A Test for Correctly Rounded SQRT

Prof. W. Kahan
Univ. of Calif. @ Berkeley

Abstract
Ideally, the computed value of SQRT(x) should be the same as if $\sqrt{x}$ had first been computed perfectly, correct to infinitely many significant bits, and then rounded to the nearest approximation that fits into the number of significant bits, say N sig. bits, in the target's format. This ideal is achievable; it is mandatory under IEEE Standard 754 for Binary Floating-Point Arithmetic, to which almost all computers built nowadays attempt to conform. The tricky part of the SQRT algorithm is getting the last rounding error right so that SQRT(x) will differ by less than 1/2 *ulp* (*U*nit in its *L*ast *P*lace) from $\sqrt{x}$. If the trick is fumbled very slightly, the error in SQRT can exceed 1/2 ulp so rarely that random testing has practically no chance of exposing the flaw; such flaws have occurred and have evaded detection by billions of random tests. The tests discussed in this note are focussed sharply enough to reveal such flaws almost immediately.

These tests are intended only for iterative SQRT programs that have already been "proved" and tested correct in all but the last bit or two. Tests different from these are needed for digit–by–digit algorithms whose early digits could be wrong only with extremely low probability.

P-adic Square Roots

These tests exploit " P–adic Square Roots." For every integer k that has such a square root, it is a string of bits proceeding infinitely to the left of the binary point whose trailing n-1 bits, for each $n > 2$, is one of k 's square roots mod 2^n , an (n-1)–bit integer I whose square matches k in its rightmost n bits; in other words, $I^2 \equiv k \mod 2^n$. Computing a P-adic square root from right to left is easy when it exists, which is just when the rightmost nonzero bit of k is preceded by at least two zeros and followed by an even number (perhaps 0) of zeros; in fact, every such k has two P-adic square roots and four square roots mod 2^n , as we shall see.

How do these integers k and their P–adic square roots figure in the tests? They provide essentially every test argument x for which the correctly rounded SQRT(x) differs by very slightly less than 1/2 ulp from $\sqrt{x}$. Among these arguments must lie all instances x for which a slightly incorrectly rounded SQRT(x) differs from $\sqrt{x}$ by very slightly more than 1/2 ulp. Therefore these are the arguments x that we shall wish to test after no errors bigger than 1/2 ulp have been revealed by random testing so extensive that we have come to suspect that such errors, if any exist, can exceed 1/2 ulp at most very slightly. That kind of suspicion is appropriate for iterative algorithms like Heron's Rule (which replaces an approximation $s \approx \sqrt{x}$ by a better $(s + x/s)/2$) because their best provable error bounds are piecewise continuous functions of x that describe intervals in which the observed errors are dense. However, error analysis is not the subject of this note, so no more will be said about proving error bounds.

Binary Floating-Point Square Roots

Every positive *normalized* (others require special tests) binary floating-point number $y = 2^E \cdot Y$ can be represented with an integer exponent E and an N–bit integer Y in the range

$$2^{N-1} \le Y < 2^N ,$$

where N is the number of significant bits supported by the floating-point format in question: $N = 24$ for IEEE Single (`float` or REAL*4) ; $N = 53$ for IEEE Double (REAL*8) ; $N = 64$ for IEEE Double Extended (`long double` or REAL*10) ; $N = 113$ for Quadruple (`long double` or REAL*16) .

We presume that the significand Y is independent of the exponent E when $y = \mathrm{SQRT}(x)$ is computed; a conscientious tester will test this presumption, paying special attention to test arguments x at the extremes of the exponent range. In the absence of Over/Underflow, the tested SQRT should satisfy $\mathrm{SQRT}(4 \cdot x) = 2 \cdot \mathrm{SQRT}(x)$ and $\mathrm{SQRT}(y^2) = |y|$ exactly for all arguments tested, including many random arguments y for which y^2 must be rounded off as well as a large number of small integers y . Only after SQRT has passed easy tests like these are the following more expensive focussed tests worth exercising.

Now set exponent $E := 0$ and restrict attention exclusively to values x in two abutting ranges:

$$x = 2^{N-j} \cdot X \quad \text{with} \quad 2^{N-1} \le X < 2^N \quad \text{and} \quad j = 0 \ \text{or} \ j = 1 .$$

The value pairs (x, Y) that most interest us are those for which $\sqrt{x} \approx Y + 1/2$ as nearly as possible. (For no x can $\sqrt{x} = Y + 1/2$ exactly since $4x = (2Y + 1)^2 > 2^{2N}$ would be an odd integer too wide to fit into the N sig. bits of a floating-point number.)

If $\sqrt{x} \approx Y + 1/2$ very nearly then $k := (2Y + 1)^2 - 4x$ must have a small magnitude; and k is an integer. It must be an odd integer, and it must satisfy

$$(2Y + 1)^2 \equiv k \mod 2^{N+2-j} .$$

Since $(2Y + 1)^2 = 8(Y(Y+1)/2) + 1$, we infer first that $k \equiv 1 \mod 8$ and, second, that when any such k is given we can obtain $2Y + 1$ from a P–adic square root of k as follows.

Constructing P-adic Square Roots

Given $k \equiv 1 \mod 8$, we shall construct its square roots $\mod 2^n$ for $n = 3, 4, 5, \ldots$ in turn by means of a well-known recurrence called "Hensel Lifting." There are four square roots for each n . To see why, suppose I is one of them; suppose $0 < I < 2^n$ and $I^2 = k \mod 2^n$. Then another square root is $2^n - I$, so we might as well swap them if necessary to arrange that $I < 2^{n-1}$ too. But then another square root is $2^{n-1} - I$, so we might as well assume $0 < I < 2^{n-2}$. This I is the smallest of four square roots, namely

$$I , \quad 2^{n-1} - I , \quad 2^{n-1} + I \quad \text{and} \quad 2^n - I ,$$

one in each respectively of the four abutting open intervals

$$(0 , 2^{n-2}) , \quad (2^{n-2}, 2^{n-1}) , \quad (2^{n-1}, 3 \cdot 2^{n-2}) \quad \text{and} \quad (3 \cdot 2^{n-2}, 2^n) .$$

(Why not more than four square roots? If there were more, there would have to be at least eight, at least two in each of those open intervals. Suppose I and J are square roots in the leftmost interval, both odd integers satisfying $0 < I \le J < 2^{n-2}$ and $J^2 - I^2 \equiv 0 \mod 2^n$. Now $(J+I)/2$ and $(J-I)/2$ must both be integers, one of them odd since their sum is odd; but their product is $((J+I)/2)\cdot((J-I)/2) = (J^2-I^2)/4 \equiv 0 \mod 2^{n-2}$. Therefore one of the factors $(J\pm I)/2 \equiv 0 \mod 2^{n-2}$, and since $0 \le (J\pm I)/2 < 2^{n-2}$ we conclude that $J = I$. This confirms that k has just four square roots $\mod 2^n$, one in each interval, or no square roots.)

Now, for any $n > 2$ let I_n, satisfying $I_n^2 \equiv k \mod 2^n$ and $0 < I_n < 2^{n-2}$, denote the smallest square root $\mod 2^n$ of a given $k \equiv 1 \mod 8$. Evidently $I_3 = 1$. To demonstrate the existence of I_n for all $n > 2$ we shall obtain I_{n+1} recursively from I_n. To this end consider integer

$$R_n := (I_n^2 - k)/2^n$$

and select

$$\begin{aligned} I_{n+1} &:= I_n \quad \text{whenever } R_n \text{ is an even integer,} \\ &:= 2^{n-1} - I_n \quad \text{whenever } R_n \text{ is odd.} \end{aligned}$$

Evidently $R_{n+1} = R_n/2$ whenever R_n is even, and $R_{n+1} = 2^{n-3} + (R_n - I_n)/2$ otherwise, so both I_{n+1} and R_{n+1} can be computed by recurrence without ever squaring big integers I_n. Moreover, R_n never gets very big; $|R_n| < 2^{n-4}$ for all sufficiently big n. Therefore floating-point arithmetic with N sig. bits suffices to compute I_{N+2} and R_{N+2} for each eligible integer $k =$ ..., -47, -39, -31, -23, -15, -7, 1, 9, 17, 25, 33, 41, 49, ... that is not too big.

I_n need not converge as $n \longrightarrow \infty$, and yet the sequence $\{ I_3, I_4, I_5, ..., I_n, ... \}$ does determine the two P-adic square roots of k. They are two's complements each of the other, and every I_{N+2} coincides with the last N bits of one of them. For instance, when $k = 1$ every $I_n = 1$; when $k = 9$ every $I_n = 3$ except $I_3 = 1$; but the P-adic square roots of $k = -7$ and $k = 17$ are nontrivial: in hexadecimal notation,

$$\sqrt{-7} = \pm...\ \text{EA39F1BF 73C0523A 19B4BB63 9C98C0B5.}$$
$$\sqrt{17} = \pm...\ \text{2DAD432D 4049AC1C 85A241F3 33D326E9.}$$

The foregoing recurrence computes P-adic square roots one bit per step $n \longrightarrow n+1$. Versions of that recurrence modified to compute several bits per step may run faster on some computers.

Selection of Test Arguments

Recall that for each eligible $k \equiv 1 \mod 8$ we seek test arguments $x = (Y + 1/2)^2 - k/4 = 2^{N-j}\cdot X$ with $2^{N-1} \le X < 2^N$ and $j = 0$ or $j = 1$, so $(2Y + 1)^2 \equiv k \mod 2^{N+2-j}$. Thus, $2Y + 1$ must be one of the four square roots $\mod 2^{N+2-j}$ of k, and $2^{N+(1-j)/2} < 2Y + 1 < 2^{N+1-j/2}$ if $|k|$ is not too big. If such a pair (x, Y) exists, the test consists of computing SQRT(x); it should round to Y + SignBit(k) if rounded correctly. (SignBit(k) is 1 if $k < 0$, otherwise 0 .) Now define abbreviations $I := I_{N+2}$ and $R := R_{N+2}$, so that $0 < I < 2^N$ and $I^2 = 2^{N+2}\cdot R + k$.

There are three cases to consider:

If $I > 2^{N+1/2}/(1 + \sqrt{2}) = 2^{N+1} - 2^{N+1/2}$, no test–pair (x, Y) exists for the chosen k and N . This seems to occur for fewer than 42% of eligible values k chosen at random.

If $2^N/(1 + \sqrt{2}) = 2^{N+1/2} - 2^N < I < 2^{N+1/2}/(1 + \sqrt{2}) = 2^{N+1} - 2^{N+1/2}$, one test–pair (x, Y) is obtained from $Y := 2^N - (I+1)/2$, so $2^{2N-1} < (Y + 1/2)^2 = 2^N \cdot X + k/4 < 2^{2N-1} \cdot 1.257...$, and from $x := 2^N \cdot X$ where $2^{N-1} < X := 2^N - I + R < 2^{N-1} \cdot 1.257...$. This seems to occur for about 17% of random eligible choices k .

If $I < 2^N/(1 + \sqrt{2}) = 2^{N+1/2} - 2^N$, two test–pairs (x, Y) are obtained, one from the same formulas as before: $Y := 2^N - (I+1)/2$, so $2^{2N-1} \cdot 1.257... < (Y + 1/2)^2 = 2^N \cdot X + k/4 < 2^{2N}$, and $x := 2^N \cdot X$ where $2^{2N-1} \cdot 1.257... < X := 2^N - I + R < 2^N$. The second test–pair comes from $Y := 2^{N-1} + (I-1)/2$, so $2^{2N-2} < (Y + 1/2)^2 = 2^{N-1} \cdot X + k/4 < 2^{2N-1}$, and from $x := 2^{N-1} \cdot X$ where $2^{N-1} < X := 2^{N-1} + I + 2R < 2^N$. This seems to occur for about 41% of random eligible choices k .

The foregoing three cases supply about one test–pair (x, Y) per eligible k on average, and more at first if k is run through eligible values in order of increasing magnitude.

. . . . . .

If the foregoing tests will be run often, an array of a million or so test arguments x with a correctly rounded SQRT(x) = Y + SignBit(k) for each is worth computing and storing on a disk.

Similar procedures can and should be used to construct test–pairs (x, Y) for each of the other “Directed” rounding modes mandated by IEEE 754.

In the mandatory default rounding mode, “To Nearest”, certain computers may fail the test described in this note not because of a hardware defect but because of double rounding that is an artifact of the way some compilers perform evaluations of arithmetic expressions. The computers are Intel x86/Pentium-based PCs and their clones, and 680x0-based Apple Macintoshes; these machines have hardware support for three floating-point formats providing respectively 24, 53 and 64 sig. bits of precision. However, most compilers for these machines lack declarations for variables with 64 sig. bits, and yet those compilers evaluate all floating-point expressions to 64 sig. bits in registers before rounding them to the narrower format of a variable stored in memory. Consequently a program intended to test SQRT rounded to 53 sig. bits may actually be testing SQRT double-rounded first to 64 and then to 53 sig. bits. Double rounding rarely matters, and then it matters little, but it matters enough to fail the test. To prevent this failure, set bits in these machines’ Floating-Point Control Register to force every arithmetic operation to be rounded to 53 sig. bits whenever its result is generated in a register that would otherwise hold 64 sig. bits.

. . . . . .

Acknowledgment
I am indebted to Prof. H. Heilbronn for teaching me how to do Hensel Lifting in the early 1960s at the University of Toronto.