The Forthcoming IEEE Standard 1788 for Interval Arithmetic

John Pryce

School of Mathematics, Cardiff University
smajdp1@cardiff.ac.uk

Abstract. This is a slightly expanded form of the author's talk of the same title at SCAN 2014, Würzburg. Angled towards people who use interval numerical methods little or not at all, it briefly describes how interval arithmetic works, the mindset required to use it effectively, why an interval arithmetic standard was needed, the setting up of IEEE Working Group P1788 for the purpose, the structure of the standard it has produced, some difficulties we encountered, and the current state of the P1788 project. During production of these Proceedings the 1788 standard has been published, but the talk's original title has been kept.

This article, slightly expanded from the author's talk of the same title at SCAN 2014, briefly describes how interval arithmetic works, the mindset required to use it effectively, why an interval arithmetic standard was needed, the setting up of IEEE Working Group P1788 for the purpose, the structure of the standard it has produced, some difficulties we encountered, and the current state of the P1788 project. During production of these Proceedings the 1788 standard has been published, but the talk's original title has been kept.

The references include a recent survey [14], a recent textbook [17], and a current web site [8], that testify to the liveliness of this area.

1 What intervals are and do

1.1 Basic ideas

Interval Arithmetic (IA) implements "validated", also called "verified", numerical calculation. That is, it can *enclose* solution components x of a problem in an interval, i.e. between lower and upper bounds

$$x \in \boldsymbol{x} = [\underline{x}, \overline{x}] = \{\, t \in \mathbb{R} \mid \underline{x} \le t \le \overline{x} \,\}.$$

It does this even in finite-precision arithmetic, with roundoff errors present.

E.g. it makes Brouwer's fixed point theorem:

> If $K \subset \mathbb{R}^n$ is compact convex, and function f is everywhere defined and continuous on K, and $f(K) \subseteq K$, then f has a fixpoint in K

verifiable when K is a box (product of intervals) in the sense that during the evaluation of f, sufficient conditions for "everywhere defined and continuous" can be found—with ease in favourable cases, but maybe requiring both brute force and finesse in trickier cases..

The history of interval arithmetic might be traced back to Archimedes, in the sense that he rigorously proved the bounds

$$3\tfrac{10}{71} < \pi < 3\tfrac{1}{7},$$

see Thomas Heath [3]. As a systematic discipline it seems to have begun in the 20th century: Teruo Sunaga (Japan, 1958) [16]; Leonid Kantorovich (USSR, 1962) [4]. The most influential work of that time was the book by Ramon Moore (USA, 1966) [11], describing for instance the first implementation of a *validated ODE solver*.

Currently significant validated software exists for global optimisation, large sparse linear systems, particle beam design for the Large Hadron Collider, and many other applications. Rather than list extensive references we refer to those in Siegfried Rump's survey article [14] and in Vladik Kreinovich's web site [8], and also to the recent introductory book by Warwick Tucker [17].

1.2 Definition of interval operations

Interval operations take *all combinations* of points in the inputs, i.e.

$$\boldsymbol{x} \bullet \boldsymbol{y} = \{\, x \bullet y \mid x \in \boldsymbol{x} \text{ and } y \in \boldsymbol{y} \,\}, \quad \text{where } \bullet \text{ is one of } \{+ \;\; - \;\; \times \;\; \div\}$$

For $\div$, disallow $0 \in \boldsymbol{y}$ for now. In finite precision *round outward*. With these definitions one has the following fact, probably first stated by Moore:

Theorem 1 (Fundamental Theorem of Interval Arithmetic). *If a function $f(x_1, \ldots, x_n)$, defined by an expression, is evaluated with interval operations on interval inputs to get $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$ then*

$$\boldsymbol{y} \text{ contains the range of } f \text{ over box } \boldsymbol{x}_1 \times \cdots \times \boldsymbol{x}_n \text{ in } \mathbb{R}^n.$$

Example 1. Let $f(x_1, x_2) = x_1 + x_2/x_1$, suppose 2-digit decimal arithmetic is used, and let the input intervals be $\boldsymbol{x}_1 = [3, 4]$, $\boldsymbol{x}_2 = [3, 5]$. We compute

$$\begin{aligned} \boldsymbol{x}_1 + \frac{\boldsymbol{x}_2}{\boldsymbol{x}_1} &= [3,4] + \frac{[3,5]}{[3,4]} = [3,4] + \left[\frac{3}{4}, \frac{5}{3}\right] \overset{\text{round}}{\longrightarrow} [3,4] + [.75, 1.7] \\ &= [3.75, 5.7] \overset{\text{round}}{\longrightarrow} [3.7, 5.7] = \boldsymbol{f}(\boldsymbol{x}_1, \boldsymbol{x}_2) = \boldsymbol{y}. \end{aligned}$$

$\boldsymbol{y}$ *does* contain the range of f over $\boldsymbol{x}_1 \times \boldsymbol{x}_2 = [3, 4] \times [3, 5]$, which with a bit of calculus is found to be $[4, 5.25]$. □

2 Why do intervals need new algorithms?

2.1 Example: interval version of Newton's iteration

Consider *Newton's method* for solving a 1-D nonlinear equation $f(x) = 0$.

A wrong approach. The usual formula is:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

A direct interval transcription of this would be

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \frac{\boldsymbol{f}(\boldsymbol{x}_k)}{\boldsymbol{f}'(\boldsymbol{x}_k)}$$

where $\boldsymbol{f}$ and $\boldsymbol{f}'$ are interval versions of (the computer code for) f and f'.

Unfortunately addition and subtraction of intervals—in infinite precision—just *adds* their widths. In symbols, $w(\boldsymbol{a} \pm \boldsymbol{b}) = w(\boldsymbol{a}) + w(\boldsymbol{b})$. In finite precision the result is even a little wider owing to roundoff.

So the width of $\boldsymbol{x}_{k+1}$ equals the width of $\boldsymbol{x}_k$ plus that of $\boldsymbol{f}(\boldsymbol{x}_k)/\boldsymbol{f}'(\boldsymbol{x}_k)$. The latter width is usually strictly > 0, so each interval cannot be narrower than the last, and usually is wider. Convergence of an interval algorithm to a root must involve the interval becoming smaller. The above simple transcription of Newton to intervals cannot possibly do that, and is bound to diverge!

A right approach. For a sensible solution to this problem, go back to basic theory. Let f be a C^1 function on a real interval I. Then by the Mean Value Theorem MVT, for any root z and any x, both in the interval, there is some ξ in the interval such that

$$f(x) = f(x) - f(z) = (x - z)f'(\xi) \tag{1}$$

so provided $f'(\xi) \neq 0$, see later,

$$z = x - \frac{f(x)}{f'(\xi)}. \tag{2}$$

A pointer to the right algorithm is in the quantifiers: ∀ root z, ∀ x, ∃ ξ. These give a geometric interpretation to equation (2), shown in Figure 1:

> *For any $x \in I$, a searchlight shone from the point $(x, f(x))$ on the curve, its rays bounded by the lowest and highest slopes of f on I, is certain to illuminate any root z (identified with $(z, 0)$ in the plane) in I.*

To convert this to something computable note that in (2):

- $f(x)$ must be computed as an *interval*, since f is program code, hence liable to roundoff.
- $f'(\xi)$ must also be an interval, for two reasons: (a) f' is program code; (b) ξ is only known to *exist*—its exact position is unknown (∃).
- However x can be a *point*. It is an arbitrary programmer-selected point in I (∀)—typically the midpoint is used in practice.

Hence, (∀) for any root $z \in I$ we *also* have

$$z \in \left(x - \frac{[f(x)]}{[f'(\xi)]} \right)$$

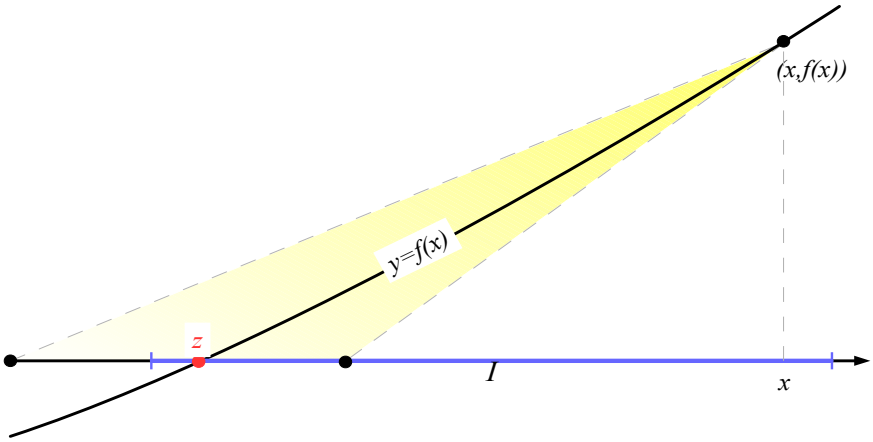

Fig. 1. Geometric view of Interval Newton: one-sided searchlight.

using classical interval notation that [...] means "some interval containing". In more current notation, renaming the interval as $\boldsymbol{x}$

$$z \in \left(x - \frac{\boldsymbol{f}([x])}{\boldsymbol{f}'(\boldsymbol{x})} \right) \qquad = \left(\mathit{point} - \frac{\mathit{interval}\text{ function of }\mathit{point}}{\mathit{interval}\text{ function of }\mathit{interval}} \right) \tag{3}$$

where $[x]$ is 1-point interval $\{x\}$ and $\boldsymbol{f}, \boldsymbol{f}'$ are interval versions of f, f'. This is the start of a satisfactory algorithm.

More general picture. Actually the searchlight shines in both directions, crucial when the range of slopes includes both positive and negative slopes, see Figure 2:

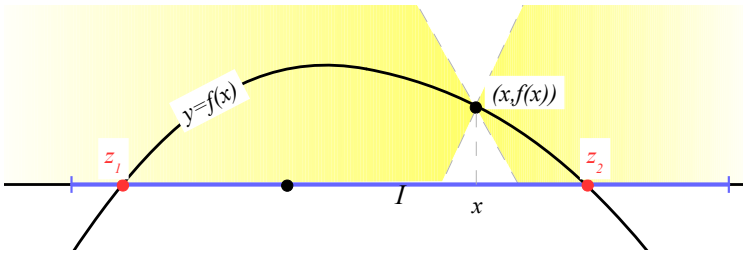

Fig. 2. Geometric view of Interval Newton: two-sided searchlight.

The same argument works as before, provided one interprets division as *reverse multiplication*, write this as $/\!\!/$:

$$c /\!\!/ b = \{ \text{ all solutions of } bx = c \}, \qquad 1788\text{'s } \texttt{mulRev}(b,c).$$

So 0/0 is "whole real line" instead of "undefined". That is, (3) is replaced by

$$z \in \Big(x - \big(\boldsymbol{f}([x] \,/\!/\, \boldsymbol{f}'(\boldsymbol{x}))\big)\Big). \tag{4}$$

This is just a restatement of formula (1), and its controlling quantifiers, two $\forall$ and one $\exists$. That is *always* valid even when (2) might divide by zero.

Now we enclose all roots even when many exist! However, the searchlight can "split" I into two pieces, as Figure 2 shows. 1788 provides a "two-output reverse multiplication" operation `mulRevToPair`, adapted to this situation.

Interval Newton iteration. We assumed the root z is in the interval $\boldsymbol{x}$ so it is safe to intersect the interval given by (4) with $\boldsymbol{x}$. This gives the 1983 method of Hansen and Greenberg [2], later refined by R.B. Kearfott [6], G. Mayer [10], P. van Hentenryck *et al.* [18], and others.

We assume the function f is C^1 on the initial interval.

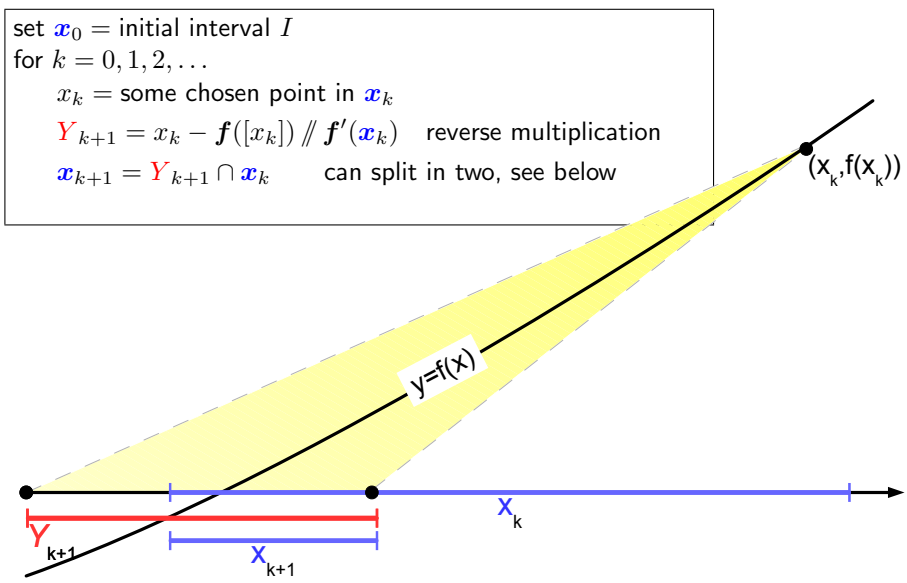

Fig. 3. One step of Interval Newton method (one-sided searchlight case).

2.2 Lessons from the example

Comments on the algorithm. Here Y_{k+1}, hence $\boldsymbol{x}_{k+1}$, is potentially a union of two intervals that can be handled independently.

This structures the computation as a binary tree that progressively divides root-clusters into smaller sets, trying (but not always succeeding) to isolate each individual root. This can be done by various tree-traversal methods, exploiting parallelism if available.

By construction it is clear that all roots in $\boldsymbol{x}_k$ must be in Y_{k+1}, hence in (the possibly split in two) $\boldsymbol{x}_{k+1}$. However the algorithm has other remarkable and less obvious properties:

- if $\boldsymbol{x}_{k+1}$ is empty, then *no root exists* in $\boldsymbol{x}_k$.
- If $0 \notin \boldsymbol{f}'(\boldsymbol{x}_k)$, then *at most one root exists* in $\boldsymbol{x}_k$ (which must be in $\boldsymbol{x}_{k+1}$).
- If Y_{k+1} is nonempty, bounded, and contained in $\boldsymbol{x}_k$, then *exactly one root exists* in $\boldsymbol{x}_k$.

Comments on the "interval mindset". The analysis leading to the algorithm wasn't rocket science—just a careful look at how the quantifiers $\forall$, $\exists$ appeared in a use of the Mean Value Theorem.

In general however, to learn how to convert mathematics to effective interval algorithms takes time and practice.

3 Genesis of the interval standard project

3.1 The need

Over the years, dozens of interval software packages have been written, and several, for instance PROFIL/BIAS, Filib++ and INTLAB, are widely used at present [7,9,13]. However, they have not quite compatible mathematical foundations, for instance different answers to these questions:

- Should theory, and software, support unbounded intervals and the empty set? Moore's interval arithmetic did not.
- Is an interval $\boldsymbol{x}$ a set of numbers? In Kaucher interval arithmetic, an interval is a set with a two-valued "mode". $[3, 4]$ is a *proper* interval, essentially the normal set. $[4, 3]$ is an *improper* interval; as a set it has the same value as $[3, 4]$ but its arithmetic rules are different.
- If $\boldsymbol{x}$ is a set of numbers, are $\pm\infty$ allowed to be members of $\boldsymbol{x}$?
- How to handle operations that are not everywhere defined on their input intervals, such as the square root of $[-2, 2]$, or division by an interval containing 0?

In addition, they have different software interfaces. Thus, currently one cannot write algorithms that are portable at a mathematical level, let alone write portable software.

3.2 Setting up a working group

In January 2008 at a conference in Dagstuhl, Germany, a project was started with the aim of producing an IA standard. In July that year it was approved by the IEEE as Working Group P1788 "A standard for interval arithmetic". In September, a conference in El Paso, USA, hosted its first face to face meeting at which the following officers were appointed.

Nathalie Revol, *Chair*
R. Baker Kearfott, *Vice Chair and Acting Chair*
William Edmonson, *Secretary*
Guillaume Melquiond, *Archivist*
J. Wolff von Gudenberg, *Web Master*
George Corliss, *Voting Tabulator*
John Pryce, *Senior Technical Editor*

Also during the project Christian Keil acted as *Deputy Technical Editor*, and Michel Hack, Vincent Lefèvre, Ian McIntosh, Dmitry Nadezhin, Ned Nedialkov and J. Wolff von Gudenberg were *Assistant Technical Editors*. About 140 people registered on the mailing list, of whom around 45 were regular voting members.

The group approved a final text in May 2014. Further revision, up to publication, then became the responsibility of the project's *sponsor ballot* group (whose membership overlaps with P1788's) and of IEEE editorial staff.

4 1788 Interval Principles

4.1 Definition of an interval

There is a framework—so called *flavors*—to support *alternative mathematical foundations*. The standard currently has just one flavor called Set-Based, in which

- An interval $\boldsymbol{x}$ is a plain *set*, whose members are *real numbers*. This excludes $\pm\infty$ as members, so intervals are subsets of the real line $\mathbb{R}$.
- Open or half-open intervals are not allowed, but unbounded intervals are.
- The empty set is an interval.

This amounts to the mathematically simple definition:

Interval means *topologically closed and connected subset of* $\mathbb{R}$.

A potential alternative flavor is *Kaucher* (or very similar *modal*) interval arithmetic [5]: an interval is not a plain set, but an ordered pair $(\underline{x}, \overline{x})$ of reals:

$$(\underline{x}, \overline{x}) \text{ "means" } \begin{cases} \text{set } [\underline{x}, \overline{x}] \subset \mathbb{R} & \text{if } \underline{x} \le \overline{x} \text{ ("proper" interval)} \\ \text{something other} & \text{if } \underline{x} > \overline{x} \text{ ("improper" interval).} \end{cases}$$

Another potentially important flavor is *Siegfried Rump's* interval arithmetic [15], which can support open or half-open intervals, and can handle finite precision under- and over-flow in a consistent and elegant way.

4.2 The Levels structure.

As in the IEEE floating-point standard 754, the 1788 standard manages complexity by distinguishing four specification levels:

Level 1. *Mathematical theory* of the set $\mathbb{IR}$ of **intervals** and their operations.
Level 2. *Finite precision* intervals—**datums**—and operations, independently of their representation.
Level 3. *Representation* of datums by **objects**, e.g. by a data structure comprising two floating-point numbers.
Level 4. *Encoding* of Level 3 objects as **bit strings**.

Inter-level maps. Maps between levels are crucial—especially those between Level 1 and Level 2, or L1 $\longleftrightarrow$ L2 for short. The P1788 group made the following design decisions, which apply to all flavors:

- Each datum *is* a mathematical interval, so the map from L2 to L1 is just inclusion:

  $$\text{L2 datums} \overset{\text{identity map}}{\longrightarrow} \text{L1 intervals} \qquad (*)$$

  (Not quite true: the datum also "knows what type it belongs to", i.e. is tagged with a unique name of its type. A programming language needs this information, since different types will be represented differently at L3.)
- Datums are organised into finite sets $\mathbb{T}$ called *interval types*. Thus each $\mathbb{T}$ may be regarded as a finite subset of $\mathbb{IR}$. The implementation has discretion on what types to provide.
- A L1 interval $\boldsymbol{x}$ maps to an interval of type $\mathbb{T}$—a $\mathbb{T}$-*interval* or $\mathbb{T}$-*datum*—by the $\mathbb{T}$-*hull* operation

  $$\text{hull}_T(\boldsymbol{x}) = \text{smallest } \mathbb{T}\text{-interval that contains } \boldsymbol{x},$$

  where "contain" has a flavor-defined meaning (which for Set-Based intervals is the usual one). This defines the map back from L1 to L2:

  $$\text{L1 intervals} \overset{\mathbb{T}\text{-hull}}{\longrightarrow} \text{L2 datums of type } \mathbb{T} \qquad (**)$$

- To do an operation $\boldsymbol{x} \bullet \boldsymbol{y}$ at L2 on $\mathbb{T}$-datums, in any flavor:
  map $\boldsymbol{x}, \boldsymbol{y}$ to L1 by (*);
  do the operation at L1;
  map back to L2 by (**).

This specification of the relation between mathematics and finite precision looks trivial but is not: it defines the whole character of the standard. Not all IA theories are clear on this issue. Time will tell whether our choice was a wise one.

This choice affects implementations. For instance an arbitrary-precision interval package must be structured as a potentially infinite set of types, each containing finitely many intervals. It cannot comprise a single type containing potentially infinitely many intervals.

Maps for Levels 3 and 4. There are two fairly obvious rules:

- L2 $\longleftrightarrow$ L3: Each L2 datum is *represented by* at least one L3 object; each L3 object *represents* at most one L2 datum.
- L3 $\longleftrightarrow$ L4: Each L3 object is *encoded by* at least one L4 bitstring; each L4 bitstring *encodes* at most one L3 object.

5 Exception handling

5.1 A hypothetical scenario

Less than 10 years hence in the Old Bailey, London, ...

The case *Crown versus Google* concerns the Google Driverless Car, GDC. One of them badly injured a pedestrian who stepped into the road in front of it.

The GDC's emergency stop system is *designed* to act faster than a good human driver (undisputed) but is it badly *implemented* (disputed)?

The software uses an interval algorithm, built on a 1788-conforming library, which applies *Brouwer's fixed point theorem*. Could this have an error? E.g., it may have thought it had enclosed a root of an equation when it had not.

Depending on what software bugs are found (if any), liability might lie with the pedestrian's negligence. Or with GDC's software implementers. Or with the 1788 library implementers. Maybe even with the mathematicians who claimed to have proved the design of 1788 is correct?

A lot of money rides on whether 1788-based code might be wrong, when deciding that a function is defined and continuous on a box.

5.2 Theoretical context

Basic problem. How (at Level 1) to treat operations that are not everywhere defined and/or continuous on their input box? For example

(real) square root $\sqrt{x}$	$\dfrac{x}{y}$	$\texttt{floor}(x)$
$\sqrt{[-2,2]}$	$\dfrac{[2,3]}{[-1,1]}$	$\texttt{floor}([2.5,4.5])$
undefined on $-2 \le x < 0$	undefined if $y = 0$	discontinuous at $x = 3,\ 4$

We decided the default is "evaluate where defined, ignore where undefined", called *non-stop* or *loose* evaluation. For instance

$$\sqrt{[-2,2]} = \{\, \sqrt{x} \mid x \in [-2,2] \text{ and } x \ge 0 \,\} = [0, \sqrt{2}]$$

with no error reported. This is similar to IEEE 754 floating-point, which responds to an invalid operation such as $0/0$ or $\infty - \infty$ by returning the result NaN and continuing to compute.

This is a valid approach for, e.g., many *global optimisation* methods. It is not valid when applying *Brouwer's theorem*, which needs a guarantee that a function is everywhere defined and continuous on a box.

It also will not do for some *graphics rendering* algorithms, which need to know a function is everywhere defined on a box, but are not bothered about continuity.

Tracking function properties. One needs a mechanism to track whether a function has these desirable properties of definedness and/or continuity. This leads to a powerful extension of the Fundamental Theorem of interval arithmetic based on well-known theorems of set theory and analysis, which can be summarised:

> If for function f given by an expression, each individual library operation in f is everywhere defined on its input set, then the same holds for f. The same is true when *defined* is replaced by *defined and continuous*.

Example 2. Let $f(x) = 1/\sqrt{x}$, composed of library operations $\texttt{sqrt}(t) = \sqrt{t}$ followed by $\texttt{recip}(t) = 1/t$. Evaluate $\boldsymbol{z} = \boldsymbol{f}(\boldsymbol{x})$ in exact (Level 1) interval arithmetic. Abbreviate *defined and continuous* to DAC.

Let the input to f be $\boldsymbol{x} = [1, 4]$. We do $\boldsymbol{y} = \texttt{sqrt}(\boldsymbol{x}) = [1, 2]$, followed by $\boldsymbol{z} = \texttt{recip}([1, 2]) = [\frac{1}{2}, 1]$. Each operation is (everywhere) DAC on its input: `sqrt` on $[1, 4]$ and `recip` on $[1, 2]$. We conclude f is DAC on this $\boldsymbol{x}$.

If $\boldsymbol{x}$ is $[0, 4]$ then `sqrt` is DAC on this $\boldsymbol{x}$ but `recip` is not DAC on the resulting $\boldsymbol{y} = [0, 2]$, so we cannot say f is DAC on $\boldsymbol{x}$.

Similarly, if $\boldsymbol{x}$ is $[-2, 4]$ then `sqrt` fails to be DAC on this $\boldsymbol{x}$ and again we cannot say f is DAC on $\boldsymbol{x}$. □

In the two last cases of this example it is easy to prove f is definitely *not* DAC on $\boldsymbol{x}$, but for complicated functions in the presence of roundoff, to prove such a negative—*definitely not everywhere DAC*—is nearly impossible. Therefore the 1788 system only provides for a definite positive, which is cheap to compute. It can also say *definitely nowhere defined* on the input, which is cheap too.

5.3 Decorations

To provide a mechanism to track such properties of functions, we rejected the IEEE 754 standard's method of *global flags*, as being obsolete for today's massively parallel platforms. Instead, 1788 provides for *decorated intervals*. Such an interval is a pair $(\boldsymbol{y}, dy)$, also written $\boldsymbol{y}_{dy}$ when convenient:

- an ordinary interval $\boldsymbol{y}$,
- a tag dy called a decoration, giving data about definedness, continuity, etc[1].

Formally, a decoration d is a label for an assertion (boolean-valued function) $p_d(f, \boldsymbol{x})$ about a function $f : \mathbb{R}^n \to \mathbb{R}$ and a box $\boldsymbol{x} \subseteq \mathbb{R}^n$, for arbitrary n. Five decorations are defined in order of "goodness", `ill` < `trv` < `def` < `dac` < `com`:

`ill` Label for ill-formed intervals, formally "f is nowhere defined".
`trv` (trivial) Always true = "no information".
`def` f is everywhere defined on $\boldsymbol{x}$.
`dac` As `def`, plus f is everywhere continuous on $\boldsymbol{x}$.
`com` As `dac`, plus f is bounded on $\boldsymbol{x}$ at Level 2, meaning that no overflow occurred while computing it[2].

[1] *dy* is just a mnemonic, "decoration for $\boldsymbol{y}$". It has nothing to do with differentials.
[2] `com` means "common", see Section 6, but also that code can *verify* it is common.

Let $(\boldsymbol{y}, dy)$ result from evaluating an arithmetic expression $f(x_1, \ldots, x_n)$

- on correctly initialised decorated interval inputs $(\boldsymbol{x}_1, dx_1), \ldots, (\boldsymbol{x}_n, dx_n)$ (the programmer's responsibility),
- using correctly written decorated interval library operations (the implementation's responsibility).

Then Moore's Fundamental Theorem says

$$\boldsymbol{y} \text{ contains the range of } f \text{ over } \boldsymbol{x}_1 \times \cdots \times \boldsymbol{x}_n,$$

and in addition,

$$\text{the decoration } dy \text{ makes a true assertion about } f \text{ over } \boldsymbol{x}.$$

For instance if dy is computed to be `def` then f has been proved to be everywhere defined on $\boldsymbol{x}$.

As with a computed range enclosure, a computed decoration is often *not sharp*. E.g. it may be `trv` (no information) or `def` (defined) when actually `dac` (defined and continuous) is true. Much of the craft of IA is knowing how to "sharpen" such information, e.g. by cutting an input box into smaller boxes handled separately.

Example 3. Consider the fixpoint problem, to solve $g(x) = x$ where

$$g(x) = 2\sqrt{x} - \tfrac{1}{2}.$$

Roots are $x = \frac{3}{2} \pm \sqrt{2} = 0.0858\ldots$ or $2.9142\ldots$

We aim to use interval fixpoint iteration

$$\boldsymbol{x}_0 = \text{initial guess}; \quad \boldsymbol{x}_{n+1} = g(\boldsymbol{x}_n) \text{ for } n = 0, 1, \ldots$$

First, use ordinary undecorated interval arithmetic.

Case A: $\boldsymbol{x}_0 = [2, 3]$. Then

$$\boldsymbol{x}_1 = \left[2\sqrt{2} - \tfrac{1}{2}, 2\sqrt{3} - \tfrac{1}{2}\right] = [2.3\ldots, 2.9\ldots] \subset \boldsymbol{x}_0.$$

This is genuine and (by Brouwer's Theorem) it proves a fixpoint exists in $\boldsymbol{x}_1$.

Case B: $\boldsymbol{x}_0 = [-1, \frac{1}{16}]$. Then

$$\boldsymbol{x}_1 = 2\sqrt{[-1, \tfrac{1}{16}]} - \tfrac{1}{2} = 2\,[0, \tfrac{1}{4}] - \tfrac{1}{2} = [0, \tfrac{1}{2}] - \tfrac{1}{2} = [-\tfrac{1}{2}, 0], \text{ again } \subset \boldsymbol{x}_0\,!$$

But there is no root in $\boldsymbol{x}_0$, let alone $\boldsymbol{x}_1$! This is spurious, due to 1788's (undecorated) square root function discarding the negative part of $\boldsymbol{x}_0$ without comment.

Now use decorated interval arithmetic. The rule for propagating decorations is, roughly, that an operation outputs the worst decoration, in the "goodness" order defined on p.10, out of the decorations on its operands and the decoration

generated while performing the operation. Showing decorations as subscripts, Case B gives

$$\begin{aligned} \boldsymbol{x}_1 &= [2]_{\mathtt{dac}} \times \sqrt{[-1, \tfrac{1}{16}]_{\mathtt{dac}}} - [\tfrac{1}{2}]_{\mathtt{dac}} \\ &= [2]_{\mathtt{dac}} \times [0, \tfrac{1}{4}]_{\mathtt{trv}} - [\tfrac{1}{2}]_{\mathtt{dac}}, \\ &= [0, \tfrac{1}{2}]_{\mathtt{trv}} - [\tfrac{1}{2}]_{\mathtt{dac}} \qquad\qquad = [-\tfrac{1}{2}, 0]_{\mathtt{trv}}. \end{aligned}$$

Recall `trv` = "no information", so the calculation is, correctly, unable to verify the conditions of Brouwer's Theorem. But Case A produces

$$\boldsymbol{x}_1 = [2.3\ldots,\ 2.9\ldots]_{\mathtt{dac}},$$

proving g is DAC on $\boldsymbol{x}_0$, as well as mapping $\boldsymbol{x}_0$ into itself—the conditions for applying Brouwer's Theorem have been verified. □

The decoration system is the feature that most distinguishes 1788 from earlier IA systems. An annex in the Standard contains a rigorous proof of correctness: a *Fundamental Theorem of Decorated Interval Arithmetic*.

6 Difficulties the group encountered

Certain issues caused long and heated debate. We are grateful for the diplomatic skills the Chair and Vice-chair sometimes needed to deploy, and the good sense of IEEE procedural guidelines for "online democracy". Here are a few examples.

The choice of foundational mathematical model. Most users of IA are in the academic community, and most of these use some form of "interval is just a set of numbers" theory and software. But Kaucher/modal theory—with intervals like $[4, 3]$—has its proponents. One of them is Nate Hayes, whose company does high-quality graphics rendering for the movie industry. For its specialised interpolation algorithms, Kaucher methods are reported to give tighter enclosures and greater speed.

The resulting tension between "intervals for knowledge" and "intervals for profit" was fruitful. Faced with two related kinds of object, it is natural to look for a theory that supports both, in a tightly coupled sense that lets both exist in a computer program and inter-operate.

We tried this over a period of many months with set-based and Kaucher intervals, but failed. For instance, unbounded intervals within Kaucher theory needed arbitrary restrictions or led to logical contradictions—briefly, Kaucher cannot handle $[3, \infty)$ consistently, and set-based cannot handle $[4, 3]$.

This was the main motivation for the *flavor* concept. It allows different theories that are "recognisably 1788" in a loosely coupled sense. The main requirement is that each flavor's intervals must include Moore's original (closed, bounded, nonempty) real intervals, called *common intervals*, and a library of operations that at Level 1, when acting on common intervals, produce the same results in all flavors.

A Kaucher standard document was promised but has not materialised yet. However at least one other theory that holds promise for effective interval computation fits into the flavor mould—that of Rump [15], especially if coupled with the Gustafson *universal numbers* system [1] (and see elsewhere in this volume). So I feel the effort put into this part of the standard has not been wasted.

The decoration scheme. The group saw from the start that checking definedness and continuity of a function can in principle be automated, and early on rejected global flags in favour of decorating individual intervals. The chosen scheme took nearly two years, off and on, to decide. Initially we used separate boolean flags for "defined", "continuous", etc. Arnold Neumaier of Vienna first proposed that decorations should be a linear sequence. Of several such schemes, we nearly adopted one with 6 decorations, till Guillaume Melquiond pointed out that one of them gave no information not already available to a programmer, so we removed it to give the current 5-decoration system.

(We are also indebted to Arnold for his earlier document, the Vienna Proposal for Interval Standardization [12], from which many ideas in 1788 are drawn.)

Input/output. I/O is important. A key reason why the Algol 60 language died and Fortran, its arguably inferior contemporary, thrived is that the latter had a language-defined I/O scheme and the former did not.

The working group debated at length on what I/O should be required and how prescriptive the standard should be. Eventually it agreed to specify an external text representation of intervals, so called *interval literals*. Examples are `[empty]`, `[1.23,4.56]` and the uncertainty form `1.23?4` which means $1.23 \pm$ (4 units in the last place), i.e. $[1.19, 1.27]$.

An implementation shall provide functions to read such literals in free format, and write them in either free (e.g., for interactive work) or fixed (for tabulation) format. However we did not standardise the conversion specifiers (such as C's `%8.3f` or Fortran's `F8.3` for output of floating-point numbers), leaving this implementation-defined, to be standardised at a future revision.

In addition to the above transformations, which generally incur roundoff, each finite-precision type $\mathbb{T}$ shall define an *exact text representation*, giving loss-free conversion between $\mathbb{T}$-intervals and text strings. For types based on IEEE 754 numbers, 1788 specifies an *interchange encoding*, giving loss-free conversion between $\mathbb{T}$-intervals and bit strings. This last is 1788's only Level 4 requirement.

What to say about accuracy? The accuracy issue for intervals is different from that for floating-point. The result $\widetilde{\boldsymbol{y}}$ of an interval library operation must enclose[3] the mathematical result $\boldsymbol{y}$. If not, it is wrong, period. If it does—even if it is the useless result $[-\infty, +\infty]$—it is valid.

After much discussion we agreed that *tightness*—how close $\widetilde{\boldsymbol{y}}$ is to the enclosed $\boldsymbol{y}$—is a quality-of-implementation issue, barring a few cases where requiring optimal tightness is reasonable.

[3] This has been called the "Thou Shalt Not Lie" principle.

So the standard acts as a "regulatory authority" here—it does not specify an accuracy, but it requires a conforming implementation to state the accuracy of each of its library operations, in a verifiable way, using a format specified by the standard. There is typically a trade-off between accuracy and speed, and the aim is to make it possible for users to make a fair comparison of the merits of different implementations.

7 Current state

The main text has around 70 pages, of which roughly 60% are Level 1, 35% Level 2, 5% Level 3, with a half-page of Level 4.

Following approval by a vote of the group in May 2014, the text was extensively reworked with help from IEEE editorial staff to fit their style guidelines.

It was signed off in November 2014 to enter the Sponsor Ballot phase, and examined by a selected group representative of academia, software developers, industry, etc., and of geographical regions. They approved it after various changes, both editorial and technical. Finally, IEEE Std 1788$^{\text{TM}}$-2015 was approved by the IEEE Standards Board in June 2015, and published at the end of that month.

In addition, a *Basic Standard for Interval Arithmetic* (BSIA) has been written by Ned Nedialkov. At around 20 pages it is a cut down version, simpler to implement and suitable for undergraduate teaching. A program that runs under an implementation of the BSIA should run and give identical results up to roundoff under an implementation of the full standard. The IEEE have approved a project, P1788.1, for the BSIA to become a separate but related standard.

A Proof of Interval Newton properties

This appendix proves the properties stated in § 2.2. It may be of interest because item (iv) of the Theorem does not seem to have appeared in the literature before. An *interval extension* of a real function f of real variables means an interval function $\boldsymbol{f}$ of corresponding interval variables such that $y = f(x_1, \ldots, x_n)$ is in $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$ whenever x_i is in $\boldsymbol{x}_i$ for each $i = 1, \ldots, n$.

Theorem 2. *Let $f : \mathbb{R} \to \mathbb{R}$ be C^1 on an interval $\boldsymbol{x}$, which may be unbounded. Let $\boldsymbol{f}$ and $\boldsymbol{f}'$ be interval extensions of f and its derivative f', and let x be any point of $\boldsymbol{x}$. Define the set*

$$Y = x - \boldsymbol{f}([x]) \mathbin{/\!\!/} \boldsymbol{f}'(\boldsymbol{x})$$

where $/\!\!/$ denotes division in the sense of reverse multiplication. (Thus Y may be empty, an interval, or the union of two disjoint unbounded intervals.) Then

(i) Y contains all zeros of f in $\boldsymbol{x}$.
(ii) If $Y \cap \boldsymbol{x} = \emptyset$, there are no zeros of f in $\boldsymbol{x}$.
(iii) If $0 \notin \boldsymbol{f}'(\boldsymbol{x})$, there is at most one zero of f in $\boldsymbol{x}$.
(iv) If Y is nonempty, bounded and $\subseteq \boldsymbol{x}$, there is exactly one zero of f in $\boldsymbol{x}$.

Proof. (i) Let $z \in \boldsymbol{x}$ with $f(z) = 0$. By the Mean Value Theorem

$$f(x) = f(x) - f(z) = (x - z)f'(\xi). \tag{5}$$

for some $\xi \in \boldsymbol{x}$. By definition of interval extension, $f'(\xi) \in \boldsymbol{f}'(\boldsymbol{x})$ and $f(x) \in \boldsymbol{f}([x])$. Hence by the definition of reverse multiplication

$$x - z \in \boldsymbol{f}([x]) \mathbin{/\!\!/} \boldsymbol{f}'(\boldsymbol{x}),$$

that is

$$z \in x - \boldsymbol{f}([x]) \mathbin{/\!\!/} \boldsymbol{f}'(\boldsymbol{x})$$

as required.

(ii) This is immediate from (i).

(iii) In (5) let both z and x be roots in $\boldsymbol{x}$. Then we have $0 = f(x) - f(z) = (x - z)f'(\xi)$. By hypothesis $0 \notin \boldsymbol{f}'(\boldsymbol{x})$ which implies $f'(\xi) \neq 0$. Hence $x - z = 0$, $x = z$, so there is at most one root.

(iv) Write $\boldsymbol{b} = \boldsymbol{f}'(\boldsymbol{x})$, $\boldsymbol{c} = \boldsymbol{f}([x])$, both being nonempty by the definition of interval extension. By hypothesis $Y = x - \boldsymbol{c} \mathbin{/\!\!/} \boldsymbol{b}$ is nonempty and bounded, so $Z = \boldsymbol{c} \mathbin{/\!\!/} \boldsymbol{b}$ is nonempty and bounded.

I claim $0 \notin \boldsymbol{b}$. For suppose $0 \in \boldsymbol{b}$. Then $0 \notin \boldsymbol{c}$, for if $0 \in \boldsymbol{c}$ then Z is the unbounded set $\mathbb{R}$, contrary to hypothesis. Now two subcases arise.
– Either $\boldsymbol{b}$ is singleton [0], making Z empty, contrary to hypothesis.
– Or, $\boldsymbol{b}$ contains 0 and another point, in which case it contains points arbitrarily close to 0. Since $0 \notin \boldsymbol{c} \neq \emptyset$, $\boldsymbol{c}$ contains a nonzero point. Together these imply $\boldsymbol{c} \mathbin{/\!\!/} \boldsymbol{b}$ is unbounded, again contrary to hypothesis.

Thus all the cases of $0 \in \boldsymbol{b}$ give a contradiction, proving $0 \notin \boldsymbol{b}$. Hence by part (iii) there is *at most* one root in $\boldsymbol{x}$ and we must show there is *at least* one. If $f(x) = 0$ there is nothing more to prove, so assume $f(x) \neq 0$.

Let b^* be the bound of $\boldsymbol{b}$ nearest 0, so it is finite, $\neq 0$ and in $\boldsymbol{b}$. Let z^* be the intercept on the x-axis of the line through $(x, f(x))$ with slope b^*, so

$$z^* = x - f(x)/b^*, \tag{6}$$

equivalently

$$f(x) = (x - z^*)b^*. \tag{7}$$

Since $f(x) \in \boldsymbol{c}$ and $b^* \in \boldsymbol{b}$, (6) shows $z^* \in Y \subseteq \boldsymbol{x}$. Also $x \in \boldsymbol{x}$ so by the Mean Value Theorem there is $\xi \in \boldsymbol{x}$ with

$$f(x) - f(z^*) = (x - z^*)f'(\xi). \tag{8}$$

Subtracting this from (7) gives

$$f(z^*) = (x - z^*)(b^* - f'(\xi)) \tag{9}$$

Now $f'(\xi)$ is in $\boldsymbol{b}$ by the latter's definition, so by the definition of b^* it has the same sign as b^* and at least as large absolute value, i.e. $f'(\xi)/b^* \geq 1$. Dividing (9) by (7) (recalling $f(x) \neq 0$) now gives

$$f(z^*)/f(x) = 1 - f'(\xi)/b^* \leq 0,$$

so $f(z^*)$ has opposite (in the weak sense) sign to $f(x)$. By the Intermediate Value Theorem f has a zero z between x and z^*. Since both the latter are in $\boldsymbol{x}$ we have $z \in \boldsymbol{x}$, and the result is proved. □

References

1. J.L. Gustafson. *The End of Error: Unum Computing*. Chapman & Hall/CRC Computational Science. Taylor & Francis, 2015.
2. E. R. Hansen and R. I. Greenberg. An interval Newton method. *J. of Applied Mathematics and Computing*, 12:89–98, 1983.
3. T.L. Heath. *A History of Greek Mathematics*. Number v. 1 in A History of Greek Mathematics. Clarendon Press, 1921.
4. Leonid V. Kantorovich. On some new approaches to computational methods and to processing of observations. *Siberian Mathematical Journal*, 3(5):701–709, 1962. in Russian.
5. Edgar W. Kaucher. Interval analysis in the extended interval space IR. *Computing Suppl.*, 2:33–49, 1980.
6. R. Baker Kearfott. A Fortran 90 environment for research and prototyping of enclosure algorithms for nonlinear equations and global optimization. *ACM TOMS*, 21(1):63–78, 1995.
7. O. Knüppel. PROFIL/BIAS – a fast interval library. *Computing*, 53(3–4):277–287, 1994.
8. V Kreinovich. Interval computations web site. `http://cs.utep.edu/interval-comp`, accessed August 28, 2014.
9. Michael Lerch, German Tischler, Jürgen Wolff von Gudenberg, Werner Hofschuster, and Walter Krämer. FILIB++, a fast interval library supporting containment computations. *ACM Trans. Math. Softw.*, 32(2):299–324, 2006.
10. G. Mayer. Epsilon-inflation in verification algorithms. *Journal of Computational and Applied Mathematics*, 60:147–169, 1995.
11. Ramon E. Moore. *Interval Analysis*. Prentice-Hall, Englewood Cliffs, N.J., 1966.
12. Arnold Neumaier. Vienna proposal for interval standardization, December 2008. `http://www.mat.univie.ac.at/~neum/ms/1788.pdf`, accessed August 28, 2014.
13. S.M. Rump. INTLAB - INTerval LABoratory. In Tibor Csendes, editor, *Developments in Reliable Computing*, pages 77–104. Kluwer Academic Publishers, Dordrecht, 1999. `http://www.ti3.tuhh.de/rump/`.
14. S.M. Rump. Verification methods: Rigorous results using floating-point arithmetic. In *Acta Numerica*, pages 287–449. Cambridge University Press, 2010.
15. S.M. Rump. Interval arithmetic over finitely many endpoints. *BIT Numerical Mathematics*, 52(4):1059–1075, 2012.
16. Teruo Sunaga. Theory of an interval algebra and its application to numerical analysis. *Research Association of Applied Geometry Memoirs*, 2:29–46, 1958.
17. Warwick Tucker. *Validated Numerics: A Short Introduction to Rigorous Computations*. Princeton University Press, 2011. ISBN: 9781400838974.
18. P. van Hentenryck, D. Macallester, and D. Kapur. Solving polynomial systems using a branch and prune approach. *SIAM J. Numer. Anal.*, 34(2):797–827, April 1997.