bounds. In this article, we explicitly explain how to choose the truncation rank of each formula, how to perform the roundoff analysis and how to choose the working precision used for the intermediate computations. Through the detailed example of the functions erf and erfc, this article gives a general scheme that can be followed for the implementation of other functions.

Our implementation is written in C and built on top of `MPFR`. It is distributed under the LGPL and it is available with this research report at `http://prunel.ccsd.cnrs.fr/ensl-00356709/`.

The outline of the paper is as follows: Section 2 begins with a general overview of the scheme used to evaluate functions in arbitrary precision by means of series. The section continues with a general discussion about erf and erfc and the ways of computing them with arbitrary precision. In particular, the algorithm used for the evaluation of the series is detailed. Section 3 is devoted to some reminders on classical techniques required for performing the roundoff analysis. In Section 4 the algorithms are completely described and the roundoff analysis is detailed. However, for the sake of clarity, proofs are omitted in that section. They are all given in an appendix at the end of the article. Section 5 gives experimental results. In particular, it gives timings comparing our implementation with `MPFR` and Maple. It also experimentally shows what is the best approximating method, depending on the evaluation point and the target precision. Finally, Section 6 is devoted to the conclusion and presents future works.

2 General overview of the algorithms

2.1 General scheme used for the evaluation in arbitrary precision

In order to evaluate a function f in arbitrary precision, it is necessary to have approximating formulas that can provide values arbitrarily close to the exact value $f(x)$. We refer to [5] for a general overview of the classical techniques in this domain. In the following, we shall restrict to the case when a power series or an asymptotic series is used.

Let us assume for instance that $f(x) = \sum_{n=0}^{+\infty} a_n\, x^n$. Two important questions must be addressed:

- In practice, only a finite number N of terms are summed (so the truncation rank is $N-1$). We must choose N in such a way that the approximation error be small enough.
- Floating-point arithmetic is used when evaluating the sum. This implies that rounding errors affect the final result. In order to keep this evaluation error small enough, the working precision should be carefully chosen.

In particular, it is important to distinguish the *target precision* (i.e. the value t' such that we eventually want to return an approximate value y satisfying $|y - f(x)| \leq 2^{-t'}\,|f(x)|$) and the *working precision* (i.e. the precision used to perform the computations). In the following, we will always denote by t the working precision.

We adopt the following strategy for the implementation of f:

1. Find a rough overestimation $N-1$ of the truncation rank.

2. Rigorously bound the roundoff errors. This allows us to choose a suitable working precision.

3. Evaluate the series. An *on-the-fly* criterion allows us to stop the evaluation as soon as possible. So, in practice, the finally used truncation rank $N^\star - 1$ is near-optimal.

A few remarks are necessary to understand this strategy. First, we remark that the roundoff errors depend on several things: the series used (the values a_n may be more or less complicated to evaluate), the evaluation scheme used to evaluate the series, and the number of operations performed. This is why we need to have an overestimation $N-1$ of the truncation rank *before* we choose the working precision. However, this estimation does not need to be very accurate, since (as we will see) only the logarithm of N matters for the roundoff errors. Of course, in practice we do not want to perform more operations than what is strictly necessary: this is why we use an on-the-fly stopping criterion at step 3.

We want that the final absolute error be smaller than $2^{-t'}\,|f(x)|$. In practice, we cut this error into two equal parts: we choose the truncation rank and the working precision in such a way that both the approximation error and the evaluation error be smaller than $2^{-t'-1}\,|f(x)|$. The overall error is bounded by the sum of both errors, which gives the desired bound.

Since we want to ensure that the value N be an overestimation, it must be determined with care. It is generally possible to bound the remainder of order N, in function of x: i.e. we usually explicitly know a function ε_N such that

$$\forall x,\ \left| f(x) - \sum_{n=0}^{N-1} a_i\, x^i \right| \leq \varepsilon_N(x).$$

Hence it suffices to choose N such that $\varepsilon_N(x) \leq 2^{-t'-1}\,|f(x)|$. The problem is that we do not know $|f(x)|$ yet. Thus, for finding N we must address two problems:

- Find an easily computable underestimation $g(x)$ of $|f(x)|$.
- Find N such that we are sure that $\varepsilon_N(x) \leq 2^{-t'-1}\,g(x)$. This implies inverting the function $N \mapsto \varepsilon_N(x)$.

The scheme that we just described is general and can be used for the implementation of other functions. Let us sum up the questions that must be addressed:

1. Find one (or several) series $\sum_{n=0}^{+\infty} a_n\, x^n$ approximating $f(x)$. It can also be an asymptotic development.

2. Choose an algorithm for the evaluation of this series.

3. Perform a roundoff analysis of this algorithm. This gives an error bound that depends on the truncation rank.

4. Find a good underestimation $g(x)$ of $|f(x)|$.

5. Find a good bound $\varepsilon_N(x)$ of the remainder of the series, and invert the relation $\varepsilon_N(x) \leq 2^{-t'-1}\,g(x)$ in function of N.

Provided that these questions find an answer, our technique applies. In the following, we will answer these questions in the case when $f = \mathrm{erf}$ or $f = \mathrm{erfc}$. More precisely:

- We give three approximating series for erf and erfc in the next Section 2.2.
- We describe evaluation algorithms in Section 2.3. These algorithms are not specific to erf or erfc and can be reused in other contexts. The actual algorithms used for evaluating erf and erfc are described in Section 4 and more precisely in Figures Algorithm 3, Algorithm 4 and Algorithm 5.
- We show how to perform a roundoff analysis in Section 3. Again, this section is general and can be reused in other contexts. The specific roundoff analyses of our implementation are given in Propositions 4, 6 and 8.
- We give explicit underestimations of $|\operatorname{erf}(x)|$ and $|\operatorname{erfc}(x)|$ in the appendix in Lemma 9.
- We give explicit bounds for the remainder of the series that we use for the implementation of erf and erfc and show how to invert them. This gives us a rough but rigorous overestimation of the necessary truncation rank for each approximation formula. These results are summed up in Recipes 1, 3 and 5.
- Finally, we use our roundoff analyses together with the estimations of the truncation ranks, in order to choose a suitable working precision t for each approximation formula. These results are summed up in Recipes 2, 4 and 6.

2.2 Approximation formulas for erf and erfc

It is easy to see that erf is odd. Thus we restrict to computing $\operatorname{erf}(x)$ when $x > 0$ without loss of generality. Except if it is explicitly mentioned, x will always be positive in the following. Moreover, $\operatorname{erf}(0) = 0$ and $\operatorname{erf}(x)$ approaches 1 as $x \to +\infty$. Thus for large x, the binary representation of $\operatorname{erf}(x)$ looks like

$$0.\underbrace{11\ldots11}_{\text{many 1s}}b_1b_2b_3\ldots$$

This is why, for large x, it is more convenient to consider the complementary error function $\operatorname{erfc}(x) = 1 - \operatorname{erf}(x)$. The graphs of these two functions are represented in Figure 2.

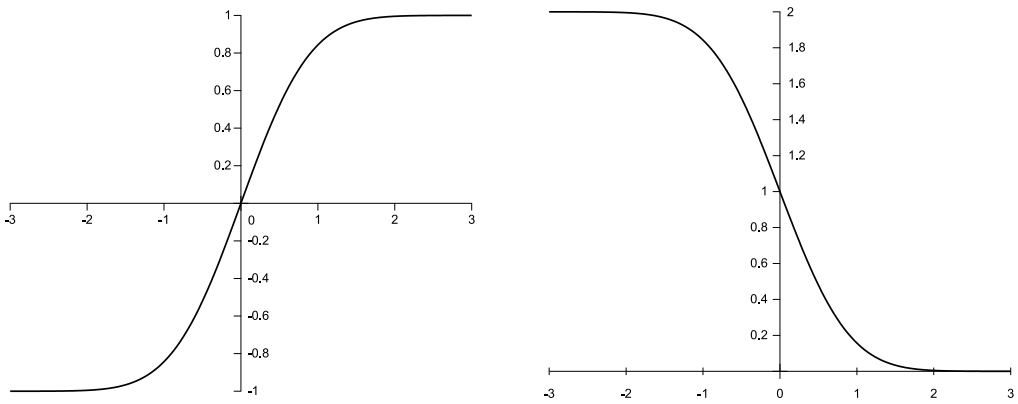
Among the formulas given in [1] we retain the following ones, suited for the computation in arbitrary precision (Equations 7.1.5, 7.1.6, 7.1.23 and 7.1.24 of [1]):

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

$$\operatorname{erf}(x) = \frac{2xe^{-x^2}}{\sqrt{\pi}}\sum_{n=0}^{+\infty}\frac{(2x^2)^n}{1\cdot3\cdot5\cdots(2n+1)}, \tag{2}$$

$$\operatorname{erfc}(x) = \frac{e^{-x^2}}{x\sqrt{\pi}}\left(1+\sum_{n=1}^{N-1}(-1)^n\,\frac{1\cdot3\cdot5\cdots(2n-1)}{(2x^2)^n}\right)+\varepsilon_N^{(3)}(x) \tag{3}$$

$$\text{where } |\varepsilon_N^{(3)}(x)| \le \frac{e^{-x^2}}{x\sqrt{\pi}}\cdot\frac{1\cdot3\cdot5\cdots(2N-1)}{(2x^2)^N}. \tag{4}$$



(a) Graph of erf (b) Graph of erfc

Figure 2: Graphs of the functions erf and erfc

These formula can also be rewritten, with a more compact form, using the symbol ${}_1F_1$ of hypergeometric functions:

$$ {}_1F_1(a,\, b;\, x) = \sum_{i=0}^{+\infty} \frac{a\,(a+1)\,\ldots\,(a+n-1)\,x^n}{b\,(b+1)\,\ldots\,(b+n-1)\,(n!)}. $$

So, we have

$$ \operatorname{erf}(x) = \frac{2x}{\sqrt{\pi}}\ {}_1F_1\left(\frac{1}{2},\, \frac{3}{2};\, -x^2\right) \tag{1'} $$

$$ \text{and}\ \operatorname{erf}(x) = \frac{2xe^{-x^2}}{\sqrt{\pi}}\ {}_1F_1\left(1,\, \frac{3}{2};\, x^2\right). \tag{2'} $$

Equation (1) is mostly interesting for small values of x. The series is alternating and the remainder is thus bounded by the first neglected term. The ratio between two consecutive terms is, roughly speaking, x^2/n. Thus, if $x < 1$, both x^2 and n contribute to reduce the magnitude of the term and the convergence is really fast. For larger values of x, the convergence is slower since only the division by n ensures that the term decreases. Though, the main problem with large arguments is not the speed of the convergence.

The main drawback of (1) for large arguments comes from the fact that the series is alternating: the sum is henceforth ill-conditioned and accuracy is lost during the evaluation. This is due to a phenomenon usually called *catastrophic cancellation* [8].

Definition 1 (Cancellation)**.** *Let a and b be two numbers. When subtracting b from a, we say that there is a cancellation of bits when the leading bits of a and b cancel out. In this case, only the last (few) bits of a and b contribute to the result. Thus, the accuracy of the result can be much smaller than the accuracy of a and b.*

Due to this cancellation phenomenon, the computations must be performed with a precision much higher than the target precision. We will quantify this phenomenon in Section 4.2: as we will see, as x increases, the use of Equation (1) quickly becomes impractical.

Equation (2) does not exhibit cancellations. Its evaluation does not require much more precision than the target precision. However, the convergence is a bit slower. Besides, it

requires to compute e^{-x^2} (the complexity of computing it is somewhat the same as computing erf itself).

Equation (3) gives a very efficient way of evaluating erfc and erf for large arguments. However $\varepsilon_N^{(3)}(x)$ cannot be made arbitrarily small by increasing N (there is an optimal value reached when $N = \lfloor x^2 + 1/2 \rfloor$). If erfc$(x)$ is to be computed with a bigger precision, one has to switch back to Equation (1) or (2).

2.3 Evaluation scheme

2.3.1 Particular form of the sums

The three sums described above exhibit the same general structure: they are polynomials or series (in the variable x^2, $2x^2$ or $1/(2x^2)$) with coefficients given by a simple recurrence involving multiplications or divisions by integers. More formally, they can all be written as

$$S(y) = \alpha_0 + \alpha_0\alpha_1\, y + \ldots + \alpha_0\alpha_1 \cdots \alpha_{N-1}\, y^{N-1}. \tag{5}$$

Namely, we have

- in the case of Equation (1), $y = -x^2$, $\alpha_0 = \dfrac{2x}{\sqrt{\pi}}$ and for $i \geq 1$, $\alpha_i = \dfrac{2i-1}{(2i+1)\,i}$;
- in the case of Equation (2), $y = 2x^2$, $\alpha_0 = \dfrac{2xe^{-x^2}}{\sqrt{\pi}}$ and for $i \geq 1$, $\alpha_i = \dfrac{1}{2i+1}$;
- in the case of Equation (3), $y = \dfrac{-1}{2x^2}$, $\alpha_0 = \dfrac{e^{-x^2}}{x\sqrt{\pi}}$ and $\alpha_i = 2i - 1$.

It is important to remark that the values α_i are integers (or inverse of integers) that fit in a 32-bit or 64-bit machine integer. The multiplication (or division) of a t-bit floating-point number by a machine integer can be performed [4] in time $\mathcal{O}(t)$. This should be compared with the multiplication of two t-bit floating-point numbers which is performed in time $\mathcal{O}(t\log(t)\log(\log(t)))$ asymptotically (and which is only quadratic for small precisions). Additions and subtractions of two t-bit floating-point numbers are also done in time $\mathcal{O}(t)$.

2.3.2 Straightforward algorithm

A natural way of evaluating the sum (5) is to successively compute the terms of the sum, while accumulating them in a variable (see Algorithm 1).

With this algorithm the following operations are performed: N additions, N multiplications by the small integers α_{i+1}, and N full-precision multiplications by y. The total cost of the algorithm is henceforth dominated by the full-precision multiplications: the complexity is $\mathcal{O}(N\,M(t))$ where $M(t)$ denotes the cost of a full-precision multiplication in precision t.

2.3.3 Concurrent Series Algorithm

We may take advantage of the particular structure of the sum (5), using a technique described by Smith [14] as a *concurrent series summation* (after an idea of Paterson and Stockmeyer [13]).

Input: y, N, t
Output: the sum $S(y)$ of Equation (5)
/* each operation is performed in precision t */
1 R $\leftarrow 0$;
2 acc $\leftarrow \alpha_0$;
3 **for** i $= 0$ **to** N $- 1$ **do**
4 | R $\leftarrow$ R + acc;
5 | acc $\leftarrow$ acc $*$ $\alpha_{\mathsf{i}+1}$ $*$ y;
6 **end**
7 **return** R;

Algorithm 1: StraightforwardAlgo()

Let L be an integer parameter. For the sake of simplicity, we assume that N is a multiple of L. The general case is easy to deduce from this particular case. Smith remarks that $S(y)$ may be expressed as follows:

$$\begin{aligned} S(y) \;=\; 1 &\cdot (\ \alpha_0 + \alpha_0\cdots\alpha_L(y^L) + \ \ldots\ + \alpha_0\cdots\alpha_{N-L}(y^L)^{N/L-1}\ ) \\ + y &\cdot (\ \alpha_0\alpha_1 + \alpha_0\cdots\alpha_{L+1}(y^L) + \ \ldots\ + \alpha_0\cdots\alpha_{N-L+1}(y^L)^{N/L-1}) \\ + \ &\ldots \\ + y^{L-1} &\cdot (\ \alpha_0\cdots\alpha_{L-1} + \alpha_0\cdots\alpha_{2L-1}(y^L) + \ \ldots\ + \alpha_0\cdots\alpha_{N-1}(y^L)^{N/L-1}\ ). \end{aligned} \tag{6}$$

We denote by S_i the sum between the parentheses of the i-th line of this array: thus

$$S(y) = S_0 + S_1\, y + \cdots + S_{L-1}\, y^{L-1}.$$

The sums S_i are computed concurrently: a variable is used to successively compute

$$\alpha_0, \quad \alpha_0\alpha_1, \quad \ldots \quad , \quad \alpha_0\cdots\alpha_{L-1}, \quad \alpha_0\cdots\alpha_L(y^L), \quad \text{etc.}$$

and the sums S_i are accumulated accordingly. The power y^L is computed once in the beginning by binary exponentiation, with at most $\log(L)$ full-precision multiplications. When computing the coefficients, the multiplications (or divisions) involved are all multiplications by machine integers, except the multiplications by y^L that occur $N/L - 1$ times.

Finally, the polynomial $S(y) = S_0 + S_1\, y + \cdots + S_{L-1}\, y^{L-1}$ is evaluated by Horner's rule and requires $L - 1$ high-precision multiplications.

The complete algorithm requires $\log(L) + N/L + L - 2$ full-precision multiplications (and $N+L-1$ additions and N multiplications/divisions by machine integers). The optimal value is obtained with $L \simeq \sqrt{N}$. The total cost is then approximately $2\sqrt{N}$ slow multiplications. The corresponding complexity is $\mathcal{O}\left(\sqrt{N}M(t) + Nt\right)$, which is strictly better than the complexity of the straightforward evaluation algorithm. We note that this method requires extra space to store the values S_i until they are used in the final Horner evaluation (Lt bits are needed). The algorithm is summed up in Figure Algorithm 2.

2.3.4 Stopping criterion in the concurrent series algorithm

In Figure Algorithm 2, the algorithm is described as if N were known in advance. However, as we explained in Section 2.1, it is preferable to stop the computation with help of an on-the-fly stopping criterion, in order to avoid useless computations.

```
   Input: y, L, N, t
   Output: the sum S(y) of Equation (5)
   /* each operation is performed in precision t                          */
 1 z ← power(y, L) ;                    /* obtained by binary exponentiation */
 2 S ← [0, ..., 0] ;                    /* array of L floating-point numbers */
 3 acc ← 1 ;
 4 i ← 0 ;                             /* indicates the S_i currently updated */
 5 for k ← 0 to N − 1 do
 6 |   acc ← acc ∗ α_k ;
 7 |   S[i] ← S[i] + acc ;
 8 |   if i = L − 1 then
 9 |   |   i ← 0 ;
10 |   |   acc ← acc ∗ z ;
11 |   else
12 |   |   i ← i + 1 ;
13 |   end
14 end
   /* now S(y) is evaluated from the S_i by Horner's rule                 */
15 R ← S[L − 1] ;
16 for i ← L − 2 downto 0 do
17 |   R ← S[i] + y ∗ R ;
18 end
19 return R;
```

Algorithm 2: ConcurrentSeries()

In Figure Algorithm 2, the terms of Equation (6) are computed successively beginning by the first column (from top to bottom), following by the second column (from top to bottom), etc. The variable k denotes the term currently being computed, while the variable i represents the corresponding line. The variable acc is used to store the current term: just after the line 6 of the algorithm was performed, $(\mathsf{acc} \cdot y^i)$ is an approximation to the k-th term of the polynomial $S(y)$.

This means that N does not really need to be known in advance: a test of the form *sum the terms until finding one whose absolute value is smaller than a given bound* can be used. Of course, $(\mathsf{acc} \cdot y^i)$ is only an approximation of the actual coefficient. If we are not careful enough, we might underestimate the absolute value and stop the summation too soon. Hence, the rounding mode should be chosen carefully when updating acc, in order to be sure to always get an overestimation of the absolute value.

3 Error analysis

Since the operations in Algorithm 2 are performed with floating-point arithmetic, they are not exact and roundoff errors must be taken into account. We have to carefully choose the precision t that is used for the computations in order to keep the roundoff errors small enough. Techniques make it possible to bound such roundoff errors rigorously. In his book [10], Higham explains in great details what should be known in this domain. We recall a few facts here, without proofs; see [10] for details.

Definition 2. *If $x \in \mathbb{R}$, we denote by $\diamond(x)$ a rounding of x (i.e. x itself if x is a floating-point number and one of the two floating-point numbers enclosing x otherwise).*

If t denotes the current precision, the quantity $u = 2^{1-t}$ is called the unit roundoff.
We will use the convenient notation $\widehat{z} = z\,\langle k\rangle$ meaning that*

$$\exists \delta_1, \ldots, \delta_k \in \mathbb{R},\ s_1, \ldots, s_k \in \{-1, 1\},\ \textit{such that}\ \widehat{z} = z \prod_{i=1}^{k} (1+\delta_i)^{s_i} \quad \textit{with}\ |\delta_i| \leq u.$$

This notation corresponds to the accumulation of k successive relative errors. The following proposition justifies it:

Proposition 1. *For any $x \in \mathbb{R}$, there exists $\delta \in \mathbb{R}$, $|\delta| \leq u$ such that $\diamond(x) = x\,(1+\delta)$.*

We now recall a few rules for manipulating error counter.

Rule 1. *If $k' \geq k$ and if we can write $\widehat{z} = z\,\langle k\rangle$, we can also write $\widehat{z} = z\,\langle k'\rangle$.*

Rule 2. *If we can write $\widehat{x} = x\,\langle k_1\rangle\,\langle k_2\rangle$ then we can write $\widehat{x} = x\,\langle k_1 + k_2\rangle$.*

Rule 3. *If $\oplus$ denotes the correctly rounded addition, the following holds:*

$$\forall (x,y) \in \mathbb{R}^2,\ x \oplus y = \diamond(x+y) = (x+y)(1+\delta) \quad \textit{for a given}\ |\delta| \leq u.$$

Hence we can write $x \oplus y = (x+y)\,\langle 1\rangle$.
The same holds for the other correctly rounded operations $\ominus$, $\otimes$, $\oslash$, etc.

*According to Higham [10], this notation has been introduced by G. W. Stewart under the name of *relative error counter*.

Rule 4. *If we can write* $\widehat{z} = (x+y)\langle k\rangle$ *then we can also write* $\widehat{z} = x\langle k\rangle + y\langle k\rangle$. *Note that the reciprocal is false in general.*

Let us do a complete analysis on a simple example. Consider a, b, c, d, e f six floating-point numbers. We want to compute $S = a\,b + c\,d\,e + f$. In practice, we may compute for instance $\widehat{S} = ((a\otimes b)\oplus((c\otimes d)\otimes e))\oplus f$. We can analyze the roundoff errors with the following simple arguments:

$$\begin{array}{rcccccccl}
\widehat{S} & = & ((a\otimes b) & \oplus & ((c\otimes d)\otimes e)) & \oplus & f & & \\
& = & ((ab)\langle 1\rangle & \oplus & (cde)\langle 2\rangle) & \oplus & f & & \text{(Rule 3 applied to } \otimes \text{ and Rule 2)} \\
& = & ((ab)\langle 2\rangle & \oplus & (cde)\langle 2\rangle) & \oplus & f & & \text{(Rule 1)} \\
& = & ((ab)\langle 2\rangle & + & (cde)\langle 2\rangle\,)\langle 1\rangle & \oplus & f & & \text{(Rule 3 applied to } \oplus\text{)} \\
& = & ((ab)\langle 3\rangle & + & (cde)\langle 3\rangle\,) & \oplus & f & & \text{(Rule 4 and Rule 2)} \\
& = & (ab)\langle 4\rangle & + & (cde)\langle 4\rangle & + & f\langle 4\rangle & & \text{(Rules 3, 4, 2 and 1).}
\end{array}$$

We now need a proposition to bound the error corresponding to a given value of the error counter:

Proposition 2. *Let* $z\in\mathbb{R}$ *and let* $\widehat{z}$ *be a floating-point number. We suppose that* $k\in\mathbb{N}$ *satisfies* $ku<1$ *and that we can write* $\widehat{z} = z\langle k\rangle$. *Then*

$$\exists\theta_k\in\mathbb{R},\ \textit{such that } \widehat{z} = z(1+\theta_k) \quad \textit{with } |\theta_k|\le\gamma_k = \frac{ku}{1-ku}.$$

In particular, as soon as $ku\le 1/2$, $\quad\gamma_k\le 2ku$.

Using this proposition, we can write $\widehat{S} = (a\,b)(1+\theta_4) + (c\,d\,e)(1+\theta'_4) + f(1+\theta''_4)$. Finally, we get

$$\left|\widehat{S}-S\right| \le \gamma_4\ (|a\,b| + |c\,d\,e| + |f|)\,.$$

Note the importance of $\widetilde{S} = |a\,b|+|c\,d\,e|+|f|$. If the terms of the sum are all non-negative, $S=\widetilde{S}$ and the relative error for the computation of the sum is bounded by γ_4. If some terms are negative, the relative error is bounded by $\gamma_4\widetilde{S}/S$. The ratio $\widetilde{S}/S$ may be extremely large: this quantifies the phenomenon of cancellation, when terms of the sum cancel out while the errors accumulate.

4 Practical implementation

4.1 General scheme

We now describe the practical details of the implementation of the three formulas for the evaluation of erf and erfc. For each formula, we follow the same general scheme, described in Section 2.1.

We first derive a bound $\varepsilon_N(x)$ of the remainder in function of N and x. By (rigorously) inverting this bound in function of N, we obtain an estimation of the necessary truncation rank $N-1$. This estimation is rough, but it is surely overestimated.

In a second time, we perform an error analysis of the algorithm (similar to the example presented in Section 3) and get a bound on the final roundoff error. It will typically be of the form

$$|S-\widehat{S}| \le \gamma_{aN}\widetilde{S}$$

where a is an integer (the order of magnitude of a is 1 to 10 approximately), S is the exact sum, $\widehat{S}$ is the computed value and $\widetilde{S}$ is the sum of the absolute values. Therefore it is sufficient to choose t such that $(2aN)2^{1-t}\,\widetilde{S} \leq 2^{-t'-1}\,g(x)$ (where g is a lower approximation of $|\operatorname{erf}(x)|$ or $|\operatorname{erfc}(x)|$). Equivalently this leads to

$$t \geq t' + 3 + \log_2(a) + \log_2(N) + \log_2(\widetilde{S}) - \log_2(g(x)).$$

Finally, we evaluate the sum using the algorithm described in Figure Algorithm 2. The parameter L is chosen as $L = \sqrt{N}$. When evaluating the sum, an on-the-fly stopping criterion is used. This allows us to stop the computation with an (almost) optimal number $N^\star$ of terms.

The fact that N is a rough estimation of $N^\star$ is not a severe issue. Suppose for instance that N is an overestimation of $N^\star$ by a factor 4. By choosing $L \simeq \sqrt{N} = \sqrt{4N^\star}$ we will eventually perform $N^\star/L + L \simeq 5\sqrt{N^\star}/2$ slow multiplications, which is only 25% greater than the optimal number $2\sqrt{N^\star}$. Moreover, since only the logarithm of N is useful for choosing the working precision t, an overestimation by a factor 4 will only lead to use 2 extra bits, which is insignificant.

We shall now give the details of the implementation of Equations (1), (2) and (3). In order to ease the description of the implementation, we report all technical lemmas and all the proofs in an appendix at the end of the article. They are not mandatory to understand the main ideas of what follows.

4.2 Practical implementation of Equation (1)

Here, we assume that Equation (1) is used to obtain an approximate value of $\operatorname{erf}(x)$ (and we suppose, without loss of generally, that $x > 0$):

$$\operatorname{erf}(x) = \frac{2x}{\sqrt{\pi}} \left(\sum_{n=0}^{N-1} (-1)^n \frac{(x^2)^n}{(2n+1)n!} \right) + \varepsilon_N^{(1)}(x)$$

where $\varepsilon_N^{(1)}(x)$ is the remainder.

We first express a relation that ensures that $\varepsilon_N^{(1)}(x)$ is smaller than $2^{-t'-1}\operatorname{erf}(x)$ (remember that t' is the target precision, given as an input).

Proposition 3. *Let E be the exponent of x. If N satisfies*

$$\frac{N}{ex^2} \log_2\left(\frac{N}{ex^2}\right) \geq \frac{t' + \max(0,\, E)}{ex^2},$$

the remainder $\varepsilon_N^{(1)}$ is bounded by $2^{-t'-1}\operatorname{erf}(x)$.

We then estimate N in function of the inputs t' and x. This estimation is obtained by rigorously inverting the equation given in Proposition 3. This leads to the following recipe:

Recipe 1. *We first evaluate $a = (t' + \max(0,\, E))/(ex^2)$. In fact, when performing the evaluation, the rounding modes are carefully chosen in such a way that one gets an overestimation a_u of a: $a_u \geq a$. Then, N is chosen according to the following recipe:*

– *if* $a_u \geq 2$,	N	$\geq$	$(ex^2)\, 2a_u/\log_2(a_u)$;
– *if* $a_u \in [0,\, 2]$,	N	$\geq$	$(ex^2)\, 2^{1/4}\, 2^{a_u/2}$.

These formulas are evaluated with appropriate rounding modes, in order to ensure that N is really greater than the actual value.

Once an overestimation $N-1$ of the truncation rank is known, we need to choose a working precision t. This precision depends on the errors that will be accumulated during the evaluation. So, we first sketch the details of the evaluation.

We compute

$$S(x) = \sum_{n=0}^{N-1} (-1)^n \frac{2}{\sqrt{\pi}} \cdot \frac{x^{2n+1}}{(2n+1)\,n!}$$

using Algorithm 2 with parameters $y = x^2$, $\alpha_0 = 2x/\sqrt{\pi}$ and for $k \geq 1$, $\alpha_k = \dfrac{-1}{k} \cdot \dfrac{2k-1}{2k+1}$. In the following, the variables acc, i, k, etc. are the variables introduced in Algorithm 2 on page 9.

In practice, we do not compute the ratio $(2k-1)/(2k+1)$. In the product $\alpha_0 \cdots \alpha_k$, these ratios simplify in cascade. In fact, we define a sequence (β_n) by $\beta_0 = \alpha_0$ and $\beta_k = 1/k$. Hence,

$$\alpha_0 \cdots \alpha_k = (-1)^k \beta_0 \cdots \beta_k \, \frac{1}{2k+1}.$$

We use the variable acc to compute $(y^L)^{\lfloor k/L \rfloor} \beta_0 \cdots \beta_k$ and we use a temporary variable tmp for the division by $2k+1$. S_i is updated by alternatively adding or subtracting tmp (instead of acc).

In the beginning, $y = x^2$ and $z = y^L$ are computed with rounding upwards. When computing α_0, the rounding modes are chosen in such a way that the computed value is greater than the exact value $2x/\sqrt{\pi}$. The variables acc and tmp are also updated with rounding upwards. Hence, the following always holds:

$$\mathsf{tmp} \cdot y^i \gtrsim \frac{2}{\sqrt{\pi}} \cdot \frac{x^{2k+1}}{(2k+1)k!}.$$

Let F be the exponent of y: $y < 2^F$. Using the fact that $\operatorname{erf}(x) \geq x/2$ when $x < 1$ and $\operatorname{erf}(x) \geq 1/2$ when $x \geq 1$ (cf. Lemma 9, in the appendix), it is easy to show that we can stop the loop as soon as

$$k \geq N \quad \text{or} \quad \mathsf{tmp} \cdot 2^{Fi} < 2^{-t' + \min(E-1,\,0) - 2}.$$

The complete algorithm is summed up in Figure Algorithm 3.

The roundoff errors are bounded using the following proposition.

Proposition 4. *If Algorithm 3 is used to compute an approximation $\widehat{S(x)}$ of the sum $S(x)$, the following holds:*

$$\widehat{S(x)} = \sum_{n=0}^{N-1} (-1)^n \frac{2x}{\sqrt{\pi}} \cdot \frac{x^{2n}}{(2n+1)\,n!} \langle 8N \rangle .$$

Thus

$$\left| \widehat{S(x)} - S(x) \right| \leq \gamma_{8N} \left(\frac{2x}{\sqrt{\pi}} \sum_{n=0}^{N-1} \frac{x^{2n}}{(2n+1)\,n!} \right) \leq \gamma_{8N} \frac{2}{\sqrt{\pi}} \int_0^x e^{v^2} \, \mathrm{d}v.$$

The bound γ_{8N} could be made tighter. However, we cannot hope a better value than γ_N since we do $\mathcal{O}(N)$ operations. Only the logarithm of this value will be of interest for choosing the working precision t. By working more carefully, we would not get more than replacing $\log(8N)$ by $\log(N)$ and it would not be of any practical benefit.

```
   Input: a floating-point number x,
          the working precision t,
          the target precision t',
          L ∈ ℕ*, N ∈ ℕ*.
   Output: an approximation of erf(x) with relative error less than 2^{-t'} obtained using
           Equation (1)
   /* each operation is performed in precision t                                   */
 1 y ← x * x ; // rounded upwards
 2 F ← exponent(y) ;
 3 if x < 1 then G ← exponent(x) − 1 else G ← 0 ;
 4 z ← power(y, L) ; // computed with rounding upwards
 5 S ← [0, ..., 0] ;
 6 acc ← √π ; // rounded downwards
 7 acc ← 2 * x/acc ; // rounded upwards
 8 i ← 0 ;
 9 k ← 0 ;
10 tmp ← acc ;
11 repeat
12 |   if (k mod 2) = 0 then S[i] ← S[i] + tmp else S[i] ← S[i] − tmp ;
13 |   k ← k + 1;
14 |   if i = L − 1 then
15 |   |   i ← 0 ;
16 |   |   acc ← acc * z ; // rounded upwards
17 |   else
18 |   |   i ← i + 1 ;
19 |   end
20 |   acc ← acc/k ; // rounded upwards
21 |   tmp ← acc/(2 * k + 1) ; // rounded upwards
22 until k = N or exponent(tmp) < G − t' − 2 − F * i ;
   /* now S(y) is evaluated from the S_i by Horner's rule                          */
23 R ← S[L − 1] ;
24 for i ← L − 2 downto 0 do
25 |   R ← S[i] + y * R ;
26 end
27 return R;
```

Algorithm 3: erfByEquation1()

This proposition allows us to choose an appropriate working precision t: the greater t is, the smaller γ_{8N} is. In practice, it suffices to use the following recipe:

Recipe 2. *Let E be the exponent of x. For the evaluation of* $\operatorname{erf}(x)$ *by Equation (1), an appropriate working precision t is*

– *when* $x < 1$,	$t \geq t' + 9 + \lceil \log_2 N \rceil$;
– *when* $x \geq 1$,	$t \geq t' + 9 + \lceil \log_2 N \rceil - E + x^2 \log_2(e)$.

In practice $\log_2(e)$ *is replaced by a value precomputed with rounding upwards. The factor* $x^2 \log_2(e)$ *that appears when* $x > 1$ *highlights the fact that the series is ill-conditioned for large values of* x.

4.3 Practical implementation of Equation (2)

Here, we assume that Equation (2) is used to obtain an approximate value of $\operatorname{erf}(x)$ (again we suppose that $x > 0$):

$$\operatorname{erf}(x) = \frac{2xe^{-x^2}}{\sqrt{\pi}} \left(\sum_{n=0}^{N-1} \frac{(2x^2)^n}{1 \cdot 3 \cdot 5 \cdots (2n+1)} \right) + \varepsilon_N^{(2)}(x)$$

where $\varepsilon_N^{(2)}(x)$ is the remainder.

We follow the same method as for Equation (1). The relation between N and t' is given by the following proposition:

Proposition 5. *Let E be the exponent of x. If N satisfies $N \geq 2x^2$ and*

$$\frac{N}{ex^2} \log_2 \left(\frac{N}{ex^2} \right) \geq \frac{t' + 3 + \max(0,\, E) - x^2 \log_2(e)}{ex^2}$$

the remainder is bounded by $\operatorname{erf}(x)\, 2^{-t'-1}$.

In order to compute an overestimation of N, we proceed as we did for the implementation of Equation (1). We first evaluate

$$a = \frac{t' + 3 + \max(0, E) - x^2 \log_2(e)}{ex^2}.$$

As in the previous section, the rounding modes are chosen carefully in order to get an overestimation a_u of the actual value a.

Recipe 3. *We deduce the formulas for computing an overestimation of N:*

– *if* $a_u \geq 2$,	*take* $N \geq (ex^2)\, 2a_u / \log_2(a_u)$;
– *if* $a_u \in [0,\, 2]$,	*take* $N \geq (ex^2)\, 2^{1/4}\, 2^{a_u/2}$;
– *if* $a_u < 0$,	*let* $N_0 \geq (ex^2)\, 2^{a_u}$ *and perform the following test:*
	– *if* $N_0 \geq 2x^2$, *take* $N = N_0$,
	– *else take* $N = \lceil 2x^2 \rceil$.

Only the case $a_u < 0$ could lead to a value N smaller than $2x^2$. If it is the case, we take $N = \lceil 2x^2 \rceil$, in order to ensure the hypothesis of Proposition 5.

Input: a floating-point number x,
the working precision t,
the target precision t',
$\mathsf{L} \in \mathbb{N}^\star$, $\mathsf{N} \in \mathbb{N}^\star$.
Output: an approximation of $\mathrm{erf}(x)$ with relative error less than $2^{-t'}$ obtained using Equation (2)

/* each operation is performed in precision t */
$\mathsf{y} \leftarrow 2 * \mathsf{x} * \mathsf{x}$; // rounded upwards
$\mathsf{E} \leftarrow \mathtt{exponent}(\mathsf{x})$;
$\mathsf{F} \leftarrow \mathtt{exponent}(\mathsf{y})$;
if $x < 1$ **then** $\mathsf{G} \leftarrow \mathsf{E} - 1$ **else** $\mathsf{G} \leftarrow 0$;
$\mathsf{z} \leftarrow \mathtt{power}(\mathsf{y}, \mathsf{L})$; // computed with rounding upwards
$\mathsf{S} \leftarrow [0, \ldots, 0]$;
$\mathsf{acc} \leftarrow \sqrt{\pi}$; // rounded downwards
$\mathsf{acc} \leftarrow 2 * \mathsf{x}/\mathsf{acc}$; // rounded upwards
$\mathsf{tmp} \leftarrow \mathsf{x} * \mathsf{x}$; // performed in precision $\mathsf{t} + \max(2\mathsf{E}, 0)$, rounded downwards
$\mathsf{tmp} \leftarrow \exp(-\mathsf{tmp})$; // rounded upwards
$\mathsf{acc} \leftarrow \mathsf{acc} * \mathsf{tmp}$; // rounded upwards
$\mathsf{i} \leftarrow 0$;
$\mathsf{k} \leftarrow 0$;
repeat
  $\mathsf{S}[\mathsf{i}] \leftarrow \mathsf{S}[\mathsf{i}] + \mathsf{acc}$;
  $\mathsf{k} \leftarrow \mathsf{k} + 1$;
  if $\mathsf{i} = \mathsf{L} - 1$ **then**
    $\mathsf{i} \leftarrow 0$;
    $\mathsf{acc} \leftarrow \mathsf{acc} * \mathsf{z}$; // rounded upwards
  else
    $\mathsf{i} \leftarrow \mathsf{i} + 1$;
  end
  $\mathsf{acc} \leftarrow \mathsf{acc}/(2 * \mathsf{k} + 1)$; // rounded upwards
until $\mathsf{k} = \mathsf{N}$ **or** $((\mathsf{k} \geq \mathsf{y})$ **and** $(\mathtt{exponent}(\mathsf{acc}) < \mathsf{G} - \mathsf{t}' - 3 - \mathsf{F} * \mathsf{i}))$;
/* now $S(y)$ is evaluated from the S_i by Horner's rule */
$\mathsf{R} \leftarrow \mathsf{S}[\mathsf{L} - 1]$;
for $\mathsf{i} \leftarrow \mathsf{L} - 2$ **downto** 0 **do**
  $\mathsf{R} \leftarrow \mathsf{S}[\mathsf{i}] + \mathsf{y} * \mathsf{R}$;
end
return R;

Algorithm 4: erfByEquation2()

We evaluate the sum

$$S(x) = \sum_{n=0}^{N-1} \frac{2xe^{-x^2}}{\sqrt{\pi}} \cdot \frac{(2x^2)^n}{1 \cdot 3 \cdots (2n+1)}$$

using Algorithm 2 with parameters $y = 2x^2$, $\alpha_0 = 2xe^{-x^2}/\sqrt{\pi}$ and for $k \geq 1$, $\alpha_k = 1/(2k+1)$. As in the implementation of Equation (1), we use upward rounding and a test for stopping the loop as soon as possible. In this case, the criterion becomes

$$k \geq N \quad \text{or} \quad \left(k \geq 2x^2 \ \text{ and } \ \mathsf{acc} \cdot 2^{Fi} < 2^{-t'+\min(E-1,0)-3}\right).$$

The complete algorithm is summed up in Figure Algorithm 4.

The roundoff errors are bounded using the following proposition.

Proposition 6. *If Algorithm 4 is used to compute an approximation* $\widehat{S(x)}$ *of the sum* $S(x)$*, the following holds:*

$$\widehat{S(x)} = \sum_{n=0}^{N-1} \frac{2xe^{-x^2}}{\sqrt{\pi}} \cdot \frac{(2x^2)^n}{1 \cdot 3 \cdots (2n+1)} \langle 16N \rangle .$$

Thus

$$\left|\widehat{S(x)} - S(x)\right| \leq \gamma_{16N}\, S(x) \leq \gamma_{16N}\, \mathrm{erf}(x).$$

Finally, an appropriate precision t is given by the following recipe:

Recipe 4. *When Equation (2) is used to evaluate* $\mathrm{erf}(x)$*, an appropriate working precision* t *is given by*

$$\boxed{t \geq t' + 7 + \lceil \log_2 N \rceil .}$$

4.4 Implementation of erfc using Equation (1) or (2)

In this section, we do not suppose that $x > 0$ anymore.

Since $\mathrm{erfc}(x) = 1 - \mathrm{erf}(x)$, we can use the previous algorithms to evaluate erfc. However, we have to take care of two things:

- firstly, the approximation of $\mathrm{erf}(x)$ should be computed with an appropriate relative error 2^{-s}. Since $\mathrm{erf}(x)$ and $\mathrm{erfc}(x)$ do not have the same order of magnitude, 2^{-s} has no reason to be the same as the target relative error $2^{-t'}$;
- secondly, contrary to erf, erfc is not odd (nor even). In particular, $\mathrm{erfc}(-x)$ and $\mathrm{erfc}(x)$ do not have the same order of magnitude and this should be considered when estimating the relative error.

We evaluate $\mathrm{erfc}(x)$ in two steps: first we compute an approximation R of $\mathrm{erf}(x)$ with a relative error less than a bound 2^{-s} (this is performed with one of the previous algorithms). Then, we compute $1 \ominus R$ with precision $t' + 3$.

Lemma 1. *If* s *is chosen according to the following recipe,* $|R - \mathrm{erf}(x)| \leq 2^{-t'-1}\, \mathrm{erfc}(x)$.

– *If* $x \leq -1$,	$s \geq t' + 1$;
– *if* $-1 < x < 0$,	$s \geq t' + 2 + E$;
– *if* $0 \leq x < 1$,	$s \geq t' + 5 + E$;
– *if* $x \geq 1$,	$s \geq t' + 3 + E + x^2 \log_2(e)$.

As a consequence of the lemma,

$$|(1-R) - \operatorname{erfc}(x)| \leq 2^{-t'-1} \operatorname{erfc}(x).$$

It follows that $|(1-R)| \leq 2 \operatorname{erfc}(x)$. Now, since $(1 \ominus R) = (1-R)\langle 1 \rangle$,

$$\big|(1 \ominus R) - (1-R)\big| \;\leq\; |(1-R)|\, 2^{1-(t'+3)} \;\leq\; 2^{-t'-1} \operatorname{erfc}(x).$$

Finally $\big|(1 \ominus R) - \operatorname{erfc}(x)\big| \leq \big|(1 \ominus R) - (1-R)\big| + \big|(1-R) - \operatorname{erfc}(x)\big| \leq 2^{-t'} \operatorname{erfc}(x)$ which proves that $(1 \ominus R)$ is an approximation of $\operatorname{erfc}(x)$ with a relative error bounded by $2^{-t'}$.
Important remark: in the case when $-1 < x < 0$ and when $t' + 2 + E \leq 1$, it is actually not necessary to perform any computation: in this case, 1 is an approximation to $\operatorname{erfc}(x)$ with a relative error less than $2^{-t'}$. More precisely, we have $|x| \leq 2^E \leq 2^{-t'-1}$, so

$$|1 - \operatorname{erfc}(x)| = |\operatorname{erf}(x)| = |\operatorname{erf}(|x|)|.$$

We can bound $|\operatorname{erf}(|x|)|$ by $2x$ (see Lemma 9 in Appendix). Moreover, since $x < 0$, $\operatorname{erfc}(x) > 1$. So we have $|1 - \operatorname{erfc}(x)| \leq 2^{-t'} \operatorname{erfc}(x)$.

The same remark holds in the case when $0 < x < 1$ and $t' + 5 + E \leq 1$.

4.5 Practical implementation of Equation (3)

We now show how to use Equation (3) for obtaining an approximate value of $\operatorname{erfc}(x)$ (we suppose again that $x > 0$):

$$\operatorname{erfc}(x) = \frac{e^{-x^2}}{x\sqrt{\pi}} \left(1 + \sum_{n=1}^{N-1} (-1)^n \frac{1 \cdot 3 \cdot 5 \cdots (2n-1)}{(2x^2)^n} \right) + \varepsilon_N^{(3)}(x)$$

where $\varepsilon_N^{(3)}(x)$ is the remainder, bounded thanks to Inequality (4).

The particularity of this formula comes from the fact that the remainder cannot be made arbitrarily small. In fact, x being given, the bound (4) is first decreasing until it reaches an optimal value with $N = \lfloor x^2 + 1/2 \rfloor$. For larger values of N, it increases. Hence, given a target relative error $2^{-t'}$, it may be possible that no value of N is satisfying. In this case, Equation (3) cannot be used. We note in particular that this formula is useless for $0 < x < 1$.
Until the end of this section, we will suppose that $x \geq 1$.

Conversely, when the relative error can be achieved, we can choose any value of N between two values $N_{\min}$ and $N_{\max}$. Obviously, we are interested in the smallest one.

Proposition 7. *If N satisfies*

$$\frac{N}{ex^2} \log_2 \left(\frac{N}{ex^2} \right) \leq \frac{-t'-3}{ex^2}$$

the remainder is bounded by $\operatorname{erfc}(x)\, 2^{-t'-1}$.

Using this proposition, it is possible to compute an overestimation N of $N_{\min}$. However, it is possible that this estimation N could be even greater than $N_{\max}$. In order to avoid this possibility, let us remark two things: firstly, we never need to consider values N greater than x^2 (since we know the bound given by Equation (4) is minimal for $N = \lfloor x^2 - 1/2 \rfloor$) and, secondly, given a candidate value N, it is always possible to check if the inequality of Proposition (7) holds.

Recipe 5. *The rule for computing N is the following: first, we compute $a = (-t' - 3)/ex^2$ choosing the rounding modes for obtaining a underestimation a_d of a and then we choose N according to the following recipe:*

- *if $a_d < -\log_2(e)/e$, Equation (3) cannot be used;*
- *else let $N_0 \geq (ex^2)\, a_d / \log_2(-a_d)$ and perform the following test:*
  - *if $N_0 \leq x^2$, let $N = N_0$,*
  - *else let $N_1 \simeq x^2$ and perform the following test:*
    - *if $\dfrac{N_1}{ex^2} \log_2\left(\dfrac{N_1}{ex^2}\right) \leq a_d$, let $N = N_1$,*
    - *else, Equation (3) cannot be used.*

Of course the value $-\log_2(e)/e$ is evaluated in such a way that we get an overestimation of the actual value. Also, in the test $N_0 \leq x^2$, the value x^2 is replaced by an underestimation.

If a suitable value N has been computed, we evaluate the sum

$$S(x) = \frac{e^{-x^2}}{x\sqrt{\pi}} + \sum_{n=1}^{N-1} (-1)^n \frac{e^{-x^2}}{x\sqrt{\pi}} \cdot \frac{1 \cdot 3 \cdots (2n-1)}{(2x^2)^n}$$

using Algorithm 2 with parameters $y = 1/(2x^2)$, $\alpha_0 = e^{-x^2}/(x\sqrt{\pi})$ and for $k \geq 1$, $\alpha_k = -(2k-1)$. In practice, we use $\alpha_k = 2k-1$ and we alternatively add and subtract acc to the partial sum (again the variables acc, i, k, etc. are the variables introduced in Algorithm 2 on page 9).

When computing α_0, the rounding modes are chosen in such a way that the computed value is an upper bound for the actual value. Besides, when acc is updated, rounding upwards is used. Hence, we can stop the loop as soon as

$$k = N \quad \text{or} \quad \mathsf{acc} \cdot 2^{Fi} < 2^{-t'-1}\, e^{-x^2}/(4x)$$

where F is the exponent of y. The algorithm is summed up in Figure Algorithm 5.

The roundoff errors are bounded using the following proposition:

Proposition 8. *If Algorithm 5 is used to compute an approximation $\widehat{S(x)}$ of the sum $S(x)$, the following holds:*

$$\left|\widehat{S(x)} - S(x)\right| \leq \gamma_{16N} \frac{e^{-x^2}}{x\sqrt{\pi}} \cdot \frac{3}{2}.$$

Recipe 6. *Using this proposition, we obtain a suitable working precision t for computing* erfc(x) *with Equation (3):*

$$t \geq t' + 9 + \lceil \log_2(N) \rceil\,.$$

4.6 Implementation of erf with Equation (3)

We finish our study by briefly explaining how Equation (3) is used to compute erfc(x) when $x \leq -1$ and to compute erf(x) for $x \geq 1$ (the symmetrical case $x < -1$ is the same).

Input: a floating-point number x,
the working precision t,
the target precision t',
$\mathsf{L} \in \mathbb{N}^\star$, $\mathsf{N} \in \mathbb{N}^\star$.
Output: an approximation of erfc(x) with relative error less than $2^{-t'}$ obtained using Equation (3)

```
/* each operation is performed in precision t                    */
E ← exponent(x);
G ← ⌈x * x * log2(e)⌉ ; // computed with rounding upwards
acc ← x * x ; // performed in precision t + 2E, rounded downwards
y = 2 * acc;
y ← 1/y ; // rounded upwards
acc ← exp(−acc) ; // rounded upwards
tmp ← x * √π ; // rounded downwards
acc ← acc/tmp ; // rounded upwards
F ← exponent(y) ;
z ← power(y, L) ; // computed with rounding upwards
S ← [0, ..., 0] ;
i ← 0 ;
k ← 0 ;
repeat
    if (k mod 2) = 0 then S[i] ← S[i] + acc else S[i] ← S[i] − acc ;
    k ← k + 1;
    if i = L − 1 then
        i ← 0 ;
        acc ← acc * z ; // rounded upwards
    else
        i ← i + 1 ;
    end
    acc ← acc * (2 * k − 1) ; // rounded upwards
until k = N or exponent(acc) < −t' − 3 − F * i − G − E ;
/* now S(y) is evaluated from the S_i by Horner's rule           */
R ← S[L − 1] ;
for i ← L − 2 downto 0 do
    R ← S[i] + y * R ;
end
return R;
```

Algorithm 5: erfcByEquation3()

Note that $\mathrm{erfc}(x) = 1 - \mathrm{erf}(x) = 1 + \mathrm{erf}(-x) = 2 - \mathrm{erfc}(-x)$. When $x \leq -1$, we obtain $\mathrm{erfc}(x)$ by computing an approximation R of $\mathrm{erfc}(-x)$ with a relative error smaller than an appropriate bound 2^{-s} and computing $2 \ominus R$ in precision $t' + 3$.

The same way, since $\mathrm{erf}(x) = 1 - \mathrm{erfc}(x)$, we obtain $\mathrm{erf}(x)$ by computing an approximation R of $\mathrm{erfc}(x)$ with a relative error smaller than an appropriate bound 2^{-s} and computing $1 \ominus R$ in precision $t' + 3$.

The appropriate values for s are given in the two following lemmas.

Lemma 2. *If $x \geq 1$, the inequality $|R - \mathrm{erfc}(x)| \leq 2^{-t'-1}\,\mathrm{erfc}(-x)$ holds when s is chosen according to the following recipe:*

$$\boxed{s \geq t' + 2 - E - x^2 \log_2(e).}$$

Remark that whenever $t' + 2 - E - x^2 \log_2(e) \leq 1$, it is not necessary to compute an approximation of $\mathrm{erfc}(x)$ and to perform the subtraction: 2 can be directly returned as a result. Indeed, $t' + 2 - E - x^2 \log_2(e) \leq 1$ implies that $e^{-x^2}/x \leq 2^{-t'}$ and hence

$$\mathrm{erfc}(x) \leq 2^{-t'} \leq 2^{-t'}\,\mathrm{erfc}(-x).$$

Since $\mathrm{erfc}(x) = 2 - \mathrm{erfc}(-x)$, it means that 2 is an approximation of $\mathrm{erfc}(-x)$ with a relative error less than $2^{-t'}$.

Lemma 3. *If $x \geq 1$ the inequality $|R - \mathrm{erfc}(x)| \leq 2^{-t'-1}\,\mathrm{erf}(x)$ holds when s is chosen according to the following recipe:*

$$\boxed{s \geq t' + 3 - E - x^2 \log_2(e).}$$

The same remark holds: if $t' + 3 - E - x^2 \log_2(e) \leq 1$, the value 1 can be directly returned as an approximation of $\mathrm{erf}(x)$ with a relative error less than $2^{-t'}$.

5 Experimental results

We have given all the details necessary for implementing each of the three equations (1), (2), and (3). They can be used for obtaining approximate values of either $\mathrm{erf}(x)$ or $\mathrm{erfc}(x)$. We also gave estimations of the order of truncation $N - 1$ and of the working precision t. In each case $\mathcal{O}(\sqrt{N})$ multiplications at precision t and $\mathcal{O}(N)$ additions and multiplications/divisions by small integers are needed for evaluating the sum. Hence, for each equation, the binary complexity of the algorithm is $\mathcal{O}(\sqrt{N}\,M(t) + Nt)$ where $M(t)$ denotes the binary complexity of a product of two numbers of precision t.

However, quite different behaviors are hidden behind this complexity. Indeed, the inputs of our algorithms are the point x and the target precision t'. Hence, N and t are functions of x and t'. As can be seen in previous sections, these functions highly depend on the equation used to evaluate $\mathrm{erf}(x)$ or $\mathrm{erfc}(x)$.

5.1 Choosing the best equation

Of course, given an input couple (x, t'), we would like to automatically choose the equation to be used, in order to minimize the computation time. For this purpose, we need to compare the complexity of the three equations. It seems quite hard to theoretically perform such a comparison:

- firstly, the estimations of N and t are different for each equation. They depend on x and t' in a complicated way. Besides, they are defined piecewise, which implies that there are many cases to study;
- secondly, comparing the three methods requires to set some assumptions on the implementation of the underlying arithmetic (e.g. complexity of the multiplication; complexity of the evaluation of exp).

Usually, the multiplication between two numbers of precision t is performed differently whether t is large or not: see [9, Section 2.4] for an overview of what is used in `MPFR`. Three different algorithms are used in `MPFR`, and each one depends on the underlying integer multiplication. The `GMP` library, used to perform the integer multiplications, uses at least four different algorithms depending on the sizes of the input data. Hence, the effective complexity $M(t)$ is very hard to accurately estimate.

In `MPFR`, the evaluation of $\exp(x)$ is performed by three different algorithms, depending on the required precision t (see [9, end of Section 2.5] for a brief overview).

- A naive evaluation of the series is used when t is smaller than a parameter t_0;
- the concurrent series technique of Smith is used when t lies between t_0 and a second threshold t_1;
- finally, if $t \geq t_1$, a binary splitting method [3] is used.

The values t_0 and t_1 have been tuned (and are thus different) for each architecture.

This shows that choosing the best equation between the three equations proposed in this paper is a matter of practice and not of theoretical study. We implemented the three algorithms in C, using `MPFR` for the floating-point arithmetic. Our code is distributed under the LGPL and freely available at `http://prunel.ccsd.cnrs.fr/ensl-00356709/`.

In order to experimentally see which equation is the best for each couple (x, t'), we ran the three implementations for a large range of values x and t'. For each couple, we compared the execution time of each implementation when evaluating $\operatorname{erf}(x)$. The experimental results are summed up in Figure 3. The colors indicate which implementation is the fastest. The experiments were performed on a 32-bit 3.00 GHz Intel Pentium D with 2.00 GB of RAM running Linux 2.6.26 and `MPFR` 2.3.1 and gcc 4.3.3. The thresholds used by `MPFR` for this architecture are $t_0 = 528$ and $t_1 = 47\,120$.

The boundary between Equations (1) and (2) seems to exhibit three phases, depending on t'. These phases approximately match the thresholds used by `MPFR` for the implementation of exp. Since Equation (2) relies on the evaluation of $\exp(-x^2)$ whereas Equation (1) does not, this is probably not a coincidence and we just see the impact of the evaluation of $\exp(-x^2)$ on the whole computation time.

The boundary between Equations (2) and (3) is more regular. In fact, we experimentally observe that as soon as Equation (3) is usable, it is more interesting than the other equations. Figure 4 shows the timings for $t' = 632$ in function of x. This is a typical situation: for small values of x, Equation (3) cannot achieve the target precision; but for $x \simeq 15$, it becomes usable and is immediately five times faster than the others. Hence, the domain where Equation (3) should be used is given by the points where the inequality of Proposition 7 has a solution, i.e. if and only if

$$\frac{-s-3}{ex^2} \geq \frac{-\log_2(e)}{e},$$

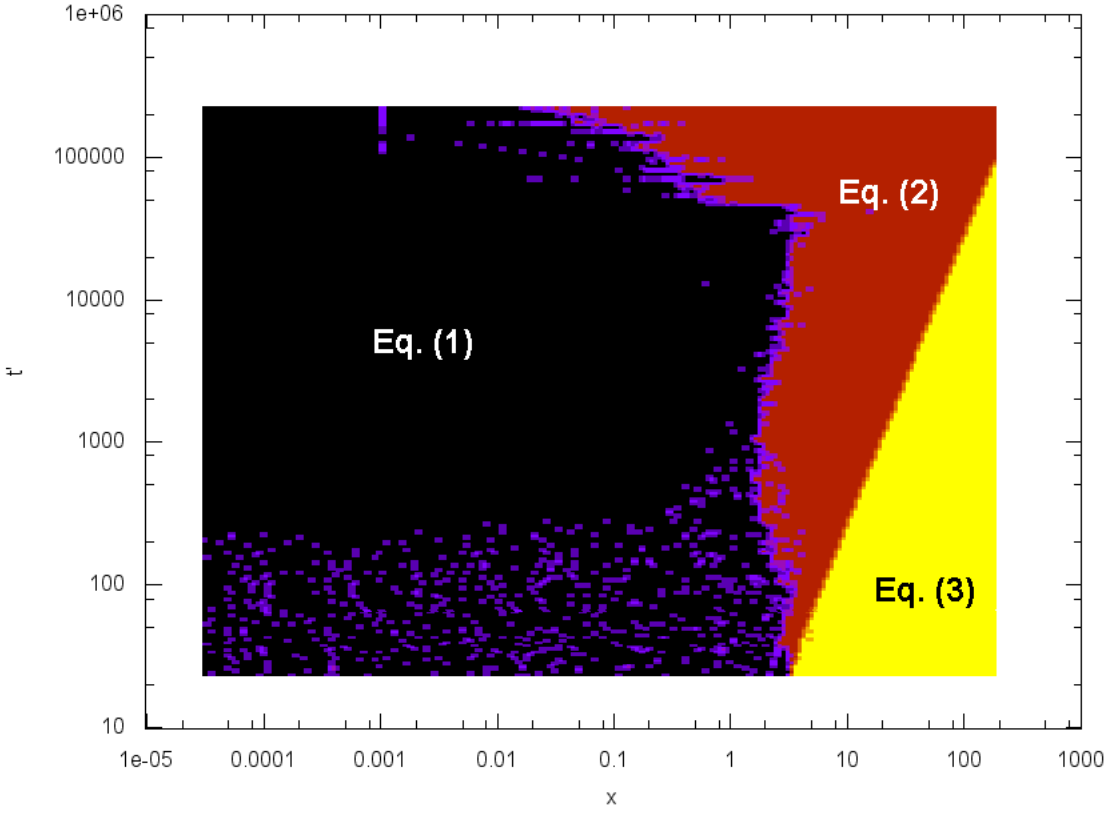


Figure 3: Comparison of the execution times of the three implementations of erf (the scale is logarithmic)

where $s \gtrsim t' + 3 - E - x^2 \log_2(e)$ is the intermediate precision given in Lemma 3. Thus the equation of the boundary is approximately $\log(t') \simeq 2\log(x)$, which corresponds to the observations.

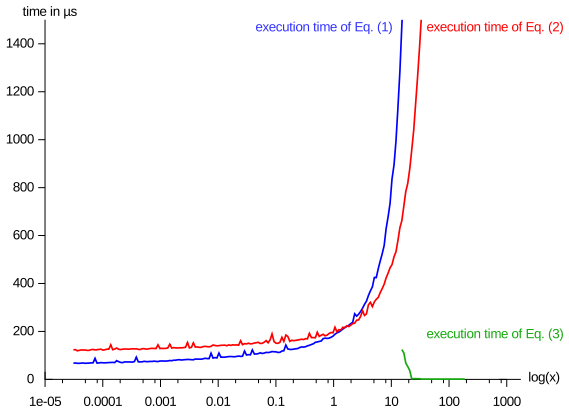


Figure 4: Execution times of the three implementations, in function of x, when $t' = 632$ (the x-axis has a logarithmic scale).

5.2 Comparison with other implementations

We compared our implementation of erf and erfc with two others reference implementations: `MPFR` and Maple. `MPFR` is probably the most relevant since we indeed compare two comparable things: since our implementation is written using `MPFR`, the underlying arithmetic is the same in both cases. Therefore, the difference of timings between our implementation and `MPFR` is completely due to the difference between the algorithms used.

Maple uses a decimal floating-point format. Besides, it is an interpreted language. Hence, the comparison between our implementation and Maple is not completely relevant. However, Maple is widely used and provides one of the rare implementations of erf and erfc in arbitrary precision.

The results of our experiments are given in table Figure 5. The experiments were performed on a 32-bit 2.40 GHz Intel Xeon with 2.00 GB of RAM running Linux 2.6.22. We used `MPFR` 2.3.1, gcc 4.1.2 and Maple 11. The values of x are chosen randomly with the same precision t' as the target precision. The table only indicates an approximate value. The target precision t' is expressed in bits. Maple is run with the variable Digits set to $\lfloor t'/\log_2(10) \rfloor$. This corresponds approximately to the same precision expressed in a decimal arithmetic. Maple remembers the values already computed. It is thus impossible to repeat the same evaluation several times for measuring the time of one single evaluation by considering the average timing. In order to overcome this difficulty we chose to successively evaluate $\mathrm{erf}(x)$, $\mathrm{erf}(x+h)$, $\mathrm{erf}(x+2h)$, etc. where h is a small increment.

The cases when Equation (3) cannot achieve the target precision are represented by the symbol "-". Our implementation is the fastest except in a few cases. In small precisions and small values of x, `MPFR` is the fastest because it uses a direct evaluation that is a bit faster when the truncation rank is small.

The case $x \simeq 88.785777$ and $t' = 7139$ has another explanation. Actually, the situation corresponds to the remark following Lemma 3: $t' + 3 - E - x^2 \log_2(e) \leq 1$. Hence, there is

x	t'	Eq. 1	Eq. 2	Eq. 3	MPFR	Maple
0.000223	99	29 µs	47 µs	-	**14 µs**	283 µs
0.005602	99	34 µs	48 µs	-	**17 µs**	282 µs
0.140716	99	40 µs	53 µs	-	**25 µs**	287 µs
3.534625	99	96 µs	**84 µs**	-	125 µs	382 µs
88.785777	99	277 520 µs	6181 µs	**< 1 µs**	2 µs	18 µs
0.000223	412	**55 µs**	87 µs	-	76 µs	739 µs
0.005602	412	**62 µs**	94 µs	-	104 µs	783 µs
0.140716	412	**88 µs**	109 µs	-	186 µs	870 µs
3.534625	412	246 µs	**198 µs**	-	663 µs	1 284 µs
88.785777	412	289 667 µs	9 300 µs	**< 1 µs**	2 µs	21 µs
0.000223	1 715	**311 µs**	562 µs	-	1 769 µs	2 513 µs
0.005602	1 715	**393 µs**	616 µs	-	2 490 µs	2 959 µs
0.140716	1 715	**585 µs**	748 µs	-	4 263 µs	3 968 µs
3.534625	1 715	1 442 µs	**1 260 µs**	-	11 571 µs	8 850 µs
88.785777	1 715	343 766 µs	22 680 µs	**< 1 µs**	2 µs	42 µs
0.000223	7 139	**3 860 µs**	7 409 µs	-	28 643 µs	38 846 µs
0.005602	7 139	**4 991 µs**	7 959 µs	-	40 066 µs	51 500 µs
0.140716	7 139	**7 053 µs**	9 227 µs	-	64 975 µs	79 308 µs
3.534625	7 139	14 744 µs	**14 144 µs**	-	157 201 µs	191 833 µs
88.785777	7 139	628 527 µs	96845 µs	46 µs	**2 µs**	213 µs
0.000223	29 717	**63 ms**	108 ms	-	654 ms	1 140 ms
0.005602	29 717	**79 ms**	119 ms	-	881 ms	1 539 ms
0.140716	29 717	**108 ms**	137 ms	-	1 375 ms	2 421 ms
3.534625	29 717	202 ms	**198 ms**	-	2 968 ms	5 320 ms
88.785777	29 717	2 005 ms	**898 ms**	-	39 760 ms	243 690 ms

Figure 5: Timings of several implementations of erf

nothing to compute and 1 can be returned immediately. In our implementation $t' + 3 - E - x^2 \log_2(e)$ is computed using `MPFR` in small precision. This takes 46 µs. `MPFR` performs the same kind of test but using hardware arithmetic. This explains that it can give an answer in 2 µs.

6 Conclusion and perspectives

We proposed three algorithms for efficiently evaluating the functions erf and erfc in arbitrary precision. These algorithms are based on three different summation formulas whose coefficients have the same general recursive structure. For evaluating the sum, we take advantage of this structure by using an algorithm due to Smith that makes it possible to evaluate the sum with the binary complexity $\mathcal{O}(Nt + M(t)\sqrt{N})$ (this should be compared to the complexity $\mathcal{O}(N\,M(t))$ of the straightforward evaluation algorithm).

We gave closed formulas for upper-bounding the truncation rank $N-1$ and we completely studied the effects of roundoff errors in the summation. We derived from this study closed formulas for the required working precision.

An interesting feature of our work comes from the fact that the errors are rigorously bounded (and not only roughly estimated): to our best knowledge, the `MPFR` library is the only other library providing implementations with such guaranteed bounds. However, there was no complete documentation explaining how to write such an implementation. The general scheme exposed in this article can be used for the implementation of other functions: we completely detailed the proofs and the algorithms in order to allow one to reproduce the same steps when implementing other functions. Examples of functions that could be implemented using this scheme include Fresnel integrals, Exponential integrals or Airy functions when $x \geq 0$. The scheme would require slight modifications for other functions such as Airy functions when $x < 0$ or Bessel functions because these functions have several zeros. Due to these zeros, it is not possible to find an underestimation g (as required in our general scheme in page 4). Nevertheless, even for such vanishing functions, the techniques used for the error analysis and the evaluation algorithm remain valid and can be reused.

We implemented the three algorithms in C and compared the efficiency of the three methods in practice. This shows that the asymptotic expansion is the best method to use, as soon as it can achieve the target accuracy. Whenever the asymptotic expansion cannot be used, one must choose between the two others equations. The domain where it is interesting to use one rather than the other depends on the underlying arithmetic. In practice, well-chosen thresholds must be chosen for each architecture. We also compared our implementation with the implementation of erf provided in `MPFR` and Maple. Our implementation is almost always the fastest one. It represents a considerable improvement for intermediate and large precisions.

However, a few remarks could lead to further improvements of our implementation. Firstly, we must remark that our analysis is based on the hypothesis that no underflow or overflow occurs during the evaluation. This assumption is reasonable since the range of exponents available in `MPFR` is really wide and can often be considered as almost unbounded in practice. However, in order to be completely rigorous, we should take this possibility into account for really guaranteeing the quality of the final result.

As shown (see Figure 4 on page 24) the formula given by Equation (3) is much more efficient than the others whenever it can be used. Thus it is interesting to extend the domain

where this formula can be used. Such an extension can be obtained the following way: let x be a point such that Equation (3) does not provide enough accuracy at this point. We suppose further that x can be written as $x = x_0 - h$ where $h > 0$ is fairly small and where x_0 is in the domain where Equation (3) is useful. Hence, an approximate value of $\operatorname{erf}(x)$ can be computed with a Taylor development of erf with center x_0:

$$\operatorname{erf}(x) = \operatorname{erf}(x_0) + \sum_{i=1}^{+\infty} a_i(-h)^i \quad \text{where} \quad a_i = \frac{\operatorname{erf}^{(i)}(x_0)}{i!}.$$

Since h is fairly small, only a few terms of this series are necessary for obtaining a good approximation. The coefficients a_i have all the same form $p_i(x_0)e^{-x_0^2}$ where p_i is a polynomial of degree $2i-2$ satisfying a simple recurrence formula. The coefficients a_i are henceforth easily computable. The value $\operatorname{erf}(x_0)$ is efficiently computed using Equation (3) (we remark that the value $e^{-x_0^2}$ is already computed during this computation, and does need to be recomputed when evaluating the coefficients a_i).

In other words, though the asymptotic expansion can not be used to directly evaluate $\operatorname{erf}(x)$ with the required accuracy, it can be used to evaluate $\operatorname{erf}(x_0)$. From the latter, the value $\operatorname{erf}(x)$ is easily recovered by analytic continuation. This continuation is easy to compute because the coefficients a_i satisfy a simple recurrence. In fact, this is true for a large class of functions, called *holonomic functions* (or *D-finite* functions). A function is called holonomic when it is solution of a linear differential equation with polynomial coefficients. For instance, erf is a solution of the equation

$$\frac{\mathrm{d}^2y}{\mathrm{d}x^2} + 2x\,\frac{\mathrm{d}y}{\mathrm{d}x} = 0.$$

When a holonomic function is analytical at point x_0, the coefficients of its series at x_0 satisfy a recurrence.

D. V. Chudnovsky and G. V. Chudnovsky proposed [6] a quasi-linear algorithm (with binary complexity $\mathcal{O}(M(t)\,\log(t)^3)$) for evaluating holonomic functions with a precision of t bits. This algorithm is called the *bit-burst* algorithm and is based on two ideas:

1. When x_0 is a rational number p/q where p and q are small integers, it is possible to efficiently sum N terms of the Taylor series of a holonomic function using a technique called *binary splitting*.

2. For a generic value x, the idea of analytic continuation is recursively used: x is written $x = x_0 + h$ where $x_0 = p/q$. This leads to the evaluation of $f(x_0)$ (by binary splitting) and the evaluation of a series in h. The coefficients of this series only depend on $f(x_0)$ (and possibly the first derivatives of f at x_0). The evaluation of this series uses the same technique recursively: h is decomposed as $h = h_0 + h'$ with $h_0 = p'/q'$, etc.

Brent already proposed in 1976 an algorithm based on the binary splitting for evaluating the exponential function [3]. In fact, the bit-burst algorithm is a generalization of this algorithm to any holonomic function. However, in practice, the constant hidden behind the "$\mathcal{O}$" is not negligible and the algorithm is only interesting for fairly high precisions. Nevertheless, it is an improvement that we plan to implement.

Another remark concerns the practical efficiency of the methods exposed in this article when the precision becomes very high. Both Smith's algorithm and the bit-burst algorithm require extra space for storing intermediate results. It can be a problem for very high precisions, if it implies that the memory be swapped on the hard disk. In this case, it would be

more interesting to use a straightforward algorithm: e.g. compute iteratively the coefficients of the sum and accumulate the result on the fly. This is slower in theory but it does not require extra memory. Hence, in practice, it could be worth using it.

Finally, the functions erf and erfc could be evaluated by other means than series. For instance, these functions have nice continued fractions developments. Such developments could be used for the evaluation in arbitrary precision. This is a promising technique that we did not study yet.

Acknowledgment

The author is very grateful to Nicolas Brisebarre, Olivier Robert and Bruno Salvy for their very fruitful comments and advice. Their help considerably increased the quality of the present article.

References

[1] M. Abramowitz and I. A. Stegun. *Handbook of Mathematical Functions*. Dover, 1965.

[2] D. H. Bailey, Y. Hida, X. S. Li, and B. Thompson. Arprec: An arbitrary precision computation package. Software and documentation available at `http://crd.lbl.gov/~dhbailey/mpdist/`.

[3] R. P. Brent. The Complexity of Multiple-Precision Arithmetic. *The Complexity of Computational Problem Solving*, pages 126–165, 1976.

[4] R. P. Brent. A fortran multiple-precision arithmetic package. *ACM Transactions on Mathematical Software (TOMS)*, 4(1):57–70, March 1978.

[5] R. P. Brent. Unrestricted algorithms for elementary and special functions. In S. Lavington, editor, *Information Processing 80: Proceedings of IFIP Congress 80*, pages 613–619. North-Holland, October 1980.

[6] D. V. Chudnovsky and G. V. Chudnovsky. Computer Algebra in the Service of Mathematical Physics and Number Theory. In D. V. Chudnovsky and R. D. Jenks, editors, *Computers in Mathematics*, volume 125 of *Lecture notes in pure and applied mathematics*, pages 109–232. Dekker, 1990.

[7] R. M. Corless, G. H. Gonnet, D. E. G. Hare, D. J. Jeffrey, and D. E. Knuth. On the Lambert W function. *Advances in Computational Mathematics*, 5(4):329–359, 1996.

[8] G. E. Forsythe. Pitfalls in computation, or why a math book isn't enough. *American Mathematical Monthly*, 77(9):931–956, 1970.

[9] 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 (TOMS)*, 33(2), 2007.

[10] N. J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, second edition, 2002.

[11] A. Hoorfar and M. Hassani. Inequalities on the Lambert W function and hyperpower function. *Journal of Inequalities in Pure and Applied Mathematics*, 9(2):1–5, 2008.

[12] N. N. Lebedev. *Special Functions & Their Applications*. Prentice-Hall, 1965.

[13] M. S. Paterson and L. J. Stockmeyer. On the number of nonscalar multiplications necessary to evaluate polynomials. *SIAM Journal on Computing*, 2(1):60–66, 1973.

[14] D. M. Smith. Efficient multiple-precision evaluation of elementary functions. *Mathematics of Computation*, 52(185):131–134, 1989.

[15] A. Ziv. Fast evaluation of elementary mathematical functions with correctly rounded last bit. *ACM Transactions on Mathematical Software (TOMS)*, 17(3):410–423, 1991.

Appendix: technical results

General purpose lemmas

Lemma 4 (Propagation of errors through a square root)**.** *Let $k \in \mathbb{N}^\star$, z and z' two numbers such that we can write $z' = z\,\langle k\rangle$. Then we can write $\sqrt{z'} = \sqrt{z}\,\langle k\rangle$.*

Proof. By hypothesis, $z' = z \prod_{i=1}^{k} (1+\delta_i)^{s_i}$ with $|\delta_i| \leq u$ and $s_i \in \{-1, 1\}$. Hence,

$$\sqrt{z'} = \sqrt{z} \prod_{i=1}^{k} \left(\sqrt{1+\delta_i}\right)^{s_i}.$$

Now, by the mean value theorem,

$$\sqrt{1+\delta_i} = 1 + \delta_i \frac{1}{2\sqrt{1+\xi_i}},$$

where ξ_i lies between 0 and δ_i. In particular, $|\xi_i| \leq u \leq 1/2$. Hence, $\sqrt{1+\xi_i} \geq \sqrt{1/2}$. It follows that

$$\left|\delta_i \frac{1}{2\sqrt{1+\xi_i}}\right| \leq u\,\frac{\sqrt{2}}{2} \leq u.$$

It proves the result. □

Lemma 5 (Propagation of errors through exp)**.** *Let $z \in \mathbb{R}$. We denote by E its exponent: $2^{E-1} \leq z < 2^E$. Let t be a precision. We note $y = z^2$ and we suppose that $\widehat{y} = \diamond(z^2)$, the operation being performed with a precision larger than $t + 2E$. Then*

$$e^{-\widehat{y}} = e^{-z^2}(1+\delta)^s \quad \textit{where } |\delta| \leq 2^{1-t} \textit{ and } s \in \{-1, 1\}.$$

In other words, we can write $e^{-\widehat{y}} = e^{-z^2}\,\langle 1\rangle$ in precision t.

Proof. Since $\widehat{y}$ is a rounding of z^2 in precision $t+2E$, we have $\widehat{y} = z^2\,(1+\theta)$ with $|\theta| \leq 2^{1-t-2E}$. In other words, $\widehat{y} = z^2 \ + h$ where $|h| \leq z^2\,2^{1-t-2E} \leq 2^{1-t}$. Hence, we have

$$e^{-\widehat{y}} = e^{-z^2}\,e^{-h}.$$

We distinguish two cases:

- If $h \geq 0$, by the mean value theorem, $\exp(-h) = 1 + (-h)\exp(\xi)$ where $\xi \in [-h, 0]$. In particular $|-h\exp(\xi)| \leq |h| \leq 2^{1-t}$.
- If $h \leq 0$, by the mean value theorem, $1/\exp(-h) = \exp(h) = 1 + h\exp(\xi)$ where $\xi \in [h, 0]$. In particular $|h\exp(\xi)| \leq |h| \leq 2^{1-t}$.

In both cases, we can write e^{-h} as $(1+\delta)^s$ with $|\delta| \leq 2^{1-t}$ and $s \in \{-1, 1\}$. □

To bound the remainder of the series, we need to approximate $n!$. We use the following estimations:

Lemma 6 (Rough estimation of $n!$)**.** *The following inequalities hold for all $n \geq 1$:*

$$\sqrt{2\pi n}\left(\frac{n}{e}\right)^n \leq n! \leq e\sqrt{n}\left(\frac{n}{e}\right)^n.$$

Proof. Consider the sequence defined for $n \geq 1$ by

$$u_n = \frac{\sqrt{2\pi n}(n/e)^n}{n!}.$$

One sees that this sequence is increasing by considering u_{n+1}/u_n. It is well known (Stirling formula) that $u_n \to 1$ as $n \to +\infty$. Therefore, for all $n \geq 1$, $u_1 \leq u_n \leq 1$. This gives the result. □

The rough estimation of N is obtained by inverting a certain relation. This relation involves the function $v \mapsto v\log_2(v)$. Hence we need to estimate its reciprocal φ. The functions φ is closely related to the Lambert W function (defined as the reciprocal of $x \mapsto x\,e^x$). The function W has been well studied [7, 11]; however, the known bounds for W do not allow us to determine accurate lower and upper bounds for φ on its whole domain. The following lemmas give such estimates.

Lemma 7 (Inverse of $v\log_2(v)$)**.** *The function $v \mapsto v\log_2(v)$ is increasing for $v \geq 1/e$. We denote by φ its inverse: $\varphi : w \mapsto \varphi(w)$ defined for $w \geq -\log_2(e)/e$ and such that for all w, $\varphi(w)\log_2(\varphi(w)) = w$. The function φ is increasing.*

The following inequalities hold:

$$\begin{array}{lrcl}
\text{if } w \in [-\log_2(e)/e,\, 0], & 2^{ew} & \leq \varphi(w) \leq & 2^w\,; \\
\text{if } w \in [0,\, 2], & 2^{w/2} & \leq \varphi(w) \leq & 2^{1/4}\,2^{w/2}\,; \\
\text{if } w \geq 2, & w/\log_2(w) & \leq \varphi(w) \leq & 2w/\log_2(w).
\end{array}$$

Proof. Showing that $v \mapsto v\log_2(v)$ is increasing for $v \geq 1/e$ does not present any difficulty. We only prove the second inequality. The others are proved using the same technique.

We denote $\varphi(w)$ by v for convenience. If $w \in [0, 2]$, it is easy to see that $v \in [1, 2]$. Now, since $\log_2(v) = w/v$, we get $w/2 \leq \log_2(v) \leq w$. Therefore

$$2^{w/2} \leq v \leq 2^w, \tag{7}$$

which gives the lower bound.

We can refine this identity: using again that $\log_2(v) = w/v$, we get

$$\frac{w}{2^w} \leq \log_2(v) \leq \frac{w}{2^{w/2}} \quad \text{and thus} \quad v \leq 2^{(w\,2^{-w/2})}.$$

For finishing the proof, we only need to show that for any $w \in [0, 2]$,

$$w\,2^{-w/2} \leq \frac{1}{4} + \frac{w}{2}.$$

The Taylor expansion of $2^{-w/2}$ is

$$2^{-w/2} = \sum_{n=0}^{+\infty} (-1)^n \frac{(w\ln(2))^n}{2^n\,n!}.$$

The series is alternating with a decreasing general term when $w \in [0,\, 2]$. Hence

$$2^{-w/2} \leq 1 - \frac{w\ln(2)}{2} + \frac{w^2\ln(2)^2}{8}.$$

From this inequality, we deduce

$$w\,2^{-w/2} - \frac{w}{2} \leq w\left(\frac{1}{2} - \frac{w\,\ln(2)}{2} + \frac{w^2\,\ln(2)^2}{8}\right).$$

Studying the variations of the right-hand term of this inequality, we get

$$w\,2^{-w/2} - \frac{w}{2} \leq \frac{4}{27\,\ln(2)} \simeq 0.2137\cdots \leq 1/4.$$

It finishes the proof.
Remark: the last inequality is proved using $v \leq v\log_2(v) \leq v^2$ for $v \geq 2$. We rewrite it $\sqrt{w} \leq v \leq w$ and then we apply the same technique. □

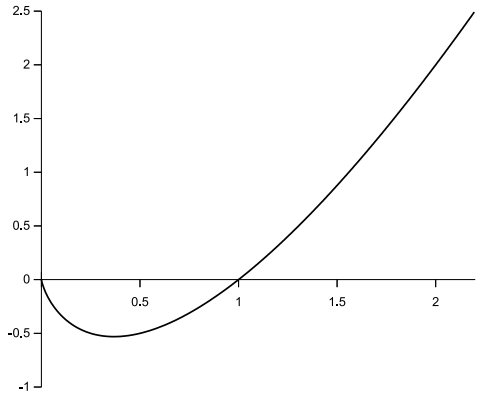


Figure 6: Graph of the function $v \mapsto v\log_2(v)$.

Lemma 8 (Inverse of $v\log_2(v)$, the other branch)**.** *The function $v \mapsto v\log_2(v)$ is decreasing for $0 \leq v \leq 1/e$. We denote by φ_2 its inverse: $\varphi_2 : w \mapsto \varphi_2(w)$ such that $\varphi_2(w)\log_2(\varphi_2(w)) = w$. The value $\varphi_2(w)$ is defined for $-\log_2(e)/e \leq w \leq 0$ and is decreasing.*

The following inequalities give an estimate of $\varphi_2(w)$:

$$\forall w \in \left[-\frac{\log_2(e)}{e},\, 0\right), \qquad \frac{1}{3}\cdot\frac{w}{\log_2(-w)} \;\leq\; \varphi_2(w) \;\leq\; \frac{w}{\log_2(-w)}.$$

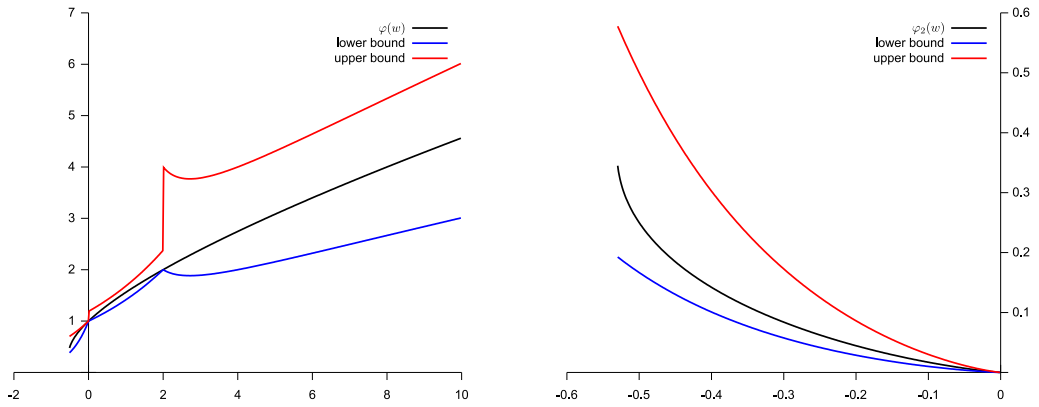
Proof. Showing that $v \mapsto v\log_2(v)$ is decreasing for $v \in [0,\, 1/e]$ does not present any difficulty. We will use the following notations that are more convenient:

$$\begin{aligned} \omega &= 1/(-w), \\ \nu &= 1/\varphi_2(w). \end{aligned}$$

Hence $\omega \geq e/\log_2(e)$ and $\nu \geq e$. Moreover, by hypothesis, $\nu/\log_2(\nu) = \omega$. It is easy to show that for any $\nu \geq e$,

$$\nu^{1/3} \leq \frac{\nu}{\log_2(\nu)} \leq \nu.$$

Therefore, $\omega \leq \nu \leq \omega^3$ and thus $\log_2(\omega) \leq \log_2(\nu) \leq 3\log_2(\omega)$. We conclude by using $\nu = \omega \log_2(\nu)$. □



(a) Graph of φ and its bounds

(b) Graph of φ_2 and its bounds

Figure 7: Illustration of Lemmas 7 and 8

When bounding relative errors, we need to estimate the values of $\mathrm{erf}(x)$ and $\mathrm{erfc}(x)$. The next lemma gives such estimates.

Lemma 9. *The following inequalities hold:*

$$\begin{array}{lrcl} \text{if } 0 < x < 1, & x/2 \leq \mathrm{erf}(x) & \leq & 2x, \\ & 1/8 \leq \mathrm{erfc}(x) & \leq & 1; \\ \\ \text{if } x \geq 1, & 1/2 \leq \mathrm{erf}(x) & \leq & 1, \\ & e^{-x^2}/(4x) \leq \mathrm{erfc}(x) & \leq & e^{-x^2}/(x\sqrt{\pi}). \end{array}$$

Proof. When $0 < x < 1$, the series given in Equation (1) is alternating with a decreasing general term. Hence

$$\frac{2x}{\sqrt{\pi}}\left(1 - \frac{x^2}{3}\right) \leq \mathrm{erf}(x) \leq \frac{2x}{\sqrt{\pi}}.$$

The inequalities for $\mathrm{erf}(x)$ are easily obtained from it.

Since erfc is decreasing, $\mathrm{erfc}(1) \leq \mathrm{erfc}(x) \leq \mathrm{erfc}(0) = 1$. Taking one more term in the series of $\mathrm{erf}(x)$, we get

$$\mathrm{erf}(x) \leq \frac{2x}{\sqrt{\pi}}\left(1 - \frac{x^2}{3} + \frac{x^4}{10}\right).$$

Applying it at $x = 1$, we get $\mathrm{erfc}(1) = 1 - \mathrm{erf}(1) \geq 1/8$.

When $x > 1$, since erf is increasing and goes to 1 as $x \to +\infty$, we have $\mathrm{erf}(1) \leq \mathrm{erf}(x) \leq 1$. This gives the inequalities for $\mathrm{erf}(x)$.

We now prove the last two inequalities of the lemma. By definition,

$$\begin{aligned}\operatorname{erfc}(x) &= \frac{2}{\sqrt{\pi}}\int_x^{+\infty} e^{-v^2}\,\mathrm{d}v \\ &= \frac{2}{\sqrt{\pi}}\int_x^{+\infty} \frac{-2v\,e^{-v^2}}{-2v}\,\mathrm{d}v \\ &= \frac{2}{\sqrt{\pi}}\left(\frac{e^{-x^2}}{2x} - \int_x^{+\infty}\frac{e^{-v^2}}{2v^2}\,\mathrm{d}v\right).\end{aligned}$$

This gives the upper bound. For the lower bound, we use the change of variable $v \leftarrow v + x$:

$$\int_x^{+\infty}\frac{e^{-v^2}}{2v^2}\,\mathrm{d}v = \int_0^{+\infty}\frac{e^{-x^2-2vx-v^2}}{2(v+x)^2}\,\mathrm{d}v.$$

Since $(v+x)^2 \geq x^2$ and $-v^2 \leq 0$,

$$\int_0^{+\infty}\frac{e^{-x^2-2vx-v^2}}{2(v+x)^2}\,\mathrm{d}v \leq \frac{e^{-x^2}}{2x^2}\int_0^{+\infty} e^{-2vx}\,\mathrm{d}v = \frac{e^{-x^2}}{4x^3}.$$

Therefore,

$$\begin{aligned}\operatorname{erfc}(x) &\geq \frac{e^{-x^2}}{x\sqrt{\pi}}\left(1-\frac{1}{2x^2}\right) \\ &\geq \frac{e^{-x^2}}{x\,2\sqrt{\pi}} \quad \text{since } x \geq 1.\end{aligned}$$

We conclude by remarking that $2\sqrt{\pi} \simeq 3.544 \leq 4$. □

Results related to the implementation of Equation (1)

Proposition 3. *Let E be the exponent of x. If N satisfies*

$$\frac{N}{ex^2}\log_2\left(\frac{N}{ex^2}\right) \geq \frac{t' + \max(0,\,E)}{ex^2},$$

the remainder $\varepsilon_N^{(1)}$ is bounded by $2^{-t'-1}\operatorname{erf}(x)$.

Proof. Remark first that for $N \geq 1$,

$$\frac{2}{\sqrt{\pi}}\cdot\frac{1}{(2N+1)\sqrt{2\pi N}} \leq \frac{1}{4}. \tag{8}$$

We distinguish the cases when $x < 1$ and when $x \geq 1$:

- if $x < 1$, $E \leq 0$ and $\operatorname{erf}(x)$ is greater than $x/2$. The hypothesis becomes $(ex^2/N)^N \leq 2^{-t'}$ and thus
$$\frac{x^{2N+1}}{2}\left(\frac{e}{N}\right)^N \leq 2^{-t'}\operatorname{erf}(x). \tag{9}$$

- if $x \geq 1$, $E > 0$ and $\mathrm{erf}(x)$ is greater than $1/2$. The hypothesis becomes $(ex^2/N)^N \leq 2^{-t'-E}$ and thus
$$\frac{x^{2N}}{2}\left(\frac{e}{N}\right)^N \leq 2^{-t'-E}\ \mathrm{erf}(x).$$
Since $x < 2^E$ we obtain Inequality (9) again.

From Inequalities (8) and (9) we deduce that

$$\frac{2}{\sqrt{\pi}}\cdot\frac{x^{2N+1}}{2N+1}\left(\frac{e}{N}\right)^N\frac{1}{\sqrt{2\pi N}} \leq 2^{-t'-1}\ \mathrm{erf}(x).$$

Using Lemma 6, we get

$$\frac{2}{\sqrt{\pi}}\cdot\frac{x^{2N+1}}{(2N+1)N!} \leq 2^{-t'-1}\ \mathrm{erf}(x).$$

Remark that the left term of the inequality is the absolute value of the general term of the series. Since it is smaller than $2^{-t'-1}\ \mathrm{erf}(x)$, it is in particular smaller than $1/2$. Since the series is alternating, we can bound the remainder by the absolute value of the first neglected term as soon as the term decreases in absolute value.

In our case, the absolute value of the general term may begin by increasing before decreasing. But when it increases, it is always greater than 1. Therefore, since here it is smaller than $1/2$, we are in the decreasing phase and we can write $\varepsilon_N^{(1)}(x) \leq 2^{-t'-1}\ \mathrm{erf}(x)$. □

Recipe 1. *We first evaluate $a = (t' + \max(0,\, E))/(ex^2)$. In fact, when performing the evaluation, the rounding modes are carefully chosen in such a way that one gets an overestimation a_u of a: $a_u \geq a$. Then, N is chosen according to the following recipe:*

– *if* $a_u \geq 2$,	N	$\geq$	$(ex^2)\, 2a_u/\log_2(a_u)$;
– *if* $a_u \in [0,\, 2]$,	N	$\geq$	$(ex^2)\, 2^{1/4}\, 2^{a_u/2}$.

These formulas are evaluated with appropriate rounding modes, in order to ensure that N is really greater than the actual value.

Proof. Proposition 3 gives a sufficient condition on N that ensures that $\varepsilon_N^{(1)}$ is bounded by $2^{-t'-1}\ \mathrm{erf}(x)$. We reason by sufficient conditions:

- from Proposition 3, it suffices that
$$\frac{N}{ex^2}\log_2\left(\frac{N}{ex^2}\right) \geq a\ ;$$
- hence, it suffices that
$$\frac{N}{ex^2}\log_2\left(\frac{N}{ex^2}\right) \geq a_u\ ;$$
- this inequality can be inverted with help of function φ (defined in Lemma 7). Namely, we have $N/(ex^2) \geq \varphi(a_u)$;
- for obtaining this inequality, it suffices that $N/(ex^2)$ be greater than an upper bound of $\varphi(a_u)$. Such an upper bound is given by Lemma 7. The recipe follows.

□

Proposition 4. *If Algorithm 3 is used to compute an approximation* $\widehat{S(x)}$ *of the sum* $S(x)$*, the following holds:*

$$\widehat{S(x)} = \sum_{n=0}^{N-1} (-1)^n \frac{2x}{\sqrt{\pi}} \cdot \frac{x^{2n}}{(2n+1)\,n!} \langle 8N \rangle .$$

Thus

$$\left|\widehat{S(x)} - S(x)\right| \leq \gamma_{8N} \left(\frac{2x}{\sqrt{\pi}} \sum_{n=0}^{N-1} \frac{x^{2n}}{(2n+1)\,n!} \right) \leq \gamma_{8N} \frac{2}{\sqrt{\pi}} \int_0^x e^{v^2}\,\mathrm{d}v.$$

The bound γ_{8N} *could be made tighter. However, we cannot hope a better value than* γ_N *since we do* $\mathcal{O}(N)$ *operations. Only the logarithm of this value will be of interest for choosing the working precision* t*. By working more carefully, we would not get more than replacing* $\log(8N)$ *by* $\log(N)$ *and it would not be of any practical benefit.*

Proof. To prove this result, we use the same techniques as those presented in the example of Section 3. The main arguments are the following.

- Line 1 of the algorithm leads to one error;
- Line 4 is obtained by binary exponentiation: by counting the number of multiplications, it is easy to show that $\widehat{z} = z\,\langle 2L-1 \rangle$;
- Line 6 involves an approximation of π ($\widehat{\pi} = \pi\,\langle 1 \rangle$) and a square root. Using Lemma 4 we get $\mathsf{acc} = \diamond(\sqrt{\widehat{\pi}}) = \diamond(\sqrt{\pi}\,\langle 1 \rangle) = \sqrt{\pi}\,\langle 2 \rangle$;
- Line 7 involves a division (the multiplication by 2 is exact);
- acc is updated at lines 16 and 20 of the algorithm. At line 16, it accumulates $2L$ errors ($2L-1$ due to z and one due to the multiplication itself). At line 20, it accumulates one error. It is hence easy to show that we can always write

  $$\widehat{\mathsf{acc}} = \mathsf{acc}\,\langle \underbrace{3}_{\text{initialization}} + \underbrace{N}_{\text{line 20 occurs at most } N \text{ times}} + \underbrace{2L\lfloor N/L \rfloor}_{\text{line 16 occurs at most } \lfloor N/L \rfloor \text{ times}} \rangle.$$

  We simplify it in $\widehat{\mathsf{acc}} = \mathsf{acc}\,\langle 3+3N \rangle$;
- We can always write $\widehat{\mathsf{tmp}} = \mathsf{tmp}\,\langle 4+3N \rangle$ since tmp is obtained from acc by one division ($2k+1$ is computed exactly in integer arithmetic);
- Eventually, $S[i]$ is obtained by less than N additions using variable tmp which allows to write

  $$S_i = \mathsf{tmp}_i\,\langle 4N+4 \rangle + \mathsf{tmp}_{i+L}\,\langle 4N+4 \rangle + \cdots + \mathsf{tmp}_{i+N-L}\,\langle 4N+4 \rangle$$

  where tmp_k denotes the value of variable tmp at step k of the algorithm;
- The evaluation by Horner's rule at line 25 accumulates 3 more errors per step (one comes from the fact that $y = x^2\,\langle 1 \rangle$, one comes from the multiplication and one comes from the addition). Therefore, during the complete loop, $3(L-1)$ errors are accumulated per term of the sum. We bound it by $3N-3$ and finally get the result with $\langle 7N+1 \rangle$. We bound it by $\langle 8N \rangle$.

□

The value $\left(\int_0^x e^{v^2}\,\mathrm{d}v\right)$ quantifies the *catastrophic cancellations* that appear when Equation (1) is used for evaluating $\mathrm{erf}(x)$. The following lemma allows us to bound it.

Lemma 10. *The following inequalities hold:*

$$\begin{array}{lccccc} \text{if } 0 < x < 1, & x & \leq & \displaystyle\int_0^x e^{v^2}\,\mathrm{d}v & \leq & 2x; \\ \text{if } x \geq 1, & \dfrac{1}{e^2}\cdot\dfrac{e^{x^2}}{x} & \leq & \displaystyle\int_0^x e^{v^2}\,\mathrm{d}v & \leq & \dfrac{e^{x^2}}{x}. \end{array}$$

Proof. For the first inequality, the lower bound is obtained by replacing e^{v^2} by 1 in the integral. The upper bound is obtained by replacing e^{v^2} by $e^{0.55^2}$ on $[0,\, 0.55]$ and by e on $[0.55,\, 1]$.

For the second inequality, the upper bound is obtained by replacing e^{v^2} by e^{vx} in the integral. The lower bound is obtained by considering the integral restricted to the interval $[x - 1/x,\, x]$. □

Recipe 2. *Let E be the exponent of x. For the evaluation of* $\mathrm{erf}(x)$ *by Equation* (1)*, an appropriate working precision t is*

– *when* $x < 1$,	$t \geq t' + 9 + \lceil \log_2 N \rceil$;
– *when* $x \geq 1$,	$t \geq t' + 9 + \lceil \log_2 N \rceil - E + x^2 \log_2(e)$.

In practice $\log_2(e)$ *is replaced by a value precomputed with rounding upwards. The factor* $x^2 \log_2(e)$ *that appears when* $x > 1$ *highlights the fact that the series is ill-conditioned for large values of* x*.*

Proof. We show the result for the first inequality. The second one is obtained the same way. We suppose that $0 < x < 1$ and $t \geq t' + 9 + \lceil \log_2 N \rceil$. Thus

$$2^{-t+8}\, N \leq 2^{-t'-1}.$$

Using the fact that $\gamma_{8N} \leq 16Nu$ (Proposition 2), we get

$$2\gamma_{8N}(2x) \leq 2^{-t'-1}\,\frac{x}{2}.$$

We conclude by using $(x/2) \leq \mathrm{erf}(x)$ (Lemma 9), $(2/\sqrt{\pi}) \leq 2$ and $\displaystyle\int_0^x e^{v^2}\,\mathrm{d}v \leq 2x$ (Lemma 10). □

Results related to the implementation of Equation (2)

The first thing we need is a way of bounding the remainder. This is achieved by the following lemma:

Lemma 11. *If $N \geq 2x^2$, the following inequality holds:*

$$\varepsilon_N^{(2)}(x) \leq 2\,\frac{2xe^{-x^2}}{\sqrt{\pi}}\cdot\frac{(2x^2)^n}{1\cdot 3\cdots(2N+1)}.$$

Proof. By definition,

$$\begin{aligned}\varepsilon_N^{(2)}(x) &= \frac{2xe^{-x^2}}{\sqrt{\pi}} \sum_{n=N}^{+\infty} \frac{(2x^2)^n}{1\cdot 3\cdots(2n+1)} \\ &= \frac{2xe^{-x^2}}{\sqrt{\pi}} \cdot \frac{(2x^2)^N}{1\cdot 3\cdots(2N+1)} \left(1 + \frac{2x^2}{2N+3} + \frac{(2x^2)^2}{(2N+3)(2N+5)} + \cdots\right).\end{aligned}$$

We bound it by the geometric series with a common ratio of $(2x^2)/(2N+3)$ and a first term equal to 1:

$$\varepsilon_N^{(2)}(x) \leq \frac{2xe^{-x^2}}{\sqrt{\pi}} \cdot \frac{(2x^2)^N}{1\cdot 3\cdots(2N+1)} \left(1 + \frac{2x^2}{2N+3} + \frac{(2x^2)^2}{(2N+3)(2N+3)} + \cdots\right)$$

Since $N \geq 2x^2$, $(2x^2)/(2N+3) \leq 1/2$ and the sum of the series is bounded by 2. □

Proposition 5. *Let E be the exponent of x. If N satisfies $N \geq 2x^2$ and*

$$\frac{N}{ex^2} \log_2\left(\frac{N}{ex^2}\right) \geq \frac{t' + 3 + \max(0,\, E) - x^2 \log_2(e)}{ex^2}$$

the remainder is bounded by $\operatorname{erf}(x)\, 2^{-t'-1}$.

Proof. We use the same kind of arguments as for Proposition 3. The product $1\cdot 3\cdots(2N+1)$ is equal to $(2N+1)!/(2^N\, N!)$. This lets us write

$$\left|\varepsilon_N^{(2)}(x)\right| \leq 4\, \frac{xe^{-x^2}}{\sqrt{\pi}} \cdot \frac{(2x^2)^N\, N!\, 2^N}{(2N+1)!} \leq 4\, \frac{xe^{-x^2}}{\sqrt{\pi}} \cdot \frac{(2x^2)^N\, N!\, 2^N}{(2N)!}.$$

We conclude using Lemmas 6 and 9. □

Recipe 3. *We deduce the formulas for computing an overestimation of N:*

- *if $a_u \geq 2$, take $N \geq (ex^2)\, 2a_u / \log_2(a_u)$;*
- *if $a_u \in [0,\, 2]$, take $N \geq (ex^2)\, 2^{1/4}\, 2^{a_u/2}$;*
- *if $a_u < 0$, let $N_0 \geq (ex^2)\, 2^{a_u}$ and perform the following test:*
  - *if $N_0 \geq 2x^2$, take $N = N_0$,*
  - *else take $N = \lceil 2x^2 \rceil$.*

Only the case $a_u < 0$ could lead to a value N smaller than $2x^2$. If it is the case, we take $N = \lceil 2x^2 \rceil$, in order to ensure the hypothesis of Proposition 5.

Proof. The proof is similar to the proof of Recipe 1. □

Proposition 6. *If Algorithm 4 is used to compute an approximation $\widehat{S(x)}$ of the sum $S(x)$, the following holds:*

$$\widehat{S(x)} = \sum_{n=0}^{N-1} \frac{2xe^{-x^2}}{\sqrt{\pi}} \cdot \frac{(2x^2)^n}{1\cdot 3\cdots(2n+1)} \langle 16N \rangle.$$

Thus

$$\left|\widehat{S(x)} - S(x)\right| \leq \gamma_{16N}\, S(x) \leq \gamma_{16N}\, \operatorname{erf}(x).$$

Proof. In fact, using the same techniques as in Proposition 4, one proves that

$$\widehat{S(x)} = \sum_{n=0}^{N-1} \frac{2xe^{-x^2}}{\sqrt{\pi}} \cdot \frac{(2x^2)^n}{1 \cdot 3 \cdots (2n+1)} \langle 7N+3 \rangle .$$

We bound it by $\langle 16N \rangle$ because it is more convenient to use powers of 2.

Note that at line 9 of the algorithm, x^2 is computed in precision $t + \max(2E, 0)$. Using Lemma 5, it allows for writing $\widehat{\alpha_0} = \alpha_0 \langle 6 \rangle$. □

Results related to the evaluation of $\operatorname{erfc}(x)$ by Equations (1) and (2)

Lemma 1. *If s is chosen according to the following recipe, $|R - \operatorname{erf}(x)| \le 2^{-t'-1} \operatorname{erfc}(x)$.*

– *If* $x \le -1$,	$s \ge t' + 1$;
– *if* $-1 < x < 0$,	$s \ge t' + 2 + E$;
– *if* $0 \le x < 1$,	$s \ge t' + 5 + E$;
– *if* $x \ge 1$,	$s \ge t' + 3 + E + x^2 \log_2(e)$.

Proof.
First case.
We suppose $x \le -1$ and $s \ge t' + 1$. From the hypotheses we get

$$|R - \operatorname{erf}(x)| \le 2^{-s} \, |\operatorname{erf}(x)| \le 2^{-t'-1} \, |\operatorname{erf}(x)|.$$

Since $x < 0$, $\operatorname{erfc}(x) \ge 1$. Moreover, $|\operatorname{erf}(x)| \le 1$. This gives the result.

Second case.
We suppose $-1 < x < 0$ and $s \ge t' + 2 + E$. From the hypotheses we get

$$|R - \operatorname{erf}(x)| \le 2^{-s} \, |\operatorname{erf}(x)| \le 2^{-t'-1} \, 2^{-E-1} \, |\operatorname{erf}(x)|.$$

Since $x < 0$, $\operatorname{erfc}(x) \ge 1$. Moreover, $|\operatorname{erf}(x)| = \operatorname{erf}(|x|) \le 2|x| \le 2^{E+1}$. This gives the result.

Third case.
It is the same as the second case but using $\operatorname{erfc}(x) \ge 1/8$.

Fourth case.
Here $x \ge 1$ and $s \ge t' + 3 + E + x^2 \log_2(e)$. Hence

$$|R - \operatorname{erf}(x)| \le 2^{-s} \operatorname{erf}(x) \le 2^{-t'-1} \, 2^{-E-2} \, e^{-x^2} \, |\operatorname{erf}(x)|.$$

Since $x \ge 1$, $\operatorname{erf}(x) \le 1$ and $\operatorname{erfc}(x) \ge e^{-x^2}/(4x) \ge e^{-x^2}/(4\,2^E)$. This gives the result. □

Results related to the implementation of Equation (3)

Lemma 12. *The following inequality holds for any $x \ge 1$:*

$$|\varepsilon_N^{(3)}(x)| \le \frac{e^{-x^2}}{x} \left(\frac{N}{ex^2} \right)^N .$$

Proof. The result is obtained from the bound given in Equation (4). The product $1\cdot 3\cdots(2N-1)$ is equal to $(2N)!/(2^N\,N!)$. We conclude using the bounds given in Lemma 6. □

Proposition 7. *If N satisfies*

$$\frac{N}{ex^2}\,\log_2\left(\frac{N}{ex^2}\right) \leq \frac{-t'-3}{ex^2}$$

the remainder is bounded by $\operatorname{erfc}(x)\,2^{-t'-1}$.

Proof. We use the previous lemma and the inequality $\operatorname{erfc}(x) \geq e^{-x^2}/(4x)$ given in Lemma 9. □

Recipe 5. *The rule for computing N is the following: first, we compute $a = (-t'-3)/ex^2$ choosing the rounding modes for obtaining a underestimation a_d of a and then we choose N according to the following recipe:*

- *if $a_d < -\log_2(e)/e$, Equation (3) cannot be used;*
- *else let $N_0 \geq (ex^2)\,a_d/\log_2(-a_d)$ and perform the following test:*
  - *if $N_0 \leq x^2$, let $N = N_0$,*
  - *else let $N_1 \simeq x^2$ and perform the following test:*
    - *if $\dfrac{N_1}{ex^2}\,\log_2\left(\dfrac{N_1}{ex^2}\right) \leq a_d$, let $N = N_1$,*
    - *else, Equation (3) cannot be used.*

Of course the value $-\log_2(e)/e$ is evaluated in such a way that we get an overestimation of the actual value. Also, in the test $N_0 \leq x^2$, the value x^2 is replaced by an underestimation.

Proof. It is the same technique as in the proof of Recipe 1. Lemma 8 is used to invert the equation of Proposition 7. □

Proposition 8. *If Algorithm 5 is used to compute an approximation $\widehat{S(x)}$ of the sum $S(x)$, the following holds:*

$$\left|\widehat{S(x)} - S(x)\right| \leq \gamma_{16N}\,\frac{e^{-x^2}}{x\sqrt{\pi}}\cdot\frac{3}{2}.$$

Proof. In this algorithm, $\widehat{y} = y\,\langle 2\rangle$, hence the binary exponentiation leads to

$$\widehat{z} = z\,\langle 3L-1\rangle\,.$$

Using the same technique as in the proof of Proposition 4, we get the bound

$$\widehat{S(x)} = \frac{e^{-x^2}}{x\sqrt{\pi}}\,\langle 8N+3\rangle + \sum_{n=1}^{N-1}(-1)^n\,\frac{e^{-x^2}}{x\sqrt{\pi}}\cdot\frac{1\cdot 3\cdots(2n-1)}{(2x^2)^n}\,\langle 8N+3\rangle\,.$$

We bound $\langle 8N+3\rangle$ by $\langle 16N\rangle$ and thus obtain

$$\left|\widehat{S(x)} - S(x)\right| \leq \gamma_{16N}\,\frac{e^{-x^2}}{x\sqrt{\pi}}\left(1 + \sum_{n=1}^{N-1}\frac{1\cdot 3\cdots(2n-1)}{(2x^2)^n}\right).$$

We now prove that

$$1+\sum_{n=1}^{N-1}\frac{1\cdot 3\cdots(2n-1)}{(2x^2)^n}\leq\frac{3}{2}\quad\text{for } N\leq\lfloor x^2+1/2\rfloor.$$

Since $N\leq\lfloor x^2+1/2\rfloor$, $N\leq x^2+1$. Moreover, the general term is decreasing. Hence, we can write

$$\sum_{n=1}^{N-1}\frac{1\cdot 3\cdots(2n-1)}{(2x^2)^n}\;\leq\;(N-1)\frac{1}{2x^2}\;\leq\;\frac{1}{2}.$$

□

Results related to the implementation of $\operatorname{erf}(x)$ by Equation (3)

Lemma 2. *If $x\geq 1$, the inequality $|R-\operatorname{erfc}(x)|\leq 2^{-t'-1}\operatorname{erfc}(-x)$ holds when s is chosen according to the following recipe:*

$$\boxed{s\geq t'+2-E-x^2\log_2(e).}$$

Proof. Since $s\geq t'+2-E-x^2\log_2(e)$, we have $2^{-s}\leq 2^{-t'-1}2^{E-1}e^{x^2}$. Moreover, we always have $2^{E-1}\leq x$ and by Lemma 9

$$\operatorname{erfc}(x)\leq\frac{e^{-x^2}}{x\sqrt{\pi}}\leq\frac{e^{-x^2}}{x}.$$

By definition, $|R-\operatorname{erfc}(x)|\leq 2^{-s}\operatorname{erfc}(x)$, thus $|R-\operatorname{erfc}(x)|\leq 2^{-t'-1}$. We conclude by remarking that $\operatorname{erfc}(-x)\geq 1$ since $x\geq 0$. □

Lemma 3. *If $x\geq 1$ the inequality $|R-\operatorname{erfc}(x)|\leq 2^{-t'-1}\operatorname{erf}(x)$ holds when s is chosen according to the following recipe:*

$$\boxed{s\geq t'+3-E-x^2\log_2(e).}$$

Proof. We proceed the same way as in the previous proof. Hence we get

$$|R-\operatorname{erfc}(x)|\leq 2^{-t'-1}\frac{1}{2}.$$

Since $x\geq 1$, Lemma 9 indicates that $\operatorname{erf}(x)\geq 1/2$, which concludes the proof. □