CORRECTLY ROUNDED CUBIC ROOT EVALUATION IN DOUBLE PRECISION

ALEXEI SIBIDANOV AND PAUL ZIMMERMANN

The cubic root $x = \sqrt[3]{a}$ is a real root of an algebraic equation:

$$f(x) = x^3 - a = 0. \tag{1}$$

There is a closed form solution for Eq. (1) but it already requires the cubic root function so other methods have to be employed e.g. *Newton iteration*.

Let x_0 be an initial approximation of the cubic root then

$$h_0 = f(x_0)/a = (x_0^3 - a)/a = (x_0^3 - a)r_a \tag{2}$$

is the relative error of Eq .(1) with respect to a and $r_a = 1/a$ is the reciprocal of a. The next better approximation x_1 can be derived as

$$x_1 = x_0 - \frac{1}{3}x_0 h_0 \tag{3}$$

with about two times more significant figures than in x_0. This procedure should be repeated until it reaches required precision.

The generalization of the Newton iteration method to higher orders gives the following rule:

$$x_{i+1} = x_i \left(1 - \frac{1}{3}h_i + \frac{2}{9}h_i^2 - \frac{14}{81}h_i^3 + \frac{35}{243}h_i^4 - \frac{91}{729}h_i^5 + \frac{728}{6561}h_i^6 - \frac{1976}{19683}h_i^7 + \frac{5434}{59049}h_i^8 - \cdots\right), \tag{4}$$

where each additional term reduces the error of the next approximation x_{i+1} by h. The coefficients of the polynomial expression in Eq. (4) are given by the series expansion of

$$\frac{1}{\sqrt[3]{1+h}} = \sum_{j=0}^{\infty} c_j h^j. \tag{5}$$

Since a and x are represented as IEEE-754 double precision floating point numbers aka *binary64* we can reduce exactly the input argument a to the $[1, 8]$ range to get $x \in [1, 2]$ range and then scale it accordingly to get the final result. The binary scaling is a cheap and exact operation in the binary64 format and particularly for the cubic root without the danger of the overflow or underflow since the limited exponent range of the final result. The argument a can be further reduced to the $[1, 2]$ range but the result has to be scaled by $2^{n/3}$ before the final refinement to get the correctly rounded result since the values of $2^{n/3}$ with $n = 1, 2$ are inexact in the binary64 format.

Date: November 22, 2021.

For arguments in the range $[1, 2]$ Newton iterations for the cubic root always converges with the initial approximation $x_0 = 1.104$ in all orders in our tests. An example is shown in Fig. 1 for the high order given in Eq. (4). It shows that with a moderately precise initial approximation Newton iterations converge rapidly. For fast evaluation the initial approximation can be selected as a low order polynomial.

The error h_0 of the minimax approximations of the cubic root function by the second, third, fourth and fifth order polynomials is shown in Fig. 2. The error h_1 after the first step are shown in Fig. 3 for the second order Newton iteration, Fig. 3 for the third order, and Fig. 3 for the quartic order. These calculations are performed in the binary64 format so the limited precision of the format is immediately seen even after the first high order iteration. So itself the cubic root calculated by this method cannot be correctly rounded due to intermediate rounding errors. The final refinement step using a compensated algorithm is needed.

The final step has to be as simple as possible so it is the second order Newton iteration (Eq. (2) and (3)) where intermediate values are represented as an unevaluated sum of two binary64 numbers so the internal precision should be about 100 bits which largely exceeds the target precision of the result of 53 bits in binary64.

The precision of the result before the final step should not hit the binary64 precision limit it should be just good enough that after the refinement—which doubles the number of significant figures—an additional refinement has to be done only in very rare cases when the rounding test fails. Based on this consideration and performance tests we select the initial cubic polynomial approximation and the third order Newton iteration step, see the top-right plots in Fig. 2 and 4.

After the refinement with the compensated algorithm, the cubic root value is represented as an unevaluated sum $a + b$ of two binary64 numbers, where $e_a \geq e_b$ (i.e., the exponent of a is larger or equal to that of b). We then apply the Fast2Sum algorithm to compute $x_2^{\text{high}} \leftarrow \circ(a+b)$, $z \leftarrow \circ(x_2^{\text{high}} - a)$, $x_2^{\text{low}} \leftarrow \circ(b-z)$, where $\circ()$ denotes the current rounding mode.

Lemma 1. *Whatever the rounding mode, we have* $|x_2^{\text{low}}| < 2^{-52}$.

Proof. For rounding to nearest, this is a direct consequence of the Fast2Sum algorithm, since in that case we have $a + b = x_2^{\text{high}} + x_2^{\text{low}}$ exactly, and since x_2^{high} is the rounding to nearest of $a + b$, we have $x_2^{\text{low}} \leq \frac{1}{2}\text{ulp}(x_2^{\text{high}})$. For directed rounding, according to [1, Theorem 3.1], x_2^{low} is a faithful rounding of the error in the FP addition $x_2^{\text{high}} = \circ(a + b)$. Let $\varepsilon = (a+b) - x_2^{\text{high}}$ be that error. Since $1 \leq a+b \leq 2$, and x_2^{high} is a directed rounding of $a+b$, we have $|\varepsilon| < \text{ulp}(1) = 2^{-52}$, thus a faithful rounding of that error cannot exceed 2^{-52}. Now if a faithful rounding of ε is $\pm 2^{-52}$, this implies $|\varepsilon| > 2^{-52} + 2^{-105}$, since $2^{-52} + 2^{-105}$ is representable in binary64. This in turn implies $\text{ulp}(b) < 2^{-105}$, otherwise $a + b$ would be an integer multiple of 2^{-105}, which would contradict $2^{-52} + 2^{-105} < |\varepsilon| < 2^{-52}$. But since $|b| < 2^{53}\text{ulp}(b)$ this yields $|b| < 2^{-52}$. In the Fast2Sum algorithm, when $x_2^{\text{high}} = \circ(a + b)$ is rounded towards a, we get $z = 0$ and $x_2^{\text{low}} = b$, thus $|x_2^{\text{low}}| < 2^{-52}$. If $x_2^{\text{high}} = \circ(a + b)$ is rounded away from a, say upwards if $b > 0$, then $z = 2^{-52}$, and since $x_2^{\text{low}} = \circ(b - z)$

is rounded in the *same* direction, we get $x_2^{\text{low}} > -z$. The same reasoning when rounding downwards for $b < 0$ also gives $|x_2^{\text{low}}| < 2^{-52}$. □

According to Lemma 1, we thus get an approximation $x_2^{\text{high}} + x_2^{\text{low}}$ of the cubic root with $|x_2^{\text{low}}| < 2^{-52}$. The difference of this approximation with the exact cubic root value is shown in Fig. 6. The maximal found error is $-\texttt{0x1.fe62ec338p} - \texttt{77} \approx -1.32 \times 10^{-23}$ and it occurs near the upper bound of the range. Thus to perform the rounding test in the round-to-nearest mode we need to check that $||x_2^{\text{low}}| - 2^{-53}| > 2^{-76}$ which means that x_2^{high} is a correctly rounded cubic root value in binary64. In the directional modes we need to check both borders $|x_2^{\text{low}}| > 2^{-76}$ and $||x_2^{\text{low}}| - 2^{-52}| > 2^{-76}$ to be sure that x_2^{high} is correctly rounded. For safety the limit 2^{-76} is increased in 2 times to 2^{-75}. Considering this limit we can conclude that the probability to fail the test is about $2^{-75}/2^{-52} \sim 10^{-7}$. There is a special case of exact cubic roots which will be described later.

If the rounding test fails we perform an additional second order Newton iteration step starting from x_2^{high} which is known to be very close to the correctly rounded cubic root just might be 1 ulp off. The difference of x_3^{high} (again $x_3 = x_3^{\text{high}} + x_3^{\text{low}}$) with the exact cubic root value is shown in Fig. 7, 8, 9, 10 when FPU is operating in various rounding modes. As it is seen the maximal visible error is about 2^{-102} on the limited number of arguments.

Unfortunately even the last refinement is not enough for the worst cases to provide the correct rounded results, fortunately there are only a few such cases so we can test arguments and return already precomputed correctly rounded values.

In the round-to-nearest mode the exact cases, when both a and x are representable in the binary64 format exactly, always pass the first rounding test and round to correct values. Unfortunately one also finds that the inexact flag is risen despite the exact roots due to the intermediate rounding errors. In the directed rounding modes both rounding tests fail for exact cubic roots and x_3^{high} can be 1 ulp off of the correctly rounded value. Such cases have to be detected and the flag has to be restored to the state just before the function call.

There are 104032 distinct binary64 numbers x in the $[1, 2]$ range which might be exact solutions of Eq. (1) with the one with largest numerator being $208063/2^{17}$, where $208063 = \lfloor 2^{53/3} \rfloor$. Thus, for exact cubic roots, and rounding to nearest, at least 35 last bits of x_2^{high} have to be zero. For exact cubic roots with a directed rounding mode, the last 35 bits of x_3^{high} should be all 0 or 1 (note that the first rounding test will always fail in that case). The test of the last bits of x alone to detect the exact cases is not enough since there are cases when the cubic root of a has 35 zero bits but it is not an exact root. For example, when we have exact relation $x = \sqrt[3]{a}$ in binary64 then $\sqrt[3]{a \pm 1 \text{ ulp}}$ would be also very close to x and thus would inherit the property of the last 35 bits. So we also need to test that the difference between x and its rounded-to-nearest value in binary64 is smaller than the smallest difference between the cubic root values of two consecutive binary64 values to detect exact cases.

Lemma 2. *Let a be a binary64 number such that $1 \le a < 8$, and $a^{1/3}$ is not exactly representable in binary64. Let x be a binary64 number such that x^3 is also a binary64*

number, and x is closest to $a^{1/3}$ (in case of tie, any value is ok). Then the distance from $a^{1/3}$ to x is at least $4.66 \cdot 10^{-17}$.

Proof. We first deal with the special cases where a is a power of 2. First a cannot be 1, since $1^{1/3}$ is exactly representable in binary64. If $a = 2$, we get $x = 165140/2^{17}$, and $|a^{1/3} - x| > 2 \cdot 10^{-6}$. If $a = 4$, we get $x = 104032/2^{16}$, and $|a^{1/3} - x| > 1 \cdot 10^{-6}$. Now assume that a is not a power of 2. Since x^3 is a binary64 number, and $x^3 \neq a$, we have $|x^3 - a| \geq \mathrm{ulp}(a)$ (since a is not a power of 2). Write $a^{1/3} = x + \varepsilon$. Then $a = x^3 + 3x^2\varepsilon + 3x\varepsilon^2 + \varepsilon^3$. Thus $|3x^2\varepsilon + 3x\varepsilon^2 + \varepsilon^3| \geq \mathrm{ulp}(a)$. In the case where $1 \leq a < 2$, we have $\mathrm{ulp}(a) = 2^{-52}$, and writing $\delta = |\varepsilon|$:

$$\delta \geq \frac{2^{-52}}{3x^2} - \frac{\delta^2}{x} - \frac{\delta^3}{3x^2},$$

where $x \leq x_0 = 165141/2^{17}$. Thus

$$\delta \geq \frac{2^{-52}}{3x_0^2} - \delta^2 - \frac{\delta^3}{3}.$$

The corresponding equation has a single real root $\delta_0 \approx 4.66 \cdot 10^{-17}$, and for $\delta < \delta_0$, the above inequality does not hold. In the case where $2 \leq a < 4$, we have $\mathrm{ulp}(a) = 2^{-51}$, and writing $\delta = |\varepsilon|$:

$$\delta \geq \frac{2^{-51}}{3x^2} - \frac{\delta^2}{x} - \frac{\delta^3}{3x^2},$$

where $x \leq x_1 = 104032/2^{16}$. Thus

$$\delta \geq \frac{2^{-51}}{3x_1^2} - \delta^2 - \frac{\delta^3}{3}.$$

The corresponding equation has a single real root $\delta_1 \approx 5.87 \cdot 10^{-17}$, and for $\delta < \delta_1$, the above inequality does not hold. In the case where $4 \leq a < 8$, we have $\mathrm{ulp}(a) = 2^{-50}$, and writing $\delta = |\varepsilon|$:

$$\delta \geq \frac{2^{-50}}{3x^2} - \frac{\delta^2}{x} - \frac{\delta^3}{3x^2},$$

where $x \leq x_2 = 2$. Thus

$$\delta \geq \frac{2^{-51}}{3x_2^2} - \delta^2 - \frac{\delta^3}{3}.$$

The corresponding equation has a single real root $\delta_2 \approx 7.40 \cdot 10^{-17}$, and for $\delta < \delta_2$, the above inequality does not hold. In summary, for $|\varepsilon| \leq \min(\delta_0, \delta_1, \delta_2)$, the inequality does not hold, thus we have $|\varepsilon| > \min(\delta_0, \delta_1, \delta_2) \geq 4.66 \cdot 10^{-17}$. □

As a consequence of Lemma 2, if the distance from the approximation $x_2^{\mathrm{high}} + x_2^{\mathrm{low}}$—or $x_3^{\mathrm{high}} + x_3^{\mathrm{low}}$—to the nearest binary64 number x is less than $2^{-53}/3$, then $a^{1/3}$ is exactly representable. Indeed, since $|x_2^{\mathrm{high}} + x_2^{\mathrm{low}} - a^{1/3}| < 2^{-76}$, and $|x_2^{\mathrm{high}} + x_2^{\mathrm{low}} - x| < 2^{-53}/3$, this yields $|a^{1/3} - x| < 2^{-53}/3 + 2^{-76} < 4.66 \cdot 10^{-17}$.

To cover the exact cases we test that the last 35 bits of x are identical then to cover the directional modes we round x to the nearest value independently of the FPU status

register in the general purpose registers assuming the exact case. Then we subtract from the rounded value x_2 or x_3 depending on the rounding mode and check that the difference is less than $2^{-53}/3$ according to Lemma 2. In fact the threshold can be any value between 2^{-76} and $2^{-53}/3$ and in the function it set to 2^{-60}. If the result passes the test we restore the status register to the state before the function has been called and return the rounded value.

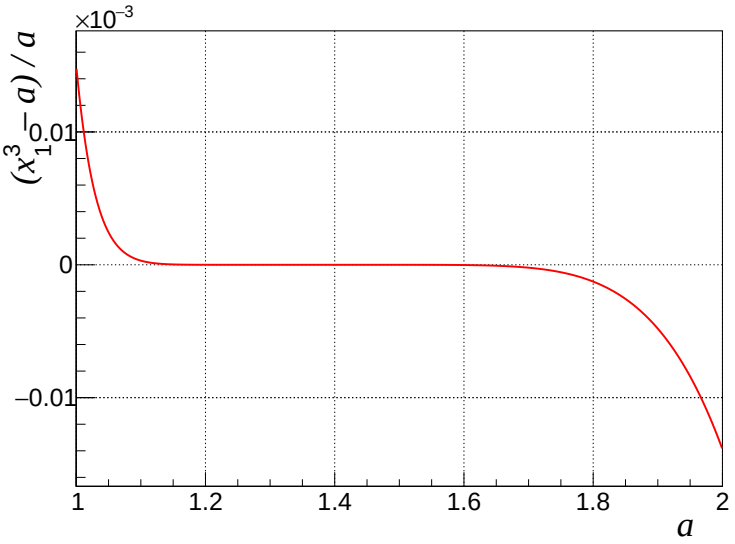

FIGURE 1. The cubic root error h_1 after the first 9th order iteration step starting from $x_0 = 1.104$.

1. Rounding Error Analysis

Below is the C code corresponding to the algorithm proposed above, with a cubic minimax polynomial for the initial approximation, a first cubic Newton iteration in double precision, and another classical second-order Newton iteration in double-double precision. Here zz is the input reduced to the range $[1, 8)]$, and z is reduced to $[1, 2)$. The constants c_0, c_1, c_2, c_3 are those of a minimax polynomial of $x^{1/3}$ over $[1, 2]$, namely (in hexadecimal notation) `c[0]=0x1.1b0babccfef9cp-1`, `c[1]=0x1.2c9a3e94d1da5p-1`, `c[2]=-0x1.4dc30b1a1ddbap-3`, `c[3]=0x1.7a8d3e4ec9b07p-6`. The value `cvt2.f` is either 1 when $1 \le zz < 2$, the approximation `0x1.428a2f98d728bp+0` of $2^{1/3}$ when $2 \le zz < 4$, or when $4 \le zz < 8$ the approximation `0x1.965fea53d6e3dp+0` of $2^{2/3}$. All variables have double precision, and we renamed some variables for a better clarity:

- `r = 1/z`
- `z2 = z*z`
- `c0 = c[0]+z*c[1]`
- `c2 = c[2]+z*c[3]`
- `y0 = c0 + z2*c2`
- `y2a = y0*y0`

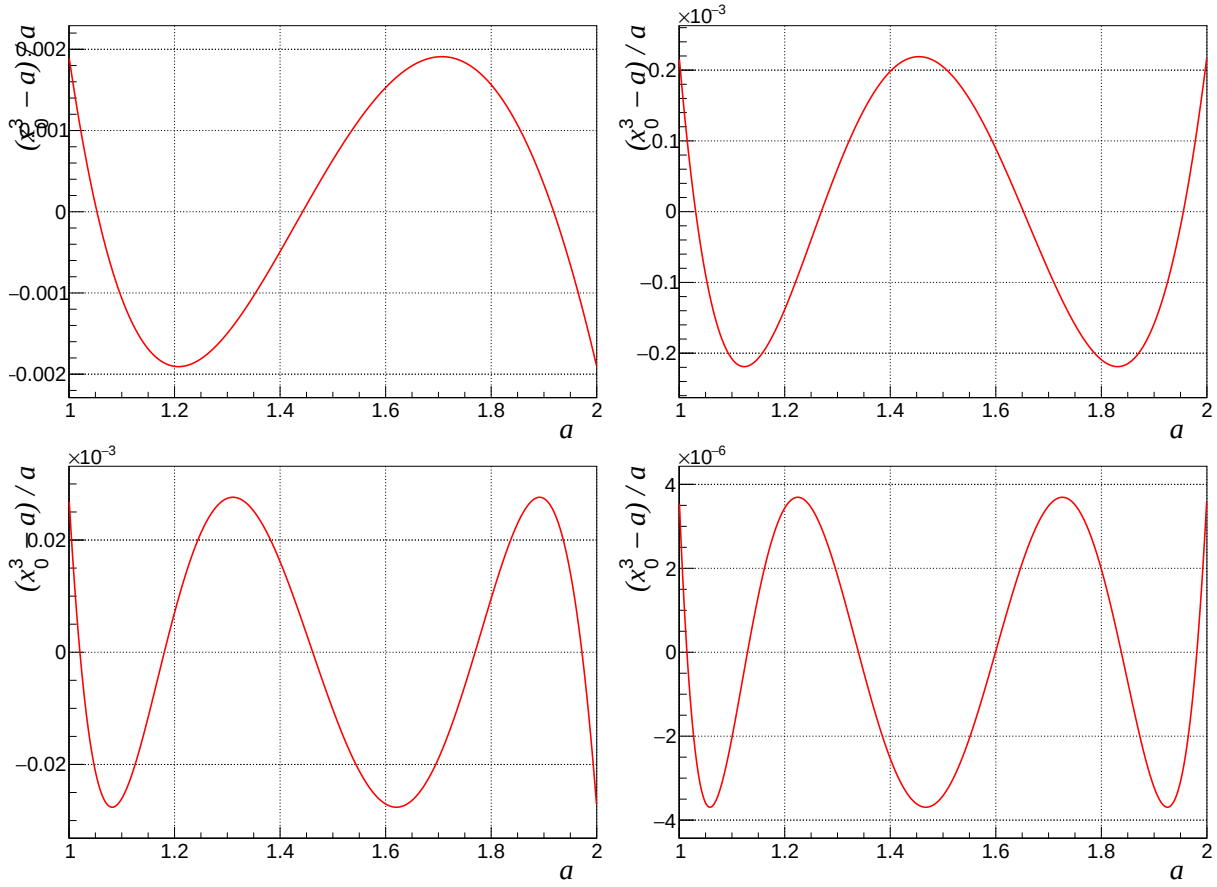

FIGURE 2. The error h_0 of initial approximations. Top-left plot – second order, top-right – third order, bottom-left – fourth order, and bottom-right – the fifth order polynomial.

- `h0 = y2a*(y0*r) - 1`
- `y1 = y0 - (h0*y0)*(u0 - u1*h0)`
- `y1 *= cvt2.f`
- `y2h = y1*y1`
- `y2l = fma(y1,y1,-y2h)`
- `y3 = y2h*y1`
- `y3l = fma(y1,y2h,-y3) + y1*y2l`
- `h1 = ((y3 - zz) + y3l)*rr`
- `dy = h1*(y1*u0)`

Then `y1 - dy` is a good approximation of $zz^{1/3}$.

If there are no rounding errors, the algorithm corresponds to a rational approximation $p(x)/q(x)$, where p has degree 84 and coefficients up to 569 digits, and $q(x) = kx^9$, where k is an integer of 569 digits (in the case $1 \le x \le 2$).

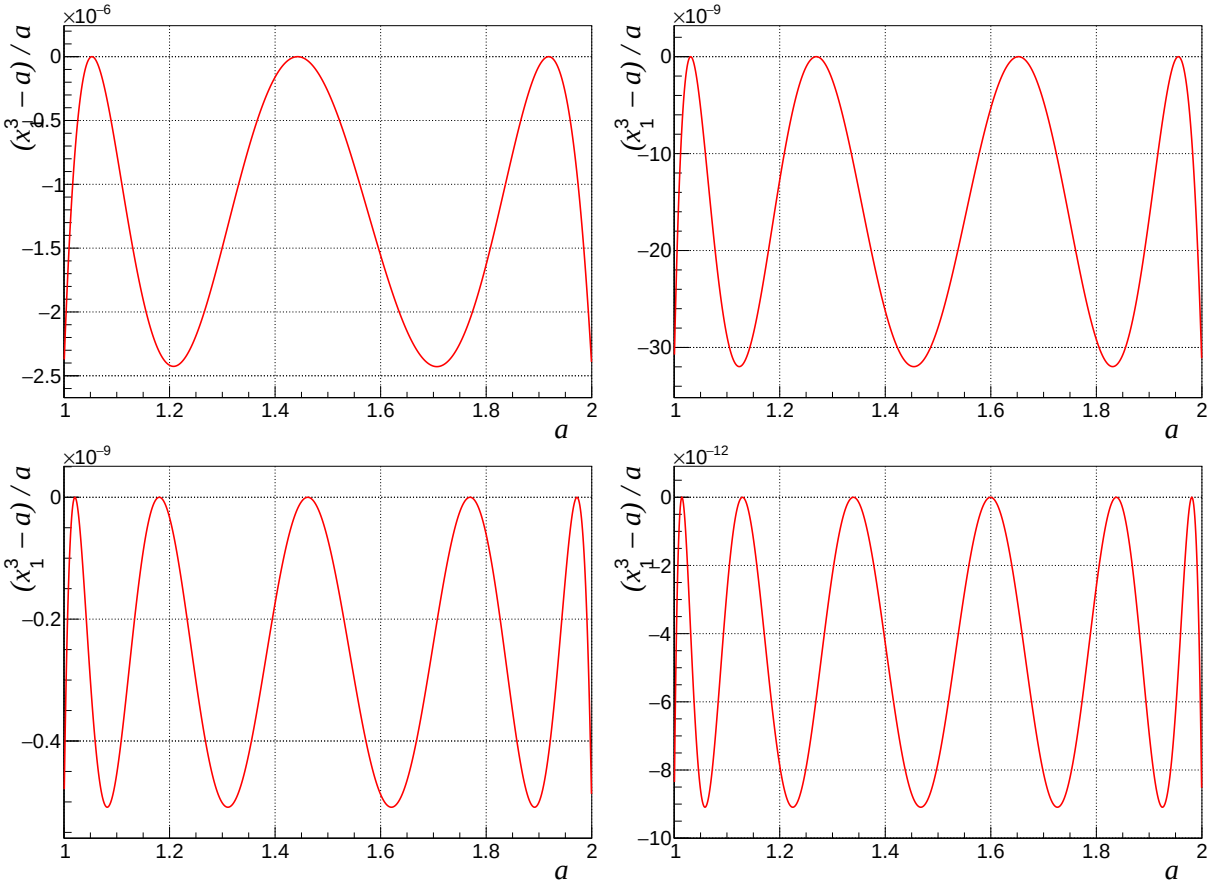

FIGURE 3. The error h_1 after the first second order Newton iteration step for various initial approximations. The plot order is the same as in Fig. 2.

To each floating-point operation which can produce a rounding error, say $a + b$, we associate a variable, say δ, representing the corresponding error. We replace the expression $a + b$ by $a + b + \delta$ in $p(x)/q(x)$, differentiate with respect to δ and replace δ by 0. This yields the first-order derivative, say $f(x)$, of the cubic root approximation $p(x)/q(x)$ with respect to the rounding error δ. We then compute the maximal absolute value of $f(x)$ over the whole interval $[1, 8]$. We call this value the *sensitivity* with respect to δ, and we denote it by s. By the Taylor theorem with explicit remainder, the error in the approximation of $x^{1/3}$ coming from the rounding error in $a + b$ is bounded by s times the maximal value of δ. And for several rounding errors $\delta_0.\delta_1, ...$ with sensitivities $s_0, s_1, ...$, the final error is bounded by $s_0 \max|\delta_0| + s_1 \max|\delta_1| + \cdots$

Note: we take into account that the subtraction `h = y2*(y*r) - 1` is exact due to Sterbenz' theorem.

The two instructions `y2h = y1*y1` and `y2l = fma(y1,y1,-y2h)` compute a double-double approximation `y2h + y2l` of `y1*y1`. In the rounding to nearest mode, we have

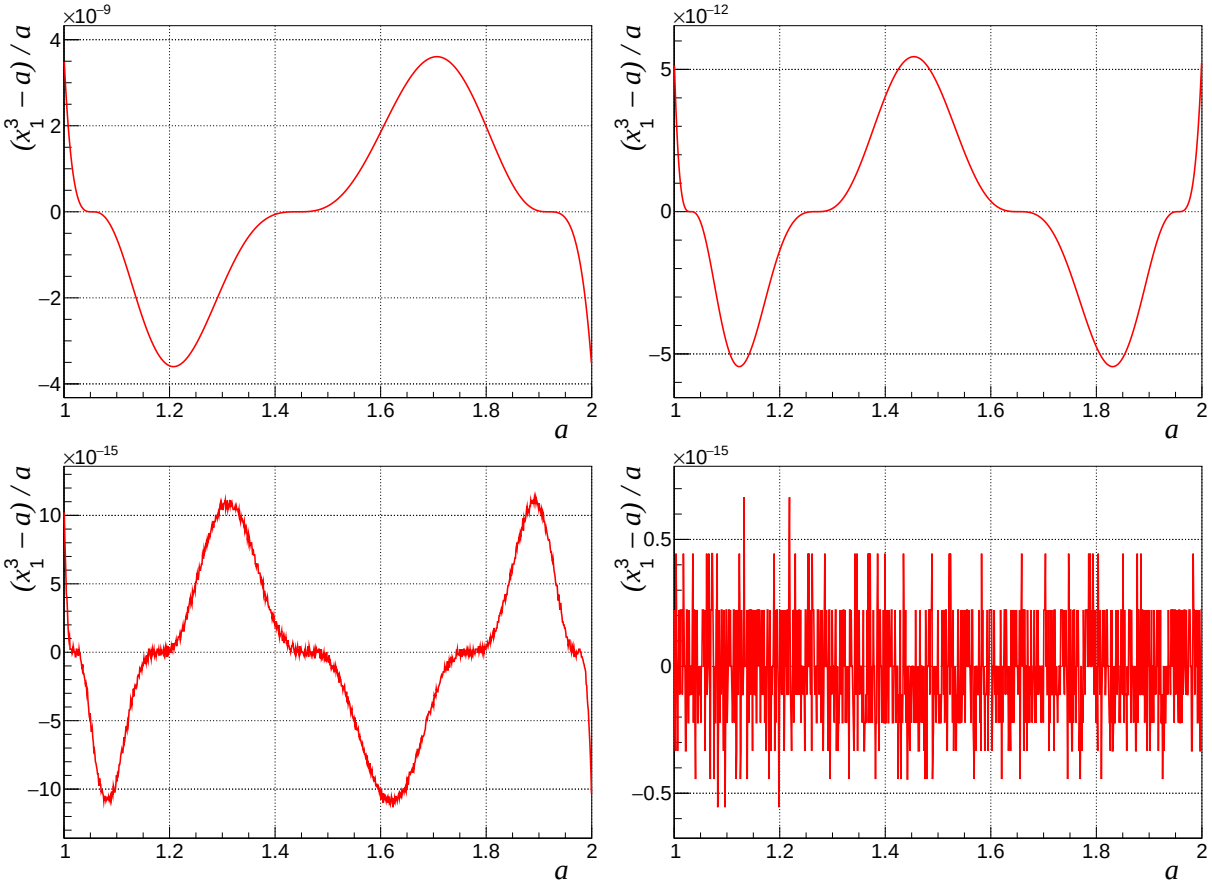

FIGURE 4. The error h_1 after the first third order Newton iteration step for various initial approximations. The plot order is the same as in Fig. 2.

exactly `y2h + y2l = y1*y1`. For directed rounding modes, since `y1*y1` can be represented exactly with 106 bits, we can write `y1*y1=h+l` with `h` being the rounding of `y1*y1` towards zero, and `l` representable in double precision. If `y2h=h`, then `y1*y1-y2h=l` and can be represented exactly, thus `y2h + y2l = y1*y1`. Now if `y2h = nextabove(h)`, then `y1*y1-y2h=h+l-(h+ulp(h))=l-ulp(h)`, and since `ulp(l)` is larger of equal to `ulp(h)` multiplied by 2^{-53}, the difference `ulp(h)-l` is exactly representable. In summary, for all rounding modes we have `y1*y1 = y2h + y2l` exactly. Similarly, we have `y2h*y1 = y3 + y3l` exactly, thus `y1*y1*y1 = y3 + y3l + delta17`, where `delta17` is the rounding error in `y1*y2l`. Since `y1` is less than 2, and `y2l` is less than `ulp(y1*y1)` which is 2^{-52}, `y1*y2l` is bounded by 2^{-51}, and the rounding error on `y1*y2l` is thus $|\delta_{17}| \leq 2^{-104}$.

When one adds all rounding error bounds from Table 1, one gets a maximum error (due to roundings) of $1.13 \cdot 10^{-26}$. If we add the $1.13 \cdot 10^{-26}$ bound for the rounding error to the $1.32 \cdot 10^{-23}$ bound for the mathematical error, we get a global bound of $1.322 \cdot 10^{-23} < 2^{-76}$, thus we can use 2^{-76} as error margin in the rounding test.

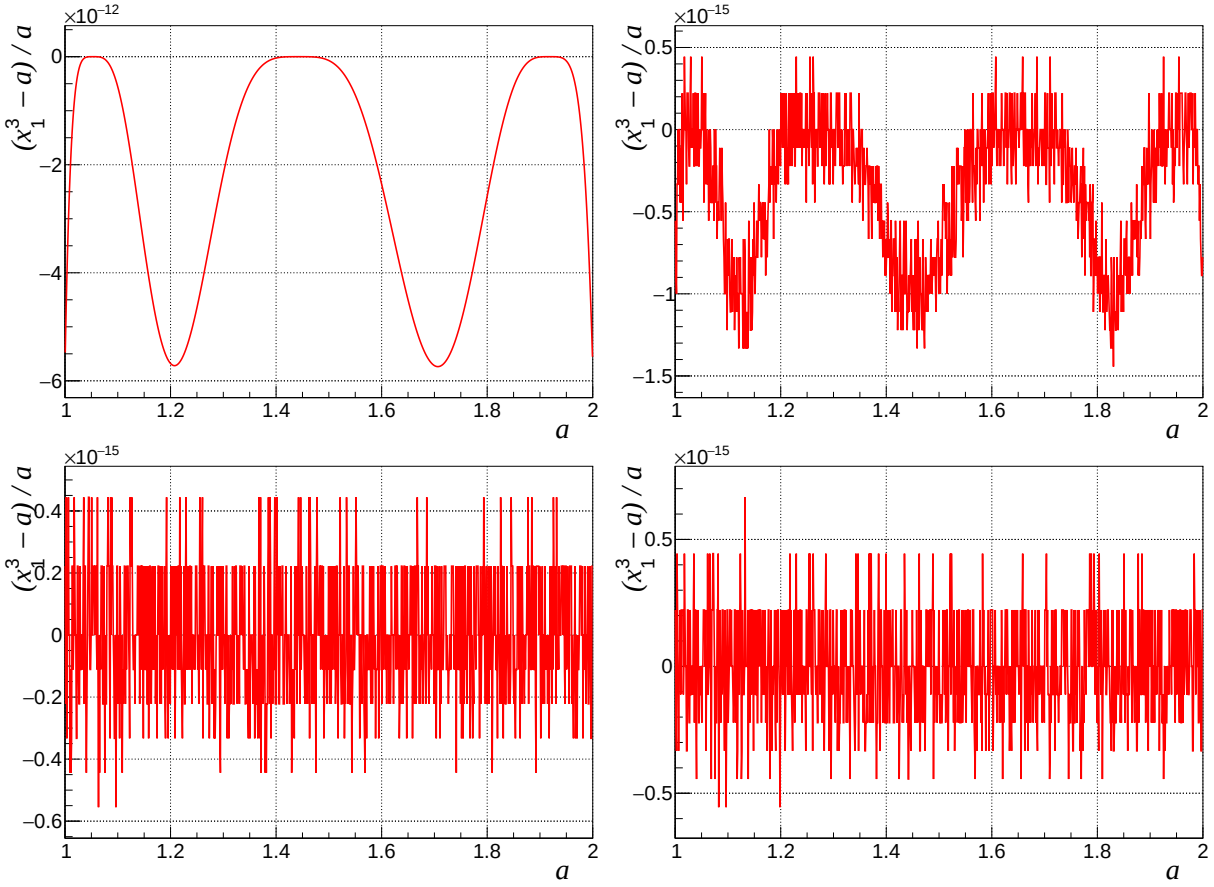

FIGURE 5. The error h_1 after the first quartic order Newton iteration step for various initial approximations. The plot order is the same as in Fig. 2.

REFERENCES

[1] BOLDO, S., GRAILLAT, S., AND MULLER, J. On the robustness of the 2Sum and Fast2Sum algorithms. *ACM Trans. Math. Softw. 44*, 1 (2017), 4:1–4:14.

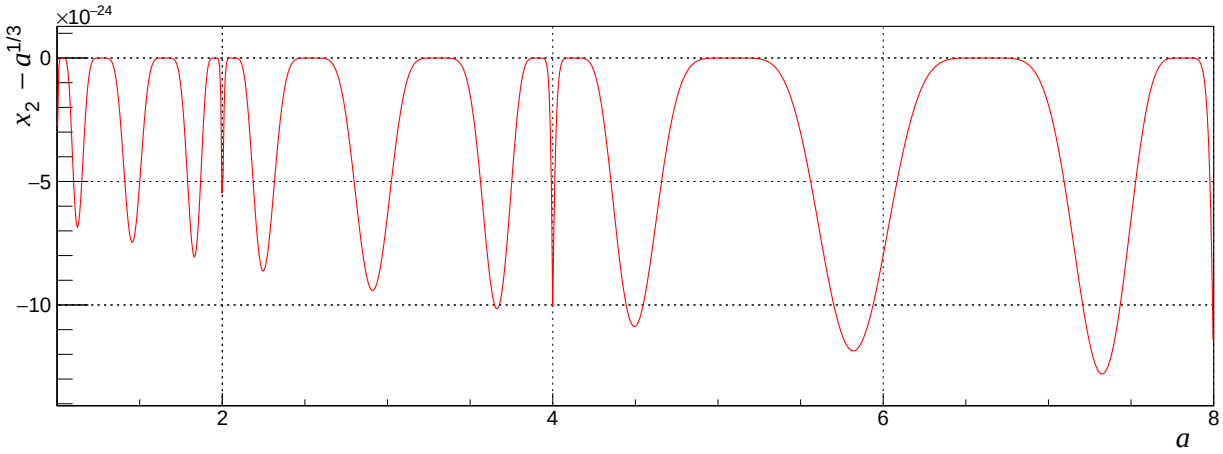

FIGURE 6. The error of the cubic root evaluation after the refinement step where the root x_2 is represented as an unevaluated sum of two numbers in binary64 $x_2 = x_2^{\text{high}} + x_2^{\text{low}}$.

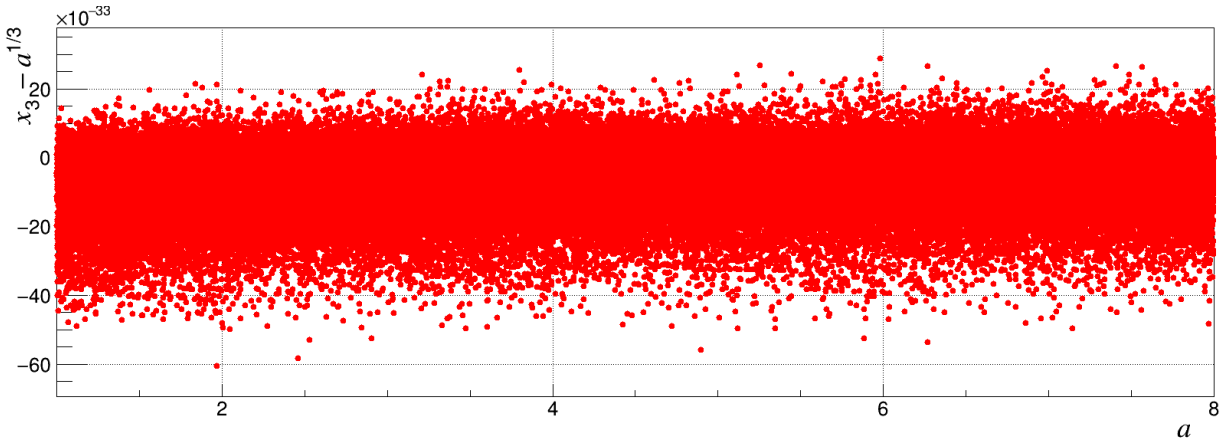

FIGURE 7. The error of the cubic root evaluation for the worst case when the rounding test fails and the additional Newton iteration step is taken. FPU is operating in the round-to-nearest mode.

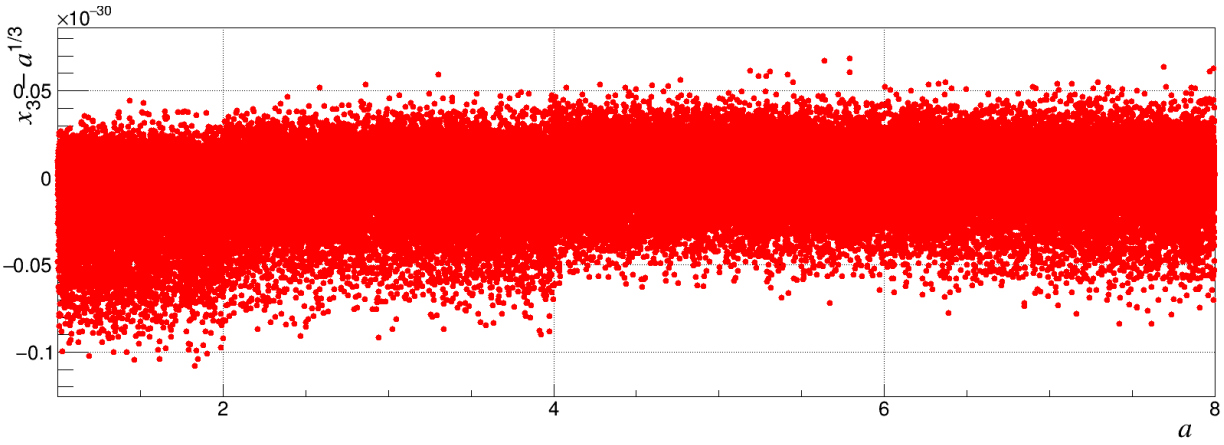

FIGURE 8. The error of the cubic root evaluation for the worst case when the rounding test fails and the additional Newton iteration step is taken. FPU is operating in the downward mode.

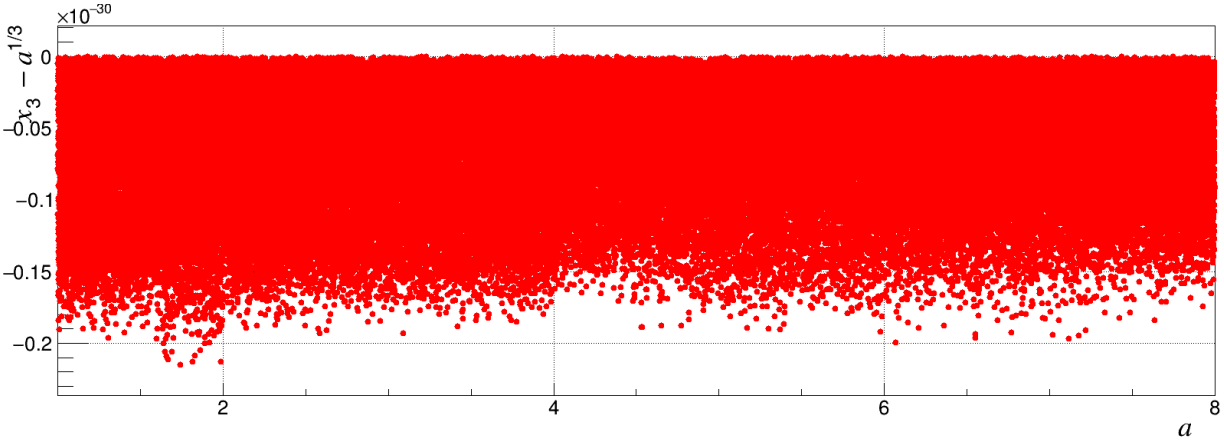

FIGURE 9. The error of the cubic root evaluation for the worst case when the rounding test fails and the additional Newton iteration step is taken. FPU is operating in the upward mode.

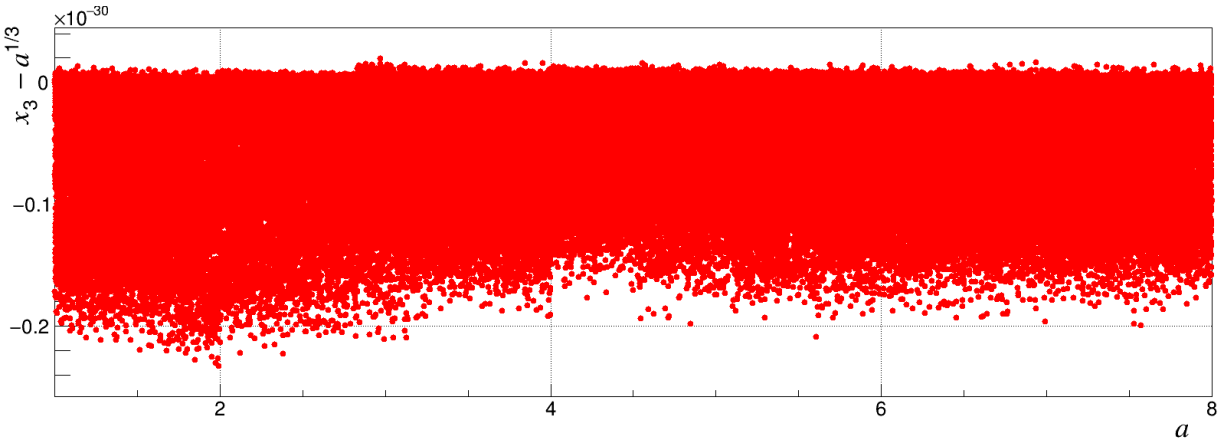

FIGURE 10. The error of the cubic root evaluation for the worst case when the rounding test fails and the additional Newton iteration step is taken. FPU is operating in the toward-zero mode.

δ_i	instruction	sensitivity s_i	$\max\lvert\delta_i\rvert$	$s_i\max\lvert\delta_i\rvert$
δ_0	`r=1/z`	$2^{-38.5}$	2^{-53}	$2^{-91.5}$
δ_1	`z2 = z*z`	$2^{-62.8}$	2^{-51}	$2^{-113.8}$
δ_2	`z*c[1]`	$2^{-60.0}$	2^{-52}	$2^{-112.0}$
δ_3	`c[0]+z*c[1]`	$2^{-60.0}$	2^{-52}	$2^{-112.0}$
δ_4	`z*c[3]`	$2^{-58.0}$	2^{-57}	$2^{-115.0}$
δ_5	`c[2]+z*c[3]`	$2^{-58.0}$	2^{-55}	$2^{-113.0}$
δ_6	`z2*c2`	$2^{-60.0}$	2^{-54}	$2^{-104.0}$
δ_7	`y0=c0+z2*c2`	$2^{-60.0}$	2^{-52}	$2^{-102.0}$
δ_8	`y2a=y0*y0`	$2^{-37.9}$	2^{-52}	$2^{-89.9}$
δ_9	`y0*r`	$2^{-36.9}$	2^{-53}	$2^{-89.9}$
δ_{10}	`y2a*(y0*r)`	$2^{-37.5}$	2^{-52}	$2^{-89.5}$
δ_{11}	`h0*y0`	$2^{-37.9}$	2^{-64}	$2^{-101.9}$
δ_{12}	`u1*h0`	$2^{-48.1}$	2^{-67}	$2^{-115.1}$
δ_{13}	`u0-u1*h0`	$2^{-48.1}$	2^{-54}	$2^{-102.1}$
δ_{14}	`(h0*y0)*(u0-u1*h0)`	$2^{-36.3}$	2^{-65}	$2^{-101.3}$
δ_{15}	`y1=y0-...`	$2^{-36.3}$	2^{-52}	$2^{-88.3}$
δ_{16}	`y1 *= cvt2.f`	$2^{-37.0}$	2^{-51}	$2^{-88.0}$
δ_{17}	error on `y1*y1*y1`	$2^{-1.5}$	2^{-104}	$2^{-105.5}$
δ_{18}	`h1 = ((y3 - zz) + y3l)*rr`	$2^{-0.5}$	2^{-90}	$2^{-90.5}$
δ_{19}	`y1*U0`	$2^{-37.4}$	2^{-52}	$2^{-89.4}$
δ_{20}	`h1*(y1*U0)`	1	2^{-91}	$2^{-91.0}$

TABLE 1. The sensitivities s_i and maximal values of δ_i for all rounding errors that can occur in the algorithm.