An Overview of the Upcoming IEEE P-1788 Working Group Document: Standard for Interval Arithmetic

Ralph Baker Kearfott
Department of Mathematics
University of Louisiana at Lafayette
Lafayette, Louisiana 70504-1010
Email:rbk@louisiana.edu

***Abstract*—We provide a summary of the goals, underlying philosophy, work, decisions, product, and completion schedule of the IEEE P-1788 working group on interval arithmetic.**

I. Introduction

Interval arithmetic, invented several times during the early twentieth century and rising to prominence with Ramon Moore's initial work [16], [17], [18], has had a consistent presence within the scientific computing world during the past 50 years. Publicly available software packages, as well as several hardware implementations, have been available throughout this time span, and continue to be developed, for platforms and languages such as Fortran 66, Fortran 77, Fortran 90 and 2003, C, C++, Matlab, Mathematica, Maple, Java, etc. It also is not uncommon for developers of higher-level packages to supply their own interval arithmetic. A partial list of interval arithmetic software and packages using or built upon interval arithmetic can be found on the web page [12]. Some of our own work containing basic introductions to interval arithmetic, as well as tutorials on certain packages and further references, is in [10], [19]. Some articles describing applications relating interval computations and soft computing appear in [9].

Several hundred researchers presently identify themselves as experts in interval computations. However, with a solid scholarly literature and web presence, many others, initially unknown to the core "interval community," are experimenting with interval arithmetic or are incorporating it into their projects.

A basic goal of interval arithmetic is to take account of both uncertainties in input data and roundoff errors in floating point computations in a mathematically rigorous way[1]. This leads to the ability to use floating point computations to provide mathematical proofs of appropriately posed assertions, to provide guarantees that a system subject to manufacturing uncertainties will remain within performance bounds, etc. Such proofs are largely effected by computing mathematically rigorous upper and lower bounds on the range of a function over hyper-rectangles, using the fundamental theorem of interval arithmetic:

Theorem 1: (Fundamental Theorem of Interval Arithmetic) Suppose $f(x_1, \cdots, x_n)$ is an expression of n variables built from the four basic arithmetic operations addition, subtraction, multiplication, and division, and other functions φ of one or more variables, collectively called "library functions." Suppose interval evaluations $\boldsymbol{\varphi}(\boldsymbol{x})$ of the library functions φ are defined such that $\boldsymbol{\varphi}(\boldsymbol{x})$ contains the range of φ over $\boldsymbol{x}$. Then the interval evaluation $\boldsymbol{f}(\boldsymbol{x})$ obtained by replacing each arithmetic operation and library function by the corresponding interval version contains the range of f over $\boldsymbol{x}$.

Proofs utilizing the fundamental theorem of interval arithmetic perhaps even have legal implications, but will be correct only if the software utilizing interval arithmetic is bug-free. Also, such software is more likely to be available if it is reasonably portable. These goals are more easily attained if the underlying system is standardized in a logical, complete way that is as simple as possible. Furthermore, programmers and implementers are less likely to make mistakes or misinterpret results if different interval arithmetic systems behave in the same, known way.

II. Why Isn't Standardization Trivial?

Basic interval arithmetic can be specified with simple, well-defined rules. For example, addition and multiplication for non-empty bounded intervals can be characterized by

$$\begin{aligned} [\underline{x}, \overline{x}] + [\underline{y}, \overline{y}] &= [\underline{x} + \underline{y}, \overline{x} + \overline{y}] \quad \text{and} \\ [\underline{x}, \overline{x}] \cdot [\underline{y}, \overline{y}] &= \left[\min\left\{\underline{x}\underline{y}, \overline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\overline{y}\right\}, \right. \\ &\qquad \left. \max\left\{\underline{x}\underline{y}, \overline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\overline{y}\right\}\right], \end{aligned} \tag{1}$$

with analogous operations for subtraction and division. Within common floating environments, it is even straightforward to unambiguously define the computer-generated results corresponding to (1). For example, to ensure interval versions of the four basic operations contain the actual ranges of those operations, it is sufficient to round the floating point computation of the lower bound (such as in (1)) down and the upper bound up. In IEEE-754 arithmetic [14], this can be

[1]However, some useful applications of interval arithmetic, such as [8], among them fuzzy applications such as in [9] are non-rigorous computations.

effected in an optimally tight way with the standard rounding modes: One simply sets the rounding mode to "down" when computing the lower end point of the result and the rounding mode to "up" when computing the upper end point.

Since this method of proceeding is seemingly so clear, why are there challenges and controversy in standardization of interval arithmetic? The following issues come to the forefront.

Evaluation over partial domains: Even though the result of an individual arithmetic operation is a very close approximation to the actual range of that operation (and rigorously contains the range), combining a sequence of such operations to evaluate an expression gives an interval that only contains the range, often with considerable overestimation beyond that due simply to roundoff error[2]. For example, suppose we wish to evaluate

$$E(x) = \sqrt{x^2 + x + 1}$$

over $\boldsymbol{x} = [-1, 1]$. The exact range of $e(x) = x^2 + x + 1$ over $[-1, 1]$ is $[0.75, 3]$, while an interval evaluation, using $[-1, 1]^2 = [0, 1]$, gives

$$e(\boldsymbol{x}) = [-1, 1]^2 + [-1, 1] + 1 = [-1, 3].$$

When we subsequently take the square root of this expression, we are left with the problem of evaluating a square root over an interval that contains zero. If we are interested only in the range of E over $\boldsymbol{x}$, we may ignore the negative part, and return $\sqrt{[0, 3]} = [0, \sqrt{3}]$ (or the smallest floating point interval containing this interval) for $E(x)$.

In many contexts, such continuation of the evaluation over the intersection of the input interval with the domain is permissible. However, in some contexts, it is not. For example, the Brouwer fixed point theorem states that if G is a continuous function over a compact set $\boldsymbol{x}$ (such as an interval vector) and the range of G over $\boldsymbol{x}$ is contained in $\boldsymbol{x}$, then there is a fixed point $x^* \in \boldsymbol{x}$, $x^* = G(x^*)$. Since the interval evaluation $\boldsymbol{G}(\boldsymbol{x})$ contains the range of G over $\boldsymbol{x}$, $\boldsymbol{G}(\boldsymbol{x}) \subseteq \boldsymbol{x}$ leads to the conclusion G has a fixed point in x. However, if, we take

$$G(x) = \sqrt{x^2 - 1} \quad and \quad \boldsymbol{x} = [-2, 2],$$

we would obtain

$$\boldsymbol{G}(\boldsymbol{x}) = \sqrt{[-1, 3]} = [0, \sqrt{3}] \subset [-2, 2],$$

and we would conclude G had a fixed point in $[-2, 2]$, an incorrect conclusion. (To see this, note that $\sqrt{x^2 - 1} = x$ implies $x^2 - 1 = x^2$, from which $-1 = 0$; thus, G can have no fixed points.)

The standard needs to somehow accommodate evaluation over partial domains, while raising an exception when needed.

Exceptions in general: The working group has devoted a significant amount of effort discussing the types of exceptions that can occur, which must be tracked, and how to track them efficiently; see the synopsis for Motion 36 in the next section.

Extended arithmetic: Conceptually, the result of a single arithmetic operation is defined to be the range of that operation over the input intervals. This is not a problem for interval division when the denominator is an interval that does not contain zero, but is a problem when it does. Furthermore, due to overestimation, the actual range of a denominator may not contain zero, while the interval evaluation does. If we consider $-\infty$ to be part of the number system, that is, if we consider the underlying set to be the two-point compactification $\overline{\mathbb{R}}$ of the real numbers, the interval evaluation of, say, $1/[0, 1]$ might be construed to be set of two disjoint intervals

$$[-\infty, -\infty] \cup [1, \infty].$$

In contrast, if the underlying system is simply the set of real numbers $\mathbb{R}$, the quantity $1/0$ is not defined, and $1/[0, 1]$ can only be defined over a partial domain. In this case, a logical definition would be $1/[0, 1] = [1, \infty)$. In the former case, the single smallest interval containing the range would be the entire set of extended reals $[-\infty, \infty]$.

For other interval pairs, such as $[1, 2]/[-1, 1]$, two disjoint intervals, namely

$$(-\infty, -1/2] \cup [1/2, \infty),$$

seem inevitable whether or not the underlying set is assumed to be $\mathbb{R}$ or $\overline{\mathbb{R}}$. It is sometimes useful for the program or application to know both parts of the range, but may also be convenient to simply return an interval containing the union of the parts, and it is appropriate in some contexts to raise or signal an exception. For consistency but also utility, the standard needs to specify interval division.

Extensions of the elementary functions: The functions φ in Theorem (1) typically are the functions such as $\sin$, $\cos$, $\exp$, asin, etc. usually accessible with common scientific programming languages such as C++ and Fortran. If φ is such a function, the logical interval value for $\varphi(\boldsymbol{x})$ (where $\boldsymbol{x}$ is an interval vector) would be the narrowest floating point interval containing the mathematical range of φ over $\boldsymbol{x}$. However, for some φ and $\boldsymbol{x}$, it may be difficult to return this optimal range with a reasonable cost. Furthermore, analogously to division, standard functions such as the inverse trigonometric functions and square roots have points where they are undefined and can have inverse images consisting of more than one connected component. A consistent, well-documented and universally available way of handling such situations, flexible enough for all common applications should therefore be defined.

Alternative systems: With one or two exceptional situations, the IEEE-754 standard data types and rounding modes can be combined naturally with intervals represented in terms of lower bound and upper bound and arithmetic defined such as in (1), with the rules clearly and simply specified. The IEEE 754 standard even provides specification for the accuracy of elementary functions that can be used in an interval standard (subject to the exception handling and need for additional specification we have listed). However, other representations and underlying systems are in use today in various applications. Some of these are

- multiple and variable precision interval arithmetic such as in [23],

[2]This is due to the *dependency problem*. See any good elementary introduction to interval arithmetic, such as [19, Section 5.2, p. 38] for an explanation.

- representation in terms of midpoint and radius ("mid-rad" arithmetic)[3] such as in [24], and
- modal or Kaucher arithmetic[4] such as in [2], [6], [21], [7], [7] or in fuzzy computing [25].

The standardization process has needed to decide how much and how to standardize such systems, common features and consistency of these systems, and conversion between representations in these systems.

Reproducibility: Is it practical to specify the arithmetic in the standard in such a way that the same computation carried out on two different systems conforming to the standard always give the same result? Is such reproducibility desirable? The IEEE P-1788 working group discussed such issues in its mailing list [13]. Two main sources that prohibit reproducibity are compiler optimizations and accuracy requirements. Guaranteeing high accuracy in all cases can require either special hardware or extensive computing time, while effcient software-implementable methods exist that will guarantee the result is within one or two floating point numbers of the exact result; However, specifying an operation to such a relaxed accuracy does not guarantee reproducibility. For example the IEEE 754R floating point standard [14, Section 11] gives requirements for reproducible results, to be available with an optional reproducibility mode. This issue has been the subject of P-1788 working group discussion.

Accurate dot product: A special (and controversial) operation tied with tight enclosures of exact results in linear algebra operations is the accurate dot product. In particular, there are ways of computing the dot product $v \circ w$ such that the computed result is the nearest machine number to the exact result. Guaranteeing such accuracy in the worst case can require either special hardware or extensive computing time, while efficient software-implementable methods exist that will guarantee the result is within one or two floating point numbers of the exact result; however, specifying an operation to such a relaxed accuracy does not guarantee reproducibility. The IEEE P-1788 working group has dealt with inclusion of a dot product specification of exact or relaxed accuracy; see the synopsis for Motion 9 in the next section.

After giving a brief history of interval standardization efforts, we present an overview of the present state of the standards document and how these issues have been resolved.

III. A Short History

The author participated in a relatively early effort at standardization through a formally sanctioned committee in the mid 1990's, in the ANSI X3J3 / ISO WG5 working groups on Fortran standardization. A working document was produced, but a consensus could not be reached, particularly within the community of experts on interval computation, and the Fortran working group needed to focus on other issues.

[3]more efficient for certain matrix operations

[4]efficiently implemented and also useful for obtaining *inner estimations*, that is, intervals guaranteed to be contained in the range, and a coherent algebraic system

Slightly after and overlapping with the Fortran standardization effort, Chenyi Hu, Jürgen Wolff von Gudenberg, Michael Nooner and others produced a document in the BLAST forum, an organization for standardizing the level-1, level-2, and level-3 Basic Linear Algebra Subroutines for modern computer architectures [5], [1]; a C++ implementation is described in [20].

Somewhat later (in the mid 2000's), Ulrich Kulisch led an effort within the IFIP (International Federation for Information Processing) Working Group 2.5 on Numerical Software to develop a recommendation for interval arithmetic within the eventual IEEE 754-2008 standard. Simultaneously, the Interval Subroutine Library (ISL) working group studied interval arithmetic standardization [22]. Due to timing and also the inability of highly interested participants to attend certain crucial working group meetings, action was not taken within the P-754 working group. However, Prof. Kulisch's efforts have formed the basis for the present IEEE P-1788 working group, officially approved to develop a separate interval standard in 2008.

Simultaneously with Prof. Kulisch's effort, Hervé Brönnimann, Guillaume Melquiond, and Sylvain Pion led an effort within the ISO/IEC JTC1/SC22/WG21 committee (International Standards Organization working group on standardization of C++) to specify a standard library for interval arithmetic, based on the BOOST library for interval arithmetic in C++ [15], [3]. The author of this article, along with George Corliss, Ned Nedialkov, John Pryce, and Spencer Smith participated in this as part of the focus of the ISL (Interval Subroutine Library) project [4].

The IEEE P-1788 working group first planning meeting occurred in January, 2008 at Schloss Dagstuhl Seminar 08021, "Numerical Validation in Current Hardware Architectures," and the PAR (Project Authorization Request) was officially approved on June 12, 2008, for a four year work period, with Nathalie Revol as chair. A two-year extension was approved in 2012, for an completion-of-work date by December 31, 2014.

IV. Current Status of the Standard

As of the writing of this article, the committee is still processing motions, and the standards document is still under development, and subject to change. However, we summarize decisions (successful motions) that have been made as well as the current state. We present the summary in the order in which corresponding motions were made[5].

1. **Notation:** The notation used in the standards document will be as in [11].
2. **Structure:** An agreement on the overall structure of the document, largely borrowed from the IEEE-754 standard.
3. **Sets of reals:** "The P1788 Interval arithmetic standard defines intervals as closed and connected sets of real numbers." That means that $\pm\infty$ may be used as bounds of an interval but are never considered

[5]We also give motion numbers.

as members of an interval. In particular, this has implications for the results of division by intervals that contain zero, as explained in the introduction to this article.

5. **The basic operations:** A four-page report giving explicit rules for computing the results of the four basic operations, including handling of unbounded intervals such as $[2,3] \cdot [1,\infty)$ and explicitly prescribing rounding modes to be used, was approved.
6. **Formats (now "interval types"):** An 8-page report was adopted. Therein, it states that multiple formats will be supported within the 1788 document, and that an implementation must support at least one format. A special *754-conforming* format is defined, as well as general terminology for formats. A basic idea is that different interval formats correspond to different floating point formats (such as single, double, decimal), through the representation of their end points. However, interval formats can go beyond that, such as representation by a midpoint and radius or representation of an interval $[a,b]$ by $[-a,b]$. (In this last format, operations on the lower and upper bounds can proceed without a change of rounding mode.) Clarifying definitions are made, and conversions shall be provided between all supported interval formats.
8. **Exception handling:** Exception handling will be provided through the concept of *decorations*. A decoration is an interval datum with extra bits giving information about the computation leading to the datum. For example, a decoration for the interval $[0,\sqrt{3}]$ resulting from computing $\sqrt{[-1,3]}$ would carry information that at least one operation in the chain leading to $[0,\sqrt{3}]$ had an argument partially outside its domain. The standard is to require "bare intervals" (without decorations), "bare decorations," and decorated intervals. A rough analogue in IEEE 754 is the NaN: the NaN can be viewed as a rudimentary form of a bare decoration.
9. **Exact dot product:** A standard-conforming interval arithmetic implementation shall include an exact dot product.
10. **Elementary functions:** A table of required elementary functions and a table of additional recommended functions has been approved.
12. **Inner addition and subtraction:** In ordinary interval arithmetic, $[-1,1] - [-1,1] = [-2,2] \neq 0$, since the first $[-1,1]$ doesn't necessarily represent the same interval as the second $[-1,1]$. If we assume it does, we get the cancellation or "inner" version of subtraction, the result of which is [0,0]. Such cancellation operations, useful in several contexts, are approved to be required.
13. **Comparison relations:** A table of required comparison operations (number order relations and set inclusion relations) has been approved.
16. **On endpoint and midpoint-radius arithmetic:** This states that a standard-conforming implementation shall have at least one "inf-sup" (representation in terms of endpoints) data type with all required operations implemented, and that conversions between inf-sup and midpoint-radius form shall be provided. The conversions "shall preserve containment and return the tightest representable interval in the target type."
17. **I/O:** Syntax for intervals in text form is defined, as well as conversions between text and internal intervals that preserve enclosure in either direction. This includes an exact text form for each interval type, from which the interval value can be recovered exactly.
18. **A first requirement for an exception-handling decoration:** Basically, the decision is that a decoration shall be provided to carry information on whether or not there were points in an argument to an operation that were outside of its domain.
19. **Implicit and explicit data types:** This clarifies some details related to endpoint and midpoint-radius arithmetic (as in item 16).
21. **Overlapping detection:** A function shall be provided to determine in which of 13 states (given in two tables) of overlapping the two argument intervals are found. For example, if $\boldsymbol{x} = [-1,0]$ and $\boldsymbol{y} = [1,2]$, every element of $\boldsymbol{x}$ is less than every element of $\boldsymbol{y}$, while if $\boldsymbol{x} = [-1,1]$ and $\boldsymbol{y} = [0,2]$, $\boldsymbol{y}$ is to the right of $\boldsymbol{x}$ but overlaps $\boldsymbol{x}$. (There are 13 such distinguishable states.)
24. **Rounding of floating point operations:** Every 1788-compliant system (regardless of whether it implements an IEEE-754 data type) shall provide rounding to nearest, rounding upwards to the next floating point number, rounding downwards to the next floating point number, and rounding to the smallest floating point number.
29. **Interchange data type:** This requirement is to provide for standardized binary transfer of interval data between different standard-conforming systems. It requires an "interchange data type" and conversion operations between each 754-conforming interval data type and the interchange datatype. It also specifies that, aside from the interchange data type, the bit-level representations of interval data types are not specified in the standard.
30. **Interval constructors:** Functions that create bare and decorated intervals from non-interval data shall be provided. This requirement specifies construction of bare and decorated intervals from two extended real (floating point) values, as well as bare and decorated intervals from a text string.
33. **Underlying number formats:** In the inf-sup as well as mid-rad (or possible other) data types, the bounds of the interval are represented by floating point numbers, in a system $\mathcal{F}$. General requirements

for $\mathcal{F}$ are specified.

34. **Notation:** Additional details (beyond Motion 1) concerning notation to be used in the standards document are specified.
36. **"Flavors:"** As we mentioned in §II, variants and different extensions of the basic interval arithmetic are in use. While these variants share properties and operations that give identical results on subsets of their domains of definition, they differ overall, and including all variants in the standard within the allotted time frame could be unwieldy. Here, "flavors" are defined, essentially, as these variants. The standard will specify a "set-based flavor" based on work done by the working group to this point, and the standard will define "common intervals" and "common operations" to be shared by all flavors. A conforming implementation shall support at least one flavor, and a program shall be able to determine which flavor is in effect. Additional flavors will be developed by the interested parties, and, after editorial review, will be included in the standard[6] or in future revisions of the standard.
37. **A detail on midpoints and radius:** There was a question on how to define the finite-precision midpoint and radius of an interval for the empty interval, an interval consisting of adjacent floating point numbers, and semi-infinite or infinite intervals. The definitions are specified (giving logically deduced reasons for the choices).
42. **Details of decorations:** A system of five decorations, "defined and continuous," "defined," "trivial," "empty[7]," and "ill-formed," is given, along with propagation rules and logical meanings.

In addition to these qualitative decisions, the working group is required to pass motions on the actual wording of the document text, with at least a $2/3$ majority. As of the writing of this article, the working group is processing such motions, with several having passed. Remaining issues within the working group (as of this article) are primarily completion and ratification of the actual standards text.

V. The Working Group, Process and Resources

As of this article, the official P-1788 working group presently consists of 89 official members, widely geographically distributed, mostly not from the United States. Most participants are affiliated with academic institutions and government laboratories, although persons affiliated with IBM, Intel, Oracle, and Sunfish Studios are active. Of the 89 registered members, 45 hold voting status. As of this article, 156 persons are subscribed to the mailing list [13] over which discussions occur and official business is transacted.

The term of the working group ends in December, 2014 after which, if the IEEE Standards Association formally approves it, the standard will be available for purchase from IEEE. Prior to that, interested persons may join the working group and participate in the standard's development; part of the participation includes access to the draft standard. See:

http://standards.ieee.org/develop/project/1788.html

After the document is delivered to the sponsor (the Microprocessor Standardization Committe, MSC), it enters the "sponsor ballot" phase, during which the public can comment. During this phase, a special balloting group is formed by invitation and appointment from the sponsor; participants are required to be either members of the IEEE Standards Association or pay a per-ballot fee. The standard is submitted for final review only if 75% of the balloting group participates, and 75% of those vote "yes" for the standard. Regardless, all negative comments must be addressed and either resolved, or changes recirculated with a new ballot. After the ballot phase, the ballot procedure is scrutinized for adherence to rules by the IEEE Standards Association. The document becomes a standard after such scrutiny.

[6] if completed by the deadline

[7] From additional analysis, the committee may be able to abolish this decoration.

Acknowledgment

We wish to thank the entire IEEE P-1788 working group, and the technical editors John Pryce and Christian Keil in particular. We also wish to thank the anonymous referee, whose careful reading led to valuable corrections and suggestions.

References

[1] L. S. Blackford, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, G. Henry, M. Heroux, L. Kaufman, A. Lumsdaine, A. Petitet, R. Pozo, K. Remington, and R. C. Whaley, "An updated set of basic linear algebra subprograms (BLAS)," *ACM Transactions on Mathematical Software*, vol. 2, no. 28, 2002.

[2] B. Bouchon-Meunier and V. Kreinovich, "From interval computations to modal mathematics: Applications and computational complexity," *SIGSAM Bulletin (ACM Special Interest Group on Symbolic and Algebraic Manipulation)*, vol. 32, no. 2, pp. 7–11, Jun. 1998.

[3] H. Brönnimann, G. Melquiond, and S. Pion, "A proposal to add interval arithmetic to the C++ Standard Library," CIS, Brooklyn Polytechnic University, Six Metrotech, Brooklyn, NY 11201, USA, Technical proposal N1843-05-0103, Aug. 2005. [Online]. Available: http://www-sop.inria.fr/geometrica/team/Sylvain.Pion/cxx/interval/inter%val_code.tgz;http://www-sop.inria.fr/geometrica/team/Sylvain.Pion/cxx/;http://boost.org/libs/numeric/interval/doc/interval.htm;http://www-sop.inria.fr/geometrica/team/Sylvain.Pion/cxx/interval/interval.p%df

[4] G. F. Corliss, R. B. Kearfott, N. Nedialkov, J. D. Pryce, and S. Smith, "Interval subroutine library mission," in *Reliable implementation of real number algorithms: Theory and practice. International seminar, Dagstuhl Castle, Germany, January 8–13, 2006. Revised papers*. Berlin, Germany / Heidelberg, Germany / London, UK / etc.: Springer-Verlag, 2008, pp. 28–43.

[5] J. Dongarra, "Basic linear algebra subprograms technical (BLAST) forum standard," *High Performance Computing Applications*, vol. 16, no. 1-2, pp. 1–199, 2001.

[6] E. Gardenes, M. A. Sainz, L. Jorba, R. Calm, R. Estela, H. Mielgo, and A. Trepat, "Modal intervals," *Reliable Computing*, vol. 7, no. 2, pp. 77–111, 2001.

[7] A. Goldsztejn, "Modal intervals revisited, part 1: A generalized interval natural extension," *Reliable Computing*, vol. 16, pp. 130–183, 2012.

[8] C. Hu and R. B. Kearfott, "Interval matrices in knowledge discovery," in *Knowledge processing with interval and soft computing*, ser. Advanced information and knowledge processing, C. Hu, R. B. Kearfott, and A. de Korvin, Eds. Berlin, Germany / Heidelberg, Germany / London, UK / etc.: Springer-Verlag, 2008, pp. 98–118.

[9] C. Hu, R. B. Kearfott, and A. de Korvin, Eds., *Knowledge processing with interval and soft computing*, ser. Advanced information and knowledge processing. Berlin, Germany / Heidelberg, Germany / London, UK / etc.: Springer-Verlag, 2008.

[10] R. B. Kearfott, *Rigorous global search: continuous problems*, ser. Nonconvex Optimization and its Applications. Dordrecht, The Netherlands: Kluwer Academic Publishers, 1996, vol. 13.

[11] R. B. Kearfott, M. T. Nakao, A. Neumaier, S. M. Rump, S. P. Shary, and P. van Hentenryck, "Standardized notation in interval analysis," *Computational Technologies*, vol. 15, no. 1, pp. 7–13, 2010.

[12] V. Kreinovich, "Interval and related software," World-Wide Web document., 2013. [Online]. Available: http://www.cs.utep.edu/interval-comp/intsoft.html

[13] IEEE Standards Association, "Working group P-1788 email archive," http://grouper.ieee.org/groups/1788/email/.

[14] ——, *IEEE Standard 754-2008: Standard for Floating-Point Arithmetic*. IEEE, 2008.

[15] G. Melquiond, S. Pion, and H. Brönnimann, "Boost C++ libraries: Interval arithmetic library," 2002, http://www.boost.org/libs/numeric/interval/doc/interval.htm.

[16] R. E. Moore, "Automatic error analysis in digital computation," Lockheed Missiles and Space Co., Sunnyvale, CA, USA, Technical Report Space Div. Report LMSD84821, 1959. [Online]. Available: http://interval.louisiana.edu/Moores_early_papers/Moore_Lockheed.pdf

[17] ——, "Interval arithmetic and automatic error analysis in digital computing." Ph.D. dissertation, Department of Mathematics, Stanford University, Stanford, CA, USA, Nov. 1962, also published as Applied Mathematics and Statistics Laboratories Technical Report No. 25. [Online]. Available: http://interval.louisiana.edu/Moores_early_papers/disert.pdf

[18] ——, *Interval analysis*. Upper Saddle River, NJ 07458, USA: Prentice-Hall, 1966.

[19] R. E. Moore, R. B. Kearfott, and M. J. Cloud, *Introduction to interval analysis*. Philadelphia, PA, USA: SIAM, 2009.

[20] M. Nooner, "IntBLAS: An interval linear algebra library in C++," Master's thesis, University of Central Arkansas, Department of Computer Science, 2006.

[21] E. D. Popova, "Multiplication distributivity of proper and improper intervals," *Reliable Computing*, vol. 7, no. 2, pp. 129–140, 2001.

[22] J. D. Pryce, G. C. Corliss, R. B. Kearfott, N. S. Nedialkov, and S. Smith, "Second note on basic interval arithmetic for IEEE754R," in *Numerical Validation in Current Hardware Architectures*, ser. Dagstuhl Seminar Proceedings, A. Cuyt, W. Krämer, W. Luther, and P. Markstein, Eds., no. 08021. Dagstuhl, Germany: Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2008. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2008/1451

[23] N. Revol and F. Rouillier, "Motivations for an arbitrary precision interval arithmetic and the mpfi library," *Reliable Computing*, vol. 11, no. 4, pp. 275–290, 2005.

[24] S. M. Rump, "Fast and parallel interval arithmetic," *BIT*, vol. 39, no. 3, pp. 534–554, Sep. 1999. [Online]. Available: http://www.springerlink.com/openurl.asp?genre=article&issn=0006-3835&vo%lume=39&issue=3&spage=534

[25] T. Rzeżuchowski and J. Wasowski, "Solutions of fuzzy equations based on Kaucher arithmetic and AE-solution sets," *Fuzzy Sets Syst.*, vol. 159, no. 16, pp. 2116–2129, Aug. 2008.