Exact Floating Point

Alan A. Jorgensen and Andrew C. Masters

1 Introduction: IEEE Standard Floating Point

Floating point was used for representing and operating on real numbers in computers starting with the Zuse Z4 computer in 1942. But there was no standard. At the instigation of Professor Emeritus William Morton Kahan, the first standard for floating point, IEEE 754, was published in 1985 (now identified as IEEE 754-1985) by the Institute of Electrical and Electronics Engineers (IEEE). The current version of the floating-point standard is ISO/IEC/IEEE 60559 [1].

To represent real numbers, standard floating point uses a data structure based on scientific notation, as shown in Fig. 1. This standard floating-point format includes representations for the sign (S), the exponent (E), and the fraction (T).

The sign S is a single bit representing the sign of the value represented, the exponent E is the offset exponent of length e, and the fraction T is the significand of length t. The length k is the overall length of the representation. The real number encoded by this formulaic representation is, in most cases, an approximation, which introduces error. Amplification of this error causes concern about the accuracy of the final result.

The IEEE floating-point standard defines "precision" as "the maximum number, p^{SFP}, of significant digits that can be represented in a format, or the number of digits to that [sic] a result is rounded" [1]. Using the IEEE standard floating-point definition of p^{SFP}, in binary format $p = t + 1$ because of the hidden bit. Instead of merely identifying the number of significant digits that can be represented, bounded floating point provides an enhanced p^{BFP} that represents the actual number of digits (bits) that are significant (have meaning), and the new variable D will represent

A. A. Jorgensen (✉) · A. C. Masters
True North Floating Point, Las Vegas, NV, USA
e-mail: aaj@truenorthfloatingpoint.com; andrew@truenorthfloatingpoint.com

H. R. Arabnia et al. (eds.), *Advances in Software Engineering, Education, and e-Learning*, Transactions on Computational Science and Computational Intelligence, https://doi.org/10.1007/978-3-030-70873-3_26

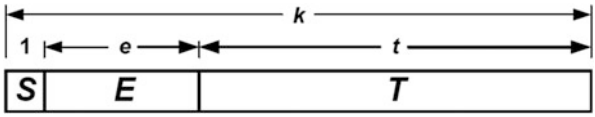

Fig. 1 Standard floating-point formatnumber of bits in the significand

the number of digits (bits) of the representation that are NOT significant, where $D = t + 1 - p^{\mathrm{SFP}}$. In IEEE standard representation, the value of D is not known, nor is the precision, p^{BFP}, the actual number of significant bits.

The value of the standard representation is shown in (1).

$$-1^{S} \bullet \left(1 + T/2^{t}\right) \bullet 2^{E-O} \text{ or } -1^{S} \bullet \left(\left(T + 2^{t}\right)/2^{t}\right) \bullet 2^{E-O} \tag{1}$$

where S, T, t, and E are defined above and O is the exponent offset. Offset O is nominally $2^{e\text{-}1}$–1.

Most real values (the results from floating-point operations) cannot be represented exactly in standard floating point [2] nor in any fixed number of digits. In bounded floating point, the value is defined to be represented "exactly" when the error between the real value and the floating-point representation is less than ½ units in the last place (ulps) as implied in [2]. In other words, for a given p^{SFP}, there is no other floating-point representation which is closer to the real value.

However, IEEE standard floating point has no mechanism for indicating that the representation of a value is exact. Thus, when only using standard floating point, there is no intrinsic means at present of determining the accuracy of a standard floating-point result. But knowing that a computation is sufficiently correct is important and sometimes vital, particularly in large complex critical computations like weather forecast modeling and other predictive modeling.

Bounded floating point provides that knowledge. Bounded floating point answers the questions that standard floating point alone cannot, such as:

- Is the result "exact"?
- How many significant digits are there in an inexact result?
- Is the result sufficiently accurate?
- Is the result precisely zero?

2 Bounded Floating Point

Recently issued US patents on bounded floating point define a mechanism that calculates and saves the range of error associated with a standard floating-point value [3, 4]. As shown in Fig. 2, bounded floating point extends the standard floating-point representation by adding an error information field identified as the "bound" field, B. Various sizes of formats are selected by the determination of the various field widths where "k" is the total format size or floating-point word length. For example, in 80-bit bounded floating point, $k = 80$.

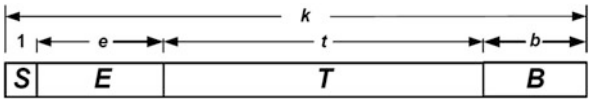

Fig. 2 Bounded floating-point format

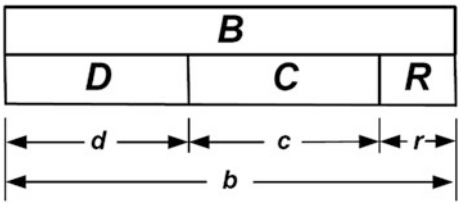

Fig. 3 Format of the bound field *B*

The bounded floating-point system, implemented in hardware, software, or a combination of the two, calculates and saves the range of error associated with a standard floating-point value, thus retaining and calculating the number of significant digits. It does this by using the bound field *B*.

The bound field *B* contains subfields (*D*, *C*, *R*, see Fig. 3) to retain error information provided by prior operations on the represented value, but the field of primary importance is the "lost bits" field, *D*. This field identifies the number of bits in the represented value that are no longer significant. If the value is exact, the value of the *D* field will be zero.

With the exception of zero detection, bounded floating point retains the exception features of standard floating point. Bounded floating point exactly identifies zero when the significant remaining bits are all zero.

Bounded floating point provides a means of identifying the required number of significant bits (or decimal digits) required in a bounded floating-point calculation. (A default value may be used for the required number of significant bits, or the programmer may specify the required number.) When a result lacks this required number of significant bits, bounded floating point identifies it. A result that lacks the required number of significant bits is represented with the "quiet" not-a-number representation, "qNaN.sig," indicating excessive loss of significance.

Under program control, bounded floating point will provide a "signaling" not-a-number representation, "sNaN.sig," when a specified bounded floating-point value does not meet a specified precision requirement. Upon initiation of this command, a specific result is tested to verify that it has the required number of significant digits.

The interval defined by bounded floating point is given by (2) as follows:

$$-1^S \bullet \left(\left(T + 2^t\right)/2^t\right) \bullet 2^{E-O} \cdots -1^S \bullet \left(\left(T + 2^t + 2^{D-1}\right)/2^t\right) \bullet 2^{E-O} \tag{2}$$

This is the same as standard floating point except that the term 2^{D-1} provides the upper bound, where *D* is the logarithm of the number of bits that are no longer significant. Importantly, when *D* is zero, this indicates that there are no insignificant digits; the bound is 1/2 ulp (2^{D-1} when $D = 0$ is 1/2) and the value of the error is less than or equal to the bound, which is the definition of an "exact" representation.

In Fig. 3, the R field of the bound is the summation of the most significant bits lost during the final truncation and is functionally equivalent to the guard and rounding bits of the standard floating-point operations. The equivalent of the "sticky bit" occurs when the remainder of the truncated bits is not zero so that one is added to the value of R, the rounding error field of the bound field. Addition to the R field carries into the C field, which is the rounding error count in units-in-the-last-place (ulps).

Range information includes the number of bits in the representation that are no longer of value (insignificant) and, therefore, are referred to as the "lost bits" (D). The number of lost bits is the logarithm of the upper bound (furthest from zero) of the error in the value represented. The lost bits include the accumulated contributions from both cancellation and rounding errors.

When the based two logarithm of the resulting sum of the rounding error, C, is greater than or equal to the resulting lost bits, D, the lost bits, D, is increased by one, and the rounding error sum, C, is set to zero. Carries out of the C field are added to the D field.

This calculation provides a worst-case interval in which the real value represented exists. The actual precision, p^{BFP}, of the value represented is no greater than $t + 1\text{-}D$. When the value of the D field for a represented value is zero, then the representation is "exact" as defined above.

The truncated floating-point value (round to zero) is the lower bound, and the upper bound is determined by the addition of the error determined by the lost bits, D (the real value represented), $V \in \mathbb{R}$, is absolutely contained in the interval of (2).

The midpoint is determined by (3), as follows:

$$-1^{S} \bullet \left(\left(\left(T + 2^{t} + 2^{D-2} \right) / 2t \right) \bullet 2^{E-O} \right. \tag{3}$$

Bounded floating point allows accuracy of the source of real values, measured or entered, to be specified. External data sources provide data with intrinsic error; for example, keyboard data entry with a limited precision input field or an industrial sensor that provides fewer significant bits than that required by the precision of the floating-point format in use.

According to Ashenhurst and Metropolis [5]:

> It is convenient, and by now more or less traditional, to distinguish three sources of error, designated generated, inherent and analytic. Generated error reflects inaccuracies due to the necessity of rounding or otherwise truncating the numeric results of arithmetic operations, inherent error reflects inaccuracies in initially given arguments and parameters, and analytic error reflects inaccuracies due to the use of a computing procedure which calculates only an approximation to the theoretical result desired.

Bounded floating point permits the representation of inherent error (inaccuracies in input parameters). If the number of significant digits in the value provided is limited, then bounded floating point can accurately represent that number by subtracting the number of bits required to represent that number from p^{SFP} to obtain D (the number of lost bits), which are then carried throughout the calculation.

Bounded floating point can manage generated error by calculating the number of bits that are lost due to "the necessity of rounding or otherwise truncating the numeric results of arithmetic operations."

Additionally, bounded floating point can be used in conjunction with implementations of the current floating-point standard. Conversion between the two formats can be accomplished when needed, which allows continued use of existing software that is dependent upon the current floating-point standard. However, error information will be lost when converting from bounded floating point to standard floating point.

3 Similar Floating-Point Numbers

Catastrophic cancellation occurs when subtracting similar numbers when error already exists [2, 6, p. 124, 7, 8, pp., 10–11, 9].

"Similar numbers" can be defined by (4) that describes the loss of significant digits, as suggested by [10].

$$\begin{aligned} D = \mathrm{Log}_2(z); \text{ iff } V \bullet z/(z+1) > M/S \\ \geq V \bullet (z-1)/z \text{ and } z \subset \left\{2^i, i = 3..p^{\mathrm{SFP}}\right\} \end{aligned} \tag{4}$$

where D is the resulting number of insignificant (lost) bits, M is the minuend, S is the subtrahend, V is the represented floating-point value ($V \in \mathbb{R}$), p^{SFP} is the number of bits in the significand including the hidden bit, and M or S is inexact. Note that for n less than 3, when guard digits are applied, the result will be "exact" [8, pp. 48–50, 11, p. J23].

The error, as represented by bounded floating point, due to the cancellation is no greater than 2^{D-1}.

Bounded floating point uses the value of D to determination the number of significant digits of a value. This is done by taking the value of p^{SFP}, which identifies the highest number of significant digits that can be represented in a format, and subtracting D, which identifies the number of insignificant or lost bits. The result, which is the enhanced p^{BFP}, of the subtraction establishes the number of significant digits in an "exact" or inexact result.

Also, by using D a determination can be made as to whether the result is sufficiently accurate. The required number of significant digits is known (either by use of a default value or by programming a number required). If the result p^{BFP} (the number of actual significant digits) is less than the required number of significant digits, the number is inexact. If the resulting p^{BFP} is equal to or greater than the required number of significant digits, the number is sufficiently accurate.

Another advantage of having the value of D known and available for use is that a determination can be made as to whether a result is precisely zero. Knowing the number of significant digits in a number allows bounded floating point to compare

the number of digits that are significant against the number of leading zeros in the result. If the significant digits of the result are all zero, the result is determined to be significantly zero. This is in contrast to standard floating point, in which all digits of the result must be zero.

Consequently, bounded floating point enables a determination of the "exactness" of a value, discloses the actual number of significant bits, and ascertains when a result is precisely zero, none of which are available without using bounded floating point.

4 Exact and Inexact Subtraction

Subtraction is "exact" when the subtrahend and minuend have no rounding error, as stated by Goldberg in "What Every Computer Scientist Should Know About Floating-Point Arithmetic" [11]. However, when inexact, but similar, values are subtracted, rounding error will cause catastrophic cancellation with a corresponding loss of significant digits , [8 , p. 11, 11].

Table 1 shows subtraction of similar values and demonstrates catastrophic cancellation, which occurs when the two values (minuend A and subtrahend B) to be subtracted are similar as defined in (4).

For a test case we have selected $A - B$ where $A = 10{,}000{,}000{,}000 \bullet \pi$ (scaled for ease of representation of the result), selected $z = 4{,}294{,}967{,}296$ (2^{32}), and used $B = A \bullet (z\text{-}1)/z$ from (4). The selection of 2^{32} indicates that there are 32 lost bits in this example.

To assure that no error was introduced by standard floating point, Table 1 provides these values as computed by Mark Mason's High Precision Calculator that was set for "High Precision" [12, 13], which provides a surplus number of digits (not the limited number of digits of the 64-bit or 128-bit floating point) for the calculations. Consequently, the values shown are "exact" values.

Using the example in Table 1, the value of B, which is 31415926**28.583341988**... is subtracted from the value of A, which is 31415926**35.897932384**... We know that standard floating-point calculations are constrained to a limited number of digits. If we consider this calculation as limited to nine decimal digits, when the

Table 1 Subtraction of similar values

High precision results
$A = 10^{10} \bullet \pi$ = 31415926**35.897932384626433832795028841971693993 75**
$z = 2^{32}$ = 4,294,967,296,496,729
$(z\text{-}1)/z$ = 0.99999999976716935634613037109 37
$B = A \bullet (z\text{-}1)/z$ = 314159265 **28.583341988290635427538339820422732508487179729515919461846351623535 1562**
$A\text{-}B$ = **7.31459039**633579840525668902154896148526382027048408053815364837646484 37

subtraction is performed, the first nine digits will all be zero (will cancel out), leaving no significant digits – clearly showing catastrophic cancellation.

This potential for the lack of significant digits in standard floating point is not new. It has been known from at least 1952 when an early floating-point patent [14] by IBM explicitly stated "... under some conditions, the major portion of the significant data digits may lie beyond the capacity of the registers. Therefore, the result obtained may have little meaning if not totally erroneous." Bounded floating point can be used to clearly identify when there is a lack of significant digits.

Table 2 presents the comparison of "exact" and inexact values differing by one binary ulp calculated with 64-bit standard floating point and 128-bit standard floating point.

The first row shows that the decimal representation of the value of A, using 64-bit floating point, is 31415926535.897930, while the second row shows that after adding only one ulp of error to A the decimal equivalent of $A + 1$ is 31415926535.897934.

The second and third rows show the decimal equivalents of A and $A + 1$ using 128-bit floating point.

Table 2 shows the effect of even a very small error of only one ulp, which creates an inexact value. When similar values are subtracted, cancellation [2] occurs, and the one-bit error is multiplied exponentially. This is a standard floating point hidden and unknown error, but this error is revealed in bounded floating point.

Table 3 demonstrates "exact" and inexact calculations in 64-bit, 128-bit, and 80-bit bounded floating-point calculations. This table illustrates the results of adding a one ulp error injected into the "A" values by adding one to the significand field (T) of the 64-bit and 128-bit standard floating-point values and adding one to the lost bits field (D) of the bounded floating-point value. Overflow is avoided by the selection of test values.

Table 2 High precision similar values calculation

Represented value	Decimal representation of value
64-bit FP A	31415926535.897930
64-bit FP $A + 1$ ulp	31415926535.897934
128-bit FP A	31415926535.8979323846264338327950**28075364135510**
128-bit FP $A + 1$ ulp	31415926535.8979323846264338327950**31384086585722**

Table 3 Exact and error-injected values – 64-bit results

Exact 64-bit result	7.31459045410156250
Inexact 64-bit result	7.31459**426879882810**
Exact 128-bit result	7.3145903963357984052566879720382048 02
Inexact 128-bit result	7.3145903963357984052566**91280760655014**
Exact BFP 80-bit result	7.314590454101562
Inexact BFP 80-bit result	7.314590

Tests were performed using IEEE standard 64-bit floating point, 128-bit standard floating point, and 80-bit bounded floating point (BFP), as seen in Table 3. The 80-bit bounded floating-point model consists of S, E, and T, which are identical to 64-bit standard floating point with a 16-bit bound field, where $d = 6$, $c = 6$, and $r = 4$. These values are chosen so that the total width, b, is a multiple of 8 bits and d (the length of the lost bits field) satisfies $d > \log_2 (t + 1)$ to ensure that a loss of all significant bits can be represented. The value for c (the accumulated rounding error in ulps) is chosen such that exponential growth rate of the loss of significant bits due to rounding error will not exceed 2^n where $n = 1/2^c$, or, in this case, 1/64. The width r of the rounding error field R is chosen to round the width b of the bound field, B, up to the nearest multiple of 8-bits.

The bold and underlined digits of the 64-bit inexact results are those digits that differ from the same digit positions of the "exact" 64-bit calculation. Similarly, the bold and underlined digits of the 128-bit inexact result differ from the "exact" 128-bit calculation. Table 3 makes it easy to see the difference in "exact" and inexact values. And when similar numbers with inexact values (such as may arise from error in earlier calculations or error from input with limited significant digits) are subtracted using floating point, these errors can multiply exponentially due to catastrophic cancellation.

5 Conclusions

Floating-point cancellation errors that occur during subtract operations on inexact operands are detectable and measurable under bounded floating point though they are invisible in IEEE standard floating-point results.

Bounded floating point answers the following questions that standard floating point cannot:

- Is the result "exact"?
  - An "exact" floating-point result, defined as a result that has error within + or - ½ units in the last place (ulps), is shown by using the value of D.
- How many significant digits are there in an inexact result?
  - The number of digits known to be insignificant, D, is subtracted from the possible number of significant digits, which is known from p^{SFP}.
- Is the result sufficiently accurate?
  - The number of significant digits needed is merely compared to the number of significant digits that is known by use of bounded floating point.
- Is the result precisely zero?
  - When bounded floating point determines the number of significant digits of the result is all zero, then the result is significantly zero.

Thus, bounded floating point precisely defines whether the real value represented is "exact" for all digits provided and, if not, defines the number of significant digits. And bounded floating point provides notification if the result is not significantly accurate.

The bounded floating-point methodology provides greater assurance that complex mission critical computations provide results sufficient to successfully complete that mission. Therefore, it is recommended that bounded floating point should be required for all mission critical systems to avoid catastrophic failures due to accumulated floating-point error.

References

1. ISO/IEC/IEEE 60559, *Information Technology – Microprocessor Systems – Floating-Point Arithmetic* (Institute of Electrical and Electronics Engineers, Piscataway, 2011)
2. D. Goldberg, What every computer scientist should know about floating-point arithmetic. ACM Comput. Surv. **23**(1), 5–48 (1991)
3. A. A. Jorgensen, Apparatus for calculating and retaining a bound on error during floating point operations and methods thereof. US Patent No. 9,817,662, 14 Nov 2017
4. A. A. Jorgensen, Apparatus for calculating and retaining a bound on error during floating point operations and methods thereof. US Patent No. 10,540,143, 21 Jan 2020
5. R.L. Ashenhurst, N. Metropolis, Error estimation in computer calculation. Am Math Monthly, Part 2: Comp Comp **72**(2), 47–58 (1965)
6. J.-M. Muller, F. de Dinechin, C.P. Jeannerod, V. Lefevre, G. Melquiond, N. Revo, D. Stehle, S. Torres, *Handbook of Floating-Point Arithmetic* (Birkhauser, Boston, 2010)
7. N.J. Higham, *Accuracy and Stability of Numerical Algorithms* (SIAM, Philadelphia, 1996), p. vii-xxviii, 1–688
8. W. M. Kahan, A Logarithm Too Clever by Half, 2004. [Online]. Available: http://people.eecs.berkeley.edu/~wkahan/LOG10HAF.TXT. Accessed 26 Feb 2019
9. W.E. Ferguson Jr., Exact computation of a sum or difference with applications to argument reduction, in *Proceedings of the 12th IEEE Symposium on Computer Arithmetic*, (Bath, 1995)
10. W.M. Kahan, Desperately needed remedies for the undebuggability of large floating-point computations in science and engineering, in *IFIP/SIAM/NIST Working Conference on Uncertainty Quantification in Scientific Computing*, (Boulder, 2011)
11. D. Goldberg, What every computer scientist should know about floating-point arithmetic. ACM Comput. Surv. **23**(1), 5–47 (1991)
12. A.A. Jorgensen, A. Masters, R. Guha, Assurance of accuracy in floating-point calculations – A software model study, in *2019 International Conference on Computational Science and Computational Intelligence (CSCI)*, (Las Vegas, 2019)
13. N. M. Mason, High Precision Calculator – Freeware, 2017. [Online]. Available: http://www.markmason.net/hpc/index.htm. Accessed 28 Feb 2020
14. H. M. Sierra, Floating decimal point arithmetic control means for calculator, United States Patent 3,037,701, 5 June 1962