Worst Cases of a Periodic Function for Large Arguments

Guillaume Hanrot* Vincent Lefèvre† Damien Stehlé‡ Paul Zimmermann§

Abstract

One considers the problem of finding hard to round cases of a periodic function for large floating-point inputs, more precisely when the function cannot be efficiently approximated by a polynomial. This is one of the last few issues that prevents from guaranteeing an efficient computation of correctly rounded transcendentals for the whole IEEE-754 double precision format. The first non-naive algorithm for that problem is presented, with a heuristic complexity of $O(2^{0.676p})$ for a precision of p bits. The efficiency of the algorithm is shown on the largest IEEE-754 double precision binade for the sine function, and some corresponding bad cases are given. We can hope that all the worst cases of the trigonometric functions in their whole domain will be found within a few years, a task that was considered out of reach until now.

1. Introduction

One of the most important paradigms of the IEEE-754 standard on floating-point arithmetic [3] is *correct rounding*. Although this correct rounding can often be efficiently computed for basic arithmetical operations and algebraic functions [4], the case of general functions is more difficult.

A general strategy, due to Ziv [10], helps one to guarantee the rounding by computing an approximation y to a function value $f(x)$, together with an upper bound on the corresponding error, including both the mathematical error and the roundoff error. This amounts to finding a small interval $[y-\varepsilon, y+\varepsilon]$ in which the actual value $f(x)$ lies. The monotonicity of the various rounding modes allows one to determine the correct rounding as soon as $y-\varepsilon$ and $y+\varepsilon$ round to the same floating-point number. If this is not the case, one increases the working precision until this happens. For a target precision of p bits, let p' denote the smallest intermediate precision such that an error bounded by $\varepsilon = 2^{-p'}|f(x)|$ allows to guarantee the correct rounding of $f(x)$. Floating-point numbers x such that p' is large are called "worst cases of f in precision p" (in a loose sense, see Definition 2 for a more precise definition).

A probabilistic argument tells us that the additional precision $p'-p$ needed to guarantee the correct rounding in precision p — except particular cases that can be determined for each function — should be of the order of $\log_2 |\mathcal{M}| + O(1)$, where $\mathcal{M}$ is the set of numbers x under consideration. For IEEE-754 double precision numbers, i.e., with $p = 53$ and $|\mathcal{M}| \leq 2^{64}$, this means that, in order to be able to correctly round $f(x)$ for *any* input x, computing approximations *up to* a precision slightly larger than 117 bits should be sufficient.

A search for worst cases of usual functions in double precision was initiated by Lefèvre and Muller in the late 90's. In particular, they proposed the first non-naive algorithm, see [5] and the references therein. The key idea is to split the tested domain into intervals, and in each interval, replace the function by a linear approximation, for which worst cases can be determined by a continued fraction type method.

In 2003, Stehlé, Lefèvre and Zimmermann (SLZ for short) improved Lefèvre and Muller's method by using higher degree approximations [8]. The worst cases of the corresponding polynomial approximations are then found by a method due to Coppersmith [1]. By using the works of Lefèvre, Muller, Stehlé and Zimmermann, almost all worst cases of univariate functions in double precision are within reach. So far, there remained one exception amongst the usual functions: worst cases of $\sin$, $\cos$ and $\tan$ for large arguments. The present paper fills this gap, thus making feasible the computation of the worst cases of all the common univariate functions over their full domains of definition, for the IEEE double precision.

More precisely, we study the specific case where we are trying to find the worst cases of a periodic function for very

*INRIA/LORIA, Projet CACAO, Bâtiment A, 615 rue du jardin botanique, F-54602 Villers-lès-Nancy Cedex, guillaume.hanrot@loria.fr

†INRIA/LIP, École Normale Supérieure de Lyon, 46 allée d'Italie, F-69364 Lyon Cedex 07, vincent.lefevre@ens-lyon.fr

‡CNRS/LIP, École Normale Supérieure de Lyon, 46 allée d'Italie, F-69364 Lyon Cedex 07, damien.stehle@ens-lyon.fr. Work started when the author was hosted and partially funded by the MAGMA group, within the University of Sydney.

§INRIA/LORIA, Projet CACAO, Bâtiment A, 615 rue du jardin botanique, F-54602 Villers-lès-Nancy Cedex, paul.zimmermann@loria.fr

18th IEEE Symposium on Computer Arithmetic(ARITH'07)
0-7695-2854-6/07 $20.00

large arguments. In a nutshell, the problem in that setting is that *a priori*, we can no longer use a small degree polynomial approximation. Indeed, the sampling is so sparse that we lose any smoothness: even if x and x' are two consecutive floating-point numbers, the values $f(x)$ and $f(x')$ can be completely different. Using the periodicity of the function and a range reduction, we present an idea grouping input numbers by arithmetic progressions, allowing to recover polynomial approximations. By using the methods described above, we devise a family of algorithms finding worst cases for a set of N consecutive floating-point numbers, especially suited to the case of large arguments. The complexities of these algorithms range from $N^{4/5+\varepsilon}$ down to $N^{7-2\sqrt{10}+\varepsilon} \leq N^{0.676}$.

Roadmap of the paper. In §2, we describe precisely the tackled problem and state our main result. In §3, we briefly recall the main known methods to find worst cases of univariate functions: we will use these algorithms as subroutines. In §4, we describe the algorithm finding the worst cases of periodic functions for large arguments. This algorithm is analysed in §5. Finally, in §6, we demonstrate the efficiency of the algorithm, by giving some bad cases for $\sin x$ in IEEE-754 double precision.

Notation. If $x \in \mathbb{R}$ and $y \in \mathbb{R}\backslash\{0\}$, we define $x \bmod y$ to be a representative of $x + \mathbb{Z}y$ which lies in $[-y/2, y/2]$, i.e., the difference between x and a closest multiple of y, taking whatever choice if x is exactly the middle of two consecutive multiples of y. In this paper, we define a floating-point number as a fraction of the form $r = m \cdot 2^{-t}$, where m and t are integers. For such numbers, we define $\mathrm{size}\,(r) = \mathrm{size}\,(m) + |t|$. For complexity statements, we adopt the bit-complexity model, and let $\mathcal{P}(n_1, \ldots, n_k)$ denote some polynomial in $n_1, \ldots, n_k$. We define $[\![a, b]\!] = [a, b] \cap \mathbb{Z}$, for any $a, b \in \mathbb{R}$. If $\vec{b} = (b_1, \ldots, b_d)$ is a vector, its L_1-norm is denoted by $\|\vec{b}\|_1 = \sum_{i=1}^{d} |b_i|$. Finally, if P and Q are univariate polynomials, we denote their resultant by $\mathrm{Res}(P, Q)$.

2. Searching for Bad Cases of a Periodic Function

In the whole paper, we consider floating-point numbers in radix 2 and precision p:

$$x = \pm m \cdot 2^e,$$

where $e \in \mathbb{Z}$ is the exponent of x, and m is the p-bit significand in $[1/2, 1)$, i.e., with $m \in 2^{-p} \cdot [\![2^{p-1}, 2^p - 1]\!]$. However, the results presented here are radix-independent. Indeed, in §4.1, the algorithm is described in terms of powers λ, μ of the radix, and nothing is specific to the binary case. See also [6].

Definition 1 *The* significand $m(y)$ *and the* exponent $\mathrm{Exp}(y)$ *of a non-zero real number y are defined by* $|y| = m(y) \cdot 2^{\mathrm{Exp}(y)}$ *such that* $1/2 \leq m(y) < 1$, *and* $\mathrm{Exp}(y) \in \mathbb{Z}$.

A bad case of a function f is a floating-point number x for which the value $f(x)$ is hard to round in the given precision:

Definition 2 *Let f be a real valued function and $\varepsilon > 0$. An ε-bad case of f in precision p is a real x such that:*

$$|2^p \cdot m(f(x)) \bmod 1| \leq \varepsilon.$$

A worst case of the function f over a finite set $\mathcal{M}$ is any input $x \in \mathcal{M}$ minimising the quantity $|2^p \cdot m(f(x)) \bmod 1|$.

In other words, an ε-bad case corresponds to an $f(x)$ at distance less or equal to ε unit in last place (ulp) from the nearest p-bit floating-point number. This definition corresponds to bad cases for the directed rounding modes. This is enough to cover all IEEE-754 rounding modes, since any ε-bad case for the rounding to nearest mode in precision p is a (2ε)-bad case in precision $p+1$ for directed rounding.

Let f be a periodic function, with period Π. We want to find bad cases of f when the input numbers x are much larger than the period. Apart from the naive method — compute $f(x)$ with sufficient precision for all x in the studied domain —, the classical methods (Lefèvre, SLZ) work in the following way:

1. Split the interval under study into sub-intervals.
2. In each sub-interval, approximate the function f by a degree-d polynomial P, e.g., a Taylor polynomial. If the approximation error is bounded by $\varepsilon_2 \mathrm{ulp}(f(x))$, the ε_1-bad cases of f must be $(\varepsilon_1 + \varepsilon_2)$-bad cases of P.
3. The $(\varepsilon_1 + \varepsilon_2)$-bad cases of P are computed, by an *ad hoc* method: Lefèvre's (based on continued fraction expansion) for $d = 1$, or SLZ (based on Coppersmith's method) for $d \geq 2$.
4. Check if the $(\varepsilon_1 + \varepsilon_2)$-bad cases of P found are ε_1-bad for f.

In our setting, the difference between two consecutive machine numbers may be so large that small-degree polynomial approximations are valid in intervals with too few floating-point numbers, thus we cannot use classical methods like Lefèvre's or the SLZ algorithms directly.

Example. Consider $\sin x$ in $[2^{1023}, 2^{1024})$, which corresponds to the largest binade of the IEEE-754 double precision. The difference between two consecutive machine numbers in that binade is $\mu = 2^{971}$, which is a huge quantity compared to the period of the sine function. Reducing μ modulo 2π does not help either, since $\mu \approx 1.95 \bmod 2\pi$. This example shows the two problems we are faced:

1. an argument reduction is needed to reduce the (large) input x to the fundamental interval $[-\Pi/2, \Pi/2]$;

2. even after that argument reduction, consecutive floating-point numbers x and x' in precision p give unrelated values $f(x)$ and $f(x')$, since $\mathrm{ulp}(x) \bmod \Pi$ is usually not small.

To demonstrate the second problem, consider the first floating-point numbers $x_i = (2^{52}+i)\mu$ of the largest IEEE-754 double precision binade, still with $\mu = 2^{971}$; the corresponding values of $\sin x_i$ are: $\sin x_0 \approx 0.563$, $\sin x_1 \approx -0.976$, $\sin x_2 \approx 0.160$, $\sin x_3 \approx 0.858$, $\sin x_4 \approx -0.795$. We see that even determining *a priori* the binade $[2^{h-1}, 2^h)$ in which $|f(x)|$ lies is a non-trivial problem (see §4.3).

In the present paper, we describe the first non-naive algorithm that finds the worst cases of a periodic function for large arguments. More precisely:

Theorem 1 *Let f be a periodic $\mathcal{C}^\infty$ function. Given as input a precision p, an exponent e and a bad case bound ε, the algorithm described in §4 finds all the values x in $2^{e-p} \cdot [\![2^{p-1}, 2^p - 1]\!]$ such that $|2^p \cdot m(f(x)) \bmod 1| \leq \varepsilon$. Moreover, if one chooses the parameter $d = 5$, if $\varepsilon = 2^{-p+O(1)}$ and if $\alpha \to \infty$, then the running time of the algorithm can be heuristically bounded by $2^{p\cdot(7-2\sqrt{10}+o(1))}$, after a precomputation of precision $e + O(1)$ bits.*

Note that since the function is $\mathcal{C}^\infty$ and periodic, its successive derivatives are bounded over $\mathbb{R}$. This fact will prove important when we will use the complexity results on the SLZ algorithm.

3. Lefèvre's and SLZ Algorithms

Two non-naive algorithms are known to find the worst cases of a univariate function. Suppose that we want to search for the worst cases of a function f, where f is $\mathcal{C}^\infty$ with uniformly bounded derivatives. For example, we can consider $f = \sin$ over $[1/2, 1)$ with $p = 53$. We expect the worst case to be an ε-bad case for $\varepsilon \approx 2^{-p}$. We are thus interested in solving equations of the type:

$$|\lambda \cdot f(\mu t) \bmod 1| \leq \varepsilon \text{ with } t \in [\![t_0, t_0 + N]\!] \qquad (1)$$

where λ, μ, and ε are positive real numbers and $t_0, N \in \mathbb{Z}$.

In the classical search for directed rounding, if we consider the restriction of f over the input interval $[2^{e-1}, 2^e)$, and assuming all outputs are in $[2^{h-1}, 2^h)$, we have $\lambda = 2^{p-h}$, $\mu = 2^{e-p}$, $t_0 = N = 2^{p-1}$, and $\varepsilon = 2^{-p+O(1)}$.

Lefèvre's and the SLZ algorithms can be adapted to solve Equation (1). In Lefèvre's algorithm, the interval $[\![t_0, t_0 + N]\!]$ is subdivided into sub-intervals; on each of these sub-intervals, the function f is approximated by a linear function; finally, the bad cases of the linear functions are computed with a variant of Euclid's algorithm for computing greatest common divisors.

In the SLZ algorithm, the linear approximations are replaced by higher degree approximations (though usually still quite small degrees), and Euclid's algorithm is replaced by the LLL-based Coppersmith method for computing the small roots of bivariate polynomials modulo an integer. The SLZ algorithm was originally described in [8]. Its complexity analysis was improved later in [7].

The SLZ algorithm is described in Figure 1. This description is slightly different from the one of [7]: the quantities $1/\mu$ and ε can now be real numbers. This modification does not create any problem for the correctness and complexity analysis of the algorithm. The following theorem is

Input: Integers $t_0, N > 0$, real numbers $\lambda, \mu, \varepsilon > 0$.
Parameters: $T, d, \alpha \in \mathbb{Z}$.
Output: All the solutions $t \in [\![t_0, t_0 + N]\!]$ to Equation (1).

1. $n := \frac{(\alpha+1)(d\alpha+2)}{2}$, $r := \frac{\alpha(\alpha+1)}{2}$, $T' := T$, $S := \emptyset$, $\{e_1, \ldots, e_n\} := \{x^i y^j, i + dj \leq d\alpha\}$.
2. $t := t_0$. While $t \leq t_0 + N$, do
3. If $t + 2T' \geq t_0 + N$, $T' := \left\lfloor \frac{t_0+N-t}{2} \right\rfloor$.
4. If $T' = 0$, then
5. Add t in S if it is solution to Equation (1).
6. $t := t + 1$, $T' := T$.
7. Else
8. $t_m := t + T'$, $P(x) := f(\mu t_m) + f'(\mu t_m)\mu x + \ldots + \frac{1}{d!} f^{(d)}(\mu t_m)(\mu x)^d$.
9. $\varepsilon' := \left(\max_{x\in[0,1]} \left|f^{(d+1)}(x)\right|\right) \cdot \frac{\lambda}{(d+1)!}(\mu T')^{d+1}$.
10. $\{g_1, \ldots, g_r\} := \{x^i (\lambda P(x) + y)^j, i + j \leq \alpha\}$.
11. Create the $r \times n$ matrix B such that $B_{k,l}$ is the coefficient of the monomial e_l in the polynomial $g_k(xT', (\varepsilon + \varepsilon')y)$.
12. LLL-reduce the rows of B. Let $\vec{b}_1, \vec{b}_2$ be the two shortest vectors of the reduced basis.
13. $z := 1$. If $\|\vec{b}_1\|_1 \geq 1$ or $\|\vec{b}_2\|_1 \geq 1$, $z := 0$.
14. Let $Q_1(x, y), Q_2(x, y)$ be the polynomials corresponding to $\vec{b}_1$ and $\vec{b}_2$.
15. $R(x) := \mathrm{Res}_y(Q_1(x, y), Q_2(x, y))$. If $R(x) = 0$, then $z := 0$.
16. If $z = 0$, then $T' := \lfloor T'/2 \rfloor$.
17. Else, for any root x of R belonging to $[\![-T', T']\!]$, add $t_m + x$ in S if it is a solution to Equation (1), $t := t + 2T' + 1$, $T' := T$.
18. Return S.

Figure 1. The SLZ algorithm.

a direct consequence of the main result of [7].

Theorem 2 *Let $d \geq 1$ be a constant integer. Let t_0, N be positive integers and $\lambda, \mu, \varepsilon$ be positive reals. Suppose that $1/\mu \geq t_0 + N$. Let T, α be positive integers,*

with $T \leq N$. Given t_0, N, λ, μ, ε, T, d and α as input, the algorithm of Figure 1 outputs all the solutions to Equation (1). Moreover, if the variable z is never set to 0 at Step 15, then the algorithm finishes in time:

$$\mathcal{P}\left(\text{size}\left(\mu\right), \text{size}\left(\lambda\right), \text{size}\left(\varepsilon\right), \alpha\right) \cdot \frac{N}{T},$$

as long as $\varepsilon \leq \mu\lambda$ and:

$$\begin{aligned}\log_2 T \quad \leq \quad & \min\left(n_1 - \frac{m + n_2 + O(1)}{d+1}, \right. \\ & \left. n_1 - \frac{(n_1+n_2)^2}{4(m+n_2)}(1+\varepsilon_1) + \varepsilon_2\right),\end{aligned}$$

with $n_1 = -\log_2 \mu, n_2 = \log_2 \lambda, m = -\log_2 \varepsilon$ and, for α growing to ∞:

$$\begin{aligned}\varepsilon_1 \quad &= \quad O(1/\alpha) \\ \varepsilon_2 \quad &= \quad \frac{1}{m+n_2}O(\alpha^2) + \frac{n_1}{m+n_2}O(\alpha) \\ & \qquad +(m+n_1+n_2)O(1/\alpha).\end{aligned}$$

Note that the complexity statement of the theorem above is only heuristic. The claimed complexity bound holds as long as the variable z is never set to 0 at any Step 15 of the algorithm, which means that all pairs of bivariate polynomials whose resultants are computed are algebraically independent. This assumption seems to be satisfied in practice for any non-algebraic function f.

By using $\lambda = 2^p$, $\mu = 2^{-p}$, $t_0 = N = 2^{p-1}$, $\varepsilon = 2^{-p+O(1)}$, $\alpha \to \infty$ while $\alpha = o(p)$, and $d = 3$, this provides an algorithm to find the worst case of a function $f : [1/2, 1) \to [1/2, 1)$ with precision p in heuristic time $2^{p(1/2+o(1))}$. This is the best algorithm known so far for solving this problem when the target precision p goes to infinity. (We are mainly interested here in the case of the IEEE-754 double precision, i.e., $p = 53$.)

4. The Algorithm

4.1. High Level Description

Analyzing why the original SLZ algorithm fails for large arguments of periodic functions leads one to the solution. The key remark is that, though consecutive values of x in the range under study yield unrelated values of $f(x)$, non-consecutive values of x can yield very close values of $f(x)$.

Indeed, consider the values of $x \bmod \Pi$, where Π is the period of f. (Here, and in the rest of the paper, we shall assume that Π is an irrational number.) Since we have many values of $x \bmod \Pi$ uniformly distributed in $[-\Pi/2, \Pi/2]$, the idea is to split this latter interval into sub-intervals over which we will be able to use polynomial approximations of f.

There is still one difficulty to overcome: the methods mentioned above require the floating-point numbers x we consider as potential bad cases to be in some *arithmetic progression*. Our solution to this problem is illustrated in Figure 2. Remember that we want to solve Equation (1):

$$|\lambda \cdot f(\mu t) \bmod 1| \leq \varepsilon \text{ with } t \in [\![t_0, t_0 + N]\!].$$

The idea is the following: consider an integer q such that $\tau := q\mu \bmod \Pi$ is small. A natural choice is to take for q the denominator of a continued fraction convergent from μ/Π. Each $t \in [\![t_0, t_0 + N]\!]$ can be uniquely written $t = sq + r$, where $0 \leq r < q$. Then μt in Equation (1) becomes $\tau s + \mu r \bmod \Pi$, and thus Equation (1) is now:

$$|\lambda \cdot g_r(\tau s) \bmod 1| \leq \varepsilon \text{ with } s \in \left[\!\!\left[\frac{t_0 - r}{q}, \frac{t_0 + N - r}{q}\right]\!\!\right],$$

where $g_r(x) := f(r\mu + x)$. We have thus transformed a problem for the function f with parameters λ, μ, t_0, N into q similar problems for the functions g_r, $0 \leq r < q$, with parameters $\lambda, \tau, \frac{t_0 - r}{q}, N/q$.

Writing $x_0 = t_0\mu$ and $x_N = (t_0 + N)\mu$, we then decompose the interval $[\![t_0, t_0 + N]\!]\,\mu$ as

$$[x_0, x_N] = \bigcup_{r=0}^{q-1} \mu \cdot A(q, r)$$

where $A(q, r)$ is the part of the arithmetic progression $r \bmod q$ which is within the interval $[t_0, t_0 + N]$, and the notation $\mu\cdot$ means that we apply a homothety (a scaling) of factor μ.

Let $I_r = \mu \cdot A(q, r)$ be an interval in the partition defined above. Over this interval, we have $f(\mu t) = f(\mu q s + \mu r) = f(\tau s + \mu r)$. The point is now that, τs being small over this latter interval, if f is regular enough, $f(\tau s + \mu r)$ can rightfully be approximated by the Taylor polynomial

$$f(\mu r) + \tau s f'(\mu r) + \sum_{k=2}^{d} (\tau s)^k \frac{f^{(k)}(\mu r)}{k!},$$

with an error of the order of $(\tau s)^{d+1}$. We are thus back to the classical case, where Lefèvre's or the SLZ algorithms apply. One difference is that the distance between two consecutive numbers to check is τ which is irrational instead of 2^{e-p}, but this makes no difference for Lefèvre's algorithm or SLZ.

Following the principle of worst cases algorithms, we might have to split our "intervals" I_r into smaller "sub-intervals": in that case, we split each arithmetic progression $A(q, r)$ into sub-progressions[1].

[1] We do not write "intervals" any more, since the numbers are not adjacent; however these subsets of numbers still form a partition of the N numbers to test.

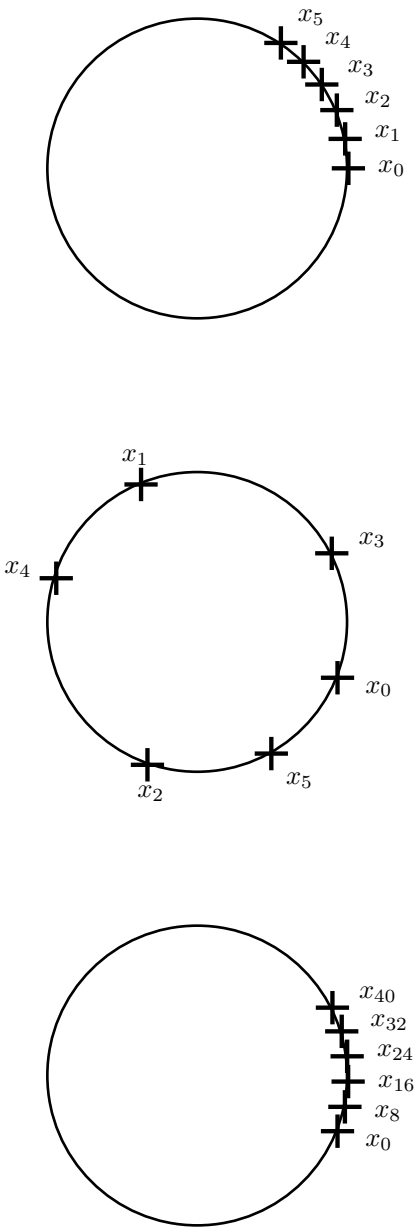

Figure 2. Top: Over $[0, 2\pi)$, successive floating-point numbers are close and in arithmetic progression. Middle: For a large binade, successive floating-point numbers are not necessarily close modulo 2π; here, the first points of the binade $[2^{69}, 2^{70}]$. Bottom: For a large binade, there exists q such that the first, the $(q+1)$-th, the $(2q+1)$-th, ... floating-point numbers are close modulo 2π and in arithmetic progression. Here, same binade as above, with $q = 8$.

In order to reduce the error coming from the use of Taylor's formula, we want τ to be as small as possible. This implies that q must be large, since one can expect $\tau \approx \Pi/q$. However, using a too large q is not a good idea since we have to deal with q different progressions. The solution of this optimization problem is given in §5.

Input: Integers $t_0, N > 0$ and real numbers $\lambda, \mu, \varepsilon, \Pi > 0$, with μ "large".
Parameters: $T, Q, d, \alpha \in \mathbb{Z}$.
Output: All the solutions $t \in [\![t_0, t_0 + N]\!]$ to Equation (1).

1. Find the largest continued fraction convergent $q \leq Q$ of the quantity μ/Π.
2. Compute $\tau \approx q\mu \bmod \Pi$ with the algorithm of Figure 4.
3. **for** $t := t_0$ **to** $t_0 + q - 1$ **do**
4. Apply the SLZ algorithm (Figure 1) with $t_0 = t$, $N = \lfloor (t_1 - t)/q \rfloor$, $\mu = \tau$, $\lambda = 2^{p - \mathrm{Exp}(f(\mu t))}$, and with the parameters d, α and $T := N$.

Figure 3. Sketch of the main algorithm

4.2. Required Working Precision

We study here the precision required in the different steps of the main algorithm (Figure 3). Only one large argument reduction is required, namely for the computation of $q\mu \bmod \Pi$; all subsequent computations can be performed with precision $O(p)$.

Lemma 1 *The worst case search with large arguments in a binade $[2^{e-1}, 2^e]$ can be implemented with a working precision of $3p + O(1)$ bits, after a precomputation with $e + 3p$ bits.*

Proof. Since the expected worst cases correspond to $\varepsilon \approx 2^{-p}$, it suffices to compute $f(x)$ with relative precision $2p + O(1)$. Since $x = \mu r + \tau s \bmod \Pi$ with $r < 2^p$ and $|\tau s| < 1$, it suffices to compute μ with relative precision $3p + O(1)$ and τ with relative precision $2p + O(1)$, which is what the algorithm in Figure 4 does.

Take a function f of period Π such that $2^{h-1} \leq \Pi < 2^h$. We denote by $\circ_w(\cdot)$ the rounding to nearest in precision w.

The error on α is at most 2^{h-w-1}; since $2^{h-1} \leq \alpha \leq 2^h$, we have $k \leq 2^{e-(h-1)}$, thus $k\alpha \leq 2^{e+1}$, and the error on $\circ_w(k\alpha)$ is at most 2^{e-w+1}. Now $\mathrm{Exp}(\beta) \leq h$, and the error on β is bounded by $2^{-3p-1} + 2^{e-w+1} \leq 2^{-3p+2}$.

The error on γ is bounded by $2^{-2p-1} + 2^{-2p+2} \leq 2^{-2p+3}$; we have $l \leq 2^{p+1}$, thus $l\alpha \leq 2^{p+h+1}$, and the error on $\circ_{h+3p}(l\alpha)$ is bounded by $2^{-2p} + 2^{h-e-2p} \leq 2^{-2p+1}$. Finally $|\tau| \leq 2^h$ and the error on τ is bounded by $2^{-2p-1} + 2^{-2p+3} + 2^{-2p+1} \leq 2^{-2p+4}$. □

Example: Consider $\sin x$ for x an IEEE-754 double precision number in $[2^{1023}, 2^{1024})$. We have $\mu = \mathrm{ulp}(x) =$

Input: A precision p, $\mu = 2^e$ with $e \geq h$ where $h = \mathrm{Exp}(\Pi)$, an integer $q < 2^p$.
Output: Approximations of $\mu \bmod \Pi$ and $q\mu \bmod \Pi$.

1. $w := e + 3p$.
2. $\alpha := \circ_w(\Pi)$.
3. $k := \lfloor \circ_w(2^e/\alpha) \rceil$.
4. $\beta := \circ_{h+3p}(2^e - \circ_w(k\alpha))$.
5. $\gamma := \circ_{h+3p}(q\beta)$.
6. $l := \lfloor \circ_{3p}(\gamma/\alpha) \rceil$.
7. $\tau := \circ_{h+2p}(\gamma - \circ_{h+3p}(l\alpha))$.
8. Return $\beta \approx \mu \bmod \Pi, \tau \approx q\mu \bmod \Pi$.

Figure 4. Approximation of $\mu \bmod \Pi$ and $q\mu \bmod \Pi$.

$2^{971} \approx 1.95 \bmod (2\pi)$. Consider $q = 15106909301$, then $\tau = q\mu \bmod (2\pi) \approx 0.441 \cdot 10^{-12}$. The next convergent has denominator $q = 14233796029594$, which gives $\tau \approx -0.757 \cdot 10^{-13}$. Note that the value of q depends on the considered binade: for the binade $[2^{511}, 2^{512})$, where $\mu = \mathrm{ulp}(x) = 2^{459} \approx 0.109 \bmod (2\pi)$, we can choose $q = 93888452023$, which gives $\tau \approx 0.371 \cdot 10^{-11}$, or $q = 1668824993486$, which gives $\tau \approx -0.101 \cdot 10^{-11}$.

Remark. The idea described above (splitting the values of x according to their repartition modulo Π) can be pushed further; given a value $x_0 + k\mu$ where $x_0 = t_0 2^{e-p}$ and $\mu = \mathrm{ulp}(x_0)$, one can give a complete description of the set of integers ℓ such that $x_0 + \ell\mu \bmod \Pi$ is close to $x_0 + k\mu \bmod \Pi$, i.e., $(\ell - k)\mu \bmod \Pi \approx 0$, in terms of the denominators q_i of the convergents of the continued fraction of μ/Π. However, if using this idea reduces the number of intervals under study, it significantly increases the number of variables (an element ℓ being described as $\sum_i a_i q_i + r$ for a_i in a given interval). The fact that Coppersmith's method behaves badly when the number of variables increases seems, however, to make this refinement of our idea pointless.

4.3. Determining the Output Exponents

An additional problem is to compute the output exponent — $\mathrm{Exp}(f(\mu t))$ in Figure 3 — for a given subset $\{(sq + r)\mu, t_0 \leq sq + r \leq t_0 + N\}$, and to check that exponent is constant on that subset. Let $x_0 = r'\mu \bmod \Pi$, $t_0 \leq r' < t_0 + q$, and $\tau = q\mu \bmod \Pi$. The corresponding reduced subset is $\{x_0 + s\tau, s \in S\}$ for some interval S of width less than $\lceil N/q \rceil$. The arguments in the reduced subset are thus all in the interval $[x_0, x_0 + h]$ with $h = \tau\lceil N/q \rceil$. The width h is usually small: since $\tau \leq \Pi/q$, we have $h \leq \Pi N/q^2$. For example with the largest IEEE-754 double precision binade $[2^{1023}, 2^{1024})$ and the sine function with $q = 15106909301$, one gets $\lceil N/q \rceil = 298116$ and $h \approx 0.132 \cdot 10^{-6}$.

For the sine function, it suffices to check that $\sin x_0$ and $\sin(x_0 + h)$ lie in the same binade. Indeed, the sine function is decreasing on $[-\pi, -\pi/2]$, increasing on $[-\pi/2, \pi/2]$, and then decreasing on $[\pi/2, \pi]$. Around the points $-\pi/2$ and $\pi/2$ where the derivative sign changes, one has $1/2 \leq |\sin x| \leq 1$ in an interval of width $2\pi/3 > h$.

In practice, one does not need to evaluate $\sin x_0$ and $\sin(x_0 + h)$, since $\sin x$ admits a constant exponent on $[\sin^{-1} \frac{1}{2^k}, \sin^{-1} \frac{1}{2^{k-1}}]$; given precomputed approximations a and b such that $\sin^{-1} \frac{1}{2^k} \leq a$ and $b \leq \sin^{-1} \frac{1}{2^{k-1}}$, it suffices to check that $[x_0, x_0 + h] \subset [a, b]$.

5. Complexity Analysis

The algorithm consists in solving the equations:

$$|\lambda \cdot f(r\mu + \tau s) \bmod 1| \leq \varepsilon,$$

with $0 \leq r < q$ and $s \in \left[\!\left[\frac{t_0 - r}{q}, \frac{t_0 + N - r}{q} \right]\!\right]$. Note that, by the classical Dirichlet theorem, for any Q one can find $0 \leq q \leq Q$ such that the corresponding τ has $|\tau| \leq \Pi/Q = O(1/Q)$.

The overall cost of the algorithm is thus bounded by Q times the cost of solving a single of the above equations. These equations can be solved by using Lefèvre's algorithm or the SLZ algorithm. To analyse the algorithm of the previous section, it thus suffices to adequately use Theorem 2. We are to apply this theorem with $\mu := \tau$, $\lambda := 2^{p - \mathrm{Exp}(f(x))}$, $N := 2^{p-1}/q$ and $\varepsilon \geq 2^{-p}$. Several parameters can be set in order to optimise the complexity of the algorithm: the degree d of the polynomials approximations, the parameter α of Coppersmith's method, the upper bound Q for the chosen convergent of the continued fraction expansion of μ/Π, the size T of the sub-intervals within Lefèvre's algorithm or the SLZ algorithm, and the quality ε of the computed bad cases.

We now use Theorem 2. For our choice of parameters, the condition "$\varepsilon \leq \mu\lambda$" is satisfied[2]. We fix a constant parameter d.

Let $m = -\log_2 \varepsilon$. With the given parameters, we have $\varepsilon_1 = O(1/\alpha)$ and $\varepsilon_2 = \frac{1}{p} O(\alpha^2) + O(\alpha) + pO(1/\alpha)$. Let $\varepsilon_3 > 0$ be an arbitrarily small constant. We fix α so that $|\varepsilon_1| \leq \varepsilon_3$ and $|\varepsilon_2| \leq \varepsilon_3 \cdot p$ when p is larger than some p_0. Our goal is to maximise the parameter T under

[2] We have $\mu\lambda = \tau 2^{p - \mathrm{Exp}(f(x))}$. Since f is periodic and $\mathcal{C}^\infty$, it is bounded in absolute value by some constant C, which together with $\tau \geq 2^{-p}$ yields $\mu\lambda \geq 1/C$, and on the other hand ε goes to zero.

the following conditions:

$$\begin{aligned}\log_2 T &\leq \min\left(n_1 - \frac{m+p}{d+1}, \right.\\ &\qquad \left. n_1 - \frac{(n_1+p)^2}{4(m+p)}(1+\varepsilon_3) - \varepsilon_3 \cdot p\right),\\ m &\leq p\\ T &\leq 2^p/q\end{aligned}$$

where $n_1 = -\log_2 \mu$. We now choose to set $T = 2^{p-n_1-c}$ for an adequate constant $c \geq 0$, which fulfills the third condition (remember $\mu = \tau = O(1/q)$). With this choice for T, the first condition is implied by:

$$\frac{2n_1}{1+\varepsilon_3} \geq O(1) + p + \max\left(\frac{m+p}{d+1}, \frac{(n_1+p)^2}{4(m+p)}\right). \quad (2)$$

Since the overall complexity is roughly $2^p/T$ and $T = 2^{p-n_1-c}$, we are trying to minimise n_1. For any fixed degree d, the best n_1 is reached when both values maximised are equal. This gives the relation:

$$\frac{2}{1+\varepsilon_3}n_1 - p = O(1) + \frac{m+p}{d+1} = O(1) + \frac{(n_1+p)^2}{4(m+p)}.$$

There exists a constant c' such that the following choice of parameters satisfies Equation (2):

$$\begin{aligned}n_1 &= (1+c'\varepsilon_3)\frac{8d+9+6\sqrt{d+1}}{16d+15} \cdot p + O(1),\\ m &= \frac{12(d+1)(\sqrt{d+1}-1)-d}{16d+15} \cdot p.\end{aligned}$$

Furthermore, this choice of parameters is extremely close to the optimal choice.

Unfortunately, it implies $\varepsilon \leq 2^{-p}$ as soon as $d \geq 5$. This means that to take advantage of a larger d, we would need a much smaller ε, but we would no longer find any ε-bad case: no ε-bad case with ε significantly smaller than 2^{-p} is expected to exist. If we fix $m = p - O(1)$, then Equation (2) can be rewritten as:

$$\frac{2}{1+\varepsilon_3}n_1 - p \geq O(1) + \max\left(\frac{2p}{d+1}, \frac{(n_1+p)^2}{8p+O(1)}\right).$$

With simple computations, we find the optimal solution $n_1 = (1+c''\varepsilon_3)(7-2\sqrt{10}) \cdot p$ and $d = 5$ for some constant $c'' > 0$.

This completes the proof of Theorem 1. The following table sums up the best parameters for $d \leq 5$.

d	Q	ε	T
1	$2^{(4/5)p} \approx 2^{0.800 \cdot p}$	$2^{-(1/5)p}$	$2^{(1/5)p}$
2	$2^{(3/4)p} \approx 2^{0.750 \cdot p}$	$2^{-(1/2)p}$	$2^{(1/4)p}$
3	$2^{(5/7)p} \approx 2^{0.714 \cdot p}$	$2^{-(5/7)p}$	$2^{(2/7)p}$
4	$\approx 2^{0.689 \cdot p}$	$\approx 2^{-0.889 \cdot p}$	$\approx 2^{0.311p}$
5	$\approx 2^{0.675 \cdot p}$	2^{-p}	$\approx 2^{0.325p}$

Remark: The analysis of this section does not provide the bounds given in the table above for $d = 1$ and $d = 2$. These bounds can be obtained by using Lefèvre's algorithm (for $d = 1$) and the refined analysis of the SLZ algorithm in the particular case $d = 2$, which can be found in [9].

6. Numerical Results and Conclusion

We have implemented the algorithm from §4 on top of the GMP library [2]. To demonstrate the efficiency of the algorithm, we have applied it to the largest IEEE-754 double precision binade, namely $[2^{1023}, 2^{1024})$, for the sine function. In that binade, no bad cases were known so far, even for a moderately small $\varepsilon \approx 2^{-30}$.

6.1. Estimates of the Running Time for the $[2^{1023}, 2^{1024})$ Binade

This binade corresponds to $\mu = \mathrm{ulp}(x) = 2^{971}$. We used the value $q = 15106909301$, with the parameters $d = 3$ (degree-3 Taylor approximation) and $\alpha = 2$. The integer q is a denominator of a convergent of $\mu/(2\pi)$, and gives $\tau = q\mu \bmod (2\pi) \approx 0.441 \cdot 10^{-12}$. For that value of q, each arithmetic progression $sq + r$ for a given r has at most 298116 elements in the range $[2^{52}, 2^{53})$, thus the maximal value of T' for the SLZ algorithm (Figure 1) is about 149058. Those values correspond to an error $\varepsilon' \approx 7 \cdot 10^{-15}$ at step 9 of the algorithm (in comparison, degree $d = 2$ would give $\varepsilon' \approx 4 \cdot 10^{-7}$ only).

With those parameters, we were able to find the following bad cases — for directed rounding — in a few days of computing time, where 1^{43} is a shortcut for a sequence of 43 consecutive ones:

$$\sin(4621478864517314 \cdot 2^{971}) = -0.0\underbrace{10010110001110101110010000111100111100010000000100000}_{53}1^{43}0001\ldots$$

$$\sin(5501214608935005 \cdot 2^{971}) = 0.00\underbrace{10011000110011100101110100111100011011010010001111111}_{53}0^{45}1011\ldots$$

As a comparison, one can estimate the time needed to find such bad cases with a naive algorithm. On a 2.4 Ghz Opteron, our naive implementation, which does a single huge argument reduction for all tested values, can check one million of $\sin x$ values around $x = 2^{1023}$ in about 6 seconds, in double precision with $\varepsilon = 2^{-40}$. Then one needs about 2^{44} random trials to find a bad case with a run of 45 zeros or ones after the 53-bit significand. This would correspond to more than 3 expected years to find a 2^{-45}-bad case, and about 850 years to check the whole binade $[2^{1023}, 2^{1024})$. With an implementation of the algorithm from §4, checking 298116 values $t \cdot 2^{971}$ with $2^{52} \leq t = sq + r < 2^{53}$ takes only 0.008 second on the same machine. This corresponds to about 4 years to check the whole binade $[2^{1023}, 2^{1024})$.

6.2. Handling the Output Exponents

Checking the exponent ranges for all q arithmetic progressions with the method described in §4.3 took less than 90 minutes. We obtained the following repartition, where a row "exponent e" indicates the number of arithmetic progressions with constant exponent e, $2^{e-1} \leq |\sin x| < 2^e$, and the row "non-constant exponent" gives the number of remaining progressions where the exponent changes (two or more values):

exponent 0	10071272234
exponent -1	2605518908
exponent -2	1224792482
exponent -3	603844405
exponent -4	300884230
exponent -5	150312764
exponent -6	75139690
exponent -7	37567206
exponent -8	18782720
exponent -9	9390695
exponent -10	4694711
exponent -11	2346722
exponent -12	1172728
exponent -13	585732
exponent -14	292234
exponent -15	145484
exponent -16	72110
exponent -17	35421
exponent -18	17078
exponent -19	7906
exponent -20	3320
exponent -21	1028
non-constant exponent	29493
Total	15106909301

The same computation can be performed on the cosine function without any major modification. On the contrary, things worsen for the $\tan$ function, because it is not $\mathcal{C}^\infty$ over $\mathbb{R}$. This should not be a major issue since the input floating-point numbers x for which $\tan(x)$ is large can be computed, and then we could restrict the domain under study to $[-\pi/2+c, \pi/2-c]$ for some small constant $c > 0$, instead of $(-\pi/2, \pi/2)$. In this smaller interval, any successive derivative is bounded.

One of the main reasons for computing the tables of worst cases is to provide tight upper bound on the required internal precision to compute correctly the main elementary functions in double precision, e.g., all C99 univariate functions. It is thus crucial to guarantee that the found bad cases are indeed the worst. The algorithm described in the present paper, and those described in [5, 8] are proved correct, but their implementations could be incorrect. A natural question is thus the following: is it possible to have an independent verification of the worst cases found over a given domain, possibly by keeping part of some intermediate values computed by the algorithms?

Acknowledgements. The authors would like to thank the anonymous referees, whose comments helped to improve the presentation of the paper.

References

[1] COPPERSMITH, D. Finding small solutions to small degree polynomials. In *Proceedings of the 2001 Cryptography and Lattices Conference (CALC'01)* (2001), vol. 2146 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 20–31.

[2] *GNU MP: The GNU Multiple Precision Arithmetic Library*, 4.2.1 ed., 2006. `http://www.swox.se/gmp/#DOC`.

[3] IEEE STANDARDS COMMITTEE 754. ANSI/IEEE standard 754-1985 for binary floating-point arithmetic. Reprinted in SIGPLAN Notices, 22(2):9–25, 1987.

[4] LANG, T., AND MULLER, J.-M. Bounds on runs of zeros and ones for algebraic functions. In *Proceedings of the 15th IEEE Symposium on Computer Arithmetic (ARITH'15)* (2001), IEEE Computer Society, pp. 13–20.

[5] LEFÈVRE, V., AND MULLER, J.-M. Worst cases for correct rounding of the elementary functions in double precision. In *Proceedings of the 15th IEEE Symposium on Computer Arithmetic (ARITH'15)* (2001), IEEE Computer Society Press, pp. 111–118.

[6] LEFÈVRE, V., STEHLÉ, D., AND ZIMMERMANN, P. Worst cases for the exponential function in the ieee 754r decimal64 format. In *Reliable Implementation of Real Number Algorithms: Theory and Practice* (2006), P. Hertling, C. M. Hoffmann, W. Luther, and N. Revol, Eds., no. 06021 in Dagstuhl Seminar Proceedings, Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany. <http://drops.dagstuhl.de/opus/volltexte/2006/748>.

[7] STEHLÉ, D. On the randomness of bits generated by sufficiently smooth functions. In *Proceedings of ANTS VII* (2006), vol. 4078 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 257–274.

[8] STEHLÉ, D., LEFÈVRE, V., AND ZIMMERMANN, P. Worst cases and lattice reduction. In *Proceedings of the 16th Symposium on Computer Arithmetic (ARITH'16)* (2003), IEEE Computer Society Press, pp. 142–147.

[9] STEHLÉ, D., LEFÈVRE, V., AND ZIMMERMANN, P. Searching worst cases of a one-variable function. *IEEE Transactions on Computers 54*, 3 (2005), 340–346.

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