An Implementation of Sin and Cos Using Gal's Accurate Tables

Pascal Leroy (phl)

2025-09-26

This document describes the implementation of functions `Sin` and `Cos` in Principia. The goals of that implementation are to be portable (including to machines that do not have a fused multiply-add instruction), achieve good performance, and ensure correct rounding.

Overview

The implementation follows the ideas described by [GB91] and uses accurate tables produced by the method presented in [SZ05]. It guarantees correct rounding with a high probability. In circumstances where it cannot guarantee correct rounding, it falls back to the (slower but correct) implementation provided by the CORE-MATH project [SZG22] [ZSG+24]. More precisely, the algorithm proceeds through the following steps:

— perform argument reduction using Cody and Waite's algorithm in double precision (see [Mul+10, p. 379]);
— if argument reduction loses too many bits (*i.e.*, the argument is close to a multiple of $\frac{\pi}{2}$), fall back to `cr_sin` or `cr_cos`;
— otherwise, uses accurate tables and polynomial approximations to compute `Sin` or `Cos` with extra accuracy;
— if the result fails Muller's rounding test ([Mul+10, pp. 397–400]), fall back to `cr_sin` or `cr_cos`;
— otherwise return the rounded result of the preceding computation.

Notation and Accuracy Model

In this document we assume a base-2 floating-point number system with M significand bits[1] similar to the IEEE formats. We define a real function $\mathfrak{m}$ and an integer function $\mathfrak{e}$ denoting the *significand* and *exponent* of a real number, respectively:

$$x = \pm\,\mathfrak{m}(x) \times 2^{\mathfrak{e}(x)} \qquad \text{with} \qquad 2^{M-1} \le \mathfrak{m}(x) \le 2^M - 1$$

Note that this representation is unique. Furthermore, if x is a floating-point number, $\mathfrak{m}(x)$ is an integer.

The *unit of the last place*[2] of x is defined as:

$$\mathfrak{u}(x) \coloneqq 2^{\mathfrak{e}(x)}$$

In particular, $\mathfrak{u}(1) = 2^{1-M}$ and:

$$\frac{|x|}{2^M} < \frac{|x|}{2^M - 1} \le \mathfrak{u}(x) \le \frac{|x|}{2^{M-1}} \tag{1}$$

We ignore the exponent bias, overflow and underflow as they play no role in this discussion.

[1] In `binary64`, $M = 53$.
[2] Intuitively, this is the ULP "above" for powers of 2.

Finally, for error analysis we use the accuracy model of [Higo2], equation (2.4): everywhere they appear, the quantities δ_i represent a roundoff factor such that $|\delta_i| < u = \mathfrak{u}\left(\frac{1}{2}\right) = 2^{-M}$ (see pages 37 and 38). We also use θ_n and γ_n with the same meaning as in [Higo2], lemma 3.1.

Approximation of $\frac{\pi}{2}$

To perform argument reduction, we need to build approximations of $\frac{\pi}{2}$ with extra accuracy and analyse the circumstances under which they may be used and the errors that they entail on the reduced argument.

Let $z \geq 0$. We start by defining the truncation function $\mathrm{Tr}(\kappa, z)$ which clears the last κ bits of the significand of z:

$$\mathrm{Tr}(\kappa, z) \coloneqq \lfloor 2^{-\kappa}\, \mathfrak{m}(z) \rfloor\, 2^{\kappa}\, \mathfrak{u}(z)$$

We have:

$$z - \mathrm{Tr}(\kappa, z) = (2^{-\kappa}\, \mathfrak{m}(z) - \lfloor 2^{-\kappa}\, \mathfrak{m}(z) \rfloor)\, 2^{\kappa}\, \mathfrak{u}(z)$$

The definition of the floor function implies that the quantity in parentheses is in $[0, 1[$ and therefore:

$$0 \leq z - \mathrm{Tr}(\kappa, z) < 2^{\kappa}\, \mathfrak{u}(z)$$

Furthermore if the bits that are being truncated start with exactly k zeros we have the stricter inequality:

$$2^{\kappa'-1}\, \mathfrak{u}(z) \leq z - \mathrm{Tr}(\kappa, z) < 2^{\kappa'}\, \mathfrak{u}(z) \quad \text{with} \quad \kappa' = \kappa - k \tag{2}$$

This leads to the following upper bound for the unit of the last place of the truncation error:

$$\mathfrak{u}(z - \mathrm{Tr}(\kappa, z)) < 2^{\kappa'-M+1}\, \mathfrak{u}(z)$$

which can be made more precise by noting that the function $\mathfrak{u}$ is always a power of 2:

$$\mathfrak{u}(z - \mathrm{Tr}(\kappa, z)) = 2^{\kappa'-M}\, \mathfrak{u}(z) \tag{3}$$

Two-Term Approximation

In this scheme we approximate $\frac{\pi}{2}$ as the sum of two floating-point numbers:

$$\frac{\pi}{2} \simeq C_1 + \delta C_1$$

which are defined as:

$$\begin{cases} C_1 & \coloneqq \mathrm{Tr}\left(\kappa_1, \frac{\pi}{2}\right) \\ \delta C_1 & \coloneqq \left[\!\left[\frac{\pi}{2} - C_1 \right]\!\right] \end{cases}$$

Equation (2) applied to the definition of C_1 yields:

$$2^{\kappa_1'-1}\, \mathfrak{u}\left(\frac{\pi}{2}\right) \leq \frac{\pi}{2} - C_1 < 2^{\kappa_1'}\, \mathfrak{u}\left(\frac{\pi}{2}\right)$$

where $\kappa_1' \leq \kappa_1$ accounts for any leading zeroes in the bits of $\frac{\pi}{2}$ that are being truncated. Accordingly equation (3) yields, for the unit of the last place:

$$\mathfrak{u}\left(\frac{\pi}{2} - C_1\right) = 2^{\kappa_1'-M}\, \mathfrak{u}\left(\frac{\pi}{2}\right)$$

Noting that the absolute error on the rounding that appears in the definition of δC_1 is bounded by $\frac{1}{2}\,\mathfrak{u}\left(\frac{\pi}{2} - C_1\right)$, we obtain the absolute error on the two-term approximation[3]:

$$\left|\frac{\pi}{2} - C_1 - \delta C_1\right| \leq \frac{1}{2}\,\mathfrak{u}\left(\frac{\pi}{2} - C_1\right) = 2^{\kappa_1'-M-1}\,\mathfrak{u}\left(\frac{\pi}{2}\right) \tag{4}$$

[3]With the chosen value $\kappa_1 = 8$, the two sides of this inequality turn out to be $2^{-103.217}$ and 2^{-101}. The difference comes from the fact that $\left|\frac{\pi}{2} - C_1 - \delta C_1\right|$ has three zeroes after the last bit of its mantissa, but the theoretical computation above assumes the worst, *i.e.*, a run of ones.

From this we derive the following upper bound for δC_1:

$$\begin{aligned}|\delta C_1| &< \frac{\pi}{2} - C_1 + \frac{1}{2}\,\mathrm{u}\left(\frac{\pi}{2} - C_1\right) \\ &< 2^{\kappa_1'}\,\mathrm{u}\left(\frac{\pi}{2}\right) + 2^{\kappa_1'-M-1}\,\mathrm{u}\left(\frac{\pi}{2}\right) = 2^{\kappa_1'}(1 + 2^{-M-1})\,\mathrm{u}\left(\frac{\pi}{2}\right)\end{aligned} \tag{5}$$

This scheme gives a representation with a significand that has effectively $2M - \kappa_1'$ bits and is such that multiplying C_1 by an integer less than or equal to 2^{κ_1} is exact.

Three-Term Approximation

In this scheme we approximate $\frac{\pi}{2}$ as the sum of three floating-point numbers:

$$\frac{\pi}{2} \simeq C_2 + C_2' + \delta C_2$$

which are defined as:

$$\begin{cases} C_2 & \coloneqq \mathrm{Tr}\left(\kappa_2, \frac{\pi}{2}\right) \\ C_2' & \coloneqq \mathrm{Tr}\left(\kappa_2, \frac{\pi}{2} - C_2\right) \\ \delta C_2 & \coloneqq \left[\!\left[\frac{\pi}{2} - C_2 - C_2'\right]\!\right] \end{cases}$$

Equation (2) applied to the definition of C_2 yields:

$$2^{\kappa_2'-1}\,\mathrm{u}\left(\frac{\pi}{2}\right) \le \frac{\pi}{2} - C_2 < 2^{\kappa_2'}\,\mathrm{u}\left(\frac{\pi}{2}\right) \tag{6}$$

where $\kappa_2' \le \kappa_2$ accounts for any leading zeroes in the bits of $\frac{\pi}{2}$ that are being truncated. Accordingly equation (3) yields, for the unit of the last place:

$$\mathrm{u}\left(\frac{\pi}{2} - C_2\right) = 2^{\kappa_2'-M}\,\mathrm{u}\left(\frac{\pi}{2}\right)$$

Similarly, equation (2) applied to the definition of C_2' yields:

$$\begin{aligned} 2^{\kappa_2''-1}\,\mathrm{u}\left(\frac{\pi}{2} - C_2\right) &\le \frac{\pi}{2} - C_2 - C_2' < 2^{\kappa_2''}\,\mathrm{u}\left(\frac{\pi}{2} - C_2\right) \\ 2^{\kappa_2'+\kappa_2''-M-1}\,\mathrm{u}\left(\frac{\pi}{2}\right) &\le \qquad\qquad\qquad < 2^{\kappa_2'+\kappa_2''-M}\,\mathrm{u}\left(\frac{\pi}{2}\right) \end{aligned}$$

where $\kappa_2'' \le \kappa_2$ accounts for any leading zeroes in the bits of $\frac{\pi}{2} - C_2$ that are being truncated. Note that normalization of the significand of $\frac{\pi}{2} - C_2$ effectively drops the zeroes at positions κ_2 to κ_2' and therefore the computation of C_2' applies to a significand aligned on position κ_2'.

It is straightforward to transform these inequalities using (6) to obtain bounds on C_2':

$$2^{\kappa_2'}\left(\frac{1}{2} - 2^{\kappa_2''-M}\right)\mathrm{u}\left(\frac{\pi}{2}\right) < C_2' < 2^{\kappa_2'}(1 - 2^{\kappa_2''-M-1})\,\mathrm{u}\left(\frac{\pi}{2}\right)$$

Equation (3) applied to the definition of C_2' yields, for the unit of the last place:

$$\begin{aligned}\mathrm{u}\left(\frac{\pi}{2} - C_2 - C_2'\right) &= 2^{\kappa_2''-M}\,\mathrm{u}\left(\frac{\pi}{2} - C_2\right) \\ &= 2^{\kappa_2'+\kappa_2''-2M}\,\mathrm{u}\left(\frac{\pi}{2}\right)\end{aligned}$$

Noting that the absolute error on the rounding that appears in the definition of δC_2 is bounded by $\frac{1}{2}\,\mathrm{u}\left(\frac{\pi}{2} - C_2 - C_2'\right)$, we obtain the absolute error on the three-term approximation:

$$\left|\frac{\pi}{2} - C_2 - C_2' - \delta C_2\right| \le \frac{1}{2}\,\mathrm{u}\left(\frac{\pi}{2} - C_2 - C_2'\right) = 2^{\kappa_2'+\kappa_2''-2M-1}\,\mathrm{u}\left(\frac{\pi}{2}\right) \tag{7}$$

and the following upper bound for δC_2:

$$|\delta C_2| < 2^{\kappa_2'+\kappa_2''-M}(1 + 2^{-M-1})\,\mathrm{u}\left(\frac{\pi}{2}\right) \tag{8}$$

This scheme gives a representation with a significand that has effectively $3M - \kappa_2' - \kappa_2''$ bits and is such that multiplying C_2 and C_2' by an integer less than or equal to 2^{κ_2} is exact.

Argument Reduction

Given an argument x, the purpose of argument reduction is to compute a pair of floating-point numbers $(\tilde{x}, \delta\tilde{x})$ such that:

$$\begin{cases} \tilde{x} + \delta\tilde{x} \cong x \pmod{\frac{\pi}{2}} \\ \tilde{x} \text{ is approximately in } \left[-\frac{\pi}{4}, \frac{\pi}{4}\right] \\ |\delta\tilde{x}| \leq \frac{1}{2}\,\mathrm{u}(\tilde{x}) \end{cases}$$

Argument Reduction for Small Angles

If $|x| < \left[\!\left[\frac{\pi}{4}\right]\!\right]$ then $\tilde{x} = x$ and $\delta\tilde{x} = 0$.

Argument Reduction Using the Two-Term Approximation

If $|x| \leq 2^{\kappa_1}\left[\!\left[\frac{\pi}{2}\right]\!\right]$ we compute:

$$\begin{cases} n & = \left\lfloor \left[\!\left[x \left[\!\left[\frac{2}{\pi} \right]\!\right] \right]\!\right] \right\rceil \\ y & = x - n\ C_1 \\ \delta y & = [\![n\ \delta C_1]\!] \\ (\tilde{x}, \delta\tilde{x}) & = \mathrm{TwoDifference}\big(y, \delta y\big) \end{cases}$$

The first thing to note is that $|n| \leq 2^{\kappa_1}$. We have:

$$|x| \leq 2^{\kappa_1}\left[\!\left[\frac{\pi}{2}\right]\!\right] = 2^{\kappa_1}\frac{\pi}{2}(1+\delta_1)$$

and:

$$\left[\!\left[x \left[\!\left[\frac{2}{\pi} \right]\!\right] \right]\!\right] = x\frac{2}{\pi}(1+\delta_2)(1+\delta_3) \tag{9}$$

from which we deduce the upper bound:

$$\begin{aligned} |n| &\leq \left\lceil 2^{\kappa_1}\frac{\pi}{2}(1+\delta_1)\frac{2}{\pi}(1+\delta_2)(1+\delta_3) \right\rceil \\ &\leq \left\lceil 2^{\kappa_1}(1+\gamma_3) \right\rceil \end{aligned}$$

If $2^{\kappa_1}\gamma_3$ is small enough (less than $1/2$), the rounding cannot cause n to exceed 2^{κ_1}. In practice we choose a relatively small value for κ_1, so this condition is met.

Now if x is close to an odd multiple of $\frac{\pi}{4}$ it is possible for misrounding to happen. There are two kinds of misrounding, with different bounds.

A misrounding of the first kind occurs if, assuming $n > 0$:

$$x < \left(n - \frac{1}{2}\right)\frac{\pi}{2} \quad \text{and} \quad \left[\!\left[x \left[\!\left[\frac{2}{\pi} \right]\!\right] \right]\!\right] > n - \frac{1}{2}$$

Using equation (9) we find that this misrounding is only possible if:

$$x > \frac{\pi}{2}\left(n - \frac{1}{2}\right)\frac{1}{(1+\delta_2)(1+\delta_3)} \geq \frac{\pi}{2}\left(n - \frac{1}{2}\right)\frac{1}{1+\gamma_2}$$

In which case the computation of n results in:

$$n\frac{\pi}{2} - x < \frac{\pi}{4}\left(1 + \frac{\gamma_2}{1+\gamma_2}(2n-1)\right)$$

In this case, misrounding causes the absolute value of the reduced angle to increase and it may thus exceed $\frac{\pi}{4}$ by as much as:

$$\frac{\pi}{4}\frac{\gamma_2}{1+\gamma_2}(2^{\kappa_1+1} - 1) \tag{10}$$

A misrounding of the second kind occurs if, assuming $n \geq 0$:

$$x > \left(n + \frac{1}{2}\right)\frac{\pi}{2} \quad \text{and} \quad \left[\!\left[x\left[\!\left[\frac{2}{\pi}\right]\!\right]\right]\!\right] < n + \frac{1}{2}$$

A derivation similar to the one above gives the following condition for this misrounding to be possible. Using equation (9):

$$x < \frac{\pi}{2}\left(n + \frac{1}{2}\right)\frac{1}{(1+\delta_2)(1+\delta_3)} \leq \frac{\pi}{2}\left(n + \frac{1}{2}\right)(1+\gamma_2)$$

we derive the bound:

$$x - n\frac{\pi}{2} < \frac{\pi}{4}(1 + \gamma_2(2n+1))$$

In this case, misrounding causes the absolute value of the reduced angle to decrease by as much as:

$$\frac{\pi}{4}\gamma_2(2^{\kappa_1+1}+1)$$

This is however not a concern for the accurate tables as it cannot cause the reduced angle to become negative.

Using the bound on $|n|$ and the fact that C_1 has κ_1 trailing zeroes, we see that the product $n\ C_1$ is exact. The subtraction $x - n\ C_1$ is exact by Sterbenz's Lemma. Finally, the last step performs an exact addition[4] using algorithm 4 of [HLB08].

To compute the overall error on argument reduction[5], first remember that, from equation (4), we have:

$$C_1 + \delta C_1 = \frac{\pi}{2} + \zeta \quad \text{with} \quad |\zeta| \leq 2^{\kappa_1'-M-1}\,\mathrm{u}\left(\frac{\pi}{2}\right)$$

The error computation proceeds as follows:

$$\begin{aligned} y - \delta y &= x - n\ C_1 - n\ \delta C_1(1+\delta_4) \\ &= x - n(C_1 + \delta C_1) - n\ \delta C_1\ \delta_4 \\ &= x - n\frac{\pi}{2} - n(\zeta + \delta C_1\ \delta_4) \end{aligned}$$

from which we deduce an upper bound on the absolute error of the reduction:

$$\begin{aligned} \left|y - \delta y - \left(x - n\frac{\pi}{2}\right)\right| &\leq 2^{\kappa_1}2^{\kappa_1'}(2^{-M-1} + 2^{-M} + 2^{-2M-1})\,\mathrm{u}\left(\frac{\pi}{2}\right) \\ &= 2^{\kappa_1+\kappa_1'-M}\left(\frac{3}{2} + 2^{-M-1}\right)\mathrm{u}\left(\frac{\pi}{2}\right) \\ &< 2^{\kappa_1+\kappa_1'-M+1}\,\mathrm{u}\left(\frac{\pi}{2}\right) \end{aligned}$$

where we have used the upper bound for δC_1 given by equation (5).

The exact TwoDifference yields a pair such that $|\delta\tilde{x}| \leq \frac{\mathrm{u}(\tilde{x})}{2} \leq 2^{-M}|\tilde{x}|$. Furthermore, misrounding of the first kind and the above error on the reduction may combine to cause $|\tilde{x}|$ to move above $\frac{\pi}{4}$ by as much as:

$$\frac{\pi}{4}\frac{\gamma_2}{1+\gamma_2}(2^{\kappa_1+1}-1) + 2^{\kappa_1+\kappa_1'-M+1}\,\mathrm{u}\left(\frac{\pi}{2}\right)$$

[4]The more efficient QuickTwoDifference is not usable here. First, note that $|y|$ is equal to $\mathrm{u}(x)$ if we take x to be the successor or the predecessor of nC_1 for any n. Ignoring rounding errors we have:

$$|\delta y| \geq n\ 2^{\kappa_1'-1}\,\mathrm{u}\left(\frac{\pi}{2}\right) \geq 2^{\kappa_1'+M-2}\,\mathrm{u}\left(\frac{\pi}{2}\right)\mathrm{u}(n)$$

where we used the bound given by equation (1). Now the computation of n can result in a value that is either in the same binade or in the binade below that of x. Therefore $\mathrm{u}(n) \geq \frac{1}{2}\,\mathrm{u}(x)$ and the above inequality becomes:

$$|\delta y| \geq 2^{\kappa_1'+M-3}\,\mathrm{u}\left(\frac{\pi}{2}\right)\mathrm{u}(x)$$

plugging $\mathrm{u}\left(\frac{\pi}{2}\right) = 2^{1-M}$ we find:

$$|\delta y| \geq 2^{\kappa_1'-2}\,\mathrm{u}(x)$$

Therefore, as long as $\kappa_1' > 2$, there exist arguments x for which $|\delta y| > |y|$.

[5]Note that this error analysis is correct even in the face of misrounding.

The accurate tables must be constructed so that the last interval covers angles misrounded in that manner[6].

In the computation of the trigonometric functions, we need $\tilde{x} + \delta\tilde{x}$ to provide enough accuracy that the final result is correctly rounded most of the time. The above error bound shows that, if $\tilde{x}$ is very small (*i.e.*, if x is very close to a multiple of $\frac{\pi}{2}$), the two-term approximation may not provide enough correct bits. Formally, say that we want to have $M + \kappa_3$ correct bits in the mantissa of $\tilde{x} + \delta\tilde{x}$. The error must be less than $2^{-\kappa_3}$ half-units of the last place of the result:

$$2^{\kappa_1+\kappa_1'-M+1}\,\mathrm{u}\left(\frac{\pi}{2}\right) \leq 2^{-\kappa_3-1}\,\mathrm{u}(\tilde{x}) \leq 2^{-\kappa_3-M}|\tilde{x}| \tag{11}$$

which leads to the following condition on the reduced angle:

$$|\tilde{x}| \geq 2^{\kappa_1+\kappa_1'+\kappa_3+1}\,\mathrm{u}\left(\frac{\pi}{2}\right) = 2^{\kappa_1+\kappa_1'+\kappa_3-M+2}$$

The rest of the implementation assumes that $\kappa_3 = 18$ to achieve correct rounding with high probability. If we choose $\kappa_1 = 8$ we find that $\kappa_1' = 5$ (because there are three consecutive zeroes at this location in the significand of $\frac{\pi}{2}$) and the desired accuracy is obtained as long as $|\tilde{x}| \geq 2^{-20} \simeq 9.5 \times 10^{-7}$.

Argument Reduction Using the Three-Term Approximation

If $|x| \leq 2^{\kappa_2}\left[\!\left[\frac{\pi}{2}\right]\!\right]$ we compute:

$$\begin{cases} n & = \left\lceil\left\lceil x\left[\!\left[\frac{2}{\pi}\right]\!\right]\right\rceil\right\rfloor \\ y & = x - n\ C_2 \\ y' & = n\ C_2' \\ \delta y & = [\![n\ \delta C_2]\!] \\ (z, \delta z) & = \mathrm{QuickTwoSum}\big(y', \delta y\big) \\ (\tilde{x}, \delta\tilde{x}) & = \mathrm{LongSub}\big(y, (z, \delta z)\big) \end{cases}$$

The products $n\ C_2$ and $n\ C_2'$ are exact thanks to the κ_2 trailing zeroes of C_2 and C_2'. The subtraction $x - n\ C_2$ is exact by Sterbenz's Lemma. QuickTwoSum performs an exact addition using algorithm 3 of [HLB08]; it is usable in this case because clearly $|\delta y| < |y'|$. LongSub is the obvious adaptation of the algorithm LongAdd presented in section 5 of [Lin81], which implements precise (but not exact) double-precision arithmetic.

It is straightforward to show, like we did in the preceding section, that:

$$|n| \leq \left\lceil 2^{\kappa_2}(1 + \gamma_3)\right\rceil$$

and therefore that $|n| \leq 2^{\kappa_2}$ as long as $2^{\kappa_2}\gamma_3 < 1/2$. Similarly, the misrounding bound (10) is applicable with κ_2 replacing κ_1.

To compute the overall error on argument reduction, first remember that, from equation (7), we have:

$$C_2 + C_2' + \delta C_2 = \frac{\pi}{2} + \zeta_1 \quad \text{with} \quad |\zeta_1| \leq 2^{\kappa_2'+\kappa_2''-2M-1}\,\mathrm{u}\left(\frac{\pi}{2}\right)$$

Let ζ_2 be the relative error introduced by LongAdd. Table 1 of [Lin81] indicates that $|\zeta_2| < 2^{2-2M}$. The error computation proceeds as follows:

$$\begin{aligned} y - y' - \delta y &= (x - n\ C_2 - n\ C_2' - n\ \delta C_2(1 + \delta_4))(1 + \zeta_2) \\ &= \left(x - n\frac{\pi}{2} - n(\zeta_1 + \delta C_2\ \delta_4)\right)(1 + \zeta_2) \\ &= x - n\frac{\pi}{2} - n(\zeta_1 + \delta C_2\ \delta_4)(1 + \zeta_2) + \left(x - n\frac{\pi}{2}\right)\zeta_2 \end{aligned}$$

[6] In practice this is not a stringent constraint because the distance between accurate table entries is much larger than this quantity.

from which we deduce an upper bound on the absolute error of the reduction, noting that $\left|x - n\frac{\pi}{2}\right| \le \frac{\pi}{4}(1 + \gamma_2(2^{\kappa_2+1} + 1))$ as per (10):

$$\begin{aligned}
&\left|y - y' - \delta y - \left(x - n\frac{\pi}{2}\right)\right| \\
&\quad \le 2^{\kappa_2+\kappa_2'+\kappa_2''}(2^{-2M-1} + 2^{-2M} + 2^{-3M-1})(1 + 2^{2-2M})\,\mathrm{u}\left(\frac{\pi}{2}\right) + 2^{2-2M}\frac{\pi}{4}(1 + \gamma_2(2^{\kappa_2+1} + 1)) \\
&\quad < 2^{\kappa_2+\kappa_2'+\kappa_2''-2M}\left(\frac{3}{2} + 2^{-M-1}\right)(1 + 2^{2-2M})\,\mathrm{u}\left(\frac{\pi}{2}\right) + 2^{-2M}\,\pi\left(1 + 3 \times 2^{\kappa_2}\,\mathrm{u}\left(\frac{\pi}{2}\right)\right) \\
&\quad < 2^{\kappa_2-2M}(2^{\kappa_2'+\kappa_2''+1} + 3)\,\mathrm{u}\left(\frac{\pi}{2}\right) + 2^{-2M}\,\pi
\end{aligned}$$

where the second inequality uses $\gamma_2(2^{\kappa_2+1} + 1) < 3u\ 2^{\kappa_2+1}$.

A sufficient condition for the reduction to guarantee κ_3 extra bits of accuracy is for this error to be less than $2^{-\kappa_3-1}\,\mathrm{u}(\tilde{x})$ which itself is less than $2^{-\kappa_3-M}|\tilde{x}|$. Therefore we want:

$$\begin{aligned}
|\tilde{x}| &\ge 2^{\kappa_3-M}\left(2^{\kappa_2}(2^{\kappa_2'+\kappa_2''+1} + 3)\,\mathrm{u}\left(\frac{\pi}{2}\right) + \pi\right) \\
&= 2^{\kappa_3-M}(2^{\kappa_2-M+1}(2^{\kappa_2'+\kappa_2''+1} + 3) + \pi)
\end{aligned}$$

and it is therefore sufficient to have:

$$|\tilde{x}| \ge 2^{\kappa_3-M}(2^{\kappa_2+\kappa_2'+\kappa_2''-M+2} + 4)$$

If we choose $\kappa_3 = 18$ as above, and $\kappa_2 = 18$ we find that $\kappa_2' = 14$ and $\kappa_2'' = 15$. Therefore, the desired accuracy is obtained as long as $|\tilde{x}| \ge 65 \times 2^{-39} \simeq 1.2 \times 10^{-10}$.

Fallback

If any of the conditions above is not met, we fall back on the CORE-MATH implementation.

Accurate Tables and Their Generation

First, a remark on the notation. This section (as well is the code in `accurate_table_generator*.hpp`) follows the notation of [SZ05]. In particular, N is related to the number of bits in the mantissa of the floating-point type, and M is related to the number of zeroes or ones after the mantissa of the accurate values. Specifically, since we use `binary64` and want 18 zeroes or ones after the mantissa (to minimize the probability of expensive fall backs to the CORE-MATH implementation), we have $N = 2^{53}$ and $M = 2^{18}$.

The accurate tables are made of triples (x_k, s_k, c_k) where x_k is chosen such that $\sin x_k$ and $\cos x_k$ are very close to machine numbers, *i.e.*, have a string of zeroes or ones after the last bit of their mantissa. The tabulated value s_k (respectively, c_k) is then obtained by rounding $\sin x_k$ (respectively, $\cos x_k$) to the nearest machine number. See Overview of the Algorithms for Sin and Cos below for the details of how these tables are used for implementing sin and cos.

The argument range (which is approximately $\left[0, \frac{\pi}{4}\right]$) is split into intervals of length 2Δ centered on $2k\Delta$. The interval[7] containing x_k is $\mathcal{I}_k := [(2k-1)\Delta, (2k+1)\Delta]$ (for $k = 0$ the interval is $\mathcal{I}_0 := [0, \Delta]$ and $x_0 = 0$). We want to select x_k as close as possible to the midpoint of $\mathcal{I}_k$, because we will need to approximate sin and cos over the interval $[-h_{\max}, h_{\max}]$ with:

$$h_{\max} := \max_k(x_k - (2k-1)\Delta, (2k+1)\Delta - x_k)$$

[7] Obviously the intervals $\mathcal{I}_k$ cannot all contain their bounds. Because of the way they are computed, the odd multiples of Δ which separate the intervals are rounded to the nearest even k. Therefore, there is an alternation of open intervals (for k odd) and closed intervals (for k even). We do not have a convenient notation for this, and anyway this detail is mostly irrelevant.

and we want this interval to be as small as possible to improve the accuracy of the approximation.

In practice we choose $\Delta = 2^{-10}$ and the accurate tables construction yields $h_{\max} < \Delta + 2^{-17.834}$. The largest perturbation for x_k is reached when $k = 1$, with 35.629 bits, but that is because, for $k = 1$, we only search below 2Δ (see Sin and Cos Around Table Entries). Other than this special value, the perturbation ranges from 23.755 bits to 35.331 bits. This is consistent with the random model of [SZ05] but note that some intervals are 3000 times "harder" than others.

The naïve table construction method described in [Gal86] (random sampling over $\mathcal{I}_k$) has two problems. First, it doesn't guarantee that x_k is close to the midpoint of $\mathcal{I}_k$ and can in fact, in the worst case, result in x_k being very close to one of the bounds of the interval, requiring the construction of a minimax polynomial over an interval which is (nearly) two times too big. Second, for each triple of the accurate table it has to inspect, on average, around $M^2 = 2^{36}$ values. On a modern processor, testing one candidate takes approximately 50 µs, which amounts to about 900 hours for each triple of the table, and therefore 43 years for the 402 triples that we need.

Consequently, we use the method described in [SZ05] to build our accurate tables. It is much faster than the method in [Gal86], but must still be implemented with great care to achieve a satisfactory speed. In the rest of this section, we describe the algorithms and optimizations that we used to generate our tables efficiently. On a modern processor (AMD Ryzen Threadripper PRO 5965WX) the entire table generation takes about 17 seconds.

The Stehlé-Zimmermann Algorithm

For convenience, we copy algorithm 1 from [SZ05].

Algorithm 1: SimultaneousBadCaseSearch.

Input: Two functions F_1 and F_2, and two positive integers M, T.
Output: All $t \in [-T, T]$ such that $|F_i(t) \bmod 1| < \frac{1}{M}$ for $i \in \{1, 2\}$.

1. Let $P_1(t)$, $P_2(t)$ be the degree-2 Taylor expansions of $F_1(t)$, $F_2(t)$.
2. Compute ϵ such that $|P_i(t) - F_i(t)| < \epsilon$ for $|t| \leq T$ and $i \in \{1, 2\}$.
3. Compute $M' = \left\lfloor \frac{1/2}{1/M+\epsilon} \right\rfloor$, $C = 3M'$, and[8] $\tilde{P}_i(\tau) = \lceil C \cdot P_i(T\tau) \rfloor$ for $i \in \{1, 2\}$.
4. Let $e_1 = 1$, $e_2 = \tau$, $e_3 = \tau^2$, $e_4 = \upsilon$, $e_5 = \phi$.
5. Let $g_1 = C$, $g_2 = CT \cdot \tau$, $g_3 = \tilde{P}_1(\tau) + 3\upsilon$, $g_4 = \tilde{P}_2(\tau) + 3\phi$.
6. Create the 4×5 integral matrix L where $L_{k,l}$ is the coefficient of the monomial e_l in g_k.
7. $V \leftarrow$ `LatticeReduce`(L).
8. Let $\boldsymbol{v}_1$, $\boldsymbol{v}_2$, $\boldsymbol{v}_3$ be the three shortest vectors of V, and $Q_i(\tau)$ the associated polynomials.
9. **if** *there exist* $i \in \{1, 2, 3\}$ *such that* $\|\boldsymbol{v}_i\|_1 \geq C$ **then return** `FAIL`.
10. Let $Q(\tau)$ be a linear combination of the Q_i's which is independent of υ and ϕ. We have $\deg Q \leq 1$.
11. Let $q(t) = Q\left(\frac{t}{T}\right)$.
    foreach t_0 *in* `IntegerRoots`$\left(q, [-T, T]\right)$ **do if** $|F_i(t_0) \bmod 1| < \frac{1}{M}$ *for all* $i \in \{1, 2\}$ **then return** t_0.

When algorithm 1 returns a value at step 11, that value can be used to build an accurate table entry. When it does not return a value at step 11, the interval $[-T, T]$ is guaranteed not to contain a point suitable for building an accurate table entry. When it fails at step 9, the algorithm must be retried over $\left[-T/2, T/2\right]$ until a conclusive answer is found at step 11. Note that, despite the phrase "All t" in the description of the result, algorithm 1 returns at most one value (because $\deg Q \leq 1$).

[8] The notation $\lceil C \cdot P_i(T\tau) \rfloor$ means that we round to the nearest each coefficient of $C \cdot P_i(T\tau)$. This gives an element of $\mathbb{Z}[\tau]$. (This footnote is missing from [SZ05] but present in the preprint [SZ04].)

The Complete Search

In order to construct the accurate tables we need to build on algorithm 1 to perform a search over a arbitrary interval. In this context, if T is small enough, we are better off doing an exhaustive search than going through the fairly expensive lattice reduction. For an exhaustive search, we let the caller specify whether the search should proceed by increasing or decreasing values to make sure that we find, to the extent possible, the solution closest to the centre of $\mathcal{I}_k$. Algorithm 2 spells out the processing of one interval specifically for the sin and cos functions.

Algorithm 2: IntervalSimultaneousBadCaseSearch.

Input: The interval to search $\left[x_{\text{mid}} - \frac{T}{N}, x_{\text{mid}} + \frac{T}{N}\right]$ defined by its midpoint x_{mid} and its scaled radius T, and a positive integer M.

Output: A $t \in [-T, T]$ such that $x_{\text{mid}} + \frac{t}{N}$ is an accurate value for sin and cos; furthermore, if the search direction is `UP`, t is as close as possible to $-T$ and the search direction is `DOWN`, t is as close as possible to T.

Let $F_1 = t \mapsto N \sin\left(x_{\text{mid}} + \frac{t}{N}\right)$ and $F_2 = t \mapsto N \cos\left(x_{\text{mid}} + \frac{t}{N}\right)$.
if $T <$ *some threshold* **then**
  switch *search direction* **do**
    case *UP* **do**
      foreach $t_0 \in [-T, T]$ *by ascending values* **do**
        if $|F_i(t_0) \bmod 1| < \frac{1}{M}$ *for all* $i \in \{1, 2\}$ **then**
          return t_0.
        end
      end
    case *DOWN* **do**
      foreach $t_0 \in [-T, T]$ *by descending values* **do**
        if $|F_i(t_0) \bmod 1| < \frac{1}{M}$ *for all* $i \in \{1, 2\}$ **then**
          return t_0.
        end
      end
  end
else
  return `SimultaneousBadCaseSearch`(F_1, F_2, M, T).
end

Algorithm 2 must be integrated into a retry loop that searches progressively farther above and below $2k\Delta$ until a solution is found. This is what algorithm 3 is doing: it first searches an interval of radius T_0 above $2k\Delta$, and retries with intervals progressively smaller if algorithm 2 returns a failure. When algorithm 2 returns an empty solution set, the last interval searched is marked as "covered" and the algorithm proceeds to perform a similar search below $2k\Delta$. If that search does not find a solution, a new interval of radius T_0 above the covered interval is tried, and so on repeatedly. The algorithm returns when a solution is found. That solution is nearly optimal in terms of its distance to $2k\Delta$ with the caveat that, since we search above $2k\Delta$ before searching below, we could return a solution above when there is a slightly closer solution below.

The value of T_0 is chosen based on the Coppersmith bound established in [SZ05]: $T^3 = \mathcal{O}(M \cdot N)$. Evidently this bound does not unambiguously determine T_0, but in practice $T_0 = \sqrt[3]{MN}$ yields good results: a value too large would cause too many splits of the search interval before finding a solution or (more likely) rejecting the interval; a value too small would require to process too many small intervals.

Representation of Functions and Polynomials

Our implementation is based on the Boost `multiprecision` library. The functions sin and cos take `cpp_rational` values as arguments and return `cpp_bin_float_50`

Algorithm 3: FullSimultaneousBadCaseSearch.

Input: Δ, the half-size of intervals, and k, the index of the interval to search.
Output: An accurate value sufficiently close to $2k\Delta$.

```
1.  T_0 ←∛(MN)
2.  J_covered ←[2kΔ, 2kΔ]
3.  loop
4.      T ←T_0
5.      loop
6.          Let J_to_cover be the interval of radius T immediately above J_covered.
7.          t ←IntervalSimultaneousBadCaseSearch(J_to_cover, UP, M).
8.          if t is a solution then
9.              return Midpoint(J_to_cover) + t/N.
10.         else if t = FAIL then
11.             T ←T/2.
12.         else
13.             J_covered = J_covered ∪ J_to_cover
14.             break
15.         end
16.     end
17.     T ←T_0
18.     loop
19.         Let J_to_cover be the interval of radius T immediately below J_covered.
20.         t ←IntervalSimultaneousBadCaseSearch(J_to_cover, DOWN, M).
21.         if t is a solution then
22.             return Midpoint(J_to_cover) + t/N.
23.         else if t = FAIL then
24.             T ←T/2.
25.         else
26.             J_covered = J_covered ∪ J_to_cover
27.             break
28.         end
29.     end
30. end
```

values (50 decimal digits should be plenty for our purposes since we are looking for values that are accurate to approximately 21 digits; at any rate, while the construction of the accurate values is expensive, checking that they are correct is straightforward so we are not concerned about accuracy issues in this context). The polynomial approximations take `cpp_rational` values as arguments, have coefficients that are `cpp_rational` and return `cpp_rational`.

Step 2 of algorithm 1 requires that we compute an upper bound for $|P_i(t) - F_i(t)|$ over an interval. To do this efficiently (and in particular, without doing multiple evaluation of the functions and the polynomials), we approximate $P_i(t) - F_i(t)$ by the degree-3 term of the Taylor expansion of $F_i(t)$ and find the extremum of this term. While ignoring the terms of degree 4 and higher introduces a small error, this error is of no importance as ϵ is typically much smaller than $1/M$ and does not affect the value of M' (see step 3).

It would be possible (and correct) to compute the polynomials P_i using a Taylor approximation around $2k\Delta$ and to translate them just like we translate the functions in algorithm 2: $t \mapsto NP_i\left(x_{\text{mid}} + \frac{t}{N}\right)$. In practice however this results in polynomial coefficients with very large numerators and denominators, greatly affecting the performance of algorithm 1. It is much more efficient to recompute the Taylor expansion around x_{mid} as this produces coefficients with smaller numerators and denominators for the same accuracy. (The coefficients of the Taylor expansion involve evaluations of sin and cos, so the two methods do not generally result in the exact same polynomial.)

Lattice Reduction

Even though the matrix L is small, a fair amount of time is spent in the lattice reduction at step 7 of algorithm 1. The classical algorithm for lattice reduction is the Lenstra-Lenstra-Lovász algorithm, described for instance in [HPS14, p. 444] but it turns out to be fairly costly in our case because the matrix elements are of type `cpp_int` and operations on this type are expensive. Instead we use the algorithm described in [NS09], which is quadratic and speeds up the search significantly.

Speculative Execution

The natural way to parallelize the search on a multicore processor is to run multiple instance of algorithm 3 for distinct intervals $\mathcal{I}_k$ each in its own thread. Unfortunately this does not work very well because the searches have a "long tail": for most intervals, a solution is found relatively quickly, but for some intervals the search has to explore values far from $2k\Delta$. So after a short time only a few "expensive" intervals are left to process, effectively doing sequential searches while most of the cores are idle. This does not properly take advantage of the available paralellism.

It is much more effective to parallelize using speculative execution as follows. Each of the intervals of radius T_0 processed in steps 5 and 17 of algorithm 3 is a *slice* and slices can be searched independently for a given value of k. As soon as a core is available, it picks a slice to process, which is guaranteed not to have been processed already, but which may not be adjacent to the interval $\mathcal{I}_{\texttt{covered}}$ (it is centered at $2k\Delta + \left(2j+1\right)\frac{T_0}{N}$ where j is the *slice index*). The execution is speculative in the sense that the search for slice index j may not be necessary if a solution exists, say, for index $j-1$. However, most slices do not contain a solution, so the amount of wasted work is minimal. Note that, for speculative execution to be deterministic, it is important, if a solution is found in slice j, to wait for the searches with indices preceding j to complete before returning, as one of these slices may contain a solution closer to $2k\Delta$.

Summary

The following table summarizes the impact of each of the optimizations described above. The "Baseline" row is the time with all optimizations enabled. Each of the other rows gives the timing with exactly one optimization disabled:

Method	Elapsed Time	Ratio
Baseline	16.6 s	
No speculative execution	107.2 s	6.5×
Reduction using Lenstra-Lenstra-Lovász	237.1 s	14.3×
Polynomials centred on $2k\Delta$	around 363 000 s	22 000×

Overview of the Algorithms for Sin and Cos

This section presents an overview of the algorithms used to compute sin and cos. They take as input the result of argument reduction, $(\tilde{x}, \delta\tilde{x})$ and produce a pair $(y, \delta y)$ which is passed to the rounding test described in [Mul+10, p. 397] to decide whether $[\![y + \delta y]\!]$ is the correctly-rounded result (this is the case with high probability) or whether we need to fall back to the CORE-MATH implementation.

The lower part of the reduced angle, $\delta\tilde{x}$, must be used with care: on the one hand, it would not make sense to explicitly compute expressions like $[\![\tilde{x} + \delta\tilde{x}]\!]$ as the second term vanishes before the first one; on the other hand, we must avoid the computation of expressions involving $\delta\tilde{x}$ that have no bearing on the final result (see [Mul+10, pp. 402–404] for a discussion of this issue). We are going to spell out the places where $\delta\tilde{x}$ is actually used and we will prove later that neglecting it in other places has no effect.

Sin Near Zero

For the sin function near zero the accurate tables method is not usable because the correction term is not small compared to the tabulated value of the function (which can be arbitrarily close to zero). Instead we use a polynomial approximation that minimizes the relative error on the result. Since $\sin t$ is an odd function and since its dominant term is t, we are using an approximation of the form:

$$\sin t \simeq t + t^3\ p_{s0}(t^2)$$

and we compute:

$$\sin x \simeq \sin(\tilde{x} + \delta\tilde{x}) \simeq \tilde{x} + \tilde{x}^3 p_{s0}(\tilde{x}^2) + \delta\tilde{x}$$

Sin and Cos Around Table Entries

Let (x_k, s_k, c_k) be an accurate table entry. The implementation of sin and cos starts by choosing, from the argument $\tilde{x}$ (which in the case of sin, is not close to zero), a k such that $\tilde{x} \in \mathcal{I}_k$ and computing $h = \tilde{x} - x_k$. We then use the addition formulæ to separate out the terms in x_k:

$$\begin{aligned}
\sin x &\simeq \sin(\tilde{x} + \delta\tilde{x}) \\
&= \sin(x_k + h + \delta\tilde{x}) \\
&= \sin x_k\ \cos(h + \delta\tilde{x}) + \cos x_k\ \sin(h + \delta\tilde{x}) \\
\cos x &\simeq \cos(\tilde{x} + \delta\tilde{x}) \\
&= \cos(x_k + h + \delta\tilde{x}) \\
&= \cos x_k\ \cos(h + \delta\tilde{x}) - \sin x_k\ \sin(h + \delta\tilde{x})
\end{aligned}$$

The quantities $\sin x_k$ and $\cos x_k$ are extremely close to s_k and c_k (this is the core of the accurates table method). The terms in $h + \delta\tilde{x}$ may be approximated by polynomials chosen to respect the parity of sin and cos and their values at 0:

$$\begin{aligned}
\sin t &\simeq t + t^3\ p_s(t^2) \\
\cos t &\simeq 1 + t^2\ p_c(t^2)
\end{aligned}$$

After neglecting the terms in $\delta\tilde{x}$ that do not contribute to the final result we obtain[9]:

$$\begin{aligned}\sin x &\simeq s_k\big(1 + h(h + 2\ \delta\tilde{x})\ p_c(h^2)\big) + c_k\big(h + \delta\tilde{x} + h^3\ p_s(h^2)\big)\\ &= (s_k + c_k h) + s_k h(h + 2\ \delta\tilde{x})\ p_c(h^2) + c_k h^3\ p_s(h^2) + c_k\delta\tilde{x}\\ \cos x &\simeq c_k\big(1 + h(h + 2\ \delta\tilde{x})\ p_c(h^2)\big) - s_k\big(h + \delta\tilde{x} + h^3\ p_s(h^2)\big)\\ &= (c_k - s_k h) + c_k h(h + 2\ \delta\tilde{x})\ p_c(h^2) - s_k h^3\ p_s(h^2) - s_k\delta\tilde{x}\end{aligned}$$

where $h(h + 2\ \delta\tilde{x})$ is an approximation of $(h + \delta\tilde{x})^2$. For accuracy reasons, the leading term of these formulæ must be computed with extra accuracy.

We are now going to look at the techniques used to obtain polynomial approximation and we will later analyze the errors committed by these formulæ.

Polynomial Approximations

The *Mathematica* function `GeneralMiniMaxApproximation` produces a minimax polynomial p such that $p\big(q(t)\big)$ approximates a function $f(t)$ by minimizing the L^∞ norm of the *residual* function defined as:

$$\mathcal{E}(t) \coloneqq \frac{f(t) - p\big(q(t)\big)}{g(t)}$$

The residual is equioscillatory and bounded. The choice of f and q is dictated by the properties desired for the polynomial approximation: parity, value at zero, etc. The error function g, however, can be freely chosen to achieve the desired error bound on the approximation (*e.g.*, minimizing the relative or absolute error).

Sin Near Zero

Near zero we are looking for an approximation that minimizes the relative error of the result over the interval $[0, \Delta]$, where Δ is chosen so that $\Delta^2 \ll 1$.

We are therefore calling `GeneralMiniMaxApproximation` with:

$$\begin{cases} q(t) & \coloneqq t^2 \\ f(t) & \coloneqq \dfrac{\sin t - t}{t^3} \\ g(t) & \coloneqq \dfrac{\sin t}{t^3} \end{cases}$$

The degree of p_{s0} is choosen so that the approximation error is less than $\mathfrak{u}(\Delta^2)$.

In practice we choose $\Delta = 2^{-10}$, and compute a degree-1 polynomial with a residual smaller than $2^{-85.560}$ (before rounding the coefficients to machine numbers).

Sin and Cos Around Table Entries

Around table entries, it would be natural to approximate cos by a polynomial minimizing the absolute or relative error. Unfortunately this creates singularities in the error analysis. To see why, consider the term $h(h + 2\ \delta\tilde{x})p_c(h^2)$ which appears in the computations above, and expand it by making the error function g and the residual $\mathcal{E}$ explicit:

$$\begin{aligned}h\ (h + 2\ \delta\tilde{x})p_c(h^2) &= h\ (h + 2\ \delta\tilde{x})\left(\frac{\cos h - 1}{h^2} + g(h)\mathcal{E}(h)\right)\\ &= \cos h - 1 + 2\ \delta\tilde{x}\frac{\cos h - 1}{h} + 2\ h\ \delta\tilde{x}\ g(h)\mathcal{E}(h) + h^2 g(h)\mathcal{E}(h)\end{aligned}$$

When $h \to 0$, the dominant term is $2\ h\ \delta\tilde{x}\ g(h)\mathcal{E}(h)$. Since $\mathcal{E}(h)$ is bounded, this term is of order $\mathcal{O}\big(h\ g(h)\big)$. To obtain an absolute or relative error on cos we would need

[9] The alert reader will note that the formulæ for sin and cos are very similar, which creates an opportunity for implementing a function that computes both together with significant performance savings.

to choose $g(h) = 1/h^2$ or $g(h) = \cos h/h^2$, respectively. However, these choices yield a dominant term of order $\mathcal{O}(1/h)$ which diverges when $h \to 0$. The root cause is that we are computing an approximation based on h alone, but we use it in combination with the term in $\delta\tilde{x}$ which was not part of the optimization.

To avoid this issue, we choose $g(h) = (\cos h - 1)/h^2$ which minimizes the relative error on $p_c(h)$ and leads to a straightforward error analysis. Therefore, for cos we call `GeneralMiniMaxApproximation` with:

$$\begin{cases} q(t) & \coloneqq t^2 \\ f(t) & \coloneqq \dfrac{\cos t - 1}{t^2} \\ g(t) & \coloneqq \dfrac{\cos t - 1}{t^2} \end{cases}$$

For sin there is no such issue, therefore we simply minimize the relative error on $\sin t$:

$$\begin{cases} q(t) & \coloneqq t^2 \\ f(t) & \coloneqq \dfrac{\sin t - t}{t^3} \\ g(t) & \coloneqq \dfrac{\sin t}{t^3} \end{cases}$$

The minimax computation results in a 1-degree polynomial p_s with with a residual smaller than $2^{-85.534}$ and a 1-degree polynomial p_c with a residual smaller than $2^{-51.466}$ (before rounding the coefficients to machine numbers).

Error Analysis

We use $[\![expr]\!]_{\cdots}$ to denote evaluation where appropriate rounding happens on each literal or operation of the expression *expr*. This notation is used for error intervals computed using the function `IEEEEvaluateWithRelativeError` in file `mathematica/ieee754_floating_point_evaluation.wl`. In particular, $[\![\,]\!]_{\cdots}$ takes into account that the evaluation of the polynomials p_{s0}, p_s, and p_c has two sources of errors: the rounding of the coefficients to machine numbers, and the error due to the floating-point operations; this may lead to asymmetrical error intervals.

TODO(phl): Try to tune the high-degree coefficient after rounding the low-degree one.

We assume that $\tilde{x}$ is positive ($\delta\tilde{x}$ may be positive or negative) and that the rounding direction is roundTiesToEven. We *do not* assume that the machine has an FMA instruction in our error analysis: even though we actually use this instruction when available (for performance) our rounding bounds are valid even in the absence of an FMA[10].

Reduced Angle

Argument reduction took the input angle x and produced a pair $(\tilde{x}, \delta\tilde{x})$ approximating the angle reduced modulo $\frac{\pi}{2}$. That approximation is correct to $M + \kappa_3$ bits. We therefore have, from equation (11):

$$x \equiv \tilde{x} + \delta\tilde{x} + \zeta_0\ \tilde{x} \pmod{\frac{\pi}{4}} \qquad |\zeta_0| \leq 2^{-\kappa_3 - M}$$

For simplicity of the error analysis we can assume $0 \leq x \leq \frac{\pi}{4}$: even though we do not do argument reduction when x is less than $\frac{\pi}{4}$, the error analysis would work exactly the same if x was simply provided as a high-accuracy value (*e.g.*, a double-double) and reduced using the techniques above.

[10] This means that the accuracy computed below is possibly pessimistic by a few hundredth of bits, which has no effect on performance.

Rounding Test

The rounding test described in [Mul+10, pp. 397–400] is done by comparing y and $\left[\!\left[y + \left[\!\left[\delta y\ e\right]\!\right]\right]\!\right]$ for equality, where $e > 1$ is computed based on the bound on the relative error of $y + \delta y$ with respect to $\sin x$ or $\cos x$. In our case, when an FMA is available, we want to compute the second part as $\left[\!\left[y + \delta y\ e\right]\!\right]$. We must analyse what this implies for the computation of e.

The proof of the rounding test is rather convoluted, but for our purpose the important part is the implication[11], when y is not a power of 2 or $\delta y \geq 0$:

$$y = \left[\!\left[y + \left[\!\left[\delta y\ e\right]\!\right]\right]\!\right] \implies \left[\!\left[\delta y\ e\right]\!\right] \leq \frac{\mathrm{u}(y)}{2} \implies \delta y\ e \left(1 - \mathrm{u}\left(\frac{1}{2}\right)\right) \leq \frac{\mathrm{u}(y)}{2}$$

When an FMA is used the smaller term has to be less that half a ULP of the larger term, otherwise the result would round away from y. This gives the tighter bound:

$$y = \left[\!\left[y + \delta y\ e\right]\!\right] \implies \delta y\ e \leq \frac{\mathrm{u}(y)}{2}$$

Similarly when y is a power of 2 and $\delta y < 0$ the proof depends on the implications:

$$y = \left[\!\left[y + \left[\!\left[\delta y\ e\right]\!\right]\right]\!\right] \implies \left|\left[\!\left[\delta y\ e\right]\!\right]\right| \leq \frac{\mathrm{u}(y)}{4} \implies -\delta y\ e \left(1 - \mathrm{u}\left(\frac{1}{2}\right)\right) \leq \frac{\mathrm{u}(y)}{4}$$

When an FMA is used the smaller term, which is negative, has to be less than half a ULP *below* the larger term otherwise the result would round below y. Because the ULP below is half the ULP above, this gives the tighter bound:

$$y = \left[\!\left[y + \delta y\ e\right]\!\right] \implies -\delta y\ e \leq \frac{\mathrm{u}(y)}{4}$$

Putting these tighter bounds together we obtain:

$$k \coloneqq \left\lfloor -\log_2 \bar{\epsilon}_1 - M \right\rfloor$$
$$e_{\mathrm{FMA}} \coloneqq 1 + \frac{1 + 2^{M+1}\ \bar{\epsilon}_1}{1 - \bar{\epsilon}_1 - 2^{-k+1}}$$

In the absence of FMA, [Mul+10] gives the same value for k and for e:

$$e_{\mathrm{NoFMA}} \coloneqq (1 - 2^{-M})^{-1}\left(1 + \frac{1 + 2^{M+1}\ \bar{\epsilon}_1}{1 - \bar{\epsilon}_1 - 2^{-k+1}}\right)$$

We see immediately that $e_{\mathrm{FMA}} < e_{\mathrm{NoFMA}}$, and it is therefore correct to perform the rounding test using e_{NoFMA}, even if an FMA is used to compute $\left[\!\left[y + \delta y\ e\right]\!\right]$.

Obviously, if the above expression does not produce a machine number, it must be rounded towards $+\infty$. In each of the error analyses below we document the concrete factor $\left[\!\left[e_{\mathrm{NoFMA}}\right]\!\right]_+$ to use for the rounding test.

Sin Near Zero

If $|\tilde{x}| \leq \Delta = 2^{-10}$ the steps of the computation are as follows:

$$\begin{cases} t_1 & \coloneqq \left[\!\left[p_{s0}(\tilde{x}^2)\right]\!\right]_{\ldots} \\ t_2 & \coloneqq \left[\!\left[\left[\!\left[\tilde{x}^2\right]\!\right]\ \tilde{x}\right]\!\right] \\ t_3 & \coloneqq \left[\!\left[\left[\!\left[t_1 t_2\right]\!\right] + \delta\tilde{x}\right]\!\right] \\ y & \coloneqq \tilde{x} \\ \delta y & \coloneqq t_3 \end{cases}$$

[11] We adapt the proof to use our notation.

We gave above the upper bound of the residual of the minimax approximation. It may be rewritten as:

$$p_{s0}(t^2) = \frac{(1+\zeta_1)\sin t - t}{t^3} \qquad |\zeta_1| < 2^{-85.560}$$

for $|t| \leq \Delta$.

The errors committed at each step are as follows:

$$\begin{cases} t_1 = p_{s0}(\tilde{x}^2)(1+\zeta_2) & \zeta_2 \in \,]-2^{-52.415}, 2^{-53.999}[\\ \quad = \dfrac{(1+\zeta_1)\sin\tilde{x} - x}{\tilde{x}^3}(1+\zeta_2) & \\ t_2 = \tilde{x}^3(1+\delta_1)(1+\delta_2) & \\ t_3 = (t_1 t_2(1+\delta_3) + \delta\tilde{x})(1+\delta_4) & \end{cases}$$

The relative error of the entire computation is:

$$r(\tilde{x}, \delta\tilde{x}) \coloneqq \frac{y+\delta y}{\sin(\tilde{x}+\delta\tilde{x}+\zeta_0\ \tilde{x})} - 1$$

TODO(phl): See if this can be proven rigourously.

The function r can be computed using *Mathematica* interval arithmetic over the triangular domain $|\tilde{x}| \leq \Delta$, $|\delta\tilde{x}| \leq 2^{-M}|\tilde{x}|$. Plotting shows that this function reaches its extrema at the corners of the domain, which is logical because we expect the minimax polynomial to reach its largest errors on the bounds of the optimization interval. In practice we find that $|r(\tilde{x}, \delta\tilde{x})| < 2^{-70.583}$ (as expected, this is a bit worse than the error originating from the angle reduction). The rounding test must be done with $e =$ `0x1.0000'AAD0'391A'Dp0`.

TODO(phl): $\kappa_3 = 19$ would gain 0.7 bits. Worthwhile?

As mentioned above (see Overview of the Algorithms for Sin and Cos), in these steps we do not compute the terms $\tilde{x}^n\ \delta\tilde{x}$ for $n > 0$. The largest such term is for $n = 2$ and the relative error that it induces is:

$$-\frac{1}{2}\frac{\tilde{x}^2\ \delta\tilde{x}}{\sin(\tilde{x}+\delta\tilde{x})}$$

This function reaches its extrema on the corners of the triangular domain and is found to be smaller than $2^{-73.999}$, so its contribution to the overall relative error would be very small.

Sin and Cos Around Table Entries

If $\tilde{x} \in \mathcal{I}_k$ we first compute $h = \tilde{x} - x_k$. It is essential that this subtraction be exact, and therefore that the conditions of Sterbenz's lemma be met: we must have $\tilde{x}/2 \leq x_k \leq 2\tilde{x}$. Based on the range of $\tilde{x}$, a sufficient condition for this is:

$$\frac{2k+1}{2}\Delta \leq x_k \leq 2(2k-1)\Delta$$

Because x_k is close to $2k\Delta$ the left part of this condition is trivially met for all $k > 0$. However the second part is only trivially met if $k > 1$. If $k = 1$ it becomes $x_1 \leq 2\ \Delta$. In other words, when building the accurate tables we must only look for x_1 *below* $2\ \Delta$. This is done by skipping the loop at step 5 of algorithm 3.

For sin we do not use the interval $\mathcal{I}_0$ (see Sin Near Zero, above) but for cos we do. That interval is special because $x_0 = 0$: the computation of h is therefore trivially correct.

Interestingly, $x_1 < 2\Delta$ is also the necessary condition to compute $h\ c_k + s_k$ exactly using an FMA as explained in section 2.1 of [SZ05]. The subtraction in the second step of their algorithm must be exact, therefore the following Sterbenz inequalities must hold[12]:

$$\frac{s_k}{2} \leq h\ c_k + s_k \leq 2\ s_k$$

[12]Note that $\mathcal{I}_0$ is uninteresting because $c_0 = 1$ and $s_0 = 0$ so the computation is trivially exact.

This may be rewritten as:

$$-\frac{1}{2}\tan x_k \le h \le \tan x_k$$

Let $x_k = 2k\Delta + \epsilon_k$ for some small ϵ_k. By the definition of h we have:

$$-\Delta - \epsilon_k \le h \le \Delta - \epsilon_k$$

Now we know that $x_k < \tan x_k$ and since $\Delta - \epsilon_k < 2k\Delta + \epsilon_k$ for all $k > 0$ the right side of the Sterbenz condition is always true. The left side is more interesting though as:

$$-\frac{1}{2}\tan x_k < -\frac{x_k}{2} = -k\Delta - \frac{\epsilon_k}{2}$$

For $k > 1$, this quantity is clearly smaller than $-\Delta - \epsilon_k$, the lower bound of h. However, when $k = 1$, $-\Delta - \epsilon_1/2 \le -\Delta - \epsilon_1$ requires that $\epsilon_1 \le 0$, in other words, that x_1 be below 2Δ. Note that we do not need to go through a similar proof for the exactness of $-h\ s_k + c_k$ because $h\ s_k$ is much smaller than c_k for all k.

Finally, for the purpose of error analysis, the relative errors of the minimax polynomials may be rewritten as:

$$p_s(t^2) = \frac{\sin t\ (1+\zeta_1) - t}{t^3} \qquad |\zeta_1| < 2^{-85.534}$$
$$p_c(t^2) = \frac{\cos t - 1}{t^2}(1+\zeta_2) \qquad |\zeta_2| < 2^{-51.466}$$

Sin

The first step of the computation is to evaluate $h\ c_k + s_k$ exactly[13] using an FMA:

TODO(phl): Examine what happens when we don't have FMA.

$$(z, \delta z) = t_0 = h\ c_k + s_k$$

where z and δz have nonoverlapping significands. For the purpose of describing the computation and analysing errors we will write $\delta z = t_0\ \delta_0$ and $z = t_0(1-\delta_0)$, where the two terms are exact.

The remaining steps of the computation are then as follows:

$$\begin{cases} t_1 & \coloneqq [\![p_s(h^2)]\!]_{\dots} \\ t_2 & \coloneqq [\![p_c(h^2)]\!]_{\dots} \\ t_3 & \coloneqq [\![h[\![h + (\delta\tilde{x} + \delta\tilde{x})]\!]]\!] \\ t_4 & \coloneqq [\![[\![h^2]\!]\ h]\!] \\ t_5 & \coloneqq \left[\!\left[[\![s_k]\!]\ t_3\right]\!]\ t_2\right] \\ t_6 & \coloneqq [\![[\![t_4\ t_1]\!] + \delta\tilde{x}]\!] \\ t_7 & \coloneqq \left[\!\left[[\![c_k]\!]\ t_6\right]\!] + t_5\right] \\ t_8 & \coloneqq [\![[\![t_0\ \delta_0]\!] + t7]\!] \\ y & \coloneqq t_0(1-\delta_0) \\ \delta y & \coloneqq t_8 \end{cases}$$

where we have made the rounding of the accurate table elements s_k and c_k explicit, and used the fact that the computation of $\delta\tilde{x} + \delta\tilde{x}$ is exact.

[13] See Proof of the FMA Computation for a proof of the correctness of this computation.

The errors commited at each step are as follows:

$$\begin{cases} t_1 = p_s(h^2)\big(1+\zeta_3\big) & \zeta_3 \in \,]-2^{-53.221}, 2^{-52.808}[\\ \quad = \dfrac{\sin h\,\big(1+\zeta_1\big) - h}{h^3}\big(1+\zeta_3\big) & \\ t_2 = p_c(h^2)\big(1+\zeta_4\big) & \zeta_4 \in \,]-2^{-52.855}, 2^{-53.160}[\\ \quad = \dfrac{\cos h - 1}{h^2}\big(1+\zeta_2\big)\big(1+\zeta_4\big) & \\ t_3 = h(h + 2\ \delta\tilde{x})(1+\delta_1)(1+\delta_2) & \\ t_4 = h^3(1+\delta_3)(1+\delta_4) & \\ t_5 = [\![s_k]\!]\ t_3\ t_2\ (1+\delta_5)(1+\delta_6) & \\ t_6 = (t_4\ t_1\ (1+\delta_7) + \delta\tilde{x})(1+\delta_8) & \\ t_7 = ([\![c_k]\!]\ t_6\ (1+\delta_9) + t_5)(1+\delta_{10}) & \\ t_8 = (t_0\ \delta_0\ (1+\delta_{11}) + t_7)(1+\delta_{12}) & \end{cases}$$

The relative error of the entire computation is:

$$r(\tilde{x}, \delta\tilde{x}) \coloneqq \frac{y + \delta y}{\sin\big(\tilde{x} + \delta\tilde{x} + \zeta_0\ \tilde{x}\big)} - 1$$

Great care is required when evaluating this expression using interval arithmetic because the terms in δ_0 in t_8 and y could be considered independent and lead to a useless bound. To avoid this issue δ_0 must be factored out to appear only once. Specifically the δ_0 term after factoring is:

$$(h[\![c_k]\!] + [\![s_k]\!])((1+\delta_{11})(1+\delta_{12}) - 1)$$

which is pleasantly small, and even though δ_{12} also appears in the term that does not involve δ_0, the effect of replicating it is acceptable.

The function r must be evaluated separately for each interval around x_k, with $\tilde{x} \in [(2k-1)\Delta, (2k+1)\Delta]$ and $|\delta\tilde{x}| < |\tilde{x}|/2^M$. This defines a trapezoidal domain and r reaches its extrema at the corners of that domain. In practice we find using *Mathematica* interval arithmetic that $|r(\tilde{x}, \delta\tilde{x})| < 2^{-68.734}$. The relative error for $k = 1$ is particularly high and is an outlier. Other than than, the worst relative errors correspond to table entries that have many ones after their accurate zeroes (or many zeroes after their accurate ones). The rounding test must be done with $e =$ `0x1.0002'6752'8572'Dp0`.

TODO(phl): Try to understand why.

TODO(phl): Should we try accurate tables with 19 bits?

As mentioned above (see Overview of the Algorithms for Sin and Cos), in these steps we do not compute the terms $h^n\ \delta\tilde{x}$ for $n > 1$. The largest such term is for $n = 2$ and the relative error that it induces is:

$$-\frac{1}{2}\frac{h^2\ \delta\tilde{x}\ \cos x_k}{\sin(\tilde{x} + \delta\tilde{x})}$$

This function reaches its extrema on the corners of the trapezoidal domain and is found to be smaller than $2^{-73.999}$ for all k, so its contribution to the overall relative error would be very small. (The term for $n = 1$, on the other hand, induces an error of the order of $2^{-63.348}$ so it must be computed.)

Cos

TODO(phl): Prove that this works, and examine what happens when we don't have FMA.

The first step of the computation is to evaluate $-h\ s_k + c_k$ exactly using an FMA:

$$(z, \delta z) = t_0 = -h\ s_k + c_k$$

where z and δz have nonoverlapping significands. For the purpose of describing the computation and analysing errors we will write $\delta z = t_0\ \delta_0$ and $z = t_0(1-\delta_0)$, where the two terms are exact.

The remaining steps of the computation are then as follows:

$$\begin{cases} t_1 & \coloneqq [\![p_s(h^2)]\!]_{\dots} \\ t_2 & \coloneqq [\![p_c(h^2)]\!]_{\dots} \\ t_3 & \coloneqq [\![h[\![h + (\delta\tilde{x} + \delta\tilde{x})]\!]]\!] \\ t_4 & \coloneqq [\![[\![h^2]\!]\ h]\!] \\ t_5 & \coloneqq \left[\!\left[[\![[\![c_k]\!]\ t_3]\!]\ t_2\right]\!\right] \\ t_6 & \coloneqq [\![[\![t_4\ t_1]\!] + \delta\tilde{x}]\!] \\ t_7 & \coloneqq \left[\!\left[-[\![[\![s_k]\!]\ t_6]\!] + t_5\right]\!\right] \\ t_8 & \coloneqq [\![[\![t_0\ \delta_0]\!] + t_7]\!] \\ y & \coloneqq t_0(1-\delta_0) \\ \delta y & \coloneqq t_8 \end{cases}$$

where we have made the rounding of the accurate table elements s_k and c_k explicit, and used the fact that the computation of $\delta\tilde{x} + \delta\tilde{x}$ is exact.

The errors commited at each step are as follows:

$$\begin{cases} t_1 = p_s(h^2)\big(1+\zeta_3\big) & \zeta_3 \in \,]-2^{-53.221}, 2^{-52.808}[\\ \quad = \dfrac{\sin h\ \big(1+\zeta_1\big) - h}{h^3}\big(1+\zeta_3\big) & \\ t_2 = p_c(h^2)\big(1+\zeta_4\big) & \zeta_4 \in \,]-2^{-52.855}, 2^{-53.160}[\\ \quad = \dfrac{\cos h - 1}{h^2}\big(1+\zeta_2\big)\big(1+\zeta_4\big) & \\ t_3 = h(h + 2\ \delta\tilde{x})(1+\delta_1)(1+\delta_2) & \\ t_4 = h^3(1+\delta_3)(1+\delta_4) & \\ t_5 = [\![c_k]\!]\ t_3\ t_2\ (1+\delta_5)(1+\delta_6) & \\ t_6 = (t_4\ t_1\ (1+\delta_7) + \delta\tilde{x})(1+\delta_8) & \\ t_7 = ([\![s_k]\!]\ t_6\ (1+\delta_9) + t_5)(1+\delta_{10}) & \\ t_8 = (t_0\ \delta_0\ (1+\delta_{11}) + t_7)(1+\delta_{12}) & \end{cases}$$

The relative error of the entire computation is:

$$r(\tilde{x}, \delta\tilde{x}) \coloneqq \frac{y + \delta y}{\sin\big(\tilde{x} + \delta\tilde{x} + \zeta_0\ \tilde{x}\big)} - 1$$

As explained above (see Sin) the expression for r must be rewritten so that δ_0 only appears once. In this case the δ_0 term is:

$$(h[\![s_k]\!] - [\![c_k]\!])(1 - (1+\delta_{11})(1+\delta_{12}))$$

and a *Mathematica* computation yields $|r(\tilde{x}, \delta\tilde{x})| < 2^{-69.217}$. Again, the largest errors come from table entries that have many ones after their accurate zeroes (or many zeroes after their accurate ones). The rounding test must be done with $e =$ `0x1.0001'B838'5D8B'6p0`.

As mentioned above (see Overview of the Algorithms for Sin and Cos), in these steps we do not compute the terms $h^n\ \delta\tilde{x}$ for $n > 1$. The largest such term is for $n = 2$ and the relative error that it induces is:

$$\frac{1}{2}\frac{h^2\ \delta\tilde{x}\ \sin x_k}{\cos(\tilde{x} + \delta\tilde{x})}$$

This function reaches its extrema on the corners of the trapezoidal domain and is found to be smaller than $2^{-74.348}$ for all k, so its contribution to the overall relative error would be very small. (The term for $n = 1$, on the other hand, induces an error of the order of $2^{-63.346}$ so it must be computed.)

Annex: The First-Order Term

Sin

TODO(phl): Move here the Sterbenz proof.

Proof of the FMA Computation

[SZ05] presents, in section 2.1, the following algorithm to compute $s_k + c_k h$ exactly:

$$\begin{cases} h' &= [\![s_k + c_k h]\!] \\ t &= h' - s_k \\ l &= [\![t - c_k h]\!] \end{cases}$$

where the computation of t is exact if h' and s_k verify the conditions of Sterbenz's Lemma. [SZ05] then states without proof that $h' + l = s_k + c_k h$.

The first thing to note is that there is a sign error in that last formula. Let's make the absolute errors ζ_i of each computation explicit:

$$\begin{cases} h' &= s_k + c_k h + \zeta_1 \\ t &= h' - s_k = c_k h + \zeta_1 \\ l &= [\![t - c_k h]\!] = [\![\zeta_1]\!] = \zeta_1 + \zeta_2 \end{cases}$$

It is clear that the sum $h' + l = s_k + c_k h + 2\ \zeta_1 + \zeta_2$ doubles the error ζ_1 (which is large) while the difference $h' - l = s_k + c_k h - \zeta_2$ eliminates it and only retains ζ_2 (which is much smaller).

To prove that $h' - l = s_k + c_k h$ we must prove that $\zeta_2 = 0$, which is equivalent to $[\![\zeta_1]\!] = \zeta_1$. In other words, if ζ_1 is a machine number then $\zeta_2 = 0$ and the equality holds.

The first thing that we show is that the exact value $s_k + c_k h$ has at most $2M+1$ bits in its binary representation. Note that $h = x - x_k$, where the subtraction is exact, and therefore there exists $N \in]-2^M, 2^M[\cap \mathbb{N}$ such that $h = N\,\mathrm{u}(x_k)$. Moreover, because $x_k \in \left[0, \frac{\pi}{4}\right]$ we have $1/2 < c_k < 1$ and therefore $\mathrm{u}(c_k) = \mathrm{u}\big(1/2\big) = 2^{-M}$. Finally, $x_k/2 < s_k < x_k$, which implies that $\mathrm{u}(s_k)$ is either $\mathrm{u}(x_k)$ or $\mathrm{u}(x_k)/2$. We need to distinguish two cases:

— If $\mathrm{u}(s_k) = \mathrm{u}(x_k)$ then:

$$\begin{aligned} s_k + c_k h &= (\mathrm{m}(s_k) + \mathrm{m}(c_k) 2^{-M} N)\,\mathrm{u}(x_k) \\ &= (\mathrm{m}(s_k) 2^M + \mathrm{m}(c_k) N)\,\mathrm{u}(x_k) 2^{-M} \end{aligned}$$

where the quantity in parentheses is an integer strictly less than 2^{2M+1} by the properties of the significand function m.

— If $\mathrm{u}(s_k) = \mathrm{u}(x_k)/2$ then:

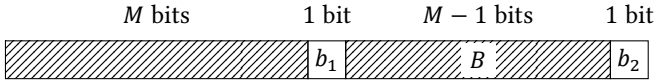
$$\begin{aligned} s_k + c_k h &= \left(\frac{\mathrm{m}(s_k)}{2} + \mathrm{m}(c_k) 2^{-M} N\right) \mathrm{u}(x_k) \\ &= (\mathrm{m}(s_k) 2^{M-1} + \mathrm{m}(c_k) N)\,\mathrm{u}(x_k) 2^{-M} \end{aligned}$$

where the quantity in parentheses is an integer strictly less than $3 \cdot 2^{2M-1} < 2^{2M+1}$.

Taken together these inequalities prove that $s_k + c_k h$ can be represented using $2M+1$ significand bits with an ULP of $\mathrm{u}(x_k) 2^{-M}$.

The computation of h' is going to round the most significand M bits of that significand and leave a remainder, ζ_1, which has potentially $M+1$ bits. We are now going to show that in fact ζ_1 has only M bits in its significand.

The following picture represents the significand of $s_k + c_k h$, with the least significant bit on the right. The leftmost M bits are the part that is rounded (either up or down). The rightmost $M+1$ bits are the ones that contribute to ζ_1. We consider different cases based on the values of bits b_1 and b_2.



— If $b_2 = 0$, the significand of $s_k + c_k h$ effectively only has $2M$ bits, and therefore the significand of ζ_1 trivially has at most M bits.

— If $b_2 = 1$, we have two cases depending on the value of b_1:

  – If $b_1 = 0$, the leading M bits are rounded *down* and therefore the significand of the positive quantity $s_k + c_k h - h' = -\zeta_1$ is exactly made of the bits (b_1, B, b_2). Since $b_1 = 0$, this is only the M bits (B, b_2) and therefore ζ_1 is a machine number (with a "hole" separating it from the significand of h').
  – If $b_1 = 1$, the leading M bits were rounded *up*[14] and therefore the significand of the positive quantity $h' - s_k + c_k h = \zeta_1$ is obtained by taking the twos complement of the bits (b_1, B, b_2). This is exactly $(0, \neg B, 1)$ where $\neg$ designates binary negation. Again, this is only M bits because of the leading zero and ζ_1 is a machine number.

Computation Without FMA

When an FMA instruction is not available, the computation proceeds as follows:

$$\begin{cases} p & = c_k h \\ (z, \delta z) & = \mathrm{LongAdd}(s_k, p) \end{cases}$$

Cos

FMA Computation

For the cos function, the equivalent to the computation described in Proof of the FMA Computation is as follows:

$$\begin{cases} h' & = [\![c_k - s_k h]\!] \\ t & = h' - c_k \\ l & = [\![t + s_k h]\!] \end{cases}$$

However, we note that $c_k > 1/2$, $h < 2\Delta$, and $s_k < 1$. Therefore $\frac{s_k h}{c_k} < 4\Delta$. While this trivially ensure that the subtraction in the evaluation of t is exact by Sterbenz's Lemma, it also means that the exact value of $c_k - s_k h$ may have, with our choice of $\Delta = 2^{-10}$, as many as $2M + 7$ bits. Therefore, it cannot be represented exactly using two machine numbers, and we have to tolerate some rounding error. The error analysis is as follows:

$$\begin{cases} h' & = (c_k - s_k h)(1 + \delta_1) \\ t & = h' - c_k \\ l & = (t + s_k h)(1 + \delta_2) = (c_k - s_k h)\delta_1(1 + \delta_2) \end{cases}$$

and:

$$h' - l = (c_k - s_k h)(1 - \delta_1 \delta_2)$$

References

[Gal86] S. Gal. "Computing elementary functions: A new approach for achieving high accuracy and good performance". In: *Accurate Scientific Computations* (Bad Neuenahr, Federal Republic of Germany, Mar. 12–14, 1985). Ed. by W. Miranker and R. Toupin. Lecture Notes in Computer Science 235. Springer Berlin Heidelberg, Sept. 1986, pp. 1–16. DOI: 10.1007/3-540-16798-6_1.

[14] Because $b_2 = 1$ the trailing bits are above the midpoint, so the handling of ties is irrelevant.

[GB91] S. Gal and B. Bachelis. "An Accurate Elementary Mathematical Library for the IEEE Floating Point Standard". In: *ACM Transactions on Mathematical Software* 17.1 (Mar. 1991), pp. 26–45.

[Hig02] N. J. Higham. *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics, 2002.

[HLB08] Y. Hida, X. S. Li, and D. H. Bailey. "Library for Double-Double and Quad-Double Arithmetic". Preprint at `https://www.davidhbailey.com/dhbpapers/qd.pdf`. May 8, 2008.

[HPS14] J. Hoffstein, J. Pipher, and J. H. Silverman. *An Introduction to Mathematical Cryptography*. Undergraduate Texts in Mathematics. Springer, 2014.

[Lin81] S. Linnainmaa. "Software for Doubled-Precision Floating-Point Computations". In: *ACM Transactions on Mathematical Software* 7.3 (Sept. 1981), pp. 272–283.
DOI: `10.1145/355958.355960`.

[Mul+10] J.-M. Muller, N. Brisebarre, F. De Dinechin, C.-P. Jeannerod, V. Lefèvre, G. Melquiond, N. Revol, D. Stehlé, and S. Torres. *Handbook of Floating-Point Arithmetic*. Birkhäuser, 2010.

[NS09] P. Q. Nguyễn and D. Stehlé. "An LLL Algorithm with Quadratic Complexity". In: *SIAM Journal on Computing* 39.3 (2009), pp. 874–903.
DOI: `10.1137/070705702`.

[SZ04] D. Stehlé and P. Zimmermann. *Gal's Accurate Tables Method Revisited*. Research Report RR-5359. INRIA, Oct. 2004.
eprint: `https://inria.hal.science/inria-00070644v1/file/rr-5359.pdf`.

[SZ05] D. Stehlé and P. Zimmermann. "Gal's accurate tables method revisited". In: *17th IEEE Symposium on Computer Arithmetic (ARITH'05)* (Cape Cod, MA, USA, June 27–29, 2005). Ed. by P. Montuschi and E. Schwarz. IEEE Computer Society, June 2005, pp. 257–264.
DOI: `10.1109/ARITH.2005.24`.

[SZG22] A. Sibidanov, P. Zimmermann, and S. Glondu. "The CORE-MATH Project". In: *2022 IEEE 29th Symposium on Computer Arithmetic (ARITH)*. IEEE, Sept. 2022, pp. 26–34.
DOI: `10.1109/ARITH54963.2022.00014`.
eprint: `https://inria.hal.science/hal-03721525v3/file/core-math-final.pdf`.

[ZSG+24] P. Zimmermann, A. Sibidanov, S. Glondu, et al. *The CORE-MATH Project*. Software. Apr. 2024.