Towards an Efficient Implementation of Interval Arithmetic

Ioannis Z. Emiris
Richard J. Fateman

Computer Science Division
University of California, Berkeley

Abstract

Interval mathematics is well studied and offers a solution to several important problems arising in computer arithmetic, such as error bounding and sign tests for constant expressions. However, modern computer algebra systems have a limited arsenal of routines dealing with intervals which, moreover, produce unnecessarily crude answers. This work attempts to fill in this gap, by proposing efficient yet simple algorithms for function evaluation and equation solving. The main contributions are, first, an evaluation procedure that outputs tight bounds for a large class of continuous differentiable functions and, second, an adaptation of Newton's iteration. Both improve upon the routines of *Maple V* and *Mathematica 2.0* for functions with complex or real interval coefficients and arguments.

1 Introduction

Computers handle numbers of finite size and this makes representations of real numbers prone to round-off and precision errors. Moreover, in many applications, the input data may be inaccurate. One way to cope with these different sources of imprecision is to keep track of the maximum error; another option is to maintain the endpoints of the interval in which the true value is known to lie through interval arithmetic. This same technique can be used to decide the sign of constant expressions, a very important question in computer algebra systems. Despite its apparent simplicity, such decisions may require involved and lengthy computations. Usually, the expression is the value of a function f at a point and by bounding f we may be able to decide its sign without computing the exact value which, in general, may not even be finitely represented.

Interval mathematics has been an area of extensive study for three decades, especially concerning its theoretical foundations; computer algebra systems, providing basic interval routines in addition to several tools for symbolic manipulation, offer an appropriate environment for experimentation. In fact, both *Maple V* and *Mathematica 2.0*, considered here, have taken some steps in this direction.

The main contribution of this work is to present powerful methods for important algebraic problems that are very useful by themselves, in addition to providing the necessary tools for answering the problems of bounding errors and deciding the sign of constant expressions. We describe in detail procedures to evaluate a function with real coefficients on a real or complex interval, in order to obtain an optimally tight interval of values for a wide class of multivariate continuous differentiable functions including the polynomial, trigonometric, exponential and logarithmic functions. For functions outside this class, the procedures return cruder bounds, while still improving upon the available facilities on *Maple V* and *Mathematica 2.0*.

The next problem is finding a root of an equation. For a large class of equations, we can adapt Newton's iteration in combination with some interval partitioning tricks and construct a fast convergent procedure that isolates real roots, for continuous and differentiable functions. It is a simple matter to extend it into isolating complex roots and to solving systems of equations.

These methods are intended for implementation, therefore, we pay particular attention to their simplicity. All of them can be implemented on any modern computer algebra system and can thus provide the basis for a larger family of interval arithmetic routines. Although the algorithms are not new, they are adapted to achieve a balance between efficiency as well as ease of implementation; the latter may rely on existing routines of *Maple V* and *Mathematica 2.0*, or available *Lisp* functions

We start with some definitions and notation in the next section. Section 3 is an overview of the current interval arithmetic operations in *Maple V* and *Mathematica 2.0*. Then our methods are presented: Section 4 deals with function evaluation and section 5 with equation solving. Section 6 concludes with a summary of the contributions of the paper and points out directions for future work.

2 Definitions

We focus on closed continuous real intervals, each denoted by the ordered pair of its lower and upper endpoint. There exist other options also; one alternative is to allow the union of disjoint intervals as one generalized interval, as in [HaG] and *Maple V*. This option offers flexibility, but is not considered here since it is not clear that the flexibility compensates for the computational overhead incurred. As another extension to our *interior* intervals, two endpoints may define an *exterior* interval, as proposed in [Ka] and discussed in [FaYa]. At present, we concentrate on interior intervals.

In this discussion, intervals are denoted by capital letters or, more explicitly, by their two endpoints, as in $X = [a, b]$ where $a \leq b$ and both $a, b \in \mathbf{R} \cup \{-\infty, \infty\}$, where $\mathbf{R}$ stands for the set of real numbers and $\mathbf{R} \cup \{-\infty, \infty\}$ is the set of extended reals. Notice that any real number c can be represented as $[c, c]$. The term interval shall be abused in the context of complex numbers to mean a rectangular range of values in the complex plane, so a number that is said to lie in interval $[-1, 0] + i[-2, 3]$, in fact lies in the rectangle with vertices $-1 - 2i$, $-2i$, $-1 + 3i$, $3i$.

Interval vectors can be defined analogously and are denoted by bold-face capitals. To illustrate, let the real interval vector $\mathbf{X} = [[-1, 2], [3, 5], [-7, -4]]$; it equals the set of 3-dimensional vectors $\{[x_1, x_2, x_3] \mid -1 \leq x_1 \leq 2,\ 3 \leq x_2 \leq 5,\ -7 \leq x_3 \leq -4\}$. Similarly, matrices can be defined with interval entries.

An important issue is how to represent the endpoints. We assume that there exist special symbols for infinity and concentrate on real endpoints. Exactness can be achieved only with rational numbers, supposing that integers of arbitrary precision are available. One drawback is that in adding exact rational endpoints, costly GCD computations are required. More significantly, huge numerators and denominators may be required to represent the endpoints. One remedy is to introduce some extra crudeness for the sake of efficiency and extend a given interval to the next proper superinterval whose endpoints are acceptably compact ratios. The other option is to represent the bounds by floating point numbers of some desired precision. The methods below do not depend on the choice of endpoint representation.

To define the basic interval arithmetic operations, let $X = [a, b]$, $Y = [c, d]$, $Z = [e, f]$ with extended real endpoints such that $a \leq b, c \leq d, e \leq f$; $-\infty$ and $+\infty$ can only appear as a lower and upper endpoint respectively. Addition ($+$) and multiplication ($\cdot$) between extended reals follow the usual rules.

$$[a, b] + [c, d] = [a + c, b + d]$$
$$[a, b] \cdot [c, d] = [\min\{ac, ad, bc, bd\}, \max\{ac, ad, bc, bd\}]$$

The space of all real intervals is a ringoid with respect to these two operations, as proven in [KuMi]. In other words,

$$
\begin{gathered}
X + Y = Y + X \\
\exists\, [0,0] = 0 : X + [0,0] = X \\
\exists\, [1,1] = 1 : X \cdot [1,1] = X \\
X \cdot [0,0] = [0,0] \cdot X = [0,0]
\end{gathered}
$$

and there exists a unique interval $[-1,-1] = -1$ which is the additive inverse of $[1,1]$, the multiplicative inverse of itself and distributes with respect to addition and multiplication. Both operations are associative but not distributive, in general.

Subtracting interval X is defined as adding $(-1) \cdot X$ and division by X is multiplication by $1/X$, although neither an additive nor a multiplicative inverse exists, since $X - X \neq 0$ and $X/X \neq 1$. For instance, $[0.5,1] - [0.5,1] = [-0.5,0.5] \neq [0,0]$. This is not a superficial observation, instead, it goes to the heart of interval mathematics and the cause of crudeness in interval arithmetic. A more formal discussion of this phenomenon is found in [NiZ]. By convention $\frac{1}{\infty} = \frac{1}{-\infty} = 0$ and $\frac{1}{0}$ is infinity with the sign dependent on the rest of the interval.

$$
\begin{gathered}
-[a,b] = [-b,-a] \\
a \cdot b \geq 0 \Rightarrow 1/[a,b] = [1/b, 1/a] \\
a < 0 \,\wedge\, b > 0 \Rightarrow 1/[a,b] = [-\infty, 1/a] \cup [1/b, \infty] \subset [-\infty, \infty]
\end{gathered}
$$

The very last inclusion is trivial and introduces some crudeness, but is required in order to adhere to our definition of interior intervals.

3 Existing Systems

This section briefly examines *Maple V* and *Mathematica 2.0* with respect to interval arithmetic as an illustration of the capabilities of certain widespread computer algebra systems. *Maple V* has an environment called `evalr` specifically defined for handling intervals of the form $[a..b, c..d, \ldots, y..z]$ where $a, b, c, d, \ldots, y, z$ can be of several types including complex numbers and extended reals. If they are real they presumably satisfy $a \leq b \leq c \leq d \leq \ldots \leq y \leq z$. The ordinary interval arithmetic is possible, but not free of bugs in our own current version of *Maple V*. An example is the reduction of $[a,b] \cdot [a,b]$ to $[a,b]^2$, which is not true in general.

$$[-1,2] \cdot [-1,2] = [-2,4] \neq [-1,2]^2 = [0,4]$$

Moreover, functions with complex or interval coefficients can be evaluated with interval arguments. However, the interval returned is usually crude, since the evaluation is done in a simplistic way. Here are two examples:

$$
\begin{gathered}
f(x) = x^3 - 2x^2 - 5x + 6 \text{ with argument } X = [0,1] \\
\textit{Maple V} \text{ Returns } [-1,7] \text{ Instead of } [0,6]
\end{gathered}
$$

$$
\begin{gathered}
f(x) = 2xe^x - \cos(x^2) \text{ with argument } X = [2,3] \\
\textit{Maple V} \text{ Returns } [28.5, 121.5] \text{ Instead of } [30.2, 121.5]
\end{gathered}
$$

Actually *Maple V* returns `syntax error` on the second example because of the incompatibility of types of the two exponents; but working through the two terms separately we obtain the interval above, to one decimal digit.

Mathematica 2.0 defines real intervals through the command `RealInterval`; it avoids the blunder above, but does not even assume that $X \cdot X = X^2$ in the symbolic case. Its evaluation procedures are less powerful than those of *Maple V*. For the first example, of a polynomial function, the answer is identical; yet, the second example does not produce anything useful. The system prints a long list of nested minimizations and maximizations that should define the endpoints, but does very little work towards even approximating those. Consider the following easier function, for which the answer again is not too enlightening.

$$f(x) = e^x \text{ with argument } X = [2, 3]$$
$$\textit{Mathematica 2.0} \text{ Returns } [\min\{e^2, e^3\}, \max\{e^2, e^3\}]$$

Both systems, when they do not fail, evaluate roughly along the lines of procedure NaivEval, to be described in the following section. The main feature of this approach is that it does not compute the derivative of the given function, thus has no information about its extrema. *Maple V* seems to include more information on trigonometric functions, for example, since in that case the extrema are identified without explicitly computing the derivative. Nonetheless, both systems provide unnecessarily crude answers, and our methods in the following section improve upon this situation.

4 Evaluation

4.1 Univariate functions

Given is a real continuous univariate function $f(x)$ together with a real interval X and the problem is to compute the value of the function when the variable x lies in this interval. The answer is in general a set $\{f(x)|x \in X\}$, assuming f is defined on X. We concentrate on continuous functions, for which this set is an interval, called the *function interval* and denoted by $f(X)$.

An algorithm that evaluates f at X must return an interval containing $f(X)$ in order to be *correct*. Furthermore, we distinguish a *tight* algorithm from a *crude* one by whether it returns exactly $f(X)$ or a proper superinterval of this. We call the output of an algorithm respectively tight or crude. Note that tightness is impossible unless both $\inf\{f(X)\}$ and $\sup\{f(X)\}$ are rational and the endpoints are represented by rationals. Below we describe a correct method, called ExtrEval, that is tight for most functions as well as easy to implement.

Let us start by considering the easier problem of evaluating $f(c)$ at some extended real number c. We assume that there exists routine PointEval that computes $f(c)$ to any desired precision or returns an interval containing it. In principle, this is a reasonable assumption for modern computer algebra systems. In addition, we shall need the tools for symbolic differentiation and equation solving.

As a warm-up, observe that, although $[\min\{f(a), f(b)\}, \max\{f(a), f(b)\}]$ is a correct and tight approximation to $f(X)$ for f monotonic over X, it is incorrect in general. Another attempt is to mimic the evaluation of the function at a real argument. We call the algorithm NaivEval and demonstrate its application to an example. Take

$$f(x) = x^3 - 2x^2 - 5x + 6 \text{ with } X = [0, 1]$$
$$X^3 = [0, 1],\ 2X^2 = [0, 2],\ 5X = [0, 5]$$
$$\texttt{NaivEval} \text{ Returns } [0, 1] - [0, 2] - [0, 5] + 6 = [-1, 7]$$
$$\text{However } f(X) = [0, 6]$$

NaivEval simply uses ordinary interval arithmetic to evaluate first the particular terms in f and then the entire expression and we shall assume that it is available as a subroutine. Note that, since the distributive laws do not hold for intervals, the outcome depends on the order of evaluation, which is arbitrary. In fact, different orders may produce different answers, all of which are correct but of varying crudeness. The reason for its crudeness is that the various occurrences of X in different terms are treated completely independently; for a formal exposition on this phenomenon refer to [HaC] and [RaRo]. There exist techniques to improve the naive approach, such as those in [DaFi], but we shall not examine them here.

We call our new procedure ExtrEval because it checks the extrema of the function f over the given interval $X = [a, b]$. The idea comes from the earlier observation that $f(a)$ and $f(b)$ are the endpoints of the interval $f(X)$, if f is either increasing or decreasing over X. So, we should only have to worry about the zeros of $f'(x)$ in X, which are obtained symbolically; if this is hard, the next section describes a method that returns intervals that may contain the zeros of a function. For non-differentiable functions, ExtrEval reduces to NaivEval.

We shall think of our algorithm as maintaining a pool of values that lie in $f(X)$ and, therefore, are the candidate endpoints for the function interval; we call this list CAND. Sometimes, instead of real values CAND is given interval approximations to them, in which case both endpoints are entered into the list. At the end, ExtrEval returns the minimum and maximum of all values in CAND.

```
 Input :   Function f : D → R, D ⊂ R, with real coefficients
           Interval X = [a,b] ⊂ R, a ≤ b
Output :   Interval Y = [m,M] ⊂ R, m ≤ M or UNDEFINED
```

```
ExtrEval :
     1. If X ⊄ D Then Return UNDEFINED
     2. Evaluate f(a), f(b) by PointEval; Put values in CAND
     3. Compute f'(x) symbolically
     4. If f'(x) exists, Then Goto (6)
     5. Evaluate f(X) by NaivEval; Obtain Y; Return Y
     6. Solve f'(x) = 0 over X; Obtain reals or intervals R_1,...,R_r
     7. Evaluate f(R_1),...,f(R_r) by PointEval or NaivEval;Put values in CAND
     8. Return min and max of CAND respectively as m and M
```

ExtrEval could be made more robust for the case that $X \not\subset D$ and $X \cap D \neq \emptyset$ by printing `PARTIALLY − UNDEFINED` and running on $X \cap D$. ExtrEval ignores the imaginary part of the function's value. To produce complex answers define f as a complex-valued function, let PointEval and NaivEval return complex intervals by applying special rules that produce complex intervals and modify step (8) to minimize and maximize endpoints along two orthogonal directions in the complex plane.

The correctness of the algorithm is not hard to prove. If the function interval does not lie between $f(a)$ and $f(b)$ then f must achieve some local, or global, minimum or maximum in the range of interest, namely X. ExtrEval finds these extrema and evaluates f at them. Note that saddle points, ie. even-multiplicity roots of f', can be ignored. Moreover, cases in which f tends to positive or negative infinity are not exceptional, since the returned interval has extended-real endpoints. ExtrEval is generally slower than NaivEval and its bottleneck is in solving f'.

For a large class of functions, including those that involve rational, exponential and logarithmic functions, the algorithm produces tight bounds as long as step (6) constructively solves for all roots of the derivative and no subroutine introduces any crudeness. Even when this happens, the slackness is introduced at secondary computations and is usually significantly smaller than that in NaivEval. Note that the crudeness increases as the width of X increases, which is a reasonable behavior.

Consider the previous example of a polynomial function defined over the reals

$$f(x) = x^3 - 2x^2 - 5x + 6$$

with $X = [0, 1] \subset D$. CAND contains $f(0) = 6, f(1) = 0$ and the values at the roots of

$$f'(x) = 3x^2 - 4x - 5$$

in X. But none of these roots lie in X, hence ExtrEval returns $[0, 6]$ which is tight.

Another example involving the exponential and a trigonometric function is the following.

$$f(x) = 2xe^x - \cos(x^2) \text{ with } X = [2, 3].$$

NewtonRoot of the next section may be used to verify that no roots of $f'(x)$ lie in X, hence the only values in CAND are $f(2) = 30.2$ and $f(3) = 121.5$, and the answer is tight.

We have considered the basic case of univariate functions with real coefficients. Our approach can be generalized to functions with interval coefficients as long as no intervals appear as exponents. PointEval would perform interval arithmetic and essentially reduce to NaivEval. ExtrEval is not substantially altered, since the derivative of a function can be naturally defined.

4.2 Multivariate functions

Let $f(\mathbf{x})$ be an n-variate continuous differentiable function defined on some subset D of $\mathbf{R}^n$, whose coefficients and value all lie in $\mathbf{R}$; bold-face lower-case letters denote real vectors, in this case of dimension n. The problem is again to evaluate this function when each variable x_i for $1 \leq i \leq n$ takes values from a given real interval $X_i = [a_i, b_i]$. Let the real interval vector whose i-th entry is X_i be written $\mathbf{X}$. Then the tight interval of function values is denoted by $f(\mathbf{X}) = \{f(\mathbf{x}) | \mathbf{x} \in \mathbf{X}\}$.

It is clear that for all i, $f(\mathbf{a})$ and $f(\mathbf{b})$ belong to $f(\mathbf{X})$, where $\mathbf{a}$ and $\mathbf{b}$ respectively have i-th entry a_i and b_i. Furthermore, every combination of n endpoints, each corresponding to a different variable and thus to a different interval, is an argument at which f has a value within $f(\mathbf{X})$. These 2^n points define the n-dimensional rectangle in $\mathbf{R}^n$ that contains exactly all values taken by $\mathbf{x}$. If f is monotonic along every coordinate, then it suffices to consider only those 2^n values; their minimum and maximum are the two endpoints of $f(\mathbf{X})$.

In general, f is not monotonic over $\mathbf{X}$, in at least one variable. Therefore, our algorithm will have to find the points at which the value of f takes extreme values. Once the partial derivatives are symbolically computed, we need to solve the system of equations defined when all of them vanish simultaneously and then evaluate f at these roots. But we are not done yet! Unlike the univariate case, the argument lies in a region bounded by hyperplanes; the behavior of f on these hyperplanes is not completely determined by the endpoints, which are the corners of the rectangle. More specifically, f may attain a maximum or minimum at the region's boundary, namely on some hyperplane $x_i = a_i$ or $x_i = b_i$.

The extended algorithm ExtrEval checks the behavior of f on the boundary hyperplanes explicitly. Setting $x_i = a_i$ and then $x_i = b_i$, it looks for solutions to the system of the $n - 1$ partial derivatives excluding the one with respect to x_i; the values at these $2n$ points are also included in list CAND of candidate endpoints for $f(\mathbf{X})$. The generalization of PointEval and NaivEval to multivariate functions is straightforward.

Input : Function $f : D \to \mathbf{R}, D \subset \mathbf{R}^n$, with real coefficients
Interval $\mathbf{X} = [[a_1, b_1], \ldots, [a_n, b_n]] \subset \mathbf{R}^n,\ a_i \leq b_i\ \forall 1 \leq i \leq n$
Output : Interval $Y = [m, M] \subset \mathbf{R}, m \leq M$ or UNDEFINED

`ExtrEval` :

1. If $\mathbf{X} \not\subset D$ Then Return UNDEFINED
2. Let $\mathbf{c_k}$ be an $n-$dimensional vector,
   the $i-$th entry being a_i or b_i, for some $1 \leq i \leq n$
3. Evaluate $f(\mathbf{c_k})$, $1 \leq k \leq 2^n$ by PointEval; Put values in CAND
4. Compute partial derivative $f_{x_i}(\mathbf{x})\ \forall i$ symbolically
5. If all f_{x_i} exist Then Goto (7)
6. Evaluate $f(\mathbf{X})$ by NaivEval; Obtain Y; Return Y
7. Solve the system $f_{x_i}(\mathbf{x}) = 0,\ 1 \leq i \leq n$, over $\mathbf{X}$ to obtain $\mathbf{R_1}, \ldots, \mathbf{R_r}$
8. Evaluate $f(\mathbf{R_1}), \ldots, f(\mathbf{R_r})$ by PointEval or NaivEval; Put values in CAND
9. For each i
   Let $x_i = a_i$
   Solve the system $f_{x_j}(\mathbf{x}) = 0, j \neq i$, over $\mathbf{X}$ to obtain $\mathbf{A}_1, \ldots, \mathbf{A}_{t_i}$
   Evaluate $f(\mathbf{A}_1), \ldots, f(\mathbf{A}_{t_i})$ by PointEval or NaivEval; Put values in CAND
10. Repeat step (9) with b_i instead of a_i
11. Return min and max of CAND respectively as m and M

The multivariate version of ExtrEval is also easy to implement on modern systems and produces tight results on a large class of functions, including all polynomials. The sharpness of its results may be compromised by the crudeness of the ancillary routines; this effect is minimized as the involved intervals grow smaller. The asymptotic complexity of ExtrEval is at least 2^n because of the evaluations at all corner points of the given region in step (3). Nevertheless, the bottleneck for most cases occurs at solving systems of equations. We conclude the discussion with an example.

$$f(x) = x^2y + x^2 - 3xy + 2y + 5,\quad \mathbf{X} = [[0, 2], [-1, 1]]$$

`ExtrEval` Returns tight $[3, 9]$

`NaivEval`, *Maple V* and *Mathematica 2.0* Return $[-7, 21]$

4.3 Complex intervals

In this section we sketch the generalization of our evaluation techniques to complex intervals. We adopt an analogous representation as for complex numbers: $\mathbf{X} = [[a_1, b_1] + i[c_1, d_1], \ldots, [a_n, b_n] + i[c_n, d_n]]$. The basic operations of interval arithmetic in the complex plane are natural extensions of those definitions in section 2. It is an easy matter to generalize procedures PointEval and NaivEval to approximate function interval $f(\mathbf{X})$ when $\mathbf{X}$ has complex endpoints. We restrict attention to functions that can be expressed as the sum of a real and an imaginary part, namely all f such that for every $\mathbf{x} \in \mathbf{C}^n$ $f(\mathbf{x}) = g(\mathbf{x}) + ih(\mathbf{x})$, where g and h can be any continuous differentiable functions. The applicability of ExtrEval can be extended to such functions by regarding them as maps from $\mathbf{R}^{2n}$ to $\mathbf{R}^2$. For $\mathbf{x} = \mathbf{y} + i\mathbf{z}$, we shall write $F(\mathbf{y}, \mathbf{z}) = [g(\mathbf{y}, \mathbf{z}), h(\mathbf{y}, \mathbf{z})]$. Then, a more general solving procedure will be called to isolate the roots of systems of equations.

5 Solution of equations

We wish to compute subintervals that may contain the roots of a given equation $f(x) = 0$ within some given interval X; f is continuous differentiable, has real coefficients and is defined on $D \subset \mathbf{R}$. There exists an extensive literature on this subject, mostly concerned with partition methods to limit the width of candidate intervals combined with iterative methods to further sharpen these intervals into approximations of the roots. Our approach is based on Newton iteration and draws from several of these theoretical works; its simplicity makes it easily implementable, yet it is powerful enough to produce fast converging intervals in the vast majority of interesting cases.

As summarized in [Mo] and [Wo], Newton iteration with intervals is well-defined, provably correct and guaranteed to converge quadratically once a *safe* starting interval is located. An interval is safe with respect to an iterative method if, in addition to containing a root, the execution of the method is guaranteed to produce interval approximations to the root, of arbitrary precision. A more detailed presentation of safe starting regions is found in [Jo].

We combine the purely iterative procedure with the initial phase of locating a starting interval. Moreover, we give away some of the more costly computations for the sake of simplicity. The main idea is for any real interval X first to check whether f has any roots in it and for this, our algorithm computes $f(X)$. If X includes some root, then $f(X)$ contains 0; this is a necessary condition which allows us to exclude certain subintervals.

Newton's method with real numbers computes a sequence of real approximations to the root according to the formula

$$x_{k+1} = x_k - f(x_k)/f'(x_k).$$

There are two conditions. First, that the original approximation is sufficiently close and second that the derivative does not vanish. In interval mathematics, Newton's method defines a sequence of intervals. The given interval is partitioned every time an iteration fails to narrow it down. For every new subinterval X we make sure that the necessary condition $0 \in f(X)$ holds, otherwise X is rejected from further consideration.

The problem of a vanishing derivative corresponds here to the derivative taking values in an interval which contains zero. Then, inverting this interval gives a new interval with at least one end-point at infinity. If there is exactly one such endpoint we decide to carry on with the computation. If the interior of $f'(X)$ contains zero, then its inverse is the entire real line and there is no point in continuing a calculation that will produce the entire line as final answer, so we choose to partition. These are steps (11) and (12) below. A different tack would be to consider the implementation of exterior intervals [Ka].

There are two parameters that protect the algorithm from a combinatorial explosion. Depending on the desired precision, we limit the number of iterations and, second, we limit the extent of partitioning within a certain interval by keeping an appropriate count. The partition bound is smaller and smaller as further subintervals are created and is controlled by `partition` − `count` in the algorithm. It is indispensable when an infinite number of roots lie in a small interval.

We suppose that the standard functions of a modern computer algebra system are available, as well as PointEval, NaivEval and ExtrEval of the previous section. The output is a list of all subintervals that the algorithm ever considered, together with an indication, for each one, of whether it may or does not contain a root.

Input : Function $f : D \to \mathbf{R}$, $D \subset \mathbf{R}$, with real coefficients
Interval $X \subset \mathbf{R}$
Output : List of subintervals partitioning X, each marked with YES or NO

NewtonRoot :

1. Compute $f'(x)$ symbolically
2. Initialize INTERVALS to contain $X_0 = X \cap D$
3. Let $i = 0$
4. Goto (9)
5. Partition : Substitute X_i in INTERVALS by A_i, B_i
6. Adjust the partition − count
7. Let $X_i = A_i$; Goto (9)
8. If there is a next interval in INTERVALS
   Then call it X_j
   Else Return
9. Compute $f(X_i)$ by ExtrEval or NaivEval
10. If $0 \notin f(X_i)$ then Output $< X_i, \mathtt{NO} >$; Goto (8)
11. Compute $Z = f'(X_i)$ by ExtrEval or NaivEval
12. If $0 \in \mathrm{open}Z$ then
    If partition − count small then Goto (5)
    Else Output $< X_i, \mathtt{YES} >$; Goto (8)
13. Newton iteration : Compute $Y = f(\mathrm{mid}X_i)$ by PointEval
14. Compute $\tilde{X}_{i+1} = \mathrm{mid}X_i - Y/Z$
15. If $X_i \cap \tilde{X}_{i+1} = \emptyset$ then return $< X_i, \mathtt{NO} >$
16. If $X_i \subset \tilde{X}_{i+1}$ then
    If partition − count small then Goto (5)
    Else Output $< X_i, \mathtt{YES} >$; Goto (8)
17. Let $X_{i+1} = X_i \cap \tilde{X}_{i+1}$
18. If X_{i+1} has desired precision then Output $< X_{i+1}, \mathtt{YES} >$
19. Increment i; Goto (8)

A few implementation remarks are in order. To check if zero lies in a closed interval $[a, b]$, ignoring the case that it is one of its endpoints, function `open` returns the corresponding open interval (a, b). Function `mid` returns the midpoint of an interval, in other words the real number $(a + b)/2$. An important observation from the theoretical works referenced at the beginning of the section is that it is not necessary to use the midpoint and that any arbitrary point in the interior of the interval will do; this is crucial when one of the endpoints is at infinity.

Data structure INTERVALS lists the subdivisions of original interval X and can be implemented as a linked list. A pointer points to the current interval, thus keeping track of what remains to be done and where to make any further partition, if need be. When the program finishes running on some interval, it can find on this list the next one to examine.

Let us illustrate with an example. Suppose that the problem is to compute a floating-point positive root with precision 5 decimal digits, of the equation $f(x) = 0$, where

$$f(x) = x^2 + x - 1, \quad \text{and} \quad X = [0, \infty].$$

NewtonRoot first computes

$$f'(x) = 2x + 1, \quad \text{and} \quad X_0 = [0, \infty].$$

Note that the midpoint of X_0 is not defined. We pick 100 as a point in $\mathtt{open}X_0$; simple observations on $f(x)$ could produce better bounds on the magnitude of roots and hence a more efficient starting point. Assume that the program represents interval endpoints with a precision of 2 decimal digits only, as long as this is enough to distinguish the two points.

$$
\begin{aligned}
X_1 &= [0, 100] \\
X_2 &= [0, 37.32] \\
X_3 &= [0, 13.83] \\
X_4 &= [0, 5.04] \\
X_5 &= [0, 1.81] \\
X_6 &= [0.19, 0.75] \\
X_7 &= [0.52, 0.75] \\
X_8 &= [0.6168, 0.6193] \\
X_9 &= [0.618033, 0.618034]
\end{aligned}
$$

The only positive solution of the equation is 0.618033989.

One significant improvement would be to apply Rolle's theorem for isolating distinct roots. Assuming that some higher derivative is well-behaved and it is easy to find its zeros, we could work our way backwards and obtain good starting intervals for the roots of f.

NewtonRoot indicates the intervals that may contain solutions, but special care must be applied to distinguish the case of double roots. One approach is to further partition the intervals of interest and perform various tests in the hope that we shall manage to isolate the roots. Another alternative is to consider the application of resultants or polynomial GCD methods for the given function and its derivative. The resultant test is easier to implement and is discussed, in its traditional context, in [DaSiTo]. The value of the resultant here is a real interval and the test is whether this interval contains zero or not. In the latter case no double roots exist; otherwise, the test is inconclusive.

It is possible to extend this approach in several directions. First, for solving simultaneous systems of equations, it is possible to adhere to the same iterative approach, as shown in [Ka] and [Wo]. Second, for obtaining complex roots, as discussed in [NiZ], the main modification is that the interval of interest becomes a rectangle in the complex plane. Lastly, NewtonRoot can be applied in principle to functions with interval coefficients, although convergence is no longer guaranteed, see [NiT].

6 Conclusion

We have presented practical methods for several important problems in the context of real and complex interval arithmetic, including the basic operations, multivariate function evaluation at interval arguments and root approximation with intervals. The evaluation procedures extend to functions with interval coefficients. The validity of the algorithms employed has been theoretically established in articles found in the extensive bibliography on the subject; here, we have aimed at achieving a balance between efficiency and simplicity. Our methods improve substantially on those built-in to *Maple V* and *Mathematica 2.0.*

We have hinted at certain improvements for our methods. Especially in what concerns our root-finding technique, the implementation of external intervals may prove useful. Secondly, looking at higher derivatives may increase the overall efficiency of this procedure. We have talked of a twofold motivation behind interval arithmetic, namely in bounding the error in computer arithmetic and in deciding the sign of constant expressions without explicitly evaluating them. There exist various directions for future development of this work, since the richness of theoretical work implies the applicability of interval mathematics in solving a variety of other problems.

Acknowledgment

The authors acknowledge useful comments from Prof. W. Kahan. The first author acknowledges financial support through a David and Lucile Packard Foundation Fellowship.

References

[DaFi] Davenport J.H. and H.-C. Fischer, Manipulation of Expressions, in *Improving Floating-Point Programming*, ed. P.J.L. Wallis, J. Wiley & Sons, Chichester, 1990.

[DaSiTo] Davenport J.H., Y. Siret and E. Tournier, *Computer Algebra: Systems and Algorithms for Algebraic Computation*, Academic Press, London, 1988.

[FaYa] Fateman R.J. and T.W. Yan, Computation with the extended rational numbers and an application to interval arithmetic, Manuscript, UC Berkeley, 1991.

[HaC] Hansen E., On the centred form, in *Topics in Interval Analysis*, ed. E. Hansen, Oxford University Press, pp. 102-106, 1969.

[HaG] Hansen E., A generalized interval arithmetic, *Interval Mathematics, Proc. Intern. Symp, Karlsruhe, May 1975*, ed. K. Nickel, Springer-Verlag, Berlin, pp. 7-18, 1975.

[Jo] Jones S.T., Locating safe starting regions for iterative methods: A heuristic algorithm, in *Interval Mathematics 1980*, ed. K. Nickel, Academic Press, New York, pp. 377-386, 1980.

[Ka] Kahan W.M., *A more Complete Interval Arithmetic*, Lecture notes for a summer course at the Univ. of Michigan, Ann Arbor, June 1968.

[KuMi] Kulisch U.W. and W.L. Miranker, *Computer Arithmetic in Theory and Practice*, Academic Press, New York, 1981.

[Mo] Moore R.E., New results on nonlinear systems, in *Interval Mathematics 1980*, ed. K. Nickel, Academic Press, New York, pp. 165-180, 1980.

[NiT] Nickel K., Triplex-Algol and its applications, in *Topics in Interval Analysis*, ed. E. Hansen, Oxford University Press, pp. 10-24, 1969.

[NiZ] Nickel K., Zeros of polynomials and other topics, in *Topics in Interval Analysis*, ed. E. Hansen, Oxford University Press, pp. 25-34, 1969.

[RaRo] Ratschek H. and J. Rockne, *Computer Methods for the Range of Functions*, Ellis Horwood Ltd, Chichester, 1984.

[Wo] Wolfe M.A., Interval methods for algebraic equations, in *Reliability in Computing*, ed. R.E. Moore, Academic Press, Boston, pp. 229-248, 1988.