# Introduction to the IEEE 1788-2015 Standard for Interval Arithmetic

Nathalie Revol

# Introduction to the IEEE 1788-2015 Standard for Interval Arithmetic

Nathalie Revol (0000-0002-2503-2274)

INRIA – LIP, ENS de Lyon, University of Lyon
46 allée d'Italie, 69364 Lyon Cedex 07, France

**Abstract.** Interval arithmetic is a tool of choice for numerical software verification, as every result computed using this arithmetic is self-verified: every result is an interval that is guaranteed to contain the exact numerical values, regardless of uncertainty or roundoff errors.
From 2008 to 2015, interval arithmetic underwent a standardization effort, resulting in the IEEE 1788-2015 standard. The main features of this standard are developed: the structure into levels, from the mathematic model to the implementation on computers; the possibility to accommodate different mathematical models, called flavors; the decoration system that keeps track of relevant events during the course of a calculation; the exact dot product for point (as opposed to interval) vectors.

**Keywords:** Interval arithmetic · Standardization · IEEE 1788-2015

This work has been presented at NSV 2017: 10th International Workshop on Numerical Software Verification, a workshop of CAV 2017 (Heidelberg, Germany): 29th International Conference on Computer Aided Verification. The final publication is available at Springer via http://dx.doi.org/10.1007/978-3-319-63501-9.

## 1 Introduction

Interval arithmetic is a valuable tool to perform self-validated numerical computations. Indeed, every calculation is performed using intervals as inputs, which are assumed to enclose the exact value of the considered inputs, and it returns intervals as outputs, which are guaranteed to enclose the exact value of the corresponding outputs. This is the most precious feature of interval arithmetic and it is called the *Fundamental Theorem of Interval Arithmetic*, abbreviated in *FTIA* in the following. It sometimes bears different names, depending on the authors. Let us quote here some big names in interval arithmetic, at least some who published their work in English: Hansen and Walster [3], Kulisch [9], Moore [11, 12], Moore again, with Kearfott and Cloud [13], Neumaier [15], Rump [18], Tucker [20] to cite only a few, in alphabetical order.

Other precious features are best exemplified on Newton's method for the determination of the zeros of a function $f$ on a given interval. First, the use of interval arithmetic permits an effective application of Brouwer theorem: when the new iterate, by Newton's method, is included in the previous one, then

this iterate contains a zero for $f$. Second, the use of an ad-hoc definition of the division makes it possible to separate zeros: in such a case, one iteration of Newton's method produces two disjoint intervals and it is guaranteed that no zero for $f$ lies in the gap between these two intervals. Quite often, each of these two disjoint new iterates contains a (strict) subset of the zeros.

The community of users of interval arithmetic was willing to preserve these precious features. However, libraries implementing interval arithmetic and interval methods usually laid on different variations on interval arithmetic, on different definitions. Examples are on the one hand the `filib++` library [10], rather close to the set-based flavor, and on the other hand the MPFI library [17], closer to the IEEE 754-2008 standard for floating-point arithmetic [6]. It was thus not possible to build a common basis of programs, test cases and benchmarks. It was then collectively decided upon standardizing interval arithmetic, in order to share the possibilities offered by interval arithmetic. The standardization effort was led under the auspices of IEEE, from October 2008 to July 2015. The bulk of the work was done via e-mail. The standardization effort proceeded via so-called "Motions", or proposals that constituted steps forward. Each motion underwent 3 weeks of discussion and possible amendments, followed by 3 weeks of vote. John Pryce, technical editor, created what is now the text of the standard, using the results of the votes as raw material. The content of the standard is developed now.

## 2 The Big Picture: Structuration into Levels

It was decided to start from the mathematical level, to have a clear and clean basis, and then to investigate and specify how the mathematical notions translate to the implementation levels. The structuration into levels is borrowed from the IEEE 754-2008 Standard for Floating-Point Arithmetic [6].

The first level is the *mathematical level*: it specifies what an interval on real numbers is, what operations on intervals do and so on.

Level 2 deals with the *discretization* of Level 1 intervals, so that they can be implemented on a computer: it defines

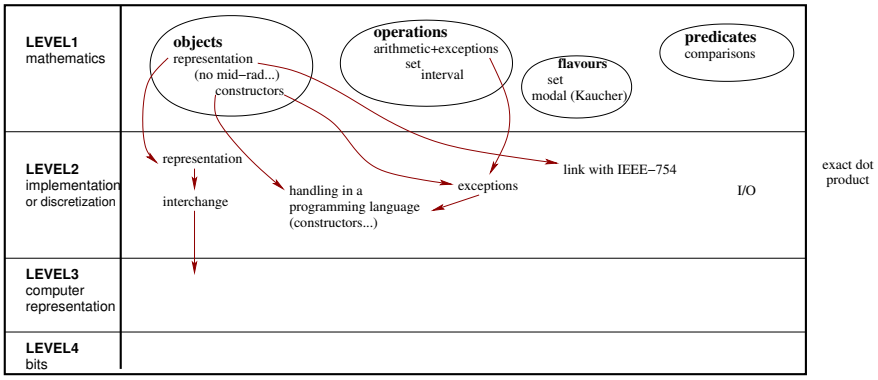
- *interval datums*, that are the representations of entities of Level 1;
- *interval types*, that are finite sets of interval datums.

The issues handled at that level are related to the fact that a continuous world is implemented using a discrete and finite environment.

"*Level 3 is about* representation *of interval datums – usually but not necessarily in terms of floating-point values*" [7]. The concern here is about the representation of an interval datum, e.g., by its endpoints, and the type of the numeric values used for this representation.

Level 4 is about the *encoding* of these representations. It is really the equivalent of Level 4 for floating-point arithmetic. However, as numerical values are not the subject of this standard, very few in IEEE 1788-2015 is said about Level 4.

The figure summarizes the structuration in levels, the content of each level and the relation betwen levels. Each topic is detailed below.




# 3 Flavors

As it has already been noted, the starting point of this work was a sound definition of interval arithmetic at the mathematical level. Everybody agrees on the meaning of $[1,2]+[3,5]$ but unfortunately no more on the meaning of $[1,+\infty)$, $[4,6]/[0,1]$ or $[2,1]$. Each of these intervals or expressions has a meaning in a specific mathematical model: set theory, Kaucher arithmetic, cset theory, modal arithmetic... but not in all of them, or not the same.

A first choice in the standard is thus to separate what has a common definition in every model, without controversy, from the rest. The standard is designed to accommodate smoothly different models, called *flavors* in the standard, as long as they coincide on common definitions. The standard can thus be seen as a common part and as providing "hooks" where different mathematical models can find a place.

## 3.1 Common intervals and operations

The common definitions and requirements are given in Clause 9 of the standard, entitled "*Operations and related items defined in all flavors*". But first, let us give the definition of a common interval: "The common intervals are defined to be the set $\mathbb{IR}$ of nonempty closed bounded real intervals." [7, Clause 7.2]. Common arithmetic operations include, if $\boldsymbol{x}$ and $\boldsymbol{y}$ denote common intervals: $-\boldsymbol{x}$, $\boldsymbol{x}+\boldsymbol{y}$, $\boldsymbol{x}-\boldsymbol{y}$, $\boldsymbol{x}\boldsymbol{y}$, $\boldsymbol{x}/\boldsymbol{y}$ if $\boldsymbol{y}$ does not contain 0, $\boldsymbol{x}^2$, $\sqrt{\boldsymbol{x}}$ if $\boldsymbol{x} \geq 0$, exponential, logarithmic and trigonometric functions on their domain, and a few more.

Two other common operations (even if they sound less than common to you) are cancellative addition and subtraction, that are the reciprocal of addition and subtraction respectively. If $\boldsymbol{x} = [\underline{x}, \bar{x}]$ and $\boldsymbol{y} = [\underline{y}, \bar{y}]$ are two common intervals such that $\bar{x} - \underline{x} \geq \bar{y} - \underline{y}$, then `cancelMinus`$(\boldsymbol{x}, \boldsymbol{y})$ returns the unique interval $\boldsymbol{z}$ such that $\boldsymbol{y} + \boldsymbol{z} = \boldsymbol{x}$, with formula $\boldsymbol{z} = [\underline{x} - \underline{y}, \bar{x} - \bar{y}]$. The operation `cancelPlus` is equivalent to `cancelMinus`$(\boldsymbol{x}, -\boldsymbol{y})$.

The set operations of intersection and convex hull of the union of two intervals are also common operations, as well as some operations specific to intervals, such as the left endpoint or the width of an interval.

### 3.2 Set-based flavor and other ones

Regarding flavors, it happens that, after many discussions regarding several mathematical models, the only fully developed flavor, and thus the only flavor included in the current version of the standard, is the flavor based on set theory; it is called the *set-based flavor*.

The set-based flavor is described in details in the second part of the standard, Clauses 10 to 14. For brevity, let us simply say that, on top of common intervals, the empty set and unbounded intervals, such as $[1, +\infty)$ are allowed, but not $[2, 1]$. Operations and functions on intervals are defined, in the set-based flavor, as follows: if $f$ is a function and $\boldsymbol{x}_1, \dots \boldsymbol{x}_n$ are intervals, $f(x_1, \dots \boldsymbol{x}_n)$ is defined as

$$f(\boldsymbol{x}_1, \dots \boldsymbol{x}_n) = \text{Hull}\{f(x_1, \dots x_n) \ : \ x_i \in \boldsymbol{x}_i \text{ for } 1 \leq i \leq n \text{ and } f(x_1, \dots x_n) \text{ is defined}\}$$

where Hull defines the convex hull of the given set, so as to return an interval. This means that for instance $\sqrt{[-1, 4]} = [0, 2]$ and $\sqrt{[-2, -1]} = \emptyset$. The first example is not a common one as the input interval is not included in the domain of the square root: the result has been computed by intersecting the input interval with the domain of the square root, prior to computing the square root. The second example is not a common one as the result is not a common interval.

In the same way as `cancelMinus` is the reverse operation of the subtraction, several *reverse* functions act on the usual functions; these reverse functions are listed in the standard as mandatory functions. They are highly useful for constraint solving methods. Furthermore, as it permits the use of unbounded intervals, the set-based flavor defines the reverse of the multiplication. This `mulRevToPair` operation is the key element, mentioned in the introduction, to separate zeros in Newton's method.

Other flavors could be added to the IEEE 1788-2015 standard. Flavors that have been considered during the discussions of the working group are Kaucher arithmetic [5], modal arithmetic [1, 2] or Rump's proposal to handle the discretization of real intervals [19]. However, the effort has not been pursued until the adoption of the corresponding flavor for the current version of the standard. Anyone willing to propose a new flavor should submit it as a revision of the standard.

## 4 Decorations

The main requirements for any flavor are, on the one hand, its compatibility with the common part and, on the other hand, the presence of a version of the FTIA. This ensures that the most precious feature of interval arithmetic is preserved.

The last precious feature mentioned in the introduction was that Brouwer theorem is made effective in interval computations. However, an example in the set-based flavor illustrates that a naive use can be misleading. Let us consider the function

$$f \ : \ x \mapsto \sqrt{x} - 1.$$

In $\mathbb{R}$, $f$ has no fixed-point. The evaluation of $f$ over the interval $\boldsymbol{x} = [-1, 4]$ in the set-based flavor yields

$$f(\boldsymbol{x}) = f([-1, 4]) = \sqrt{[-1, 4]} - 1 = [0, 2] - 1 = [-1, 1].$$

This new interval $f(\boldsymbol{x})$ is enclosed in $\boldsymbol{x}$. However Brouwer theorem does not apply in this case, as $f$ is not continuous on $\boldsymbol{x}$ and not even everywhere defined.

The dilemma faced by the working group was thus to choose between two possibilities. The first one consists in returning an invalid value for $f(\boldsymbol{x})$, similar to computations with NaN values in the IEEE 754-2008 standard for floating-point arithmetic, at the risk of getting this invalid value so often that interval arithmetic would be impracticable. The second one consists in returning $f(\boldsymbol{x}) = [-1, 1]$ in this example and in providing the user with a means to check whether the computation encountered "out-of-domain" values during its course. The choice made in the IEEE 1788-2015 standard is the second one: each result comes with a piece of information about the circumstances of its computation.

Furthermore, it has been decided to avoid any kind of global information and to attach this piece of information to the result itself. In the IEEE 1788-2015 standard, this piece of information is called a *decoration* and an interval with a decoration attached to it is a *decorated interval*.

This choice is the result of long and hot discussions within the working group: issues about memory usage (a decoration uses extra memory, perturbs padding...), computation time (each operation must compute its result, but also compute and propagate its decoration) as well as development time (operations get more delicate to implement) were raised.

For common intervals, the only decoration is `common`, abbreviated as `com`. For the set-based flavor, the set of decorations is the set {`com`, `dac`, `def`, `trv`, `ill`}. Here is the meaning of each decoration (what follows is an excerpt of the standard).

| Value | Short description | Property | Definition |
|---|---|---|---|
| `com` | common | $p_{\texttt{com}}(f, \boldsymbol{x})$ | $\boldsymbol{x}$ is a bounded, nonempty subset of Dom($f$); $f$ is continuous at each point of $\boldsymbol{x}$; and the computed interval $f(\boldsymbol{x})$ is bounded; |
| `dac` | defined & continuous | $p_{\texttt{dac}}(f, x)$ | $\boldsymbol{x}$ is a nonempty subset of Dom($f$), and the restriction of $f$ to $\boldsymbol{x}$ is continuous; |
| `def` | defined | $p_{\texttt{def}}(f, \boldsymbol{x})$ | $\boldsymbol{x}$ is a nonempty subset of Dom($f$); |
| `trv` | trivial | $p_{\texttt{trv}}(f, \boldsymbol{x})$ | always true (so gives no information); |
| `ill` | ill-formed | $p_{\texttt{ill}}(f, \boldsymbol{x})$ | Not an Interval; formally Dom($f$) = $\emptyset$. |

These are listed according to the propagation order, which may also be thought of as a quality-order of $(f, \boldsymbol{x})$ pairs—decorations above `trv` are "good" and those below are "bad".

## 5 Level 2: Discretization Issues

It has been mentioned that the starting point for every decision in the standard was the mathematical level, or Level 1. Going from Level 1 to Level 2 implies to consider issues related to the use of a finite set of intervals, called an interval type.

Let us assume that the result of some computation, at Level 1, is an interval $\boldsymbol{z}$. At Level 2, a result must be returned, that belongs to the given interval type and that satisfies the FTIA, i.e., the result must be a representable interval that encloses $\boldsymbol{z}$. This result can be much larger than $\boldsymbol{z}$, in case of an overflow: in this case, a bounded interval (at Level 1) can be represented as an unbounded interval at Level 2.

However, the standard prevents an implementation to be too lazy and to return too large intervals. Indeed, it requires that the interval computed at Level 2 is of good quality, by specifying the accuracy of the results: for most operations, the result at Level 2 must be the tightest possible, that is, it is the smallest, for inclusion, interval of the given interval type that encloses $\boldsymbol{z}$.

Another issue when going from Level 1 to Level 2 is that, at Level 2, every operation must return a result. However, at Level 1, for some operations and some particular inputs, there might be no valid result. When the Level 1 result does not exist, the operation at Level 2 returns either a special value indicating this event (e.g., NaN for most of the numeric functions) or a value considered reasonable in practice. For instance, *the mid() function at Level 2 in the set-based flavor, returns the midpoint of* $\emptyset$ *as NaN, and of* $\mathbb{R}$ *as* 0*; this illustrates flavor-defined values. Both values are undefined at Level 1. It was considered that no numeric value of mid($\emptyset$) makes sense, but that some algorithms are simplified by returning a default value* 0 *for mid($\mathbb{R}$).*

*The relation between a Level 1 operation and a version of it at Level 2, is summarized as follows* in the standard. *The latter evaluates the Level 1 operation on the Level 1 values denoted by its inputs. If (at those inputs) the operation has no value, an exception is signaled, or some default value returned, or both, in a flavor-defined way. Otherwise the returned value is converted to a Level 2 result of an appropriate Level 2 datatype. If the result is of interval type, overflow may occur in some flavors, causing an exception to be signaled.*

Furthermore, implementors of IEEE-1788 compliant libraries raised the following issue: one of the most difficult part in the development of the libraries happened to be the implementation of tight conversions, for instance when the endpoints of an interval have different types. The standard accounts for this difficulty in the following way. *An implementation may support an extended form of literals, e.g., using number literals in the syntax of the host language of the implementation. It may restrict the support of literals at Level 2, by relaxing conversion accuracy of hard cases: rational number literals, long strings, etc. What extensions and restrictions of this kind are permitted is flavor-defined.*

## 6 Exact Dot Product

Another topic, related to Level 2 exclusively, is the recommendation that the dot product of vectors with scalar, floating-point endpoints, is evaluated as if in exact arithmetic – hence the name of *Exact Dot Product* – up to the final rounding. In particular, no intermediate underflow or overflow is possible. This recommendation is based on the argument that interval arithmetic offers the possibility to get high-quality numerical results. Including the exact dot product in the standard augments the numerical quality of computed results.

## 7 Libraries Implementing the Standard

Up to the author's knowledge, only two libraries are compliant with the IEEE 1788-2015 standard. One of them, `libieee1788` [14], has been developed by Marco Nehmeier. It is a C++ library that offers every possibility described in the standard. In particular, intervals can be given to constructors with almost any type for each of the endpoints. The other, a bit earlier, library is Octave Interval [4], developed by Oliver Heimlich. As its name suggests, it is called through Gnu Octave, which makes it very easy to use and thus allows to test and check ideas and algorithms very rapidly. To keep the development simple, the only numerical type allowed for the endpoints in particular is the `binary64` (double-precision) format for floating-point numbers defined in [6].

Unfortunately, neither author is still in academy and it is not clear how these libraries will be maintained over time.

## 8 Conclusion

This succinct summary of the IEEE 1788-2015 standard aims at giving a glimpse of the discussions that led to the development of this standard. The focus was on the less-than-obvious choices that have been made. As implementations, and even more crucially applications, are still in their infancy, no definitive conclusion can be drawn regarding the validity of these choices. The author hopes that this standard will reveal useful for a large audience, through the already available libraries, and wishes that these libraries will find a sustainable support.

## Acknowledgments

The author would like to thank Alessandro Abate and Sylvie Boldo for their kind invitation. The author would also like to thank all participants to the working group that developed the IEEE 1788-2015 standard for providing the material for this standard, for their differing points of view that questioned every proposal and that eventually contributed to create a solid standard, and for their never-fading enthusiasm. Last but not least, special thanks go to R. B. Kearfott [8] and J. Pryce [16] for the collective work and for their inspiring papers.

## References


1. A. Goldsztejn. Modal Intervals Revisited, Part 1: A Generalized Interval Natural Extension. *Reliable Computing*, 16:130–183, 2012.
2. A. Goldsztejn. Modal Intervals Revisited, Part 2: A Generalized Interval Mean Value Extension. *Reliable Computing*, 16:184–209, 2012.
3. E.R. Hansen and G.W. Walster. *Global optimization using interval analysis (2nd ed.)*. Marcel Dekker, 2003.
4. O. Heimlich. Interval arithmetic in GNU Octave. In *SWIM 2016: Summer Workshop on Interval Methods*, 2016.
5. E. Kaucher. Interval Analysis in the Extended Interval Space IR. In *Fundamentals of Numerical Computation (Computer-Oriented Numerical Analysis)*, pages 33–49. Springer, 1980.
6. IEEE: Institute of Electrical and Electronic Engineers. *754-2008 - IEEE Standard for Floating-Point Arithmetic*. IEEE Computer Society, 2008.
7. IEEE: Institute of Electrical and Electronic Engineers. *1788-2015 - IEEE Standard for Interval Arithmetic*. IEEE Computer Society, New York, June 2015.
8. R. B. Kearfott. An overview of the upcoming IEEE P-1788 working group document: Standard for interval arithmetic. In *IFSA/NAFIPS*, pages 460–465, 2013.
9. U.W. Kulisch. *Computer Arithmetic and Validity: Theory, Implementation, and Applications*. de Gruyter, Berlin, 2008.
10. M. Lerch, G. Tischler, J. Wolff von Gudenberg, W. Hofschuster, and W. Krämer. filib++ A Fast Interval Library Supporting Containment Computations. *Transactions on Mathematical Software*, 32(2):299–324, 2006.
11. R.E. Moore. *Interval analysis*. Prentice Hall, 1966.
12. R.E. Moore. *Methods and applications of interval analysis*. SIAM Studies in Applied Mathematics, 1979.
13. R.E. Moore, R.B. Kearfott, and M.J. Cloud. *Introduction to Interval Analysis*. SIAM, 2009.
14. M. Nehmeier. libieeep1788: A C++ Implementation of the IEEE interval standard P1788. In *Norbert Wiener in the 21st Century (21CW), 2014 IEEE Conference on*, pages 1–6. IEEE, 2014.
15. A. Neumaier. *Interval methods for systems of equations*. Cambridge University Press, 1990.
16. J. Pryce. The Forthcoming IEEE Standard 1788 for Interval Arithmetic. In *International Symposium on Scientific Computing, Computer Arithmetic, and Validated Numerics - LNCS 9553*, pages 23–39. Springer, 2015.
17. N. Revol and F. Rouillier. Motivations for an Arbitrary Precision Interval Arithmetic and the MPFI Library. *Reliable Computing*, 11(4):275–290, 2005.
18. S. M. Rump. Verification Methods: Rigorous Results using Floating-Point Arithmetic. *Acta Numerica*, 19:287449, 2010.
19. S. M. Rump. Interval arithmetic over finitely many endpoints. *BIT Numerical Mathematics*, 52(4):1059–1075, 2012.
20. Warwick Tucker. *Validated Numerics – A Short Introduction to Rigorous Computations*. Princeton University Press, 2011.