THE MPFR LIBRARY: ALGORITHMS AND PROOFS

THE MPFR TEAM

Contents

https://www.mpfr.org/

1. Notations and Assumptions

In the whole document, $\mathcal{N}()$ denotes rounding to nearest, $\mathcal{Z}()$ rounding toward zero, $\triangle()$ rounding toward positive infinity, $\bigtriangledown()$ rounding toward negative infinity, and $\circ()$ any of those four rounding modes.

In the whole document, except special notice, all variables are assumed to have the same precision, usually denoted p.

2. Error calculus

Let p — 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 $\frac{1}{2} \leq |m| < 1$ and $e := \textsc{Exp}(x)$, and we define $\mathrm{ulp}(x) := 2^{\textsc{Exp}(x)-p}$. Any time a number appears in a ulp, it is implicitly assumed to be nonzero. Unless specified otherwise, when rounding is involved, one also assumes that no underflows nor overflows occur.

2.1. **Ulp calculus.**

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

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

Rule 2. *If a and b have same precision p, 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 $\mathrm{ulp}(a) = 2^{e_a - p} \le 2^{e_b - p} = \mathrm{ulp}(b)$. □

Rule 3. *For any $x \neq 0$ and any rounding mode $\circ(\cdot)$, we have* $\mathrm{ulp}(x) \le \mathrm{ulp}(\circ(x))$*, and equality holds when rounding toward zero, toward $-\infty$ for $x > 0$, or toward $+\infty$ for $x < 0$.*

Proof. Without loss of generality, assume $x > 0$. Since $x \in [2^{e-1}, 2^e[$ and both 2^{e-1} and 2^e are exactly representable, one has $\circ(x) \in [2^{e-1}, 2^e]$, and in rounding toward zero, $\circ(x) \in [2^{e-1}, 2^e[$. For any $x, y \in [2^{e-1}, 2^e[$, one has $\mathrm{ulp}(x) = \mathrm{ulp}(y) = \mathrm{ulp}(2^{e-1}) < \mathrm{ulp}(2^e)$. QED. □

Rule 4. *Let x be a real number, and $y = \circ(x)$. Then $|x-y| \le \frac{1}{2}\mathrm{ulp}(x) \le \frac{1}{2}\mathrm{ulp}(y)$ in rounding to nearest, and $|x - y| \le \mathrm{ulp}(x) \le \mathrm{ulp}(y)$ in the other rounding modes.*

Proof. In the binade $[2^{e-1}, 2^e]$, the distance between two consecutive machine numbers is $\mathrm{ulp}(x)$. Hence $|x - y| \le c \cdot \mathrm{ulp}(x)$ with $c = \frac{1}{2}$ in rounding to nearest and $c = 1$ in the other rounding modes. And $\mathrm{ulp}(x) \le \mathrm{ulp}(y)$ from Rule 3, which completes the proof. □

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

Proof. Let e be the exponent of a, i.e., $2^{e-1} \le |a| < 2^e$. Then $\frac{1}{2}|a| \cdot \mathrm{ulp}(b) < \frac{1}{2}2^e \cdot \mathrm{ulp}(b) = \mathrm{ulp}(2^{e-1}b) \le \mathrm{ulp}(ab)$. The other inequality is actually equivalent: replace a by $\frac{1}{a}$ and b by ab. □

Rule 7. *Let x be a nonzero real number, $\circ(\cdot)$ be any rounding, and $u := \circ(x)$. Then $\frac{1}{2}|u| < |x| < 2|u|$.*

Proof. Since no underflows nor overflows occur, u is finite and nonzero. Assume $|x| \ge 2|u|$. 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 8. $\frac{1}{2}|a| \cdot \mathrm{ulp}(1) < \mathrm{ulp}(a) \le |a| \cdot \mathrm{ulp}(1)$.

Proof. The left inequality comes from Rule 6 with $b = 1$, and the right one from $|a|\mathrm{ulp}(1) \ge \frac{1}{2}2^{e_a}2^{1-p} = \mathrm{ulp}(a)$. □

Rule 9.

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

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

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

Proof. This is a simple consequence of Rule 4. For rounding to nearest, we have $|u - x| \leq \frac{1}{2}\text{ulp}(t)$ for $t = u$ or $t = x$, hence by Rule 1 $|u - x| \leq 2^{-p}$. □

Rule 11. *Assume $x_1, \ldots, x_n$ are n floating-point numbers in precision p, and we compute an 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 $2(n-1)\text{ulp}(u_n)$.*

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 $0 \leq \theta_i \leq 2^{1-p}$. 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 $0 \leq \theta \leq 2^{1-p}$ by the intermediate value theorem. Since $1 - nt \leq (1-t)^n \leq 1$, we get $|x_1 x_2 \ldots x_n - u_n| \leq (n-1)2^{1-p}|u_n| \leq 2(n-1)\text{ulp}(u_n)$ by Rule 1. □

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

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

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

$$
\begin{aligned}
\text{error}(w) \ &\leq \ c_w \text{ulp}(w) + k_u \text{ulp}(u) + k_v \text{ulp}(v) \\
&= \ (c_w + k_u 2^d + k_v 2^{d'}) \text{ulp}(w)
\end{aligned}
$$

$$\text{If } \ (u \geq 0 \ \text{ and } \ v \geq 0) \ \text{ or } \ (u \leq 0 \ \text{ and } \ v \leq 0)$$

$$\text{error}(w) \ \leq \ (c_w + k_u + k_v)\,\text{ulp}(w)$$

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

2.4. **Generic error of multiplication.** We want to compute the generic error of the multiplication. We assume here $u, v > 0$ are approximations of exact values respectively x and y, with $|u - x| \leq k_u \, \mathrm{ulp}(u)$ and $|v - y| \leq k_v \, \mathrm{ulp}(v)$.

$$w = \circ(uv)$$

$$\begin{aligned}
\mathrm{error}(w) &= |w - xy| \\
&\leq |w - uv| + |uv - xy| \\
&\leq c_w \mathrm{ulp}(w) + \frac{1}{2}[|uv - uy| + |uy - xy| + |uv - xv| + |xv - xy|] \\
&\leq c_w \mathrm{ulp}(w) + \frac{u + x}{2} k_v \mathrm{ulp}(v) + \frac{v + y}{2} k_u \mathrm{ulp}(u) \\
&\leq c_w \mathrm{ulp}(w) + \frac{u(1 + c_u^+)}{2} k_v \mathrm{ulp}(v) + \frac{v(1 + c_v^+)}{2} k_u \mathrm{ulp}(u) \quad \text{[Rule 9]} \\
&\leq c_w \mathrm{ulp}(w) + (1 + c_u^+) k_v \mathrm{ulp}(uv) + (1 + c_v^+) k_u \mathrm{ulp}(uv) \quad \text{[Rule 6]} \\
&\leq [c_w + (1 + c_u^+) k_v + (1 + c_v^+) k_u] \mathrm{ulp}(w) \quad \text{[Rule 3]}
\end{aligned}$$

$$\text{Note:} \qquad \text{If } \; w = \mathcal{N}(uv) \; \text{ then } \; c_w = \frac{1}{2} \; \text{ else } \; c_w = 1$$

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

$$\begin{aligned}
w &= \circ(\frac{1}{u}) \\
\text{Note:} & \qquad \mathrm{error}(u) \leq k_u \, \mathrm{ulp}(u)
\end{aligned}$$

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

$$\begin{aligned}
\text{Note:} \quad & \frac{u}{c_u} \leq x \qquad \text{[Rule 7]} \\
& \text{for } u = \bigtriangledown(x), \; c_u = 1 \text{ else } c_u = 2 \\
& \text{then: } \frac{1}{x} \leq c_u \frac{1}{u}
\end{aligned}$$

$$
\begin{aligned}
\text{error}(w) &\leq c_w \text{ulp}(w) + c_u \frac{k_u}{u^2} \text{ulp}(u) \\
&\leq c_w \text{ulp}(w) + 2.c_u.k_u \text{ulp}(\frac{u}{u^2}) \quad [\text{Rule } 6] \\
&\leq [c_w + 2.c_u.k_u].\text{ulp}(w) \quad [\text{Rule } 3]
\end{aligned}
$$

$$
\text{Note:} \qquad \text{If } \; w = \mathcal{N}(\frac{1}{u}) \; \text{ then } \; c_w = \frac{1}{2} \; \text{ else } \; c_w = 1
$$

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

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

$$
\text{Note:} \qquad \text{error}(u) \leq k_u \, \text{ulp}(u), \;\; \text{error}(v) \leq k_v \, \text{ulp}(v)
$$

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

$$
\begin{aligned}
\text{Note:} \qquad & \frac{\text{ulp}(u)}{v} \leq 2\text{ulp}(\frac{u}{v}) \qquad [\text{Rule } 6] \\
& 2\text{ulp}(\frac{u}{v}) \leq 2\text{ulp}(w) \qquad [\text{Rule } 3]
\end{aligned}
$$

$$
\begin{aligned}
\text{Note:} \qquad & x \leq c_u u \text{ and } \frac{v}{c_v} \leq y \qquad [\text{Rule } 7] \\
& \text{with for } u = \triangle(x), \; c_u = 1 \text{ else } c_u = 2 \\
& \text{and for } v = \triangledown(y), \; c_v = 1 \text{ else } c_v = 2 \\
& \text{then: } \frac{x}{y} \leq c_u c_v \frac{u}{v}
\end{aligned}
$$

$$
\begin{aligned}
\text{error}(w) &\leq c_w \text{ulp}(w) + 2.k_u \text{ulp}(w) + c_u.c_v.\frac{k_v u}{vv} \text{ulp}(v) \\
&\leq c_w \text{ulp}(w) + 2.k_u \text{ulp}(w) + 2.c_u.c_v.k_v \text{ulp}(\frac{u.v}{v.v}) \quad [\text{Rule } 6] \\
&\leq [c_w + 2.k_u + 2.c_u.c_v.k_v].\text{ulp}(w) \quad [\text{Rule } 3]
\end{aligned}
$$

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

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 $w(|\theta_u| + |\theta_v|)$.

2.7. **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 $v = \circ(\sqrt{u})$, then:

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

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

$$\begin{aligned}
\mathrm{error}(v) &\leq c_v \mathrm{ulp}(v) + \frac{1}{\sqrt{u}.(1 + \sqrt{c_u^-})} k_u \mathrm{ulp}(u) \\
&\leq c_v \mathrm{ulp}(v) + \frac{2}{1 + \sqrt{c_u^-}} k_u \mathrm{ulp}(\sqrt{u}) \qquad \text{[Rule 6]} \\
&\leq (c_v + \frac{2k_u}{1 + \sqrt{c_u^-}}) \mathrm{ulp}(v). \qquad \text{[Rule 3]}
\end{aligned}$$

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

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

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

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

$$\begin{aligned}
\mathrm{error}(v) &\leq c_v \mathrm{ulp}(v) + c_u^* e^u k_u \mathrm{ulp}(u) \\
&\leq c_v \mathrm{ulp}(v) + 2 c_u^* u k_u \mathrm{ulp}(e^u) \qquad \text{[Rule 6]} \\
&\leq (c_v + 2 c_u^* u k_u) \mathrm{ulp}(v) \qquad \text{[Rule 3]} \\
&\leq (c_v + c_u^* 2^{\mathrm{Exp}(u)+1} k_u) \mathrm{ulp}(v)
\end{aligned}$$

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

2.9. **Generic error of the logarithm.** Assume x and u are positive values, with $|u-x| \le k_u \mathrm{ulp}(u)$. We additionally assume $u \le 4x$. Let $v = \circ(\log u)$.

$$\begin{aligned}
\mathrm{error}(v) &= |v - \log x| \le |v - \log u| + |\log u - \log x| \\
&\le c_v \mathrm{ulp}(v) + |\log \frac{x}{u}| \le c_v \mathrm{ulp}(v) + 2\frac{|x-u|}{u} \\
&\le c_v \mathrm{ulp}(v) + \frac{2k_u\, \mathrm{ulp}(u)}{u} \le c_v \mathrm{ulp}(v) + 2k_u\, \mathrm{ulp}(1) \qquad \text{[Rule 8]} \\
&\le c_v \mathrm{ulp}(v) + 2k_u\, 2^{1-e_v} \mathrm{ulp}(v) \le (c_v + k_u 2^{2-e_v}) \mathrm{ulp}(v).
\end{aligned}$$

We used at line 2 the inequality $|\log t| \le 2|t-1|$ which holds for $t \ge \rho$, where $\rho \approx 0.203$ satisfies $\log \rho = 2(\rho - 1)$. At line 4, e_v stands for the exponent of v, i.e, $v = m \cdot 2^{e_v}$ with $1/2 \le |m| < 1$.

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

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

Respectively, if the relative error is $\le \delta$, then the error is at most $\delta |u| \le \delta 2^p \mathrm{ulp}(u)$. (Going from the ulp-error to the relative error and back, we lose 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 [13] proposes the following framework: we associate to each variable the cumulated number k of roundings that were made. The ith rounding introduces a relative error of δ_i, with $|\delta_i| \le 2^{1-p}$, 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 $\varepsilon = 2^{1-p}$. In particular, if all roundings are away, the final relative error is at most $k\varepsilon = k \cdot 2^{1-p}$, thus at most $2k$ ulps.

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

If the rounding are not away from zero, the following lemma is still useful [13, Lemma 3.1]:

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

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

The same holds if some terms $1+\delta_i$ are replaced by $1/(1+\delta_i)$.

Proof. The maximum values of θ are obtained when all the δ_i are ϵ, or all are $-\epsilon$, thus it suffices to prove

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

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

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

Now assume some of the terms $1+\delta$ are replaced by $1/(1+\delta)$. The worst case is when $1/(1+\delta) = 1/(1-\epsilon)$ or $1/(1+\epsilon)$. If $1/(1+\delta) = 1/(1+\epsilon)$, we can write $1/(1+\delta) = 1-\delta'$ with $|\delta'| = \epsilon/(1+\epsilon) < \epsilon$, thus this is covered by the previous proof. If $1/(1+\delta) = 1/(1-\epsilon)$, it suffices to prove that $1/(1-\epsilon)^n \le 1/(1-n\epsilon)$, i.e., that $(1-\epsilon)^n \ge 1-n\epsilon$, which is true. □

3. Low level functions

3.1. 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
```

3.2. The mpfr_cmp2 function.

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

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 $r \ge s$, 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 nth 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 += GMP_NUMB_BITS;

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

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

/* now dif > 1 */

res += GMP_NUMB_BITS - 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]);
```

3.3. **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), $\mathcal{Z}$ (toward zero), or ∞ (away from zero).

Algorithm `mpfr_sub`.
Input: b, c of same 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 $\circ(b-c) < b-c$
$\mathtt{an} \leftarrow \lceil \frac{\mathrm{prec}(a)}{w} \rceil$, $\mathtt{bn} \leftarrow \lceil \frac{\mathrm{prec}(b)}{w} \rceil$, $\mathtt{cn} \leftarrow \lceil \frac{\mathrm{prec}(c)}{w} \rceil$
$\mathtt{cancel} \leftarrow \mathtt{mpfr\_cmp2}(b, c)$; $\quad \mathtt{diff\_exp} \leftarrow \mathrm{Exp}(b) - \mathrm{Exp}(c)$
$\mathtt{shift_b} \leftarrow (-\mathtt{cancel}) \bmod w$; $\quad \mathtt{cancel_b} \leftarrow (\mathtt{cancel} + \mathtt{shift_b})/w$
if $\mathtt{shift_b} > 0$ **then** $\mathtt{b}[0\ldots\mathtt{bn}] \leftarrow \mathtt{mpn\_rshift}(\mathtt{b}[0\ldots\mathtt{bn}-1], \mathtt{shift_b})$; $\mathtt{bn} \leftarrow \mathtt{bn} + 1$
$\mathtt{shift_c} \leftarrow (\mathtt{diff\_exp} - \mathtt{cancel}) \bmod w$; $\quad \mathtt{cancel_c} \leftarrow (\mathtt{cancel} + \mathtt{shift_c} - \mathtt{diff\_exp})/w$
if $\mathtt{shift_c} > 0$ **then** $\mathtt{c}[0\ldots\mathtt{cn}] \leftarrow \mathtt{mpn\_rshift}(\mathtt{c}[0\ldots\mathtt{cn}-1], \mathtt{shift_c})$; $\mathtt{cn} \leftarrow \mathtt{cn} + 1$
$\mathrm{Exp}(a) \leftarrow \mathrm{Exp}(b) - \mathtt{cancel}$; $\quad \mathrm{sign}(a) \leftarrow \mathrm{sign}(b)$
$a[0\ldots\mathtt{an}-1] \leftarrow b[\mathtt{bn} - \mathtt{cancel_b} - \mathtt{an} \ldots \mathtt{bn} - \mathtt{cancel_b} - 1]$
$a[0\ldots\mathtt{an}-1] \leftarrow a[0\ldots\mathtt{an}-1] - c[\mathtt{cn} - \mathtt{cancel_c} - \mathtt{an} \ldots \mathtt{cn} - \mathtt{cancel_c} - 1]$

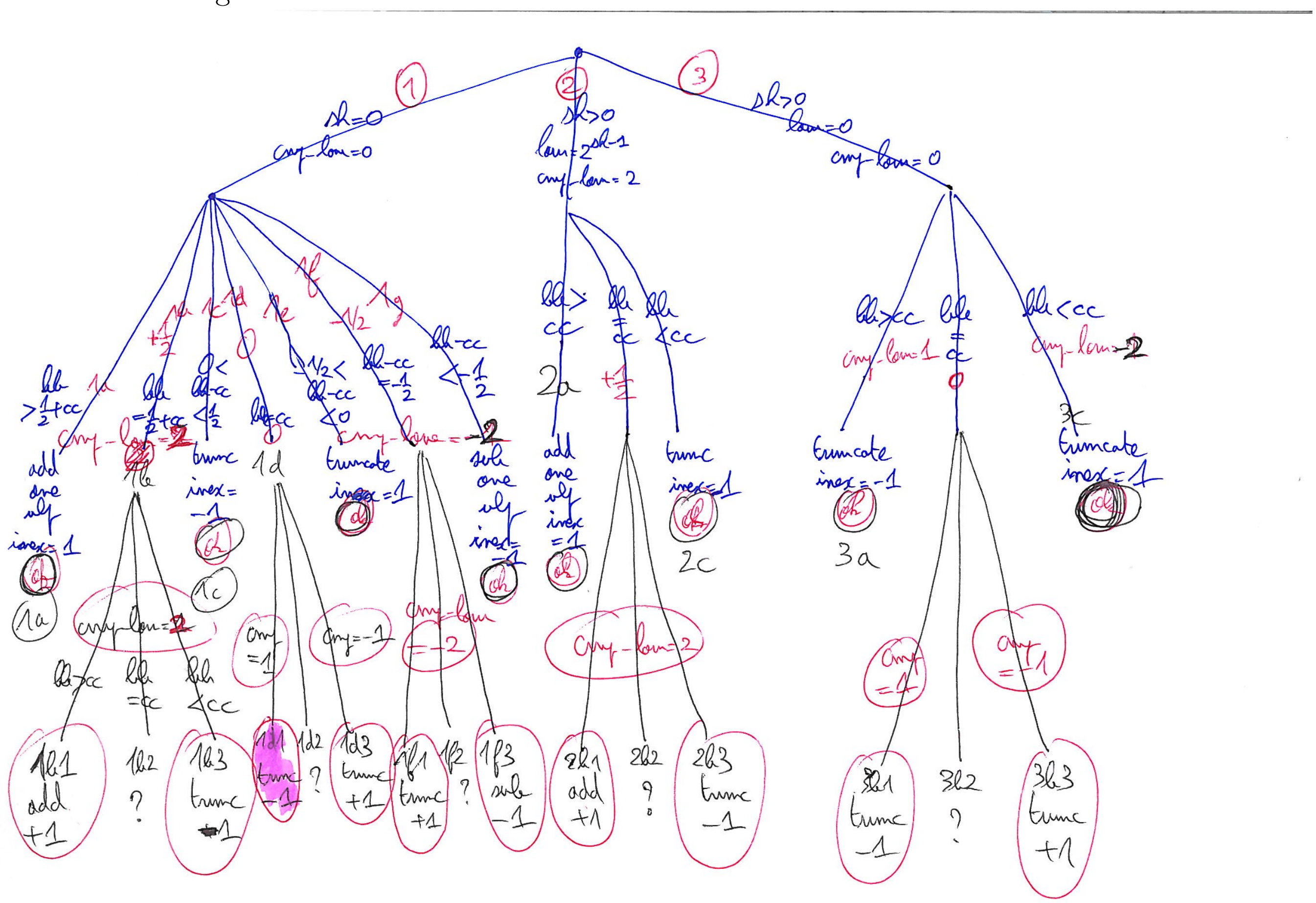
$$\texttt{sh} \leftarrow \texttt{an} \cdot w - \text{prec}(a); \quad r \leftarrow a[0] \bmod 2^{\texttt{sh}}; \quad a[0] \leftarrow a[0] - r$$

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

The rounding is determined by a left-to-right subtraction of the neglected limb of b and c, until one is able to determine the correct rounding *and* the correct ternary value. After the above algorithm, there are three cases where one cannot conclude:

(1) if $\texttt{sh} = 0$, since the low part of $b - c$ can have any value between -1 ulp and 1 ulp. The result might be $a - 1$, a or $a + 1$;
(2) if $\texttt{sh} > 0$ and $r = 2^{\texttt{sh}-1}$: the result might be a or $a + 1$;
(3) if $\texttt{sh} > 0$ and $r = 0$: the result is always a, but we cannot determine the ternary value.

In those three cases we look at the most significant neglected limbs from b and c until we can conclude. In case 1 the first limb is special, since it will rule out one of the possible results $a-1$, a or $a+1$. Up from the second limb, the analysis is invariant. The corresponding tree is the following:



3.4. **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: $1/2 \le b < 1$ and $1/2 \le c < 1$, $b = bh + bl$ and $c = ch + cl$ ($B = 2^{32} or 2^{64}$). The error comes from:

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

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

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

3.5. **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 $n \ge p$ 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.

3.5.1. *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 $0 \le u_0 < 1$. Since v_1 has n significant bits, we have $2^{n-1} \le v_1 < 2^n$. (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 $2^{2n-2} \le u_1 < 2^{2n}$.) The integer division of u_1 by v_1 yields q_1 and r such that $u_1 = q_1 v_1 + r$, with $0 \le r < v_1$, and q_1 having exactly n bits. In that case we have

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

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

3.5.2. *Truncated divisor.* This is the hard case. Write $u = u_1 + u_0$, and $v = v_1 + v_0$, where $0 \le u_0, v_0 < 1$ 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 $u - (q_1 - 2)v > u_1 - (q_1 - 2)(v_1 + 1) \ge q_1 v_1 - (q_1 - 2)(v_1 + 1) = 2(v_1 + 1) - q_1 \ge 2^n - q_1 > 0$. 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}.

3.5.3. *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 $q \in (q_1 - 2, q_1 - 1)$

3.5.4. *Using Mulders' short division.* For larger operands, Mulders' short division might be faster than calling GMP's integer division. A detailed description of Mulders' short division for integers can be found in [12]. We assume that we want the quotient integer significant on $n - 1$ limbs, and we perform a short division on n limbs. Let q be the real quotient u/v, scaled so that it has exactly n limbs; let q_1 be the integer division we would perform using GMP's integer division as described above, and let q_2 be the approximate quotient returned by Algorithm `ShortDiv` or `FoldDiv` from [12]. From the above analysis, we know that $q_1 - 2 < q < q_1 + 1$, the divisor being truncated or not. From Theorems 1 and 2 from [12], we have $q_1 - 2n \le q_2 \le q_1 + 2n$. It thus follows:

$$q_1 - (2n + 2) < q < q_2 + (2n + 1),$$

and in all cases the difference between q and q_2 is less than $2n + 2$ ulps (on n limbs). Since we want to round q on $n - 1$ limbs, and usually $2n + 2$ is small compared to the limb value, in most cases we will be able to round correctly.

In the rare cases where we are not able to round correctly, we can either revert to the above method using integer division, or better use the approximate quotient q_2 to deduce the exact quotient q_1 and the corresponding remainder, which will trade a division for a multiplication.

3.6. **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 toward 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.

3.7. **The inverse square root.** The inverse square root (function `mpfr_rec_sqrt`) is based on Ziv's strategy and the `mpfr_mpn_rec_sqrt` function, which given a precision p, and an input $1 \le a < 4$, returns an approximation x satisfying

$$x - \frac{1}{2} \cdot 2^{-p} \le a^{-1/2} \le x + 2^{-p}.$$

The `mpfr_mpn_rec_sqrt` function is based on Newton's iteration and the following lemma, the proof of which can be found in [7]:

Lemma 3. *Let $A, x > 0$, and $x' = x + \frac{x}{2}(1 - Ax^2)$. Then*

$$0 \le A^{-1/2} - x' = \frac{3}{2}\frac{x^3}{\theta^4}(A^{-1/2} - x)^2,$$

for some $\theta \in (x, A^{-1/2})$.

We first describe the recursive iteration:

Algorithm `ApproximateInverseSquareRoot`.
Input: $1 \le a, A < 4$ and $1/2 \le x < 1$ with $x - \frac{1}{2} \cdot 2^{-h} \le a^{-1/2} \le x + 2^{-h}$
Output: X with $X - \frac{1}{2} \cdot 2^{-n} \le A^{-1/2} \le X + 2^{-n}$, where $n \le 2h - 3$
  $r \leftarrow x^2$ [exact]
  $s \leftarrow Ar$ [exact]

$t \leftarrow 1 - s$ [rounded at weight 2^{-2h} toward $-\infty$]
$u \leftarrow xt$ [exact]
$X \leftarrow x + u/2$ [rounded at weight 2^{-n} to nearest]

Lemma 4. *If $h \geq 11$, $0 \leq A - a < 2^{-h}$, then the output X of algorithm* `ApproximateInverseSquareRoot` *satisfies*

$$X - \frac{1}{2} \cdot 2^{-n} \leq A^{-1/2} \leq X + 2^{-n}. \tag{2}$$

Proof. Firstly, $a \leq A < a + 2^{-h}$ yields $a^{-1/2} - \frac{1}{2} \cdot 2^{-h} \leq A^{-1/2} \leq a^{-1/2}$, thus $x - 2^{-h} \leq A^{-1/2} \leq x + 2^{-h}$.

Lemma 3 implies that the value x' that would return Algorithm `ApproximateInverseSquareRoot` if there was no rounding error satisfies $0 \leq A^{-1/2} - x' = \frac{3}{2}\frac{x^3}{\theta^4}(A^{-1/2} - x)^2$. Since $\theta \in (x, A^{-1/2})$, and $A^{-1/2} \leq x + 2^{-h}$, we have $x \leq \theta + 2^{-h}$, which yields $\frac{x^3}{\theta^3} \leq (1 + \frac{2^{-h}}{\theta})^3 \leq (1 + 2^{-10})^3 \leq 1.003$ since $\theta \geq 1/2$ and $h \geq 11$. Thus $0 \leq A^{-1/2} - x' \leq 3.01 \cdot 2^{-2h}$.

Finally the errors while rounding $1-s$ and $x+u/2$ in the algorithm yield $\frac{1}{2}\cdot 2^{-n} \leq x' - X \leq \frac{1}{2} \cdot 2^{-n} + \frac{1}{2} \cdot 2^{-2h}$, thus the final inequality is:

$$\frac{1}{2} \cdot 2^{-n} \leq A^{-1/2} - X \leq \frac{1}{2} \cdot 2^{-n} + 3.51 \cdot 2^{-2h}.$$

For $2h \geq n + 3$, we have $3.51 \cdot 2^{-2h} \leq \frac{1}{2} \cdot 2^{-n}$, which concludes the proof. □

The initial approximation is obtained using a bipartite table for $h = 11$. More precisely, we split a 13-bit input $a = a_1 a_0.a_{-1} \ldots a_{-11}$ into three parts of 5, 4 and 4 bits respectively, say α, β, γ, and we deduce a 11-bit approximation $x = 0.x_{-1}x_{-2} \ldots x_{-11}$ of the form $T_1[\alpha, \beta] + T_2[\alpha, \gamma]$, where both tables have 384 entries each. Those tables satisfy:

$$x + (\frac{1}{4} - \varepsilon)2^{-11} \leq a^{-1/2} \leq x + (\frac{1}{4} + \varepsilon)2^{-11},$$

with $\varepsilon \leq 1.061$. Note that this does not fulfill the initial condition of Algorithm `ApproximateInverseSquareRoot`, since we have $x - 0.811 \cdot 2^{-h} \leq a^{-1/2} \leq x + 1.311 \cdot 2^{-h}$, which yields $X - \frac{1}{2} \cdot 2^{-n} \leq A^{-1/2} \leq X + 1.21 \cdot 2^{-n}$, thus the right bound is not a priori fulfilled. However, the only problematic case is $n = 19$, which gives exactly $(n + 3)/2 = 11$, since for $12 \leq n \leq 18$, the error terms in 2^{-2h} are halved. An exhaustive search of all possible inputs for $h = 11$ and $n = 19$ gives

$$X - \frac{1}{2} \cdot 2^{-n} \leq A^{-1/2} \leq X + 0.998 \cdot 2^{-n},$$

the worst case being $A = 1990149, X = 269098$ (scaled by 2^{19}). Thus as soon as $n \geq 2$, Eq. (2) is fulfilled.

In summary, Algorithm `ApproximateInverseSquareRoot` provides an approximation X of $A^{-1/2}$ with an error of at most one ulp. However, if the input A was itself truncated at precision $\geq p$ from an input A_0 — for example when the output precision p is less than the input precision — then we have $|X - A^{-1/2}| \leq \mathrm{ulp}(X)$, and $|A^{-1/2} - A_0^{-1/2}| \leq \frac{1}{2}|A - A_0|A^{-3/2} \leq \frac{1}{2}\frac{|A-A_0|}{A}A^{-1/2} \leq 2^{-p}A^{-1/2} \leq \mathrm{ulp}(X)$, thus $|X - A_0^{-1/2}| \leq 2\,\mathrm{ulp}(X)$.

3.8. **The `mpfr_remainder` and `mpfr_remquo` functions.** The `mpfr_remainder` and `mpfr_remquo` are useful functions for argument reduction. Given two floating-point numbers x and y, `mpfr_remainder` computes the correct rounding of $x \operatorname{cmod} y := x - qy$, where $q = \lfloor x/y \rceil$, with ties rounded to the nearest even integer, as in the rounding to nearest mode.

Additionally, `mpfr_remquo` returns a value congruent to q modulo 2^n, where n is a small integer (say $n \le 64$, see the documentation), and having the same sign as q or being zero. This can be efficiently implemented by calling `mpfr_remainder` on x and $2^n y$. Indeed, if $x = r' \operatorname{cmod} (2^n y)$, and $r' = q'y + r$ with $|r| \le y/2$, then $q \equiv q' \mod 2^n$. No double-rounding problem can occur, since if $x/(2^n y) \in \mathbb{Z} + 1/2$, then $r' = \pm 2^{n-1} y$, thus $q' = \pm 2^{n-1}$ and $r = 0$.

Whatever the input x and y, it should be noted that if $\mathrm{ulp}(x) \ge \mathrm{ulp}(y)$, then $x - qy$ is always exactly representable in the precision of y unless its exponent is smaller than the minimum exponent. To see this, let $\mathrm{ulp}(y) = 2^{-k}$; multiplying x and y by 2^k we get $X = 2^k x$ and $Y = 2^k y$ such that $\mathrm{ulp}(Y) = 1$, and $\mathrm{ulp}(X) \ge \mathrm{ulp}(Y)$, thus both X and Y are integers. Now perform the division of X by Y, with quotient rounded to nearest: $X = qY + R$, with $|R| \le Y/2$. Since R is an integer, it is necessarily representable with the precision of Y, and thus of y. The quotient q of x/y is the same as that of X/Y, the remainder $x - qy$ is $2^{-k} R$.

We assume without loss of generality that $x, y > 0$, and that $\mathrm{ulp}(y) = 1$, i.e., y is an integer.

Algorithm Remainder.
Input: x, y with $\mathrm{ulp}(y) = 1$, a rounding mode $\circ$
Output: $x \operatorname{cmod} y$, rounded according to $\circ$
1. If $\mathrm{ulp}(x) < 1$, decompose x into $x_h + x_l$ with $\mathrm{ulp}(x_h) \ge 1$ and $0 \le x_l < 1$.
1a. $r \leftarrow \mathrm{Remainder}(x_h, y)$ [exact, $-y/2 \le r \le y/2$]
1b. if $r < y/2$ or $x_l = 0$ then return $\circ(r + x_l)$
1c. else return $\circ(r + x_l - y) = \circ(x_l - r)$
2. Write $x = m \cdot 2^k$ with $k \ge 0$
3. $z \leftarrow 2^k \mod y$ [binary exponentiation]
4. Return $\circ(mz \operatorname{cmod} y)$.

Note: at step (1a) the auxiliary variable r has the precision of y; since x_h and y are integers, so is r and the result is exact by the above reasoning. At step (1c) we have $r = y/2$, thus $r - y$ simplifies to $-r$.

4. High level functions

4.1. **The cosine function.** To evaluate $\cos x$ with a target precision of n bits, we use the following algorithm with working precision m, after an additive argument reduction which reduces x in the interval $[-\pi, \pi]$, using the `mpfr_remainder` function:

$k \leftarrow \lfloor \sqrt{n/2} \rfloor$
$r \leftarrow x^2$ rounded up
$r \leftarrow r/2^{2k}$
$s \leftarrow 1, t \leftarrow 1$
for l **from** 1 **while** $\mathrm{Exp}(t) \ge -m$
  $t \leftarrow t \cdot r$ 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 1ulp(r) and remains 1ulp(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 $3l$ulp(t). Hence as long as $3l$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 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 ϵ_i the maximal error on s after the ith step in the do-loop, we have $\epsilon_0 = 2l_0 2^{-m}$ and $\epsilon_{k+1} \leq 4\epsilon_k + 2^{-m}$, giving $\epsilon_k \leq (2l_0 + 1/3)2^{2k-m}$.

4.2. **The sine function.** The sine function is computed from the cosine, with a working precision of m bits, after an additive argument reduction in $[-\pi, \pi]$:

$c \leftarrow \cos x$ 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 ulp() $\leq 2^{-m}$, all absolute errors are less than 2^{-m}. We denote by ϵ_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 $2^{-m}(\frac{5}{2}2^{1-e} + 1) \leq 2^{3-m-e}$.

4.2.1. *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}.

4.3. **The tangent function.** The tangent function is computed from the `mpfr_sin_cos` function, which computes simultaneously $\sin x$ and $\cos x$ with a working precision of m bits:

$s, c \leftarrow \circ(\sin x), \circ(\cos x)$ [to nearest]
$t \leftarrow \circ(s/c)$ [to nearest]

We have $s = \sin(x)(1+\theta_1)$ and $c = \cos(x)(1+\theta_2)$ with $|\theta_1|, |\theta_2| \leq 2^{-m}$, thus $t = (\tan x)(1+\theta)^3$ with $|\theta| \leq 2^{-m}$. For $m \geq 2$, $|\theta| \leq 1/4$, $|(1+\theta)^3 - 1| \leq 4|\theta|$, thus we can write $t = (\tan x)(1 + 4\theta)$, thus $|t - \tan x| \leq 4\mathrm{ulp}(t)$.

4.4. **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 (see [15]). 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 $0 \leq r < \log 2$, 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 $\mathcal{O}(l^{1/2})$ nonscalar multiplications — i.e., with both operands of full n-bit size — as described in [19], 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`. (Note: the algorithm from Paterson and Stockmeyer was rediscovered by Smith, who named it "concurrent series" in [24].)

4.5. **The logarithm function.** The logarithm function `mpfr_log` is defined using this approximated formula [18] based on the arithmetic-geometric mean (denoted by AG):

$$\log x \approx \frac{\pi}{2\ \mathrm{AG}(1{,}4/\mathrm{s})} - m \log 2 + o(\log x\ 2^{-p})$$

with $s = x \cdot 2^m > 2^{p/2}$.

From the arithmetic-geometric mean, we deduce the logarithm, naively bounding the round-off errors. The only point is that the subtraction may be a cancellation: a maximum of $\log\left(\frac{p \log 2}{|x-1|}\right)$ bits can be lost.

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

4.6.1. *Taylor expansion.*

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

`erf_0`(z, n), assumes $z^2 \leq n/e$
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 $\textsc{Exp}(u) < \textsc{Exp}(s) - m$ and $k \geq z^2$ **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 ε_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 $\varepsilon_k \leq 4k$.

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

Let σ_k and ν_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 $\sigma_k := \text{EXP}(s_{k-1}) - \text{EXP}(s_k)$ and $\nu_k := \text{EXP}(u_k) - \text{EXP}(s_k)$. The ulp-error τ_k on s_k satisfies $\tau_k \le \frac{1}{2} + \tau_{k-1}2^{\sigma_k} + (1+8k)2^{\nu_k}$.

The halting condition $k \ge z^2$ ensures that $u_j \le u_{j-1}$ for $j \ge k$, thus the series $\sum_{j=k}^{\infty} u_j$ is an alternating series, and the truncated part is bounded by its first term $|u_k| < \text{ulp}(s_k)$. 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).

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

More precisely, [1, formulæ 7.1.23 and 7.1.24] gives:

$$\sqrt{\pi} x e^{x^2} \operatorname{erfc} x \approx 1 + \sum_{k=1}^{n} (-1)^k \frac{1 \times 3 \times \cdots \times (2k-1)}{(2x^2)^k},$$

with the error bounded in absolute value by the next term and of the same sign.

4.7. **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]:

$$\begin{array}{lrcl}
 & u & \leftarrow & \circ(e^x) \\
(3) & v & \leftarrow & \circ(u^{-1}) \\
(4) & w & \leftarrow & \circ(u+v) \\
(5) & s & \leftarrow & \frac{1}{2}w \\
(6) & & &
\end{array}$$

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

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

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

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

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

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

$(\star)$
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 6],

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

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)
$w \leftarrow \circ(u + v)$

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

$(\star)$
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)
$s \leftarrow \circ(\frac{w}{2})$

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

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.

4.8. **The inverse hyperbolic cosine function.** The `mpfr_acosh` function implements the inverse hyperbolic cosine. For $x < 1$, it returns NaN; for $x = 1$, $\operatorname{acosh} x = 0$; for $x > 1$, the formula $\operatorname{acosh} x = \log(\sqrt{x^2 - 1} + x)$ is implemented 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]

Let us first assume that $r \neq 0$. The error on q is at most $1\,\mathrm{ulp}(q)$, thus that on r is at most $\mathrm{ulp}(r) + \mathrm{ulp}(q) = (1 + E)\,\mathrm{ulp}(r)$ with $d = \mathrm{Exp}(q) - \mathrm{Exp}(r)$ and $E = 2^d$. Since r is smaller than

$x^2 - 1$, we can use the simpler formula for the error on the square root, which gives a bound $(\frac{3}{2} + E)\,\mathrm{ulp}(s)$ for the error on s, and $(2 + E)\,\mathrm{ulp}(t)$ for that on t. This gives a final bound of $(\frac{1}{2} + (2+E)2^{2-\mathrm{Exp}(u)})\,\mathrm{ulp}(u)$ for the error on u (§2.9). We have: $2 + E \le 2^{1+\max(1,d)}$. Thus the rounding error on the calculation of $\operatorname{acosh} x$ can be bounded by $(\frac{1}{2} + 2^{3+\max(1,d)-\mathrm{Exp}(u)})\,\mathrm{ulp}(u)$.

If we obtain $r = 0$, which means that x is near from 1, we need another algorithm. One has $x = 1 + z$, with $0 < z < 2^{-p}$, where p is the intermediate precision (which may be smaller than the precision of x). The formula can be rewritten: $\operatorname{acosh} x = \log(1 + \sqrt{z(2+z)} + z) = \sqrt{2z}(1 - \varepsilon(z))$ where $0 < \varepsilon(z) < z/12$. We use the following algorithm:

$q \leftarrow \circ(x - 1)$ [down]
$r \leftarrow 2q$
$s \leftarrow \circ(\sqrt{r})$ [nearest]

The error on q is at most $1\,\mathrm{ulp}(q)$, thus the error on r is at most $1\,\mathrm{ulp}(r)$. Since r is smaller than $2z$, we can use the simpler formula for the error on the square root, which gives a bound $\frac{3}{2}\,\mathrm{ulp}(s)$ for the error on s. The error on $\operatorname{acosh} x$ is bounded by the sum of the error bound on $\sqrt{2z}$ and $\varepsilon(z)\sqrt{2z} < \frac{2^{-p}}{12} 2^{1+\mathrm{Exp}(s)} = \frac{1}{6}\,\mathrm{ulp}(s)$. Thus the rounding error on the calculation of $\operatorname{acosh} x$ can be bounded by $\left(\frac{3}{2} + \frac{1}{6}\right)\,\mathrm{ulp}(s) < 2\,\mathrm{ulp}(s)$.

4.9. **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]:

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

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 $x \ge 0$. We can deduce $e^x \ge 1$ and $0 \le e^{-x} \le 1$.

[2] $\circ()$ represent the rounding error and $\mathrm{error}(u)$ the error associated with the calculation of u

error(u) $u \leftarrow \bigtriangledown(e^x)$ ($\bullet$)	$\|u - e^x\| \leq \mathrm{ulp}(u)$	
error(v) $v \leftarrow \bigtriangleup(u^{-1})$ ($\bullet\bullet$)	$\|v - e^{-x}\|$ $\leq \|v - u^{-1}\| + \|u^{-1} - e^{-x}\|$ $\leq \mathrm{ulp}(v) + \frac{1}{u \cdot e^x}\|u - e^x\|$ $\leq \mathrm{ulp}(v) + \frac{1}{u^2}\mathrm{ulp}(u) \quad (\star)$ $\leq \mathrm{ulp}(v) + 2\mathrm{ulp}(\frac{1}{u}) \quad (\star\star)$ $\leq 3\,\mathrm{ulp}(v) \quad (\star\star\star)$	($\star$) 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 6], $a \cdot \mathrm{ulp}(b) \leq 2 \cdot \mathrm{ulp}(a \cdot b)$ 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) $w \leftarrow \circ(u - v)$	$\|w - (e^x - e^{-x})\|$ $\leq \|w - (u - v)\| + \|u - e^x\| + \|-v + e^{-x}\|$ $\leq \mathrm{ulp}(w) + \mathrm{ulp}(u) + 3\mathrm{ulp}(v)$ $\leq \mathrm{ulp}(w) + 4\mathrm{ulp}(u) \quad (\star)$ $\leq (1 + 4 \cdot 2^{\mathrm{Exp}(u) - \mathrm{Exp}(w)})\mathrm{ulp}(w) \quad (\star\star)$	($\star$) With $v \leq 1 \leq u$ then $\mathrm{ulp}(v) \leq \mathrm{ulp}(u)$ ($\star\star$) see subsection 2.3
error(s) $s \leftarrow \circ(\frac{w}{2})$	$\mathrm{error}(s) = \mathrm{error}(w)$ $\leq (1 + 4 \cdot 2^{\mathrm{Exp}(u) - \mathrm{Exp}(w)})\mathrm{ulp}(w)$	

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 :

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

4.10. **The inverse hyperbolic sine function.** The `mpfr_asinh` (asinh$x$) 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

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

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) $s \leftarrow \circ(x^2)$	$\|s - x^2\|$ $\leq$ ulp(s) $(\star)$	
error(t) $t \leftarrow \bigtriangledown(s+1)$ $(\bullet)$	$\|t - (x^2+1)\|$ $\leq$ 2ulp(t) $(\star)$	$(\star)$ see subsection 2.3
error(u) $u \leftarrow \circ(\sqrt{t})$	$\|u - \sqrt{x^2+1}\|$ $\leq$ 3ulp(u) $(\star)$	$(\star)$ see subsection 2.7 with $(\bullet)$
error(v) $v \leftarrow \circ(u+x)$	$\|v - (\sqrt{x^2+1}+x)\|$ $\leq$ 5ulp(v) $(\star)$	$(\star)$ see subsection 2.3
error(w) $w \leftarrow \circ(\log v)$	$\|w - \log(\sqrt{x^2+1}+x)\|$ $\leq$ $(1+5.2^{2-\text{Exp}(w)})$ulp($w$) $\star$	$(\star)$ see subsection 2.9

That shows the rounding error on the calculation of asinh$x$ can be bound by $(1 + 5.2^{2-\text{Exp}(w)})$ ulp on the result. So, to calculate the size of intermediary variables, we have to add, at least, $\lceil \log_2(1+5.2^{2-\text{Exp}(w)}) \rceil$ bits the wanted precision.

4.11. **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 θ_i denoting variables such that $|\theta_i| \leq 2^{-p}$. Firstly, u is exact, assuming x is exact with precision p. 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 $\mathrm{ulp}(v) = 2^k \mathrm{ulp}(r)$, with $k \geq 0$; then the error on r is bounded by $\frac{1}{2}(2^k+1)\mathrm{ulp}(r)$. We can thus write $r = (e^{2x}-1)(1+\theta_3)^{2^k+1}$, and then $s = \tanh(x) \cdot (1+\theta_4)^{2^k+4}$.

Lemma 5. *For $0 < x \leq 1/2$ and $0 < y \leq x^{-1/2}$, we have:*

$$0 < (1+x)^y - 1 \leq 1.4 \cdot y \cdot x.$$

Proof. We have $(1+x)^y = e^{y \cdot \log(1+x)}$, with $0 < y \cdot \log(1+x) \leq x^{-1/2} \cdot \log(1+x)$. The function $x^{-1/2} \cdot \log(1+x)$ is increasing on $]0, 1/2]$, and reaches ≈ 0.573 for $x = 1/2$. Thus $0 < y \cdot \log(1+x) < 0.574$. Now it is easy to see that for $0 < t < 0.574$, we have $|e^t - 1| \leq 1.4\,t$. Thus $0 < (1+x)^y - 1 \leq 1.4 \cdot y \cdot \log(1+x)$. The result follows from $\log(1+x) \leq x$ for $0 < x \leq 1/2$. ☐

First, note that for $t > 0$ and $q \geq 1$, one has $|(1-t)^q - 1| \leq |(1+t)^q - 1|$ due to the triangle inequality on the development. Thus $|(1+\theta_4)^{2^k+4} - 1| \leq |(1+2^{-p})^{2^k+4} - 1|$. Then one can apply the above lemma for $x = 2^{-p}$ and $y = 2^k + 4$, assuming $2^k + 4 \leq 2^{p/2}$. We get $|(1+\theta_4)^{2^k+4} - 1| \leq |(1+2^{-p})^{2^k+4} - 1| \leq 1.4(2^k+4)2^{-p}$, and thus we can write $s = \tanh(x)[1 + 1.4(2^k+4)\theta_5]$ with $|\theta_5| \leq 2^{-p}$. Since $2^k + 4 \leq 2^{\max(3,k+1)}$, the relative error on s is thus bounded by $2^{\max(4,k+2)-p}$.

The condition $2^k + 4 \leq 2^{p/2}$ is checked in the code with $\max(3, k+1) \leq \lfloor p/2 \rfloor$, so that:

$$2^k + 4 \leq 2^{\max(3,k+1)} \leq 2^{\lfloor p/2 \rfloor} \leq 2^{p/2}.$$

4.12. **The inverse hyperbolic tangent function.** The `mpfr_atanh` (atanh$x$) 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:

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

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

error(s) $s \leftarrow \triangle(1+x)$ ($\bullet$)	$|s-(1+x)| \leq 2\mathrm{ulp}(s)$ ($\star$)	see subsection 2.3
error(t) $t \leftarrow \triangledown(1-x)$ ($\bullet\bullet$)	$|t-(1-x)| \leq (1+2^{\mathrm{Exp}(x)-\mathrm{Exp}(t)})\mathrm{ulp}(t)$ ($\star$)	($\star$) see subsection 2.3
error(u) $u \leftarrow \circ(\frac{s}{t})$	$|u-\dfrac{1+x}{1-x}| \leq (1+2\times 2+ \cdots 2\times(1+2^{\mathrm{Exp}(x)-\mathrm{Exp}(t)}))\mathrm{ulp}u$ ($\star$) $\leq (7+2^{\mathrm{Exp}(x)-\mathrm{Exp}(t)+1})\mathrm{ulp}(u)$	($\star$) see subsection 2.5 with ($\bullet$) and ($\bullet\bullet$)
error(v) $v \leftarrow \circ(\log(u))$	$|v-(\log\dfrac{1+x}{1-x})| \leq (1+(7+2^{\mathrm{Exp}(x)-\mathrm{Exp}(t)+1}) \cdots \times 2^{2-\mathrm{Exp}(v)})\mathrm{ulp}(v)$ ($\star$) $\leq (1+7\times 2^{2-\mathrm{Exp}(v)}+ \cdots 2^{\mathrm{Exp}(x)-\mathrm{Exp}(t)-\mathrm{Exp}(v)+3})\mathrm{ulp}(v)$	($\star$) see subsection 2.9
error(w) $w \leftarrow \circ(\frac{1}{2}v)$	$|w-\dfrac{1}{2}\log\dfrac{1+x}{1-x}| \leq (1+7\times 2^{2-\mathrm{Exp}(v)}+ \cdots 2^{\mathrm{Exp}(x)-\mathrm{Exp}(t)-\mathrm{Exp}(v)+3})\mathrm{ulp}(w)$ $\star$	($\star$) exact

That shows the rounding error on the calculation of atanh$x$ can be bound by $(1+7\times 2^{2-\mathrm{Exp}(v)}+2^{\mathrm{Exp}(x)-\mathrm{Exp}(t)-\mathrm{Exp}(v)+3})$ ulp on the result. So, to calculate the size of intermediary variables, we have to add, at least, $\lceil\log_2(1+7\times 2^{2-\mathrm{Exp}(v)}+2^{\mathrm{Exp}(x)-\mathrm{Exp}(t)-\mathrm{Exp}(v)+3})\rceil$ bits the wanted precision.

4.13. **The arc-sine function.**

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

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

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

4.14. **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)
```

4.15. **The arc-tangent function.** The arc-tangent function admits the following argument reduction:

$$\arctan x = 2 \arctan \frac{x}{1+\sqrt{1+x^2}} = 2 \arctan \frac{\sqrt{1+x^2}-1}{x}.$$

If applied once, it reduces the argument to $|x| < 1$, then each successive application reduces x by a factor of at least 2.

Assume $|x| \leq 1$. We approximate $\frac{\sqrt{1+x^2}-1}{x}$ using the following algorithm:

$s \leftarrow \circ(x^2)$ [nearest]
$t \leftarrow \circ(1+s)$ [nearest]
$u \leftarrow \circ(\sqrt{t})$ [nearest]
$v \leftarrow \circ(u-1))$ [nearest]
$w \leftarrow \circ(\frac{v}{x})$ [nearest]

Assuming all computations are done with precision p, and denoting θ_i a value such that $|\theta_i| \leq 2^{-p}$, we have $s = x^2(1+\theta_1)$, $t = (1+s)(1+\theta_2) = (1+x^2(1+\theta_1))(1+\theta_2) = (1+x^2)(1+\theta_3)^2$, $u = \sqrt{t}(1+\theta_4) = \sqrt{1+x^2}(1+\theta_3)(1+\theta_4) = \sqrt{1+x^2}(1+\theta_5)^2$. Now let us write $u-1 = (\sqrt{1+x^2}-1)(1+\lambda)$; we have

$$\lambda = \frac{\sqrt{1+x^2}}{\sqrt{1+x^2}-1}(2\theta_5+\theta_5^2).$$

For $|x| \leq 1$, we have $2/x^2 \leq \frac{\sqrt{1+x^2}}{\sqrt{1+x^2}-1} \leq (2+\sqrt{2})/x^2$, and for $p \geq 5$ the expression $(2+\sqrt{2})(2\theta_5+\theta_5^2)$ is bounded by $7\theta_5$, thus we can write $u-1 = (\sqrt{1+x^2}-1)(1+7\theta_5/x)$. It follows $v = (u-1)(1+\theta_6)$ and $w = \frac{\sqrt{1+x^2}-1}{x}(1+7\theta_5/x)(1+\theta_7)^2$. Still for $|x| \leq 1$ and $p \geq 5$, the product $(1+7\theta_5/x)(1+\theta_7)^2$ can be written $(1+10\theta_8/x)$, thus we have $w = \frac{\sqrt{1+x^2}-1}{x}(1+10\theta_8/x)$, and the relative error on w is bounded by $10/x \cdot 2^{-p}$.

Now if we want to apply several times this argument reduction, we need to analyze the error when x is not exact, but say $x = \bar{x}(1+\varepsilon\theta)$, with $\varepsilon \geq 0$ a parameter, and $|\theta| \leq 2^{-p}$. The above analysis remains valid with x replaced by $\bar{x}(1+\varepsilon\theta)$, thus we get $w = f(x)(1+10\theta_8/x)$, with $f(x) = \frac{\sqrt{1+x^2}-1}{x}$, and $x = \bar{x}(1+\varepsilon\theta)$. The derivative of f is bounded by $1/2$ for $|x| \leq 1$, thus $|f(x) - f(\bar{x})| \leq \frac{1}{2}\varepsilon\bar{x}$. Thus we have

$$w = \frac{\sqrt{1+\bar{x}^2}-1}{\bar{x}}(1+\frac{\varepsilon\theta_9}{2})(1+10\theta_8/x).$$

Assuming $\varepsilon \leq 21/x$, we have $w = \frac{\sqrt{1+\bar{x}^2}-1}{\bar{x}}(1+\frac{21}{2}\theta_{10}/x)(1+10\theta_8/x) = \frac{\sqrt{1+\bar{x}^2}-1}{\bar{x}}(1+21\theta_{11}/x)$, as long as $|x| \geq 210 \cdot 2^{-p}$. Since initially $\varepsilon = 0$, this proves by induction that $\varepsilon \leq 21/x$, and the relative error on w after several argument reductions is bounded by $21/x \cdot 2^{-p}$, where x is the last reduced argument. Since the arc-tangent function has a derivative less than 1, the corresponding absolute error bound for $\arctan w$ is also $21/x \cdot 2^{-p}$. Since $\arctan \frac{\sqrt{1+x^2}-1}{x} \geq 5x/16$ for $0 \leq x \leq 1$, the relative error bound on $\arctan \frac{\sqrt{1+x^2}-1}{x}$ is $2^{7-p}/x^2 \leq 2^{9-2\mathrm{Exp}(x)-p}$.

4.15.1. *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

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

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

Writing

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

or

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

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

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

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

$$\begin{aligned} \arctan x &= \arctan A_0 + \arctan s_1 \\ &= \arctan A_0 + \arctan A_1 + \arctan s_2 \\ &= \arctan A_0 + \cdots + \arctan A_N + \arctan s_{N+1} \end{aligned}$$

the number s_N is decreasing toward 0 and we then have

$$\arctan x = \textstyle\sum_{i=0}^{i=\infty} \arctan A_i$$

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 in 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 $\arctan(1/2)$.

4.15.2. *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

$$s_{k+1} = \frac{s_k - A_k}{1 + s_k A_k} \text{ and } s_0 = x$$

(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

$$\begin{aligned} \arctan x &= \arctan A_0 + \arctan s_1 \\ &= \arctan A_0 + \arctan A_1 + \arctan s_2 \\ &= \arctan A_0 + \cdots + \arctan A_N + \arctan s_{N+1} \end{aligned}$$

the number s_N is very rapidly decreasing toward 0 and we then have

$$\arctan x = \sum_{i=0}^{i=\infty} \arctan A_i$$

(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

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

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

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

With $c_{N_0} \leq 1.64$. If $N_0 \geq 1$ then $c_{N_0} \leq 1.27$. If $N_0 \geq 2$ then $c_{N_0} \leq 1.07$. It remains to determine the right 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

$$\begin{aligned}
\frac{1}{2\times 2^{m_i}+1}\left[\frac{1}{2^{2^{i-1}}}\right]^{2\times 2^{m_i}+1} &= \frac{1}{2\times 2^{m_i}+1}\left[2^{-2^{i-1}}\right]^{2\times 2^{m_i}+1} \\
&\leq \frac{1}{2\times 2^{m_i}+1}\left[2^{-2^{i-1}}\right]^{2\times 2^{m_i}} \\
&\leq \frac{1}{2\times 2^{m_i}+1} 2^{-2^{i+m_i}}
\end{aligned}$$

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 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 yields 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)

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

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

$$\frac{2.4}{2^{2^{N_0}}} \leq 2^{-prec\_arctan-1}$$

and

$$(3 \times N_0 - 2)2^{-Prec} \leq 2^{-prec\_arctan-1}$$

i.e.

$$N_0 \geq \frac{\ln\left(prec\_arctan + 1 + \frac{\ln 2.4}{\ln 2}\right)}{\ln 2}$$

that we approach by (since the logarithm is expensive):

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

and we finally have:

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

4.15.3. *Estimate of the relative error.* we say that a and b have relative error δ 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 all previous estimates:
  (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 $\frac{1}{\text{arctan } A_i} \frac{1}{2 \times 2^{m_i} + 1} 2^{-2^{i+m_i}}$.
  (c) The multiplication arctan $A_i = A_i[\frac{\text{arctan } A_i}{A_i}]$ takes 1 ulp of relative error.
  (d) Doing the subtraction $s_i - A_i$ is a gradual underflow operation: it decreases the precision of $s_i - A_i$.
  (e) The multiplication $a_i A_i$ creates 1 ulp of error. This is not much and this relative error is further reduced by adding 1.

(1) We have
$$\begin{aligned} \text{arctan } b(1 + \Delta) &= \text{arctan}(b + b\Delta) \\ &\sim \text{arctan } b + \tfrac{1}{1+b^2}(b\Delta) \\ &= [\text{arctan } b][1 + \{\tfrac{b}{(1+b^2)\text{ arctan } b}\}\Delta] \end{aligned}$$
A rapid analysis gives $0 \leq \frac{b}{(1+b^2)\text{ arctan } b} \leq 1$ 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 $|b|\delta$ (δ 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 $|\text{arctan } x|$ 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:

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

4.15.4. *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.

4.16. **The Euclidean distance function.** The `mpfr_hypot` function implements the Euclidean distance function:

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

If one of the variables is zero, then hypot is computed using the absolute value of the other variable. Assume that $0 < y \le x$. Using the first degree Taylor polynomial, we have:

$$0 < \sqrt{x^2 + y^2} - x < \frac{y^2}{2x}.$$

Let p_x, p_y be the precisions of the input variables x and y respectively, p_z the output precision and $z = \circ_{p_z}(\sqrt{x^2+y^2})$ the expected result. Let us assume, as it is the case in MPFR, that the minimal and maximal acceptable exponents (respectively e_{min} and e_{max}) satisfy $2 < e_{max}$ and $e_{max} = -e_{min}$.

When rounding to nearest, if $p_x \le p_z$ and $\frac{p_z+1}{2} < \mathrm{Exp}(x) - \mathrm{Exp}(y)$, we have $\frac{y^2}{2x} < \frac{1}{2}\mathrm{ulp}_{p_z}(x)$; if $p_z < p_x$, the condition $\frac{p_x+1}{2} < \mathrm{Exp}(x) - \mathrm{Exp}(y)$ ensures that $\frac{y^2}{2x} < \frac{1}{2}\mathrm{ulp}_{p_x}(x)$. In both cases, these inequalities show that $z = \mathcal{N}_{p_z}(x)$, except that tie case is rounded toward positive infinity since hypot(x,y) is strictly greater than x.

With the other rounding modes, the conditions $p_z/2 < \mathrm{Exp}(x) - \mathrm{Exp}(y)$ if $p_x \le p_z$, and $p_x/2 < \mathrm{Exp}(x) - \mathrm{Exp}(y)$ if $p_z < p_x$ mean in a similar way that $z = \circ_{p_z}(x)$, except that we need to add one ulp to the result when rounding toward positive infinity and x is exactly representable with p_z bits of precision.

When none of the above conditions are satisfied, we use the following algorithm, whose precision is guaranteed when $\mathrm{Exp}(x) - \mathrm{Exp}(y) \le e_{max} - 1$:

Algorithm `hypot_1`
Input: x and y with $|y| \le |x|$, p the working precision with $p \ge p_z$.
Output: $\sqrt{x^2+y^2}$ with $\begin{cases} p-4 \text{ bits of precision if } p < \max(p_x, p_y), \\ p-2 \text{ bits of precision if } p \ge \max(p_x, p_y). \end{cases}$
$s \leftarrow \lfloor (e_{max} - 1)/2 \rfloor - \mathrm{Exp}(x)$
$x_s \leftarrow \mathcal{Z}(x \times 2^s)$
$y_s \leftarrow \mathcal{Z}(y \times 2^s)$
$u \leftarrow \mathcal{Z}(x_s^2)$
$v \leftarrow \mathcal{Z}(y_s^2)$
$w \leftarrow \mathcal{Z}(u + v)$
$t \leftarrow \mathcal{Z}(\sqrt{w})$
$z \leftarrow \mathcal{Z}(t/2^s)$

In order to avoid undue overflow during computation, we shift inputs' exponents by $s = \lfloor \frac{e_{max}-1}{2} \rfloor - \mathrm{Exp}(x)$ before computing squares and shift back the output's exponent by $-s$ using the fact that $\sqrt{(x.2^s)^2 + (y.2^s)^2}/2^s = \sqrt{x^2+y^2}$. We show below that overflow cannot occur, and underflow cannot occur either when $\mathrm{Exp}(x) - \mathrm{Exp}(y) \le e_{max} - 1$.

We check first that the exponent shift does not cause overflow and, in the same time, that the squares of the shifted inputs never overflow. For x, we have $\mathrm{Exp}(x) + s = \lfloor (e_{max}-1)/2 \rfloor$, so $\mathrm{Exp}(x_s^2) \le e_{max} - 1$ and neither x_s nor x_s^2 overflows. Therefore we have: $\mathcal{Z}(x_s^2) \le x_s^2 < 2^{e_{max}-1}$. For y, note that we have $y_s \le x_s$ because $y \le x$, thus y_s and y_s^2 do not overflow.

Secondly, let us see that the exponent shift does not cause underflow. For x, we know that $0 \le \mathrm{Exp}(x) + s$, thus neither x_s nor x_s^2 underflows. For y, the condition $\mathrm{Exp}(x) - \mathrm{Exp}(y) \le e_{max} - 1$ implies that $\lfloor (e_{max}-1)/2 \rfloor - s - \mathrm{Exp}(y) \le e_{max} - 1$, and since $e_{max}/2 - 1 \le \lfloor (e_{max}-1)/2 \rfloor$, we deduce $e_{min}/2 = -e_{max}/2 \le \mathrm{Exp}(y) + s$, which shows that y_s and its square do not underflow (even when taking the rounding into account since the scaling is exact).

Thirdly, the addition does not overflow because $u + v < 2x_s^2$ and it was shown above that $\mathcal{Z}(x_s^2) < 2^{e_{max}-1}$. It cannot underflow because both operands are positive.

Fourthly, as $x_s < t$, the square root does not underflow. Due to the exponent shift, we have $1 \le x_s$, then w is greater than 1 and thus greater than its square root t, so the square root does overflow.

Finally, let us show that the back shift raises neither underflow nor overflow unless the exact result is greater than or equal to $2^{e_{max}}$. Because no underflow has occurred so far $\mathrm{Exp}(x) \le \mathrm{Exp}(t) - s$ which shows that it does not underflow. And all roundings being toward zero, we have $z \le \sqrt{x^2+y^2}$, so if $2^{e_{max}} \le z$, then the exact value is also greater than or equal to $2^{e_{max}}$.

Let us analyse now the error of the algorithm hypot_1:

			$p < \min(p_x, p_y)$	$\max(p_x, p_y) \le p$
$x_s \leftarrow \mathcal{Z}(x \times 2^s)$	$\mathrm{error}(x_s)$	$\le$	1 ulp(x_s)	exact
$y_s \leftarrow \mathcal{Z}(y \times 2^s)$	$\mathrm{error}(y_s)$	$\le$	1 ulp(y_s)	exact
$u \leftarrow \mathcal{Z}(x_s^2)$	$\mathrm{error}(u)$	$\le$	6 ulp(u)	1 ulp(u)
$v \leftarrow \mathcal{Z}(y_s^2)$	$\mathrm{error}(v)$	$\le$	6 ulp(v)	1 ulp(v)
$w \leftarrow \mathcal{Z}(u+v)$	$\mathrm{error}(w)$	$\le$	13 ulp(w)	3 ulp(w)
$t \leftarrow \mathcal{Z}(\sqrt{w})$	$\mathrm{error}(t)$	$\le$	14 ulp(t)	4 ulp(t)
$z \leftarrow \mathcal{Z}(t/2^s)$	exact.			

And in the intermediate case, if $\min(p_x, p_y) \le p < \max(p_x, p_y)$, we have

$w \leftarrow \mathcal{Z}(u+v)$ $\quad$ $\mathrm{error}(w) \le 8$ ulp(w)

$t \leftarrow \mathcal{Z}(\sqrt{w})$ $\quad$ $\mathrm{error}(t) \le 9$ ulp(t).

Thus, 2 bits of precision are lost when $\max(p_x, p_y) \le p$ and 4 bits when p does not satisfy this relation.

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

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

The algorithm used for this calculation is as follows:

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

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

error(u) $u \leftarrow \circ(x \times y)$	$\|u - (xy)\| \leq ulp(u)$	
error(v) $v \leftarrow \circ(z + u)$	$\|v - (z + xy)\| \leq \mathrm{ulp}(v) + \|(z + u) - (z + xy)\|$ $\leq (1 + 2^{e_u - e_v})\mathrm{ulp}(v)$ $(\star)$	$(\star)$ see subsection 2.3

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.

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

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

The algorithm used for this calculation is as follows:

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

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

error(u) $u \leftarrow \circ(e^x)$	$\|u - e^x\| \leq ulp(u)$	
error(v) $v \leftarrow \circ(u - 1)$	$\|v - (e^x - 1)\| \leq (1 + 2^{e_u - e_v})\mathrm{ulp}(v)$ $(\star)$	$(\star)$ see subsection 2.3

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.

4.19. **The log1p function.** The `mpfr_log1p` function implements the log1p function as:

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

We could use the argument reduction

$$\mathrm{log1p}(x) = 2\mathrm{log1p}\left(\frac{x}{1 + \sqrt{1 + x}}\right),$$

which reduces x to about $\sqrt{x}$ when $x \gg 1$, and in any case to less than $x/2$ when $x > 0$. However, if $1+x$ can be computed exactly with the target precision, then it is more efficient to directly call the logarithm, which has its own argument reduction. If $1+x$ cannot be computed exactly, this implies that x is either very small, in which case no argument reduction is needed, or very large, in which case $\mathrm{log1p}(x) \approx \log x$.

The algorithm used for this calculation is as follows (with rounding to nearest):

$$
\begin{aligned}
v &\leftarrow \circ(1+x) \\
w &\leftarrow \circ(\log v)
\end{aligned}
$$

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

error(v) $v \leftarrow \circ(1+x)$	$\lvert v-(1+x)\rvert \leq \frac{1}{2}\mathrm{ulp}(v)$	
error(w) $w \leftarrow \circ(\log v)$	$\lvert w-\log(1+x)\rvert \leq (\frac{1}{2}+2^{1-e_w})\mathrm{ulp}(w)$	($\star$) see subsection 2.9

The 2^{1-e_w} factor in the error reflects the possible loss of accuracy in $1+x$ when x is small. Note that if $v=\circ(1+x)$ is exact, then the error bound simplifies to $2^{1-e_w}\mathrm{ulp}(w)$, i.e., 2^{1-p}, where p is the working precision.

4.20. **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:

$$
\begin{aligned}
u &\leftarrow \circ(\log(x)) \\
v &\leftarrow \circ(\log(t)) \\
w &\leftarrow \circ(\frac{u}{v})
\end{aligned}
$$

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)
$u \leftarrow \triangle(\log(x))$ $(\bullet)$ $\quad |u - \log(x)| \leq \mathrm{ulp}(u)$

error(v)
$v \leftarrow \triangledown(\log t)$ $(\bullet\bullet)$ $\quad |v - \log t| \leq \mathrm{ulp}(v)$

error(w)
$w \leftarrow \circ(\frac{u}{v})$ $\quad |v - (\frac{\log x}{\log t})| \leq 5\mathrm{ulp}(w)$ $(\star)$ $\quad (\star)$ see subsection 2.6

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.

4.21. **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:

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

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)
$u \leftarrow \circ(\log(x))$ $\quad |u - \log(x)| \leq \mathrm{ulp}(u)$ $\star$

error(v)
$v \leftarrow \triangle(y \times u)$ $(\bullet)$

$$\begin{aligned} |v - y\log(x)| &\leq \mathrm{ulp}(v) + |yu - y\log(x)| && \\ &\leq \mathrm{ulp}(v) + y|u - \log(x)| && (\star) \text{ with [Rule 6]} \\ &\leq \mathrm{ulp}(v) + y\mathrm{ulp}(u) && (\star) \text{ with [Rule 3]} \\ &\leq \mathrm{ulp}(v) + 2\mathrm{ulp}(yu) \ (\star) && \\ &\leq 3\mathrm{ulp}(v) \ (\star\star) && \end{aligned}$$

error(w)
$w \leftarrow \circ(e^v)$ $\quad |w - e^v| \leq (1 + 3 \cdot 2^{\mathrm{Exp}(v)+1})\mathrm{ulp}(w)$ $\quad (\star)$ see subsection 2.8 with $c_u^* = 1$ for $(\bullet)$

That shows the rounding error on the calculation of x^y can be bounded 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 α and β are algebraic numbers with $\alpha \neq 0$, $\alpha \neq 1$, and $\beta \notin \mathbb{Q}$, then α^β is transcendental. This is of little help for us since β 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.

4.22. **The integer power.** The integer power `mpfr_pow_ui` is computed as follows. We compute an approximation of x^n by binary exponentiation. The error analysis for binary exponentiation is the same as if we did naive exponentiation, computing $x^2, x^3, ..., x^{n-1}, x^n$ with $n - 1$ successive multiplications, and Lemma 1 gives the error bound.

4.23. **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.

4.24. **The k-th root.** The k-th root `mpfr_root` of $x > 0$ is computed as follows. First write $x = m \cdot 2^e$ with m and e integers, e multiple of k. If $m < 2^{k(p-1)}$, where p is the target precision — plus 1 for rounding to nearest —, we multiply m by 2^{kt} for some integer $t > 0$ and subtract kt from e such that $2^{k(p-1)} \leq m \cdot 2^{kt} < 2^{kp}$, i.e., the integer square root of $m \cdot 2^{kt}$ has exactly p bits.

We thus now have $x = m \cdot 2^e$ where $2^{k(p-1)} \leq m$ and e multiple of k. We then call the `mpz_root` function from GMP, which computes s such that $s^k \leq m < (s+1)^k$, and tells us if the left equality holds or not (this gives the round bit for directed rounding, and the sticky bit for rounding to nearest, in which case the round bit is the least significant bit of s). If s has more than p bits, the round and sticky bits can both be determined from the low bits of s.

Note: this algorithm is inefficient since it deals with intermediate values of $O(kp)$ bits.

4.25. **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 toward zero.

Approximate x by $m \cdot 2^e$ with m an integer
having exactly w bits, such that $|x - m \cdot 2^e| \leq 2^e$
$s \leftarrow 0$
$t \leftarrow 2^w$
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 ϵ_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 $2^e \leq m2^{e+1-w}$. 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 τ_k on the error on s made so far.

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

Asymptotic Expansion. For $x \to \infty$ we have the following non-converging expansion [1, formula 5.1.51]:

$$\mathrm{Ei}(x) \sim e^x\left(\frac{1}{x}+\frac{1}{x^2}+\frac{2}{x^3}+\frac{6}{x^4}+\frac{24}{x^5}+\cdots\right).$$

The kth is of the form $k!x^{-k-1}$. The smallest value is obtained for $k \approx x$, and is of the order of e^{-x}. Thus assuming the error term is bounded by the first neglected term, we can use that expansion as long as $e^{-x} \le 2^{-p}$ where p is the target precision, i.e. when $x \ge p \log 2$.

For $x < 0$, the function `mpfr_eint` returns the value of $E_1(-x)$, defined by [1, formula 5.1.11]:

$$E_1(x) = -\gamma - \log(x) - \sum_{k=1}^{\infty} \frac{(-x)^k}{k\,k!}.$$

We use the very same algorithm and error analysis as above (including the asymptotic expansion).

4.26. **The gamma function.** The gamma function is computed by Spouge's method [26]:

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

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 [22, Section 2.6], the relative error is bounded by $a^{-1/2}(2\pi)^{-a-1/2}$ for $a \ge 3$ and $\Re(z) \ge 0$. See also [25].

4.27. **The incomplete gamma function.** The incomplete gamma function is defined for $x \ge 0$ as (it can be defined by continuity for a being zero or a negative integer):

$$\Gamma(a,x) = \int_x^{\infty} t^{a-1}\exp(-t)\,\mathrm{d}t = \Gamma(a) - \gamma(a,x),$$

with the complementary incomplete gamma function $\gamma(a,x)$ defined by:

$$\gamma(a,x) = \int_0^x t^{a-1}\exp(-t)\,\mathrm{d}t,$$

satisfying the two Taylor expansions [1, formula 6.5.29]:

$$\gamma(a,x) = x^a\sum_{k=0}^{\infty}\frac{(-x)^k}{(a+k)k!} = x^a\exp(-x)\sum_{k=0}^{\infty}\frac{x^k}{a(a+1)\cdots(a+k)}.$$

We use the second expansion, and we stop adding terms of the series when the current term is less than one ulp of the current sum, and $|x/(a+k)| < 1/2$. The last condition ensures that the tail of the series is less than twice the current term, thus the approximated sum s satisfies $s = S(1+\theta)^3$ with S the exact sum, and $|\theta| < 2^{-w}$ with w the working precision. If the series was computed up to index k, the error on $\gamma(a, x)$ is less than $4k + 14$ ulps. Subtracting from $\Gamma(a)$ can yield a huge cancellation: we take into account the exact error bound, which is deduced from the different exponents involved. Further details can be found in the source code.

For negative integers a we use the following formula [1, formula 6.5.19]:

$$\Gamma(-n, x) = \frac{(-1)^n}{n!}[E_1(x) - e^{-x} \sum_{j=0}^{n-1} \frac{(-1)^j\, j!}{x^{j+1}}].$$

4.27.1. *Legendre's continued fraction.* In [9], Gautschi discusses the computation of $\Gamma(a, x)$ with Legendre's continued fraction:

$$x^{-a} e^x \Gamma(a, x) = \frac{1}{x+} \frac{1-a}{1+} \frac{1}{x+} \frac{2-a}{x+} \frac{2}{x+} \cdots$$

where by contracting two consecutive terms we get:

$$x^{-a} e^x \Gamma(a, x) = \sum_{k=0}^{\infty} t_k, \tag{7}$$

with $t_k = \rho_0 \rho_1 \cdots \rho_k$, $\rho_0 = 1$, $\sigma_0 = 1$, and for $k \geq 1$, $\rho_k = -a_k \sigma_{k-1}/(1 + a_k \sigma_{k-1})$, $\sigma_k = 1/(1 + a_k \sigma_{k-1})$, and:

$$a_k = \frac{k(a-k)}{(x + 2k - 1 - a)(x + 2k + 1 - a)}.$$

Gautschi proves that when $x \geq 1/4$ and $a \leq x + 1/4$, then $|\rho_k| \leq 1$ for all $k \geq 1$. However, this is not enough to bound the tail of the above series.

We consider here the case where $a \leq -1$, thus $a_k \leq 0$ (remember $\Gamma(a, x)$ is only defined for $x \geq 0$). When k goes to infinity, $a_k = -1/4 + x/(4k) + O(1/k^2)$. We prove that for $k \geq 2\max(x, |a|)$, we have $a_k \geq -1/4 + x/(8k)$. Indeed,

$$a_k + \frac{1}{4} - \frac{x}{8k} = \frac{x(4k^2 - 2kx + 2xa - x^2 - a^2 + 1) + 2k(a^2 - 1)}{8k(x + 2k - 1 - a)(x + 2k + 1 - a)}.$$

The denominator is positive for $k \geq 1$ as long as $x - a + 1 > 0$, which holds since we assumed $a \leq -1$. The second term $2k(a^2 - 1)$ in the numerator is non-negative since $a \leq -1$; the factor of x in the first term satisfies since $k \geq x$:

$$4k^2 - 2kx + 2xa - x^2 - a^2 + 1 \geq 4k^2 - 2kx - 2k|a| - x^2 - a^2 + 1.$$

Since the right-hand side is symmetric in x and $|a|$, we can assume without loss of generality that $x \geq |a|$. Then $4k^2 - 2kx + 2xa - x^2 - a^2 + 1 \geq 4k^2 - 4kx - 2x^2 + 1 = (2k - x)^2 - 3x^2 + 1$, which is positive when $k \geq 2x$.

Since $a_k \geq -1/4 + x/(8k)$, we have $-1/4 \leq a_k \leq 0$, and since $\sigma_k = 1/(1 + a_k \sigma_{k-1})$ with $\sigma_0 = 1$, it follows by induction that $1 \leq \sigma_k \leq 2$ for all $k \geq 0$. It yields for $a \leq -1$ and $k \geq 2\max(x, |a|)$:

$$\rho_k = \frac{-a_k \sigma_{k-1}}{1 + a_k \sigma_{k-1}} \leq -4a_k \leq 1 - x/(2k).$$

Assume now that we stop the series computation in Eq. (7) after t_k. The neglected part is:

$$t_k(\rho_{k+1} + \rho_{k+1}\rho_{k+2} + \rho_{k+1}\rho_{k+2}\rho_{k+3} + \cdots).$$

Let $u_{k,\ell} := \rho_k\rho_{k+1}\cdots\rho_\ell$. Since $\rho_j \le 1 - x/(2j)$ and $\log(1-u) \le -u$ for $0 < u < 1$:

$$\log u_{k,\ell} = \sum_{j=k}^{\ell} \log \rho_j \le \sum_{j=k}^{\ell} \log(1 - \frac{x}{2j}) \le -x \sum_{j=k}^{\ell} \frac{1}{2j} = -\frac{x}{2}(\psi(\ell+1) - \psi(k+1)).$$

Using $\log(t-1) \le \psi(t) \le \log t$ yields:

$$\log u_{k,\ell} \le \frac{x}{2}(\log(k+1) - \log(\ell)).$$

Now since $\rho_k < 1$:

$$\rho_k + \rho_k\rho_{k+1} + \rho_k\rho_{k+1}\rho_{k+2} + \cdots \le 1 + \sum_{\ell=k+1}^{\infty} u_{k,\ell} \le 1 + \sum_{j=1}^{\infty} \exp(\frac{x}{2}\log\frac{k+1}{k+j}).$$

We split the sum into blocks of $k+1$ consecutive terms:

$$\begin{aligned}
\sum_{\ell=k+1}^{\infty} u_{k,\ell} &\le \sum_{j=1}^{\infty} \left(\frac{k+1}{k+j}\right)^{x/2} \\
&\le (k+1)\left(\frac{k+1}{k+1}\right)^{x/2} + (k+1)\left(\frac{k+1}{2k+2}\right)^{x/2} + (k+1)\left(\frac{k+1}{3k+3}\right)^{x/2} + \cdots \\
&\le (k+1)(1 + (1/2)^{x/2} + (1/3)^{x/2} + \cdots) = (k+1)\zeta(x/2).
\end{aligned}$$

Lemma 6. *Assume $a \le -1$, and we stop the series computation in Eq. (7) after t_k, with $k \ge 2\max(x, |a|)$. Then the error on the sum of the right hand side of Eq. (7) is bounded by*

$$t_k(1 + (k+1)\zeta(x/2)).$$

4.28. **The Riemann Zeta function.**

4.28.1. *Special cases.* As usual, special inputs are first taken into account: NaN, infinities, zeros, but also values close enough to 0 (see below), even negative integers, and 1 (pole).

Let us focus on $\zeta(s)$, where s has a small exponent. In theory, this case could be handled by the reflection formula (§4.28.3), but as the term $\zeta(1-s)$ of this formula is close to the pole of ζ at 1, this method would be slow (and could even yield an internal overflow for tiny s), while the correctly rounded result can be determined very quickly when s is close enough to 0. We have around 0:

$$\zeta(s) = -\frac{1}{2} - \frac{1}{2}\log(2\pi)s + \ldots$$

and for $|s| \le 2^{-4}$, we have $|\zeta(s) + 1/2| \le |s|$. Thus if $|s| \le 2^{-4}$ and $|s| \le \frac{1}{4}\mathrm{ulp}(1/2)$ in the target precision p, we can deduce the correct rounding for any rounding mode. The second condition can be rewritten: $|s| \le 2^{-2-p}$. For $p \ge 2$, the second condition implies the first one, and it is sufficient to have $\mathrm{Exp}(s) \le -2-p$. For $p = 1$, if we assume $\mathrm{Exp}(s) \le -2-p$, then $|s| \le \frac{1}{2}2^{-2-p} = 2^{-4}$, so that both conditions are also satisfied. Thus, for any target precision $p \ge 1$, a sufficient condition is $\mathrm{Exp}(s) \le -2-p$, or equivalently $\mathrm{Exp}(s) + 1 < -p$.

4.28.2. *Case $s \geq 1/2$.* The algorithm for the Riemann Zeta function for $s \geq 1/2$ is due to Jean-Luc Rémy and Sapphorain Pétermann [20, 21]. 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 kth 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

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

4.28.3. *Case $s < 1/2$.* For $s < 1/2$, we use the reflection formula:

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

For simplicity, to avoid taking into account the error from the Γ and ζ inputs, we will ensure that $1-s$ is represented exactly. Thus its precision may need to be much larger than the target precision. However, for efficiency reasons, the internal working precision q will still be based only on the target precision as usual.

So, assuming that no underflows nor overflows occur, the terms $\Gamma(1-s)$ and $\zeta(1-s)$ will each have an error factor of the form $1 + \theta$, with $|\theta| \leq 2^{-q}$.

Assuming that Γ and ζ have a larger complexity than the other terms, we would like the other error factors not to be significantly larger than $1 + 2^{-q}$. Otherwise we would have wasted time by computing $\Gamma(1-s)$ and $\zeta(1-s)$ with more precision than really needed.

Concerning the term π^{s-1}, if the constant π is represented by a variable x with a precision p_x to the nearest, then $(1-2^{-p_x})x \le \pi \le (1+2^{-p_x})x$. So,

$$(1+2^{-p_x})^{s-1}x^{s-1} \le \pi^{s-1} \le (1-2^{-p_x})^{s-1}x^{s-1},$$

i.e.,

$$(1-2^{-p_x})^{1-s}\pi^{s-1} \le x^{s-1} \le (1+2^{-p_x})^{1-s}\pi^{s-1}.$$

We want $(1-s)2^{-p_x}$ to be small and of the order of 2^{-q}. We will choose $p_x = q - \textsc{Exp}(1-s)$, so that $(1-s)2^{-p_x} < 2^{-q}$. The value x^{s-1} will be computed and rounded to the working precision by the `mpfr_pow` function, giving another error term of the form $1+\theta$, with $|\theta| \le 2^{-q}$.

Concerning the term $\sin\left(\frac{\pi s}{2}\right)$, because of the factor π in the argument, we will do the range reduction ourselves: it will much simpler and faster than the one in `mpfr_sin`, and this will allow us to select the intermediate precision more accurately.

(WORK IN PROGRESS – May not correspond to the implementation yet.)

4.28.4. *The integer argument case.* In case of an integer argument $s \ge 2$, the `mpfr_zeta_ui` function computes $\zeta(s)$ using the following formula from [4]:

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

where

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

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
  $T \leftarrow \lfloor T2^{1-s}\rfloor$
  $S = S + T$
while $T \ne 0$.

Since $\frac{1}{1-2^{1-s}} = 1 + 2^{1-s} + 2^{2(1-s)} + \cdots$, and at iteration i we have $T = \lfloor S2^{i(1-s)}\rfloor$, the error on S after this loop is bounded by $n+l+1$, where l is the number of loops.

Finally we compute $q = \lfloor\frac{2^pS}{d_n}\rfloor$, 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 $q \ge 2^p$, and the error on the division is less that $\frac{1}{2}\mathrm{ulp}_p(q)$. The error on S itself is bounded by $(n+l+1)/d_n \le (n+l+1)2^{1-p}$ — see the conjecture below. Since $2^{1-p} \le \mathrm{ulp}_p(q)$, 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 $\frac{3}{(3+\sqrt{8})^n} \le \frac{3}{2^p}$, 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 $p = 1000$.
Large argument case. When $3^{-s} < 2^{-p}$, then $\zeta(s) \approx 1 + 2^{-s}$ to a precision of p bits. More precisely, let $r(s) := \zeta(s) - (1 + 2^{-s}) = 3^{-s} + 4^{-s} + \cdots$. The function $3^s r(s) = 1 + (3/4)^s + (3/5)^s + \cdots$ decreases with s, thus for $s \geq 2$, $3^s r(s) \leq 3^2 \cdot r(2) < 4$. This yields:

$$|\zeta(s) - (1 + 2^{-s})| < 4 \cdot 3^{-s}.$$

If the upper bound $4 \cdot 3^{-s}$ is less than $\frac{1}{2}\mathrm{ulp}(1) = 2^{-p}$, the correct rounding of $\zeta(s)$ is either $1 + 2^{-s}$ for rounding to zero, $-\infty$ or nearest, and $1 + 2^{-s} + 2^{1-p}$ for rounding to $+\infty$.

4.29. **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 $\mathrm{AGM}(a, b)$ 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** $\textsc{Exp}(v_n) - \textsc{Exp}(v_n - u_n) \leq p - 2$ **do**
  $v_{n+1} = \circ(u_n + v_n)/2$ [division by 2 is exact]
  if $\textsc{Exp}(v_n) - \textsc{Exp}(v_n - u_n) \leq p/4$ **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 ϵ 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 ϵ^4. [Note that w — and therefore s, t — need to be computed to precision $p/2$ only.]

Lemma 7. *Assuming $u \le v$ are two p-bit floating-point numbers, then $u' = \circ(\sqrt{\circ(uv)})$ and $v' = \circ(u+v)/2$ satisfy:*

$$u \le u', v' \le v.$$

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

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 $s = \circ(u^2) \le s'$, thus $u = \circ(\sqrt{s}) \le \circ(\sqrt{s'}) = u'$. We prove similarly that $u' \le v$. □

Remark. We cannot assume that $u' \le v'$. 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 θ denotes a generic value such that $|\theta| \le 2^{-p}$. 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 $v_n = b_n(1+\theta)^\beta$ with $\alpha, \beta \le e_n$. 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 $e_n \le 3n/2$.

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

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

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

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

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

Let q be the precision used to evaluate s, t and w in the **else** case. Since $|v_n - u_n| \le 2^{(3-p)/4}v_n$, it follows $|s| \le 1.8 \cdot 2^{-p/4}v_n$ for $q \ge 4$. Then $t \le 0.22 \cdot 2^{-p/2}v_n$. 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. $\frac{v_n}{v_{n+1}} \le \frac{1}{1-2^{(3-p)/4}} \le 2$ for $p \ge 7$. We deduce $w \le 0.22 \cdot 2^{-p/2}\frac{v_n^2}{v_{n+1}}(1 + 2^{-q}) \le 0.47 \cdot 2^{-p/2}v_n \le 0.94 \cdot 2^{-p/2}v_{n+1}$.

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

Since $b_n = v_n(1+3n\theta)$, and we assumed $3n|\theta|/2 \le 1$, we have $b_n \le 3v_n$, 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 $v_n \le 2v_{n+1}$, 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 $r = \circ(v_{n+1} - w)$.

4.30. **The Bessel functions.**

4.30.1. *Bessel function $J_n(z)$ of first kind.* The Bessel function $J_n(z)$ of first kind and integer order n is defined as follows [1, Eq. (9.1.10)]:

$$J_n(z) = (z/2)^n \sum_{k=0}^{\infty} \frac{(-z^2/4)^k}{k!(k+n)!}. \tag{8}$$

It is real for all real z, tends to 0 by oscillating around 0 when $z \to \pm\infty$, and tends to 0 when $z \to 0$, except J_0 which tends to 1.

We use the following algorithm, with working precision w, and rounding to nearest. Warning! This algorithm assumes that no underflows/overflows occur.

$x \leftarrow \circ(z^n)$
$y \leftarrow \circ(z^2)/4$ [division by 4 is exact]
$u \leftarrow \circ(n!)$
$t \leftarrow \circ(x/u)/2^n$ [division by 2^n is exact]
$s \leftarrow t$
for k from 1 do
  $t \leftarrow -\circ(ty)$
  $t \leftarrow \circ(t/k)$
  $t \leftarrow \circ(t/(k+n))$
  $s \leftarrow \circ(s+t)$

if $|t| < \mathrm{ulp}(s)$ and $z^2 \leq 2k(k+n)$ then return s.

The condition $z^2 \leq 2k(k+n)$ ensures that the next term of the expansion is smaller than $|t|/2$, thus the sum of the remaining terms is smaller than $|t| < \mathrm{ulp}(s)$. Using Higham's method, with θ denoting a random variable of value $|\theta| \leq 2^{-w}$ — different instances of θ denoting different values — we can write $x = z^n(1+\theta)$, $y = z^2/4(1+\theta)$, and before the for-loop $s = t = (z/2)^n/n!(1+\theta)^3$. Now write $t = (z/2)^n(-z^2/4)^k/(k!(k+n)!)(1+\theta)^{e_k}$ at the end of the for-loop with index k; each loop involves a factor $(1+\theta)^4$, thus we have $e_k = 4k+3$. Now let T be an upper bound on the values of $|t|$ and $|s|$ during the for-loop, and assume we exit at $k = K$. The roundoff error in the additions $\circ(s+t)$, including the error in the series truncation, is bounded by $(K/2+1)\mathrm{ulp}(T)$. The error in the value of t at step k is bounded by $\epsilon_k := T|(1+\theta)^{4k+3} - 1|$; if we assume $(4k+3)2^{-w} \leq 1/2$, Lemma 2 yields $\epsilon_k \leq 2T(4k+3)2^{-w}$. Summing from $k=0$ to K, this gives an absolute error bound on s at the end of the for-loop of:

$$(K/2+1)\mathrm{ulp}(T) + 2(2K^2+5K+3)2^{-w}T \leq (4K^2+21/2K+7)\mathrm{ulp}(T),$$

where we used $2^{-w}T \leq \mathrm{ulp}(T)$.

Large index n. For large index n, formula 9.1.62 from [1] gives $|J_n(z)| \leq |z/2|^n/n!$. Together with $n! \geq \sqrt{2\pi n}(n/e)^n$, which follows from example from [1, Eq. 6.1.38], this gives:

$$|J_n(z)| \leq \frac{1}{\sqrt{2\pi n}}\left(\frac{ze}{2n}\right)^n.$$

Large argument. For large argument z, formula (8) requires at least $k \approx z/2$ terms before starting to converge. If $k \leq z/2$, it is better to use formula 9.2.5 from [1], which provides at least 2 bits per term:

$$J_n(z) = \sqrt{\frac{2}{\pi z}}[P(n,z)\cos\chi - Q(n,z)\sin\chi],$$

where $\chi = z - (n/2+1/4)\pi$, and $P(n,z)$ and $Q(n,z)$ are two diverging series:

$$P(n,z) \approx \sum_{k=0}^{\infty}(-1)^k\frac{\Gamma(1/2+n+2k)(2z)^{-2k}}{(2k)!\Gamma(1/2+n-2k)}, \quad Q(n,z) \approx \sum_{k=0}^{\infty}(-1)^k\frac{\Gamma(1/2+n+2k+1)(2z)^{-2k-1}}{(2k+1)!\Gamma(1/2+n-2k-1)}.$$

If n is real and nonnegative — which is the case here —, the remainder of $P(n,z)$ after k terms does not exceed the $(k+1)$th term and is of the same sign, provided $k > n/2 - 1/4$; the same holds for $Q(n,z)$ as long as $k > n/2 - 3/4$ [1, 9.2.10].

If we first approximate $\chi = z - (n/2+1/4)\pi$ with working precision w, and then approximate $\cos\chi$ and $\sin\chi$, there will be a huge relative error if $z > 2^w$. Instead, we use the fact that for n even,

$$P(n,z)\cos\chi - Q(n,z)\sin\chi = \frac{1}{\sqrt{2}}(-1)^{n/2}[P(n,z)(\sin z + \cos z) + Q(n,z)(\cos z - \sin z)],$$

and for n odd,

$$P(n,z)\cos\chi - Q(n,z)\sin\chi = \frac{1}{\sqrt{2}}(-1)^{(n-1)/2}[P(n,z)(\sin z - \cos z) + Q(n,z)(\cos z + \sin z)],$$

where $\cos z$ and $\sin z$ are computed accurately with `mpfr_sin_cos`, which uses in turn `mpfr_remainder`.

If we consider $P(n,z)$ and $Q(n,z)$ together as one single series, its term of index k behaves like $\Gamma(1/2+n+k)/k!/\Gamma(1/2+n-k)/(2z)^k$. The ratio between the term of index $k+1$ and that of index k is about $k/(2z)$, thus starts to diverge when $k \approx 2z$. At that point, the kth term is $\approx e^{-2z}$, thus if $z > p/2\log 2$, we can use the asymptotic expansion.

4.30.2. *Bessel function $Y_n(z)$ of second kind.* Like $J_n(z)$, $Y_n(z)$ is a solution of the linear differential equation:

$$z^2y'' + zy' + (z^2 - n^2)y = 0.$$

We have $Y_{-n}(z) = (-1)^n Y_n(z)$ according to [1, Eq. (9.1.5)]; we now assume $n \geq 0$. When $z \to 0^+$, $Y_n(z)$ tends to $-\infty$; when $z \to +\infty$, $Y_n(z)$ tends to 0 by oscillating around 0 like $J_n(z)$. We deduce from [27, Eq. (9.23)]:

$$Y_n(-z) = (-1)^n[Y_n(z) + 2iJ_n(z)],$$

which shows that for $z > 0$, $Y_n(-z)$ is real only when z is a zero of $J_n(z)$; assuming those zeroes are irrational, $Y_n(z)$ is thus NaN for z negative.

Formula 9.1.11 from [1] gives:

$$\begin{aligned} Y_n(z) &= -\frac{(z/2)^{-n}}{\pi}\sum_{k=0}^{n-1}\frac{(n-k-1)!}{k!}(z^2/4)^k + \frac{2}{\pi}\log(z/2)J_n(z) \\ &- \frac{(z/2)^n}{\pi}\sum_{k=0}^{\infty}(\psi(k+1)+\psi(n+k+1))\frac{(-z^2/4)^k}{k!(n+k)!}, \end{aligned}$$

where $\psi(1) = -\gamma$, γ being Euler's constant (see §5.2), and $\psi(n+1) = \psi(n) + 1/n$ for $n \geq 1$.

Rewriting the above equation, we get

$$\pi Y_n(z) = -(z/2)^{-n}S_1 + S_2 - (z/2)^n S_3,$$

where $S_1 = \sum_{k=0}^{n-1}\frac{(n-k-1)!}{k!}(z^2/4)^k$ is a finite sum, $S_2 = 2(\log(z/2)+\gamma)J_n(z)$, and $S_3 = \sum_{k=0}^{\infty}(h_k + h_{n+k})\frac{(-z^2/4)^k}{k!(n+k)!}$, where $h_k = 1 + 1/2 + \cdots + 1/k$ is the kth harmonic number. Once we have estimated $-(z/2)^{-n}S_1 + S_2$, we know to which relative precision we need to estimate the infinite sum S_3. For example, if $(z/2)^n$ is small, typically a small relative precision on S_3 will be enough.

We use the following algorithm to estimate S_1, with working precision w and rounding to nearest:

$y \leftarrow \circ(z^2)/4$ [division by 4 is exact]
$f \leftarrow 1$ [as an exact integer]
$s \leftarrow 1$
for k from $n-1$ downto 0 do
  $s \leftarrow \circ(ys)$
  $f \leftarrow (n-k)(k+1)f$ [$n!(n-k)!k!$ as exact integer]
  $s \leftarrow \circ(s+f)$
$f \leftarrow \sqrt{f}$ [integer, exact]
$s \leftarrow \circ(s/f)$

Let $(1+\theta)^{\epsilon_j} - 1$ be the maximum relative error on s after the look for $k = n-j$, $1 \leq j \leq n$, i.e., the computed value is $s_k(1+\theta)^{\epsilon_j}$ where s_k would be the value computed with no roundoff error, and $|\theta| \leq 2^{-w}$. Before the loop we have $\epsilon_0 = 0$. After the instruction $s \leftarrow \circ(ys)$

the relative error can be written $(1+\theta)^{\epsilon_{j-1}+2}-1$, since $y = z^2/4(1+\theta')$ with $|\theta'| \le 2^{-w}$, and the product involves another rounding error. Since f is exact, the absolute error after $s \leftarrow \circ(s+f)$ can be written $|s_{\max}||(1+\theta)^{\epsilon_{j-1}+3}-1|$, where $|s_{\max}|$ is a bound for all computed values of s during the loop. The absolute error at the end of the for-loop can thus be written $|s_{\max}||(1+\theta)^{3n}-1|$, and $|s_{\max}||(1+\theta)^{3n+1}-1|$ after the instruction $s \leftarrow \circ(s/f)$. If $(3n+1)2^{-w} \le 1/2$, then using Lemma 2, $|(1+\theta)^{3n+1}-1| \le 2(3n+1)2^{-w}$. Let e be the exponent difference between the maximum value of $|s|$ during the for-loop and the final value of s, then the relative error on the final s is bounded by

$$(3n+1)2^{e+1-w}.$$

Assuming we compute $(z/2)^n$ with correct rounding — using for example the `mpfr_pow_ui` function — and divide S_1 by this approximation, the relative error on $(z/2)^{-n}S_1$ will be at most $(3n+3)2^{e+1-w}$.

The computation of S_2 is easier, still with working precision w and rounding to nearest:

$t \leftarrow \circ(\log(z/2))$
$u \leftarrow \circ(\gamma)$
$v \leftarrow 2 \circ (t+u)$ [multiplication by 2 is exact]
$x \leftarrow \circ(J_n(z))$
$s \leftarrow \circ(vx)$

Since $z/2$ is exact, the error on t and u is at most one ulp, thus from §2.3 the ulp-error on v is at most $1/2 + 2^{\mathrm{Exp}(t)-\mathrm{Exp}(v)} + 2^{\mathrm{Exp}(u)-\mathrm{Exp}(v)} \le 1/2 + 2^{e+1}$, where $e = \max(\mathrm{Exp}(t), \mathrm{Exp}(u)) - \mathrm{Exp}(v)$. Assuming $e+2 < w$, then $1/2 + 2^{e+1} \le 2^{w-1}$, thus the total error on v is bounded by $|v|$, thus we can take $c^+ = 2$ for v in the product $s \leftarrow \circ(vx)$ (cf §2.4); similarly $c^+ = 2$ applies to $x \leftarrow \circ(J_n(z))$, thus §2.4 yields the following bound for the ulp-error on s:

$$1/2 + 3(1/2 + 2^{e+1}) + 3(1/2) = 7/2 + 3 \cdot 2^{e+1} \le 2^{e+4}.$$

(Indeed, the smallest possible value of e is -1.)

The computation of S_3 mimics that of $J_n(z)$. The only difference is that we have to compute the extra term $h_k + h_{n+k}$, that we maintain as an exact rational p/q, p and q being integers:

$x \leftarrow \circ(z^n)$
$y \leftarrow \circ(z^2)/4$ [division by 4 is exact]
$u \leftarrow \circ(n!)$
$t \leftarrow \circ(x/u)/2^n$ [division by 2^n is exact]
$p/q \leftarrow h_n$ [exact rational]
$u \leftarrow \circ(pt)$
$s \leftarrow \circ(u/q)$
for k from 1 do
  $t \leftarrow -\circ(ty)$
  $t \leftarrow \circ(t/k)$
  $t \leftarrow \circ(t/(k+n))$
  $p/q \leftarrow p/q + 1/k + 1/(n+k)$ [exact]
  $u \leftarrow \circ(pt)$
  $u \leftarrow \circ(u/q)$
  $s \leftarrow \circ(s+u)$

if $|u| < \mathrm{ulp}(s)$ and $z^2 \le 2k(k+n)$ then return s.

Using $(h_{k+1}+h_{n+k+1})k \le (h_k+h_{n+k})(k+1)$, which is true for $k \ge 1$ and $n \ge 0$, the condition $z^2 \le 2k(k+n)$ ensures that the next term of the expansion is smaller than $|t|/2$, thus the sum of the remaining terms is smaller than $|t| < \mathrm{ulp}(s)$. The difference with the error analysis of J_n is that here $e_k = 6k+5$ instead of $e_k = 4k+3$. Denote U an upper bound on the values of u, s during the for-loop — note that $|u| \ge |t|$ by construction — and assume we exit at $k = K$. The error in the value of u at step k is bounded by $\epsilon_k := U|(1+\theta)^{6k+5} - 1|$; Assuming $(6k+5)2^{-w} \le 1/2$, Lemma 2 yields $\epsilon_k \le 2U(6k+5)2^{-w}$, and the sum from $k = 0$ to K gives an absolute error bound on s at the end of the for-loop bounded by:

$$(K/2+1)\mathrm{ulp}(U) + 2(3K^2+8K+5)2^{-w}U \le (6K^2+33/2K+11)\mathrm{ulp}(U),$$

where we used $2^{-w}U \le \mathrm{ulp}(U)$.

4.31. **The Dilogarithm function.** The `mpfr_li2` function computes the real part of the dilogarithm function defined by:

$$\mathrm{Li}_2(x) = -\int_0^x \frac{\log(1-t)}{t} dt.$$

The above relation defines a multivalued function in the complex plane, we choose a branch so that $\mathrm{Li}_2(x)$ is real for x real, $x < 1$ and we compute only the real part for x real, $x \ge 1$.

When $x \in]0, \frac{1}{2}]$, we use the series (see [28, Eq. (5)]):

$$\mathrm{Li}_2(x) = \sum_{n=0}^{\infty} \frac{B_n}{(n+1)!}(-\log(1-x))^{n+1}$$

where B_n is the n-th Bernoulli number.

Otherwise, we perform an argument reduction using the following identities (see [10]):

$$\begin{array}{llcl}
x \in [2, +\infty[& \Re(\mathrm{Li}_2(x)) & = & \frac{\pi^2}{3} - \frac{1}{2}\log^2(x) - \mathrm{Li}_2\left(\frac{1}{x}\right) \\
x \in]1, 2[& \Re(\mathrm{Li}_2(x)) & = & \frac{\pi^2}{6} - \log(x)\left[\log(x-1) - \frac{1}{2}\log(x)\right] + \mathrm{Li}_2\left(1 - \frac{1}{x}\right) \\
 & \mathrm{Li}_2(1) & = & \frac{\pi^2}{6} \\
x \in \left]\frac{1}{2}, 1\right[& \mathrm{Li}_2(x) & = & \frac{\pi^2}{6} - \log(x)\log(1-x) - \mathrm{Li}_2(1-x) \\
 & \mathrm{Li}_2(0) & = & 0 \\
x \in [-1, 0[& \mathrm{Li}_2(x) & = & -\frac{1}{2}\log^2(1-x) - \mathrm{Li}_2\left(\frac{x}{x-1}\right) \\
x \in]-\infty, -1[& \mathrm{Li}_2(x) & = & -\frac{\pi^2}{6} - \frac{1}{2}\log(1-x)[2\log(-x) - \log(1-x)] + \mathrm{Li}_2\left(\frac{1}{1-x}\right).
\end{array}$$

Assume first $0 < x \le \frac{1}{2}$, the odd Bernoulli numbers being zero (except $B_1 = -\frac{1}{2}$), we can rewrite $\mathrm{Li}_2(x)$ in the form:

$$\mathrm{Li}_2(x) = -\frac{\log^2(1-x)}{4} + S(-\log(1-x))$$

where

$$S(z) = \sum_{k=0}^{\infty} \frac{B_{2k}}{(2k+1)!} z^{2k+1}.$$

Let $S_N(z) = \sum_{k\le N} \frac{B_{2k}}{(2k+1)!} z^{2k+1}$ the N-th partial sum, and $R_N(z)$ the truncation error. The even Bernoulli numbers verify the following inequality for all $n \ge 1$ ([1, Inequalities 23.1.15]):

$$\frac{2(2n)!}{(2\pi)^{2n}} < |B_{2n}| < \frac{2(2n)!}{(2\pi)^{2n}} \left(\frac{1}{1-2^{1-2n}}\right),$$

so we have for all $N \ge 0$

$$\frac{|B_{2N+2}|}{(2N+3)!}|z|^{2N+3} < \frac{2|z|}{(1-2^{-2N-1})(2N+3)} \left|\frac{z}{2\pi}\right|^{2N+2},$$

showing that $S(z)$ converges for $|z| < 2\pi$. As the series is alternating, we then have an upper bound for the truncation error $|R_N(z)|$ when $0 < z \le \log 2$:

$$|R_N(z)| < 2^{\mathrm{Exp}(z)-6N-5}.$$

The partial sum $S_N(z)$ computation is implemented as follows:

Algorithm `li2_series`
Input: z with $z \in]0, \log 2]$
Output: $\circ(S(z))$
$u \leftarrow \triangle(z^2)$
$v \leftarrow \triangle(z)$
$s \leftarrow \triangle(z)$
for k **from** 1 **do**
  $v \leftarrow \triangle(uv)$
  $v \leftarrow \triangle(v/(2k))$
  $v \leftarrow \triangle(v/(2k))$
  $v \leftarrow \triangle(v/(2k+1))$
  $v \leftarrow \triangle(v/(2k+1))$
  $w \leftarrow \mathcal{N}(vB'_k)$
  $s \leftarrow \mathcal{N}(s+w)$
  if $|w| < \mathrm{ulp}(s)$ **then** return s.

where $B'_k = B_{2k}(2k+1)!$ is an exact representation in `mpn` integers.

Let p the working precision. Using Higham's method, before entering the loop we have $u = z^2(1+\theta)$, $v = s = z(1+\theta)$ where different instances of θ denote different variables and $|\theta| \le 2^{-p}$. After the k-th loop, $v = z^{2k+1}/((2k+1)!2k(2k+1))(1+\theta)^{6k}$, $w = B_{2k}z^{2k+1}/(2k+1)!(1+\theta)^{6k+1}$.

When $x \in]0, \frac{1}{2}]$, $\mathrm{Li}_2(x)$ calculation is implemented as follows

Algorithm `li2_0..+`$\frac{1}{2}$
Input: x with $x \in \left]0, \frac{1}{2}\right]$, the output precision n, and a rounding mode $\circ_n$
Output: $\circ_n(\mathrm{Li}_2(x))$

$u \leftarrow \mathcal{N}(1-x)$	$\mathrm{error}(u)$	$\le$	$\frac{1}{2}\mathrm{ulp}(u)$
$u \leftarrow \triangle(-\log(u))$	$\mathrm{error}(u)$	$\le$	$(1+2^{-\mathrm{Exp}(u)})\mathrm{ulp}(u)$
$t \leftarrow \triangle(S(u))$	$\mathrm{error}(t)$	$\le$	$(k+1)2^{1-\mathrm{Exp}(t)}\mathrm{ulp}(t)$
$v \leftarrow \triangle(u^2)$			
$v \leftarrow \triangle(v/4)$	$\mathrm{error}(v)$	$\le$	$(5+2^{2-\mathrm{Exp}(u)})\mathrm{ulp}(v)$
$s \leftarrow \mathcal{N}(t-v)$ $\quad error(s) \le 2^{\kappa_s}\mathrm{ulp}(s)$			

if s cannot be exactly rounded according to the given mode $\circ_n$
then increase the working precision and restart calculation
else return $\circ_n(s)$

where $\kappa_s = \max(-1, \lceil \log_2(k+1) \rceil + 1 - \mathrm{EXP}(s), \max(1, -\mathrm{EXP}(u)) - 1 - \mathrm{EXP}(s))$

When x is large and positive, we can use an asymptotic expansion near $+\infty$ using the fact that $\mathrm{Li}_2\left(\frac{1}{x}\right) = \frac{1}{x} + O\left(\frac{1}{x^2}\right)$ (see below):

$$\left| \mathrm{Li}_2(x) + \frac{\log^2 x}{2} - \frac{\pi^2}{3} \right| \leq \frac{2}{x}$$

which gives the following algorithm:

Algorithm `li2_asympt_pos`
Input: x with $x \geq 38$, the output precision n, and a rounding mode $\circ_n$
Output: $\circ_n(\Re(\mathrm{Li}_2(x)))$ if it can round exactly, a failure indicator if not
$u \leftarrow \mathcal{N}(\log x)$
$v \leftarrow \mathcal{N}(u^2)$
$g \leftarrow \mathcal{N}(v/2)$
$p \leftarrow \mathcal{N}(\pi)$
$q \leftarrow \mathcal{N}(p^2)$
$h \leftarrow \mathcal{N}(q/3)$
$s \leftarrow \mathcal{N}(g - h)$
if s cannot be exactly rounded according to the given mode $\circ_n$
then return *failed*
else return $\circ_p(n)$

Else, if $x \in [2, 38[$ or if $x \geq 38$ but the above calculation cannot give exact rounding, we use the relation

$$\mathrm{Li}_2(x) = -S\left(-\log(1 - \frac{1}{x})\right) + \frac{\log^2\left(1 - \frac{1}{x}\right)}{4} - \frac{\log^2 x}{2} + \frac{\pi^2}{3},$$

which is computed as follows:

Algorithm `li2_2..+∞`
Input: x with $x \in [2, +\infty[$, the output precision n, and a rounding mode $\circ_n$
Output: $\circ_n(\Re(\mathrm{Li}_2(x)))$

$$
\begin{array}{llll}
y \leftarrow \mathcal{N}(-1/x) & \text{error}(y) & \leq & \frac{1}{2}\text{ulp}(y) \\
u \leftarrow \triangle(-\log(1+y)) & \text{error}(u) & \leq & (1+2^{1-\text{EXP}(u)})\text{ulp}(u) \\
q \leftarrow \mathcal{N}(-S(u)) & \text{error}(q) & \leq & 2(k+1)2^{-\text{EXP}(q)}\text{ulp}(q) \\
v \leftarrow \triangle(u^2) & & & \\
v \leftarrow \triangle(v/4) & \text{error}(v) & \leq & (5+2^{3-\text{EXP}(u)})\text{ulp}(v) \\
r \leftarrow \mathcal{N}(q+v) & \text{error}(r) & \leq & 2^{\kappa_r}\text{ulp}(r) \\
w \leftarrow \triangle(\log x) & & & \\
w \leftarrow \mathcal{N}(w^2) & & & \\
w \leftarrow \mathcal{N}(w/2) & \text{error}(w) & \leq & \frac{9}{2}\text{ulp}(w) \\
s \leftarrow \mathcal{N}(r-w) & \text{error}(s) & \leq & 2^{\kappa_s}\text{ulp}(s) \\
p \leftarrow \triangle(\pi) & & & \\
p \leftarrow \mathcal{N}(p^2) & & & \\
p \leftarrow \mathcal{N}(p/3) & \text{error}(p) & \leq & \frac{19}{2}\text{ulp}(p) \leq 2^{2-\text{EXP}(p)}\text{ulp}(p) \\
t \leftarrow \mathcal{N}(s+p) & \text{error}(t) & \leq & 2^{\kappa_t}\text{ulp}(t)
\end{array}
$$

if t can be exactly rounded according to $\circ_n$
then return $\circ_n(t)$
else increase working precision and restart calculation

with

$$
\begin{array}{lll}
\kappa_r & = & 2+\max(-1, \lceil \log_2(k+1) \rceil + 1 - \text{EXP}(r), 3 + \max(1, -\text{EXP}(u)) + \text{EXP}(v) - \text{EXP}(r)) \\
\kappa_s & = & 2+\max(-1, \kappa_r + \text{EXP}(r) - \text{EXP}(s), 3 + \text{EXP}(w) - \text{EXP}(s)) \\
\kappa_t & = & 2+\max(-1, \kappa_s + \text{EXP}(s) - \text{EXP}(t), 2 - \text{EXP}(t))
\end{array}
$$

When $x \in]1,2[$, we use the relation

$$
\text{Li}_2(x) = S(\log x) + \frac{\log^2 x}{4} - \log x \log(x-1) + \frac{\pi^2}{6}
$$

which is implemented as follows

Algorithm `li2_1..2`
Input: x with $x \in]1,2[$, the output precision n, and a rounding mode $\circ_n$
Output: $\circ_n(\Re(\text{Li}_2(x)))$

$$
\begin{array}{llll}
l \leftarrow \triangle(\log x) & \text{error}(l) & \leq & \text{ulp}(l) \\
q \leftarrow \mathcal{N}(S(l)) & \text{error}(q) & \leq & (k+1)2^{1-\text{EXP}(q)}\text{ulp}(q) \\
u \leftarrow \mathcal{N}(l^2) & & & \\
u \leftarrow \mathcal{N}(u/4) & \text{error}(u) & \leq & \frac{5}{2}\text{ulp}(u) \\
r \leftarrow \mathcal{N}(q+u) & \text{error}(r) & \leq & (3+(k+1)2^{1-\text{EXP}(q)})\text{ulp}(r) \\
y \leftarrow \mathcal{N}(x-1) & \text{error}(y) & \leq & \frac{1}{2}\text{ulp}(y) \\
v \leftarrow \triangle(\log y) & \text{error}(v) & \leq & (1+2^{-\text{EXP}(v)})\text{ulp}(v) \\
w \leftarrow \mathcal{N}(ul) & \text{error}(w) & \leq & (\frac{15}{2}+2^{1-\text{EXP}(v)})\text{ulp}(w) \\
s \leftarrow \mathcal{N}(r-w) & \text{error}(s) & \leq & (11+(k+1)2^{1-\text{EXP}(q)}+2^{1-\text{EXP}(v)})\text{ulp}(s) \\
p \leftarrow \triangle(\pi) & & & \\
p \leftarrow \mathcal{N}(p^2) & & & \\
p \leftarrow \mathcal{N}(p/6) & \text{error}(p) & \leq & \frac{19}{2}\text{ulp}(p) \\
t \leftarrow \mathcal{N}(s+p) & \text{error}(t) & \leq & (31+(k+1)2^{1-\text{EXP}(q)}+2^{1-\text{EXP}(v)})\text{ulp}(t)
\end{array}
$$

if t can be exactly rounded according to $\circ_n$

then return $\circ_n(t)$
else increase working precision and restart calculation

we use the fact that $S(\log x) \geq 0$ and $u \geq 0$ for error(r), that $r \geq 0$ and $-\log x \log(x-1) \geq 0$ for error(s), and that $s \geq 0$ for error(t).

When $x = 1$, we have a simpler value $\mathrm{Li}_2(1) = \frac{\pi^2}{6}$ whose computation is implemented as follows

Algorithm `li2_1`
Input: the output precision p, and a rounding mode $\circ_p$
Output: $\circ_p(\frac{\pi^2}{6})$
$u \leftarrow \triangle(\pi)$
$u \leftarrow \mathcal{N}(u^2)$
$u \leftarrow \mathcal{N}(u/6) \quad \text{error}(u) \leq \frac{19}{2}\text{ulp}(u)$
if u can be exactly rounded according to $\circ_p$
then return $\circ_p(u)$
else increase working precision and restart calculation

When $x \in \left]\frac{1}{2}, 1\right[$, we use the relation

$$\mathrm{Li}_2(x) = -S(-\log x) - \log x \log(1-x) + \frac{\log^2 x}{4} + \frac{\pi^2}{6}$$

which is implemented as follows

Algorithm `li2_0.5..1`
Input: x with $x \in \left]\frac{1}{2}, 1\right[$, the output precision n, and a rounding mode $\circ_n$
Output: $\circ_n(Li2(x))$
$l \leftarrow \triangle(-\log x) \quad \text{error}(l) \leq \text{ulp}(l)$
$q \leftarrow \mathcal{N}(-S(l)) \quad \text{error}(q) \leq (k+1)2^{1-\text{Exp}(q)}\text{ulp}(q)$
$y \leftarrow \mathcal{N}(1-x) \quad \text{error}(y) \leq \frac{1}{2}\text{ulp}(y)$
$u \leftarrow \triangle(\log y) \quad \text{error}(u) \leq (1+2^{-\text{Exp}(v)})\text{ulp}(u)$
$v \leftarrow \mathcal{N}(ul) \quad \text{error}(v) \leq (\frac{9}{2}+2^{1-\text{Exp}(v)})\text{ulp}(v)$
$r \leftarrow \mathcal{N}(q+v) \quad \text{error}(r) \leq 2^{\kappa_r}\text{ulp}(r)$
$w \leftarrow \mathcal{N}(l^2)$
$w \leftarrow \mathcal{N}(u/4) \quad \text{error}(w) \leq \frac{5}{2}\text{ulp}(w)$
$s \leftarrow \mathcal{N}(r+w) \quad \text{error}(s) \leq 2^{\kappa_s}\text{ulp}(s)$
$p \leftarrow \triangle(\pi)$
$p \leftarrow \mathcal{N}(p^2)$
$p \leftarrow \mathcal{N}(p/6) \quad \text{error}(p) \leq \frac{19}{2}\text{ulp}(p) \leq 2^{3-\text{Exp}(p)}\text{ulp}(p)$
$t \leftarrow \mathcal{N}(s+p) \quad \text{error}(t) \leq 2^{\kappa_t}\text{ulp}(t)$
if t can be exactly rounded according to $\circ_n$
then return $\circ_n(t)$
else increase working precision and restart calculation

where

$$\begin{aligned}
\kappa_r &= 2 + \max(3, \lceil \log_2(k+1) \rceil + 1 - \text{Exp}(q), 1 - \text{Exp}(u)) \\
\kappa_s &= 2 + \max(-1, \kappa_r + \text{Exp}(r) - \text{Exp}(s), 2 + \text{Exp}(w) - \text{Exp}(s)) \\
\kappa_t &= 2 + \max(-1, \kappa_s + \text{Exp}(s) - \text{Exp}(t), 3 - \text{Exp}(t))
\end{aligned}$$

Near 0, we can use the relation

$$\mathrm{Li}_2(x) = \sum_{n=0}^{\infty} \frac{x^k}{k^2}$$

which is true for $|x| \le 1$ [FIXME: ref]. If $x \le 0$, we have $0 \le \mathrm{Li}_2(x) - x \le \frac{x^2}{4}$ and if x is positive, $0 \le \mathrm{Li}_2(x) - x \le (\frac{\pi^2}{6} - 1)x^2 \le x^2 \le 2^{2\mathrm{EXP}(x)+1}\mathrm{ulp}(x)$.

When $x \in [-1, 0[$, we use the relation

$$\mathrm{Li}_2(x) = -S(-\log(1-x)) - \frac{\log^2(1-x)}{4}$$

which is implemented as follows

Algorithm `li2_-1..0`
Input: x with $x \in]-1, 0[$, the output precision n, and a rounding mode $\circ_n$
Output: $\circ_n(Li2(x))$

$y \leftarrow \mathcal{N}(1-x)$	$\mathrm{error}(y)$	$\le$	$\frac{1}{2}\mathrm{ulp}(y)$
$l \leftarrow \triangle(\log y)$	$\mathrm{error}(l)$	$\le$	$(1 + 2^{-\mathrm{EXP}(l)})\mathrm{ulp}(l)$
$r \leftarrow \mathcal{N}(-S(l))$	$\mathrm{error}(r)$	$\le$	$(k+1)2^{1-\mathrm{EXP}(r)}\mathrm{ulp}(r)$
$u \leftarrow \mathcal{N}(-l^2)$			
$u \leftarrow \mathcal{N}(u/4)$	$\mathrm{error}(u)$	$\le$	$(\frac{9}{2} + 2^{-\mathrm{EXP}(l)})\mathrm{ulp}(u)$
$s \leftarrow \mathcal{N}(r+u)$	$\mathrm{error}(s)$	$\le$	$2^{\kappa_s}\mathrm{ulp}(s)$

if s can be exactly rounded according to $\circ_n$
then return $\circ_n(s)$
else increase working precision and restart calculation

with

$$\kappa_s = 2 + \max(3, \lceil \log_2(k+1) \rceil + 1 - \mathrm{EXP}(r), -\mathrm{EXP}(l))$$

When x is large and negative, we can use an asymptotic expansion near $-\infty$:

$$\left| \mathrm{Li}_2(x) + \frac{\log^2(-x)}{2} + \frac{\pi^2}{3} \right| \le \frac{1}{|x|}$$

which gives the following algorithm:

Algorithm `li2_asympt_neg`
Input: x with $x \le -7$, the output precision n, and a rounding mode $\circ_n$
Output: $\circ_n(\mathrm{Li}_2(x))$ if it can round exactly, a failure indicator if not
$l \leftarrow \mathcal{N}(\log(-x))$
$f \leftarrow \mathcal{N}(l^2)$
$g \leftarrow \mathcal{N}(f/2)$
$p \leftarrow \mathcal{N}(\pi)$
$q \leftarrow \mathcal{N}(p^2)$
$h \leftarrow \mathcal{N}(q/3)$
$s \leftarrow \mathcal{N}(g-h)$
if s cannot be exactly rounded according to the given mode $\circ_n$
then return *failed*
else return $\circ_n(s)$

When $x \in]-7,-1[$ or if the above computation cannot give exact rounding, we use the relation

$$\mathrm{Li}_2(x) = S\left(\log\left(1-\frac{1}{x}\right)\right) - \frac{\log^2(-x)}{4} - \frac{\log(-x)\log(1-x)}{2} + \frac{\log^2(1-x)}{4} + \frac{\pi^2}{6}$$

which is implemented as follows

Algorithm `li2_-∞..-1`
Input: x with $x \in]-\infty,-1[$, the output precision n, and a rounding mode $\circ_n$
Output: $\circ_n(\mathrm{Li}_2(x))$

$$\begin{array}{llcl}
y \leftarrow \mathcal{N}(-1/x) & & & \\
z \leftarrow \mathcal{N}(1+y) & & & \\
z \leftarrow \mathcal{N}(\log z) & & & \\
o \leftarrow \mathcal{N}(S(z)) & \mathrm{error}(o) & \le & (k+1)2^{1-\mathrm{Exp}(o)}\mathrm{ulp}(o) \\
y \leftarrow \mathcal{N}(1-x) & & & \\
u \leftarrow \triangle(\log y) & \mathrm{error}(u) & \le & (1+2^{-\mathrm{Exp}(u)})\mathrm{ulp}(u) \\
v \leftarrow \triangle(\log(-x)) & \mathrm{error}(v) & \le & \mathrm{ulp}(v) \\
w \leftarrow \mathcal{N}(uv) & & & \\
w \leftarrow \mathcal{N}(w/2) & \mathrm{error}(w) & \le & (\frac{9}{2}+1)\mathrm{ulp}(w) \\
q \leftarrow \mathcal{N}(o-w) & \mathrm{error}(q) & \le & 2^{\kappa_q}\mathrm{ulp}(q) \\
v \leftarrow \mathcal{N}(v^2) & & & \\
v \leftarrow \mathcal{N}(v/4) & \mathrm{error}(v) & \le & \frac{9}{2}\mathrm{ulp}(v) \\
r \leftarrow \mathcal{N}(q-v) & \mathrm{error}(r) & \le & 2^{\kappa_r}\mathrm{ulp}(r) \\
w \leftarrow \mathcal{N}(u^2) & & & \\
w \leftarrow \mathcal{N}(w/4) & \mathrm{error}(w) & \le & \frac{17}{2}\mathrm{ulp}(w) \\
s \leftarrow \mathcal{N}(r+w) & \mathrm{error}(s) & \le & 2^{\kappa_s}\mathrm{ulp}(s) \\
p \leftarrow \triangle(\pi) & & & \\
p \leftarrow \mathcal{N}(p^2) & & & \\
p \leftarrow \mathcal{N}(p/6) & \mathrm{error}(p) & \le & \frac{19}{2}\mathrm{ulp}(p) \le 2^{3-\mathrm{Exp}(p)}\mathrm{ulp}(p) \\
t \leftarrow \mathcal{N}(s-p) & \mathrm{error}(t) & \le & 2^{\kappa_t}\mathrm{ulp}(t)
\end{array}$$

if t can be exactly rounded according to $\circ_n$
then return $\circ_n(t)$
else increase working precision and restart calculation

where

$$\begin{aligned}
\kappa_q &= 1+\max(3, \lceil\log_2(k+1)\rceil + 1 - \mathrm{Exp}(q)) \\
\kappa_r &= 2+\max(-1, \kappa_q + \mathrm{Exp}(q) - \mathrm{Exp}(r), 3 + \mathrm{Exp}(v) - \mathrm{Exp}(r)) \\
\kappa_s &= 2+\max(-1, \kappa_r + \mathrm{Exp}(r) - \mathrm{Exp}(s), 3 + \mathrm{Exp}(w) - \mathrm{Exp}(s)) \\
\kappa_t &= 2+\max(-1, \kappa_s + \mathrm{Exp}(s) - \mathrm{Exp}(t), 3 - \mathrm{Exp}(t))
\end{aligned}$$

4.32. **The Digamma Function.** The Digamma function `mpfr_digamma` is defined by:

$$\psi(x) = \frac{d}{dx}\log\Gamma(x),$$

and is computed from the asymptotic series [25]

$$\psi(x) \sim \log x - \frac{1}{2x} - \sum_{n=1}^{\infty}\frac{B_{2n}}{2nx^{2n}}.$$

(We assume the error in the sum is bounded by the first neglected term.) Since $B_{2n} \approx 2(2n)!/(2\pi)^{-2n}$, the terms of the sum decrease until $n \approx \pi x$, and then the error term is $\approx e^{-2\pi x}$. If x is too small with respect to the target precision, we use the formula [25]:

$$\psi(x+j) = \frac{1}{x+j-1} + \frac{1}{x+j-2} + \cdots + \frac{1}{x} + \psi(x),$$

and compute $\psi(x+j)$ instead with the asymptotic formula.

4.33. **The Airy Function.**

Warning: the current implementation is not made for large arguments. It works fine typically for $|x| \leq 500$. For larger inputs, other methods will be implemented in a close future.

4.33.1. *Definitions.* The Ai function is a solution of the differential equation $y''(x) = x\,y(x)$. It has a power series developed at 0 that is defined for each $x \in \mathbb{C}$ (we will consider only the case $x \in \mathbb{R}$ in the following):

$$\mathrm{Ai}(x) = \sum_{i=0}^{+\infty} a_i\, x^i, \tag{9}$$

where the sequence a_i satisfies the following recurrence:

$$\left\{\begin{array}{rcl} a_0 = \mathrm{Ai}(0) & = & 1/(\Gamma(2/3)\,\sqrt[3]{9}) \\ a_1 = \mathrm{Ai}'(0) & = & -1/(\Gamma(1/3)\,\sqrt[3]{3}) \\ a_2 & = & 0 \\ \forall n \geq 0, a_{i+3} & = & a_i/((i+2)\,(i+3)). \end{array}\right.$$

For each i, we define $t_i = a_i\, x^i$. The sequence $(t_i)_i$ satisfies a similar recurrence.

We denote by C the function defined by

$$C(x) = \sum_{i=0}^{+\infty} |a_i| \cdot |x|^i.$$

This function is involved in the condition number of the series (9).

4.33.2. *Notations.* For the error analysis, we refer to classical techniques and lemmas (see e.g. [13]). In particular, we use Stewart's error counter: $x\,\langle k\rangle$ represents any value $\widehat{x}$ of the form

$$\widehat{x} = x \prod_{i=1}^{k} (1+\theta_i)^{\pm 1}$$

where $|\theta_i| \leq u = 2^{1-p}$ with p the current precision.

It is known that $x\,\langle k\rangle = x(1+\mu)$ where $|\mu| \leq \gamma_k$ with $\gamma_k \leq 2ku$ when $2ku \leq 1$ (see [13]).

As usual, we denote by $\textsc{Exp}(x)$ the exponent of x (the smallest $E \in \mathbb{Z}$ such that $|x| < 2^E$).

4.33.3. *Technical results.*

Fact 1. *The function C satisfies the following inequalities:*

$$\left\{\begin{array}{rl} C(x) \leq 1 & \text{if} \quad 0 \leq x < 1 \\ C(x) \leq \frac{1}{2}\, x^{-1/4} \exp\left(\frac{2}{3}\, x^{3/2}\right) & \text{if} \quad x \geq 1 \end{array}\right. \tag{10}$$

Fact 2. *For $x \in [1/4, 1]$, the following holds:*

$$\begin{cases} x/\Gamma(x) & \in & [1/16,\, 1] \\ \Gamma'(x) & \in & [-16,\, -1/2]. \end{cases}$$

So, if $a = \circ(1/3)$ computed in precision $\mathsf{wprec}+4$, then we can write $\Gamma(a) = \Gamma(1/3)\,\langle 1\rangle$ in precision wprec (the same holds for $a = \circ(2/3)$).

Proof. Let $\alpha = 1/3$ or $\alpha = 2/3$. We suppose that $a = \circ(\alpha)$. So we can write $a = (1+\theta)\,\alpha$ where $|\theta| \le 2^{-3-\mathsf{wprec}}$, so we have

$$\Gamma(a) = \Gamma(\alpha + \alpha\,\theta) = \Gamma(\alpha) + \alpha\,\theta\Gamma'(\xi),$$

where ξ lies between α and a. In particular, $\xi \in [1/4,\, 1]$. Thus

$$\Gamma(a) = \Gamma(\alpha)\left(1 + \theta\frac{\alpha}{\Gamma(\alpha)}\,\Gamma'(\xi)\right) = \Gamma(\alpha)\,(1+\theta')$$

with $|\theta'| \le 16|\theta| \le 2^{1-\mathsf{wprec}}$. ∎

Algorithm: `mpfr_ai`

`/* except when mentioned, the precision used is wprec */`

$\mathsf{temp} \leftarrow \circ(2/3)$; `/* with precision` $\mathsf{wprec}+4$ `*/`

$\mathsf{temp} \leftarrow \Gamma(\mathsf{temp})$;

$\mathsf{t0} \leftarrow \sqrt[3]{9}$;

$\mathsf{t0} \leftarrow \mathsf{t0} \cdot \mathsf{temp}$;

$\mathsf{t0} \leftarrow 1/\mathsf{t0}$;

$\mathsf{temp} \leftarrow \circ(1/3)$; `/* with precision` $\mathsf{wprec}+4$ `*/`

$\mathsf{temp} \leftarrow \Gamma(\mathsf{temp})$;

$\mathsf{t1} \leftarrow \sqrt[3]{3}$;

$\mathsf{t1} \leftarrow \mathsf{t1} \cdot \mathsf{temp}$;

$\mathsf{t1} \leftarrow -x/\mathsf{t1}$;

$\mathsf{s} \leftarrow \mathsf{t0} + \mathsf{t1}$;

$\mathsf{k} \leftarrow 2$;

while true do

  $\mathsf{t0} \leftarrow \mathsf{t0} \cdot x^3/(\mathsf{k}\cdot(\mathsf{k}+1))$;

  $\mathsf{t1} \leftarrow \mathsf{t1} \cdot x^3/((\mathsf{k}+1)\cdot(\mathsf{k}+2))$;

  $\mathsf{k} \leftarrow \mathsf{k}+3$;

  $\mathsf{s} \leftarrow \mathsf{s} + \mathsf{t0} + \mathsf{t1}$;

  if $(\mathrm{Exp}(\mathsf{t0}) \le \mathrm{Exp}(\mathsf{s}) - \mathsf{prec} - 3)$

  and $(\mathrm{Exp}(\mathsf{t1}) \le \mathrm{Exp}(\mathsf{s}) - \mathsf{prec} - 3)$

  and $|x|^3 \le \mathsf{k}\cdot(\mathsf{k}+1)/2$break;

end

$\mathsf{evalErr} \leftarrow 4 + \log_2(k) - \mathrm{Exp}(\mathsf{s})$;

if $|x| \ge 1$ **then** $\mathsf{evalErr} \leftarrow \mathsf{evalErr} + (2/3)\log_2(e)x^{3/2} - \log_2(x)/4 - 1$;

$\mathsf{correctBits} \leftarrow \min(\mathsf{prec}+1,\, \mathsf{wprec} - \mathsf{evalErr}) - 1$;

4.33.4. *Algorithm.* Algorithm 1 is run to obtain an approximate value of $\mathrm{Ai}(x)$ with a relative error bounded by $2^{-\mathsf{prec}}$. Except if this is explicitly mentioned, the operations are performed with correct rounding and with precision wprec. The link between prec and wprec will be expressed in Section 4.33.4.

Analysis : it is clear that, while entering in the loop for the j-th time, $k = 3j-1$, $\mathsf{t0} \simeq t_{3j-3}$, $\mathsf{t1} \simeq t_{3j-2}$ and $\mathsf{s} \simeq \sum_{i=0}^{k} t_i$.

Let K be the value of k when exiting the loop. Hence $K = 3j+2$ where j denotes the number of times that the loop has been performed. Moreover $\mathsf{s} \simeq \sum_{i=0}^{K} t_i$, $\mathsf{t0} \simeq t_{3j}$, and $\mathsf{t1} \simeq t_{3j+1}$

Hypothesis 1. *In the following, we suppose that* $2 \cdot (4K) \cdot 2^{1-\mathsf{wprec}} \leq 1$.

Roundoff errors. Before entering the loop, $\mathsf{t0} = t_0 \langle 4 \rangle$ and $\mathsf{t1} = t_1 \langle 4 \rangle$. A trivial recurrence shows that, at the end of the j-th execution of the loop, we have $\mathsf{t0} = t_{3j} \langle 4+5j \rangle$ and $\mathsf{t1} = t_{3j+1} \langle 4+5j \rangle$. At the end of the j-th execution of the loop, s has been obtained by the accumulation of $2j+1$ additions. So, when exiting the loop, all the terms of the sum have accumulated at most $4+5j+2j+1 = 7j+5$ errors, which we conveniently bound by $4K$ (remember that $K = 3j+2$). So we can write

$$\mathsf{s} = \sum_{k=0}^{K} t_k \langle 4K \rangle .$$

For each k, we know that $t_k \langle 4K \rangle = t_k(1+\mu^{(k)})$ where

$$|\mu^{(k)}| \leq \gamma_{4K} \leq 2 \cdot (4K) \cdot 2^{1-\mathsf{wprec}} \quad \text{(here we use the hypothesis 1).}$$

Hence

$$\mathsf{s} = \left(\sum_{k=0}^{K} t_k \right) + \left(\sum_{k=0}^{K} t_k \, \mu^{(k)} \right),$$

and we can bound the roundoff errors by

$$\left| \mathsf{s} - \sum_{i=0}^{K} t_i \right| \leq \sum_{i=0}^{K} |t_i| \, \gamma_{4K} \leq \gamma_{4K} \underbrace{\sum_{i=0}^{+\infty} |t_k|}_{C(x)} \leq C(x) \cdot (8K \, 2^{1-\mathsf{wprec}}).$$

We remark that, by definition of $\mathsf{evalErr}$ and by Fact 1, we have

$$C(x) \leq 2^{\mathrm{Exp}(\mathsf{s})+\mathsf{evalErr}-4-\log_2(K)}.$$

So, finally

$$\boxed{\left| \mathsf{s} - \sum_{i=0}^{K} t_i \right| \leq 2^{\mathrm{Exp}(\mathsf{s})+\mathsf{evalErr}-\mathsf{wprec}}.}$$

Remark: during the algorithm $\mathsf{t0} = t_{3j} \langle 4+5j \rangle$, so in particular $|t_{3j}| \leq 2|\mathsf{t0}|$. The same remark holds for $\mathsf{t1}$ and t_{3j+1}.

Approximation error. The stopping criterion ensures that, for $n \geq K$, $|t_{n+1}| \leq |t_{n-2}|/2$. Besides, we recall that $K = 3j+2$. Thus

$$\left|\sum_{k=j+1}^{+\infty} t_{3k}\right| \le \frac{t_{3j}}{2} + \frac{t_{3j}}{4} + \frac{t_{3j}}{8} + \cdots \le |t_{3j}|.$$

Using the remark above, we can bound $|t_{3j}|$ by $2|\mathsf{t0}| \le 2^{1+\textsc{Exp}(\mathsf{t0})}$. The stopping criterion of the loop ensures that $\textsc{Exp}(\mathsf{t0}) \le \textsc{Exp}(\mathsf{s}) - \mathsf{prec} - 3$ so we conclude that

$$\left|\sum_{k=j+1}^{+\infty} t_{3k}\right| \le 2^{\textsc{Exp}(\mathsf{s})-\mathsf{prec}-2}.$$

Likewise,

$$\left|\sum_{k=j+1}^{+\infty} t_{3k+1}\right| \le 2^{\textsc{Exp}(\mathsf{s})-\mathsf{prec}-2}.$$

From this, we deduce the following upper bound on the approximation error:

$$\boxed{\left|\mathrm{Ai}(x) - \sum_{i=0}^{K} t_i\right| \le 2^{\textsc{Exp}(\mathsf{s})-\mathsf{prec}-1}.}$$

Overall error. By definition of $\mathsf{correctBits}$, we finally get

$$\boxed{|\mathrm{Ai}(x) - \mathsf{s}| \le 2^{\textsc{Exp}(\mathsf{s})-\mathsf{correctBits}}.}$$

Determination of wprec. The variable prec must be chosen slightly larger than the final target precision, in order to bypass the TMD. In practice we keep a few guard bits, which ensures that we do not encounter bad cases too often. The Ziv' loop is performed over prec.

We would like the roundoff errors and the approximation error to be approximately of the same order of magnitude, i.e.

$$\mathsf{wprec} = \mathsf{prec} + 1 + \mathsf{evalErr}.$$

The value $\mathsf{evalErr}$ depends on $\textsc{Exp}(\mathsf{s})$ (that we do not know, *a priori*) and on K (idem). We may estimate $K \simeq \mathsf{prec}$ (anyway, only the logarithm of this value is used, so we do not care too much of this value) Concerning $\textsc{Exp}(\mathsf{s})$, when $x \ge 0$, it is possible to rigorously estimate it with the following inequalities. When $x < 0$, it is more obfuscated since $\mathrm{Ai}(x)$ can be arbitrarily close to zero. I do not have any estimation yet.

Fact 3. *The following inequalities hold:*

$$\begin{cases} \mathrm{Ai}(x) \ge 1/8 & \text{if} \quad 0 \le x \le 1 \\ \mathrm{Ai}(x) \ge \frac{1}{4}\, x^{-1/4} \exp\left(-\frac{2}{3}\, x^{3/2}\right) & \text{if} \quad x \ge 1. \end{cases}$$

These estimates are used to set the initial value of wprec. When $x \le 0$, we initially suppose that $\textsc{Exp}(\mathrm{Ai}(x)) \ge -10$. More precisely, we use a variable $\mathsf{assumedExp}$ to remember this assumption (initially $\mathsf{assumedExp} = 10$).

Representing the condition number by a variable cond, and using Fact 1 we can set

$$\begin{cases} \mathsf{cond} = 0 & \text{if} \quad |x| \le 1 \\ \mathsf{cond} = \lceil \frac{2}{3} \log_2(e) x^{3/2} \rceil - \lfloor \frac{\log_2(x)}{4} \rfloor - 1 & \text{if} \quad |x| \ge 1. \end{cases}$$

Using Fact 3, we can set the initial value of wprec the following way:

$$\begin{array}{lll} \mathsf{wprec} = \mathsf{prec} + 1 + 4 + \lceil \log_2(\mathsf{prec}) \rceil + \mathsf{cond} + \mathsf{assumedExp} & \text{if} & x \le 0 \\ \mathsf{wprec} = \mathsf{prec} + 1 + 4 + \lceil \log_2(\mathsf{prec}) \rceil + \mathsf{cond} + 3 & \text{if} & 0 \le x \le 1 \\ \mathsf{wprec} = \mathsf{prec} + 1 + 4 + \lceil \log_2(\mathsf{prec}) \rceil + \mathsf{cond} + 2 + \lceil \frac{2}{3} \log_2(e) x^{3/2} \rceil + \lceil \frac{\log_2(x)}{4} \rceil & \text{if} & x > 1. \end{array}$$

When the algorithm exits the loop (line 22 of the algorithm), several cases are possible:

- correctBits can be negative: this typically happens when $x \geq 0$ and the $\mathrm{Ai}(x)$ is almost zero. In this case, the initial assumption that $\mathrm{Exp}(\mathrm{Ai}(x)) \geq -\mathsf{presumedExp}$ is false and wprec was badly chosen. This is **not** due to a bad case in Ziv' strategy. We choose to double assumedExp and set the new value of wprec as

$$\mathsf{wprec} = \mathsf{prec} + 1 + 4 + \lceil \log_2(\mathsf{k}) \rceil + \mathsf{cond} + \mathsf{assumedExp};$$

- correctBits can be positive but smaller than prec. The cause of this phenomenon is the same as in the previous case. However, since we have at least one correct bit, we get an important information: s is a first approximation of $\mathrm{Ai}(x)$. Hence we do not rely on assumedExp for choosing the new value of wprec:

$$\mathsf{wprec} = \mathsf{prec} + 1 + 4 + \lceil \log_2(\mathsf{k}) \rceil + \mathsf{cond} - \mathrm{Exp}(\mathsf{s});$$

- finally, if correctBits $\geq$ prec but if we cannot round, it means that we really are in a bad case of Ziv' strategy. In this case, we update prec (according to the usual MPFR strategy) and we recompute a new working precision from it: as in the previous case, we can rely on $\mathrm{Exp}(\mathsf{s})$. The only unknown is the new truncation rank. We assume that it will not be multiplied by more than 4 and we set:

$$\mathsf{wprec} = \mathsf{prec} + 1 + 4 + \lceil \log_2(4\mathsf{k}) \rceil + \mathsf{cond} - \mathrm{Exp}(\mathsf{s}).$$

4.34. **Radix Conversion.** The `mpfr_get_str` function with size 0 and base b chooses an output precision of $1 + \lceil e \log(2)/\log(b) \rceil$ for a precision of e bits if b is not a power of two [17]. However, the code uses instead $1 + \lceil ey \rceil$, where y is an upper 76-bit approximation of $\log(2)/\log(b)$. When do both values differ? In the case $b = 2^k$, the worst case is when the first output digit contains only one significant bit, thus $1 + \lceil (e-1)\log(2)/\log(b) \rceil$ digits are necessary, and also sufficient.

Let y be the 76-bit upper approximation of $x = \log(2)/\log(b)$. Both values differ when there is an integer n such that $xe \leq n < ye$, i.e., $x < n/e < y$. This means that n/e is a better approximation of x than y. Let p/q be the first convergent of x such that $|x - p/q| < |x - y|$, then necessarily $e \geq q$.

Example: for $b = 10$ we have

$$y = \frac{45490366779583341627641}{2^{77}},$$

with $|x - y| \approx 0.3 \cdot 2^{-23}$. The first convergent such that $|x - p/q| < |x - y|$ is $p/q = 174131244785/578451474249$, and thus for $e < 578451474249$ the formula $1 + \lceil ex \rceil$ is correct. In fact for $e = 578451474249$ it is exact too, since $p/q < x$. To improve the bound we can consider semi-convergents $(p_{k-1} + ap_k)/(q_{k-1} + aq_k)$ with $a = 1, 2, \ldots$, where $p_k/q_k = p/q$ here. In this example this gives the bound $e < 1074541795081$ for $a = 1$. We get the following bounds (for powers of two there is no error), checked independently by Mark Dickinson:

3, 975675645481	21, 500866275153	37, 1412595553751	52, 4234025992181
5, 751072483167	22, 1148143737877	38, 2296403499681	53, 1114714558973
6, 880248760192	23, 2963487537029	39, 227010038198	54, 653230957562
7, 186564318007	24, 930741237529	40, 3574908346547	55, 1113846215983
9, 1951351290962	25, 751072483167	41, 458909109357	56, 385930970803
10, 1074541795081	26, 1973399062219	42, 1385773590791	57, 676124411642
11, 890679595344	27, 1193652440098	43, 945885487008	58, 330079387370
12, 727742578896	28, 319475419871	44, 1405607880410	59, 276902299279
13, 1553566199646	29, 1645653531910	45, 421759749499	60, 2304608467893
14, 253019868939	30, 1190119072066	46, 376795094250	61, 1364503143363
15, 947578699731	31, 2605117443408	47, 1352868311988	62, 414481628603
17, 628204683310	33, 1138749817330	48, 1133739896162	
18, 2280193268258	34, 1611724268329	49, 186564318007	
19, 2290706306707	35, 820222240621	50, 842842574535	
20, 645428387961	36, 1760497520384	51, 1435927298893	

The smallest bound is $e = 186564318007$ for $b = 7$ and $b = 49$.

4.35. **Summary.** Table 1 presents the 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 $1 \pm k_u 2^{1-p}$.

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

TABLE 1. Generic error

Remark : 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 $\mathrm{err}(w)/\mathrm{ulp}(w) \le c_w + \max(k_u + k_v/2, k_u/2 + k_v) \le c_w + \frac{3}{2}\max(k_u, k_v)$.

5. Mathematical constants

5.1. **The constant π.** The computation of π uses the Brent-Salamin 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 [23], 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 (see [2] for a formal proof of the error analysis):

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

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 $|\epsilon'| \le 1$.

5.2. **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 [5]. (See also [25].) 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 [5], 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 [5]:

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

Let us introduce $I_k = \int_n^\infty \frac{e^{-u}}{u^k} \mathrm{d}u$. 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 $n \geq 1$, we have:

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

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 ϵ_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. $n \geq \mathrm{prec}\frac{\log 2}{2}$:

[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 $x \geq 1$, we have $0 < \frac{\theta}{6x} < 1$, and $e^t < 1 + 2t$ for $0 < t < 1$, thus $(x!)^2 \leq 2\pi x^{2x} e^{-2x}(x + \frac{1}{3})$.

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

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

5.2.1. *A faster formula.* Brent and McMillan give in [6] a faster algorithm (B2) using the modified Bessel functions, which was used by Gourdon and Demichel to compute 108,000,000 digits of γ 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 $I_0 \ge \frac{e^{2n}}{\sqrt{4\pi n}}$ (see [6] 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 $K_0 \le \sqrt{\frac{\pi}{4n}} e^{-2n}$ and $\frac{K_0}{I_0} \le \pi e^{-4n}$ as in formula (5) of [6]. Let $I'_0 = \sum_{k=0}^{K-1} \frac{n^{2k}}{(k!)^2}$ with $K = \lceil \beta n \rceil$, and β is the root of $\beta(\log \beta - 1) = 3$ ($\beta \sim 4.971\ldots$) then

$$|I_0 - I'_0| \le \frac{\beta}{2\pi(\beta^2-1)} \frac{e^{-6n}}{n}.$$

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:

$$|K_0 - K'_0| \le \frac{(8n+1)}{16\sqrt{2n}} \frac{e^{-6n}}{n} \le \frac{1}{2} \frac{e^{-6n}}{n}.$$

We get from this

$$\left| \frac{K_0}{I_0} - \frac{K'_0}{I'_0} \right| \le \frac{1}{2I_0} \frac{e^{-6n}}{n} \le \sqrt{\frac{\pi}{n}} e^{-8n}.$$

Let $S'_0 = \sum_{k=1}^{K-1} \frac{n^{2k}}{(k!)^2} H_k$, then using $\frac{H_{k+1}}{H_k} \le \frac{k+1}{k}$ and the same bound K than for I'_0 ($4n \le K \le 5n$), we get:

$$|S_0 - S'_0| \le \frac{\beta}{2\pi(\beta^2-1)} H_k \frac{e^{-6n}}{n}.$$

We deduce:

$$\left| \frac{S_0}{I_0} - \frac{S'_0}{I'_0} \right| \le e^{-8n} H_K \frac{\sqrt{4\pi n}}{\pi(\beta^2-1)} \frac{\beta}{n} \le e^{-8n}.$$

Hence we have

$$\left| \gamma - \left(\frac{S'_0 - K'_0}{I'_0} - \log n \right) \right| \le (1 + \sqrt{\frac{\pi}{n}}) e^{-8n} \le 3e^{-8n}.$$

5.3. **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 [11]):

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

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 $w \geq 3$.

5.4. **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 $4/3 f(K) \leq 4/3 \cdot 4^{-K}$. 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]

[4] `https://web.archive.org/web/20090624123133/http://www-2.cs.cmu.edu/~adamchik/articles/catalan/catalan.htm`

$x \leftarrow \circ(\sqrt{3})$ [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 $w \geq 2$ —, $\text{ulp}(x) = 1/2\text{ulp}(y)$, thus the error on y is at most $3/2\text{ulp}(y)$. The generic error on the logarithm (§2.9) gives an error bound of $1 + \frac{3}{2} \cdot 2^{2-\text{Exp}(z)} = 4$ ulps for z (since $3 \leq y < 4$, we have $1 \leq z < 2$, so $\text{Exp}(z) = 1$). 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 11 ulps. Finally that on g is at most $11 + 9/2 = 31/2$ ulps. Taking into account the truncation error, this gives $33/2$ ulps.

References

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

[2] Bertot, Y., Rideau, L., and Théry, L. Distant decimals of

π

. *Journal of Automated Reasoning* (2017), 1–45.

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

[4] Borwein, P. An efficient algorithm for the Riemann zeta function, Jan. 1995. 9 pages, `http://www.cecm.sfu.ca/~pborwein/PAPERS/P155.pdf`.

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

[6] 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.

[7] Brent, R. P., and Zimmermann, P. *Modern Computer Arithmetic.* Version 0.1.1, 2006. `https://members.loria.fr/PZimmermann/mca/pub226.html`.

[8] Demmel, J., and Hida, Y. Accurate floating point summation. `https://people.eecs.berkeley.edu/~demmel/AccurateSummation.ps`, May 2002.

[9] Gautschi, W. A computational procedure for incomplete gamma functions. *ACM Trans. Math. Softw. 5*, 4 (1979), 466–481.

[10] Ginsberg, E. S., and Zaborowski, D. The dilogarithm function of a real argument [s22]. *Communications of the ACM 18*, 4 (April 1975), 200–202.

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

[12] Harvey, D., and Zimmermann, P. Short division of long integers. In *Proceedings of the 20th IEEE Symposium on Computer Arithmetic* (2011), E. Antelo, D. Hough, and P. Ienne, Eds., IEEE Computer Society, pp. 7–14.

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

[14] Hull, T. E., and Abrham, A. Variable precision exponential function. *ACM Trans. Math. Softw. 12*, 2 (1986), 79–91.

[15] Jeandel, E. Évaluation rapide de fonctions hypergéométriques. Rapport technique 242, Institut National de Recherche en Informatique et en Automatique, July 2000. 17 pages, `https://inria.hal.science/inria-00069930`.

[16] Jones, C. B. A significance rule for multiple-precision arithmetic. *ACM Trans. Math. Softw. 10*, 1 (1984), 97–107.

[17] Matula, D. W. In-and-out conversions. *Communications of the ACM 11*, 1 (1968), 47–50.

[18] Muller, J.-M. *Elementary Functions. Algorithms and Implementation.* Birkhauser, 1997.

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

[20] Pétermann, Y.-F. S., and Rémy, J.-L. Arbitrary precision error analysis for computing $\zeta(s)$ with the Cohen-Olivier algorithm: Complete description of the real case and preliminary report on the general case. Research Report 5852, INRIA, 2006.

[21] Pétermann, Y.-F. S., and Rémy, J.-L. On the Cohen-Olivier algorithm for computing $\zeta(s)$: Error analysis in the real case for an arbitrary precision. *Advances in Applied Mathematics 38* (2007), 54–70.

[22] 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`.

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

[24] Smith, D. M. Algorithm 693. a Fortran package for floating-point multiple-precision arithmetic. *ACM Trans. Math. Softw. 17*, 2 (1991), 273–283.

[25] Smith, D. M. Algorithm 814: Fortran 90 software for floating-point multiple precision arithmetic, gamma and related functions. *ACM Trans. Math. Softw. 27*, 4 (2001), 377–387.

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

[27] Temme, N. M. *Special Functions. An Introduction to the Classical Functions of Mathematical Physics.* John Wiley & Sons, Inc., 1996.

[28] Vollinga, J., and WeinZierl, S. Numerical evaluation of multiple polylogarithms. *Computer Physics Communications 167* (2005), 177–194.