Augmented precision square roots, 2-D norms, and discussion on correctly rounding $\sqrt{x^2+y^2}$

Nicolas Brisebarre, Mioara Joldeş,
Érik Martin-Dorel, Jean-Michel Muller
Laboratoire LIP
CNRS, ENS Lyon, INRIA, Univ. Claude Bernard Lyon 1
Lyon France
Email: first-name.Last-name@ens-lyon.fr

Peter Kornerup
Dept. of Mathematics and Computer Science
University of Southern Denmark
Odense, Denmark
Email: kornerup@imada.sdu.dk

***Abstract*—Define an "augmented precision" algorithm as an algorithm that returns, in precision-p floating-point arithmetic, its result as the unevaluated sum of two floating-point numbers, with a relative error of the order of 2^{-2p}. Assuming an FMA instruction is available, we perform a tight error analysis of an augmented precision algorithm for the square root, and introduce two slightly different augmented precision algorithms for the 2D-norm $\sqrt{x^2+y^2}$. Then we give tight lower bounds on the minimum distance (in ulps) between $\sqrt{x^2+y^2}$ and a midpoint when $\sqrt{x^2+y^2}$ is not itself a midpoint. This allows us to determine cases when our algorithms make it possible to return correctly-rounded 2D-norms.**

***Index Terms*—Floating-point arithmetic; compensated algorithms; square-root; Correct rounding; 2D-norms; accurate computations.**

I. Introduction

In some applications, just returning a floating-point approximation y_h to the exact result y of a function or arithmetic operation may not suffice. It may be useful to also return an estimate y_ℓ of the error (i.e., $y \approx y_h + y_\ell$). For simple enough functions (e.g., addition or multiplication), it is even possible to have $y = y_h + y_\ell$ exactly. Having an estimate of the error makes it possible to re-use it later on in a numerical algorithm, in order to at least partially compensate for that error. Such *compensated* algorithms have been suggested in the literature for summation of many floating-point numbers [1]–[6], computation of dot products [7], and evaluation of polynomials [8].

We will call *augmented-precision algorithm* an algorithm that returns, in radix-β, precision-p floating-point arithmetic, an approximation $y_h + y_\ell$ (i.e., an unevaluated sum of two floating-point numbers) to an exact result $y = f(x)$ (or $f(x_1, x_2)$) such that

- $y_\ell \leq \frac{1}{2}\,\mathrm{ulp}(y_h)$;
- there exists a small constant C (say, $C \ll \beta^p$) such that

$$|y - (y_h + y_\ell)| < C \cdot \beta^{-2p} \cdot y.$$

This work is partially supported by the CIBLE programme of Région Rhône-Alpes, France.

This work is partially supported by the TaMaDi project of the french *Agence Nationale de la Recherche.*

When $y = y_h + y_\ell$ exactly, the transformation that generates y_h and y_ℓ from the inputs of f is called an *error-free transform* in the literature.

The unevaluated sum $y_h + y_\ell$ is called an *expansion*. Several algorithms have been suggested for performing arithmetic on such expansions [3], [9]–[11].

In the first part of this paper, we briefly recall two well-known error-free transforms used later on in the paper. Then, we analyze an augmented-precision algorithm for the square-root. In the third part, we use that algorithm for designing augmented-precision algorithms for computing $\sqrt{x^2+y^2}$, where x and y are floating-point numbers. Such a calculation appears in many domains of scientific computing. It is also an important step when computing complex square roots. The naive method—i.e., straightforward implementation of the formula $\sqrt{x^2+y^2}$—may lead to spurious overflows or underflows. When there are no overflows/underflows, it is quite accurate (an elementary calculation shows that on a radix-2, precision-p floating-point system, the relative error is bounded by $2^{-p+1} + 2^{-2p}$).

Friedland [12] avoids spurious overflows by computing $\sqrt{x^2+y^2}$ as $|x| \cdot \sqrt{1+(y/x)^2}$ if $|x| \geq |y|$, and $|y| \cdot \sqrt{1+(x/y)^2}$ otherwise.

Kahan[1], and Midy and Yakovlev [13] normalize the computation using a power of the radix of the computer system: in radix 2, if $|x| \geq |y|$, let b_x be the largest power of 2 less than or equal to x, what they actually compute is $b_x \cdot \sqrt{(x/b_x)^2 + (y/b_x)^2}$. Their solution is less portable (and possibly on some systems, less fast) than Friedland's solution, yet it will be in general slightly more accurate, since division and multiplication by b_x is exact. Our augmented-precision algorithms will derive from this one. As noticed by Kahan, the IEEE 754 Standard for Floating-Point Arithmetic [14] defines functions `scaleB` and `logB` that make this scaling of x and y easier to implement.

Hull et al. [15] use the naive method along with the exception-handling possibilities specified by the IEEE 754-1985 Standard to recover a correct result when the naive method fails.

[1]Unpublished lecture notes

In the fourth part, we investigate the possibility of correctly rounding $\sqrt{x^2+y^2}$ (assuming round-to-nearest). This requires solving the *table maker's dilemma* for that function, i.e., finding a lower bound on the smallest possible nonzero distance (in ulps, or in terms of relative distance) between $\sqrt{x^2+y^2}$ and a midpoint, where a midpoint is the exact middle of two consecutive floating-point numbers.

II. Two well-known Error-free transforms

A. The Fast2Sum algorithm

The Fast2Sum algorithm was first introduced by Dekker [16], but the three operations of this algorithm already appeared in 1965 as a part of a summation algorithm, called "Compensated sum method," due to Kahan [1]. Under conditions spelled out by Theorem 1, it returns the floating term s nearest to a sum $a+b$ and the error term $t=(a+b)-s$. Throughout the paper, $\mathrm{RN}(u)$ means "u rounded to the nearest even" (see [19]).

Algorithm 1. *Fast2Sum*

$s \leftarrow \mathrm{RN}(a+b)$
$z \leftarrow \mathrm{RN}(s-a)$
$t \leftarrow \mathrm{RN}(b-z)$
return (s,t)

The following theorem is due to Dekker.

Theorem 1 (Fast2Sum algorithm [16])**.** *Assume the floating-point system being used has radix $\beta \leq 3$, subnormal numbers available, and provides correct rounding to nearest.*

Let a and b be floating-point numbers, and assume that the exponent of a is larger than or equal to that of b (this condition is satisfied if $|a| \geq |b|$). Algorithm 1 computes two floating-point numbers s and t that satisfy the following:

- *$s+t=a+b$ exactly;*
- *s is the floating-point number that is closest to $a+b$.*

B. The 2MultFMA algorithm

The FMA instruction makes it possible to evaluate $\pm ax \pm b$, where a, x, and b are floating-point numbers, with one final rounding only. That instruction was introduced in 1990 on the IBM RS/6000. It allows for faster and, in general, more accurate dot products, matrix multiplications, and polynomial evaluations. It also makes it possible to design fast algorithms for correctly rounded division and square root [17].

The FMA instruction is included in the newly revised IEEE 754-2008 standard for floating-point arithmetic [14].

If an FMA instruction is available, then, to compute the error of a floating-point multiplication $x_1 \cdot x_2$, one can design a very simple algorithm, which only requires two consecutive operations, and works for any radix and precision, provided the product does not overflow and $e_{x_1}+e_{x_2} \geq e_{\min}+p-1$, where e_{x_1} and e_{x_2} are the exponents of x_1 and x_2, and $e_{\min}$ is the minimum exponent of the floating-point system.

Algorithm 2 (2MultFMA(x_1,x_2))**.**

$r_1 \leftarrow \mathrm{RN}(x_1 \cdot x_2)$
$r_2 \leftarrow \mathrm{RN}(x_1 \cdot x_2 - r_1)$
return (r_1,r_2)

III. Augmented-precision real square root with an FMA

Let us now present an augmented-precision real square root algorithm. That algorithm is straightforwardly derived from the following theorem, given in [18] (see also [19]):

Theorem 2 (Computation of square root residuals using an FMA [18])**.** *Assume x is a precision-p, radix-β, positive floating-point number. If σ is $\sqrt{x}$ rounded to a nearest floating-point number then*

$$x-\sigma^2$$

is exactly computed using one FMA instruction, with any rounding mode, provided that

$$2e_\sigma \geq e_{\min}+p-1, \tag{1}$$

where e_σ is the exponent of σ.

Algorithm 3 (Augmented computation of $\sqrt{x}$)**.**

$\sigma \leftarrow \mathrm{RN}(\sqrt{x})$
$t \leftarrow x-\sigma^2$ *(exact operation through an FMA)*
$r \leftarrow \mathrm{RN}(t/(2\sigma))$
return (σ,r)

Notice that similar approximations are used in [20] in a different context (to return a correctly-rounded square root from an accurate enough approximation), as well as in [10] for manipulating floating-point expansions. This is not surprising, since behind this approximation there is nothing but the Taylor expansion of the square-root. What we do claim here, is that we have been able to compute a very tight error bound for Algorithm 3 (indeed, an asymptotically optimal one, as we will see later on). That error bound is given by the following theorem, which shows that the number r returned by Algorithm 3 is a very sharp estimate of the error $\sqrt{x}-\sigma$.

Theorem 3. *In radix-2, precision-p arithmetic, if the exponent e_x of the FP number x satisfies $e_x \geq e_{\min}+p$, then the output (σ,r) of Algorithm 3 satisfies $\sigma=\mathrm{RN}(\sqrt{x})$ and*

$$\left|(\sigma+r)-\sqrt{x}\right| < 2^{-p-1}\,\mathrm{ulp}(\sigma),$$

and

$$\left|(\sigma+r)-\sqrt{x}\right| < 2^{-2p}\cdot\sigma.$$

Proof. First, if $e_x \geq e_{\min}+p$ then $x \geq 2^{e_{\min}+p}$, so that

$$\sqrt{x} \geq 2^{\frac{e_{\min}+p}{2}} \geq 2^{\left\lfloor \frac{e_{\min}+p}{2} \right\rfloor},$$

which implies

$$\sigma=\mathrm{RN}(\sqrt{x}) \geq 2^{\left\lfloor \frac{e_{\min}+p}{2} \right\rfloor},$$

therefore,

$$e_\sigma \geq \left\lfloor \frac{e_{\min}+p}{2} \right\rfloor,$$

so that we have,

$$2e_\sigma \geq e_{\min} + p - 1.$$

Therefore, Theorem 2 applies: $t = x - \sigma^2$ is a floating-point number, so that it is exactly computed using an FMA instruction.

Now, since $\sigma = \mathrm{RN}(\sqrt{x})$ and σ is a normal number (a square root never underflows or overflows), and since the square root of a floating-point number is never equal to a midpoint [20], [21], we have

$$|\sigma - \sqrt{x}| < 2^{-p} \cdot 2^{e_\sigma},$$

which gives

$$|t| = |\sigma^2 - x| = |\sigma - \sqrt{x}| \cdot |\sigma + \sqrt{x}| < 2^{-p+e_\sigma} \cdot (2\sigma + 2^{e_\sigma - p}).$$

Notice that 2σ is a floating-point number, and that $\mathrm{ulp}(2\sigma) = 2^{e_\sigma - p + 2}$. Therefore *there is no floating-point number between* 2σ *and* $2\sigma + 2^{e_\sigma - p}$. Hence, since $|t|/2^{-p+e_\sigma}$ is a floating-point number less than $2\sigma + 2^{e_\sigma - p}$, we obtain

$$|t| \leq 2^{-p+e_\sigma+1} \cdot \sigma,$$

implying

$$\left|\frac{t}{2\sigma}\right| \leq 2^{-p+e_\sigma}.$$

Also, since 2^{-p+e_σ} is a floating-point number, the monotonicity of the round-to-nearest function implies

$$\left|\mathrm{RN}\left(\frac{t}{2\sigma}\right)\right| \leq 2^{-p+e_\sigma}.$$

From these two inequalities, we deduce

$$\left|\mathrm{RN}\left(\frac{t}{2\sigma}\right) - \frac{t}{2\sigma}\right| \leq 2^{-2p-1+e_\sigma}. \tag{2}$$

Notice (we will need that in Section IV) that

$$\left|\mathrm{RN}\left(\frac{t}{2\sigma}\right)\right| \leq 2^{-p} \cdot \sigma. \tag{3}$$

Now, define a variable ϵ as

$$\sqrt{x} = \sigma + \frac{t}{2\sigma} + \epsilon,$$

where

$$\begin{aligned}
\epsilon &= \sqrt{x} - \sigma - \frac{t}{2\sigma} \\
&= \frac{t}{\sqrt{x}+\sigma} - \frac{t}{2\sigma} \\
&= t \cdot \frac{2\sigma - (\sqrt{x} - \sigma)}{(\sqrt{x}+\sigma)\cdot 2\sigma} \\
&= -\frac{(\sigma - \sqrt{x})^2}{2\sigma},
\end{aligned}$$

from which we deduce

$$|\epsilon| < \frac{2^{-2p+2e_\sigma}}{2\sigma} \leq 2^{-2p-1+e_\sigma}. \tag{4}$$

By combining (2) and (4), we finally get

$$\left|\left(\sigma + \mathrm{RN}\left(\frac{t}{2\sigma}\right)\right) - \sqrt{x}\right| < 2^{-2p+e_\sigma}.$$

This gives an error in ulps as well as a relative error: since $\mathrm{ulp}(\sigma) = 2^{e_\sigma - p + 1}$ we obtain

$$\left|\left(\sigma + \mathrm{RN}\left(\frac{t}{2\sigma}\right)\right) - \sqrt{x}\right| < 2^{-p-1}\,\mathrm{ulp}(\sigma),$$

and

$$\left|\left(\sigma + \mathrm{RN}\left(\frac{t}{2\sigma}\right)\right) - \sqrt{x}\right| < 2^{-2p} \cdot \sigma.$$

■

Notice that the bound given by Theorem 3 is quite tight. Consider as an example the case $p = 24$ (binary32 precision of the IEEE 754-2008 Standard). Assume x is the floating-point number

$$x = 8402801 \cdot 2^{-23} = 1.00169193744659423828125,$$

then one easily gets

$$\sigma = 8395702 \cdot 2^{-23} = 1.00084567070000732421875,$$

and

$$r = -16749427 \cdot 2^{-48} \approx -5.950591841497 \times 10^{-8},$$

which gives

$$\left|(\sigma + r) - \sqrt{x}\right| = 0.9970011\cdots \times 2^{-48} \times \sigma,$$

to be compared to our bound $2^{-48} \times \sigma$.

Furthermore, the error bounds given by Theorem 3 are *asymptotically optimal*, as we can exhibit a family (for p multiple of 3) of input values parametrized by the precision p, such that for these input values, $|(\sigma + r) - \sqrt{x}|/\sigma$ is asymptotically equivalent to 2^{-2p} as $p \to \infty$. Just consider, for p being a multiple of 6, the input number

$$x = 2^p + 2^{p/3+1} + 2,$$

and for p odd multiple of 3, the input number

$$x = 2^{p-1} + 2^{p/3} + 1.$$

If p is multiple of 6 (the case where p is an odd multiple of 3 is very similar), tedious yet not difficult calculations show that

$$\begin{aligned}
\sqrt{x} &= 2^{p/2} + 2^{-p/6} + 2^{-p/2} - 2^{-1-5p/6} - 2^{-7p/6} \\
&\quad + 3\cdot 2^{-1-11p/6} + \cdots, \\
\sigma &= 2^{p/2} + 2^{-p/6}, \\
t &= 2 - 2^{-p/3}, \\
t/(2\sigma) &= 2^{-p/2} \cdot \left(1 - 2^{-p/3-1} - 2^{-2p/3} + 2^{-p-1}\right. \\
&\quad \left. + 3^{-4p/3} - \cdots\right), \\
r &= 2^{-p/2} \cdot \left(1 - 2^{-p/3-1} - 2^{-2p/3} + 2^{-p}\right), \\
\sigma + r &= 2^{p/2} + 2^{-p/6} + 2^{-p/2} - 2^{-1-5p/6} - 2^{-7p/6} \\
&\quad + 2^{-3p/2},
\end{aligned}$$

so that

$$\sqrt{x} - (\sigma + r) \sim_{p\to\infty} 2^{-3p/2} \cdot \left(-1 + 3 \cdot 2^{-1-p/3}\right),$$

from which we derive

$$|\sqrt{x} - (\sigma + r)| \sim_{p\to\infty} 2^{-p-1}\,\mathrm{ulp}(\sigma),$$

and

$$\left|\frac{\sqrt{x} - (\sigma + r)}{\sigma}\right| \sim_{p\to\infty} 2^{-2p}\left(1 - 3\cdot 2^{-1-p/3}\right),$$

which shows the asymptotic optimality of the bounds given by Theorem 3.

IV. Augmented-precision 2D norms

We suggest two very slightly different algorithms. Algorithm 5 requires three more operations (a Fast2Sum) than Algorithm 4, but it has a slightly better error bound. Again, as for the square-root algorithm, these algorithms derive quite naturally from the Taylor series for the square root: the novelty we believe we bring here is that we provide proven and tight error bounds.

Algorithm 4 (Augmented computation of $\sqrt{x^2+y^2}$)**.**

1: ***if*** $|y| > |x|$ ***then***
2: $\quad$ *swap*(x, y)
3: ***end if***
4: $b_x \leftarrow$ *largest power of* 2 *less than or equal to* x
5: $\hat{x} \leftarrow x/b_x$ {*exact operation*}
6: $\hat{y} \leftarrow y/b_x$ {*exact operation*}
7: $(s_x, \rho_x) \leftarrow \mathrm{2MultFMA}(\hat{x}, \hat{x})$
8: $(s_y, \rho_y) \leftarrow \mathrm{2MultFMA}(\hat{y}, \hat{y})$
9: $(s_h, \rho_s) \leftarrow \mathrm{Fast2Sum}(s_x, s_y)$
10: $s_\ell \leftarrow \mathrm{RN}(\rho_s + \mathrm{RN}(\rho_x + \rho_y))$
11: $r_1 \leftarrow \mathrm{RN}(\sqrt{s_h})$
12: $t \leftarrow s_h - r_1^2$ {*exact operation through an FMA*}
13: $r_2 \leftarrow \mathrm{RN}(t/(2r_1))$
14: $c \leftarrow \mathrm{RN}(s_\ell/(2s_h))$
15: $r_3 \leftarrow \mathrm{RN}(r_2 + r_1 c)$
16: $r_1' \leftarrow r_1 \cdot b_x$ {*exact operation*}
17: $r_3' \leftarrow r_3 \cdot b_x$ {*exact operation*}
18: $(r_h, r_\ell) \leftarrow \mathrm{Fast2Sum}(r_1', r_3')$
19: ***return*** (r_h, r_ℓ)

Algorithm 5 (Slightly more accurate augmented-precision computation of $\sqrt{x^2+y^2}$)**.**

1: ***if*** $|y| > |x|$ ***then***
2: $\quad$ *swap*(x, y)
3: ***end if***
4: $b_x \leftarrow$ *largest power of* 2 *less than or equal to* x
5: $\hat{x} \leftarrow x/b_x$ {*exact operation*}
6: $\hat{y} \leftarrow y/b_x$ {*exact operation*}
7: $(s_x, \rho_x) \leftarrow \mathrm{2MultFMA}(\hat{x}, \hat{x})$
8: $(s_y, \rho_y) \leftarrow \mathrm{2MultFMA}(\hat{y}, \hat{y})$
9: $(s_h, \rho_s) \leftarrow \mathrm{Fast2Sum}(s_x, s_y)$
10: $s_\ell \leftarrow \mathrm{RN}(\rho_s + \mathrm{RN}(\rho_x + \rho_y))$
11: $(s_h', s_\ell') \leftarrow \mathrm{Fast2Sum}(s_h, s_\ell)$
12: $r_1 \leftarrow \mathrm{RN}(\sqrt{s_h'})$
13: $t \leftarrow s_h' - r_1^2$ {*exact operation through an FMA*}
14: $r_2 \leftarrow \mathrm{RN}(t/(2r_1))$
15: $c \leftarrow \mathrm{RN}(s_\ell'/(2s_h'))$
16: $r_3 \leftarrow \mathrm{RN}(r_2 + r_1 c)$
17: $r_1' \leftarrow r_1 \cdot b_x$ {*exact operation*}
18: $r_3' \leftarrow r_3 \cdot b_x$ {*exact operation*}
19: $(r_h, r_\ell) \leftarrow \mathrm{Fast2Sum}(r_1', r_3')$
20: ***return*** (r_h, r_ℓ)

Notice that if one is just interested in getting a very accurate floating-point approximation to $\sqrt{x^2+y^2}$ (that is, if one does not want to compute the error term r_ℓ), then it suffices to replace the last Fast2Sum instruction by "$r_h \leftarrow \mathrm{RN}(r_1' + r_3')$" in both algorithms. Also notice that if the functions `scaleB` and `logB` defined by the IEEE 754-2008 Standard are available and efficiently implemented, one can replace lines 4, 5 and 6 of both algorithms by

$e_x \leftarrow$ `logB`(x)
$\hat{x} \leftarrow$ `scaleB`$(x, -e_x)$
$\hat{y} \leftarrow$ `scaleB`$(y, -e_x)$

and lines 16 and 17 of Algorithm 4, or lines 17 and 18 of Algorithm 5 by

$r_1' \leftarrow$ `scaleB`(r_1, e_x)
$r_3' \leftarrow$ `scaleB`(r_3, e_x).

We have the following result

Theorem 4 (Accuracy of algorithms 4 and 5)**.** *We assume that a radix-2, precision-p (with $p \geq 8$), floating-point arithmetic is used and that there are no underflows or overflows.*

The result (r_h, r_ℓ) returned by Algorithm 4 satisfies

$$r_h + r_\ell = \sqrt{x^2+y^2} + \epsilon,$$

with

$$|\epsilon| < \left(\frac{15}{2}\cdot 2^{-2p} + 30\cdot 2^{-3p}\right)\cdot r_h,$$

and

$$r_\ell \leq \frac{1}{2}\,\mathrm{ulp}(r_h).$$

The result (r_h, r_ℓ) returned by Algorithm 5 satisfies

$$r_h + r_\ell = \sqrt{x^2+y^2} + \epsilon',$$

with

$$|\epsilon'| < \left(\frac{47}{8}\cdot 2^{-2p} + 26\cdot 2^{-3p}\right)\cdot r_h,$$

and

$$r_\ell \leq \frac{1}{2}\,\mathrm{ulp}(r_h).$$

Since the proofs are very similar for both algorithms, we only give the proof for Algorithm 5.

Proof of the error bound for Algorithm 5.

The computations of b_x, $\hat{x}$, and $\hat{y}$ are obviously errorless. We have

$$\begin{aligned}\hat{x}^2 + \hat{y}^2 &= s_x + s_y + \rho_x + \rho_y \\ &= s_h + \rho_s + \rho_x + \rho_y,\end{aligned}$$

with $|\rho_x| \leq 2^{-p} s_x$, $|\rho_y| \leq 2^{-p} s_y$, and $|\rho_s| \leq 2^{-p} s_h$.

We easily find

$$|\rho_x + \rho_y| \leq 2^{-p}(s_x + s_y) = 2^{-p}(s_h + \rho_s)$$

and define $u = \mathrm{RN}(\rho_x + \rho_y)$.

We have $|u| \leq \mathrm{RN}\left(2^{-p}(s_x + s_y)\right)$, so that

$$|u| \leq 2^{-p} s_h$$

and

$$|u - (\rho_x + \rho_y)| \leq 2^{-2p} \cdot s_h. \tag{5}$$

We therefore get

$$|\rho_s + u| \leq 2^{-p+1} \cdot s_h,$$

so that

$$s_\ell \leq 2^{-p+1} \cdot s_h$$

when $p \geq 2$. Also,

$$|s_\ell - (\rho_s + u)| \leq 2^{-2p+1} \cdot s_h.$$

This, combined with (5), gives

$$|s_\ell - (\rho_s + \rho_x + \rho_y)| \leq 3 \cdot 2^{-2p} \cdot s_h,$$

implying that

$$\hat{x}^2 + \hat{y}^2 = s_h + s_\ell + \epsilon_0, \text{ with } |\epsilon_0| \leq 3 \cdot 2^{-2p} \cdot s_h.$$

We also have, $s'_h + s'_\ell = s_h + s_\ell$, $|s'_\ell| \leq 2^{-p} \cdot s'_h$, and $|s_\ell| \leq 2^{-p+1} \cdot s_h$, so that

$$s_h \leq \frac{1 + 2^{-p}}{1 - 2^{-p+1}} \cdot s'_h \leq (1 + 3 \cdot 2^{-p} + 7 \cdot 2^{-2p}) \cdot s'_h,$$

when $p \geq 4$. Which gives

$$\begin{array}{c} \hat{x}^2 + \hat{y}^2 = s'_h + s'_\ell + \epsilon_0, \\ \text{with } |\epsilon_0| \leq (3 \cdot 2^{-2p} + 10 \cdot 2^{-3p}) \cdot s'_h, \end{array} \tag{6}$$

when $p \geq 5$.

Now,

$$\begin{array}{rcl} \sqrt{\hat{x}^2 + \hat{y}^2} & = & \sqrt{s'_h + s'_\ell + \epsilon_0} \\ & = & \sqrt{s'_h} \cdot \left(1 + \frac{s'_\ell + \epsilon_0}{2 s'_h} + \epsilon_1\right), \end{array}$$

with

$$|\epsilon_1| \leq \frac{1}{8} \frac{(s'_\ell + \epsilon_0)^2}{(s'_h)^2} \cdot \left(\frac{1}{1 - \left|\frac{s'_\ell + \epsilon_0}{s'_h}\right|}\right).$$

From the bounds on s'_ℓ and ϵ_0 we get

$$\left|\frac{s'_\ell + \epsilon_0}{s'_h}\right| \leq 2^{-p} + 3 \cdot 2^{-2p} + 10 \cdot 2^{-3p},$$

which gives

$$\begin{array}{rcl} |\epsilon_1| & \leq & \frac{1}{8} \cdot \frac{\left(2^{-p} + 3 \cdot 2^{-2p} + 10 \cdot 2^{-3p}\right)^2}{1 - (2^{-p} + 3 \cdot 2^{-2p} + 10 \cdot 2^{-3p})} \\ & < & 2^{-2p-3} + 2^{-3p}, \end{array}$$

when $p \geq 6$. Hence,

$$\sqrt{\hat{x}^2 + \hat{y}^2} = \sqrt{s'_h} \cdot \left(1 + \frac{s'_\ell}{2 s'_h} + \epsilon_2\right) \tag{7}$$

with

$$|\epsilon_2| \leq |\epsilon_1| + \left|\frac{\epsilon_0}{2 s'_h}\right| < \frac{13}{8} \cdot 2^{-2p} + 6 \cdot 2^{-3p}.$$

In Eq. (7), $\sqrt{s'_h}$ is approximated by $r_1 + r_2$ using Algorithm 3. Therefore, from Theorem 3, we have

$$\sqrt{s'_h} = r_1 + r_2 + \epsilon_3,$$

with

$$|\epsilon_3| < 2^{-2p} \cdot r_1.$$

Since $|s'_\ell / (2 s'_h)| \leq 2^{-p-1}$, so that $|c| \leq 2^{-p-1}$ too, and

$$\left|c - \frac{s'_\ell}{2 s'_h}\right| \leq 2^{-2p-2},$$

hence

$$\sqrt{\hat{x}^2 + \hat{y}^2} = (r_1 + r_2 + \epsilon_3) \cdot (1 + c + \epsilon_4),$$

with

$$|\epsilon_4| \leq \left|c - \frac{s'_\ell}{2 s'_h}\right| + |\epsilon_2| < \frac{15}{8} \cdot 2^{-2p} + 6 \cdot 2^{-3p}.$$

From the bound (3) obtained in the proof of Theorem 3, we have $|r_2| \leq 2^{-p} \cdot |r_1|$. All this gives

$$\sqrt{\hat{x}^2 + \hat{y}^2} = r_1 + r_2 + r_1 c + \epsilon_6,$$

with

$$\begin{array}{rcl} |\epsilon_6| & \leq & |\epsilon_3| + |r_1 \epsilon_4| + |r_2 c| + |r_2 \epsilon_4| + |\epsilon_3| \cdot |1 + c\,\epsilon_4| \\ & \leq & r_1 \cdot \left(\frac{35}{8} \cdot 2^{-2p} + \frac{63}{8} \cdot 2^{-3p} + 6 \cdot 2^{-4p} \right. \\ & & \left. + \frac{15}{16} \cdot 2^{-5p} + 3 \cdot 2^{-6p}\right) \\ & \leq & r_1 \cdot \left(\frac{35}{8} \cdot 2^{-2p} + 8 \cdot 2^{-3p}\right), \end{array}$$

when $p \geq 6$. From the previously obtained bounds on r_2 and c,

$$|r_2 + r_1 c| \leq \frac{3}{2} \cdot 2^{-p} \cdot r_1$$

so that

$$r_3 = r_2 + r_1 c + \epsilon_7,$$

with

$$|\epsilon_7| \leq \frac{3}{2} \cdot 2^{-2p} \cdot r_1,$$

and

$$|r_3| \leq \frac{3}{2} \cdot 2^{-p} \cdot (1 + 2^{-p}) \cdot r_1.$$

We therefore conclude that

$$r_h + r_\ell = r_1 + r_3 = \sqrt{x^2 + y^2} + \epsilon_8,$$

with

$$|\epsilon_8| \le |\epsilon_6| + |\epsilon_7| \le \left(\frac{47}{8}\cdot 2^{-2p} + 8\cdot 2^{-3p}\right)\cdot r_1.$$

From $r_h + r_\ell = r_1 + r_3$, $|r_\ell| \le 2^{-p}|r_h|$, and $|r_3| \le \frac{3}{2}\cdot 2^{-p}\cdot(1+2^{-p})\cdot r_1$, we get

$$\begin{aligned} |r_1| &< \frac{1+2^{-p}}{1-\frac{3}{2}\cdot 2^{-p}\cdot(1+2^{-p})}\cdot|r_h| \\ &\le \left(1+\tfrac{5}{2}\cdot 2^{-p}+\tfrac{11}{2}\cdot 2^{-2p}\right)\cdot|r_h|, \end{aligned}$$

when $p \ge 6$. From this we finally deduce that when $p \ge 8$,

$$|\epsilon_8| \le \left(\frac{47}{8}\cdot 2^{-2p} + 23\cdot 2^{-3p}\right)\cdot|r_h|.$$

■

V. Can we round $\sqrt{x^2+y^2}$ correctly?

In the following we call a *midpoint* a value exactly halfway between consecutive floating-point numbers. In any radix, there are many floating-point values x and y such that $\sqrt{x^2+y^2}$ is a midpoint [21]. A typical example, in the "toy" binary floating-point system of precision $p = 8$ is $x = 253_{10} = 11111101_2$, $y = 204_{10} = 11001100_2$, for which $\sqrt{x^2+y^2} = 325_{10} = 101000101_2$.

If the minimum nonzero distance (in terms of relative distance, or in terms of ulps) between $\sqrt{x^2+y^2}$ and a midpoint is η, then correctly rounding $\sqrt{x^2+y^2}$ can be done as follows:

- approximate $\sqrt{x^2+y^2}$ by some value s with error less than $\eta/2$;
- if s is within $\eta/2$ from a midpoint m, then necessarily $\sqrt{x^2+y^2}$ is exactly equal to that midpoint: we should return $\mathrm{RN}(m)$;
- otherwise, we can safely return $\mathrm{RN}(s)$.

Hence our purpose in this section is to find lower bounds on the distance between $\sqrt{x^2+y^2}$ and a midpoint. Notice that in the special case where x and y have the same exponent, Lang and Muller provide similar bounds in [22].

As previously, we assume a binary floating-point arithmetic of precision p. Let x and y be floating-point numbers. Without loss of generality, we assume $0 < y \le x$. Let e_x and e_y be the exponents of x and y. Define $\delta = e_x - e_y$, so $\delta \ge 0$. We will now consider two cases.

1. If δ is large

First, let us notice that if x is large enough compared to y, our problem becomes very simple. More precisely, we have

$$\begin{aligned} \sqrt{x^2+y^2} &= x\cdot\sqrt{1+\frac{y^2}{x^2}} \\ &= x + \frac{y^2}{2x} + \epsilon, \end{aligned}$$

with

$$-\frac{1}{8}\frac{y^4}{x^3} < \epsilon < 0.$$

When $y \le 2^{-p/2}x$, we have

$$0 < \frac{y^2}{2x} \le 2^{-p-1}x < \frac{1}{2}\,\mathrm{ulp}(x),$$

so that

$$\left|x - \sqrt{x^2+y^2}\right| = \frac{y^2}{2x} + \epsilon < \frac{1}{2}\,\mathrm{ulp}(x).$$

Hence when $y \le 2^{-p/2}x$, $\sqrt{x^2+y^2}$ is far from a midpoint. Furthermore, in such a case, correctly rounding $\sqrt{x^2+y^2}$ is straightforward: it suffices to return x. Notice that $\delta \ge p/2+1$ implies $y \le 2^{-p/2}x$. So, let us now focus on the case $\delta < p/2+1$, i.e. $\delta \le \lfloor (p+1)/2 \rfloor$.

2. If δ is small

Since x and y are floating-point numbers, there exist integers M_x, M_y, e_x, and e_y such that

$$\begin{cases} x = M_x\cdot 2^{e_x-p+1} \\ y = M_y\cdot 2^{e_y-p+1}, \end{cases}$$

with $0 < M_x, M_y \le 2^p - 1$. Assume $\sqrt{x^2+y^2}$ is within ϵ ulps from a midpoint of the form $(M_s+1/2)\cdot 2^{e_s-p+1}$ (with $|\epsilon|$ nonzero and much less than $1/2$). Notice that $x \le s \le \Delta(x\sqrt{2})$, where $\Delta(u)$ means u rounded up, so that e_s is e_x or e_x+1. We have,

$$\sqrt{M_x^2\cdot 2^{2e_x} + M_y^2\cdot 2^{2e_y}} = \left(M_s + \frac{1}{2} + \epsilon\right)\cdot 2^{e_s},$$

which gives

$$\epsilon = 2^{-e_s}\sqrt{M_x^2\cdot 2^{2e_x} + M_y^2\cdot 2^{2e_y}} - \left(M_s + \frac{1}{2}\right).$$

This implies

$$\begin{aligned} \epsilon &= 2^{-e_s}\cdot\frac{2^{2e_y}\left(M_x^2\cdot 2^{2\delta}+M_y^2\right) - 2^{2e_s}\left(M_s^2+M_s+\frac{1}{4}\right)}{2^{e_y}\sqrt{M_x^2\cdot 2^{2\delta}+M_y^2} + 2^{e_s}\left(M_s+\frac{1}{2}\right)} \\ &= 2^{-e_s}\frac{N}{D}. \end{aligned} \tag{8}$$

Now since $M_s \le 2^p - 1$, $2^{e_s}\cdot\left(M_s+\frac{1}{2}\right)$ is less than 2^{p+e_s} and $|\epsilon| < 1/2$, then $2^{e_y}\sqrt{M_x^2\cdot 2^{2\delta}+M_y^2}$ is less than 2^{p+e_s} too, so that the term D in (8) is less than 2^{p+e_s+1}.

Notice that if $e_s = e_x + 1$ we can improve on that bound. In that case,

$$\begin{aligned} \sqrt{M_x^2\cdot 2^{2e_x} + M_y^2\cdot 2^{2e_y}} &< \sqrt{2^{2p} + 2^{2p-2\delta}}\cdot 2^{e_x} \\ &= \tfrac{1}{2}\sqrt{1+2^{-2\delta}}\cdot 2^{p+e_s}, \end{aligned}$$

so that, when $e_s = e_x + 1$,

$$D < \left(\sqrt{1+2^{-2\delta}}\cdot 2^p + 1\right)\cdot 2^{e_s}.$$

(since $|\epsilon| < 1/2$, $\left(M_s+\frac{1}{2}\right)$ is within 2^{e_s-1} from $\sqrt{M_x^2\cdot 2^{2e_x}+M_y^2\cdot 2^{2e_y}}$). Let us now focus on the term N in (8). It is equal to

$$2^{2e_x-2\delta}\left(M_x^2\cdot 2^{2\delta} + M_y^2 - 2^{2(e_s-e_x)+2\delta}\left(M_s^2+M_s+\frac{1}{4}\right)\right),$$

therefore

- if $e_s = e_x + 1$ or $\delta > 0$, then N is an integer multiple of $2^{2e_s-2(e_s-e_x)-2\delta} = 2^{2e_s-2-2\delta}$. Hence, if ϵ is nonzero, its absolute value is at least
$$2^{-e_s} \cdot \frac{2^{2e_s-2-2\delta}}{\left(\sqrt{1+2^{-2\delta}} \cdot 2^p + 1\right) \cdot 2^{e_s}} = \frac{2^{-2-2\delta}}{\sqrt{1+2^{-2\delta}} \cdot 2^p + 1};$$
- if $e_s = e_x$ and $\delta > 0$, then again N is an integer multiple of $2^{2e_s-2(e_s-e_x)-2\delta}$. Hence, if ϵ is nonzero, its absolute value is at least
$$2^{-e_s} \cdot \frac{2^{2e_s-2(e_s-e_x)-2\delta}}{2^{p+e_s+1}} \geq 2^{-p-1-2\delta}.$$
- if $e_s = e_x$ and $\delta = 0$ then
$$N = 2^{2e_s}\left(M_x^2 + M_y^2 - M_s^2 - M_s - \frac{1}{4}\right)$$
is a multiple of $2^{2e_s}/4$, so that if ϵ is nonzero, its absolute value is at least 2^{-p-3}.

To summarize what we have obtained so far in the case "delta is small", whenever $\epsilon \neq 0$, its absolute value is lower-bounded by
$$2^{-p-3}$$
in the case $\delta = 0$; and
$$\min\left\{\frac{2^{-2-2\delta}}{\sqrt{1+2^{-2\delta}} \cdot 2^p + 1}; 2^{-p-1-2\delta}\right\} = \frac{2^{-2-2\delta}}{\sqrt{1+2^{-2\delta}} \cdot 2^p + 1}$$
in the case $\delta > 0$.

Now we can merge the various cases considered above and deduce

Theorem 5. *If x and y are radix-2, precision-p, floating-point numbers, then either $\sqrt{x^2+y^2}$ is a midpoint, or it is at a distance of at least*
$$\frac{2^{-2\lfloor (p+1)/2 \rfloor - 2}}{\sqrt{2} \cdot 2^p + 1} \operatorname{ulp}\left(\sqrt{x^2+y^2}\right)$$
from a midpoint.

When x and y are close, we obtain a much sharper result. For instance, when they are within a factor of 2, δ is equal to 0 or 1, which gives

Theorem 6. *If x and y are radix-2, precision-p, floating-point numbers such that $|x/2| \leq |y| \leq 2 \cdot |x|$, then either $\sqrt{x^2+y^2}$ is a midpoint, or it is at a distance at least*
$$\frac{2^{-p-3}}{\sqrt{5} + 2^{-p+1}} \operatorname{ulp}\left(\sqrt{x^2+y^2}\right)$$
from a midpoint.

Tables I and II compare the actual minimum distance to a midpoint (obtained through exhaustive computation) and the bounds we have obtained in this section, in the case of "toy" floating-point systems of precision $p = 10$ and 15 (an exhaustive search was not possible for significantly wider formats). One can see on these tables that in the cases $\delta = 0$ or $\delta = 1$, our bounds are close to the minimum distance (a consequence is that there is little hope of significantly improving the bound given in Theorem 6), and that for larger values of δ, our bounds remain of the same order of magnitude as the minimum distance.[2]

δ	actual minimum distance to a midpoint	our bound to that distance
0	1.24×10^{-4} ulp	1.22×10^{-4} ulp
1	5.73×10^{-5} ulp	5.45×10^{-5} ulp
2	9.49×10^{-5} ulp	1.48×10^{-5} ulp
3	8.76×10^{-6} ulp	3.78×10^{-6} ulp
4	2.01×10^{-6} ulp	9.51×10^{-7} ulp
5	6.24×10^{-7} ulp	2.38×10^{-7} ulp

Table I
COMPARISON BETWEEN THE ACTUAL MINIMUM DISTANCE TO A MIDPOINT (OBTAINED THROUGH EXHAUSTIVE COMPUTATION) AND THE BOUNDS OBTAINED USING OUR METHOD, IN THE CASE OF A "TOY" FLOATING-POINT SYSTEM OF PRECISION $p = 10$. WHEN $\delta \geq 6$, $\sqrt{x^2+y^2}$ IS NECESSARILY FAR FROM A MIDPOINT.

δ	actual minimum distance to a midpoint	our bound to that distance
0	3.81×10^{-6} ulp	3.81×10^{-6} ulp
1	1.71×10^{-6} ulp	1.70×10^{-6} ulp
2	4.65×10^{-7} ulp	4.62×10^{-7} ulp
3	2.38×10^{-7} ulp	1.18×10^{-7} ulp
4	5.96×10^{-8} ulp	2.97×10^{-8} ulp
5	1.49×10^{-8} ulp	7.44×10^{-9} ulp
6	3.76×10^{-9} ulp	1.86×10^{-9} ulp
7	9.86×10^{-10} ulp	4.56×10^{-10} ulp
8	3.81×10^{-5} ulp	1.16×10^{-10} ulp

Table II
COMPARISON BETWEEN THE ACTUAL MINIMUM DISTANCE TO A MIDPOINT (OBTAINED THROUGH EXHAUSTIVE COMPUTATION) AND THE BOUNDS OBTAINED USING OUR METHOD, IN THE CASE OF A "TOY" FLOATING-POINT SYSTEM OF PRECISION $p = 15$. WHEN $\delta \geq 9$, $\sqrt{x^2+y^2}$ IS NECESSARILY FAR FROM A MIDPOINT.

VI. APPLICATION: CORRECT ROUNDING OF $\sqrt{x^2+y^2}$

Various properties can be deduced from the analyses performed in the paper. Examples are:

[2]Note that in the case $p = 15$, there is a large difference between the actual minimum distance and our bound when $\delta = 8$. This is due to the fact that, when $\delta = 8$, a very large part of the pairs (x, y) fall in the category "$y < 2^{-p/2}x$", for which we have seen that $\sqrt{x^2+y^2}$ is always far from a midpoint.

- we can obtain $\sqrt{x^2+y^2}$ correctly rounded in the binary32 format of the IEEE 754-2008 standard if Algorithm 4 or Algorithm 5 is run in the binary64 format (or a wider format);
- we can obtain $\sqrt{x^2+y^2}$ correctly rounded in the binary64 format of the IEEE 754-2008 standard if Algorithm 4 or Algorithm 5 is run in the binary128 format;
- if $|x/2| \leq |y| \leq |2x|$, we can obtain $\sqrt{x^2+y^2}$ correctly rounded in the binary64 format of the IEEE 754-2008 standard if Algorithm 4 or Algorithm 5 is run in the INTEL binary80 format.

CONCLUSION

We have given a very tight error bound for a simple augmented-precision algorithm for the square root. We have also introduced two slightly different augmented-precision algorithms for computing $\sqrt{x^2+y^2}$. Then, we have given bounds on the distance between $\sqrt{x^2+y^2}$ and a midpoint, where x and y are floating-point numbers and $\sqrt{x^2+y^2}$ is not a midpoint. These bounds can be used to provide correctly-rounded 2D-norms (either using one of our algorithms, or another one).

REFERENCES

[1] W. Kahan, "Pracniques: further remarks on reducing truncation errors," *Commun. ACM*, vol. 8, no. 1, p. 40, 1965.

[2] I. Babuška, "Numerical stability in mathematical analysis," in *Proceedings of the 1968 IFIP Congress*, vol. 1, 1969, pp. 11–23.

[3] D. M. Priest, "Algorithms for arbitrary precision floating point arithmetic," in *Proceedings of the 10th IEEE Symposium on Computer Arithmetic (Arith-10)*, P. Kornerup and D. W. Matula, Eds. IEEE Computer Society Press, Los Alamitos, CA, Jun. 1991, pp. 132–144.

[4] ——, "On properties of floating-point arithmetics: Numerical stability and the cost of accurate computations," Ph.D. dissertation, University of California at Berkeley, 1992.

[5] M. Pichat, "Correction d'une somme en arithmétique à virgule flottante," *Numerische Mathematik*, vol. 19, pp. 400–406, 1972, in French.

[6] S. M. Rump, T. Ogita, and S. Oishi, "Accurate floating-point summation part I: Faithful rounding," *SIAM Journal on Scientific Computing*, vol. 31, no. 1, pp. 189–224, 2008. [Online]. Available: http://link.aip.org/link/?SCE/31/189/1

[7] T. Ogita, S. M. Rump, and S. Oishi, "Accurate sum and dot product," *SIAM Journal on Scientific Computing*, vol. 26, no. 6, pp. 1955–1988, 2005.

[8] S. Graillat, P. Langlois, and N. Louvet, "Algorithms for accurate, validated and fast computations with polynomials," *Japan Journal of Industrial and Applied Mathematics*, vol. 26, no. 2, pp. 215–231, 2009.

[9] J. R. Shewchuk, "Adaptive precision floating-point arithmetic and fast robust geometric predicates," *Discrete Computational Geometry*, vol. 18, pp. 305–363, 1997. [Online]. Available: http://link.springer.de/link/service/journals/00454/papers97/18n3p305.pdf

[10] Y. Hida, X. S. Li, and D. H. Bailey, "Algorithms for quad-double precision floating-point arithmetic," in *Proceedings of the 15th IEEE Symposium on Computer Arithmetic (ARITH-16)*, N. Burgess and L. Ciminiera, Eds., Vail, CO, Jun. 2001, pp. 155–162.

[11] K. Briggs, "The `doubledouble` library," 1998, available at http://www.boutell.com/fracster-src/doubledouble/doubledouble.html.

[12] P. Friedland, "Algorithm 312: Absolute value and square root of a complex number," *Communications of the ACM*, vol. 10, no. 10, p. 665, Oct. 1967.

[13] P. Midy and Y. Yakovlev, "Computing some elementary functions of a complex variable," *Mathematics and Computers in Simulation*, vol. 33, pp. 33–49, 1991.

[14] IEEE Computer Society, *IEEE Standard for Floating-Point Arithmetic*. IEEE Standard 754-2008, Aug. 2008, available at http://ieeexplore.ieee.org/servlet/opac?punumber=4610933.

[15] T. E. Hull, T. F. Fairgrieve, and P. T. P. Tang, "Implementing complex elementary functions using exception handling," *ACM Transactions on Mathematical Software*, vol. 20, no. 2, pp. 215–244, Jun. 1994.

[16] T. J. Dekker, "A floating-point technique for extending the available precision," *Numerische Mathematik*, vol. 18, no. 3, pp. 224–242, 1971.

[17] P. W. Markstein, "Computation of elementary functions on the IBM RISC System/6000 processor," *IBM Journal of Research and Development*, vol. 34, no. 1, pp. 111–119, Jan. 1990.

[18] S. Boldo and M. Daumas, "Representable correcting terms for possibly underflowing floating point operations," in *Proceedings of the 16th Symposium on Computer Arithmetic*, J.-C. Bajard and M. Schulte, Eds. IEEE Computer Society Press, Los Alamitos, CA, 2003, pp. 79–86. [Online]. Available: http://perso.ens-lyon.fr/marc.daumas/SoftArith/BolDau03a.pdf

[19] J.-M. Muller, N. Brisebarre, F. de Dinechin, C.-P. Jeannerod, V. Lefèvre, G. Melquiond, N. Revol, D. Stehlé, and S. Torres, *Handbook of Floating-Point Arithmetic*. Birkhäuser Boston, 2010, ACM G.1.0; G.1.2; G.4; B.2.0; B.2.4; F.2.1., ISBN 978-0-8176-4704-9.

[20] P. Markstein, *IA-64 and Elementary Functions: Speed and Precision*, ser. Hewlett-Packard Professional Books. Prentice-Hall, Englewood Cliffs, NJ, 2000.

[21] C.-P. Jeannerod, N. Louvet, J.-M. Muller, and A. Panhaleux, "Midpoints and exact points of some algebraic functions in floating-point arithmetic," *IEEE Transactions on Computers*, vol. 60, no. 2, Feb. 2011.

[22] T. Lang and J.-M. Muller, "Bound on run of zeros and ones for algebraic functions," in *Proceedings of the 15th IEEE Symposium on Computer Arithmetic (ARITH-16)*, N. Burgess and L. Ciminiera, Eds., Jun. 2001, pp. 13–20.