Computer Algebra Style (Multiprecision) Interval and Complex Arithmetic

Franky Backeljauw[1], Annie Cuyt[1], Brigitte Verdonk[1], and Johan Vervloet[1,2]

[1]Dept of Mathematics and Computer Science
University of Antwerp (UIA)
Universiteitsplein 1, B2610 Antwerp, Belgium
{backelj,cuyt,verdonk,jvvloet}@uia.ua.ac.be
[2]Research Assistant FWO–Vlaanderen

1 Motivation

Using the approach in [2, 4, 5], interval bounds for values in $\mathbb{R}$ or $\mathbb{R}\cup[-\infty,+\infty]$, which is what one is interested in in most industrial applications, can be computed. These approaches, however, do not in all cases reflect all that is known about the interval valued expressions, as is required during prototyping or in a computer algebra environment.

A similar remark holds for the complex arithmetic guidelines proposed in Annex G of the latest C programming language standard [1]. The approach is sufficiently correct when some additional background information is available about the evaluated expression. The lack of such information, however, may lead to the ambiguous interpretation of results.

In the authors' implementation, a more theoretical point of view on interval and complex arithmetic is proposed to tackle the above issue. It is natural, when taking a computer algebra style viewpoint, to deal not only with double precision, but also consider true higher precisions.

2 Interval Arithmetic: Theory versus Implementation

In both [2] and [4], the authors review the implementation of the basic operations in interval arithmetic, and in particular discuss the different approaches given in the literature for interval division when the divisor interval contains zero.

Division by an interval containing zero is a special case of an interval function for which the interval arguments contain points outside the domain of the underlying point function. In [5] a general approach is presented to deal with such situations and to remove any restrictions on the domain of interval functions. This approach fully exploits the availability of the underlying IEEE hardware and has been efficiently implemented in [3].

While interval division is defined differently in [2] and [5] when the divisor contains zero, part of the difference can be traced back to the following. Underlying any implementation of interval arithmetic are two sets, a number set $\mathbb{S}$ and a set $I(\mathbb{S})$ of intervals, which is a subset of $2^{\mathbb{S}}$. In [2]

$$\begin{aligned} \mathbb{S} =& \mathbb{R} \\ I(\mathbb{S}) =& \{[a,b] \mid a,b \in \mathbb{S}, a \le b\} \cup \{]-\infty, b] \mid b \in \mathbb{S}\} \\ & \cup \{[a, +\infty[\mid a \in \mathbb{S}\} \cup \{]-\infty, +\infty[\} \cup \{\emptyset\} \end{aligned}$$

while in [5]

$$\begin{aligned} \mathbb{S} &= \mathbb{R} \cup \{-\infty, +\infty\} \\ I\mathbb{S} &= \{[a,b] \mid a,b \in \mathbb{S}, a \le b\} \end{aligned}$$

In both cases, an interval can be easily represented by a pair of (properly rounded) IEEE floating-point numbers. This ease of representation comes at a price, however, because the choice of the set $\mathbb{S}$ has crucial implications for the definition of interval functions. Even though an interval function satisfies the containment principle, it can only contain the range of the underlying point function in $\mathbb{S}$. If the range of the point function is a subset of $\mathbb{S}$, there is no problem. But if the underlying point function is undefined or complex-valued for some values of the interval arguments, returning an element of $I(\mathbb{S})$ may lead to unintuitive results, as we shall illustrate. This also explains why in [2]

$$[1,2]/[0,0] = \emptyset$$

while according to [5]

$$[1,2]/[0,0] = \{\infty\} \cap (\mathbb{R} \cup \{-\infty, +\infty\}) \subset [-\infty, +\infty]$$

In this presentation we give an alternative, computer algebra style approach to remove restrictions on the domain of interval functions. To achieve this, we allow for the efficient representation of non-real results. We indicate some important properties and advantages of this approach and show how the presented ideas can be implemented in a multiprecision interval arithmetic library without performance overhead.

3 Complex Arithmetic: Theory versus Implementation

In the same way that the approach in [3] is designed to offer interval arithmetic in a way that seamlessly blends in with IEEE floating-point arithmetic, the Annex

G of the latest C programming language standard [1] lists recommendations for implementations of complex arithmetic which fully respect the underlying IEEE floating-point arithmetic.

We explained how in interval arithmetic this leads to discussions about expressions where the result is either undefined or not exclusively real. In complex arithmetic conflicting results also come from the difficulty of representing and computing with the Riemann infinity. Support for projective infinity has mostly been dropped in floating-point implementations and hence implementations of complex arithmetic struggle in situations which require correct infinity arithmetic.

In order to salvage possibly incorrect complex results, the Annex G suggests to assign a double meaning to the IEEE NaN (Not-a-Number) when it occurs in complex expressions involving infinities: a NaN real or imaginary part indicates either an undefined value or an unknown value. This, however, does not fully take care of all problems, as the following example illustrates.

$$\begin{aligned}
&\texttt{double } x, y\\
&\texttt{complex } z\\
&x = 2\\
&y = (x^2 - 4)/(x - 2)\\
&z = 2^{1024} - 10^{340}\texttt{i}\\
&z = z \times (y + x\texttt{i})
\end{aligned}$$

The correct mathematical result is $z = \texttt{undefined}$. The Annex G proposal however returns $z = \infty + \infty\texttt{i}$ because it fails to recognize y as mathematically undefined. When trying to rectify the result of $z \times (y + x\texttt{i})$ involving an infinite z, it jumps to the other interpretation of the NaN value y.

We shall indicate how this type of problems can be overcome by the introduction of yet additional special values. Our alternative approach gives rise to an efficient implementation which is fully compliant with the theory of complex analysis, as would be required in a proper computer algebra style implementation.

References

[1] ANSI/ISO/IEC 9899-1999. *International C Standard, Annex G: IEC 60599 – compatible complex arithmetic (informative)*, ANSI, 1999.

[2] T. Hickey, J. Qun, and M. Van Emden. "Interval arithmetic: from principles to implementation", *Journal of the ACM*, 2001, Vol. 48, No. 5, pp. 1038–1068.

[3] Sun Microsystems, *Interval arithmetic in the Forte[tm] C++ compiler*; available at http://www.sun.com/forte/cplusplus/interval.

[4] D. Ratz, *On extended interval arithmetic and inclusion isotonicity*, Technical Report, Institut für Angewandte Mathematik, Universität Karlsruhe, 1996.

[5] G. W. Walster, *The extended real interval system*, Technical Report, 1998, available at http://www.mscs.mu.edu/ globsol/readings.html.