4ECM Stockholm 2004

Validated Numerics for Pedestrians

Warwick Tucker

Abstract. The aim of this paper is to give a very brief introduction to the emerging area of validated numerics. This is a rapidly growing field of research faced with the challenge of interfacing computer science and pure mathematics. Most validated numerics is based on interval analysis, which allows its users to account for both rounding and discretization errors in computer-aided proofs. We will illustrate the strengths of these techniques by converting the well-known bisection method into a efficient, validated root finder.

1. Introduction

Since the creation of the digital computer, numerical computations have played an increasingly fundamental role in modeling physical phenomena for science and engineering. With regards to computing speed and memory capacity, the early computers seem almost amusingly crude compared to their modern counterparts. Nevertheless, real-world problems were solved, and the speed-up due to the use of machines pushed the frontier of feasible computing tasks forward. Through a myriad of small developmental increments, we are now on the verge of producing Peta-flop/Peta-byte computers – an incredible feat which must have seemed completely unimaginable fifty years ago.

Due to the inherent limitations of any finite-state machine, numerical computations are almost never carried out in a mathematically precise manner. As a consequence, they do not produce exact results, but rather approximate values that usually, but far from always, are near the true ones. In addition to this, external influences, such as an over-simplified mathematical model or a discrete approximation of the same, introduce additional inaccuracies into the calculations. As a result, even a seemingly simple numerical algorithm is virtually impossible to analyze with regards to its accuracy. To do so would involve taking into account every single floating point operation performed throughout the entire computation. It is somewhat amazing that a program *performing only two floating point operations* can be challenging to analyze! At speeds of one billion operations per second, any medium-sized program is clearly out of reach. This is a particularly valid point for complex systems, which require enormous models and very long computer runs. The grand example in this setting is weather prediction, although much simpler systems display the same kind of inaccessibility.

This state of affairs has led us to the rather awkward position where we can perform formidable computing tasks at very high speed, but where we do not have the capability to judge the validity of the final results. The question *"Are we just getting the wrong answers faster?"* is therefore a valid one, albeit slightly unkind.

Fortunately, there are computational models in which approximate results are automatically provided with guaranteed error bounds. The simplest such model – *interval analysis* – was developed by Ramon Moore in the 1960's, see [Mo66]. At the time, however, computers were still at an early stage of development, and the additional costs associated with keeping track of the computational errors were deemed as too high. Furthermore, without special care in formulating the numerical algorithms, the produced error bounds would inevitably become overly pessimistic, and therefore quite useless.

Today, the development of interval methods has reached a high level of sophistication: tight error bounds can be produced – in many cases even faster than non-rigorous computations can provide an "approximation". As a testament to this, several highly non-trivial results in pure mathematics have recently been proved using computer-aided methods based on such interval techniques, see, e.g., [Ha95], [Tu02], and [GM03]. We have now reached the stage where we can demand rigor *as well as* speed from our numerical computations. In light of this, it is clear that the future development of scientific computation must include techniques for performing *validated numerics.*

2. Interval arithmetic

In this section, we will briefly describe the fundamentals of interval arithmetic. For a concise reference on this topic, see, e.g., [AH83], [KM81], [Mo66], or [Mo79]. For early papers on the topic see, [Yo31], [Wa56], and [Su58].

Let $\mathbb{IR}$ denote the set of closed intervals. For any element $[a] \in \mathbb{IR}$, we adapt the notation $[a] = [\underline{a}, \bar{a}]$. If $\star$ is one of the operators $+, -, \times, \div$, we define arithmetic operations on elements of $\mathbb{IR}$ by

$$[a] \star [b] = \{a \star b\colon a \in [a], b \in [b]\},$$

except that $[a] \div [b]$ is undefined if $0 \in [b]$. Working exclusively with closed intervals, we can describe the resulting interval in terms of the endpoints of the operands:

$$\begin{aligned}
[a] + [b] &= [\underline{a} + \underline{b}, \bar{a} + \bar{b}] \\
[a] - [b] &= [\underline{a} - \bar{b}, \bar{a} - \underline{b}] \\
[a] \times [b] &= [\min(\underline{a}\underline{b}, \underline{a}\bar{b}, \bar{a}\underline{b}, \bar{a}\bar{b}), \max(\underline{a}\underline{b}, \underline{a}\bar{b}, \bar{a}\underline{b}, \bar{a}\bar{b})] \\
[a] \div [b] &= [a] \times [1/\bar{b}, 1/\underline{b}], \quad \text{if } 0 \notin [b].
\end{aligned}$$

To increase speed, it is customary to break the formula for multiplication into nine cases (depending of the signs of the endpoints), where only one case involves more than two multiplications. When computing with finite precision, directed rounding must also be taken into account, see, e.g., [KM81] or [Mo79].

It follows immediately from the definitions that addition and multiplication are both associative and commutative. The distributive law, however, does *not* always hold. As an example, we have

$$[-1,1]([-1,0]+[3,4]) = [-1,1][2,4] = [-4,4]$$

whereas

$$[-1,1][-1,0]+[-1,1][3,4] = [-1,1]+[-4,4] = [-5,5].$$

This unusual property is important to keep in mind when representing functions as part of a computer program. Interval arithmetic satisfies a weaker rule than the distributive law, which we shall refer to as *sub-distributivity*:

$$[a]([b]+[c]) \subseteq [a][b]+[a][c].$$

Another key feature of interval arithmetic is that it is *inclusion monotonic*, i.e., if $[a] \subseteq [a']$, and $[b] \subseteq [b']$, then

$$[a] \star [b] \subseteq [a'] \star [b'],$$

where we demand that $0 \notin [b']$ for division.

Finally, we can turn $\mathbb{IR}$ into a metric space by equipping it with the Hausdorff distance:

$$d([a],[b]) = \max\{|\underline{a}-\underline{b}|, |\bar{a}-\bar{b}|\}. \tag{2.1}$$

3. Interval-valued functions

One of the main points of studying interval arithmetic is that we want a simple way of enclosing the *range* of a real-valued function. Let $D \subseteq \mathbb{R}$, and consider a function $f\colon D \to \mathbb{R}$. We define the range of f over D to be the set

$$R(f;D) = \{f(x)\colon x \in D\}.$$

Except for the most trivial cases, mathematics provides few tools to describe the range of a given function f over a specific domain D. Indeed, today there exists an entire branch of mathematics and computer science – Optimization Theory – devoted to “simply” finding the smallest element of the set $R(f;D)$. We shall see that interval arithmetic provides a helping hand in this matter.

As a first step, we begin by attempting to extend the real functions to *interval functions*. By this, we mean functions who take and return intervals rather than real numbers. We already have the theory to extend rational functions, i.e., functions on the form $f(x) = p(x)/q(x)$, where p and q are polynomials. Simply substituting all occurrences of the real variable x with the interval variable $[x]$ (and the real arithmetic operators with their interval counterparts) produces a rational interval function $F([x])$, called the *natural* interval extension of f. As long as no singularities are encountered, we have $R(f;[x]) \subseteq F([x])$, by the inclusion monotonicity property.

For future reference, we define the class of standard functions to be the set

$$\begin{aligned}\mathfrak{S} \quad = \quad & \{a^x, \log_a x, x^{p/q}, \operatorname{abs} x, \sin x, \cos x, \tan x, \ldots \\ & \ldots, \sinh x, \cosh x, \tanh x, \arcsin x, \arccos x, \arctan x\}.\end{aligned}$$

By using the fact that these functions are piecewise monotonic, it is possible to extend all standard functions to the interval realm: any $f \in \mathfrak{S}$ has a *sharp* interval extension F. By sharp, we mean that the interval evaluation $F([x])$ produces the *exact* range of f over the domain $[x]$:

$$f \in \mathfrak{S} \Rightarrow R(f; [x]) = F([x]).$$

Note that, in particular, this implies that $F([x, x]) = f(x)$, i.e., F and f are identical on $\mathbb{R}$.

Of course, the class of standard functions is too small for most practical applications. We will use them as building blocks for more complicated functions as follows.

Definition 3.1. Any real-valued function expressed as a finite number of standard functions combined with constants, arithmetic operations, and compositions is called an elementary function. The class of elementary functions is denoted by $\mathfrak{E}$.

Thus a representation of an elementary function is defined in terms of its sub-expressions. The leaves of the tree of sub-expressions (sometimes called a Directed Acyclic Graph – or a DAG for short) are either constants or the variable of the function, see Figure 1.

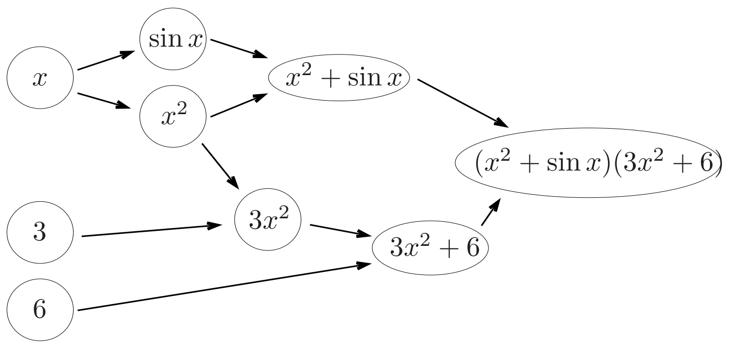

FIGURE 1. A DAG for $f(x) = (x^2 + \sin x)(3x^2 + 6)$.

It is important to note that, due to the intrinsic nature of interval arithmetic, the interval extension F depends on the particular representation of f. To illustrate this point, consider the functions $f(x) = x - x$ and $g(x) = 0$. Their natural interval extensions are $F([x]) = [x] - [x]$ and $G([x]) = [0, 0]$, respectively. Although f and g are identical over $\mathbb{R}$, their extensions differ over $\mathbb{IR}$.

Nevertheless, given a real-valued function f, any one of its (well-defined) interval extensions F satisfies $R(f;[x]) \subseteq F([x])$ due to the inclusion monotonicity property:

Theorem 3.2 (The fundamental theorem of interval analysis)**.** *Given an elementary function f, and a natural interval-extension F such that $F([x])$ is well defined for some $[x] \in \mathbb{IR}$, we have*

(1) $[z] \subseteq [z'] \subseteq [x] \Rightarrow F([z]) \subseteq F([z'])$, (*inclusion monotonicity*)

(2) $R(f;[x]) \subseteq F([x])$. (*range enclosure*)

For a proof, see, e.g., [Mo66].

Of course, the enclosure $F([x])$ is rarely sharp, and may in fact grossly overestimate $R(f;[x])$. If f is sufficiently regular, however, this overestimation can be made arbitrarily small by subdividing $[x]$ into many smaller intervals, evaluating F over each sub-interval, and then taking the union of all resulting sets. To make this statement more precise, we define $\mathfrak{E}_{\mathfrak{L}}$ to be the set of all (representations of) elementary functions whose sub-expressions are Lipschitz:

$$\mathfrak{E}_{\mathfrak{L}} = \{f \in \mathfrak{E}\colon \text{each sub-expression of } f \text{ is Lipschitz}\}.$$

Theorem 3.3 (Tight range enclosure)**.** *Consider $f\colon I \to \mathbb{R}$ with $f \in \mathfrak{E}_{\mathfrak{L}}$, and let F be an inclusion isotonic interval extension of f such that $F([x])$ is well defined for some $[x] \subseteq I$. Then there exists a positive real number K, depending on F and $[x]$, such that, if $[x] = \cup_{i=1}^{k}[x^{(i)}]$, then*

$$R(f;[x]) \subseteq \bigcup_{i=1}^{k} F([x^{(i)}]) \subseteq F([x])$$

and

$$w\left(\bigcup_{i=1}^{k} F([x^{(i)}])\right) \le w\big(R(f;[x])\big) + K \max_{i=1,\dots,k} w\left([x^{(i)}]\right).$$

Here, $w([x]) = \bar{x} - \underline{x}$ denotes the *width* of $[x]$. For a proof of this theorem, see, e.g., [Mo66].

In essence, the second part of Theorem 3.3 says that, if the listed conditions are satisfied, then the overestimation of the range tends to zero no slower than linearly as the domain shrinks:

$$d\big(R(f;[x]), F([x])\big) = \mathcal{O}(w([x])),$$

where $d(\cdot,\cdot)$ is the Hausdorff distance, as defined in (2.1). Since Lipschitz functions satisfy $w\big(R(f;[x])\big) = \mathcal{O}(w([x]))$, it also follows that

$$w\big(F([x])\big) = \mathcal{O}(w([x])),$$

i.e., the width of the enclosure scales (at most) linearly with $w([x])$, see Figure 2.

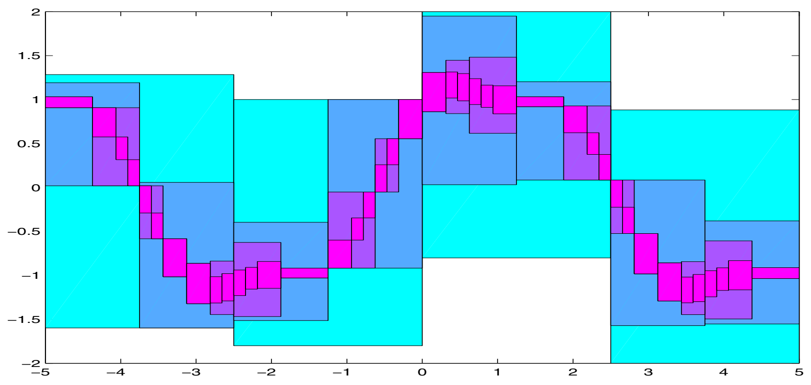

FIGURE 2. Successively tighter interval enclosures of $f(x) = \cos^3 x + \sin x$.

4. The bisection method

As a simple illustration of the powers of interval analysis, we will study the bisection method. This is a well-known algorithm for locating a zero of a continuous function. To be precise, let f be continuous on $[a, b]$, and suppose that $f(a)f(b) < 0$. Then, by the intermediate-value theorem, f has at least one root $\alpha \in (a, b)$. The bisection method proceeds as follows: Initially, we set $a_0 = a$ and $b_0 = b$. At stage k, we compute the midpoint $c_k = (a_k + b_k)/2$. Now there are three possibilities. If $f(c_k) = 0$, then we can set $\alpha = c_k$, and terminate the search. If $f(a_k)f(c_k) < 0$, we set $a_{k+1} = a_k$ and $b_{k+1} = c_k$. If $f(a_k)f(c_k) > 0$, we set $a_{k+1} = c_k$ and $b_{k+1} = b_k$. The search is guaranteed to converge to a root since we have $|a_k - b_k| = 2^{-k}|a_0 - b_0|$. When programming the bisection method, it is common to end the search when some predefined tolerance is met, e.g., $|a_k - b_k| \leq$ `tol`. A `C++` implementation of the real-valued bisection method is presented in Figure 3.

```
void bisect(pfcn f, double a, double b, double tol)
{                                   // We are assuming that f(a)*f(b) < 0.
  double c = (a + b)/2;
  double fc = f(c);
  if ( (b - a < tol) || (fc == 0) ) // If the tolerance is met, or f(c) = 0
     cout << c << endl;             // ... print the midpoint.
  else {                            // Otherwise...
     double ff = f(a)*fc;
     if ( ff < 0 )
       bisect(f, a, c, tol);        // ... check the left half, or
     else if ( ff > 0 )
       bisect(f, c, b, tol);        // ... check the right half.
  }
}
```

FIGURE 3. A recursive implementation of the real-valued bisection method.

There are, however, several flaws with the bisection method, when used as a root-finding device. One is the problem of finding points a and b satisfying the starting condition $f(a)f(b) < 0$. Indeed, if f is of constant sign, with the exception of a very small set, it may be impossible to even start the search. An example is given by the class of functions

$$f_\rho(x) = 1 - 2e^{-\rho^2(x-1/2)^2}.$$

Clearly, $f_\rho(1/2) = -1$ for all ρ. Nevertheless, when taking ρ large, almost all other function values are very close to $+1$, see Figure 4(a).

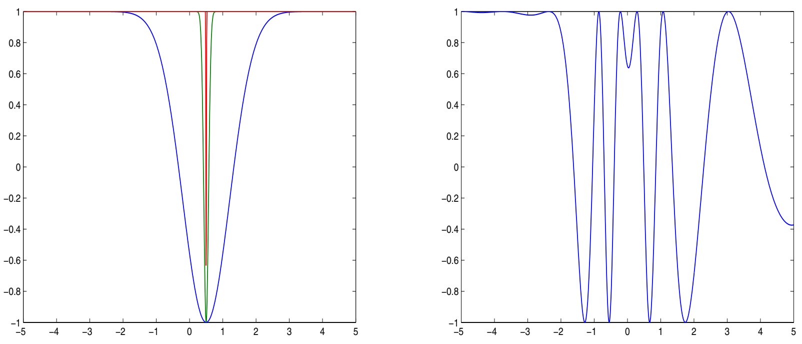

FIGURE 4. (a) $f_\rho(x) = 1 - 2e^{-\rho^2(x-1/2)^2}$ for $\rho \in \{1, 10, 100\}$.
(b) $f(x) = \sin\big(\sin(x) + 15/(x^2+1)\big)$

A second problem occurs when f has several roots within the search domain. Suppose that f has N simple roots $\alpha_1 < \alpha_2 < \cdots < \alpha_N$ in $[a, b]$. Then the bisection method will find the even-labeled roots with probability zero, whilst the odd-labeled roots are located with uniform probability, see [Co77].

The interval bisection methods deals elegantly with both problems. Instead of aiming directly at finding a root of f, it discards subsets of $[a, b]$ that are *guaranteed* to be root-free. By using the second part of Theorem 3.2, it follows that $0 \notin F([x]) \Rightarrow 0 \notin R(f; [x])$. Therefore, the strategy of the interval version bisection scheme is to recursively bisect the search space, retaining only those subintervals $[x_i]$ satisfying $0 \in F([x_i])$. Such intervals are called *feasible*, since they *may* contain roots of f. Once a feasible subinterval has reached a width smaller than the tolerance `tol`, it is sent to the output. A `C++` implementation of the interval-valued bisection method is presented in Figure 5. Note how much clearer the code is compared to its real-valued counterpart in Figure 3.

When the search has been exhausted, we are left with a collection of feasible intervals $[x_1], \dots, [x_M]$ whose union contains *all* roots of f within $[a, b]$:

$$Z = \{\alpha \in [a, b] \colon f(\alpha) = 0\} \subseteq \bigcup_{i=1}^{M} [x_i] = S.$$

```
void bisect(pfcn F, interval X, double tol)
{
  if ( subset(0.0, F(X)) ) // If zero is contained in F(X)
    if ( width(X) < tol )  // ... and the tolerance is met
      cout << X << endl;   // ... print the subinterval.
    else {                 // Otherwise, divide and conquer.
      bisect(F, interval(min(X), mid(X)), tol);
      bisect(F, interval(mid(X), max(X)), tol);
    }
}
```

Figure 5. A recursive implementation of the interval-valued bisection method.

Of course, the set S may grossly overestimate Z. If, however, f satisfies the assumptions of Theorem 3.3, we can expect a very good agreement between S and Z, as long as the tolerance `tol` is kept reasonably small.

In the case where f has only simple roots in $[a, b]$, there are several ways of determining weather a feasible interval $[x_i]$ contains a unique root or not, see, e.g., [AH83], [HH95], [Mo66], or [Mo79].

5. Examples

We will now provide two concrete examples of the interval-valued bisection method. The first example deals with the family of problematic functions f_ρ, described earlier. We will fix the search region to $[x] = [-5, 5]$, and set the tolerance to `tol` $= 10^{-10}$. It is clear that each f_ρ has two simple roots, both approaching $1/2$ as ρ increases.

Table 1. Interval enclosures of the roots of $f_\rho(x) = 1 - 2e^{-\rho^2(x-1/2)^2}$ for varying ρ.

ρ	α_1	α_2
10^0	$[-0.3325546111592, -0.3325546110863]$	$[+1.3325546111445, +1.3325546112174]$
10^1	$[+0.4167445388156, +0.4167445388885]$	$[+0.5832554610969, +0.5832554611698]$
10^2	$[+0.4916744538786, +0.4916744539515]$	$[+0.5083255461067, +0.5083255461796]$
10^3	$[+0.4991674453776, +0.4991674454505]$	$[+0.5008325546077, +0.5008325546806]$
10^4	$[+0.4999167445203, +0.4999167445931]$	$[+0.5000832553923, +0.5000832554652]$
10^5	$[+0.4999916744418, +0.4999916745147]$	$[+0.5000083255436, +0.5000083256164]$
10^6	$[+0.4999991674412, +0.4999991675141]$	$[+0.5000008325441, +0.5000008326170]$

In Table 1, we clearly see the numerically computed root-enclosures behave as expected. Note that for very large values of ρ, *even starting* the real-valued bisection method would require almost as much work as actually finding the roots of f_ρ.

The second example deals with a function having many roots within the search domain. We will study the function $f(x) = \sin\big(\sin(x) + 15/(x^2 + 1)\big)$, which is not completely trivial to analyze by hand. Once again, the search region is fixed to $[x] = [-5, 5]$, and the tolerance is set to `tol` $= 10^{-10}$.

The graph of Figure 4(b) indicates that f has nine roots in the search region. This is indeed confirmed by the output of the interval-valued bisection method, detailed in Table 2.

TABLE 2. Interval enclosures of the roots of $f(x) = \sin\big(\sin(x) + 15/(x^2+1)\big)$.

α_1	$[-1.61951630492695, -1.61951630485419]$
α_2	$[-1.04787158852560, -1.04787158845284]$
α_3	$[-0.69981597283914, -0.69981597276638]$
α_4	$[-0.39748093411618, -0.39748093404342]$
α_5	$[+0.49000622362655, +0.49000622369932]$
α_6	$[+0.85439020273042, +0.85439020280319]$
α_7	$[+1.35143495448574, +1.35143495455850]$
α_8	$[+2.29537873135996, +2.29537873143273]$
α_9	$[+4.12523527877056, +4.12523527884333]$

Note, however, that the real-valued bisection method, started on the domain $[a, b]$, would fail to locate the four even-labeled roots for almost any choices of $a \in [-5, \alpha_1)$ and $b \in (\alpha_9, 5]$.

References

[AH83] Alefeld, G., Herzberger, J., Introduction to Interval Computations. Academic Press, New York, 1983.

[Co77] Corliss, G., *Which Root Does the Bisection Algorithm Find?*, SIAM Review **19** (1977), 325–327.

[CXSC] CXSC – C++ eXtension for Scientific Computation, version 2.0. Available from
`http://www.math.uni-wuppertal.de/org/WRST/xsc/cxsc.html`

[GM03] Gabai, D., Meyerhoff, G.R., Thurston, N., *Homotopy hyperbolic 3-manifolds are hyperbolic*, Annals of Mathematics, **157:2** (2003), 335–431.

[Ha95] Hass, J., Hutchings, M., Schlafly, R. *The double bubble conjecture*, Electronic Research Announcements of the AMS **1** (1995), 98–102.

[HH95] Hammer, R. et al., C++ toolbox for Verified Computing. Springer-Verlag, Berlin, 1995.

[INv4] INTLAB – INTerval LABoratory, version 4.1.2. Available from
`http://www.ti3.tu-harburg.de/~rump/intlab/`

[KM81] Kulisch, U.W., Miranker, W.L., Computer Arithmetic in Theory and Practice. Academic Press, 1981.

[Mo66] Moore, R.E., Interval Analysis. Prentice-Hall, Englewood Cliffs, New Jersey, 1966.

[Mo79] Moore, R.E., Methods and Applications of Interval Analysis. SIAM Studies in Applied Mathematics, Philadelphia, 1979.

[PrBi] PROFIL/BIAS – Programmer's Runtime Optimized Fast Interval Library/Basic Interval Arithmetic Subroutines. Available from `http://www.ti3.tu-harburg.de/Software/PROFILEnglisch.html`

[Su58] Sunaga, T., *Theory of an Interval Algebra and its Application to Numerical Analysis*. RAAG Memoirs, **2** (1958), 29–46.

[Tu02] Tucker, W., *A Rigorous ODE Solver and Smale's 14th Problem*. Found. Comp. Math., **2:1** (2002), 53–117.

[Wa56] Warmus, M., *Calculus of Approximations*. Bulletin de l'Académie Polonaise de Sciences, **4:5** (1956), 253–257.

[Yo31] Young, R.C., *The algebra of multi-valued quantities*. Mathematische Annalen, **104** (1931), 260–290.

Warwick Tucker
Department of Mathematics
Uppsala University
Box 480
Uppsala, Sweden
e-mail: `warwick@math.uu.se`