# THE MPFR LIBRARY: ALGORITHMS AND PROOFS

THE MPFR TEAM

## Contents

`http://www.mpfr.org/`

## 1. Error calculus

Let $n$ — the working precision — be a positive integer (considered fixed in the following). We write any nonzero real number $x$ in the form $x = m \cdot 2^e$ with $\frac{1}{2} \le |m| < 1$ and $e := \mathrm{EXP}(x)$, and we define $\mathrm{ulp}(x) := 2^{\mathrm{EXP}(x)-n}$.

### 1.1. **Ulp calculus.**

**Rule 1.** $2^{-n}|x| < \mathrm{ulp}(x) \le 2^{-n+1}|x|$.

*Proof.* Obvious from $x = m \cdot 2^e$ with $\frac{1}{2} \le |m| < 1$. □

**Rule 2.** *If $a$ and $b$ have same precision $n$, and $|a| \le |b|$, then* $\mathrm{ulp}(a) \le \mathrm{ulp}(b)$.

*Proof.* Write $a = m_a \cdot 2^{e_a}$ and $b = m_b \cdot 2^{e_b}$. Then $|a| \le |b|$ implies $e_a \le e_b$, thus $\mathrm{ulp}(a) = 2^{e_a - n} \le 2^{e_b - n} = \mathrm{ulp}(b)$. □

**Rule 3.** *Let $x$ be a real number, and $y = \circ(x)$. Then $|x - y| \le \frac{1}{2}\min(\mathrm{ulp}(x), \mathrm{ulp}(y))$ in rounding to nearest, and $|x - y| \le \min(\mathrm{ulp}(x), \mathrm{ulp}(y))$ for the other rounding modes.*

*Proof.* First consider rounding to nearest. By definition, we have $|x - y| \le \frac{1}{2}\mathrm{ulp}(y)$. If $\mathrm{ulp}(y) \le \mathrm{ulp}(x)$, then $|x - y| \le \frac{1}{2}\mathrm{ulp}(y) \le \frac{1}{2}\mathrm{ulp}(x)$. The only difficult case is when $\mathrm{ulp}(x) < \mathrm{ulp}(y)$, but then necessarily $y$ is a power of two; since in that case $y - \frac{1}{2}\mathrm{ulp}(y)$ is exactly representable, the maximal possible difference between $x$ and $y$ is $\frac{1}{4}\mathrm{ulp}(y) = \frac{1}{2}\mathrm{ulp}(x)$, which concludes the proof in the rounding to nearest case.

In rounding to zero, we always have $\mathrm{ulp}(y) \le \mathrm{ulp}(x)$, so the rule holds. In rounding away from zero, the only difficult case is when $\mathrm{ulp}(x) < \mathrm{ulp}(y)$, but then $y$ is a power of two, and since $y - \frac{1}{2}\mathrm{ulp}(y)$ is exactly representable, the maximal possible difference between $x$ and $y$ is $\frac{1}{2}\mathrm{ulp}(y) = \mathrm{ulp}(x)$, which concludes the proof. □

**Rule 4.** $\frac{1}{2}|a| \cdot \mathrm{ulp}(b) < \mathrm{ulp}(ab) < 2|a| \cdot \mathrm{ulp}(b)$.

*Proof.* Write $a = m_a 2^{e_a}$, $b = m_b \cdot 2^{e_b}$, and $ab = m2^e$ with $\frac{1}{2} \le m_a, m_b, m < 1$, then $\frac{1}{4}2^{e_a+e_b} \le |ab| < 2^{e_a+e_b}$, thus $e = e_a+e_b$ or $e = e_a+e_b-1$, which implies $\frac{1}{2}2^{e_a}\mathrm{ulp}(b) \le \mathrm{ulp}(ab) \le 2^{e_a}\mathrm{ulp}(b)$ using $2^{e_b-n} = \mathrm{ulp}(b)$, and the rule follows from the fact that $|a| < 2^{e_a} \le 2|a|$ (equality on the right side can occur only if $e = e_a + e_b$ and $m_a = \frac{1}{2}$, which are incompatible). □

**Rule 5.** $\mathrm{ulp}(2^k a) = 2^k \mathrm{ulp}(a)$.

*Proof.* Easy: if $a = m_a \cdot 2^{e_a}$, then $2^k a = m_a \cdot 2^{e_a+k}$. □

**Rule 6.** *Let $x > 0$, $o(\cdot)$ be any rounding, and $u := o(x)$, then $\frac{1}{2}u < x < 2u$.*

*Proof.* Assume $x \ge 2u$, then $2u$ is another representable number which is closer from $x$ than $u$, which leads to a contradiction. The same argument proves $\frac{1}{2}u < x$. □

**Rule 7.** $\frac{1}{2}|a| \cdot \mathrm{ulp}(1) < \mathrm{ulp}(a) \le |a| \cdot \mathrm{ulp}(1)$.

*Proof.* The left inequality comes from Rule 4 with $b = 1$, and the right one from $|a|\mathrm{ulp}(1) \ge \frac{1}{2}2^{e_a}2^{1-n} = \mathrm{ulp}(a)$. □

**Rule 8.** *For any $x \ne 0$ and any rounding mode $o(\cdot)$, we have* $\mathrm{ulp}(x) \le \mathrm{ulp}(o(x))$*, and equality holds when rounding towards zero, towards $-\infty$ for $x > 0$, or towards $+\infty$ for $x < 0$.*

*Proof.* Without loss of generality, assume $x > 0$. Let $x = m \cdot 2^e$ with $\frac{1}{2} \le m < 1$. As $\frac{1}{2}2^{e_x}$ is a machine number, necessarily $o(x) \ge \frac{1}{2}2^{e_x}$, thus by Rule 2, then $\mathrm{ulp}(o(x)) \ge 2^{e_x-n} = \mathrm{ulp}(x)$. If we round towards zero, then $o(x) \le x$ and by Rule 2 again, $\mathrm{ulp}(o(x)) \le \mathrm{ulp}(x)$. □

**Rule 9.**

$$\begin{aligned}&\text{For } \ \mathit{error}(u) \le k_u \mathrm{ulp}(u), \ \ u.c_u^- \le x \le u.c_u^+ \\ &\text{with } \ c_u^- = 1 - k_u 2^{1-p} \text{ and } c_u^+ = 1 + k_u 2^{1-p}\end{aligned}$$

$$\begin{aligned}&\text{For } \ u = o(x), \ \ u.c_u^- \le x \le u.c_u^+ \\ &\text{if } \ u = \triangle(x), \ \text{then } c_u^+ = 1 \\ &\text{if } \ u = \bigtriangledown(x), \ \text{then } c_u^- = 1 \\ &\text{if } \ \text{for } x < 0 \text{ and } u = Z(x), \ \text{then } c_u^+ = 1 \\ &\text{if } \ \text{for } x > 0 \text{ and } u = Z(x), \ \text{then } c_u^- = 1 \\ &\text{else } \ c_u^- = 1 - 2^{1-p} \text{ and } c_u^+ = 1 + 2^{1-p}\end{aligned}$$

1.2. **Relative error analysis.** Another way to get a bound on the error, is to bound the relative error. This is sometimes easier than using the "ulp calculus" especially when performing only multiplications or divisions.

**Rule 10.** *If $u := \circ_p(x)$, then we can write both:*

$$u = x(1 + \theta_1), \qquad x = u(1 + \theta_2),$$

*where $|\theta_i| \le 2^{-p}$ for rounding to nearest, and $|\theta_i| < 2^{1-p}$ for directed rounding.*

*Proof.* This is a simple consequence of Rule 3. For rounding to nearest, we have $|u - x| \le \frac{1}{2}\mathrm{ulp}(t)$ for $t = u$ or $t = x$, hence by Rule 1 $|u - x| \le 2^{-p}$. □

**Rule 11.** *Assume $x_1, \ldots, x_n$ are $n$ floating-point numbers in precision $p$, and we compute a approximation of their product with the following sequence of operations: $u_1 = x_1, u_2 = \circ(u_1 x_2), \ldots, u_n = \circ(u_{n-1} x_n)$. If rounding away from zero, the total rounding error is bounded by $2(n-1)\mathrm{ulp}(u_n)$.*

*Proof.* We can write $u_1 x_2 = u_2(1-\theta_2), \ldots, u_{n-1}x_n = u_n(1-\theta_n)$, where $0 \le \theta_i \le 2^{1-p}$. We get $x_1 x_2 \ldots x_n = u_n(1-\theta_2)\ldots(1-\theta_n)$, which we can write $u_n(1-\theta)^{n-1}$ for some $0 \le \theta \le 2^{1-p}$ by the intermediate value theorem. Since $1 - nt \le (1-t)^n \le 1$, we get $|x_1 x_2 \ldots x_n - u_n| \le (n-1)2^{1-p}|u_n| \le 2(n-1)\mathrm{ulp}(u_n)$ by Rule 1. □

1.3. **Generic error of addition/subtraction.** We want to compute the generic error of the subtraction, the following rules apply to addition too.

$$\text{Note:} \quad \mathit{error}(u) \le k_u\,\mathrm{ulp}(u), \quad \mathit{error}(v) \le k_v\,\mathrm{ulp}(v)$$

$$\begin{aligned}\text{Note:} \quad & \mathrm{ulp}(w) = 2^{e_w-p}, \quad \mathrm{ulp}(u) = 2^{e_u-p}, \quad \mathrm{ulp}(v) = 2^{e_v-p} \;\text{ with } p \text{ the accuracy}\\ & \mathrm{ulp}(u) = 2^{d+e_w-p}, \quad \mathrm{ulp}(v) = 2^{d'+e_w-p}, \quad \text{with } \; d = e_u - e_w \;\; d' = e_v - e_w\end{aligned}$$

$$\begin{aligned}\mathit{error}(w) &\le c_w\mathrm{ulp}(w) + k_u\mathrm{ulp}(u) + k_v\mathrm{ulp}(v)\\ &\le (c_w + k_u 2^d + k_v 2^{d'})\mathrm{ulp}(w)\end{aligned}$$

$$\text{If } \; (u \ge 0 \;\text{ and }\; v \ge 0) \;\text{ or }\; (u \le 0 \;\text{ and }\; v \le 0)$$

$$\mathit{error}(w) \le (c_w + k_u + k_v)\,\mathrm{ulp}(w)$$

$$\text{Note:} \quad \text{If } \; w = N(u+v) \;\text{ Then }\; c_w = \frac{1}{2} \;\text{ else }\; c_w = 1$$

1.4. **Generic error of multiplication.** We want to compute the generic error of the multiplication. We assume here $u, v > 0$.

$$\begin{aligned} w &= o(u.v)\\ \text{Note:} \quad & \mathit{error}(u) \le k_u\,\mathrm{ulp}(u), \quad \mathit{error}(v) \le k_v\,\mathrm{ulp}(v)\end{aligned}$$

$$\begin{aligned}
error(w) &= |w - x.y| \\
&\leq |w - uv| + |uv - xy| \\
&\leq c_w \mathrm{ulp}(w) + \frac{1}{2}[|uv - uy| + |uy - xy| + |uv - xv| + |xv - xy|] \\
&\leq c_w \mathrm{ulp}(w) + \frac{u + x}{2} k_v \mathrm{ulp}(v) + \frac{v + y}{2} k_u \mathrm{ulp}(u) \\
&\leq c_w \mathrm{ulp}(w) + \frac{u(1 + c_u^+)}{2} k_v \mathrm{ulp}(v) + \frac{v(1 + c_v^+)}{2} k_u \mathrm{ulp}(u) \quad \text{[Rule 9]} \\
&\leq c_w \mathrm{ulp}(w) + (1 + c_u^+) k_v \mathrm{ulp}(u.v) + (1 + c_v^+) k_u \mathrm{ulp}(u.v) \quad \text{[Rule 4]} \\
&\leq [c_w + (1 + c_u^+) k_v + (1 + c_v^+) k_u] \mathrm{ulp}(w) \quad \text{[Rule 8]}
\end{aligned}$$

Note: If $w = N(u + v)$ Then $c_w = \frac{1}{2}$ else $c_w = 1$

1.5. **Generic error of inverse.** We want to compute the generic error of the inverse. We assume $u > 0$.

$$w = o(\frac{1}{u})$$

Note: $error(u) \leq k_u \, \mathrm{ulp}(u)$

$$\begin{aligned}
error(w) &= |w - \frac{1}{x}| \\
&\leq |w - \frac{1}{u}| + |\frac{1}{u} - \frac{1}{x}| \\
&\leq c_w \mathrm{ulp}(w) + \frac{1}{ux}|u - x| \\
&\leq c_w \mathrm{ulp}(w) + \frac{k_u}{ux} \mathrm{ulp}(u)
\end{aligned}$$

Note: $\frac{u}{c_u} \leq x$ [Rule 6]

for $u = \bigtriangledown(x)$, $c_u = 1$ else $c_u = 2$

then: $\frac{1}{x} \leq c_u \frac{1}{u}$

$$\begin{aligned}
error(w) &\leq c_w \mathrm{ulp}(w) + c_u \frac{k_u}{u^2} \mathrm{ulp}(u) \\
&\leq c_w \mathrm{ulp}(w) + 2.c_u.k_u \mathrm{ulp}(\frac{u}{u^2}) \quad \text{[Rule 4]} \\
&\leq [c_w + 2.c_u.k_u].\mathrm{ulp}(w) \quad \text{[Rule 8]}
\end{aligned}$$

Note: If $w = N(\frac{1}{u})$ Then $c_w = \frac{1}{2}$ else $c_w = 1$

1.6. **Generic error of division.** We want to compute the generic error of the division. Without loss of generality, we assume all variables are positive.

$$w = o(\frac{u}{v})$$

Note: $\quad error(u) \leq k_u \,\mathrm{ulp}(u), \quad error(v) \leq k_v \,\mathrm{ulp}(v)$

$$\begin{aligned}
error(w) &= |w - \frac{x}{y}| \\
&\leq |w - \frac{u}{v}| + |\frac{u}{v} - \frac{x}{y}| \\
&\leq c_w \mathrm{ulp}(w) + \frac{1}{vy}|uy - vx| \\
&\leq c_w \mathrm{ulp}(w) + \frac{1}{vy}[|uy - xy| + |xy - vx|] \\
&\leq c_w \mathrm{ulp}(w) + \frac{1}{vy}[y k_u \mathrm{ulp}(u) + x k_v \mathrm{ulp}(v)] \\
&\leq c_w \mathrm{ulp}(w) + \frac{k_u}{v}\mathrm{ulp}(u) + \frac{k_v x}{vy}\mathrm{ulp}(v)
\end{aligned}$$

Note: $\quad \dfrac{\mathrm{ulp}(u)}{v} \leq 2\mathrm{ulp}(\frac{u}{v}) \quad$ [Rule 4]

$2\mathrm{ulp}(\frac{u}{v}) \leq 2\mathrm{ulp}(w) \quad$ [Rule 8]

Note: $\quad x \leq c_u u$ and $\dfrac{v}{c_v} \leq y \quad$ [Rule 6]

with for $u = \triangle(x),\ c_u = 1$ else $c_u = 2$

and for $v = \bigtriangledown(y),\ c_v = 1$ else $c_v = 2$

then: $\dfrac{x}{y} \leq c_u c_v \dfrac{u}{v}$

$$\begin{aligned}
error(w) &\leq c_w \mathrm{ulp}(w) + 2.k_u \mathrm{ulp}(w) + c_u.c_v.\frac{k_v u}{vv}\mathrm{ulp}(v) \\
&\leq c_w \mathrm{ulp}(w) + 2.k_u \mathrm{ulp}(w) + 2.c_u.c_v.k_v \mathrm{ulp}(\frac{u.v}{v.v}) \quad \text{[Rule 4]} \\
&\leq [c_w + 2.k_u + 2.c_u.c_v.k_v].\mathrm{ulp}(w) \quad \text{[Rule 8]}
\end{aligned}$$

Note: $\quad$ If $w = N(\frac{u}{v})$ Then $c_w = \frac{1}{2}$ else $c_w = 1$

Note that we can obtain a slightly different result by writing $uy - vx = (uy - uv) + (uv - vx)$ instead of $(uy - xy) + (xy - vx)$.

Another result can be obtained using a relative error analysis. Assume $x = u(1 + \theta_u)$ and $y = v(1 + \theta_v)$. Then $|\frac{u}{v} - \frac{x}{y}| \leq \frac{1}{vy}|uy - uv| + \frac{1}{vy}|uv - xv| = \frac{u}{y}(|\theta_u| + |\theta_v|)$. If $v \leq y$ and $\frac{u}{v} \leq w$, this is bounded by $w(|\theta_u| + |\theta_v|)$.

1.7. **Generic error of square root.** We want to compute the generic error of the square root of a floating-point number $u$, itself an approximation to a real $x$, with $|u-x| \leq k_u \mathrm{ulp}(u)$. If $v = o(\sqrt{u})$, then:

$$\begin{aligned}
\mathit{error}(v) := |v - \sqrt{x}| &\leq |v - \sqrt{u}| + |\sqrt{u} - \sqrt{x}| \\
&\leq c_v \mathrm{ulp}(v) + \frac{1}{\sqrt{u} + \sqrt{x}} |u - x| \\
&\leq c_v \mathrm{ulp}(v) + \frac{1}{\sqrt{u} + \sqrt{x}} k_u \mathrm{ulp}(u)
\end{aligned}$$

Since by Rule 9 we have $u.c_u^- \leq x$, it follows $\frac{1}{\sqrt{x}+\sqrt{u}} \leq \frac{1}{\sqrt{u}.(1+\sqrt{c_u^-})}$:

$$\begin{aligned}
\mathit{error}(v) &\leq c_v \mathrm{ulp}(v) + \frac{1}{\sqrt{u}.(1 + \sqrt{c_u^-})} k_u \mathrm{ulp}(u) \\
&\leq c_v \mathrm{ulp}(v) + \frac{2}{1 + \sqrt{c_u^-}} k_u \mathrm{ulp}(\sqrt{u}) \qquad \text{[Rule 4]} \\
&\leq (c_v + \frac{2k_u}{1 + \sqrt{c_u^-}}) \mathrm{ulp}(v). \qquad \text{[Rule 8]}
\end{aligned}$$

If $u$ is less than $x$, we have $c_u^- = 1$ and we get the simpler formula $|v - \sqrt{x}| \leq (c_v + k_u)\mathrm{ulp}(v)$.

1.8. **Generic error of the exponential.** We want to compute the generic error of the exponential.

$$\begin{aligned}
v &= o(e^u) \\
\text{Note:} \quad & \mathit{error}(u) \leq k_u \, \mathrm{ulp}(u)
\end{aligned}$$

$$\begin{aligned}
\mathit{error}(v) &= |v - e^x| \\
&\leq |v - e^u| + |e^u - e^x| \\
&\leq c_v \mathrm{ulp}(v) + e^t |u - x| \text{ with Rolle's theorem, for } t \in [x, u] \text{ or } t \in [u, x]
\end{aligned}$$

$$\begin{aligned}
\mathit{error}(v) &\leq c_v \mathrm{ulp}(v) + c_u^* e^u k_u \mathrm{ulp}(u) \\
&\leq c_v \mathrm{ulp}(v) + 2c_u^* u k_u \mathrm{ulp}(e^u) \qquad \text{[Rule 4]} \\
&\leq (c_v + 2c_u^* u k_u) \mathrm{ulp}(v) \qquad \text{[Rule 8]} \\
&\leq (c_v + c_u^* 2^{\mathrm{EXP}(u)+1} k_u) \mathrm{ulp}(v)
\end{aligned}$$

$$\begin{array}{lll}
\text{Note:} & & u = m_u 2^{e_u} \text{ and } \mathrm{ulp}(u) = 2^{e_u - p} \text{ with } p \text{ the accuracy} \\
\text{Case} & x \leq u & c^*_u = 1 \\
\text{Case} & u \leq x & \\
& & x \leq u + k_u \mathrm{ulp}(u) \\
& & e^x \leq e^u e^{k_u \mathrm{ulp}(u)} \\
& & e^x \leq e^u e^{k_u 2^{e_u - p}} \\
& \text{then} & c^*_u = e^{k_u 2^{\mathrm{EXP}(u) - p}}
\end{array}$$

1.9. **Generic error of the logarithm.** We want to compute the generic error of the logarithm.

$$\begin{array}{rcl}
v & = & o(\log u) \\
\text{Note:} & & \mathit{error}(u) \leq k_u \,\mathrm{ulp}(u)
\end{array}$$

$$\begin{array}{rcll}
\mathit{error}(v) & = & |v - \log x| & \\
& \leq & |v - \log u| + |\log u - \log x| & \\
& \leq & c_v \mathrm{ulp}(v) + \log |\frac{x}{u}| & \\
& \leq & c_v \mathrm{ulp}(v) + \frac{|x-u|}{u} & \\
& \leq & c_v \mathrm{ulp}(v) + \frac{k_u \,\mathrm{ulp}(u)}{u} & \\
& \leq & c_v \mathrm{ulp}(v) + k_u \,\mathrm{ulp}(1) & \text{[Rule 7]}
\end{array}$$

$$\begin{array}{ll}
\text{Note:} & \mathrm{ulp}(1) = 2^{1-p}, \text{ and } \mathrm{ulp}(v) = 2^{e_v - p} \text{ with } p \text{ the accuracy} \\
& \mathrm{ulp}(1) = 2^{1 - e_v + e_v - p} = 2^{1-e_v} \mathrm{ulp}(v)
\end{array}$$

$$\begin{array}{rcl}
\mathit{error}(v) & \leq & c_v \mathrm{ulp}(v) + k_u 2^{1-e_v} \mathrm{ulp}(v) \\
& \leq & (c_v + k_u 2^{1-e_v}) \mathrm{ulp}(v)
\end{array}$$

1.10. **Ulp calculus vs relative error.** The error in ulp (ulp-error) and the relative error are related as follows.

Let $n$ be the working precision. Consider $u = \circ(x)$, then the error on $u$ is at most $\mathrm{ulp}(u) = 2^{\mathrm{EXP}(u)-n} \leq |u| \cdot 2^{1-n}$, thus the relative error is $\leq 2^{1-n}$.

Respectively, if the relative error is $\leq \delta$, then the error is at most $\delta |u| \leq \delta 2^n \mathrm{ulp}(u)$. (Going from the ulp-error to the relative error and back, we loose a factor of two.)

It is sometimes more convenient to use the relative error instead of the error in ulp (ulp-error), in particular when only multiplications or divisions are made. In that case, Higham [9] proposes the following framework: we associate to each variable the cumulated number

$k$ of roundings that were made. The $i$th rounding introduces a relative error of $\delta_i$, with $|\delta_i| \leq 2^{1-n}$, i.e. the computed result is $1+\delta_i$ times the exact result. Hence $k$ successive roundings give a error factor of $(1+\delta_1)(1+\delta_2)\cdots(1+\delta_k)$, which is between $(1-\varepsilon)^k$ and $(1+\varepsilon)^k$ with $\varepsilon = 2^{1-n}$. In particular, if all roundings are away, the final relative error is at most $k\varepsilon = k \cdot 2^{1-n}$, thus at most $2k$ ulps.

**Lemma 1.** *If a value if computed by $k$ successive multiplications or divisions, each with rounding away from zero, and precision $n$, then the final error is bounded by $2k$ ulps.*

If the rounding are not away from zero, the following lemma is still useful [8]:

**Lemma 2.** *Let $\delta_1, \ldots, \delta_n$ be $n$ real values such that $|\delta_i| \leq \epsilon$, for $n\epsilon < 1$. Then we can write $\prod_{i=1}^{n}(1+\delta_i) = 1+\theta$ with*

$$|\theta| \leq \frac{n\epsilon}{1-n\epsilon}.$$

*Proof.* The maximum values of $\theta$ are obtained when all the $\delta_i$ are $\epsilon$, or all are $-\epsilon$, thus it suffices to prove

$$(1+\epsilon)^n \leq 1 + \frac{n\epsilon}{1-n\epsilon} = \frac{1}{1-n\epsilon} \quad \text{and} \quad (1-\epsilon)^n \geq 1 - \frac{n\epsilon}{1-n\epsilon} = \frac{1-2n\epsilon}{1-n\epsilon}.$$

For the first inequality, we have $(1+\epsilon)^n = e^{n\log(1+\epsilon)}$, and since $\log(1+x) \leq x$, it follows $(1+\epsilon)^n \leq e^{n\epsilon} = \sum_{k\geq 0}\frac{(n\epsilon)^k}{k!} \leq \sum_{k\geq 0}(n\epsilon)^k = \frac{1}{1-n\epsilon}$.

For the second inequality, we first prove by induction that $(1-\epsilon)^n \geq 1-n\epsilon$ for integer $n \geq 0$. It follows $(1-\epsilon)^n(1-n\epsilon) \geq (1-n\epsilon)^2 \geq 1-2n\epsilon$, which concludes the proof. □

## 2. Low level functions

### 2.1. The mpfr_add function.

```
mpfr_add (A, B, C, rnd)
/* on suppose B et C de me^me signe, et EXP(B) >= EXP(C) */

0. d = EXP(B) - EXP(C) /* d >= 0 par hypothe`se */
1. Soient B1 les prec(A) premiers bits de B, et B0 le reste
          C1 les bits de C correspondant a` B1, C0 le reste
/* B0, C1, C0 peuvent e^tre vides, mais pas B1 */

              <----------- A ---------->
              <----------- B1 --------->$<------ B0 ----->
                <---------- C1 ------->$<------------ C0 ----------->

2. A <- B1 + (C1 >> d)
3. q <- compute_carry (B0, C0, rnd)
4. A <- A + q
```

### 2.2. The mpfr_cmp2 function.

This function computes the exponent shift when subtracting $c > 0$ from $b \geq c$. In other terms, if $\mathrm{EXP}(x) := \lfloor \frac{\log x}{\log 2} \rfloor$, it returns $\mathrm{EXP}(b) - \mathrm{EXP}(b-c)$.

This function admits the following specification in terms of the binary representation of the mantissa of $b$ and $c$: if $b = u10^n r$ and $c = u01^n s$, where $u$ is the longest common prefix to $b$ and $c$, and $(r, s)$ do not start with $(0, 1)$, then mpfr_cmp2$(b, c)$ returns $|u| + n$ if $r \geq s$, and $|u| + n + 1$ otherwise, where $|u|$ is the number of bits of $u$.

As it is not very efficient to compare $b$ and $c$ bit-per-bit, we propose the following algorithm, which compares $b$ and $c$ word-per-word. Here $b[n]$ represents the $n$th word from the mantissa of $b$, starting from the most significant word $b[0]$, which has its most significant bit set. The values $c[n]$ represent the words of $c$, after a possible shift if the exponent of $c$ is smaller than that of $b$.

```
n = 0; res = 0;
while (b[n] == c[n])
   n++;
   res += BITS_PER_MP_LIMB;

/* now b[n] > c[n] and the first res bits coincide */

dif = b[n] - c[n];
while (dif == 1)
   n++;
   dif = (dif << BITS_PER_MP_LIMB) + b[n] - c[n];
   res += BITS_PER_MP_LIMB;

/* now dif > 1 */

res += BITS_PER_MP_LIMB - number_of_bits(dif);

if (!is_power_of_two(dif))
   return res;

/* otherwise result is res + (low(b) < low(c)) */
do
   n++;
while (b[n] == c[n]);
return res + (b[n] < c[n]);
```

2.3. **The mpfr_sub function.** The algorithm used is as follows, where $w$ denotes the number of bits per word. We assume that $a$, $b$ and $c$ denote different variables (if $a := b$ or $a := c$, we have first to copy $b$ or $c$), and that the rounding mode is either $\mathcal{N}$ (nearest), $\mathcal{Z}$ (towards zero), or $\infty$ (away from zero).

```
Algorithm mpfr_sub.
Input: $b$, $c$ of opposite sign with $b > c > 0$, a rounding mode $\circ \in \{\mathcal{N}, \mathcal{Z}, \infty\}$
Side effect: store in $a$ the value of $\circ(b - c)$
Output: 0 if $\circ(b - c) = b - c$, 1 if $\circ(b - c) > b - c$, and $-1$ if $\circ(b - c) < b - c$
$\mathtt{an} \leftarrow \lceil \frac{\mathrm{prec}(a)}{w} \rceil$, $\mathtt{bn} \leftarrow \lceil \frac{\mathrm{prec}(b)}{w} \rceil$, $\mathtt{cn} \leftarrow \lceil \frac{\mathrm{prec}(c)}{w} \rceil$
$\mathtt{cancel} \leftarrow \mathtt{mpfr\_cmp2}(b, c)$;   $\mathtt{diff\_exp} \leftarrow \mathrm{EXP}(b) - \mathrm{EXP}(c)$
$\mathtt{shift_b} \leftarrow (-\mathtt{cancel}) \bmod w$;   $\mathtt{cancel_b} \leftarrow (\mathtt{cancel} + \mathtt{shift_b})/w$
**if** $\mathtt{shift_b} > 0$ **then** $\mathtt{b}[0 \ldots \mathtt{bn}] \leftarrow \mathtt{mpn\_rshift}(\mathtt{b}[0 \ldots \mathtt{bn} - 1], \mathtt{shift_b})$; $\mathtt{bn} \leftarrow$
```

```
$\mathtt{bn}+1$
$\mathtt{shift_c} \leftarrow (\mathtt{diff\_exp} - \mathtt{cancel}) \bmod w$;    $\mathtt{cancel_c} \leftarrow (\mathtt{cancel} + \mathtt{shift_c} - \mathtt{diff\_exp})/w$
if $\mathtt{shift_c} > 0$ then $\mathtt{c}[0 \ldots \mathtt{cn}] \leftarrow \mathtt{mpn\_rshift}(\mathtt{c}[0 \ldots \mathtt{cn}-1], \mathtt{shift_c})$; $\mathtt{cn} \leftarrow \mathtt{cn}+1$
$\mathrm{EXP}(a) \leftarrow \mathrm{EXP}(b) - \mathtt{cancel}$;    $\mathrm{sign}(a) \leftarrow \mathrm{sign}(b)$
$a[0 \ldots \mathtt{an}-1] \leftarrow b[\mathtt{bn} - \mathtt{cancel_b} - \mathtt{an} \ldots \mathtt{bn} - \mathtt{cancel_b} - 1]$
$a[0 \ldots \mathtt{an}-1] \leftarrow a[0 \ldots \mathtt{an}-1] - c[\mathtt{cn} - \mathtt{cancel_c} - \mathtt{an} \ldots \mathtt{cn} - \mathtt{cancel_c} - 1]$
$\mathtt{sh} \leftarrow \mathtt{an} \cdot w - \mathrm{prec}(a)$;    $r \leftarrow a[0] \bmod 2^{\mathtt{sh}}$;    $a[0] \leftarrow a[0] - r$
if $\circ = \mathcal{N}$ and $\mathtt{sh} > 0$ then
    if $r > 2^{\mathtt{sh}-1}$ then $a \leftarrow a + \mathrm{ulp}(a)$; return 1 elif $0 < r < 2^{\mathtt{sh}-1}$ then return $-1$
elif $\circ \in \{\mathcal{Z}, \infty\}$ and $r > 0$ then
    if $\circ = \mathcal{Z}$ return $-1$ else $a \leftarrow a + \mathrm{ulp}(a)$; return 1
$\mathtt{bl} \leftarrow \mathtt{bn} - \mathtt{an} - \mathtt{cancel_b}$
$\mathtt{cl} \leftarrow \mathtt{cn} - \mathtt{an} - \mathtt{cancel_c}$
for $k = 0$ while $\mathtt{bl} > 0$ and $\mathtt{cl} > 0$ do
    $\mathtt{bl} \leftarrow \mathtt{bl} - 1$; $\mathtt{bp} \leftarrow b[\mathtt{bl}]$
    $\mathtt{cl} \leftarrow \mathtt{cl} - 1$; $\mathtt{cp} \leftarrow c[\mathtt{cl}]$
    if $\circ = \mathcal{N}$ and $k = 0$ and $\mathtt{sh} = 0$ then
        if $\mathtt{cp} \geq 2^{w-1}$ then return $-1$
        $r \leftarrow \mathtt{bp} - \mathtt{cp}$;    $\mathtt{cp} \leftarrow \mathtt{cp} + 2^{w-1}$
    if $\mathtt{bp} < \mathtt{cp}$ then
        if $\circ = \mathcal{Z}$ then $a \leftarrow a - \mathrm{ulp}(a)$;    if $\circ = \infty$ then return 1 else return $-1$
    if $\mathtt{bp} > \mathtt{cp}$ then
        if $\circ = \mathcal{Z}$ then return $-1$ else $a \leftarrow a + \mathrm{ulp}(a)$; return 1
if $\circ = \mathcal{N}$ and $r > 0$ then
    if $a[0] \operatorname{div} 2^{\mathtt{sh}}$ is odd then $a \leftarrow a + \mathrm{ulp}(a)$; return 1 else return $-1$
Return 0.
```

where $b[i]$ and $c[i]$ is meant as 0 for negative $i$, and $c[i]$ is meant as 0 for $i \geq \mathtt{cn}$ ($\mathtt{cancel_b} \geq 0$, but $\mathtt{cancel_c}$ may be negative).

## 2.4. **The `mpfr_mul` function.**

`mpfr_mul` uses two algorithms: if the precision of the operands is small enough, a plain multiplication using `mpn_mul` is used (there is no error, except in the final rounding); otherwise it uses `mpfr_mulhigh_n`.

In this case, it trunks the two operands to $m$ limbs: $1/2 \leq b < 1$ and $1/2 \leq c < 1$, $b = bh + bl$ and $c = ch + c$ $(B = 2^{32} or 2^{64})$. The error comes from:

- Truncation: $\leq bl.ch + bh.cl + bl.cl \leq bl + cl \leq 2B^{-m}$
- Mulder: Assuming $error(Mulder(n)) \leq error(mulhigh\_basecase(n))$,

$$\begin{aligned} error(mulhigh(n) &\leq (n-1)(B-1)^2 B^{-n-2} + \cdots + 1(B-1)^2 B^{-2n} \\ &= \sum_{i=1}^{n-1} (n-i)(B-1)^2 B^{-n-1-i} = (B-1)^2 B^{-n-1} \sum_{i=1}^{n-1} B^{-i} \\ &= (b-1)^2 B^{-n-1} \frac{B^{1-n} - n + nB - B}{(1-B)^2} \leq nB^{-n}. \end{aligned}$$

Total error: $\leq (m+2)B^{-m}$.

2.5. **The `mpfr_div` function.** The goals of the code of the `mpfr_div` function include the fact that the complexity should, while preserving correct rounding, depend on the precision required on the result rather than on the precision given on the operands.

Let $u$ be the dividend, $v$ the divisor, and $p$ the target precision for the quotient. We denote by $q$ the real quotient $u/v$, with infinite precision, and $n \geq p$ the working precision. The idea — as in the square root algorithm below — is to use GMP's integer division: divide the most $2n$ or $2n-1$ significant bits from $u$ by the most $n$ significant bits from $v$ will give a good approximation of the quotient's integer significand. The main difficulties arise when $u$ and $v$ have a larger precision than $2n$ and $n$ respectively, since we have to truncate them. We distinguish two cases: whether the divisor is truncated or not.

2.5.1. *Full divisor.* This is the easy case. Write $u = u_1 + u_0$ where $u_0$ is the truncated part, and $v = v_1$. Without loss of generality we can assume that $\mathrm{ulp}(u_1) = \mathrm{ulp}(v_1) = 1$, thus $u_1$ and $v_1$ are integers, and $0 \leq u_0 < 1$. Since $v_1$ has $n$ significant bits, we have $2^{n-1} \leq v_1 < 2^n$. (We normalize $u$ so that the integer quotient gives exactly $n$ bits; this is easy by comparing the most significant bits of $u$ and $v$, thus $2^{2n-2} \leq u_1 < 2^{2n}$.) The integer division of $u_1$ by $v_1$ yields $q_1$ and $r$ such that $u_1 = q_1 v_1 + r$, with $0 \leq r < v_1$, and $q_1$ having exactly $n$ bits. In that case we have

$$q_1 \leq q = \frac{u}{v} < q_1 + 1.$$

Indeed, $q = \frac{u}{v} \geq \frac{u_1}{v_1} = \frac{q_1 v_1 + r}{v_1}$, and $q \leq \frac{u_1 + u_0}{v_1} \leq q_1 + \frac{r + u_0}{v_1} < q_1 + 1$, since $r + u_0 < r + 1 \leq v_1$.

2.5.2. *Truncated divisor.* This is the hard case. Write $u = u_1 + u_0$, and $v = v_1 + v_0$, where $0 \leq u_0, v_0 < 1$ with the same conventions as above. We prove in that case that:

$$q_1 - 2 < q = \frac{u}{v} < q_1 + 1. \tag{1}$$

The upper bound holds as above. For the lower bound, we have $u - (q_1 - 2)v > u_1 - (q_1 - 2)(v_1 + 1) \geq q_1 v_1 - (q_1 - 2)(v_1 + 1) = 2(v_1 + 1) - q_1 \geq 2^n - q_1 > 0$. This lower bound is the best possible, since $q_1 - 1$ would be wrong; indeed, consider $n = 3$, $v_1 = 4$, $v_0 = 7/8$, $u = 24$: this gives $q_1 = 6$, but $u/v = 64/13 < q_1 - 1 = 5$.

As a consequence of Eq. (1), if the open interval $(q_1 - 2, q_1 + 1)$ contains no rounding boundary for the target precision, we can deduce the correct rounding of $u/v$ just from the value of $q_1$. In other words, for directed rounding, the two only "bad cases" are when the binary representation of $q_1$ ends with $\underbrace{0000}_{n-p}$ or $\underbrace{0001}_{n-p}$. We even can decide if rounding is correct, since when $q_1$ ends with 0010, the exact value cannot end with 0000, and similarly when $q_1$ ends with 1111. Hence if $n = p + k$, i.e. if we use $k$ extra bits with respect to the target precision $p$, the failure probability is $2^{1-k}$.

2.5.3. *Avoiding Ziv's strategy.* In the failure case ($q_1$ ending with $000\ldots000x$ with directed rounding, or $100\ldots000x$ with rounding to nearest), we could try again with a larger working precision $p$. However, we then need to perform a second division, and we are not sure this new computation will enable us to conclude. In fact, we can conclude directly. Recall that $u_1 = q_1 v_1 + r$. Thus $u = q_1 v + (r + u_0 - q_1 v_0)$. We have to decide which of the following five cases holds: (a) $q_1 - 2 < q < q_1 - 1$, (b) $q = q_1 - 1$, (c) $q_1 - 1 < q < q_1$, (d) $q = q_1$, (e) $q_1 < q < q_1 + 1$.

```
$s \leftarrow q_1 v_0$
**if** $s < r + u_0$ **then** $q \in (q_1, q_1 + 1)$
**elif** $s = r + u_0$ **then** $q = q_1$
**else**
   $t \leftarrow s - (r + u_0)$
   **if** $t < v$ **then** $q \in (q_1 - 1, q_1)$
**elif** $t = v$ **then** $q = q_1 - 1$
**else** $q \in (q_1 - 2, q_1 - 1)$
```

2.6. **The `mpfr_sqrt` function.** The `mpfr_sqrt` implementation uses the `mpn_sqrtrem` function from GMP's `mpn` level: given a positive integer $m$, it computes $s$ and $r$ such that $m = s^2 + r$ with $s^2 \le m < (s+1)^2$, or equivalently $0 \le r \le 2s$. In other words, $s$ is the integer square root of $m$, rounded towards zero.

The idea is to multiply the input significand by some power of two, in order to obtain an integer significand $m$ whose integer square root $s$ will have exactly $p$ bits, where $p$ is the target precision. This is easy: $m$ should have either $2p$ or $2p-1$ bits. For directed rounding, we then know that the result significand will be either $s$ or $s+1$, depending on the square root remainder $r$ being zero or not.

```
Algorithm FPSqrt.
Input: $x = m \cdot 2^e$, a target precision $p$, a rounding mode $\circ$
Output: $y = \circ_p(\sqrt{x})$
If $e$ is odd, $(m', f) \leftarrow (2m, e-1)$, else $(m', f) \leftarrow (m, e)$
Write $m' := m_1 2^{2k} + m_0$, $m_1$ having $2p$ or $2p-1$ bits, $0 \le m_0 < 2^{2k}$
$(s, r) \leftarrow$ SqrtRem$(m_1)$
If round to zero or down or $r = m_0 = 0$, return $s \cdot 2^{k+f/2}$
else return $(s+1) \cdot 2^{k+f/2}$.
```

In case the input has more than $2p$ or $2p-1$ bits, it needs to be truncated, but the crucial point is that that truncated part will not overlap with the remainder $r$ from the integer square root, so the *sticky bit* is simply zero when both parts are zero.

For rounding to nearest, the simplest way is to ask $p+1$ bits for the integer square root — thus $m$ has now $2p+1$ or $2p+2$ bits. In such a way, we directly get the rounding bit, which is the parity bit of $s$, and the sticky bit is determined as above. Otherwise, we have to compare the value of the whole remainder, i.e. $r$ plus the possible truncated input, with $s + 1/4$, since $(s+1/2)^2 = s^2 + s + 1/4$. Note that equality can occur — i.e. the "nearest even rounding rule" — only when the input has at least $2p+1$ bits; in particular it can not happen in the common case when input and output have the same precision.

## 3. High level functions

3.1. **The cosine function.** To evaluate $\cos x$ with a target precision of $n$ bits, we use the following algorithm with working precision $m$:

```
$k \leftarrow \lfloor \sqrt{n/2} \rfloor$
$r \leftarrow x^2$ rounded up
$r \leftarrow r/2^{2k}$
$s \leftarrow 1, t \leftarrow 1$
**for** $l$ **from** 1 **while** EXP$(t) \ge -m$
```

```
        $t \leftarrow t \cdot r$ rounded up
        $t \leftarrow \frac{t}{(2l-1)(2l)}$ rounded up
        $s \leftarrow s + (-1)^l t$ rounded down
    **do** $k$ times
        $s \leftarrow 2s^2$ rounded up
        $s \leftarrow s - 1$
    return $s$
```

The error on $r$ after $r \leftarrow x^2$ is at most $1\mathrm{ulp}(r)$ and remains $1\mathrm{ulp}(r)$ after $r \leftarrow r/2^{2k}$ since that division is just an exponent shift. By induction, the error on $t$ after step $l$ of the for-loop is at most $3l\mathrm{ulp}(t)$. Hence as long as $3l\mathrm{ulp}(t)$ remains less than $\leq 2^{-m}$ during that loop (this is possible as soon as $r < 1/\sqrt{2}$) and the loop goes to $l_0$, the error on $s$ after the for-loop is at most $2l_0 2^{-m}$ (for $|r| < 1$, it is easy to check that $s$ will remain in the interval $[\frac{1}{2}, 1[$, thus $\mathrm{ulp}(s) = 2^{-m}$). (An additional $2^{-m}$ term represents the truncation error, but for $l = 1$ the value of $t$ is exact, giving $(2l_0 - 1) + 1 = 2l_0$.)

Denoting by $\epsilon_i$ the maximal error on $s$ after the $i$th step in the do-loop, we have $\epsilon_0 = 2l_0 2^{-m}$ and $\epsilon_{k+1} \leq 4\epsilon_k + 2^{-m}$, giving $\epsilon_k \leq (2l_0 + 1/3)2^{2k-m}$.

3.2. **The sine function.** The sine function is computed from the cosine, with a working precision of $m$ bits:

```
    $c \leftarrow \cos x$ rounded away
    $t \leftarrow c^2$ rounded away
    $u \leftarrow 1 - t$ rounded to zero
    $s \leftarrow \mathrm{sign}(x)\sqrt{u}$ rounded to zero
```

This algorithm ensures that the approximation $s$ is between zero and $\sin x$.

Since all variables are in $[-1, 1]$, where $\mathrm{ulp}() \leq 2^{-m}$, all absolute errors are less than $2^{-m}$. We denote by $\epsilon_i$ a generic error with $0 \leq \epsilon_i < 2^{-m}$. We have $c = \cos x + \epsilon_1$; $t = c^2 + \epsilon_2 = \cos^2 x + 4\epsilon_3$; $u = 1 - t - \epsilon_4 = 1 - \cos^2 x - 5\epsilon_5$; $|s| = \sqrt{u} - \epsilon_6 = \sqrt{1 - \cos^2 x - 5\epsilon_5} - \epsilon_6 \geq |\sin x| - \frac{5\epsilon_5}{2|s|} + \epsilon_6$ (by Rolle's theorem, $|\sqrt{u} - \sqrt{u'}| \leq \frac{1}{2\sqrt{v}}|u - u'|$ for $v \in [u, u']$, we apply it here with $u = 1 - \cos^2 x - 5\epsilon_5$, $u' = 1 - \cos^2 x$.)

Therefore, if $2^{e-1} \leq |s| < 2^e$, the absolute error on $s$ is bounded by $2^{-m}(\frac{5}{2}2^{1-e} + 1) \leq 2^{3-m-e}$.

3.2.1. *An asymptotically fast algorithm for sin and cos.* We extend here the algorithm proposed by Brent for the exponential function to the simultaneous computation of sin and cos. The idea is the following. We first reduce the input $x$ to the range $0 < x < 1/2$. Then we decompose $x$ as follows:

$$x = \sum_{i=1}^{k} \frac{r_i}{2^{2^i}},$$

where $r_i$ is an integer, $0 \leq r_i < 2^{2^{i-1}}$.

We define $x_j = \sum_{i=j}^{k} \frac{r_i}{2^{2^i}}$; then $x = x_1$, and we can write $x_j = \frac{r_j}{2^{2^j}} + x_{j+1}$. Thus with $S_j := \sin \frac{r_j}{2^{2^j}}$ and $C_j := \cos \frac{r_j}{2^{2^j}}$:

$$\sin x_j = S_j \cos x_{j+1} + C_j \sin x_{j+1}, \quad \cos x_j = C_j \cos x_{j+1} - S_j \sin x_{j+1}.$$

The $2k$ values $S_j$ and $C_j$ can be computed by a binary splitting algorithm, each one in $O(M(n)\log n)$. Then each pair $(\sin x_j, \cos x_j)$ can be computed from $(\sin x_{j+1}, \sin x_{j+1})$ with four multiplies and two additions or subtractions.
Error analysis. We use here Higham's method. We assume that the values of $S_j$ and $C_j$ are approximated up to a multiplicative factor of the form $(1+u)^3$, where $|u| \leq 2^{-p}$, $p \geq 4$ being the working precision. We also assume that $\cos x_{j+1}$ and $\sin x_{j+1}$ are approximated with a factor of the form $(1+u)^{k_j}$. With rounding to nearest, the values of $S_j \cos x_{j+1}$, $C_j \sin x_{j+1}$, $C_j \cos x_{j+1}$ and $S_j \sin x_{j+1}$ are thus approximated with a factor $(1+u)^{k_j+4}$. The value of $\sin x_j$ is approximated with a factor $(1+u)^{k_j+5}$ since there all terms are nonnegative.

We now analyze the effect of the cancellation in $C_j \cos x_{j+1} - S_j \sin x_{j+1}$. We have $\frac{r_j}{2^{2^j}} < 2^{-2^{j-1}}$, and for simplicity we define $l := 2^{j-1}$; thus $0 \leq S_j \leq 2^{-l}$, and $1 - 2^{-2l-1} \leq C_j \leq 1$. Similarly we have $x_{j+1} < 2^{-2l}$, thus $0 \leq \sin x_{j+1} \leq 2^{-2l}$, and $1 - 2^{-4l-1} \leq \cos x_{j+1} \leq 1$. The error is multiplied by a maximal ratio of

$$\frac{C_j \cos x_{j+1} + S_j \sin x_{j+1}}{C_j \cos x_{j+1} - S_j \sin x_{j+1}} \leq \frac{1 + 2^{-l} \cdot 2^{-2l}}{(1-2^{-2l-1})(1-2^{-4l-1}) - 2^{-l} \cdot 2^{-2l}},$$

which we can bound by

$$\frac{1+2^{-3l}}{1-2^{-2l}} \leq \frac{1}{(1-2^{-2l})(1-2^{-3l})} \leq \frac{1}{1-2^{-2l+1}}.$$

The product of all those factors for $j \geq 1$ is bounded by 3 (remember $l := 2^{j-1}$).

In summary, the maximal error is of the form $3[(1+u)^{5k} - 1]$, where $2^{2^{k-1}} < p \leq 2^{2^k}$. For $p \geq 4$, $5k \cdot 2^{-p}$ is bounded by $5/16$, and $(1+2^{-p})^{5k} - 1 \leq e^{5k \cdot 2^{-p}} - 1 \leq \frac{6}{5} \cdot 5k \cdot 2^{-p} = 6k \cdot 2^{-p}$. Thus the final relative error bound is $18k \cdot 2^{-p}$. Since $k \leq 6$ for $p \leq 2^{64}$, this gives a uniform relative error bound of $2^{-p+7}$.

## 3.3. The tangent function.

The tangent function is computed from the cosine, using $\tan x = \mathrm{sign}(x)\sqrt{\frac{1}{\cos^2 x} - 1}$, with a working precision of $m$ bits:

```
$c \leftarrow \cos x$ rounded down
$t \leftarrow c^2$ rounded down
$v \leftarrow 1/t$ rounded up
$u \leftarrow v - 1$ rounded up
$s \leftarrow \mathrm{sign}(x)\sqrt{u}$ rounded away from 0
```

The absolute error on $c$ is at most $\mathrm{ulp}(c)$. Hence the error on $t$ is at most $\mathrm{ulp}(t) + 2c\mathrm{ulp}(c) \leq 5\mathrm{ulp}(t)$, that on $v$ is at most $\mathrm{ulp}(v) + 5\mathrm{ulp}(t)/t^2 \leq \mathrm{ulp}(v) + 10\mathrm{ulp}(1/t) \leq 11\mathrm{ulp}(v)$, that on $u$ is at most $\mathrm{ulp}(u) + 11\mathrm{ulp}(v) \leq (1+11\cdot 2^e)\mathrm{ulp}(u)$ where $e$ is the exponent difference between $v$ and $u$. The final error on $s$ is $\leq \mathrm{ulp}(s) + (1+11\cdot 2^e)\mathrm{ulp}(u)/2/\sqrt{u} \leq \mathrm{ulp}(s) + (1+11\cdot 2^e)\mathrm{ulp}(u/\sqrt{u}) \leq (2+11\cdot 2^e)\mathrm{ulp}(s)$.

## 3.4. The exponential function.

The `mpfr_exp` function implements three different algorithms. For very large precision, it uses a $\mathcal{O}(M(n)\log^2 n)$ algorithm based on binary splitting, based on the generic implementation for hypergeometric functions in the file `generic.c` (see [10]). This algorithm is used only for precision greater than for example 10000 bits on an Athlon.

For smaller precisions, it uses Brent's method; if $r = (x - n\log 2)/2^k$ where $0 \le r < \log 2$, then

$$\exp(x) = 2^n \cdot \exp(r)^{2^k}$$

and $\exp(r)$ is computed using the Taylor expansion:

$$\exp(r) = 1 + r + \frac{r^2}{2!} + \frac{r^3}{3!} + \cdots$$

As $r < 2^{-k}$, if the target precision is $n$ bits, then only about $l = n/k$ terms of the Taylor expansion are needed. This method thus requires the evaluation of the Taylor series to order $n/k$, and $k$ squares to compute $\exp(r)^{2^k}$. If the Taylor series is evaluated using a naive way, the optimal value of $k$ is about $n^{1/2}$, giving a complexity of $\mathcal{O}(n^{1/2}M(n))$. This is what is implemented in `mpfr_exp2_aux`.

If we use a baby step/giant step approach, the Taylor series can be evaluated in $\mathcal{O}(l^{1/2})$ operations, thus the evaluation requires $(n/k)^{1/2} + k$ multiplications, and the optimal $k$ is now about $n^{1/3}$, giving a total complexity of $\mathcal{O}(n^{1/3}M(n))$. This is implemented in the function `mpfr_exp2_aux2`.

3.5. **The error function.** Let $n$ be the target precision, and $x$ be the input value. For $|x| \ge \sqrt{n\log 2}$, we have $|\mathrm{erf}\, x| = 1$ or $1^-$ according to the rounding mode. Otherwise we use the Taylor expansion.

3.5.1. *Taylor expansion.*

$$\mathrm{erf}\, z = \frac{2}{\sqrt{\pi}} \sum_{k=0}^{\infty} \frac{(-1)^k}{k!(2k+1)} z^{2k+1}$$

```
erf_0($z, n$), assumes $z^2 \le n/e$
working precision is $m$
$y \leftarrow \circ(z^2)$ [rounded up]
$s \leftarrow 1$
$t \leftarrow 1$
**for** $k$ **from** 1 **do**
    $t \leftarrow \circ(yt)$ [rounded up]
    $t \leftarrow \circ(t/k)$ [rounded up]
    $u \leftarrow \circ(\frac{t}{2k+1})$ [rounded up]
    $s \leftarrow \circ(s + (-1)^k u)$ [nearest]
    **if** $\mathrm{EXP}(u) < \mathrm{EXP}(s) - m$ and $k \ge z^2$ **then** break
$r \leftarrow 2 \circ (zs)$ [rounded up]
$p \leftarrow \circ(\pi)$ [rounded down]
$p \leftarrow \circ(\sqrt{p})$ [rounded down]
$r \leftarrow \circ(r/p)$ [nearest]
```

Let $\varepsilon_k$ be the ulp-error on $t$ (denoted $t_k$) after the loop with index $k$. According to Lemma 1, since $t_k$ is computed after $2k$ roundings ($t_0 = 1$ is exact), we have $\varepsilon_k \le 4k$.

The error on $u$ at loop $k$ is thus at most $1 + 2\varepsilon_k \le 1 + 8k$.

Let $\sigma_k$ and $\nu_k$ be the exponent shifts between the new value of $s$ at step $k$ and respectively the old value of $s$, and $u$. Writing $s_k$ and $u_k$ for the values of $s$ and $u$ at the end of step $k$, we have $\sigma_k := \mathrm{EXP}(s_{k-1}) - \mathrm{EXP}(s_k)$ and $\nu_k := \mathrm{EXP}(u_k) - \mathrm{EXP}(s_k)$. The ulp-error $\tau_k$ on $s_k$ satisfies $\tau_k \leq \frac{1}{2} + \tau_{k-1} 2^{\sigma_k} + (1+8k) 2^{\nu_k}$.

The halting condition $k \geq z^2$ ensures that $u_j \leq u_{j-1}$ for $j \geq k$, thus the series $\sum_{j=k}^{\infty} u_j$ is an alternating series, and the truncated part is bounded by its first term $|u_k| < \mathrm{ulp}(s_k)$. So the ulp-error between $s_k$ and $\sum_{k=0}^{\infty} \frac{(-1)^k z^2}{k!(2k+1)}$ is bounded by $1 + \tau_k$.

Now the error after $r \leftarrow 2 \circ (zs)$ is bounded by $1 + 2(1+\tau_k) = 2\tau_k + 3$. That on $p$ after $p \leftarrow \circ(\pi)$ is 1 ulp, and after $p \leftarrow \circ(\sqrt{p})$ we get 2 ulps (since $p \leftarrow \circ(\pi)$ was rounded down).

The final error on $r$ is thus at most $1 + 2(2\tau_k + 3) + 4 = 4\tau_k + 11$ (since $r$ is rounded up and $p$ is rounded down).

3.5.2. *Very large arguments.* Since $\operatorname{erfc} x \leq \frac{1}{\sqrt{\pi} x e^{x^2}}$, we have for $x^2 \geq n \log 2$ (which implies $x \geq 1$) that $\operatorname{erfc} x \leq 2^{-n}$, thus $\operatorname{erf} x = 1$ or $\operatorname{erf} x = 1 - 2^{-n}$ according to the rounding mode.

3.6. **The hyperbolic cosine function.** The `mpfr_cosh` ($\cosh x$) function implements the hyperbolic cosine as :

$$\cosh x = \frac{1}{2}\left(e^x + \frac{1}{e^x}\right).$$

The algorithm used for the calculation of the hyperbolic cosine is as follows[1]:

$$u \leftarrow o(e^x)$$

(2) $$v \leftarrow o(u^{-1})$$

(3) $$w \leftarrow o(u+v)$$

(4) $$s \leftarrow \frac{1}{2} w$$

(5)

Now, we have to bound the rounding error for each step of this algorithm. First, let consider the parity of hyperbolic cosine $(\cosh(-x) = \cosh(x))$ : the problem is reduced to calculate $\cosh x$ with $x \geq 0$. We can deduce $e^x \geq 1$ and $0 \leq e^{-x} \leq 1$.

[1] $o()$ represent the rounding error and $error(u)$ the error associate with the calculation of $u$

error($u$)
$u \leftarrow o(e^x)$
$-\infty$ $(\bullet)$

$$|u - e^x| \leq \mathrm{ulp}(u)$$

error($v$)
$v \leftarrow o(u^{-1})$
$+\infty$ $(\bullet\bullet)$

$$\begin{aligned}
& |v - e^{-x}| \\
\leq\ & |v - u^{-1}| + |u^{-1} - e^{-x}| \\
\leq\ & \mathrm{ulp}(v) + \frac{1}{u \cdot e^x}|u - e^x| \\
\leq\ & \mathrm{ulp}(v) + \frac{1}{u^2}\mathrm{ulp}(u) \ \ (\star) \\
\leq\ & \mathrm{ulp}(v) + 2\mathrm{ulp}(\frac{1}{u}) \ \ (\star\star) \\
\leq\ & 3\,\mathrm{ulp}(v) \ \ (\star\star\star)
\end{aligned}$$

$(\star)$
With $\frac{1}{e^x} \leq \frac{1}{u}$,
for that we must have $u \leq e^x$,
it is possible with a rounding of
$u$ to $-\infty$ $(\bullet)$
$(\star\star)$
From inequation [Rule 4],

$$a \cdot \mathrm{ulp}(b) \leq 2 \cdot \mathrm{ulp}(a \cdot b)$$

if $a = \frac{1}{u^2}$, $b = u$ then

$$\frac{1}{u^2}\mathrm{ulp}(u) \leq 2\mathrm{ulp}(\frac{1}{u})$$

$(\star\star\star)$
If $\mathrm{ulp}(\frac{1}{u}) \leq ulp(v)$,
it is possible with a rounding of
$v$ to $+\infty$ $(\bullet)$

error($w$)
$w \leftarrow o(u + v)$

$$\begin{aligned}
& |w - (e^x + e^{-x})| \\
\leq\ & |w - (u + v)| + |u - e^x| + |v - e^{-x}| \\
\leq\ & \mathrm{ulp}(w) + \mathrm{ulp}(u) + 3\mathrm{ulp}(v) \\
\leq\ & \mathrm{ulp}(w) + 4\mathrm{ulp}(u) \ \ (\star) \\
\leq\ & 5\mathrm{ulp}(w) \ \ (\star\star)
\end{aligned}$$

$(\star)$
With $v \leq 1 \leq u$
then $\mathrm{ulp}(v) \leq \mathrm{ulp}(u)$
$(\star\star)$
With $u \leq w$
then $\mathrm{ulp}(u) \leq \mathrm{ulp}(w)$

error($s$)
$s \leftarrow o(\frac{w}{2})$

$$\begin{aligned}
\mathrm{error}(s) &= \mathrm{error}(w) \\
&\leq 5\mathrm{ulp}(s)
\end{aligned}$$

That shows the rounding error on the calculation of $\cosh x$ can be bound by 5 ulp on the result. So, to calculate the size of intermediary variables, we have to add, at least, $\lceil \log_2 5 \rceil = 3$ bits the wanted precision.

3.7. **The inverse hyperbolic cosine function.** The `mpfr_acosh` function implements the inverse hyperbolic cosine as $\mathrm{acosh} x = \log(\sqrt{x^2 - 1} + x)$, using the following algorithm:

```
$q \leftarrow \circ(x^2)$ [down]
$r \leftarrow \circ(q - 1)$ [down]
$s \leftarrow \circ(\sqrt{r})$ [nearest]
$t \leftarrow \circ(s + x)$ [nearest]
$u \leftarrow \circ(\log t)$ [nearest]
```

The error on $q$ is at most 1 ulp, thus that on $r$ is at most $\mathrm{ulp}(r) + \mathrm{ulp}(q) = (1 + 2^{\mathrm{EXP}(q) - \mathrm{EXP}(r)})\mathrm{ulp}(r)$. Since $r$ is smaller than $x^2 - 1$, we can use the simpler formula for the error on the square root, which gives as bound $(\frac{3}{2} + 2^{\mathrm{EXP}(q) - \mathrm{EXP}(r)})\mathrm{ulp}(s)$ for the error on $s$, and $(2 + 2^{\mathrm{EXP}(q)-\mathrm{EXP}(r)})\mathrm{ulp}(t)$ for that on $t$. This gives a final bound of $\frac{1}{2} + (2 + 2^{\mathrm{EXP}(q)-\mathrm{EXP}(r)})2^{1-\mathrm{EXP}(u)}$ for the error on $u$.

Since $2 + 2^{\mathrm{EXP}(q)-\mathrm{EXP}(r)} \leq 2^{1+\max(1,\mathrm{EXP}(q)-\mathrm{EXP}(r))}$, that shows the rounding error on the calculation of acosh$x$ can be bounded by $\frac{1}{2} + 2^{2+\max(1,\mathrm{EXP}(q)-\mathrm{EXP}(r))-\mathrm{EXP}(u)}\mathrm{ulp}(u)$.

3.8. **The hyperbolic sine function.** The `mpfr_sinh` ($\sinh x$) function implements the hyperbolic sine as :

$$\sinh x = \frac{1}{2}\left(e^x - \frac{1}{e^x}\right).$$

The algorithm used for the calculation of the hyperbolic sine is as follows[2]:

$$\begin{array}{rcl} u & \leftarrow & o(e^x) \\ v & \leftarrow & o(u^{-1}) \\ w & \leftarrow & o(u-v) \\ s & \leftarrow & \frac{1}{2}w \end{array}$$

Now, we have to bound the rounding error for each step of this algorithm. First, let consider the parity of hyperbolic sine ($\sinh(-x) = -\sinh(x)$) : the problem is reduced to calculate $\sinh x$ with $x \geq 0$. We can deduce $e^x \geq 1$ and $0 \leq e^{-x} \leq 1$.

[2] $o()$ represent the rounding error and $error(u)$ the error associate with the calculation of $u$

error$(u)$
$u \leftarrow \bigtriangledown(e^x)$
$(\bullet)$

$$|u - e^x| \leq \mathrm{ulp}(u)$$

error$(v)$
$v \leftarrow \bigtriangleup(u^{-1})$
$(\bullet\bullet)$

$$\begin{aligned} & |v - e^{-x}| \\ \leq\ & |v - u^{-1}| + |u^{-1} - e^{-x}| \\ \leq\ & \mathrm{ulp}(v) + \frac{1}{u \cdot e^x}|u - e^x| \\ \leq\ & \mathrm{ulp}(v) + \frac{1}{u^2}\mathrm{ulp}(u) \ \ (\star) \\ \leq\ & \mathrm{ulp}(v) + 2\mathrm{ulp}(\frac{1}{u}) \ \ (\star\star) \\ \leq\ & 3\,\mathrm{ulp}(v) \ \ (\star\star\star) \end{aligned}$$

$(\star)$
With $\frac{1}{u} \leq \frac{1}{e^x}$,
for that we must have $e^x \leq u$,
it is possible with $u = \bigtriangledown(e^x)$ $(\bullet)$
$(\star\star)$
From inequation [Rule 4],

$$a \cdot \mathrm{ulp}(b) \leq 2 \cdot \mathrm{ulp}(a \cdot b)$$

if $a = \frac{1}{u^2}$, $b = u$ then

$$\frac{1}{u^2}\mathrm{ulp}(u) \leq 2\mathrm{ulp}(\frac{1}{u})$$

$(\star\star\star)$
If $\mathrm{ulp}(\frac{1}{u}) \leq \mathrm{ulp}(v)$,
it is possible with $v = \bigtriangleup(u^{-1})$ $(\bullet\bullet)$

error$(w)$
$w \leftarrow o(u + v)$

$$\begin{aligned} & |w - (e^x - e^{-x})| \\ \leq\ & |w - (u - v)| + |u - e^x| + |-v + e^{-x}| \\ \leq\ & \mathrm{ulp}(w) + \mathrm{ulp}(u) + 3\mathrm{ulp}(v) \\ \leq\ & \mathrm{ulp}(w) + 4\mathrm{ulp}(u) \ \ (\star) \\ \leq\ & (1 + 4 \cdot 2^{\mathrm{EXP}(u) - \mathrm{EXP}(w)})\mathrm{ulp}(w) \ \ (\star\star) \end{aligned}$$

$(\star)$
With $v \leq 1 \leq u$
then $\mathrm{ulp}(v) \leq \mathrm{ulp}(u)$
$(\star\star)$
see subsection 1.3

error$(s)$
$s \leftarrow o(\frac{w}{2})$

$$\begin{aligned} \mathrm{error}(s) &= \mathrm{error}(w) \\ &\leq (1 + 4 \cdot 2^{\mathrm{EXP}(u) - \mathrm{EXP}(w)})\mathrm{ulp}(w) \end{aligned}$$

That show the rounding error on the calculation of $\sinh x$ can be bound by $(1 + 4 \cdot 2^{\mathrm{EXP}(u) - \mathrm{EXP}(w)})\mathrm{ulp}(w)$, then the number of bits need to add to the want accuracy to define intermediary variable is :

$$N_t = \lceil \log_2(1 + 4 \cdot 2^{\mathrm{EXP}(u) - \mathrm{EXP}(w)}) \rceil$$

3.9. **The inverse hyperbolic sine function.** The `mpfr_asinh` (acosh$x$) function implements the inverse hyperbolic sine as :

$$\mathrm{asinh} = \log\left(\sqrt{x^2 + 1} + x\right).$$

The algorithm used for the calculation of the inverse hyperbolic sine is as follows

$$\begin{array}{rcl} s & \leftarrow & o(x^2) \\ t & \leftarrow & o(s+1) \\ u & \leftarrow & o(\sqrt{t}) \\ v & \leftarrow & o(u+x) \\ w & \leftarrow & o(\log v) \end{array}$$

Now, we have to bound the rounding error for each step of this algorithm. First, let consider the parity of hyperbolic arc sine ($\text{asinh}(-x) = -\text{asinh}(x)$) : the problem is reduced to calculate asinh$x$ with $x \geq 0$.

| | | |
|---|---|---|
| error($s$) <br> $s \leftarrow o(x^2)$ | $\|s - x^2\|$ <br> $\leq$ ulp($s$) ($\star$) | |
| error($t$) <br> $t \leftarrow \bigtriangledown(s+1)$ <br> ($\bullet$) | $\|t - (x^2+1)\|$ <br> $\leq$ 2ulp($t$) ($\star$) | ($\star$) <br> see subsection 1.3 |
| error($u$) <br> $u \leftarrow o(\sqrt{t})$ | $\|u - \sqrt{x^2+1}\|$ <br> $\leq$ 3ulp($u$) ($\star$) | ($\star$) <br> see subsection 1.7 <br> with ($\bullet$) |
| error($v$) <br> $v \leftarrow o(u+x)$ | $\|v - (\sqrt{x^2+1}+x)\|$ <br> $\leq$ 5ulp($v$) ($\star$) | ($\star$) <br> see subsection 1.3 |
| error($w$) <br> $w \leftarrow o(\log v)$ | $\|w - \log(\sqrt{x^2+1}+x)\|$ <br> $\leq$ $(1+5.2^{1-\text{EXP}(w)})$ulp($w$) $\star$ | ($\star$) <br> see subsection 1.9 |

That shows the rounding error on the calculation of asinh$x$ can be bound by $(1 + 5.2^{1-\text{EXP}(w)})$ ulp on the result. So, to calculate the size of intermediary variables, we have to add, at least, $\lceil \log_2(1+5.2^{1-\text{EXP}(w)}) \rceil$ bits the wanted precision.

3.10. **The hyperbolic tangent function.** The hyperbolic tangent (`mpfr_tanh`) is computed from the exponential:

$$\tanh x = \frac{e^{2x}-1}{e^{2x}+1}.$$

The algorithm used is as follows, with working precision $p$ and rounding to nearest:

```
$u \leftarrow \circ(2x)$
$v \leftarrow \circ(e^u)$
$w \leftarrow \circ(v+1)$
$r \leftarrow \circ(v-1)$
$s \leftarrow \circ(r/w)$
```

Now, we have to bound the rounding error for each step of this algorithm. First, thanks to the parity of hyperbolic tangent — $\tanh(-x) = -\tanh(x)$ — we can consider without loss of generality that $x \geq 0$.

We use Higham's notation, with $\theta_i$ denoting variables such that $|\theta_i| \leq 2^{-p}$. Firstly, $u$ is exact. Then $v = e^{2x}(1+\theta_1)$ and $w = (e^{2x}+1)(1+\theta_2)^2$. The error on $r$ is bounded by $\frac{1}{2}\mathrm{ulp}(v) + \frac{1}{2}\mathrm{ulp}(r)$. Assume $\mathrm{ulp}(v) = 2^e \mathrm{ulp}(r)$, with $e \geq 0$; then the error on $r$ is bounded by $\frac{1}{2}(2^e+1)\mathrm{ulp}(r)$. We can thus write $r = (e^{2x}-1)(1+\theta_3)^{2^e+1}$, and then $s = \tanh(x)(1+\theta_4)^{2^e+4}$.

**Lemma 3.** *For $|x| \leq 1/2$, and $|y| \leq |x|^{-1/2}$, we have:*

$$|(1+x)^y - 1| \leq 2|y|x.$$

*Proof.* We have $(1+x)^y = e^{y\log(1+x)}$, with $|y\log(1+x)| \leq |x|^{-1/2}|\log(1+x)|$. The function $|x|^{-1/2}\log(1+x)$ is increasing on $[-1/2, 1/2]$, and takes as values $\approx -0.490$ in $x = -1/2$ and $\approx 0.286$ in $x = 1/2$, thus is bounded in absolute value by $1/2$. This yields $|y\log(1+x)| \leq 1/2$. Now it is easy to see that for $|t| \leq 1/2$, we have $|e^t - 1| \leq 1.3|t|$. Thus $|(1+x)^y - 1| \leq 1.3|y||\log(1+x)|$. The result follows from $|\log(1+x)| \leq 1.4|x|$ for $|x| \leq 1/2$, and $1.3 \times 1.4 \leq 2$. □

Applying the above lemma for $x = \theta_4$ and $y = 2^e + 4$, assuming $2^e + 4 \leq 2^{p/2}$, we get $s = \tanh(x)[1 + 2(2^e+4)\theta_5]$. Since $2^e + 4 \leq 2^{\max(3,e+1)}$, the relative error on $s$ is thus bounded by $2^{\max(4,e+2)-p}$.

3.11. **The inverse hyperbolic tangent function.** The `mpfr_atanh` (acosh$x$) function implements the inverse hyperbolic tangent as :

$$\mathrm{atanh} = \frac{1}{2}\log\frac{1+x}{1-x}.$$

The algorithm used for the calculation of the inverse hyperbolic tangent is as follows

$$\begin{aligned} s &\leftarrow o(1+x) \\ t &\leftarrow o(1-x) \\ u &\leftarrow o(\frac{s}{t}) \\ v &\leftarrow o(\log u) \\ w &\leftarrow o(\frac{1}{2}v) \end{aligned}$$

Now, we have to bound the rounding error for each step of this algorithm. First, let consider the parity of hyperbolic arc tangent ($\mathrm{atanh}(-x) = -\mathrm{atanh}(x)$) : the problem is reduced to calculate atanh$x$ with $x \geq 0$.

| | | |
|---|---|---|
| error($s$)<br>$s \leftarrow \triangle(1+x)$<br>($\bullet$) | $\|s-(1+x)\|$<br>$\leq\ 2\mathrm{ulp}(s)\ (\star)$ | see subsection 1.3 |
| error($t$)<br>$t \leftarrow \bigtriangledown(1-x)$<br>($\bullet\bullet$) | $\|t-(1-x)\|$<br>$\leq\ (1+2^{\mathrm{EXP}(x)-\mathrm{EXP}(t)})\mathrm{ulp}(t)\ (\star)$ | ($\star$)<br>see subsection 1.3 |
| error($u$)<br>$u \leftarrow o(\frac{s}{t})$ | $\|u-\frac{1+x}{1-x}\|$<br>$\leq\ (1+2\times 2+$<br>$\cdots\ 2\times(1+2^{\mathrm{EXP}(x)-\mathrm{EXP}(t)}))\mathrm{ulp}u\ (\star)$<br>$\leq\ (7+2^{\mathrm{EXP}(x)-\mathrm{EXP}(t)+1})\mathrm{ulp}(u)$ | ($\star$)<br>see subsection 1.5<br>with ($\bullet$) and ($\bullet\bullet$) |
| error($v$)<br>$v \leftarrow o(\log(u))$ | $\|v-(\log\frac{1+x}{1-x})\|$<br>$\leq\ (1+(7+2^{\mathrm{EXP}(x)-\mathrm{EXP}(t)+1})$<br>$\cdots\ \times 2^{1-\mathrm{EXP}(v)})\mathrm{ulp}(v)\ (\star)$<br>$\leq\ (1+7\times 2^{1-\mathrm{EXP}(v)}+$<br>$\cdots\ 2^{\mathrm{EXP}(x)-\mathrm{EXP}(t)-\mathrm{EXP}(v)+2})\mathrm{ulp}(v)$ | ($\star$)<br>see subsection 1.9 |
| error($w$)<br>$w \leftarrow o(\frac{1}{2}v)$ | $\|w-\frac{1}{2}\log\frac{1+x}{1-x}\|$<br>$\leq\ (1+7\times 2^{1-\mathrm{EXP}(v)}+$<br>$\cdots\ 2^{\mathrm{EXP}(x)-\mathrm{EXP}(t)-\mathrm{EXP}(v)+2})\mathrm{ulp}(w)\ \star$ | ($\star$) exact |

That shows the rounding error on the calculation of atanh$x$ can be bound by $(1+7\times 2^{1-\mathrm{EXP}(v)}+2^{\mathrm{EXP}(x)-\mathrm{EXP}(t)-\mathrm{EXP}(v)+2})$ ulp on the result. So, to calculate the size of intermediary variables, we have to add, at least, $\lceil\log_2(1+7\times 2^{1-\mathrm{EXP}(v)}+2^{\mathrm{EXP}(x)-\mathrm{EXP}(t)-\mathrm{EXP}(v)+2})\rceil$ bits the wanted precision.

3.12. **The arc-sine function.**

(1) We use the formula $\arcsin x = \arctan\frac{x}{\sqrt{1-x^2}}$
(2) We will have the when $x$ is near 1 we will experience uncertainty problems:
(3) If $x = a(1+\delta)$ with $\delta$ being the relative error then we will have

$$1-x = 1-a-a\delta = (1-a)[1-\frac{a}{1-a}\delta]$$

Ans so when using the arc tangent programs we need to take into account that decrease in precision.

(4) We will have

3.13. **The arc-cosine function.**

(1) Obviously, we used the formula

$$\arccos x = \frac{\pi}{2} - \arcsin x$$

(2) The problem of arccos is that it is 0 at 1, so, we have a cancellation problem to treat at 1.

(3) (Suppose $x \geq 0$, this is where the problem happens) The derivative of arccos is $\frac{-1}{\sqrt{1-x^2}}$ and we will have

$$\frac{1}{2\sqrt{1-x}} \leq |\frac{-1}{\sqrt{1-x^2}}| = \frac{1}{\sqrt{(1-x)(1+x)}} \leq \frac{1}{\sqrt{1-x}}$$

So, integrating the above inequality on $[x, 1]$ we get

$$\sqrt{1-x} \leq \arccos x \leq 2\sqrt{1-x}$$

(4) The important part is the lower bound that we get which tell us a upper bound on the cancellation that will occur:
The terms that are canceled are $\pi/2$ and $\arcsin x$, their order is 2. The number of canceled terms is so

```
1-1/2*MPFR_EXP(1-x)
```

## 3.14. The arc-tangent function.

### 3.14.1. *Binary splitting.*

The Taylor series for arctan is suitable for analysis using Binary splitting.

This method is detailed for example in “Pi and The AGM” p 334. It is efficient for rational numbers and is non efficient for non rational numbers.

The efficiency of this method is then quite limited. One can then wonder how to use it for non rational numbers.

Using the formulas

$$\arctan(-x) = -\arctan x \ \text{ and } \ \arctan x + \arctan \frac{1}{x} = \frac{\pi}{2}\text{sign}(x)$$

we can restrict ourselves to $0 \leq x \leq 1$.

Writing

$$x = \sum_{i=1}^{\infty} \frac{u_i}{2^i} \ \text{ with } \ u_i \in \{0, 1\}$$

or

$$x = \sum_{i=1}^{\infty} \frac{u_i}{2^{2^i}} \ \text{ with } \ u_i \in \{0, 1, \dots, 2_1^{2^{i-1}}\} \text{ if } i > 1 \text{ and } u_1 \in \{0, 1\}$$

we can compute cos, sin or exp using the formulas

$$\cos(a+b) = \cos a \cos b - \sin a \sin b$$
$$\sin(a+b) = \sin a \cos b + \cos a \sin b$$
$$\exp(a+b) = (\exp a)(\exp b)$$

Unfortunately for arctan there is no similar formulas. The only formula known is

$$\arctan x + \arctan y = \arctan \frac{x+y}{1-xy} + k\pi \ \text{ with } \ k \in \mathcal{Z}$$

we will use

$$\arctan x = \arctan y + \arctan \frac{x-y}{1+xy}$$

with $x, y > 0$ and $y < x$.

Summarizing we have the following facts:

(1) We can compute efficiently arctan $\frac{u}{2^{2^k}}$ with $k \geq 0$ and $u \in \{0, 1, \ldots, 2^{2^{k-1}}\}$
(2) We have a sort of addition formula for arctan, the term $k\pi$ being zero.

So I propose the following algorithm for $x$ given in $[0, 1]$.

(1) Write $v_k = 2^{2^k}$
(2) Define
$$s_{k+1} = \frac{s_k - A_k}{1 + s_k A_k} \text{ and } s_0 = x$$
(3) $A_k$ is chosen such that
$$0 \leq s_k - A_k < \frac{1}{v_k}$$
and $A_k$ is of the form $\frac{u_k}{v_k}$ with $u_k \in \mathcal{N}$.
(4) We have the formula
$$\begin{array}{rcl} \arctan x &=& \arctan A_0 + \arctan s_1 \\ &=& \arctan A_0 + \arctan A_1 + \arctan s_2 \\ &=& \arctan A_0 + \cdots + \arctan A_N + \arctan s_{N+1} \end{array}$$
the number $s_N$ is decreasing towards 0 and we then have
$$\arctan x \quad = \quad \textstyle\sum_{i=0}^{i=\infty} \arctan A_i$$

The drawbacks of this algorithm are:

(1) Complexity of the process is high, higher than the AGM. Nevertheless there is some hope that this can be more efficient than AGM is the domain where the number of bits is high but not too large.
(2) There is the need for division which is computationally expensive.
(3) We may have to compute arctan $(1/2)$.

3.14.2. *Estimate of absolute error.* By that analysis we mean that $a$ and $b$ have absolute error $D$ if $|a - b| \leq D$.
I give a remind of the algorithm:

(1) Write $v_k = 2^{2^k}$
(2) Define
$$s_{k+1} = \frac{s_k - A_k}{1 + s_k A_k} \text{ \textit{and} } s_0 = x$$
(3) $A_k$ is chosen such that
$$0 \leq s_k - A_k < \frac{1}{v_k}$$
and $A_k$ is of the form $\frac{u_k}{v_k}$ with $u_k \in \mathcal{N}$.
(4) We have the formula
$$\begin{array}{rcl} \arctan x &=& \arctan A_0 + \arctan s_1 \\ &=& \arctan A_0 + \arctan A_1 + \arctan s_2 \\ &=& \arctan A_0 + \cdots + \arctan A_N + \arctan s_{N+1} \end{array}$$
the number $s_N$ is very rapidly decreasing towards 0 and we then have
$$\arctan x \quad = \quad \textstyle\sum_{i=0}^{i=\infty} \arctan A_i$$

(5) The approximate arc tangent is then

$$\sum_{i=0}^{i=N_0} \arctan_{m_i} A_i$$

with $\arctan_{m_i}$ being the sum of the first $2^{m_i}$ terms of the Taylor series for arctan

We need to estimate all the quantities involved in the computation.

(1) We have the upper bound

$$0 \le s_{k+1} = \frac{s_k - A_k}{1 + s_k A_k} \le s_k - A_k \le \frac{1}{v_k}$$

(2) The remainder of the series giving $\arctan x$ is

$$\begin{array}{rcl} \sum_{i=N_0+1}^{\infty} \arctan A_i & \le & \sum_{i=N_0+1}^{\infty} A_i \\ & \le & \sum_{i=N_0+1}^{\infty} s_i \\ & \le & \sum_{i=N_0+1}^{\infty} \frac{1}{v_{i-1}} \\ & \le & \sum_{i=N_0}^{\infty} \frac{1}{v_i} \\ & \le & \sum_{i=N_0}^{\infty} \frac{1}{2^{2^i}} = \frac{c_{N_0}}{2^{2^{N_0}}} \end{array}$$

With $c_{N_0} \le 1.64$. If $N_0 \ge 1$ then $c_{N_0} \le 1.27$. If $N_0 \ge 2$ then $c_{N_0} \le 1.07$.
It remains to determine the good $N_0$.

(3) The partial sum of the Taylor series for arctan have derivative bounded by 1 and consequently don't increase error.

(4) The error created by using the partial sum of the Taylor series of arctan is bounded by

$$\frac{(A_i)^{2\times 2^{m_i}+1}}{2 * 2^{m_i} + 1}$$

and is thus bounded by

$$\begin{array}{rcl} \frac{1}{2*2^{m_i}+1} * [\frac{1}{2^{2^{i-1}}}]^{2\times 2^{m_i}+1} & = & \frac{1}{2*2^{m_i}+1} * [2^{-2^{i-1}}]^{2\times 2^{m_i}+1} \\ & \le & \frac{1}{2*2^{m_i}+1} * [2^{-2^{i-1}}]^{2\times 2^{m_i}} \\ & \le & \frac{1}{2*2^{m_i}+1} * 2^{-2^{i+m_i}} \end{array}$$

The calculation of $\frac{\arctan A_i}{A_i}$ is done by using integer arithmetic and returning a fraction that is converted to mpfr type so there is no error. But to compute $\arctan A_i = A_i[\frac{\arctan A_i}{A_i}]$ we need to use real arithmetic (a little drawback of Jeandel procedure generic.c) so there is $1ulp$ error.
In total this is $(N_0)ulp$.

(5) Addition give $1ulp$ There are $(N_0 - 1)$ addition so we need to take $(N_0 - 1)ulp$.

(6) The division create errors:
   (a) Having errors in the computation of $A_i$ is of no consequences: It changes the quantity being arc-tangented and that's all. Errors concerning the computation of $s_{N+1}$ in contrary adds to the error.
   (b) The subtract operation $s_i - A_i$ has the effect of subtracting very near numbers. But $A_i$ has exactly the same first 1 and 0 than $s_i$ so we can expect this operation to be nondestructive.
   (c) Extrapolating from the previous result we can expect that the error of the quantity $\frac{s_i - A_i}{1+s_i A_i}$ is $err(s_i) + 1ulp$

(7) The total sum of errors is then (if no errors are done in the counting of errors)

$$\begin{array}{rcl}
Err(\text{arctan}) & = & \sum_{i=0}^{i=N_0} \frac{1}{2*2^{m_i}+1} 2^{-2^{i+m_i}} + \frac{2}{2^{2^{N_0}}} + (N_0 - 1)2^{-Prec} \\
& + & (N_0 - 1)2^{-Prec} + (N_0)2^{-Prec} \quad [m_i = N_0 - i] \\
& = & \sum_{i=0}^{i=N_0} \frac{1}{2*2^{N_0 - i}+1} 2^{-2^{N_0}} + \frac{2}{2^{2^{N_0}}} + (3 * N_0 - 2)2^{-Prec} \\
& = & \sum_{i=0}^{i=N_0} \frac{1}{2*2^{i}+1} 2^{-2^{N_0}} + \frac{2}{2^{2^{N_0}}} + (3 * N_0 - 2)2^{-Prec} \\
& \leq & \{\sum_{i=0}^{i=\infty} \frac{1}{2*2^{i}+1}\} 2^{-2^{N_0}} + \frac{2}{2^{2^{N_0}}} + (3 * N_0 - 2)2^{-Prec} \\
& \leq & \{0.77\} 2^{-2^{N_0}} + \frac{1.63}{2^{2^{N_0}}} + (3 * N_0 - 2)2^{-Prec} \\
& = & \frac{2.4}{2^{2^{N_0}}} + (3 * N_0 - 2)2^{-Prec}
\end{array}$$

This is what we wish thus $Err(\text{arctan}) < 2^{-prec\_arctan}$ with $prec\_arctan$ is the requested precision on the arc-tangent. We thus want:

$$\begin{array}{c}
\frac{2.4}{2^{2^{N_0}}} \leq 2^{-prec\_arctan - 1} \\
et \ \ (3 * N_0 - 2)2^{-Prec} \leq 2^{-prec\_arctan - 1}
\end{array}$$

i.e. $N_0 \geq \frac{\ln(prec\_arctan + 1 + \frac{\ln 2.4}{\ln 2})}{\ln 2}$ that we approach by (since the logarithm is expensive):

$$N_0 = ceil(\log(prec\_arctan + 2.47) * 1.45)$$

and we finally have:

$$Prec = prec\_arctan + \{1 + ceil(\frac{\ln(3N_0 - 2)}{\ln 2})\}$$

3.14.3. *Estimate of the relative error.* we say that $a$ and $b$ have relative error $\delta$ if

$$a = b(1 + \Delta) \text{ with } |\Delta| \leq \delta$$

This is the error definition used in mpfr. So we need to redo everything in order to have a consistent analysis.

(1) We can use freely all previous estimate:
   (a) Remainder estimate:

$$\sum_{i=N_0+1}^{\infty} \text{arctan } A_i \leq \frac{c_{N_0}}{2^{2^{N_0}}}$$

   so the relative error will be $\frac{1}{\text{arctan } x} \frac{c_{N_0}}{2^{2^{N_0}}}$.
   (b) The relative error created by using a partial sum of Taylor series is bounded by $\frac{1}{\text{arctan } A_i} \frac{1}{2*2^{m_i}+1} * 2^{-2^{i+m_i}}$.
   (c) Multiplication $\text{arctan } A_i = A_i[\frac{\text{arctan } A_i}{A_i}]$ take 1 ulp of relative error.
   (d) Doing the subtraction $s_i - A_i$ is a gradual underflow operation: it decrease the precision of $s_i - A_i$.
   (e) The multiplication $a_i A_i$ create 1 ulp of error. This is not much and this relative error is further reduced by adding 1.

(1) We have

$$\begin{array}{rcl} \arctan b(1+\Delta) & = & \arctan(b+b\Delta) \\ & \sim & \arctan b + \frac{1}{1+b^2}(b\Delta) \\ & = & [\arctan b][1 + \{\frac{b}{(1+b^2)\arctan b}\}\Delta] \end{array}$$

A rapid analysis gives $0 \leq \frac{b}{(1+b^2)\arctan b} \leq 1$ and then we can say that the function arctan does not increase the relative error.

(2) So we have two possible solutions:
   (a) Do a relative analysis of our algorithm.
   (b) Use the previous analysis since the absolute error $D$ is obviously equal to $|b|\delta$ ($\delta$ being the relative error)

   it is not hard to see that second solution is certainly better: The formulas are additive. Our analysis will work without problems.

(3) It then suffices to replace in the previous section $2^{-prec\_arctan}$ by $2^{-prec\_arctan} \arctan x$.

(4) If $|x| \leq 1$ then $|\arctan x|$ is bounded below by $|x|\frac{4}{\pi} \sim |x|1.27$. So it suffices to have an absolute error bounded above by

$$2^{-prec\_arctan}|x|1.27$$

   In this case we will add $2 - MPFR\_EXP(x)$ to *prec_arctan*

(5) If $|x| \geq 1$ then $\arctan x$ is bounded below by $\frac{\pi}{4}$. So it suffices to have an absolute error bounded above by

$$2^{-prec\_arctan}1.27$$

   we will add 1 to *prec_arctan*.
   In this case we need to take into account the error caused by the subtraction:

$$\arctan x = \pm\frac{\pi}{2} - \arctan\frac{1}{x}$$

3.14.4. *Implementation defaults.*

(1) The computation is quite slow, this should be improved.
(2) The precision should be decreased after the operation $s_i - A_i$. And several other improvement should be done.

3.15. **The euclidean distance function.** The `mpfr_hypot` function implements the euclidean distance function:

$$\mathrm{hypot}(x,y) = \sqrt{x^2+y^2}.$$

The algorithm used is as follows, where all roundings are done towards zero:

$$\begin{array}{rcl} u & \leftarrow & o(x^2) \\ v & \leftarrow & o(y^2) \\ w & \leftarrow & o(u+v) \\ s & \leftarrow & o(\sqrt{w}) \end{array}$$

Since the inputs $x$ and $y$ are exact, we have $|u - x^2| \leq \mathrm{ulp}(u)$ and $|v - y^2| \leq \mathrm{ulp}(v)$; then $|w - (x^2+y^2)| \leq \mathrm{ulp}(w) + |u - x^2| + |v - y^2| \leq 3\mathrm{ulp}(w)$ since $w$ is greater or equal to $u$ and $v$. For the last step $s \leftarrow o(\sqrt{w})$, we use the last formula from §1.7, which gives $|s - \sqrt{x^2+y^2}| \leq 4\mathrm{ulp}(s)$.

3.16. **The floating multiply-add function.** The `mpfr_fma` (fma$(x, y, z)$) function implements the floating multiply-add function as :

$$\text{fma}(x, y, z) = z + x \times y.$$

The algorithm used for this calculation is as follows:

$$\begin{array}{rcl} u & \leftarrow & o(x \times y) \\ v & \leftarrow & o(z + u) \end{array}$$

Now, we have to bound the rounding error for each step of this algorithm.

$$\begin{array}{llcll} \text{error}(u) & & & & \\ u \leftarrow o(x \times y) & |u - (xy)| & \leq & ulp(u) & \\ & & & & \\ \text{error}(v) & |v - (z + xy)| & \leq & \text{ulp}(v) + |(z + u) - (z + xy)| & (\star) \\ v \leftarrow o(z + u) & & \leq & (1 + 2^{e_u - e_v})\text{ulp}(v) \;\; (\star) & \text{see subsection 1.3} \end{array}$$

That shows the rounding error on the calculation of fma$(x, y, z)$ can be bound by $(1 + 2^{e_u - e_v})$ulp on the result. So, to calculate the size of intermediary variables, we have to add, at least, $\lceil \log_2(1 + 2^{e_u - e_v}) \rceil$ bits the wanted precision.

3.17. **The expm1 function.** The `mpfr_expm1` (expm1$(x)$) function implements the expm1 function as :

$$\text{expm1}(x) = e^x - 1.$$

The algorithm used for this calculation is as follows:

$$\begin{array}{rcl} u & \leftarrow & o(e^x) \\ v & \leftarrow & o(u - 1) \end{array}$$

Now, we have to bound the rounding error for each step of this algorithm.

$$\begin{array}{llcll} \text{error}(u) & & & & \\ u \leftarrow o(e^x) & |u - e^x| & \leq & ulp(u) & \\ \text{error}(v) & & & & (\star) \\ v \leftarrow o(u - 1) & |v - (e^x - 1)| & \leq & (1 + 2^{e_u - e_v})\text{ulp}(v) \;\; (\star) & \text{see subsection 1.3} \end{array}$$

That shows the rounding error on the calculation of expm1$(x)$ can be bound by $(1 + 2^{e_u - e_v})$ulp on the result. So, to calculate the size of intermediary variables, we have to add, at least, $\lceil \log_2(1 + 2^{e_u - e_v}) \rceil$ bits the wanted precision.

3.18. **The log1p function.** The `mpfr_log1p` ($\mathrm{log1p}(x)$) function implements the log1p function as :

$$\mathrm{log1p}(x) = \log(1 + x).$$

The algorithm used for this calculation is as follows:

$$\begin{array}{rcl} u & \leftarrow & o(x) \\ v & \leftarrow & o(1+u) \\ w & \leftarrow & o(\log(v)) \end{array}$$

Now, we have to bound the rounding error for each step of this algorithm.

| | | |
|---|---|---|
| error($u$) <br> $u \leftarrow o(x)$ | $\|u - x\| \leq ulp(u)$ | |
| error($v$) <br> $v \leftarrow o(1+u)$ | $\|v - (1+x)\| \leq (1 + 2^{e_u - e_v})\mathrm{ulp}(v)$ | ($\star$) see subsection 1.3 |
| error($w$) <br> $w \leftarrow$ <br> $o(\log(v))$ | $\|v - \log v\| \leq (1 + (1 + 2^{e_u-e_v})2^{1-e_w})$ <br> $\ldots\ \mathrm{ulp}(w)$ | ($\star$) see subsection 1.9 |

That shows the rounding error on the calculation of $\log 1\mathrm{p}(x)$ can be bound by $(1 + (1 + 2^{e_u-e_v})2^{1-e_w})$ulp on the result. So, to calculate the size of intermediary variables, we have to add, at least, $\lceil \log_2(1 + (1 + 2^{e_u-e_v})2^{1-e_w}) \rceil$ bits the wanted precision.

3.19. **The log2 or log10 function.** The `mpfr_log2` or `mpfr_log10` function implements the log in base 2 or 10 function as :

$$\mathrm{log2}(x) = \frac{\log x}{\log 2}$$

or

$$\mathrm{log10}(x) = \frac{\log x}{\log 10}.$$

The algorithm used for this calculation is the same for log2 or log10 and is described as follows for $t = 2$ or 10:

$$\begin{array}{rcl} u & \leftarrow & o(\log(x)) \\ v & \leftarrow & o(\log(t)) \\ w & \leftarrow & o(\frac{u}{v}) \end{array}$$

Now, we have to bound the rounding error for each step of this algorithm with $x \geq 0$ and $y$ is a floating number.

| | | |
|---|---|---|
| error($u$) $u \leftarrow \triangle(\log(x))$ ($\bullet$) | $\|u - \log(x)\| \leq \mathrm{ulp}(u)$ | |
| error($v$) $v \leftarrow \triangledown(\log t)$ ($\bullet\bullet$) | $\|v - \log t\| \leq \mathrm{ulp}(v)$ | |
| error($w$) $w \leftarrow o(\frac{u}{v})$ | $\|v - (\dfrac{\log x}{\log t})\| \leq 5\mathrm{ulp}(w)$ ($\star$) | ($\star$) see subsection 1.6 |

That shows the rounding error on the calculation of $log2$ or $log10$ can be bound by 5ulp on the result. So, to calculate the size of intermediary variables, we have to add, at least, 3 bits the wanted precision.

3.20. **The power function.** The `mpfr_pow` function implements the power function as:

$$\mathrm{pow}(x, y) = e^{y \log(x)}.$$

The algorithm used for this calculation is as follows:

$$\begin{aligned} u &\leftarrow o(\log(x)) \\ v &\leftarrow o(yu) \\ w &\leftarrow o(e^v) \end{aligned}$$

Now, we have to bound the rounding error for each step of this algorithm with $x \geq 0$ and $y$ is a floating number.

| | | |
|---|---|---|
| error($u$) $u \leftarrow o(\log(x))$ | $\|u - \log(x)\| \leq \mathrm{ulp}(u)$ $\star$ | |
| error($v$) $v \leftarrow \triangle(y \times u)$ ($\bullet$) | $\|v - y\log(x)\| \leq \mathrm{ulp}(v) + \|yu - y\log(x)\|$ | |
| | $\leq \mathrm{ulp}(v) + y\|u - \log(x)\|$ | ($\star$) with [Rule 4] |
| | $\leq \mathrm{ulp}(v) + y\mathrm{ulp}(u)$ | ($\star$) |
| | $\leq \mathrm{ulp}(v) + 2\mathrm{ulp}(yu)$ ($\star$) | with [Rule 8] |
| | $\leq 3\mathrm{ulp}(v)$ ($\star\star$) | |
| error($w$) $w \leftarrow o(e^v)$ | $\|w - e^v\| \leq (1 + 3 \cdot 2^{\mathrm{EXP}(v)+1})\mathrm{ulp}(w)$ | ($\star$) see subsection 1.8 with $c_u^* = 1$ for ($\bullet$) |

That shows the rounding error on the calculation of $x^y$ can be bound by $1 + 3 \cdot 2^{\mathrm{EXP}(v)+1}$ ulps on the result. So, to calculate the size of intermediary variables, we have to add, at least, $\lceil \log_2(1 + 3 \cdot 2^{\mathrm{EXP}(v)+1}) \rceil$ bits to the wanted precision.

Exact results. We have to detect cases where $x^y$ is exact, otherwise the program will loop forever. The theorem from Gelfond/Schneider (1934) states that if $\alpha$ and $\beta$ are algebraic numbers with $\alpha \neq 0$, $\alpha \neq 1$, and $\beta \notin \mathbb{Q}$, then $\alpha^\beta$ is transcendental. This is of little help for us since $\beta$ will always be a rational number. Let $x = a2^b$, $y = c2^d$, and assume $x^y = e2^f$, where $a, b, c, d, e, f$ are integers. Without loss of generality, we can assume $a, c, e$ odd integers.

If $x$ is negative: either $y$ is integer, then $x^y$ is exact if and only if $(-x)^y$ is exact; or $y$ is rational, then $x^y$ is a complex number, and the answer is NaN (Not a Number). Thus we can assume $a$ (and therefore $e$) positive.

If $y$ is negative, then $x^y = a^y 2^{by}$ can be exact only when $a = 1$, and in that case we also need that $by$ is an integer.

We have $a^{c2^d} 2^{bc2^d} = e2^f$ with $a, c, e$ odd integers, and $a, e > 0$. As $a$ is an odd integer, necessarily we have $a^{c2^d} = e$ and $2^{bc2^d} = 2^f$, thus $bc2^d = f$.

If $d \geq 0$, then $a^c$ must be an integer: this is true if $c \geq 0$, and false for $c < 0$ since $a^{c2^d} = \frac{1}{a^{-c2^d}} < 1$ cannot be an integer. In addition $a^{c2^d}$ must be representable in the given precision.

Assume now $d < 0$, then $a^{c2^d} = a^{c1/2^{d'}}$ with $d' = -d$, thus we have $a^c = e^{2^{d'}}$, thus $a^c$ must be a $2^{d'}$-th power of an integer. Since $c$ is odd, $a$ itself must be a $2^{d'}$-th power.

We therefore propose the following algorithm:

```
Algorithm CheckExactPower.
Input: $x = a2^b$, $y = c2^d$, $a, c$ odd integers
Output: *true* if $x^y$ is an exact power $e2^f$, *false* otherwise
**if** $x < 0$ **then**
   **if** $y$ is an integer **then** return CheckExactPower$(-x, y)$
   **else** return *false*
**if** $y < 0$ **then**
   **if** $a = 1$ **then** return *true* **else** return *false*
**if** $d < 0$ **then**
   **if** $a2^b$ is not a $2^{-d}$ power **then** return *false*
return *true*
```

Detecting if the result is exactly representable is not enough, since it may be exact, but with a precision larger than the target precision. Thus we propose the following: modify Algorithm CheckExactPower so that it returns an upper bound $p$ for the number of significant bits of $x^y$ when it is exactly representable, i.e. $x^y = m \cdot 2^e$ with $|m| < 2^p$. Then if the relative error on the approximation of $x^y$ is less than $\frac{1}{2}$ ulp, then rounding it to nearest will give $x^y$.

3.21. **The real cube root.** The `mpfr_cbrt` function computes the real cube root of $x$. Since for $x < 0$, we have $\sqrt[3]{x} = -\sqrt[3]{-x}$, we can focus on $x > 0$.

Let $n$ be the number of wanted bits of the result. We write $x = m \cdot 2^{3e}$ where $m$ is a positive integer with $m \geq 2^{3n-3}$. Then we compute the integer cubic root of $m$: let $m = s^3 + r$ with $0 \leq r$ and $m < (s+1)^3$. Let $k$ be the number of bits of $s$: since $m \geq 2^{3n-3}$, we have $s \geq 2^{n-1}$ thus $k \geq n$. If $k > n$, we replace $s$ by $\lfloor s2^{n-k} \rfloor$, $e$ by $e + (k - n)$, and update $r$ accordingly so that $x = (s^3 + r)2^{3e}$ still holds (be careful that $r$ may no longer be an integer in that case).

Then the correct rounding of $\sqrt[3]{x}$ is:

$$\begin{array}{rl} s2^e & \text{if } r = 0 \text{ or round down or round nearest and } r < \frac{3}{2}s^2 + \frac{3}{4}s + \frac{1}{8}, \\ (s+1)2^e & \text{otherwise.} \end{array}$$

Note: for rounding to nearest, one may consider $m \geq 2^{3n}$ instead of $m \geq 2^{3n-3}$, i.e. taking $n+1$ instead of $n$. In that case, there is no need to compare the remainder $r$ to $\frac{3}{2}s^2 + \frac{3}{4}s + \frac{1}{8}$:

we just need to know whether $r = 0$ or not. The even rounding rule is needed only when the input $x$ has at least $3n + 1$ bits, since the cube of a odd number of $n + 1$ bits has at least $3n + 1$ bits.

3.22. **The exponential integral.** The exponential integral `mpfr_eint` is defined as in [1, formula 5.1.10]: for $x > 0$,

$$\mathrm{Ei}(x) = \gamma + \log x + \sum_{k=1}^{\infty} \frac{x^k}{k\,k!},$$

and for $x < 0$ it gives NaN.

We use the following integer-based algorithm to evaluate $\sum_{k=1}^{\infty} \frac{x^k}{k\,k!}$, using working precision $w$. For any real $v$, we denote by $\mathrm{trunc}(v)$ the nearest integer towards zero.

```
Decompose $x$ into $m \cdot 2^e$ with $m$ integer [exact]
If necessary, truncate $m$ to $w$ bits and adjust $e$
$s \leftarrow 0$
$t \leftarrow 2^w$
for $k := 1$ do
    $t \leftarrow \mathrm{trunc}(tm2^e/k)$
    $u \leftarrow \mathrm{trunc}(t/k)$
    $s \leftarrow s + u$
Return $s \cdot 2^{-w}$.
```

Note: in $t \leftarrow \mathrm{trunc}(tm2^e/k)$, we first compute $tm$ exactly, then if $e$ is negative, we first divide by $2^{-e}$ and truncate, then divide by $k$ and truncate; this gives the same answer than dividing once by $k2^{-e}$, but it is more efficient. Let $\epsilon_k$ be the absolute difference between $t$ and $2^w x^k/k!$ at step $k$. We have $\epsilon_0 = 0$, and $\epsilon_k \leq 1 + \epsilon_{k-1} m 2^e/k + t_{k-1} m 2^{e+1-w}/k$, since the error when approximating $x$ by $m2^e$ is less than $m2^{e+1-w}$. Similarly, the absolute error on $u$ at step $k$ is at most $\nu_k \leq 1 + \epsilon_k/k$, and that on $s$ at most $\tau_k \leq \tau_{k-1} + \nu_k$. We compute all these errors dynamically (using MPFR with a small precision), and we stop when $|t|$ is smaller than the bound $\tau_k$ on the error on $s$ made so far.

At that time, the truncation error when neglecting terms of index $k + 1$ to $\infty$ can be bounded by $(|t| + \epsilon_k)/k(|x|/k + |x|^2/k^2 + \cdots) \leq (|t| + \epsilon_k)|x|/k/(k - |x|)$.

3.23. **The gamma function.** The gamma function is computed by Spouge's method [13]:

$$\Gamma(z+1) \approx (z+a)^{z+1/2} e^{-z-a} \left[ \sqrt{2\pi} + \sum_{k=1}^{\lceil a \rceil - 1} \frac{c_k(a)}{z+k} \right],$$

which is valid for $\Re(z + a) > 0$, where

$$c_k(a) = \frac{(-1)^{k-1}}{(k-1)!} (a-k)^{k-1/2} e^{a-k}.$$

Here, we choose the free parameter $a$ to be an integer.

According to [11, Section 2.6], the relative error is bounded by $a^{-1/2}(2\pi)^{-a-1/2}$ for $a \geq 3$ and $\Re(z) \geq 0$.

3.24. **The Riemann Zeta function.** The algorithm for the Riemann Zeta function is due to Jean-Luc Rémy and Sapphorain Pétermann. For $s < 1/2$ we use the functional equation

$$\zeta(s) = 2^s \pi^{s-1} \sin\left(\frac{\pi s}{2}\right) \Gamma(1-s)\zeta(1-s).$$

For $s \geq 1/2$ we use the Euler-MacLaurin summation formula, applied to the real function $f(x) = x^{-s}$ for $s > 1$:

$$\zeta(s) = \sum_{k=1}^{N-1} \frac{1}{k^s} + \frac{1}{2N^s} + \frac{1}{(s-1)N^{s-1}} + \sum_{k=1}^{p} \frac{B_{2k}}{2k} \binom{s+2k-2}{2k-1} \frac{1}{N^{s+2k-1}} + R_{N,p}(s),$$

with $|R_{N,p}(s)| < 2^{-d}$, where $B_k$ denotes the $k$th Bernoulli number,

$$p = \max\left(0, \lceil \frac{d \log 2 + 0.61 + s \log(2\pi/s)}{2} \rceil\right),$$

and $N = \lceil 2^{(d-1)/s} \rceil$ if $p = 0$, $N = \lceil \frac{s+2p-1}{2\pi} \rceil$ if $p > 0$.

This computation is split into three parts:

$$A = \sum_{k=1}^{N-1} k^{-s} + \frac{1}{2} N^{-s},$$

$$B = \sum_{k=1}^{p} T_k = N^{-1-s} s \sum_{k=1}^{p} C_k \Pi_k N^{-2k+2},$$

$$C = \frac{N^{-s+1}}{s-1},$$

where $C_k = \frac{B_{2k}}{(2k)!}$, $\Pi_k = \prod_{j=1}^{2k-2}(s+j)$, and $T_k = N^{1-2k-s} C_k \Pi_k$.

Rémy and Pétermann proved the following result:

**Theorem 1.** *Let $d$ be the target precision so that $|R_{N,p}(s)| < 2^{-d}$. Assume $\Pi = d - 2 \geq 11$, i.e. $d \geq 13$. If the internal precisions for computing $A$, $B$, $C$ satisfy respectively*

$$D_A \geq \Pi + \lceil \frac{3}{2} \frac{\log N}{\log 2} \rceil + 5, \quad D_B \geq \Pi + 14, \quad D_C \geq \Pi + \lceil \frac{1}{2} \frac{\log N}{\log 2} \rceil + 7,$$

*then the relative round-off error is bounded by $2^{-\Pi}$, i.e. if $z$ is the approximation computed, we have*

$$|\zeta(s) - z| \leq 2^{-\Pi} |\zeta(s)|.$$

3.24.1. *The integer argument case.* In case of an integer argument $s \geq 2$, the `mpfr_zeta_ui` function computes $\zeta(s)$ using the following formula from [3]:

$$\zeta(s) = \frac{1}{d_n(1-2^{1-s})} \sum_{k=0}^{n-1} \frac{(-1)^k (d_n - d_k)}{(k+1)^s} + \gamma_n(s),$$

where

$$|\gamma_n(s)| \leq \frac{3}{(3+\sqrt{8})^n} \frac{1}{1-2^{1-s}} \quad \text{and} \quad d_k = n \sum_{i=0}^{k} \frac{(n+i-1)! 4^i}{(n-i)!(2i)!}.$$

It can be checked that the $d_k$ are integers, and we compute them exactly, like the denominators $(k+1)^s$. We compute the integer

$$S = \sum_{k=0}^{n-1} (-1)^k \lfloor \frac{d_n - d_k}{(k+1)^s} \rfloor.$$

The absolute error on $S$ is at most $n$. We then perform the following iteration (still with integers):

```
$T \leftarrow S$
**do**
   $T \leftarrow \lfloor T 2^{1-s} \rfloor$
   $S = S + T$
**while** $T \neq 0$.
```

Since $\frac{1}{1-2^{1-s}} = 1 + 2^{1-s} + 2^{2(1-s)} + \cdots$, and at iteration $i$ we have $T = \lfloor S 2^{i(1-s)} \rfloor$, the error on $S$ after this loop is bounded by $n + l + 1$, where $l$ is the number of loops.

Finally we compute $q = \lfloor \frac{2^p S}{d_n} \rfloor$, where $p$ is the working precision, and we convert $q$ to a $p$-bit floating-point value, with rounding to nearest, and divide by $2^p$ (this last operation is exact). The final error in ulps is bounded by $1 + 2^\mu(n+l+2)$. Since $S/d_n$ approximates $\zeta(s)$, it is larger than one, thus $q \geq 2^p$, and the error on the division is less that $\frac{1}{2}\text{ulp}_p(q)$. The error on $S$ itself is bounded by $(n+l+1)/d_n \leq (n+l+1)2^{1-p}$ — see the conjecture below. Since $2^{1-p} \leq \text{ulp}_p(q)$, and taking into account the error when converting the integer $q$ (which may have more than $p$ bits), and the mathematical error which is bounded by $\frac{3}{(3+\sqrt{8})^n} \leq \frac{3}{2^p}$, the total error is bounded by $n + l + 4$ ulps.
Analysis of the sizes. To get an accuracy of around $p$ bits, since $\zeta(s) \geq 1$, it suffices to have $|\gamma_n(s)| \leq 2^{-p}$, i.e. $(3+\sqrt{8})^n \geq 2^p$, thus $n \geq \alpha p$ with $\alpha = \frac{\log 2}{\log((3+\sqrt{8})} \approx 0.393$. It can be easily seen that $d_n \geq 4^n$, thus when $n \geq \alpha p$, $d_n$ has at least $0.786p$ bits. In fact, we conjecture $d_n \geq 2^{p-1}$ when $n \geq \alpha p$; this conjecture was experimentally verified up to $p = 1000$.

3.25. **The arithmetic-geometric mean.** The arithmetic-geometric mean (AGM for short) of two positive numbers $a \leq b$ is defined to be the common limits of the sequences $(a_n)$ and $(b_n)$ defined by $a_0 = a$, $b_0 = b$, and for $n \geq 0$:

$$a_{n+1} = \sqrt{a_n b_n}, \quad b_{n+1} = \frac{a_n + b_n}{2}.$$

We approximate $\text{AGM}(a, b)$ as follows, with working precision $p$:

```
$s_1 = \circ(ab)$
$u_1 = \circ(\sqrt{s_1})$
$v_1 = \circ(a+b)/2$ [division by 2 is exact]
**for** $n := 1$ **while** $\text{EXP}(v_n) - \text{EXP}(v_n - u_n) \leq p - 2$ **do**
   $v_{n+1} = \circ(u_n + v_n)/2$ [division by 2 is exact]
   **if** $\text{EXP}(v_n) - \text{EXP}(v_n - u_n) \leq p/4$ **then**
      $s = \circ(u_n v_n)$
      $u_{n+1} = \circ(\sqrt{s})$
   **else**
      $s = \circ(v_n - u_n)$
      $t = \circ(s^2)/16$ [division by 16 is exact]
```

```
        $w = \circ(t/v_{n+1})$
        return $r = \circ(v_{n+1} - w)$
    **endif**
```

The rationale behind the **if**-test is the following. When the relative error between $a_n$ and $b_n$ is less than $2^{-p/4}$, we can write $a_n = b_n(1+\epsilon)$ with $|\epsilon| \leq 2^{-p/4}$. The next iteration will compute $a_{n+1} = \sqrt{a_n b_n} = b_n\sqrt{1+\epsilon}$, and $b_{n+1} = (a_n + b_n)/2 = b_n(1+\epsilon/2)$. The second iteration will compute $a_{n+2} = \sqrt{a_{n+1}b_{n+1}} = b_n\sqrt{\sqrt{1+\epsilon}(1+\epsilon/2)}$, and $b_{n+2} = (a_{n+1} + b_{n+1})/2 = b_n(\sqrt{1+\epsilon}/2 + 1/2 + \epsilon/4)$. When $\epsilon$ goes to zero, the following expansions hold:

$$\begin{aligned}
\sqrt{\sqrt{1+\epsilon}(1+\epsilon/2)} &= 1 + \frac{1}{2}\epsilon - \frac{1}{16}\epsilon^2 + \frac{1}{32}\epsilon^3 - \frac{11}{512}\epsilon^4 + O(\epsilon^5) \\
\sqrt{1+\epsilon}/2 + 1/2 + \epsilon/4 &= 1 + \frac{1}{2}\epsilon - \frac{1}{16}\epsilon^2 + \frac{1}{32}\epsilon^3 - \frac{5}{256}\epsilon^4 + O(\epsilon^5),
\end{aligned}$$

which shows that $a_{n+2}$ and $b_{n+2}$ agree to $p$ bits. In the algorithm above, we have $v_{n+1} \approx b_n(1+\epsilon/2)$, $s = -b_n\epsilon$ [exact thanks to Sterbenz theorem], then $t \approx \epsilon^2 b_n^2/16$, and $w \approx (b_n/16)\epsilon^2/(1+\epsilon/2) \approx b_n(\epsilon^2/16 - \epsilon^3/32)$, thus $v_{n+1} - w$ gives us an approximation to order $\epsilon^4$. [Note that $w$ — and therefore $s, t$ — need to be computed to precision $p/2$ only.]

**Lemma 4.** *Assuming $u \leq v$ are two $p$-bit floating-point numbers, then $u' = \circ(\sqrt{\circ(uv)})$ and $v' = \circ(u+v)/2$ satisfy:*

$$u \leq u', v' \leq v.$$

*Proof.* It is clear that $2u \leq u + v \leq 2v$, and since $2u$ and $2v$ are representable numbers, $2u \leq \circ(u+v) \leq 2v$, thus $u \leq v' \leq v$.

The result for $u'$ is more difficult to obtain. We use the following result: if $x$ is a $p$-bit number, $s = \circ(x^2)$, and $t = \circ(\sqrt{s})$ are computed with precision $p$ and rounding to nearest, then $t = x$.

Apply this result to $x = u$, and let $s' = \circ(uv)$. Then $s = \circ(u^2) \leq s'$, thus $u = \circ(\sqrt{s}) \leq \circ(\sqrt{s'}) = u'$. We prove similarly that $u' \leq v$. □

*Remark.* We cannot assume that $u' \leq v'$. Take for example $u = 9$, $v = 12$, with precision $p = 4$. Then $(u+v)/2$ rounds to 10, whereas $\sqrt{uv}$ rounds to 112, and $\sqrt{112}$ rounds to 11.

We use Higham error analysis method, where $\theta$ denotes a generic value such that $|\theta| \leq 2^{-p}$. We note $a_n$ and $b_n$ the exact values we would obtain for $u_n$ and $v_n$ respectively, without round-off errors. We have $s_1 = ab(1+\theta)$, $u_1 = a_1(1+\theta)^{3/2}$, $v_1 = b_1(1+\theta)$. Assume we can write $u_n = a_n(1+\theta)^\alpha$ and $v_n = (1+\theta)^\beta$ with $\alpha, \beta \leq e_n$. We thus can take $e_1 = 3/2$. Then as long as the **if**-condition is not satisfied, $v_{n+1} = b_{n+1}(1+\theta)^{e_n+1}$, and $u_{n+1} = a_{n+1}(1+\theta)^{e_n+3/2}$, which proves that $e_n \leq 3n/2$.

When the **if**-condition is satisfied, we have $\mathrm{EXP}(v_n - u_n) < \mathrm{EXP}(v_n) - p/4$, and since exponents are integers, thus $\mathrm{EXP}(v_n - u_n) \leq \mathrm{EXP}(v_n) - (p+1)/4$, i.e. $|v_n - u_n|/v_n < 2^{(3-p)/4}$.

Assume $n \leq 2^{p/4}$, which implies $3n|\theta|/2 \leq 1$, which since $n \geq 1$ implies in turn $|\theta| \leq 2/3$. Under that hypothesis, $(1+\theta)^{3n/2}$ can be written $1 + 3n\theta$ (possibly with a different $|\theta| \leq 2^{-p}$ as usual). Then $|b_n - a_n| = |v_n(1+3n\theta) - u_n(1+3n\theta')| \leq |v_n - u_n| + 3n|\theta|v_n$.

For $p \geq 4$, $3n|\theta| \leq 3/8$, and $1/(1+x)$ for $|x| \leq 3/8$ can be written $1 + 5x'/3$ for $x'$ in the same interval. This yields:

$$\begin{aligned}\frac{|b_n - a_n|}{b_n} &= \frac{|v_n - u_n| + 3n|\theta|v_n}{v_n(1+3n\theta)} \leq \frac{|v_n - u_n|}{v_n} + 5n|\theta|\frac{|v_n - u_n|}{v_n} + \frac{8}{5}(6n\theta) \\ &\leq \frac{13}{8} \cdot 2^{(3-p)/4} + \frac{48}{5} \cdot 2^{-3p/4} \leq 5.2 \cdot 2^{-p/4}.\end{aligned}$$

Write $a_n = b_n(1+\epsilon)$ with $|\epsilon| \leq 5.2 \cdot 2^{-p/4}$. We have $a_{n+1} = b_n\sqrt{1+\epsilon}$ and $b_{n+1} = b_n(1+\epsilon/2)$. Since $\sqrt{1+\epsilon} = 1 + \epsilon/2 - \frac{1}{8}\nu^2$ with $|\nu| \leq |\epsilon|$, we deduce that $|b_{n+1} - a_{n+1}| \leq \frac{1}{8}\nu^2|b_n| \leq 3.38 \cdot 2^{-p/2}b_n$. After one second iteration, we get similarly $|b_{n+2} - a_{n+2}| \leq \frac{1}{8}(3.38 \cdot 2^{-p/2})^2 b_n \leq \frac{3}{2}2^{-p}b_n$.

Let $q$ be the precision used to evaluate $s$, $t$ and $w$ in the **else** case. Since $|v_n - u_n| \leq 2^{(3-p)/4}v_n$, it follows $|s| \leq 1.8 \cdot 2^{-p/4}v_n$ for $q \geq 4$. Then $t \leq 0.22 \cdot 2^{-p/2}v_n$. Finally due to the above Lemma, the difference between $v_{n+1}$ and $v_n$ is less than that between $u_n$ and $v_n$, i.e. $\frac{v_n}{v_{n+1}} \leq \frac{1}{1-2^{(3-p)/4}} \leq 2$ for $p \geq 7$. We deduce $w \leq 0.22 \cdot 2^{-p/2}\frac{v_n^2}{v_{n+1}}(1+2^{-q}) \leq 0.47 \cdot 2^{-p/2}v_n \leq 0.94 \cdot 2^{-p/2}v_{n+1}$.

The total error is bounded by the sum of four terms:

- the difference between $a_{n+2}$ and $b_{n+2}$, bounded by $\frac{3}{2}2^{-p}b_n$;
- the difference between $b_{n+2}$ and $v_{n+2}$, if $v_{n+2}$ was computed directly without the final optimization; since $v_{n+2} = b_{n+2}(1+\theta)^{3(n+2)/2}$, if $n+2 \leq 2^{p/4}$, similarly as above, $(1+\theta)^{3(n+2)/2}$ can be written $1 + 3(n+2)\theta$, thus this difference is bounded by $3(n+2) \cdot 2^{-p}b_{n+2} \leq 3(n+2) \cdot 2^{-p}b_n$;
- the difference between $v_{n+2}$ computed directly, and with the final optimization. We can assume $v_{n+2}$ is computed directly in infinite precision, since we already took into account the rounding errors above. Thus we want to compute the difference between

  $$\frac{\sqrt{u_n v_n} + \frac{u_n+v_n}{2}}{2} \quad \text{and} \quad \frac{u_n+v_n}{2} - \frac{(v_n-u_n)^2}{8(u_n+v_n)}.$$

  Writing $u_n = v_n(1+\epsilon)$, this simplifies to:

  $$\frac{\sqrt{1+\epsilon} + 1 + \epsilon/2}{2} - \left(\frac{1+\epsilon/2}{2} - \frac{\epsilon^2}{18+8\epsilon}\right) = \frac{-1}{256}\epsilon^4 + O(\epsilon^5).$$

  For $|\epsilon| \leq 1/2$, the difference is bounded by $\frac{1}{100}\epsilon^4 v_n \leq \frac{1}{100}2^{3-p}v_n$.
- the round-off error on $w$, assuming $u_n$ and $v_n$ are exact; we can write $s = (v_n - u_n)(1+\theta)$, $t = \frac{1}{16}(v_n-u_n)^2(1+\theta)^2$, $w = \frac{(v_n-u_n)^2}{16v_{n+1}}(1+\theta)^4$. For $q \geq 4$, $(1+\theta)^4$ can be written $1+5\theta$, thus the round-off error on $w$ is bounded by $5\theta w \leq 4.7 \cdot 2^{-p/2-q}v_{n+1}$. For $q \geq p/2$, this gives a bound of $4.7 \cdot 2^{-p}v_{n+1}$.

Since $b_n = v_n(1+3n\theta)$, and we assumed $3n|\theta|/2 \leq 1$, we have $b_n \leq 3v_n$, thus the first two errors are less than $(9n + 45/2)2^{-p}v_n$; together with the third one, this gives a bound of $(9n+23)2^{-p}v_n$; finally since we proved above that $v_n \leq 2v_{n+1}$, this gives a total bound of $(18n+51)2^{-p}v_{n+1}$, which is less than $(18n+51)\mathrm{ulp}(r)$, or twice this in the improbable case where there is an exponent loss in the final subtraction $r = \circ(v_{n+1} - w)$.

| $w$ | $\mathrm{err}(w)/\mathrm{ulp}(w) \le c_w + \ldots$ | special case |
|---|---|---|
| $o(u+v)$ | $k_u 2^{e_u-e_w} + k_v 2^{e_v-e_w}$ | $k_u + k_v$ if $uv \ge 0$ |
| $o(u \cdot v)$ | $(1+c_u^+)k_u + (1+c_v^+)k_v$ | $2k_u + 2k_v$ if $u \ge x$, $v \ge y$ |
| $o(1/v)$ | $4k_v$ | $2k_v$ if $v \le y$ |
| $o(u/v)$ | $4k_u + 4k_v$ | $2k_u + 2k_v$ if $v \le y$ |
| $o(\sqrt{u})$ | $2k_u/(1+\sqrt{c_u^-})$ | $k_u$ if $u \le x$ |
| $o(e^u)$ | $e^{k_u 2^{e_u-p}} 2^{e_u+1} k_u$ | $2^{e_u+1} k_u$ if $u \ge x$ |
| $o(\log u)$ | $k_u 2^{1-e_w}$ | |

TABLE 1. Generic error for several operations, assuming all variables have a mantissa of $p$ bits, and no overflow/underflow occurs. The inputs $u$ and $v$ are approximations of $x$ and $y$ with $|u-x| \le k_u \mathrm{ulp}(u)$ and $|v-y| \le k_v \mathrm{ulp}(v)$. The additional rounding error $c_w$ is $1/2$ for rounding to nearest, and 1 otherwise. The value $c_u^\pm$ equals $1 \pm k_u 2^{1-p}$.

3.26. **Summary.** Remark 1: in the addition case, when $uv > 0$, necessarily one of $2^{e_u-e_w}$ and $2^{e_v-e_w}$ is less than 1/2, thus $\mathrm{err}(w)/\mathrm{ulp}(w) \le c_w + \max(k_u + k_v/2, k_u/2 + k_v) \le c_w + \frac{3}{2}\max(k_u, k_v)$.

## 4. Mathematical constants

4.1. **The constant $\pi$.** The computation of $\pi$ uses the AGM formula

$$\pi = \frac{\mu^2}{D},$$

where $\mu = \mathrm{AGM}(\frac{1}{\sqrt{2}}, 1)$ is the common limit of the sequences $a_0 = 1, b_0 = \frac{1}{\sqrt{2}}, a_{k+1} = (a_k + b_k)/2, b_{k+1} = \sqrt{a_k b_k}$, $D = \frac{1}{4} - \sum_{k=1}^{\infty} 2^{k-1}(a_k^2 - b_k^2)$. This formula can be evaluated efficiently as shown in [12], starting from $a_0 = A_0 = 1, B_0 = 1/2, D_0 = 1/4$, where $A_k$ and $B_k$ represent respectively $a_k^2$ and $b_k^2$:

```
$S_{k+1} \leftarrow (A_k + B_k)/4$
$b_k \leftarrow \sqrt{B_k}$
$a_{k+1} \leftarrow (a_k + b_k)/2$
$A_{k+1} \leftarrow a_k^2$
$B_{k+1} \leftarrow 2(A_{k+1} - S_{k+1})$
$D_{k+1} \leftarrow D_k - 2^k(A_{k+1} - B_{k+1})$
```

For each variable $x$ approximation a true value $\tilde{x}$, denote by $\epsilon(x)$ the exponent of the maximal error, i.e. $x = \tilde{x}(1 \pm \delta)^e$ with $|e| \le \epsilon(x)$, and $\delta = 2^{-p}$ for precision $p$ (we assume all roundings to nearest). We can prove by an easy induction that $\epsilon(a_k) = 3 \cdot 2^{k-1} - 1$, for $k \ge 1$, $\epsilon(A_k) = 3 \cdot 2^k - 1$, $\epsilon(B_k) = 6 \cdot 2^k - 6$. Thus the relative error on $B_k$ is at most $12 \cdot 2^{k-p}$, — assuming $12 \cdot 2^{k-p} \le 1$ — which is at most $12 \cdot 2^k \mathrm{ulp}(B_k)$, since $1/2 \le B_k$.

If we stop when $|A_k - B_k| \le 2^{k-p}$ where $p$ is the working precision, then $|\mu^2 - B_k| \le 13 \cdot 2^{k-p}$. The error on $D$ is bounded by $\sum_{j=0}^{k} 2^j(6 \cdot 2^{k-p} + 12 \cdot 2^{k-p}) \le 6 \cdot 2^{2k-p+2}$, which gives a relative error less than $2^{2k-p+7}$ since $D_k \ge 3/16$.

Thus we have $\pi = \frac{B_k(1+\epsilon)}{D(1+\epsilon')}$ with $|\epsilon| \le 13 \cdot 2^{k-p}$ and $|\epsilon'| \le 2^{2k-p+7}$. This gives $\pi = \frac{B_k}{D}(1+\epsilon'')$ with $|\epsilon''| \le 2\epsilon + \epsilon' \le (26 + 2^{k+7})2^{k-p} \le 2^{2k-p+8}$, assuming $|\epsilon'| \le 1$.

4.2. **Euler's constant.** Euler's constant is computed using the formula $\gamma = S(n) - R(n) - \log n$ where:

$$S(n) = \sum_{k=1}^{\infty} \frac{n^k(-1)^{k-1}}{k!k}, \quad R(n) = \int_n^{\infty} \frac{\exp(-u)}{u} du \sim \frac{\exp(-n)}{n} \sum_{k=0}^{\infty} \frac{k!}{(-n)^k}.$$

This identity is attributed to Sweeney by Brent [4]. We have $S(n) =_2 F_2(1,1;2,2;-n)$ and $R(n) = \mathrm{Ei}(1,n)$.

EVALUATION OF $S(n)$. As in [4], let $\alpha \sim 4.319136566$ the positive root of $\alpha + 2 = \alpha \log \alpha$, and $N = \lceil \alpha n \rceil$. We approximate $S(n)$ by

$$S'(n) = \sum_{k=1}^{N} \frac{n^k(-1)^{k-1}}{k!k}.$$

The remainder term $S(n) - S'(n)$ is bounded by:

$$|S(n) - S'(n)| \le \sum_{k=N+1}^{\infty} \frac{n^k}{k!}.$$

Since $k! \ge (k/e)^k$, and $k \ge N + 1 \ge \alpha n$, we have:

$$|S(n) - S'(n)| \le \sum_{k=N+1}^{\infty} \left(\frac{ne}{k}\right)^k \le \sum_{k=N+1}^{\infty} \left(\frac{e}{\alpha}\right)^k \le 2\left(\frac{e}{\alpha}\right)^N \le 2e^{-2n}$$

since $(e/\alpha)^\alpha = e^{-2}$.

To approximate $S'(n)$, we use the binary splitting method, which computes integers $T$ and $Q$ such that $S'(n) = \frac{T}{Q}$ exactly, then we compute $t = \circ(T)$, and $s = \circ(t/Q)$, both with rounding to nearest. If the working precision is $w$, we have $t = T(1+\theta_1)$ and $s = t/Q(1+\theta_2)$ where $|\theta_i| \le 2^{-w}$. If follows $s = T/Q(1+\theta_1)(1+\theta_2)$, thus the error on $s$ is less than 3 ulps, since $(1+2^{-w})^2 \le 1 + 3 \cdot 2^{-w}$.

EVALUATION OF $R(n)$. We estimate $R(n)$ using the terms up to $k = n - 2$, again as in [4]:

$$R'(n) = \frac{e^{-n}}{n} \sum_{k=0}^{n-2} \frac{k!}{(-n)^k}.$$

Let us introduce $I_k = \int_n^{\infty} \frac{e^{-u}}{u^k} du$. We have $R(n) = I_1$ and the recurrence $I_k = \frac{e^{-n}}{n^k} - kI_{k+1}$, which gives

$$R(n) = \frac{e^{-n}}{n} \sum_{k=0}^{n-2} \frac{k!}{(-n)^k} + (-1)^{n-1}(n-1)!I_n.$$

Bounding $n!$ by $(n/e)^n \sqrt{2\pi(n+1)}$ which holds[3] for $n \ge 1$, we have:

$$|R(n) - R'(n)| = (n-1)!I_n \le \frac{n!}{n} \int_n^{\infty} \frac{e^{-n}}{u^n} du \le \frac{n^{n-1}}{e^n} \sqrt{2\pi(n+1)} \frac{e^{-n}}{(n-1)n^{n-1}}$$

[3]Formula 6.1.38 from [1] gives $x! = \sqrt{2\pi} x^{x+1/2} e^{-x+\frac{\theta}{12x}}$ for $x > 0$ and $0 < \theta < 1$. Using it for $x \ge 1$, we have $0 < \frac{\theta}{6x} < 1$, and $e^t < 1 + 2t$ for $0 < t < 1$, thus $(x!)^2 \le 2\pi x^{2x} e^{-2x}(x + \frac{1}{3})$.

and since $\sqrt{2\pi(n+1)}/(n-1) \leq 1$ for $n \geq 9$:

$$|R(n) - R'(n)| \leq e^{-2n} \quad \text{for } n \geq 9.$$

Thus we have:

$$|\gamma - S'(n) - R'(n) - \log n| \leq 3e^{-2n} \quad \text{for } n \geq 9.$$

To approximate $R'(n)$, we use the following:

```
$m \leftarrow \mathrm{prec}(x) - \lfloor \frac{n}{\log 2} \rfloor$
$a \leftarrow 2^m$
$s \leftarrow 1$
**for** $k$ **from** 1 **to** $n$ **do**
    $a \leftarrow \lfloor \frac{ka}{n} \rfloor$
    $s \leftarrow s + (-1)^k a$
$t \leftarrow \lfloor s/n \rfloor$
$x \leftarrow t/2^m$
return $r = e^{-n} x$
```

The absolute error $\epsilon_k$ on $a$ at step $k$ satisfies $\epsilon_k \leq 1 + k/n\epsilon_{k-1}$ with $\epsilon_0 = 0$. As $k/n \leq 1$, we have $\epsilon_k \leq k$, whence the error on $s$ is bounded by $n(n+1)/2$, and that on $t$ by $1+(n+1)/2 \leq n+1$ since $n \geq 1$. The operation $x \leftarrow t/2^m$ is exact as soon as $\mathrm{prec}(x)$ is large enough, thus the error on $x$ is at most $(n+1)\frac{e^n}{2^{\mathrm{prec}(x)}}$. If $e^{-n}$ is computed with $m$ bits, then the error on it is at most $e^{-n}2^{1-m}$. The error on $r$ is then $(n+1+2/n)2^{-\mathrm{prec}(x)} + \mathrm{ulp}(r)$. Since $x \geq \frac{2}{3}n$ for $n \geq 2$, and $x2^{-\mathrm{prec}(x)} < \mathrm{ulp}(x)$, this gives an error bounded by $\mathrm{ulp}(r)+(n+1+2/n)\frac{3}{2n}\mathrm{ulp}(r) \leq 4\mathrm{ulp}(r)$ for $n \geq 2$ — if $\mathrm{prec}(x) = \mathrm{prec}(r)$. Now since $r \leq \frac{e^{-n}}{n} \leq \frac{1}{8}$, that error is less than $\mathrm{ulp}(1/2)$.

Final computation. We use the formula $\gamma = S(n) - R(n) - \log n$ with $n$ such that $e^{-2n} \leq \mathrm{ulp}(1/2) = 2^{-\mathrm{prec}}$, i.e. $n \geq \mathrm{prec}\frac{\log 2}{2}$:

```
$s \leftarrow S'(n)$
$l \leftarrow \log(n)$
$r \leftarrow R'(n)$
return $(s - l) - r$
```

Since the final result is in $[\frac{1}{2}, 1]$, and $R'(n) \leq \frac{e^{-n}}{n}$, then $S'(n)$ approximates $\log n$. If the target precision is $m$, and we use $m + \lceil \log_2(\mathrm{prec}) \rceil$ bits to evaluate $s$ and $l$, then the error on $s-l$ will be at most $3\mathrm{ulp}(1/2)$, so the error on $(s-l)-r$ is at most $5\mathrm{ulp}(1/2)$, and adding the $3e^{-2n}$ truncation error, we get a bound of $8\mathrm{ulp}(1/2)$. [**FIXME: check with new method to compute S**]

4.2.1. *A faster formula.* Brent and McMillan give in [5] a faster algorithm (B2) using the modified Bessel functions, which was used by Gourdon and Demichel to compute 108,000,000 digits of $\gamma$ in October 1999:

$$\gamma = \frac{S_0 - K_0}{I_0} - \log n,$$

where $S_0 = \sum_{k=1}^{\infty} \frac{n^{2k}}{(k!)^2} H_k$, $H_k = 1 + \frac{1}{2} + \cdots + \frac{1}{k}$ being the $k$-th harmonic number, $K_0 = \sqrt{\frac{\pi}{4n}} e^{-2n} \sum_{k=0}^{\infty} (-1)^k \frac{[(2k)!]^2}{(k!)^3(64n)^k}$, and $I_0 = \sum_{k=0}^{\infty} \frac{n^{2k}}{(k!)^2}$.

We have $I_0 \geq \frac{e^{2n}}{\sqrt{4\pi n}}$ (see [5] page 306). From the remark following formula 9.7.2 of [1], the remainder in the truncated expansion for $K_0$ up to $k$ does not exceed the $(k+1)$-th term, whence $K_0 \leq \sqrt{\frac{\pi}{4n}} e^{-2n}$ and $\frac{K_0}{I_0} \leq \pi e^{-4n}$ as in formula (5) of [5]. Let $I_0' = \sum_{k=0}^{K-1} \frac{n^{2k}}{(k!)^2}$ with $K = \lceil \beta n \rceil$, and $\beta$ is the root of $\beta(\log \beta - 1) = 3$ ($\beta \sim 4.971\ldots$) then

$$|I_0 - I_0'| \leq \frac{\beta}{2\pi(\beta^2 - 1)} \frac{e^{-6n}}{n}.$$

Let $K_0' = \sqrt{\frac{\pi}{4n}} e^{-2n} \sum_{k=0}^{4n-1} (-1)^k \frac{[(2k)!]^2}{(k!)^3 (64n)^k}$, then bounding by the next term:

$$|K_0 - K_0'| \leq \frac{(8n+1)}{16\sqrt{2}n} \frac{e^{-6n}}{n} \leq \frac{1}{2} \frac{e^{-6n}}{n}.$$

We get from this

$$\left| \frac{K_0}{I_0} - \frac{K_0'}{I_0'} \right| \leq \frac{1}{2I_0} \frac{e^{-6n}}{n} \leq \sqrt{\frac{\pi}{n}} e^{-8n}.$$

Let $S_0' = \sum_{k=1}^{K-1} \frac{n^{2k}}{(k!)^2} H_k$, then using $\frac{H_{k+1}}{H_k} \leq \frac{k+1}{k}$ and the same bound $K$ than for $I_0'$ ($4n \leq K \leq 5n$), we get:

$$|S_0 - S_0'| \leq \frac{\beta}{2\pi(\beta^2 - 1)} H_k \frac{e^{-6n}}{n}.$$

We deduce:

$$\left| \frac{S_0}{I_0} - \frac{S_0'}{I_0'} \right| \leq e^{-8n} H_K \frac{\sqrt{4\pi n}}{\pi(\beta^2 - 1)} \frac{\beta}{n} \leq e^{-8n}.$$

Hence we have

$$\left| \gamma - \left( \frac{S_0' - K_0'}{I_0'} - \log n \right) \right| \leq (1 + \sqrt{\frac{\pi}{n}}) e^{-8n} \leq 3e^{-8n}.$$

4.3. **The $\log 2$ constant.** This constant is used in the exponential function, and in the base 2 exponential and logarithm.

We use the following formula (formula (30) from [7]):

$$\log 2 = \frac{3}{4} \sum_{n \geq 0} (-1)^n \frac{n!^2}{2^n (2n+1)!}.$$

Let $w$ be the working precision. We take $N = \lfloor w/3 \rfloor + 1$, and evaluate exactly using binary spitting:

$$\frac{T}{Q} = \frac{3}{4} \sum_{n \geq 0}^{N-1} (-1)^n \frac{n!^2}{2^n (2n+1)!},$$

where $T$ and $Q$ are integers. Since the series has alternating signs with decreasing absolute values, the truncating error is bounded by the first neglected term, which is less than $2^{-3N-1}$ for $N \geq 2$; since $N \geq (w+1)/3$, this error is bounded by $2^{-w-2}$.

We then use the following algorithm:

```
$t \leftarrow \circ(T)$ [rounded to nearest]
$q \leftarrow \circ(Q)$ [rounded to nearest]
$x \leftarrow \circ(t/q)$ [rounded to nearest]
```

Using Higham's notation, we write $t = T(1+\theta_1)$, $q = Q(1+\theta_2)$, $x = t/q(1+\theta_3)$ with $|\theta_i| \leq 2^{-w}$. We thus have $x = T/Q(1+\theta)^3$ with $|\theta| \leq 2^{-w}$. Since $T/Q \leq 1$, the total absolute error on $x$ is thus bounded by $3|\theta| + 3\theta^2 + |\theta|^3 + 2^{-w-2} < 2^{-w+2}$ as long as $w \geq 3$.

4.4. **Catalan's constant.** Catalan's constant is defined by

$$G = \sum_{k=0}^{\infty} \frac{(-1)^k}{(2k+1)^2}.$$

We compute it using formula (31) of Victor Adamchik's document "33 representations for Catalan's constant"[4]:

$$G = \frac{\pi}{8}\log(2+\sqrt{3}) + \frac{3}{8}\sum_{k=0}^{\infty} \frac{(k!)^2}{(2k)!(2k+1)^2}.$$

Let $f(k) = \frac{(k!)^2}{(2k)!(2k+1)^2}$, and $S(0,K) = \sum_{k=0}^{K-1} f(k)$, and $S = S(0,\infty)$. We compute $S(0,K)$ exactly by binary splitting. Since $f(k)/f(k-1) = \frac{k(2k-1)}{2(2k+1)^2} \leq 1/4$, the truncation error on $S$ is bounded by $4/3f(K) \leq 4/3 \cdot 4^{-K}$. Since $S$ is multiplied by $3/8$, the corresponding contribution to the absolute error on $G$ is $2^{-2K-1}$. As long as $2K+1$ is greater or equal to the working precision $w$, this truncation error is less than one ulp of the final result.

```
$K \leftarrow \lceil \frac{w-1}{2} \rceil$
$T/Q \leftarrow S(0,K)$ [exact, rational]
$T \leftarrow 3T$ [exact, integer]
$t \leftarrow \circ(T)$ [up]
$q \leftarrow \circ(Q)$ [down]
$s \leftarrow \circ(t/q)$ [nearest]
$x \leftarrow \circ(\sqrt{3})$ [up]
$y \leftarrow \circ(2+x)$ [up]
$z \leftarrow \circ(\log y)$ [up]
$u \leftarrow \circ(\pi)$ [up]
$v \leftarrow \circ(uz)$ [nearest]
$g \leftarrow \circ(v+s)$ [nearest]
Return $g/8$ [exact].
```

The error on $t$ and $q$ is less than one ulp; using the generic error on the division, since $t \geq T$ and $q \leq Q$, the error on $s$ is at most $9/2$ ulps.

The error on $x$ is at most 1 ulp; since $1 < x < 2$ — assuming $w \geq 2$ —, $\mathrm{ulp}(x) = 1/2\mathrm{ulp}(y)$, thus the error on $y$ is at most $3/2\mathrm{ulp}(y)$. The generic error on the logarithm gives an error bound of $1 + \frac{3}{2} \cdot 2^{1-\mathrm{EXP}(z)} = \frac{5}{2}$ ulps for $z$ (since $3 \leq y < 4$, we have $1 \leq z < 2$, so $\mathrm{EXP}(z) = 1$). The error on $u$ is at most 1 ulp; thus using the generic error on the multiplication, since both $u$ and $z$ are upper approximations, the error on $v$ is at most 8 ulps. Finally that on $g$ is at most $8 + 9/2 = 25/2$ ulps. Taking into account the truncation error, this gives $27/2$ ulps.

## References

[1] Abramowitz, M., and Stegun, I. A. *Handbook of Mathematical Functions.* Dover, 1973. http://members.fortunecity.com/aands/toc.htm.

[4] http://www-2.cs.cmu.edu/~adamchik/articles/catalan/catalan.htm

[2] Borwein, J. M., and Borwein, P. B. *Pi and the AGM: A Study in Analytic Number Theory and Computational Complexity.* Wiley, 1998.

[3] Borwein, P. An efficient algorithm for the Riemann zeta function, Jan. 1995. 9 pages. Preprint available at `http://eprints.cecm.sfu.ca/archive/00000107/`.

[4] Brent, R. P. A Fortran multiple-precision arithmetic package. *ACM Trans. Math. Softw. 4*, 1 (1978), 57–70.

[5] Brent, R. P., and McMillan, E. M. Some new algorithms for high-precision computation of Euler's constant. *Mathematics of Computation 34*, 149 (1980), 305–312.

[6] Demmel, J., and Hida, Y. Accurate floating point summation. `http://www.cs.berkeley.edu/~demmel/AccurateSummation.ps`, May 2002.

[7] Gourdon, X., and Sebah, P. The logarithmic constant: $\log 2$, Jan. 2004.

[8] Graillat, S. *Fiabilité des algorithmes numériques : pseudosolutions structurées et précision.* PhD thesis, Université de Perpignan Via Domitia, 2005.

[9] Higham, N. J. *Accuracy and Stability of Numerical Algorithms*, 2nd ed. SIAM, 2002.

[10] Jeandel, E. Évaluation rapide de fonctions hypergéométriques. Rapport technique 242, Institut National de Recherche en Informatique et en Automatique, July 2000. 17 pages, `http://www.inria.fr/rrrt/rt-0242.html`.

[11] Pugh, G. R. *An Analysis of the Lanczos Gamma Approximation.* PhD thesis, University of British Columbia, 2004. `http://oldmill.uchicago.edu/~wilder/Code/gamma/docs/Pugh.pdf`.

[12] Schönhage, A., Grotefeld, A. F. W., and Vetter, E. *Fast Algorithms: A Multitape Turing Machine Implementation.* BI Wissenschaftverlag, 1994.

[13] Spouge, J. L. Computation of the gamma, digamma, and trigamma functions. *SIAM Journal on Numerical Analysis 31*, 3 (1994), 931–944.