A New Range-Reduction Algorithm

N. Brisebarre, D. Defour, P. Kornerup, J.-M Muller and N. Revol

***Abstract*— Range-reduction is a key point for getting accurate elementary function routines. We introduce a new algorithm that is fast for input arguments belonging to the most common domains, yet accurate over the full double-precision range.**

***Index Terms*— Range-reduction, elementary function evaluation, floating-point arithmetic.**

I. INTRODUCTION

ALGORITHMS for the evaluation of elementary functions give correct results only if the argument is within a given small interval, usually centered at zero. To evaluate an elementary function $f(x)$ for any x, it is necessary to find some "transformation" that makes it possible to deduce $f(x)$ from some value $g(x^*)$, where

- x^*, called the *reduced argument*, is deduced from x;
- x^* belongs to the convergence domain of the algorithm implemented for the evaluation of g.

In practice, range-reduction needs care for the trigonometric functions. With these functions, x^* is equal to $x - kC$, where k is an integer and C an integer multiple of $\pi/4$. Also of potential interest is the case $C = \ln(2)$, for implementation of the exponential function.

A poor range-reduction method may lead to catastrophic accuracy problems when the input argument is large or close to an integer multiple of C. It is easy to understand why a poor range-reduction algorithm gives inaccurate results. The naive method consists of performing the computations

$$\begin{aligned} k &= \left\lfloor \frac{x}{C} \right\rfloor \\ x^* &= x - kC \end{aligned}$$

using the machine precision. When kC is close to x, almost all the accuracy, if not all, is lost when performing the subtraction $x - kC$. For instance, if $C = \pi/2$ and $x = 8248.251512$ the correct value of x^* is $-2.14758367 \cdots \times 10^{-12}$, and the corresponding value of k is 5251. Directly computing $x - k\pi/2$ on a calculator with 10-digit decimal arithmetic (assuming rounding to the nearest, and replacing $\pi/2$ by the nearest exactly-representable number), then one gets -1.0×10^{-6}. Hence, such a poor range-reduction would lead to a computed value of $\cos(x)$ equal to -1.0×10^{-6}, whereas the correct value is $-2.14758367 \cdots \times 10^{-12}$.

A first solution to overcome the problem consists of using arbitrary-precision arithmetic, but this may make the computation much slower. Moreover, it is not that easy to predict on the fly the precision with which the computation should be performed.

Most common input arguments to the trigonometric functions are small (say, less than 8), or sometimes medium (say, between 8 and approximately 2^{60}). They are rarely huge (say, greater than 2^{60}). We want to design methods that are fast for the frequent cases, and accurate for all cases. A rough estimate, based on SUN fdlibm library, is that the cost of trigonometric range-reduction – when reduction is necessary – is approximately one third of the total function evaluation cost.

First we describe Payne and Hanek's method [11] which provides an accurate range-reduction, but has the drawback of being fairly expensive in term of operations; this method is very commonly implemented, it is used in SUN fdlibm library in particular.

To know with which precision the intermediate calculations must be carried on to get an accurate result, one must know the *worst cases*, that is, the input arguments that are hardest to reduce. Also, to estimate the average performance of the algorithms (and to tune them so that these performances are good), one must have at least a rough estimate of the statistical distribution of the reduced arguments. These two problems are dealt with at the end of this section.

In the second section we present our algorithm dedicated to the reduction of small and medium size arguments. In the third section we compare our method with some other available methods, which justifies the use of our algorithm for small and medium size arguments.

A. The Payne and Hanek Reduction Algorithm

We assume in this subsection that we want to perform range-reduction for the trigonometric functions, with

Manuscript received ???, revised ???

David Defour is with Université de Perpignan, Perpignan, France; Nicolas Brisebarre, Jean-Michel Muller and Nathalie Revol are with Laboratoire LIP, CNRS/ENS Lyon/INRIA/Univ. Lyon 1, Lyon, France; Nicolas Brisebarre is also with Université Jean Monnet, Saint-Étienne, France; Peter Kornerup is with SDU, Odense, Denmark.

$$\overbrace{\boxed{\alpha_0.\alpha_{-1}\cdots\alpha_{n-e+2}}}^{\text{bits of Left}(e,p)}\overbrace{\boxed{\alpha_{n-e+1}\cdots\alpha_{-n-e-1-p}}}^{\text{bits of Middle}(e,p)}\overbrace{\boxed{\alpha_{-n-e-2-p}\alpha_{-n-e-3-p}\cdots}}^{\text{bits of Right}(e,p)}$$

Fig. 1. *The splitting of digits of $4/\pi$ in Payne and Hanek's reduction method.*

$C = \pi/4$, and that the convergence domain of the algorithm used for evaluating the functions contains[1] $I = [0, \pi/4]$. An adaptation to other cases is straightforward.

From an input argument x, we want to find the reduced argument x^* and an integer k, that satisfy:

$$k = \left\lfloor \frac{4}{\pi} x \right\rfloor \qquad x^* = \frac{\pi}{4}\left(\frac{4}{\pi}x - k\right) \tag{1}$$

Once x^* is known, it suffices to know k mod 8 to calculate $\sin(x)$ or $\cos(x)$ from x^*. If x is large, or if x is very close to a multiple of $\pi/4$, the direct use of (1) to determine x^* may require the knowledge of $4/\pi$ with very large precision, and a cost-expensive multiple-precision computation if we wish the range-reduction to be accurate.

Now let us present Payne and Hanek's reduction method [11], [12]. Assume an n-bit mantissa, radix 2 floating-point format (the number of bits n includes the possible hidden bit; for instance, with an IEEE double-precision format, $n = 53$). Let x be the positive floating-point argument to be reduced and let e be its unbiased exponent, so

$$x = X \times 2^{e-n+1}$$

where X is an n-bit integer satisfying $2^{n-1} \leq X < 2^n$. We can assume $e \geq -1$ (since if $e < -1$, no reduction is necessary). Let

$$\alpha_0.\alpha_{-1}\alpha_{-2}\alpha_{-3}\alpha_{-4}\alpha_{-5}\ldots$$

be the infinite binary expansion of $\alpha = 4/\pi$, and define an integer parameter p, used to specify the required accuracy of the range-reduction. Then rewrite $\alpha = 4/\pi$ as

$$\begin{aligned}&\text{Left}(e,p) \times 2^{n-e+2}\\ &\quad + (\text{Middle}(e,p) + \text{Right}(e,p)) \times 2^{-n-e-1-p},\end{aligned}$$

where $\text{Left}(e,p) = 0$ if $e < n+2$, else

$$\left\{\begin{array}{lcl}\text{Left}(e,p) & = & \alpha_0\alpha_{-1}\cdots\alpha_{n-e+2},\\ \text{Middle}(e,p) & = & \alpha_{n-e+1}\alpha_{n-e}\cdots\alpha_{-n-e-1-p},\\ \text{Right}(e,p) & = & 0.\alpha_{-n-e-2-p}\alpha_{-n-e-3-p}\cdots.\end{array}\right.$$

Fig. 1 shows the splitting of the binary expansion of α.

The basic idea of the Payne-Hanek reduction method is to notice that, if p is large enough, $\text{Middle}(e,p)$ contains the only bits of $\alpha = 4/\pi$ that matter for the range-reduction. Since

$$\begin{aligned}\frac{4}{\pi}x &= \text{Left}(e,p) \times X \times 8\\ &+ \text{Middle}(e,p) \times X \times 2^{-2n-p}\\ &+ \text{Right}(e,p) \times X \times 2^{-2n-p},\end{aligned}$$

the number $\text{Left}(e,p) \times X \times 8$ is a multiple of 8, so that once multiplied by $\pi/4$ (see Eq. (1)), it will have no influence on the trigonometric functions. $\text{Right}(e,p) \times X \times 2^{-2n-p}$ is less than 2^{-n-p}; therefore it can be made as small as desired by adequately choosing p.

How p is chosen will be explained in Section II-C.

B. Worst Cases

Assume we want the reduced argument to belong to $[-C/2, C/2)$. Define $x \bmod^* C$ as the number $y \in [-C/2, C/2)$ such that $y = x - kC$, where k is an integer.

There are two important points that must be considered when trying to design accurate yet fast range-reduction algorithms.

- First, what is the "worst case"? That is, what will be the smallest possible absolute value of the reduced argument for all possible inputs in a given format. That value will allow us to immediately deduce the precision with which the reduction must be carried on to make sure that, even for the most difficult cases, the returned result will be accurate enough.
- What is the statistical distribution of the smallest absolute values of the reduced arguments? That is, given a small value ϵ, what is the probability that the reduced argument will have an absolute value less than ϵ? This point is important if we want to design algorithms that are fast for the most frequent cases, and remain accurate on all cases.

Computing the worst case is rather easy, using an algorithm due to Kahan [4] (a C program that implements the method can be found at http://http.cs.berkeley.edu/~wkahan/. A Maple program is given in [9]). The algorithm uses the continued-fraction theory. For instance, a few minutes of calculation suffice to find the double-precision number between 8 and

[1] In practice, we can reduce to an interval of size slightly larger than C, to facilitate the reduction.

$2^{63}-1$ that is closest to a multiple of $\pi/4$. This number is:

$$\begin{aligned}\Gamma_{\pi/4} &= 6411027962775774 \times 2^{-48} \\ &\approx 22.776546738526000979.\end{aligned}$$

The distance between $\Gamma_{\pi/4}$ and the closest multiple of $\pi/4$ is

$$\epsilon_{\pi/4} \approx 3.094903 \times 10^{-19} \approx 0.71 \times 2^{-61}.$$

So if we apply a range-reduction from a double-precision argument in $[8, 2^{63}-1]$ to $[-\pi/4, \pi/4)$, and if we wish to get a reduced argument with relative accuracy better than $2^{-\mu}$, we must perform the range reduction with absolute error better than $2^{-\mu-61}$.

Also, the double-precision number greater than 8 and less than 710 which is closest to a multiple of $\ln(2)$ is:

$$\begin{aligned}\Gamma_{\ln(2)} &= 7804143460206699 \times 2^{-49} \\ &\approx 13.8629436111989061.\end{aligned}$$

The distance between $\Gamma_{\ln(2)}$ and the closest multiple of $\ln(2)$ is

$$\epsilon_{\ln(2)} \approx 1.972015 \times 10^{-17} > 2^{-56}.$$

In that case, we considered only numbers less than 710, since exponentials of numbers larger than that are mere overflows in double-precision arithmetic.

C. Statistical distribution of the reduced arguments

Now, let us turn to the statistical distribution of reduced arguments.

We assume that C is a positive fractional multiple of π or $\ln(2)$. Let e_{min} and e_{max} be two rational integers such that $2^{e_{min}} \leq C/2 < 2^{e_{min}+1}$ and $e_{min} \leq e_{max}$.

Let $p \in \mathbb{N}$ such that $2^{-p+1} \leq C$, our aim is to estimate the number of floating-point numbers x with n-bit mantissas and exponents between e_{min} and e_{max} such that

$$|x \bmod^* C| < 2^{-p}. \tag{2}$$

where $x \bmod^* C$ is defined as the unique number $y \in [-C/2, +C/2)$ such that $y = x - kC$, where k is an integer.

Let E be a rational integer such that $e_{min} \leq E \leq e_{max}$. As $2^{-p+1} \leq C$, we have $2^{-p} < 2^{e_{min}+1} \leq 2^{E+1}$. Therefore, $2^{-p} \leq 2^E$ i.e., $p + E \geq 0$.

We start by estimating the number of floating-point numbers x with n-bit mantissas and exponent E that satisfy (2). Hence, we search for the $j \in \mathbb{N}$, $2^{n-1} \leq j \leq 2^n - 1$ such that the inequality

$$\left| kC - \frac{j}{2^{n-1}} 2^E \right| < 2^{-p} \tag{3}$$

has solutions in $k \in \mathbb{Z}$. Such k necessarily satisfy

$$\frac{1}{C}\left(-\frac{1}{2^p} + \frac{j}{2^{n-1}}2^E\right) < k < \frac{1}{C}\left(\frac{1}{2^p} + \frac{j}{2^{n-1}}2^E\right). \tag{4}$$

We note that, as $p + E \geq 0$ and $j \geq 2^{n-1}$, the left hand-side of (4) is positive. Hence,

$$\underbrace{\max\left\{1, \left\lceil \frac{1}{C}\left(-\frac{1}{2^p} + 2^E\right)\right\rceil\right\}}_{m_E} \leq k \leq \underbrace{\left\lfloor \frac{1}{C}\left(\frac{1}{2^p} + 2^{E+1} - \frac{2^E}{2^{n-1}}\right)\right\rfloor}_{M_E} \tag{5}$$

since $2^{n-1} \leq j \leq 2^n - 1$, and these inequalities are sharp since the upper bound in (4) is irrational, and the lower bound is either zero or an irrational number. The number of possible k is exactly

$$N_E = M_E - m_E + 1. \tag{6}$$

Inequality (3) is equivalent to

$$\left| kC2^{n-1-E} - j \right| < 2^{n-1-p-E}. \tag{7}$$

Hence, for every k satisfying (5), there are exactly

$$\begin{aligned}&\min\left(2^n - 1, \lfloor kC2^{n-1-E} + 2^{n-1-p-E}\rfloor\right) - \\ &\quad \max\left(2^{n-1}, \lceil kC2^{n-1-E} - 2^{n-1-p-E}\rceil\right) + 1\end{aligned} \tag{8}$$

integers j solutions since the numbers $kC2^{n-1-E} - 2^{n-1-p-E}$ and $kC2^{n-1-E} + 2^{n-1-p-E}$ are irrational (we saw before that $k \neq 0$).

As $2^{-p+1} \leq C$, if $k \geq m_E + 1$, we have

$$2^{n-1} \leq \lceil kC2^{n-1-E} - 2^{n-1-p-E} \rceil$$

and, if $k \leq M_E - 1$, we have

$$2^n - 1 \geq \lfloor kC2^{n-1-E} + 2^{n-1-p-E} \rfloor.$$

Now, to analyse (8), we have to distinguish two cases.

First case: $2^{n-1-p-E} \geq 1/2$ i.e., $n - E \geq p$.
This case is the easy one, and equation (7) yields the conclusion. For every k, $m_E + 1 \leq k \leq M_E - 1$, there are exactly 2^{n-p-E} integer solutions j since the numbers $kC2^{n-1-E} - 2^{n-1-p-E}$ and $kC2^{n-1-E} + 2^{n-1-p-E}$ are irrational. When $k \in \{m_E, M_E\}$, we can only say that there are at least 1 and at most 2^{n-p-E} integer solutions j. Notice that these solutions can easily be enumerated by a program. Therefore, the number of floating-point numbers x with n-bit mantissas and exponent E that satisfy (2) is upper bounded by $N_E 2^{n-p-E}$, and lower bounded by $(N_E - 2)2^{n-p-E} + 2$.

Second case: $2^{n-1-p-E} < 1/2$ **i.e.** $n - E < p$.

We need results about uniform distribution of sequences [8] that we briefly recall now.

For a real number x, $\{x\}$ denotes the fractional part of x i.e. $\{x\} = x - \lfloor x \rfloor$ and $||x||$ denotes the distance from x to the nearest integer, namely

$$||x|| = \min_{n \in \mathbb{Z}} |x - n| = \min(\{x\}, 1 - \{x\}).$$

Let us recall the following definitions from [8].

Definition 1: Let $(x_n)_{n \geq 1}$ be a given sequence of real numbers. Let N be a positive integer.

For a subset $\mathcal{E}$ of $[0,1)$, the counting function $A(\mathcal{E}; N; (x_n))$ is the number of terms x_n, $1 \leq n \leq N$, for which $\{x_n\} \in \mathcal{E}$.

Let $y_1, \dots, y_N$ be a finite sequence of real numbers. The number

$$D_N((y_n)) = \sup_{0 \leq a < b \leq 1} \left| \frac{A([a,b); N; (y_n))}{N} - (b - a) \right|$$

is called the *discrepancy* of the sequence $y_1, \dots, y_N$. For an infinite sequence (x_n) of real numbers (or for a finite sequence containing at least N terms), $D_N((x_n))$ is meant to be the discrepancy of the initial segment formed by the first N terms of (x_n).

Thus, in particular, the number of values x_n with $1 \leq n \leq N$ satisfying $\{x_n\} \in [a,b)$, for any $0 \leq a < b \leq 1$, is bounded from above by $N\big[(b-a) + D_N((x_n))\big]$. Hence, the number of values $kC2^{n-1-E}$, with $m_E \leq k \leq M_E$, that satisfy equation (7), i.e. that satisfy $0 \leq \{kC2^{n-1-E}\} < 2^{n-1-p-E}$ or $1 - 2^{n-1-p-E} < \{kC2^{n-1-E}\} < 1$ is bounded from above by $N_E\big(2^{n-p-E} + 2D_{N_E}((kC2^{n-1-E}))\big)$.

Definition 2: Let μ be a positive real number or infinity. The irrational number α is said to be of type μ if μ is the supremum of all γ for which $\liminf_{\substack{q \to \infty, \\ q \in \mathbb{N}}} q^\gamma ||q\alpha|| = 0$.

Theorem 3.2 from [8, Chap 2.] states the following result.

Theorem 1: Let α be of finite type μ. Then, for every $\varepsilon > 0$, the discrepancy $D_N(u)$ of $u = (n\alpha)$ satisfies

$$D_N(u) = O(N^{(-1/\mu)+\varepsilon}).$$

Let us apply this theorem to values of interest for this paper, namely $C = q\ln(2)$ and $C = q\pi$ with $q \in \mathbb{Q}^*$.

- If C is a nonzero fractional multiple of $\ln(2)$. We know from [2] that any nonzero fractional multiple of $\ln(2)$ has a type ≤ 2.9. Thus, the number of floating-point numbers x with n-bit mantissas and exponent E that satisfy (2) is upper bounded by $2^{n-p-E}(N_E + O({N_E}^{(19/29)+\varepsilon}))$ for every $\varepsilon > 0$.
- If C is a nonzero fractional multiple of π. We know from [3] that any nonzero fractional multiple of π has a type ≤ 7.02. Hence, the number of floating-point numbers x with n-bit mantissas and exponent E that satisfy (2) is upper bounded by $2^{n-p-E}(N_E + O({N_E}^{(301/351)+\varepsilon}))$ for every $\varepsilon > 0$.

From this theorem, we can deduce the following result.

Proposition 1: Let C be a positive fractional multiple of π or $\ln(2)$. Let e_{min} and e_{max} be two rational integers such that $2^{e_{min}} \leq C/2 < 2^{e_{min}+1}$ and $e_{min} \leq e_{max}$. Let $p \in \mathbb{N}$ such that $2^{-p} \leq C/2$. The number ν_E of floating-point numbers x with n-bit mantissas and exponent E between e_{min} and e_{max} such that

$$|x \bmod^* C| < 2^{-p} \tag{9}$$

satisfies

- $2^{n-p-E}(N_E - 2) + 2 \leq \nu_E \leq 2^{n-p-E}N_E$ *if $n - E \geq p$. In that case, ν_E is easily computable by a program;*
- $\nu_E = 2^{n-p-E}(N_E + O({N_E}^{\delta+\varepsilon}))$ *if $n - E \geq p$, for every $\varepsilon > 0$, with $\delta \leq 19/29$ for C nonzero fractional multiple of $\ln(2)$, and $\delta \leq 301/351$ for C nonzero fractional multiple of π.*

where

$$\begin{aligned} N_E &= \left\lfloor \tfrac{1}{C}\left(\tfrac{1}{2^p} + 2^{E+1} - \tfrac{2^E}{2^{n-1}}\right)\right\rfloor \\ &- \left\lceil \tfrac{1}{C}\left(-\tfrac{1}{2^p} + 2^E\right)\right\rceil + 1. \end{aligned}$$

From this proposition, numerous experiments, and a well-known result by Khintchine [5], [6] that states that almost all real numbers are of type 1, we can assume that for any E, we have

$$\nu_E \approx \left\lfloor 2^{n-p-E}N_E \right\rfloor. \tag{10}$$

We have checked this result by computing all reduced arguments for some values of n, e_{min} and e_{max} such that this exhaustive computation remains possible in a reasonable delay. Some obtained results are given in Fig. 2, 3 and 4. These results show that the estimate provided by (10) is a good one. These estimates will be used at the end of Section II-C.

II. A New High-radix Reduction Method

In this section, we assume that we perform range-reduction for the trigonometric functions, with $C = \pi/2$. Extension to other values of C (such as a fractional multiple of π – still for the trigonometric functions – or a fractional multiple of $\ln(2)$ – for the exponential function) is straightforward.

As stated before, our general philosophy is that we must give results that are:

1) always correct, even for rare cases;
2) computed as quickly as possible for frequent cases.

A way to deal with these requirements is to build a fast algorithm for input arguments with a small exponent,

ϵ	actual number	expected number
2^{-4}	7485	7552
2^{-5}	3744	3744
2^{-6}	1872	1872
2^{-7}	936	936
2^{-8}	468	468
2^{-9}	235	234
2^{-10}	118	117
2^{-11}	60	57
2^{-12}	31	27
2^{-13}	16	12
2^{-14}	10	5
2^{-15}	5	0
2^{-16}	3	0
2^{-17}	2	0
2^{-18}	1	0

Fig. 2. *Actual number of reduced arguments of absolute value less than ϵ, and expected number using (10), for various values of ϵ, in the case $C = \ln(2)$, $n = 14$, $e_{min} = 2$ and $e_{max} = 6$. Notice that the estimation obtained from (10) is adequate.*

ϵ	actual number $e_{min} = e_{max} = 5$	expected number
2^{-4}	20992	20992
2^{-5}	10496	10496
2^{-6}	5248	5248
2^{-7}	2624	2624
2^{-8}	1312	1312
2^{-9}	656	656
2^{-10}	328	328
2^{-11}	164	164
2^{-12}	82	82
2^{-13}	41	41
2^{-14}	0	20
2^{-15}	0	10
2^{-16}	0	5
2^{-17}	0	2
2^{-18}	0	1
2^{-19}	0	0

Fig. 3. *Actual number of reduced arguments of absolute value less than ϵ, and expected number using (10), for various values of ϵ, in the case $C = \pi/4$, $n = 18$, with $e_{min} = e_{max} = 5$. The estimation given by (10) is adequate.*

and to use a slower yet still accurate algorithm for input argument with a large exponent.

A. *Medium-size arguments (in $[8, 2^{63} - 1]$)*

To do so, in the following we focus on input arguments with a "reasonably small" exponent. More precisely, we assume that the double-precision input argument x has absolute value less than $2^{63} - 1$. For larger arguments, we assume that Payne and Hanek's method will be used, or that $x \bmod^* C$ will be computed using multiple-precision arithmetic. For straightforward symmetry reasons, we can assume that x is positive. We also assume that x is larger than or equal to 8. We then proceed as follows:

1) We define $I(x)$ as x rounded to the nearest integer. x is split into its residual part $\rho(x) = x - I(x)$ and $I(x)$, which is split into eight 7-bit parts $I_i(x)$ for $0 \leq i \leq 7$ as follows:

$$\begin{cases} I_7(x) = I(2^{-56}x), \\ I_6(x) = I\left(2^{-48}\left(x - \left(2^{56}I_7(x)\right)\right)\right), \\ I_5(x) = I\left(2^{-40}\left(x - \left(2^{56}I_7(x) + 2^{48}I_6(x)\right)\right)\right), \\ I_4(x) = I\left(2^{-32}\left(x - \sum_{i=5}^{7} 2^{8i}I_i(x)\right)\right), \\ I_3(x) = I\left(2^{-24}\left(x - \sum_{i=4}^{7} 2^{8i}I_i(x)\right)\right), \\ I_2(x) = I\left(2^{-16}\left(x - \sum_{i=3}^{7} 2^{8i}I_i(x)\right)\right), \\ I_1(x) = I\left(2^{-8}\left(x - \sum_{i=2}^{7} 2^{8i}I_i(x)\right)\right), \\ I_0(x) = I\left(x - \sum_{i=1}^{7} 2^{8i}I_i(x)\right), \\ \rho(x) \ = x - \sum_{i=0}^{7} 2^{8i}I_i(x), \end{cases}$$

so that

$$x = 2^{56}I_7(x) + 2^{48}I_6(x) + \ldots + 2^{8}I_1(x) + I_0(x) + \rho(x).$$

Note that $\rho(x)$ is exactly representable in double-precision, and that for $x \geq 2^{52}$, we have $\rho(x) = 0$ and $I(x) = x$. Also, since $x \geq 8$, the last mantissa bit of $\rho(x)$ has a weight greater than or equal to 2^{-49}.

Important remark: One could get a very similar algorithm, certainly easier to understand, by replacing the values $I_k(x)$ by the values $J_k(x)$

ϵ	actual number $e_{min} = e_{max} = 7$	expected number
2^{-4}	20844	20992
2^{-5}	10421	10432
2^{-6}	5216	5216
2^{-7}	2608	2608
2^{-8}	1304	1304
2^{-9}	652	652
2^{-10}	326	326
2^{-11}	163	163
2^{-12}	80	81
2^{-13}	41	40
2^{-14}	20	20
2^{-15}	9	10
2^{-16}	5	5
2^{-17}	2	2
2^{-18}	1	1
2^{-19}	0	0

Fig. 4. *Actual number of reduced arguments of absolute value less than ϵ, and expected number using (10), for various values of ϵ, in the case $C = \pi/4$, $n = 18$, with $e_{min} = e_{max} = 7$. Again, the estimation given by (10) is adequate.*

defined as

$$\begin{cases} J_0(x) & \text{contains bits} & 0 \text{ to } 7 & \text{of} & I(x), \\ J_1(x) & \text{contains bits} & 8 \text{ to } 15 & \text{of} & I(x), \\ J_2(x) & \text{contains bits} & 16 \text{ to } 23 & \text{of} & I(x), \\ J_3(x) & \text{contains bits} & 24 \text{ to } 31 & \text{of} & I(x), \\ J_4(x) & \text{contains bits} & 32 \text{ to } 39 & \text{of} & I(x), \\ J_5(x) & \text{contains bits} & 40 \text{ to } 47 & \text{of} & I(x), \\ J_6(x) & \text{contains bits} & 48 \text{ to } 55 & \text{of} & I(x), \\ J_7(x) & \text{contains bits} & 56 \text{ to } 63 & \text{of} & I(x), \end{cases}$$

but that would lead to tables twice as large as the ones required by our algorithm. Indeed, the values I_0 up to I_7 are stored on 8 bits each, but the sign bit will not be used and thus only 7 bits are necessary to index the tables.

The general idea behind our algorithm is to compute first

$$\begin{aligned} S(x) = & \\ & (I_0(x)) \bmod^* \pi/2 + (2^8 I_1(x)) \bmod^* \pi/2 \\ & +(2^{16} I_2(x)) \bmod^* \pi/2 \\ & \vdots \\ & +(2^{56} I_7(x)) \bmod^* \pi/2 \\ & +\rho(x). \end{aligned}$$

It holds that $x - S(x)$ is a multiple of $\pi/2$ and $S(x)$ will be smaller than x, but in general $S(x)$ will not be the desired reduced argument: a second, simpler reduction step will be necessary. In practice, the various possible values of $|(2^{8i} I_i(x))| \bmod^* \pi/2$ are stored in tables as a sum of two or three floating-point numbers.

As mentioned above, our goal is to always provide correct results even for the worst case for which we lose 61 bits of accuracy. Then we need to store $(I_i(x) \bmod^* \pi/2)$ with at least

$$\begin{aligned} & 61 \text{ (leading zeros)} \\ & \quad +53 \text{ (non-zero significant bits)} \\ & \quad +g \text{ (extra guard bits)} \\ & \quad = 114 + g \text{ bits.} \end{aligned}$$

To reach that precision (with a value of g equal to 39, which will be deduced in the following), all the numbers $(|2^{8i} I_i(x)| \bmod^* \pi/2)$, which belong to $[-1, 1]$, are stored in tables as the sum of three double-precision numbers:

$$\begin{cases} T_{hi}(i,w) & \text{is the multiple of } 2^{-49} \text{ that is} \\ & \text{closest to } ((2^{8i} w) \bmod^* \pi/2) \\ T_{med}(i,w) & \text{is the multiple of } 2^{-99} \text{ that is} \\ & \text{closest to } ((2^{8i} w) \bmod^* \pi/2) \\ & \qquad -T_{hi}(i,w) \\ T_{lo}(i,w) & \text{is the double-precision number} \\ & \text{that is closest to} \\ & ((2^{8i} w) \bmod^* \pi/2) - T_{hi}(i,w) \\ & \qquad -T_{med}(i,w) \end{cases}$$

where w is a 7-bit nonnegative integer.

Note that $T_{hi}(i,w) = T_{med}(i,w) = T_{lo}(i,w) = 0$ for $w = 0$. The three tables T_{hi}, T_{med} and T_{lo} need 10 address bits. The total amount of memory required by these tables is $3 \cdot 2^{10} \cdot 8 = 24$ Kbytes. From the definitions, one can easily deduce $|T_{med}(i,w)| \leq 2^{-50}$ and $|T_{lo}(i,w)| \leq 2^{-100}$. $T_{hi}(i,w) + T_{med}(i,w) + T_{lo}(i,w)$ approximates $(2^{8i} w) \bmod^* \pi/2$ with 153 bits of precision, which corresponds to $g = 39$. Computing T_{hi}, T_{med} and T_{lo} for the 1024 different possible values of (i,w) allows to get slightly sharper bounds, given in Table 1.

TABLE I

MAXIMUM VALUES OF T_{hi}, T_{med} AND T_{lo}.

$\max_{i,w} \lvert T_{hi}(i,w) \rvert$	$\max_{i,w} \lvert T_{med}(i,w) \rvert$	$\max_{i,w} \lvert T_{lo}(i,w) \rvert$
$0.784696\cdots$	$0.997607\cdots \times 2^{-50}$	$0.998214\cdots \times 2^{-100}$

2) Define

$$S_{hi}(x) = \left(\sum_{i=0}^{7} \mathrm{sign}(I_i(x)) T_{hi}(i, |I_i(x)|) \right) + \rho(x).$$

Its absolute value is bounded by $2\pi + \frac{1}{2}$, which is less than 8. Since $S_{hi}(x)$ is a multiple of 2^{-49} and has absolute value less than 8, it is exactly representable in double-precision floating-point arithmetic (it is even representable with 52 bits only). Therefore, with a correctly rounded arithmetic (such as the one provided on any system that follows the IEEE-754 standard for floating-point arithmetic), it will be *exactly computed*, without any rounding error. Also, consider

$$\begin{cases} S_{med}(x) = \sum_{i=0}^{7} \text{sign}(I_i(x)) T_{med}(i, |I_i(x)|), \\ S_{lo}(x) \quad = \sum_{i=0}^{7} \text{sign}(I_i(x)) T_{lo}(i, |I_i(x)|). \end{cases}$$

The number $S_{med}(x)$ is a multiple of 2^{-99} and its absolute value is less than 2^{-47}. Hence, it is exactly representable, and exactly computed, in double-precision floating-point arithmetic. $|S_{lo}|$ is less than 2^{-97}, and if S_{lo} is computed with round-to-nearest arithmetic as a balanced binary tree of additions:

$$\begin{aligned} &\big[\big(\text{sign}(I_0(x))T_{lo}(0, |I_0(x)|) \\ &\qquad + \text{sign}(I_1(x))T_{lo}(1, |I_1(x)|)\big) \\ &+\big(\text{sign}(I_2(x))T_{lo}(2, |I_2(x)|) \\ &\qquad + \text{sign}(I_3(x))T_{lo}(3, |I_3(x)|)\big)\big] \\ &\quad + \big[\big(\text{sign}(I_4(x))T_{lo}(4, |I_4(x)|) \\ &\qquad\quad + \text{sign}(I_5(x))T_{lo}(5, |I_5(x)|)\big) \\ &\quad + \big(\text{sign}(I_6(x))T_{lo}(6, |I_6(x)|) \\ &\qquad\quad + \text{sign}(I_7(x))T_{lo}(7, |I_7(x)|)\big)\big] \end{aligned} \tag{11}$$

then the rounding error is less than 3×2^{-151}. For each of the values $T_{lo}(i, I_i(x))$, the fact that is it rounded to the nearest yields an accumulated error (for these eight values) less than 8×2^{-154}. Thus the absolute error on $S_{lo}(x)$ is less than or equal to $8 \times 2^{-154} + 3 \times 2^{-151} = 2^{-149}$.

Since $S_{hi}(x) + S_{med}(x)$ is exactly computed, the number $S(x) = S_{hi}(x) + S_{med}(x) + S_{lo}(x)$ is equal to x minus an integer multiple of $\pi/2$ plus an error bounded by 2^{-149}.

And yet, $S(x)$ may not be the final reduced argument, since its absolute value may be significantly larger than $\pi/4$. We therefore may have to add or subtract a multiple of $\pi/2$ from $S(x)$ to get the final result, and straightforward calculations show that this multiple can only be $k\pi/2$ with $k = 1, 2, 3$ or 4.

B. Small arguments (smaller than 8)

Define $C_{hi}(k)$, for $k = 1, 2, 3, 4$, as the multiple of 2^{-49} that is closest to $k\pi/2$. $C_{hi}(k)$ is exactly representable as a double-precision number. Define $C_{med}(k)$ as the multiple of 2^{-99} that is closest to $k\pi/2 - C_{hi}(k)$ and $C_{lo}(k)$ as the double-precision number that is closest to $k\pi/2 - C_{hi}(k) - C_{med}(k)$.

We now proceed as follows:

- If $|S_{hi}(x)| \leq \pi/4$ then we define

$$\begin{aligned} R_{hi}(x) &= S_{hi}(x), \\ R_{med}(x) &= S_{med}(x), \\ R_{lo}(x) &= S_{lo}(x). \end{aligned}$$

- Else, let k_x be such that $C_{hi}(k_x)$ is closest to $|S_{hi}(x)|$. We successively compute:
  - If $S_{hi}(x) > 0$

$$\begin{aligned} R_{hi}(x) &= S_{hi}(x) - C_{hi}(k_x), \\ R_{med}(x) &= S_{med}(x) - C_{med}(k_x), \\ R_{lo}(x) &= S_{lo}(x) - C_{lo}(k_x). \end{aligned}$$

  - Else,

$$\begin{aligned} R_{hi}(x) &= S_{hi}(x) + C_{hi}(k_x), \\ R_{med}(x) &= S_{med}(x) + C_{med}(k_x), \\ R_{lo}(x) &= S_{lo}(x) + C_{lo}(k_x). \end{aligned}$$

  Again, $R_{hi}(x)$ and $R_{med}(x)$ are exactly representable (hence, they are exactly computed) in double-precision arithmetic:
  - $R_{hi}(x)$ has an absolute value less than $\pi/4$ and is a multiple of 2^{-49};
  - $R_{med}(x)$ has an absolute value less than $2^{-47} + 2^{-50}$ and is a multiple of 2^{-99}.

  $|R_{lo}(x)|$ is less than $2^{-97} + 2^{-100}$, and it is computed with error less than or equal to $2^{-149} + 2^{-150} + 2^{-154} = 49 \times 2^{-154}$:
  - 2^{-149} is the error bound on S_{lo};
  - 2^{-154} bounds the error due to the floating-point representation of $C_{lo}(k_x)$;
  - 2^{-150} bounds the rounding error that occurs when computing $S_{lo}(x) \pm C_{lo}(k_x)$ in round-to-nearest mode.

Therefore, the number $R(x) = R_{hi}(x) + R_{med}(x) + R_{lo}(x)$ is equal to x minus an integer multiple of $\pi/2$ plus an error bounded by $49 \times 2^{-154} < 2^{-148}$.

This step is also used (alone, without the previous steps) to reduce small input arguments, less than 8. This allows our algorithm to perform range-reduction for both kind of arguments, small and medium size. The reduced argument is now stored as the sum of three double-precision numbers, $R_{hi}(x)$, $R_{med}(x)$, and $R_{lo}(x)$. We want to return the reduced argument as the sum of two double-precision numbers (one double-precision number may not suffice if we wish to compute trigonometric functions with very good accuracy). To do that, we will use the Fast2sum algorithm presented hereafter.

C. Final step

We will get the final result of the range-reduction as follows. Let p be an integer parameter, $1 \leq p \leq 44$, used to specify the required accuracy. This choice comes from the fact that we work in double precision arithmetic, and that in the most frequent cases, the final relative error will be bounded by 2^{-100+p}: to allow an accurate double precision function result even in the very worst case, we must have a relative error significantly less than 2^{-53}. The problem here is only to propagate the possible carry when summing the three components $R_{hi}(x)$, $R_{med}(x)$ and $R_{lo}(x)$. This is performed using floating-point addition and the following result.

Theorem 2 (Fast2sum algorithm): [7, page 221, Thm. C] *Let a and b be floating-point numbers, with $|a| \geq |b|$. Assume the used floating-point arithmetic provides correctly rounded results with rounding to the nearest. The following algorithm*

```
fast2sum(a,b):
    s := a + b
    z := s - a
    r := b - z
```

computes two floating-point numbers s and r that satisfy:

- *$r + s = a + b$ exactly;*
- *s is the floating-point number which is closest to $a + b$.*

We now consider the different possibles cases:

- If $|R_{hi}(x)| > 1/2^p$, then, since $|R_{med}(x)| < 2^{-47} + 2^{-50}$, the reduced argument will be close to $R_{hi}(x)$. In that case, we first compute
$$t_{med}(x) = R_{med}(x) + R_{lo}(x).$$
The error on $t_{med}(x)$ is bounded by the former error on $R_{lo(x)}$ plus the rounding error due to the addition. Assuming rounding to nearest, this last error is less than or equal to 2^{-100}. Hence, the error on $t_{med}(x)$ is less than or equal to $2^{-100} + 2^{-148}$. Then, we perform (without rounding error)
$$(y_{hi}, y_{lo}) = \text{fast2sum}(R_{hi}(x), t_{med}(x)).$$
After that, the two floating-point numbers (y_{hi}, y_{lo}) represent the reduced argument with an absolute error bounded by $2^{-100} + 2^{-148} \approx 2^{-100}$. Hence, the *relative error* on the reduced argument will be bounded by a value very close to 2^{-100+p}.
- If $R_{hi}(x) = 0$, then we perform
$$(y_{hi}, y_{lo}) = \text{fast2sum}(R_{med}(x), R_{lo}(x)).$$
After that, since the absolute value of the reduced argument is always larger than 0.71×2^{-61}, the two floating-point numbers (y_{hi}, y_{lo}) represent the reduced argument with a relative error smaller than
$$\frac{49 \times 2^{-154}}{0.71 \times 2^{-61}} < 2^{-86}.$$
- If $0 < |R_{hi}(x)| \leq 2^{-p}$, then, since the absolute value of the reduced argument is always larger than 0.71×2^{-61}, and since $|R_{lo}(x)| < 2^{-97} + 2^{-100}$, most of the information on the reduced argument is in $R_{hi}(x)$ and $R_{med}(x)$. We first perform
$$(y_{hi}, t_{med}) = \text{fast2sum}(R_{hi}(x), R_{med}(x)).$$
Let k be the integer satisfying
$$2^{-k} \leq |y_{hi}| < 2^{-k+1}.$$
We easily find
$$|t_{med}| \leq 2^{-k-53}.$$
After that, we compute
$$y_{lo} = t_{med} + R_{lo}(x).$$
The rounding error due to this addition is bounded by 2^{-k-107}. Hence, the two floating-point numbers (y_{hi}, y_{lo}) represent the reduced argument with an absolute error smaller than
$$49 \times 2^{-154} + \max\{2^{-k-107}, 2^{-150}\}.$$
Therefore, (y_{hi}, y_{lo}) represent the reduced argument with a *relative* error better than
$$49 \times 2^{-154+k} + \max\{2^{-107}, 2^{-150+k}\}.$$
which is less than $\leq 2^{-87}$ since the absolute value of the reduced argument is less than 0.71×2^{-61}, which implies $2^{-k} \leq 2^{-61}$.

A first solution is to try to make the various error bounds equal. This is done by choosing $p = 14$. By doing that, in the worst case, the bound on the relative error will be 2^{-86}, which is quite good. We should notice that in this case, assuming (10) with $C = \pi/2$, the probability that $|R_{hi}(x)|$ be less than 2^{-p} is around 7.8×10^{-5}.

A possibly better solution is to make the most frequent case (i.e., $|R_{hi}(x)| > 2^{-p}$) more accurate, and to assume that a more accurate yet slower algorithm is used in the other cases (an easy solution is to split the variables into 4 floating-point values, instead of 3 as we did here). This is done by using a somewhat smaller value of p. For instance, with $p = 10$ and $C = \pi/2$, still assuming (10), the probability that $|R_{hi}(x)| < 2^{-p}$ is around 1.25×10^{-3}. In the most frequent case ($|R_{hi}(x)| \geq 2^{-p}$), the error bound on the computed reduced argument will be 2^{-90}. Due to its low probability, the other case can be processed with an algorithm hundred times slower without significantly changing the average time of computation, cf. Amdahl's law.

D. The algorithm

We can now sketch the complete algorithm:

Algorithm Range-Reduction:

Input: A double-precision floating-point number $x > 0$ and an integer $p > 0$ specifying the required precision in bits.

Output: The reduced argument y given as the sum of two double-precision floating-point numbers y_{hi} and y_{lo}, such that[2] $-\pi/4 \leq y < \pi/4$ and $y = x - k\frac{\pi}{2}$ within an error given in the analysis of Section II-C, for some integer k.

Method:

```
if $x \geq 2^{63} - 1$ then
    {Apply the method of Payne and Hanek.}
else if $x \leq 8$ then
        $S_{hi} \leftarrow x$; $S_{med} \leftarrow 0$; $S_{lo} \leftarrow 0$;
else
    $I \leftarrow \mathrm{round}(x)$; $\rho \leftarrow x - I$;
    $S_{hi} \leftarrow \rho$; $S_{med} \leftarrow 0$; $S_{lo} \leftarrow 0$;
    $i \leftarrow 7$;
    $j \leftarrow 56$;
    while $i \geq 0$ do
        $w \leftarrow \mathrm{round}(I >> j)$;
        $S_{hi} \leftarrow S_{hi} + \mathrm{sign}(w)T_{hi}(i, |w|)$;
        $S_{med} \leftarrow S_{med} + \mathrm{sign}(w)T_{med}(i, |w|)$;
        $I \leftarrow I - (w << j)$; $i \leftarrow i - 1$; $j \leftarrow j - 8$
    $S_{lo} \leftarrow \sum_{i=0}^{7} \mathrm{sign}(w)T_{lo}(i, |w|)$ (cf. 11);
if $|S_{hi}| \geq \pi/4$ then
    $k \leftarrow \mathrm{Reduce}(|S_{hi}|)$
    $S_{hi} \leftarrow S_{hi} + \mathrm{sign}(S_{hi})C_{hi}(k)$;
    $S_{med} \leftarrow S_{med} + \mathrm{sign}(S_{hi})C_{med}(k)$;
    $S_{lo} \leftarrow S_{lo} + \mathrm{sign}(S_{hi})C_{lo}(k)$;
if $|S_{hi}| > 2^{-p}$ then
    $temp \leftarrow S_{med} + S_{lo}$;
    $(y_{hi}, y_{lo}) \leftarrow \mathrm{fast2sum}(S_{hi}, temp)$;
else if $S_{hi} = 0$ then
    $(y_{hi}, y_{lo}) \leftarrow \mathrm{fast2sum}(S_{med}, S_{lo})$;
else
    $(y_{hi}, temp) \leftarrow \mathrm{fast2sum}(S_{hi}, S_{med})$;
    $y_{lo} \leftarrow temp + S_{lo}$.
```

Where: The function $\mathrm{Reduce}(|S_{hi}|)$ chooses the appropriate multiple k of $\pi/2$, represented as the triple $(C_{hi}(k), C_{med}(k), C_{lo}(k))$.

III. Cost of the algorithm

In this section we compare our method to other algorithms on the same input range $[8, 2^{63}-1]$: Payne and Hanek's methods (see Section (I-A)) and the Modular range-reduction method described in [1]. Concerning Payne and Hanek's method we used the version of the algorithm used by Sun Microsystems [10]. We chose as criteria for the evaluation of the algorithms the table size, the number of table accesses and the number of floating-point multiplications, divisions and additions.

TABLE II
Comparison of our algorithm with Payne and Hanek's algorithm and the modular range-reduction algorithm.

	# Elementary operations	# Table accesses	Table size in Kbytes
Our algorithm	3/33	3/27	24(20)
Payne & Hanek	55/103	1	0.14
Modular range-reduction	150	53	2

Table II shows the potential advantages of our algorithm for small and medium-sized input argument. Payne and Hanek's method over that range doesn't need much memory, but roughly requires three times as many operations. The Modular range-reduction has the same characteristics as Payne and Hanek's method concerning the table size needed and the number of elementary operations involved, but requires more table accesses. Our algorithm is then a good compromise between table size and number of operations for range-reduction of medium size argument.

To get more accurate figures than by just counting the operations, we have implemented this algorithm in ANSI-C. The program can be downloaded from http://gala.univ-perp.fr/˜ddefour/high_radix.tgz. This implementation shows that our algorithm is 4 to 5 times faster, depending on the required final precision, than the Sun implementation of Payne and Hanek's algorithm, provided that the tables are in main memory (which will be true when the trigonometric functions are frequently called in a numerical program. And when they are not frequently called, the speed of range-reduction is no longer an issue). Our algorithm is then a good compromise between table size and delay for range-reduction of small and medium-sized arguments.

A variant of our algorithm would consist in first computing S_{hi}, S_{med} and R_{hi}, R_{med} only. Then, during the fourth step of the algorithm, if the accuracy does not suffice, compute T_{lo} and R_{lo}. This slight modification can reduce the number of elementary operations in the (most frequent) cases where no extra accuracy is needed. We can also reduce the table size by 4 Kbytes by storing the T_{lo} values in single-precision only, instead of using

[2] In fact, the absolute value of the reduced argument is less than $\pi/4$ plus the largest possible value of $|S_{med} + S_{lo}|$, hence, less than $\pi/4+2^{-47}+2^{-97}$. In practice, this has no influence on the elementary function algorithms.

double-precision.

Another variant (that can be useful depending on the processor and compiler), would be to replace the loop "while $i \geq 0$" with "while $I <> 0$ and $i \geq 0$". In that case (for a medium-sized argument x), the number N of double-precision floating-point operations becomes $N = 17 + 2\lceil \log_{256} x \rceil$, i.e., $13 \leq N \leq 34$. Also, the number of table accesses becomes $4 + 3\lceil \log_{256} x \rceil$.

IV. Conclusions

We have presented an algorithm for accurate range-reduction of input arguments with absolute value less than $2^{63} - 1$. This table-based algorithm gives accurate results for the most frequent cases. In order to cover the whole double-precision domain for input arguments, we suggest using Payne and Hanek's algorithm for huge arguments. A major drawback of our method lies in the table size needed, thus a future effort will be to reduce the table size, while keeping a good tradeoff between speed and accuracy.

References

[1] M. Daumas, C. Mazenc, X. Merrheim, and J. M. Muller. "Modular range-reduction: A new algorithm for fast and accurate computation of the elementary functions," *Journal of Universal Computer Science*, 1(3):162–175, March 1995.

[2] M. Hata, "Legendre type polynomials and irrationality measures," *J. reine angew. Math.* **407** (1990), 99–125.

[3] M. Hata, "Rational approximations to π and some other numbers," *Acta Arith.* **63** (1993), n°4, 335–349.

[4] W. Kahan. *Minimizing q*m-n*, text accessible electronically at http://http.cs.berkeley.edu/∼wkahan/. At the beginning of the file "nearpi.c", 1983.

[5] A. Ya. Khintchine, "Einige Sätze über Kettenbrüche, mit Anwendungen auf die Theorie der diophantischen Approximationen", *Math. Ann.* **92** (1924), 115–125.

[6] A. Ya. Khintchine, *Continued Fractions*. The University of Chicago Press, Chicago Ill., London, 1964.

[7] D. Knuth. *The Art of Computer Programming,* volume 2. Addison Wesley, Reading, MA, 1973.

[8] L. Kuipers and H. Niederreiter, *Uniform distribution of sequences*, Pure and Applied Mathematics. Wiley-Interscience [John Wiley & Sons], New York-London-Sydney, 1974.

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

[10] K. C. Ng. "Argument reduction for huge arguments: Good to the last bit." Technical report, SunPro, 1992. `http://www.validlab.com/arg.pdf`

[11] M. Payne and R. Hanek. "Radian reduction for trigonometric functions," *SIGNUM Newsletter,* 18:19–24, 1983.

[12] R. A. Smith. "A continued-fraction analysis of trigonometric argument reduction," *IEEE Transactions on Computers*, 44(11):1348–1351, November 1995.