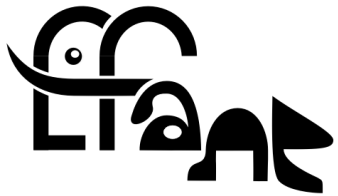
Ball arithmetic in MPC

Andreas Enge

CANARI project-team
INRIA Bordeaux
andreas.enge@inria.fr
http://enge.math.u-bordeaux.fr/

MPFR/MPC/MPFI/FLINT Workshop, Bordeaux, June 17–19, 2024



Dagstuhl 2018: New number type to implement

Complex ball arithmetic

- Used internally to replace hand calculations of error bounds
- As a tool to implement Taylor and Laurent series
- As a building block for polynomials, class polynomials, etc.
- Only four basic arithmetic operations and square root?
- Need for real ball arithmetic? interface with MPFI?
- Representation as rectangles? as balls?

Dagstuhl 2018: New number type to implement

Complex ball arithmetic

- Used internally to replace hand calculations of error bounds
- As a tool to implement Taylor and Laurent series
- As a building block for polynomials, class polynomials, etc.
- Only four basic arithmetic operations and square root?
- Need for real ball arithmetic? interface with MPFI?
- Representation as rectangles? as balls?

Release 1.3.0 “Ipomoea batatas”, December 2022:

- Ball arithmetic (marked experimental)
- New function `mpc_agm`
- New function `mpc_eta_fund`

Algorithm for AGM

If any of x_1, y_1, x_2 or y_2 equals 0, then the algorithm has either finished in the second step, or the corresponding norm is computed exactly already at precision $2p < p'$, a contradiction. So the case of interest is that these four input values are non-zero. Then write $z_k = 2^{e_k}(x'_k + i2^{d_k}y'_k)$ with integers x'_k and y'_k satisfying $2^{p-1} \leqslant x'_k, y'_k < 2^p$, and furthermore $d_k \geqslant 0$ since $y_k \geqslant x_k$. We have $n_k = 2^{2e_k} \circ (m_k)$ with the integer $m_k = (x'_k)^2 + 2^{2d_k}(y'_k)^2$. If any of the exponent differences satisfies $d_k \leqslant p$, then $m_k < 2^{4p} \leqslant 2^{p'}$, and n_k is computed exactly at precision p', a contradiction.

So d_1, $d_2 > p$. The integer m_k, read from the lowest to the most significant bits, consists of $(x'_k)^2$ encoded with $2p$ bits (with potentially a leading bit 0), followed by $2d_k - 2p \geqslant 2$ bits 0, followed by $(y'_k)^2$ encoded with $2p$ bits (with potentially one leading bit 0). So the $2p$ most significant bits of m_k, excluding leading bits 0, have the following shape: If $y'_k > \sqrt{2} \cdot 2^{p-1}$, they encode exactly $(y'_k)^2$. If $y'_k < \sqrt{2} \cdot 2^{p-1}$, the $2p-1$ leading bits encode $(y'_k)^2$, and they are followed by a digit 0, so the $2p$ leading bits encode $2(y'_k)^2$.

Since $n_1 = n_2$, in particular their leading $2p$ bits coincide; so either $(y'_1)^2 = (y'_2)^2$, $(y'_1)^2 = 2(y'_2)^2$ or $2(y'_1)^2 = (y'_2)^2$, where the last two cases are impossible for integers. Comparing exponents of the n_k yields $e_1 + d_1 = e_2 + d_2$, so $y_1 = y_2$, a contradiction since the algorithm has proceeded beyond the second step. ∎

3.10 `mpc_agm`

ssec:agm

Definition. Let a, b be non-zero complex numbers. Define sequences of arithmetic means (a_n) and geometric means (b_n) by $a_0 = a$, $b_0 = b$, $a_{n+1} = \frac{a_n + b_n}{2}$ and $b_{n+1} = \sqrt{a_n b_n}$. At each step, there is a choice of sign for the square root. If a_n and b_n form an (unoriented) angle different from 0 and π, then they define a two-dimensional pointed cone in the complex plane. Notice that a_{n+1} lies in this cone. Following [4] we call *right* the choice that makes also b_{n+1} lie in the cone, and following [5] we call the resulting sequences *optimal*. (An equivalent definition is that $|a_{n+1} - b_{n+1}| < |a_{n+1} + b_{n+1}|$.) There is a common limit of the sequences, the *arithmetic-geometric mean* $\mathrm{AGM}(a, b)$.

It is immediate that AGM is symmetric, that is, $\mathrm{AGM}(a, b) = \mathrm{AGM}(b, a)$, and homogeneous, that is, $\mathrm{AGM}(\lambda a, \lambda b) = \lambda\,\mathrm{AGM}(a, b)$ for any non-zero complex number λ.

So we may assume that $|a| \geqslant |b|$, and $\mathrm{AGM}(a, b) = a\,\mathrm{AGM}(a_0, b_0)$ with $a_0 = 1$, $b_0 = b/a$, $|b_0| \leqslant 1$.

We need to examine the corner cases. If one or both of a and b are zero, all geometric means are zero, and $\mathrm{AGM}(a, b) = 0$. If the angle between a and b is 0, then b_0 is a positive real number, and $\mathrm{AGM}(1, b_0)$ may be computed with `mpfr`. If the angle between a and b is π, then b_0 is in $[-1, 0)$. The arithmetic mean of 1 and -1 is zero, so $\mathrm{AGM}(1, -1) = 0$. If $b_0 > -1$, we take the first geometric mean with a positive imaginary part, so that also $\Im(\mathrm{AGM}(1, b_0))$ will be positive. Notice that $\Im(\mathrm{AGM}(1, b_0))$ has the same sign as $\Im(b_0)$ unless b_0 is real, so this choice determines our branch cut for AGM.

So in the following, we analyse the computation of $\mathrm{AGM}(1, \widetilde{b_0})$ with $\widetilde{b_0} = \circ(b_0)$ for b_0 in the unit disk centered at the origin (except -1 and 1), where the real and imaginary parts of $\widetilde{b_0}$ are rounded towards 0, which ensures $\left|\Re(\widetilde{b_0})\right| < 1$ and $|\widetilde{b_0}| \leqslant 1$ with absolute errors of the real and imaginary parts at most 1 ulp. In the following, relative errors will

be most convenient to work with; by Proposition 10 we have $\mathrm{relerror}(\widetilde{b_0}) \leqslant 2^{1-p}$, where $p \geqslant 2$ is the working precision.

Warm-up — a note on angle 0. As said, when the angle between a and b is 0, which is immediately detected from $\Im(\widetilde{b_0}) = 0$ (computed at any precision), then $\mathrm{AGM}(a, b) = a\,\mathrm{AGM}(1, b_0)$ with a real number $0 < b_0 < 1$. We need to consider the error induced by computing $\circ\left(a \cdot \circ\left(\mathrm{AGM}(1, \widetilde{b_0})\right)\right)$ instead.

Recall that $b_0 = (1 + \vartheta)\widetilde{b_0}$ with $-\varepsilon \leqslant \vartheta \leqslant \varepsilon = 2^{1-p}$. Since for positive real numbers each step of the AGM is increasing in each of the two arguments, so is the AGM itself, and we have

$$\mathrm{AGM}(1, b_0) = \mathrm{AGM}\left(1, (1+\vartheta)\widetilde{b_0}\right) \leqslant \mathrm{AGM}\left(1+\varepsilon, (1+\varepsilon)\widetilde{b_0}\right) \leqslant (1+\varepsilon)\,\mathrm{AGM}\left(1, \widetilde{b_0}\right).$$

By the same kind of argument

$$\mathrm{AGM}(1, b_0) \geqslant (1-\varepsilon)\,\mathrm{AGM}\left(1, \widetilde{b_0}\right).$$

So the relative error of at most ε in the input $\widetilde{b_0}$ is preserved by the AGM.

By Proposition 7 applied to non-representable numbers and Proposition 3 this relative error translates into an absolute error satisfying

$$\left|\mathrm{AGM}(1, b_0) - \mathrm{AGM}\left(1, \widetilde{b_0}\right)\right| \leqslant 2 \cdot 2^{\mathrm{Exp}\left(\mathrm{AGM}(1, \widetilde{b_0})\right) - p} \leqslant 2 \cdot 2^{\mathrm{Exp}\left(\circ\left(\mathrm{AGM}(1, \widetilde{b_0})\right)\right) - p}$$

of at most 2 ulp of the rounded value, and the final rounding of the `mpfr_agm` function leads to a total error bounded by 3 ulp.

We multiply this value by the exact a; applying (2) with $k_1 = 3$ and $k_2 = 0$ and taking the final rounding into account leads to an error of at most 7 ulp for the real and for the imaginary parts of the complex AGM.

The first iteration — entering a quadrant. If $\Re(\widetilde{b_0}) < 0$, then significant cancellation can occur for the arithmetic mean in the first iteration, which thus needs to be analysed separately.

From now on, we use arbitrary rounding modes and apply Proposition 11 with $c = 1$. We let $b_1 = \sqrt{b_0}$ and $\widetilde{b_1} = \circ\left(\sqrt{\widetilde{b_0}}\right)$ with

$$\mathrm{relerror}(\widetilde{b_1}) \leqslant 2^{1-p} + \left(1 + 2^{1-p}\right) 2^{1-p} \leqslant (\sqrt{2})^3 \cdot 2^{1-p}$$

by (18) (where we use $\varepsilon \leqslant \varepsilon_1$ since $\varepsilon_1 \leqslant 2^{1-p} \leqslant 1$, ε_1 being the relative error on $\widetilde{b_0}$) and Proposition 11, and where we have bounded $2 + 2^{1-p} \leqslant 2.5$ by $(\sqrt{2})^3 \approx 2.83$. We let $a_1 = 1/2 + b_0/2$ and $\widetilde{a_1} = \circ(1/2 + \widetilde{b_0}/2)$. The imaginary part of $\widetilde{a_1}$ has an error of at most 1 ulp, and the same holds for the real part if $\Re(\widetilde{b_0}) = 0$ or equivalently $\Re(b_0) = 0$; the real part of $\widetilde{a_1}$ has an absolute error bounded by

$$\max\left(2, 2^{-\mathrm{Exp}(\Re(\widetilde{a_1}))-1}\right) \mathrm{ulp}\left(\Re(\widetilde{a_1})\right).$$

Algorithm for AGM

Indeed, let x be the real part of $\widetilde{b_0}$, and y the real part of $\widetilde{a_1}$, we have $y = \circ(1/2 + x/2)$. Remember that $-1 < x < 1$. If $-1/2 \leqslant x < 1$, then $x/2 \geqslant -1/4$ and $y \geqslant 1/4$, and $\mathrm{Exp}(y) \geqslant \mathrm{Exp}(x/2)$, and from §1.3.1 the error on y is at most 2 ulp (one from the error on $x/2$, and one from the addition). If on the other hand $-1 < x < -1/2$, then $1/2 + x/2$ is exact by Sterbenz's lemma, and the only error comes from that on $x/2$, which by §1.3.1 is bounded above by $2^{\mathrm{Exp}(x/2)-\mathrm{Exp}(y)}\mathrm{ulp}(x/2)$. Since $-1/2 < x/2 < -1/4$, we have $\mathrm{Exp}(x/2) = -1$ and the above result follows. This bound also holds when $\Re(b_0) = 0$. So by Proposition 10,

$$\mathrm{relerror}(\widetilde{a_1}) \leqslant \max\left(2, 2^{-\mathrm{Exp}(\Re(\widetilde{a_1}))-1}\right) \cdot 2^{1-p}.$$

If $\Re(\widetilde{b_0}) \geqslant 0$, then $\mathrm{Exp}(\Re(\widetilde{a_1})) \geqslant 0$, and we obtain a bound that is similar to the one above for $\mathrm{relerror}(\widetilde{b_1})$. If $\widetilde{b_0}$ approaches -1, the relative error in $\widetilde{a_1}$ becomes arbitrarily bad, as $\widetilde{a_1}$ becomes arbitrarily small.
Letting

$$k_1 = \max\left(3, -2\,\mathrm{Exp}(\Re(\widetilde{a_1})) - 2\right) \tag{24}$$ eq:agmk1

we obtain an upper bound of

$$\mathrm{relerror}(\widetilde{a_1}), \mathrm{relerror}(\widetilde{b_1}) \leqslant \left(\sqrt{2}\right)^{k_1} \cdot 2^{1-p} \tag{25}$$ eq:agmerr1

on the loss of precision in the first iteration.

Notice that, independently of the rounding mode, $\widetilde{a_1}$ and $\widetilde{b_1}$ lie in the same complex quadrant of numbers having non-negative real part and an imaginary part of the same sign as that of $\widetilde{b_0}$ (or of positive imaginary part if $\widetilde{b_0}$ is real). During the remainder of the algorithm, we will not leave this quadrant and thus not see any more cancellation in the arithmetic mean.

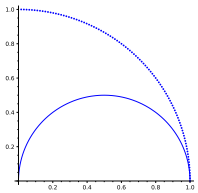


Figure 1: Extremal (from above) values of a_1 (solid curve) and b_1 (dotted curve) after the first iteration of the AGM for $\Im(b_0) \geqslant 0$. The values are bounded from below by the x-axis.

The idea of the analysis. Let us first give the basic idea of the following, rather technical analysis. Assume a target precision of N bits, that is, a target relative error of about 2^{-N}. If a_0 and b_0 are of the same order of magnitude, the AGM iteration converges quadratically, that is, the number of correct digits doubles in each step, and we need about $\log_2 N$ iterations. Unfortunately, when a_0 and b_0 are of different orders of magnitude, we have slower convergence. To illustrate this, consider $a_0 = 1$ and $b_0 = 2^{-e}$; during the first iterations, $a_n \approx 2^{-n}$ and $b_n \approx 2^{-e/2^n}$. So we need to increase the number of iterations by roughly $\log_2$ of the absolute value of the exponent of b_0/a_0; the precise bound $B\left(N, \widetilde{b_0}, \widetilde{a_1}\right)$ on the number of iterations is given in (36). Moreover, when $\Re(b/a) < 0$, the situation of very different exponents in a and b may occur after one iteration through cancellation as explained above.

Unlike Newton iterations, AGM iterations are not auto-correcting: due to rounding errors, we lose a constant number of bits per iteration; so to reach the desired precision of N bits, we need to carry out all computations at a working precision of $p \in N + O\left(B(N, \widetilde{b_0}, \widetilde{a_1})\right)$. The following discussion provides explicit bounds for all these quantities.

Rounding error propagation. Let $a_n = \frac{a_{n-1}+b_{n-1}}{2}$, $c_n = a_{n-1}b_{n-1}$, $b_n = \sqrt{c_n}$. The computation of $\widetilde{a_1}$ and $\widetilde{b_1}$ and their error analysis have been given above. For $n \geqslant 2$, we compute the sequences

$$\begin{aligned}
\widetilde{a_n} &= \circ\left(\frac{\widetilde{a_{n-1}} + \widetilde{b_{n-1}}}{2}\right), \\
\widetilde{c_n} &= \circ\left(\widetilde{a_{n-1}}\widetilde{b_{n-1}}\right), \\
\widetilde{b_n} &= \circ\left(\sqrt{\widetilde{c_n}}\right).
\end{aligned}$$

Then one sees by induction that $\widetilde{a_n}$ and $\widetilde{b_n}$ lie in the same quadrant as $\widetilde{a_1}$ and $\widetilde{b_1}$ (or, for that matter, a_1 and b_1).

Let $\alpha_n = \mathrm{relerror}(\widetilde{a_n})$, $\gamma_n = \mathrm{relerror}(\widetilde{c_n})$ and $\beta_n = \mathrm{relerror}(\widetilde{b_n})$.
By (7) and Proposition 11,

$$\alpha_n \leqslant \sqrt{2}\,\max(\alpha_{n-1}, \beta_{n-1})\left(1 + 2^{1-p}\right) + 2^{1-p}. \tag{26}$$ eq:agmalpha

By (10) and Proposition 11,

$$\gamma_n \leqslant (\alpha_{n-1} + \beta_{n-1} + \alpha_{n-1}\beta_{n-1})\left(1 + 2^{1-p}\right) + 2^{1-p}. \tag{27}$$ eq:agmgamma

By (18) and Proposition 11,

$$\beta_n \leqslant \frac{1}{\sqrt{3}}\gamma_n(1 + 2^{1-p}) + 2^{1-p} \text{ if } \gamma_n \leqslant \frac{1}{4}. \tag{28}$$ eq:agmbeta

Let $r_n = \left((\sqrt{2})^{n+1} - 1\right) r_1$ for some integer r_1 such that $\alpha_1, \beta_1 \leqslant r_1 2^{1-p}$. We may use $r_1 = (\sqrt{2})^{k_1}$ with k_1 as in (24), so that $r_n \leqslant (\sqrt{2})^{n+k_1+1}$. We now show by induction

Algorithm for AGM

that $\alpha_n, \beta_n \leqslant r_n 2^{1-p}$ if the number n of iterations is not too large compared to the working precision p. For $n = 1$, this follows from (25) and the definition of r_1.

As a preparation for the induction step and taking the shape of (26) to (28) into account, we carry out the following computation for $n \geqslant 2$, $p \geqslant 2$, $\frac{289(\sqrt{2}+1)}{512} \leqslant \gamma \leqslant 2$ (we will only need $\gamma \in \{\sqrt{2}, 2\}$ later) and $0 \leqslant \delta \leqslant 1$ (we will only need $\delta \in \{0, 1\}$):

$$\begin{aligned} R(n,\gamma,\delta) &:= \left(\gamma r_{n-1} + \delta r_{n-1}^2\, 2^{1-p}\right)\left(1 + 2^{1-p}\right) + 1 \\ &= \left(\gamma + \delta r_{n-1}\, 2^{1-p}\right) r_{n-1} + 1 + 2^{1-p}\left(\gamma r_{n-1} + \delta r_{n-1}^2\, 2^{1-p}\right) \\ &\leqslant \left(\gamma + \delta(\sqrt{2})^{n+k_1+2-2p}\right) r_{n-1} + 1 + 2^{1-p}\left((\sqrt{2})^{n+k_1+2} + 2^{n+k_1+1-p}\right). \end{aligned}$$

So assuming $p \geqslant \frac{n+k_1+10}{2}$ we have

$$\begin{aligned} R(n,\gamma,\delta) &\leqslant \left(\gamma + \delta(\sqrt{2})^{-8}\right) r_{n-1} + 289/256 \\ &\leqslant \frac{\left(\gamma + \delta(\sqrt{2})^{-8}\right)}{\sqrt{2}} r_n + 289/256 - \frac{\left(\gamma + \delta(\sqrt{2})^{-8}\right)}{\sqrt{2}}\left(r_n - \sqrt{2}\, r_{n-1}\right) \\ &\leqslant \frac{\left(\gamma + \delta(\sqrt{2})^{-8}\right)}{\sqrt{2}} r_n + 289/256 - \frac{\left(\gamma + \delta(\sqrt{2})^{-8}\right)}{\sqrt{2}}\left(\sqrt{2} - 1\right)(\sqrt{2})^3 \\ &\quad \text{since } r_1 \geqslant (\sqrt{2})^3 \\ &\leqslant \left(\frac{\gamma}{\sqrt{2}} + \frac{\delta}{16\sqrt{2}}\right) r_n \text{ since } \delta \geqslant 0 \text{ and } \gamma \geqslant \frac{289(\sqrt{2}+1)}{512}. \end{aligned}$$

Using the induction hypothesis for $n \geqslant 2$ on (26) and (27) yields

$$\begin{aligned} \alpha_n &\leqslant R(n, \sqrt{2}, 0)\, 2^{1-p} \leqslant r_n 2^{1-p}, \\ \gamma_n &\leqslant R(n, 2, 1)\, 2^{1-p} \leqslant \sqrt{2} \cdot \frac{33}{32} \cdot r_{n-1}\, 2^{1-p} \\ &\leqslant \sqrt{6}\, r_{n-1}\, 2^{1-p} \\ &\leqslant 4 \cdot (\sqrt{2})^{n+k_1} \cdot (\sqrt{2})^{-(n_1+k_1+8)} = 1/4. \end{aligned}$$

So (28) is valid, and substituting γ_n by its upper bound $\sqrt{6}\, r_{n-1}\, 2^{1-p}$ yields

$$\beta_n \leqslant R(n, \sqrt{2}, 0)\, 2^{1-p} \leqslant r_n 2^{1-p}.$$

To summarise, we compute in iteration n approximations $\widetilde{a_n}$ and $\widetilde{b_n}$ to a_n and b_n with a relative error bounded above by

$$2^{\frac{n+k_1+3}{2}-p} \text{ assuming that the working precision satisfies } p \geqslant \frac{n+k_1}{2} + 5. \qquad (29)$$ eq:agmprec

Mathematical error. We also need to estimate the error made by carrying out only a finite number of iterations. Let z satisfy $\Re(z) \geqslant 0$ and $z \neq 0, 1$, and consider the optimal

AGM sequences (a'_n) and (b'_n) computed (with infinite precision) from $a'_0 = 1$ and $b'_0 = z$. Let $N' \in \mathbb{N}$ and

$$n \geqslant B'(N', z) := \max\left(1, \lceil \log_2 |\log_2 |z|| \rceil\right) + \lceil \log_2(N'+2) \rceil + 2$$

(where $\log_2 0$ is to be understood as $-\infty$). Then by [6, Prop. 3.3, p. 88],

$$a'_n = (1 + \vartheta_2)\,\mathrm{AGM}(1, z) \text{ with } |\vartheta_2| \leqslant 2^{-N'}.$$

(Notice that here, the relative error as defined in [6, Def. 1.2, p. 20] is taken with the roles of the correct and the approximated values reversed compared to our definition.)

In our context, we may have $\Re(b_0) < 0$, but after one iteration, $\Re(b_1/a_1) \geqslant 0$. So we consider $z = b'_0 = b_1/a_1$, so that by homogeneity, $a_{n+1} = a_1 a'_n$, $b_{n+1} = a_1 b'_n$ and $\mathrm{AGM}(1, b_0) = \mathrm{AGM}(a_1, b_1) = a_1\,\mathrm{AGM}(1, z)$. Thus we need bounds on

$$|z| = \left|\frac{b_1}{a_1}\right| = \frac{|\sqrt{b_0}|}{|a_1|}.$$

Since we have to decide on the number of iterations from approximations to b_0 and a_1 instead of the correct values, we may wish to replace b_0 by $\widetilde{b_0}$ and a_1 by $\widetilde{a_1}$. (In fact the arguments will be quite delicate and switch between the different quantities.) The relative errors derived above on $\widetilde{b_0}$ of 2^{1-p} and on $\widetilde{a_1}$ of $(\sqrt{2})^{k_1}\, 2^{1-p}$ with $p \geqslant k_1/2+5 \geqslant 5$ show that

$$\left(1 - 2^{-4}\right)\left|\widetilde{b_0}\right| \leqslant |b_0| \leqslant \left(1 + 2^{-4}\right)\left|\widetilde{b_0}\right| \text{ and } \left(1 - 2^{-4}\right)|\widetilde{a_1}| \leqslant |a_1| \leqslant \left(1 + 2^{-4}\right)|\widetilde{a_1}|.$$

Since $\widetilde{b_0} = \circ(b_0)$ with both parts rounded towards zero, we even have $\left|\widetilde{b_0}\right| \leqslant |b_0|$. Altogether we have the (much coarser) following bounds:

$$\left|\widetilde{b_0}\right| \leqslant |b_0| \leqslant \sqrt{2}\left|\widetilde{b_0}\right| \text{ and } \frac{1}{\sqrt{2}}|\widetilde{a_1}| \leqslant |a_1| \leqslant \sqrt{2}\,|\widetilde{a_1}|. \qquad (30)$$ eq:agmabsb0a1

The quantity of interest is a (double) logarithm, so it is helpful to consider the (easily available) exponents of the numbers. Since $\widetilde{b_0} = \circ(b_0)$ with both parts rounded towards zero we have

$$\mathrm{Exp}(\Re(\widetilde{b_0})) = \mathrm{Exp}(\Re(b_0)) \text{ and } \mathrm{Exp}(\Im(\widetilde{b_0})) = \mathrm{Exp}(\Im(b_0)). \qquad (31)$$ eq:agmexpb0

Recall that $\widetilde{a_1} = \circ\left(\frac{1+\widetilde{b_0}}{2}\right)$ so that $\Im(\widetilde{a_1}) = \Im(\widetilde{b_0})/2$ does not require any additional rounding; together with (31) this yields

$$\mathrm{Exp}(\Im(\widetilde{a_1})) = \mathrm{Exp}(\Im(\widetilde{b_0})) - 1 = \mathrm{Exp}(\Im(b_0)) - 1 = \mathrm{Exp}(\Im(a_1)). \qquad (32)$$ eq:agmexpima1

However $\Re(\widetilde{a_1}) = \circ(1 + \circ(\Re(b_0)))/2$ is computed with two roundings, so it takes a bit more work to relate it to $\Re(a_1) = (1 + \Re(b_0))/2$, which we postpone to the following case distinction since we do not need a completely general result.

Algorithm for AGM

The above bound B' for the required number of iterations n grows with $\left|\log_2|z|\right|$, so problematic situations may occur when $|z|$ is either very large or very small; the former may happen when $|b_0|$ is large and/or $|a_1|$ is small, the latter when $|b_0|$ is small and/or $|a_1|$ is large. Remember that in our setting we have the bounds

$$|b_0| \leqslant 1, \text{ so that } |a_1| = \left|\frac{b_0+1}{2}\right| \leqslant 1.$$

thus interesting situations can occur only when either $|b_0|$ is small or $|a_1|$ is small, that is, b_0 is close to -1; and both cannot happen simultaneously. This leads to the following case distinction:

(a) Assume $\operatorname{Exp}(\Re(\widetilde{b_0})), \operatorname{Exp}(\Im(\widetilde{b_0})) \leqslant -1$.

By (31) this is equivalent to $\operatorname{Exp}(\Re(b_0)), \operatorname{Exp}(\Im(b_0)) \leqslant -1$, whence $\Re(b_0), \Im(b_0) < 1/2$ and

$$|b_0| \leqslant \frac{\sqrt{2}}{2} < 1.$$

This implies

$$1 - \frac{\sqrt{2}}{2} \leqslant 2|a_1| \leqslant 1 + \frac{\sqrt{2}}{2} \text{ and } 1/8 \leqslant |a_1| \leqslant 1$$

and we know $|\sqrt{b_0}| \geqslant \left|\sqrt{\widetilde{b_0}}\right|$ by (30). Collecting these upper and lower bounds, we obtain for $z = \sqrt{b_0}/a_1$ that

$$\sqrt{\left|\widetilde{b_0}\right|} \leqslant |z| \leqslant 8 \text{ and } \left|\log_2|z|\right| \leqslant \max\left(3, -\frac{1}{2}\log_2\left|\widetilde{b_0}\right|\right).$$

Using the estimate

$$\left|\widetilde{b_0}\right| \geqslant \max\left(|\Re(\widetilde{b_0})|, |\Im(\widetilde{b_0})|\right) \geqslant 2^{\max\left(\operatorname{Exp}(\Re(\widetilde{b_0})), \operatorname{Exp}(\Im(\widetilde{b_0}))\right)-1}$$

we finally obtain

$$\left|\log_2|z|\right| \leqslant \max\left(3, -\frac{1}{2}\max\left(\operatorname{Exp}(\Re(\widetilde{b_0})), \operatorname{Exp}(\Im(\widetilde{b_0}))\right) + \frac{1}{2}\right). \quad (33) \quad \texttt{eq:agmlb}$$

(b) Assume $\operatorname{Exp}(\Im(\widetilde{b_0})) \leqslant -1$ and $\operatorname{Exp}(\Re(\widetilde{a_1})) \leqslant -2$.

By (32) the first condition implies $\operatorname{Exp}(\Im(a_1)) \leqslant -2$ or $|\Im(a_1)| < 1/4$. The second condition together with the first inequality of Proposition 3 implies that $\operatorname{Exp}\left((1+\Re(\widetilde{b_0}))/2\right) \leqslant -2$, or $\left|1+\Re(\widetilde{b_0})\right| < 1/2$. So in fact $-1 < \Re(\widetilde{b_0}) < -1/2$, and then $-1 < \Re(b_0) < -1/2$ as well. This in turn implies $0 < \Re(a_1) < 1/4$.

Putting these together, we obtain

$$1/2 \leqslant |\Re(b_0)| \leqslant |b_0| \text{ or } 1/\sqrt{2} \leqslant |\sqrt{b_0}|, \text{ and } |a_1| \leqslant \sqrt{2}/4,$$

and we already know

$$|b_0| \leqslant 1 \text{ or } |\sqrt{b_0}| \leqslant 1$$

from our general setting and

$$|a_1| \geqslant |\widetilde{a_1}|/\sqrt{2}$$

by (30), whence

$$2 \leqslant |z| \leqslant \frac{\sqrt{2}}{|\widetilde{a_1}|} \text{and } \left|\log_2|z|\right| \leqslant -\log_2|\widetilde{a_1}| + \frac{1}{2}.$$

Using the estimate

$$|\widetilde{a_1}| \geqslant \max\left(|\Re(\widetilde{a_1})|, |\Im(\widetilde{a_1})|\right) \geqslant 2^{\max(\operatorname{Exp}(\Re(\widetilde{a_1})), \operatorname{Exp}(\Im(\widetilde{a_1})))-1}$$

and (32) we finally obtain

$$\left|\log_2|z|\right| \leqslant -\max\left(\operatorname{Exp}(\Re(\widetilde{a_1})), \operatorname{Exp}(\Im(\widetilde{b_0})) - 1\right) + \frac{3}{2}. \quad (34) \quad \texttt{eq:agmla}$$

(c) In the remaining case, since we are not in (a), at least one of $\operatorname{Exp}\left(\Re(\widetilde{b_0})\right)$ and $\operatorname{Exp}\left(\Im(\widetilde{b_0})\right)$, or equivalently by (31) at least one of $\operatorname{Exp}(\Re(b_0))$ and $\operatorname{Exp}(\Im(b_0))$ is 0 or larger, so that

$$\frac{1}{2} \leqslant \max(|\Re(b_0)|, |\Im(b_0)|) \leqslant |b_0| \leqslant 1 \text{ and } \frac{1}{\sqrt{2}} \leqslant \sqrt{|b_0|} \leqslant 1.$$

If $\operatorname{Exp}\left(\Im(\widetilde{b_0})\right) \geqslant 0$, then $\operatorname{Exp}(\Im(a_1)) \geqslant -1$ by (32) and

$$|a_1| \geqslant |\Im(a_1)| \geqslant 1/4.$$

Otherwise since we are not in (b), $\operatorname{Exp}(\Re(\widetilde{a_1})) \geqslant -1$ or $|\Re(\widetilde{a_1})| \geqslant 1/4$; as it is known to be positive in fact $\Re(\widetilde{a_1}) \geqslant 1/4$. In precision at least 5, the value $1/4 \cdot 31/32$ is representable and smaller than $1/4$ so that the value rounded to $\Re(\widetilde{a_1})$ satisfies $(1+\Re(\widetilde{b_0}))/2 > 31/128$ and $\Re(\widetilde{b_0}) > -33/64$. In precision at least 5, the value $-34/64$ is representable and smaller than $-33/64$ so that the unrounded value satisfies $\Re(b_0) > -17/32$, which implies $\Re(a_1) > 15/64 > 1/8$ and

$$|a_1| \geqslant |\Re(a_1)| \geqslant 1/8.$$

We also know $|a_1| \leqslant 1$ in our general setting. Altogether

$$\frac{1}{\sqrt{2}} \leqslant |z| \leqslant 8 \text{ and } \left|\log_2|z|\right| \leqslant 3. \quad (35) \quad \texttt{eq:agml}$$

Algorithm for AGM

Letting $L(\widetilde{b_0}, \widetilde{a_1})$ denote the above bound on $|\log_2|z||$ depending on the exponents occurring in $\widetilde{b_0}$ and $\widetilde{a_1}$, and counting the additional first iteration to compute $\widetilde{a_1}$ and $\widetilde{b_1}$, not present in the above analysis, we fix a number of iterations n such that $n \geqslant B(N, \widetilde{b_0}, \widetilde{a_1})$ with

$$B(N, \widetilde{b_0}, \widetilde{a_1}) = \max\left(1, \left\lceil \log_2 L(\widetilde{b_0}, \widetilde{a_1}) \right\rceil\right) + \lceil \log_2(N+4) \rceil + 3. \tag{36}$$

eq:agmbound

Then

$$a_n = (1+\vartheta_2)\,\mathrm{AGM}(1, b_0) \text{ with } |\vartheta_2| \leqslant 2^{-(N+2)}.$$

Total error and working precision. Combining with (29), we obtain for a sufficiently large precision p and sufficiently many iterations n that $\mathrm{AGM}(1, b_0) = \frac{1+\vartheta_1}{1+\vartheta_2}\widetilde{a_n} = (1+\vartheta)\widetilde{a_n}$ with $|\vartheta_1| \leqslant 2^{\frac{n+k_1+3}{2}-p}$ and $|\vartheta_2| \leqslant 2^{-(N+2)}$, so that

$$|\vartheta| \leqslant \frac{|\vartheta_1| + |\vartheta_2|}{|1 - |\vartheta_2||} \leqslant \frac{4}{3}\left(2^{\frac{n+k_1+3}{2}-p} + 2^{-(N+2)}\right)$$

for $N \geqslant 2$. So after $n = B(N, \widetilde{b_0}, \widetilde{a_1})$ steps of the AGM iteration at a working precision of $p \geqslant N + \frac{n+k_1+7}{2} \geqslant \frac{n+k_1+10}{2}$, we obtain $\widetilde{a_n}$ which approximates $\mathrm{AGM}(1, b_0)$ with a relative error bounded by $\frac{2}{3} \cdot 2^{-N}$.

Finally, we let $z = \mathrm{AGM}(a, b) = a\,\mathrm{AGM}(1, b_0)$ and $\widetilde{z} = \circ(a\widetilde{a_n})$, where a is known exactly. By (10) and Proposition 11, using that $N \geqslant 2$ and $k_1 \geqslant 3$ imply $n = B(N, \widetilde{b_0}, \widetilde{a_1}) \geqslant 7$ and $p \geqslant N + 8 \geqslant 10$, this leads to a relative error bounded by 2^{-N}.

Summary. In our analysis, the working precision p depends on

$$k_1 = \max(3, -2\,\mathrm{Exp}(\Re(\widetilde{a_1})) - 2)$$

of (24), which in turn depends not only on the input data, but also on the working precision of the first AGM iteration. It is not enough to carry out this computation at arbitrarily low precision: Since $\widetilde{b_0}$ is computed as a/b or b/a depending on the respective sizes of the numbers and it is rounded, the computation of $\Re(\widetilde{a_1})$ requires double rounding, which may even lead to a wrong exponent if the precision is too low. Precisely, the exponent of $\Re(\widetilde{a_1})$ may only be wrong if $\Re(\widetilde{b_0})$ is so close to -1 that it is rounded towards zero to -1 plus 1 ulp, that is, $\Re(\widetilde{b_0}) = -(1 - 2^{-p})$. Then $\Re(\widetilde{a_1}) = 2^{-p-1}$ with $\mathrm{Exp}(\Re(\widetilde{a_1})) = -p$ has lost all information on $\Re(b_0)$ and reflects only the rounding error. In all other cases a significant digit is retained and the exponent is correct. So k_1 may be computed by increasing the precision for this precomputation until $\mathrm{Exp}(\Re(\widetilde{a_1})) > -p$.

Then we fix a desired accuracy N (around the target precision plus a safety margin), compute $L(\widetilde{b_0}, \widetilde{a_1})$ by (33), (34) or (35) and the number of iterations $n = B(N, \widetilde{b_0}, \widetilde{a_1})$ by (36) (more often than not, this will result in $n = \lceil \log_2 N \rceil + 5$). Then the working precision is given by

$$p = N + \left\lceil \frac{n + k_1 + 7}{2} \right\rceil.$$

Using Propositions 9 and 7, the complex relative error of 2^{-N} may be translated into an error expressed in ulp. With $\widetilde{z} = \widetilde{x} + i\widetilde{y}$ the computed approximation of $z = \mathrm{AGM}(a, b)$, let $k_R = \max(\mathrm{Exp}(\widetilde{y}) - \mathrm{Exp}(\widetilde{x}) + 1, 0) + 1$, and $k_I = \max(\mathrm{Exp}(\widetilde{x}) - \mathrm{Exp}(\widetilde{y}) + 1, 0) + 1$. Then we have $\mathrm{error}(\widetilde{x}) \leqslant 2^{k_R+p-N}\mathrm{ulp}(\widetilde{x})$ and $\mathrm{error}(\widetilde{y}) \leqslant 2^{k_I+p-N}\mathrm{ulp}(\widetilde{y})$.

In practice, one should take this additional loss into account. If rounding fails after the first computation, nevertheless the values of k_R and k_I will most likely not change with a larger precision. So one should let $k' = \max(k_R, k_I)$, replace N by $N + k'$ and adapt the number of iterations and the working precision accordingly.

The number of iterations is also slightly pessimistic in practice, in particular the additional constant in (36). So the computations may be stopped earlier if the numbers occurring in the AGM iterations do not change any more, since then additional iterations will fix the result.

4 Complex ball arithmetic

We propose a simple implementation of complex balls, which keep track of rounding errors over several operations. The originality of our implementation is that it uses complex relative errors as in §1.1.4. A complex ball of type `mpcb_t` is defined by a non-zero centre c of type `mpc_t` and a relative radius r of type `mpcr_t`, and it represents all complex numbers $z = c(1+\vartheta)$ with $|\vartheta| \leqslant r$, or equivalently the closed circle with centre c and radius $r|c|$. In the following, we use the notation (c, r) for this complex ball.

The radius type represents the non-negative real half-axis from 0 to ∞, including this special infinite number. It is implemented internally as a non-negative floating point value with a signed 64 bit integer mantissa m, normalised to 31 bits, so that two mantissas may be multiplied without rounding. The sign is used to encode $r = \infty$, a zero mantissa encodes 0, otherwise a mantissa is always positive. The exponent is encoded as a 64 bit integer e such that (for finite radii) $r = m \cdot 2^e$ with $0 \leqslant m < 2^{31}$. In most applications a radius $r \geqslant 1$ will be meaningless, so that in practice we will almost always have $e \leqslant -31$, or $r = \infty$.

Mathematical functions are then understood to work on sets and to "round up": They return a complex ball containing the set obtained by applying the function to every combination of arguments from the input balls. Reasonable efforts are made to return small balls, but there is no guarantee that the returned ball is minimal.

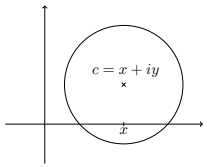
In any case, the centre of the resulting ball is obtained by applying the corresponding MPC function on the centres of the input balls with rounding to nearest. The analyses of §3 then provide upper bounds on the radius.

Compared to a representation decomposed along the real and imaginary parts, with separate relative or absolute errors, which leads to rectangles instead of circles, our representation simplifies multiplicative operations and makes additive operations more complicated. Determining branch cuts, which often depend on the decomposition of a complex number into real and imaginary parts, are probably also made more complicated. The biggest drawback is that intervals centred in 0 may not be represented at all. More generally, intervals containing 0 are also not meaningful: They correspond to balls

Algorithm for AGM using ball arithmetic

with $r \geqslant 1$, which means that even the most significant digit of the centre is uncertain.

4.1 Crossing the axes



To correctly evaluate functions at branch cuts, it may be useful to examine how complex balls are positioned with respect to the axes. As already mentioned above, a ball (c, r) contains the origin if and only if $r \geqslant 1$.

For $c = x + iy$, it crosses (or touches) the real axis if and only if it contains the point x, which means that

$$|x - c| \leqslant r|c| \Leftrightarrow |y| \leqslant r|c| \Leftrightarrow y^2 \leqslant r^2\left(x^2 + y^2\right) \Leftrightarrow \left(1 - r^2\right) y^2 \leqslant r^2 x^2.$$

When does a complex ball touch the real axis? This means that $(1-r^2)y^2 = r^2x^2$, which happens obviously when $r = 0$ and $y = 0$, so that the ball is a point located on the real axis; or when $r = 1$ and $x = 0$, so that the ball is centred on the imaginary axis and touches the origin. All other cases are actually impossible with machine-representable numbers. Assuming otherwise, recall that x, y and r are rational numbers with denomitor a power of 2. If r is an integer, then it is either 1, which we already covered; or larger, and then the origin is contained in the interior of the ball, which must cross the real axis. So let $r = a/2^n$ with a an odd integer and $n \geqslant 1$. Then $4^n - a^2 = 4^n(1 - r^2) = (2^{-n}rxy^{-1})^2$ is the square of an odd integer. But all odd squares are 1 modulo 4, whereas the left-hand side is 3 modulo 4, a contradiction.

Symmetrically, the complex ball $(x + iy, r)$ crosses (or touches) the imaginary axis if and only if

$$\left(1 - r^2\right) x^2 \leqslant r^2 y^2.$$

It touches the imaginary axis if and only if $r = 0$ and $x = 0$, so that the ball is a point on the imaginary axis; or $r = 1$ and $y = 0$, so that the ball is centred on the real axis and touches the origin.

The ball has a common point with the negative real axis (including the origin) if and only if either $x \leqslant 0$ and the ball has a common point with the real axis (since then x is such a common point); or $x > 0$ and the ball contains the origin.

4.2 `mpc_agm`

Implementing the AGM (see §3.10) for complex balls would require to check whether the input crosses the negative real axis, where we have placed the branch cut (which is

inherited from the branch cut of the complex square root). However if the input is a complex number, which can be considered to be exact, then an implementation using complex balls can obtain a correctly rounded result with a greatly simplified analysis compared to §3.10.

In a first step, assuming that $|a| \geqslant |b|$, we compute $b_0 = b/a$ as a complex ball centred around $\widetilde{b_0}$.

If $\Im(\widetilde{b_0}) = 0$, then the angle between a and b is 0 or π. Regardless of the rounding direction, $\Re(\widetilde{b_0})$ also has the same sign as $\Re(b_0)$. If $\Re(\widetilde{b_0}) > 0$, then the angle is 0 and the computation can be outsourced to a real AGM as described in §3.10. If $\Re(\widetilde{b_0}) < 0$, then the angle is π, and the complex ball containing b_0 crosses the negative real axis; the first step of the AGM with an implementation of `mpcb_sqrt` that respects the branch cut would end up with a ball containing $\pm\sqrt{|b_0|} \cdot i$ and thus the origin, which is completely useless. We may, however, in this case use an implementation of the square root placing the branch cut differently and returning a ball centred at some approximation of $+\sqrt{|b_0|} \cdot i$, and then continue with the AGM iterations.

After n iterations starting from 1 and a ball around b_0, we end up with balls $(\widetilde{a_n}, r_a)$ and $(\widetilde{b_n}, r_b)$ such that the exact values satisfy $a_n = (1 + \vartheta_a)\widetilde{a_n}$ with $|\vartheta_a| \leqslant r_a$ and $b_n = (1 + \vartheta_b)\widetilde{b_n}$ with $|\vartheta_b| \leqslant r_b$.

By [6, p.87] we have $|\operatorname{AGM}(1, b_0) - a_n| \leqslant |a_n - b_n|$. Plugging the expressions for a_n and b_n into this inequality yields

$$|\operatorname{AGM}(1, b_0) - (1 + \vartheta_a)\widetilde{a_n}| \leqslant |\widetilde{a_n} - \widetilde{b_n}| + |\vartheta_a\widetilde{a_n}| + |\vartheta_b\widetilde{b_n}|$$

The lower triangle inequality gives

$$|\operatorname{AGM}(1, b_0) - (1 + \vartheta_a)\widetilde{a_n}| \geqslant |\operatorname{AGM}(1, b_0) - \widetilde{a_n}| - |\vartheta_a| \cdot |\widetilde{a_n}|,$$

and putting these inequalities together we obtain

$$|\operatorname{AGM}(1, b_0) - \widetilde{a_n}| \leqslant \left(\left|\frac{\widetilde{a_n} - \widetilde{b_n}}{\widetilde{a_n}}\right| + 2|\vartheta_a|\right)|\widetilde{a_n}| + |\vartheta_b| \cdot |\widetilde{b_n}|.$$

Write $\widetilde{b_n} = (1 + \vartheta_{a,b})\widetilde{a_n}$ and $r_{a,b} = |\vartheta_{a,b}|$; then we obtain

$$|\operatorname{AGM}(1, b_0) - \widetilde{a_n}| \leqslant \left(r_{a,b} + 2r_a + r_b(1 + r_{a,b})\right)|\widetilde{a_n}| \leqslant \left(2(r_{a,b} + r_a) + r_b\right)|\widetilde{a_n}| \text{ if } r_b \leqslant 1.$$

Otherwise said, $\operatorname{AGM}(1, b_0)$ is contained in the ball $\left(\widetilde{a_n}, r_{a,b}+2(r_a+r_b)\right)$, and multiplying this with the ball $(a, 0)$ we obtain a ball containing $\operatorname{AGM}(a, b)$. If this can be rounded to a unique MPC number with the desired rounding mode, then we have correctly computed `mpc_agm`; otherwise we need to repeat the computations at a higher precision, and the exponent of the radius gives an indication on the necessary precision increase.

In practice convergence of the sequences of $\widetilde{a_n}$ and $\widetilde{b_n}$ is often such that $\widetilde{a_n} = \widetilde{b_n}$ and thus $r_{a,b} = 0$. Otherwise the two generally differ by only 1 ulp, and a very coarse estimate of $r_{a,b}$ as a power of 2

$$r_{a,b} \leqslant 2^{\max\left(\operatorname{Exp}(\Re(\widetilde{a_n}-\widetilde{b_n})),\operatorname{Exp}(\Im(\widetilde{a_n}-\widetilde{b_n}))\right)+1-(\min(\operatorname{Exp}(\Re(\widetilde{a_n})),\operatorname{Exp}(\Im(\widetilde{a_n})))-1)}$$

is enough, where $\operatorname{Exp}(0)$ is considered as $-\infty$.

Complex balls

Centre and radius as relative error

$$\begin{aligned}(c, r) &= \{z \in \mathbb{C} : |z - c| \leqslant r|c|\} \\ &= \{z = c(1 + \vartheta) : |\vartheta| \leqslant r\}\end{aligned}$$

Complex balls

Centre and radius as relative error

$$\begin{aligned}(c, r) &= \{z \in \mathbb{C} : |z - c| \leqslant r|c|\} \\ &= \{z = c(1 + \vartheta) : |\vartheta| \leqslant r\}\end{aligned}$$

- Scaling is easy

$$s(c, r) \subseteq (sc, r) \text{ for } s \in \mathbb{R}$$

- Multiplication is easy

$$(c_1, r_1) + (c_2, r_2) \subseteq (c_1 c_2, r_1 + r_2 + r_1 r_2)$$

- Square root is easy

$$\sqrt{(c, r)} \subseteq (\sqrt{c}, r/2)$$

- Addition is more difficult

$$(c_1, r_1) + (c_2, r_2) \subseteq (c_1 + c_2, (|c_1|r_1 + |c_2|r_2)/|c_1 + c_2|)$$

0 is not representable.

Complex balls in C

```
typedef struct {
   mpc_t  c;
   mpcr_t r;
} mpcb_t;
```

Complex balls in C

```
typedef struct {
   mpc_t  c;
   mpcr_t r;
} mpcb_t;
```

```
typedef struct {
   int64_t mant;
   int64_t exp;
} mpcr_t;
```

Complex balls in C

```
typedef struct {
   mpc_t  c;
   mpcr_t r;
} mpcb_t;
```

```
typedef struct {
   int64_t mant;
   int64_t exp;
} mpcr_t;
```

- radius 0: $\texttt{mant} = 0$
- radius ∞: $\texttt{mant} < 0$
- 31 bit normalised mantissa: $2^{30} \leqslant \texttt{mant} < 2^{31}$

Functions on radii

Surprisingly many and quirky functions...
Results are normalised and rounded up (unless exception).

- Predicates
  - `mpcr_inf_p`
  - `mpcr_zero_p`
  - `mpcr_lt_half_p`
  - `mpcr_cmp`
- Setters
  - `mpcr_set_inf`
  - `mpcr_set_zero`
  - `mpcr_set_one`
  - `mpcr_set`
  - `mpcr_set_ui64_2si64`
  - `mpcr_max`
- Output
  - `mpcr_out_str`

Functions on radii

- Arithmetic
  - `mpcr_add`, `mpcr_mul`, ...
  - `mpcr_sub_rnd`
    Takes `MPFR_RNDU`, `MPFR_RNDD`.
  - `mpcr_c_abs_rnd`
    Used for error of addition

$$(c_1, r_1) + (c_2, r_2) \subseteq (c_1 + c_2, (|c_1|r_1 + |c_2|r_2)/|c_1 + c_2|)$$

Functions on radii

- Arithmetic
  - `mpcr_add`, `mpcr_mul`, ...
  - `mpcr_sub_rnd`
    Takes `MPFR_RNDU`, `MPFR_RNDD`.
  - `mpcr_c_abs_rnd`
    Used for error of addition

$$(c_1, r_1) + (c_2, r_2) \subseteq (c_1 + c_2, (|c_1|r_1 + |c_2|r_2)/|c_1 + c_2|)$$

  - `mpcr_add_rounding_error (r, p, rnd)`
    Accounts for shift of centre by $1/2$ ulp or 1 ulp depending on `rnd`.
    Adds $(1 + r)2^{-p}$ or twice this to r.
    Called once at the end of each operation.

Principles of complex balls

- $(c_1, r_1) \circ (c_2, r_2) \subseteq (c_1 \circ c_2, r)$
- *One* precision for real and imaginary part
- Initialised *without* precision: `mpcb_init (z)`, `mpcb_clear (z)`
- Precisions are tracked automatically.
  - $z_1 \circ z_2$ gets minimum precision of z_1 and z_2.
  - Precision is not decreased when radius increases.

Setting complex balls

- `mpcb_set_inf (z)`
- `mpcb_set (z, z1)`

Challenges: Exact input and inputs with errors.

Setting complex balls

- `mpcb_set_inf (z)`
- `mpcb_set (z, z1)`

Challenges: Exact input and inputs with errors.

- `mpcb_set_ui_ui (z, re, im, prec)`
  Uses maximum of `prec` and `sizeof (ulong)`.
- `mpcb_set_c (z, c, prec, err_re, err_im)`
  Assumes c has `err_re` and `err_im` half-ulp errors.
  - If `prec` large and `err_re` and `err_im` $= 0$: exact, $r = 0$.
  - Otherwise, r encodes `err_re`, `err_im` and rounding of c.

Computations with complex balls

- `mpcb_neg (z, z1)`
- `mpcb_add (z, z1, z2)`
- `mpcb_mul (z, z1, z2)`
- `mpcb_sqr (z, z1, z2)`
- `mpcb_pow_ui (z, z1, e)`
- `mpcb_sqrt (z, z1)`
- `mpcb_div (z, z1, z2)`
- `mpcb_div_2ui (z, z1, e)`

Rounding complex balls

- `mpcb_round (c, z, rnd)`
  Rounds the centre of z to c (with its own precision).

Rounding complex balls

- `mpcb_round (c, z, rnd)`
  Rounds the centre of z to c (with its own precision).
- `mpcb_can_round (z, prec_re, prec_im, rnd)`
  `true`
  if rounding any complex (mathematical) number in z to
  a complex (floating point) number of precision `(prec_re, prec_im)`
  in direction `rnd`
  yields the same result and ternary return value.

Rounding complex balls

- `mpcb_round (c, z, rnd)`
  Rounds the centre of z to c (with its own precision).
- `mpcb_can_round (z, prec_re, prec_im, rnd)`
  `true`
  if rounding any complex (mathematical) number in z to
  a complex (floating point) number of precision `(prec_re, prec_im)`
  in direction `rnd`
  yields the same result and ternary return value.
- Beware of infinite loops for exact results.

“Normal” functions using balls

- `mpc_agm`
  Uses a priori error analysis.
  Tests compare with `mpc_mpcb_agm`.

“Normal” functions using balls

- `mpc_agm`
  Uses a priori error analysis.
  Tests compare with `mpc_mpcb_agm`.
- `mpc_eta_fund (rop, z, rnd)`

$$
\begin{aligned}
q^{1/24} &= \exp(2\pi i z/24) \\
q &= \left(q^{1/24}\right)^{24} \\
\eta &= q^{1/24}\left(1 - q - q^2 + q^5 + q^7 - q^{12} - q^{15} + \cdots\right)
\end{aligned}
$$

$q^{1/24}$ is computed as `mpc` with an a priori error analysis,
then handled with `mpcb_set_c`.

What is next?

- Question the design choices.
- Handle 0?
  - Exact 0 is easily encoded, but needs case distinctions.
  - Balls around 0 need absolute radius encoding
    and more case distinctions.
- Use balls internally for series (special values of special functions)?
- Use balls in existing functions instead of error analysis?
- Implement functions on balls?