Interval Computations
No 1, 1993

NO PATHOLOGIES FOR INTERVAL NEWTON'S METHOD

Yılmaz Akyıldız* and Mohammad I. Al-Suwaiyel

Interval methods for enclosing zeros of nonlinear functions are compared with alternate techniques on concrete examples and their inherent differences are discussed.

ИНТЕРВАЛЬНЫЙ МЕТОД НЬЮТОНА СВОБОДЕН ОТ ПАТОЛОГИЙ

И.Акилдиз, М.И.Аль-Сувайель

Интервальные методы для локализации нулей нелинейных функций сравниваются на конкретных примерах с альтернативными подходами; обсуждаются различия, присущие этим методам.

1. Introduction

In their recent article, "*Pathological Functions for Newton's Method*", in The American Mathematical Monthly [1], the authors explicitly show that the following claim in a widely used numerical analysis text [2] is true:

* Supported by a sabbatical grant from King Fahd University of Petroleum & Minerals, Dhahran, Saudi Arabia.

In the solution of equations by numerical methods, a commonly used stopping criterion

$$|x_{n+1} - x_n| < \epsilon, \tag{1}$$

where x_n is the nth term of the sequence generated by the method, and $\epsilon > 0$ is the tolerance, can fail.

They show that

i) The function

$$f(x) = \frac{c\ exp(-\frac{1}{2}(x^2 + x\sqrt{x^2+1}))}{\sqrt{x + \sqrt{x^2+1}}}$$

has no real roots, but nevertheless its Newton's iteration generates a sequence for which (1) is satisfied for any ϵ.

ii) The function

$$h(x) = \sqrt{[3]}xe^{-x^2}$$

appears to converge where there are no roots, but it has a real root, zero, to which Newton's method will never converge, (unless zero is the starting point). Functions f and h are useful as concrete examples demonstrating that stopping criterion (1) for the solution of equations by usual iterative methods can fail.

In this note, we will try to explain why such pathologies cannot occur if *interval arithmetic* is employed. We shall first review some of the most important concepts in interval mathematics, and then apply interval methods to the functions f and h above. We shall also demonstrate further drastic differences between ordinary and interval Newton's methods on two more simpler functions. Such interval computations on individual functions will be more than just numerical checks, as applied interval methods can give results that have *mathematical certainty* since the effects of round-off error are fully taken into account. That is, properly designed and implemented interval methods are *totally reliable* in the sense that they give results with mathematical certainty.

An interval algorithm will produce a list of solutions whose elements are small intervals of uncertainty. If the proper algorithm is correctly implemented with *directed rounding* (as it can be done, for example, in

Mathematica Version:2.2 and Maple V Release 2), completion of this algorithm constitutes a computational but *mathematically rigorous proof* that all solutions are within the intervals given in the list. For a list of other uses of interval arithmetic in mathematical analysis and theoretical physics see [3], which also has some interesting philosophical remarks on *"computer proof"* in its conclusion.

2. Elementary facts about interval arithmetic

First, let us review the rudiments of how computers perform arithmetic. Standard computing environments provide two ways of working with numbers: *exact arithmetic* with integer and rational numbers and *inexact arithmetic* with *floating point numbers*. There is little that needs to be said about exact arithmetic. Once a canonical form for the numbers is chosen, there is only one right answer. Symbolic computations, such as word processing, are done with integers and this is why they are completely dependable, error-free operations. Since the computations in mathematics deal in general with real numbers and not just with integers and rationals, exact arithmetic is not directly applicable to them, and we will accordingly concentrate on floating point arithmetic.

Floating point numbers are manipulated and stored in a sign-exponent-fraction representation with a fixed number of digits available for exponent and fraction. The details of the representation vary, but any given format is capable of representing only finitely many numbers; we shall refer to these as the *representable* numbers in that format. The most common way to do inexact arithmetic is to use fixed precision floating point arithmetic. In this system a number is stored as a mantissa and an exponent. The mantissa always contains the same number of bits and, since there is no way to tell how precise a number is, all numbers are treated as exact. The problem with this is that the arithmetic is necessarily wrong. For example, if we are working within a system where all numbers have five significant digits, then $1.2345 + 0.012345$ cannot be correctly represented. In order to represent the answer, it must be rounded to five significant digits; the effect of this rounding is to make the arithmetic wrong. Elementary arithmetic operations with representable operands often produce results with too many digits to be representable. When this happens, what is normally done is to "round off" the result, i.e., to return a representable number approximating the exact result. In

clean computing environments, the returned result may indeed always be exact result "correctly rounded", i.e., that representable number which is nearest to the exact result, but this is unfortunately far from being universally the case. It is also possible for a result to be too big or too small to be representable with the limited range of exponents available. (Some computer algebra systems can satisfactorily handle such situations with their *arbitrary-precision arithmetic*).

The primary way that errors get introduced in the inexact arithmetic is by trying to represent numbers that have no finite representation in the system. For example the number $\frac{1}{3}$ cannot be represented exactly as a floating point number in the binary or decimal systems. The advantage of fixed precision floating point arithmetic is that it is easy to implement in hardware (making it very fast) and behavior of the errors thus introduced is well understood.

Each individual arithmetic operation, then can-and usually will-introduce some amount of *round-off error*. It is thus possible, in principle, to apply elementary methods of error propagation to derive strict bounds on the error in the result of any given sequence of arithmetic operations. However, every numerical analyst knows that, in practice, except for simple computations or ones with particular transparent structure, these theoretical error estimates are usually too complicated to be feasible.

Interval analysis, by contrast, provides a tool for estimating and controlling these errors automatically. Instead of approximating a real value x by a machine number, the usually unknown real value x is approximated by an interval X having machine number upper and lower boundaries. The interval X contains the value x. The width of this interval may be used as a measure for the quality of the approximation. The calculations therefore have to be executed using intervals instead of real numbers and hence the real arithmetic has to be replaced by interval arithmetic.

For example, consider the representation of $\frac{1}{3}$ using four significant figures. As an alternative to the nearest rounded result 0.3333 we can use the interval $[0.3333, 0.3334]$. This may seem less precise than the form we are used to, but in fact it is more truthful since it reveals the level of uncertainty that is present. It also shows that 0.3333 is an underestimate of the exact value. Hence an interval representation provides information about the accuracy of a computed result which a single model number, by its nature, cannot do.

In short, an *interval number* is just a closed real interval $[a, b]$, i.e., it consists of the set $\{x : a \leq x \leq b\}$ of real numbers between and including the end points a and b. A real number x is equivalent to a (degenerate) interval $[x, x]$. The end points a and b of a given interval $[a, b]$ may not be representable on a given computer. In such a case, a is rounded to the largest machine number which is less than or equal to a, and b is rounded to the smallest machine number which is greater than or equal to b. This process is called *outward rounding.*

So, the idea is to carry through the computation strict upper and lower bounds for all quantities encountered, i.e., intervals guaranteed to contain the corresponding exact quantities. The end points of these interval are representable numbers. To propagate the error bounds a step at a time, it is only necessary to have available procedures for "doing elementary operations, $op \in \{+, -, *, /\}$, on intervals". Let $A = [a_1, b_1]$ and $B = [a_2, b_2]$ be two intervals. We define $A\, op\, B = \{x\ op\ y : x \in A\ and\ y \in B\}$. To see what this means, let us consider the case of multiplication. A procedure for multiplying intervals means one which, given two intervals $[a_1, b_1]$ and $[a_2, b_2]$ with representable end points, produces a third interval $[a_3, b_3]$, again with representable end points, such that

$$x \in [a_1, b_1]\ and\ y \in [a_2, b_2] \Longrightarrow x * y \in [a_3, b_3].$$

Here is one way in which a procedure for multiplying intervals can be constructed. First form the four exact products $a_1 * a_2, a_1 * b_2, b_1 * a_2, b_1 * b_2$. To construct (the best possible) a_3, find the smallest of these exact products and round down to the next smaller representable number. Similarly, to construct b_3, find the largest and round up. This is called *directed rounding.* In fact, all operations can be defined in terms of addition, subtraction, multiplication, and division of the end points of the intervals, although multiplication and division may require comparison of several results, (thus making interval computations expensive). The result of these operations is a finite interval except when we compute A/B and $0 \in B$. In that case, we need *extended interval arithmetic*, which is supported by, e.g., *Mathematica* Version: 2.2, to get two semi-infinite intervals or else the whole real line. For example, *Mathematica* will give

$$Interval[\{1, 2\}]/Interval[\{-3, 4\}] = Interval[\{-\infty, -\frac{1}{3}\}, \{\frac{1}{4}, \infty\}];$$

where $Interval[\{a, b\}]$ stands for the closed interval [a, b] on the real line.

Interval mathematics can be said to have begun with the appearance of R.E. Moore's book *Interval Analysis* in 1966, [4]. According to Hansen [5], one of the pioneers of the subject, *"the introduction of interval analysis is the single most important advance in numerical analysis other than the advent of the modern computer and high-level programming language."*

Around mid 80's the Institute of Electrical and Electronics Engineering (IEEE) has introduced a standard for floating point arithmetic which provides, among many other things, facilities making it easy to construct interval arithmetic procedures which always produce the best possible results within the limitations imposed by the set of representable numbers. Floating point environments conforming to this standard are becoming more widely available. This development promises to make good-quality and efficient interval arithmetic much more accessible than in the past. Meanwhile, hardware companies, such as Sun and Cyrix, may well soon include micro-programmed interval arithmetic as a new data type in their forthcoming products. This would provide very fast execution times for interval algorithms, say much less than twice those of floating-point arithmetic. Using such an approach to computing, we can provide answers of rigorously guaranteed accuracy in a comparable time it takes to compute in the ordinary way.

The problem of bounding the range of a real-valued function is probably the central problem of interval analysis. Many general and special purpose algorithms are known in this context [6]. The following property is essential for interval evaluation of an arithmetic function $f(x)$, [5]:

The fundamental theorem of interval analysis: *Provided the value of x is replaced by the corresponding interval operations and operations on standard functions, then one gets the so-called* **interval extension** *F of f; additionally, the following important inclusion is valid:*

$$R(f, X) := \{f(x) : x \in X\} \subset F(X).$$

Essentially, this means that it is possible to compute an interval extension F of f by the same operations as evaluating $f(x)$ using real arithmetic. What we make sure is that outwardly rounded interval arithmetic is used throughout the computations. This assures that the computed result contains the range, $R(f, X)$, despite the difference of rounding errors.

We would like to emphasize that the value of an interval extension of a function is not precisely the range of the function over its interval operand, but only contains this range. A minor rearrangement in a formula can make a major difference in the sharpness of the estimates obtained by applying interval arithmetic to that formula. A general principle is that it is advantages to write expressions with as few occurrences of individual variables as possible.

As a numerical example, let $f(x) = x^2 - 2x + 2$ and $Y = [0, 1]$. Here are three interval extensions for this function:

$$F_1(X) = X^2 - 2X + 2,\ F_2(X) = X(X-2) + 2,\ F_3(X) = (X-1)^2 + 1.$$

Clearly,

$$F_3(Y) = [1, 2] \subset F_2(Y) = [0, 2] \subset F_1(Y) = [0, 3].$$

We note that F_3 gives the exact range.

It is still an unsolved problem how to construct interval extensions whose values differ from the range as little as possible. For discussion of efficient ways of formulating interval extension see [6,7]. Concepts of directed rounding and machine interval arithmetic are carefully presented in [8]. Also, not all programs doing floating point computations can be translated in a straightforward way into useful interval arithmetic programs producing strict bounds on the results. Thus, we are faced with the challenge of writing efficient interval arithmetic programs, [9].

3. Interval methods for inclusion of zeros

We consider the problem of finding *all* the roots of a continuously differentiable scalar function $f(x)$ in a given finite, closed interval $X \subset \mathbf{R}$. It is required only that there exists an interval evaluation for the function f in the interval X. Interval methods allow us to find a set of intervals of smallest possible width such that each interval includes one (or more) zeros of f. As it is explicitly shown in [1], adequate stopping criteria for non-interval methods can be unreliable or difficult to devise. On the other hand, natural stopping criteria exists for interval iterations. One can simply iterate until either the bounds are sufficiently sharp or no further reduction of interval bounds possible. There are various interval methods

for the root finding problem. (For a *Mathematica* implementation of interval bisection method see [10]). In this article we shall consider the interval Newton's method which has so many remarkable properties.

Interval arithmetic with directed rounding does not involve deep concepts, but it can be quite powerful. For example, if interval arithmetic with directed rounding is used to compute an interval extension F of f, and $F([a,b])$ does not contain zero, then this is a rigorous proof (regardless of the machine word length, etc.) that there is no root of f in $[a,b]$.

As a numerical example, let $f(x) = 6x(2-x) - 10$ and $Y = [0,1]$. We can take $F(X) = 6X(2-X) - 10$. Since $0 \in f(Y) = [-10, 2]$ we cannot conclude anything. But, bisection of Y into $U = [0, 1/2]$ and $V = [1/2, 1]$ yields $F(U) = [-10, -4]$ and $F(V) = [-7, -1]$, neither of which containing zero, thus proving that f has no zero in Y. In passing, we would like to mention that since $f(0) * f(1) > 0$, the number of roots in the interval $[0,1]$ is either zero or even. On the other hand, the interval method above gives a definitive answer and (computationally) proves that f has no zero in Y.

Example 1: Let us apply the above argument to the function f in the introduction. We need to consider only its numerator, $exp(-\frac{1}{2}(x^2 + x\sqrt{x^2+1}))$. Since the exponentiation and the term in the exponent are strictly increasing functions, a proper implementation of interval mathematics will bound the image by the values of the function at the boundary points, which are both positive numbers. Thus, the image of any (bounded) interval under this expression will not contain zero, i.e., f has no zeros on the real line. For example, *Mathematica* gives

$$In[\,] := Exp[-(x^2 + xSqrt[x^2+1])/2]/.x- > Interval[\{-10.^4, 10.^4\}]$$
$$Out[\,] = Interval[\{5.024596\ 10^{-43429449}, 1.59858686\ 10^{21714724}\}]$$

3.1. Interval Newton's methods

The interval Newton's method was first derived by Moore in [4] as follows: The mean value theorem says

$$f(x) - f(x^*) = (x - x^*)f'(\xi),$$

where ξ is some point between x and x^*. If x^* is a zero of f, then

$$x^* = x - \frac{f(x)}{f'(\xi)}.$$

Letting $\xi = x$ gives the ordinary Newton's method. Let X be an interval containing both x and x^*. Clearly $\xi \in X$. Let $F'(X)$ be an interval extension of $f'(x)$. Thus, we have $f'(\xi) \in F'(X)$. Hence, $x^* \in N(x, X)$, where

$$N(x, X) = x - \frac{f(x)}{F'(X)}.$$

By taking $x = middle(X)$, the midpoint of X, we consider the algorithm

$$X_{n+1} = X_n \cap N(X_n),$$

where

$$N(X) = middle(X) - \frac{f(middle(X))}{F'(X)}.$$

This is called the *interval Newton's algorithm.* It can also be generalized to higher dimensions. (It is not necessary to choose x to be the midpoint of X, any other value in X will also do). Since division is used, *extended interval arithmetic* should be employed in case $F'X)$ contains zero. (Actually, there is a way of getting around this because in practice such an unbounded interval is intersected with a finite interval).

The interval Newton's algorithm has the following remarkable reliability and efficiency properties:

1. If an interval X_0 contains a zero x^* of $f(x)$, then so does X_n for all $n = 0, 1, 2, \ldots$. Furthermore, the intervals X_n form a nested sequence converging to x^*. That is, every zero of f in the initial interval X_0 will always be found and correctly bounded.
2. If $0 \notin F'(X_n)$, at least half of X_n is eliminated in the next step. That is, convergence can be rapid even when width of X_n is large.
3. If $X_n \cap N(X_n)$ is empty, then there is no zero of f in X_n. That is, if there is no zero of f in X_0, the algorithm will automatically prove this fact in a finite number of iterations.
4. If $N(X) \subset X$, then there is a zero of f in X. That is, the existence of a zero of f in a given interval can also be proved automatically without extra computing. Furthermore, it is guaranteed that the interval Newton's iteration will converge to it.

The proofs of above can be found in [4]. We would like to add that, in practice, when rounding occurs, an interval containing $N(X_n)$ will be computed. Hence, even with rounding, all the properties above will still be applicable. This is why our computations by the computer will be more than just numerical checks but actual proofs, (provided, of course, the algorithms are correctly implemented).

Since interval computations cannot, in general, be performed by hand and since the present day computer algebra systems do not provide symbolic interval computations, we will have to work with numerical examples on specific problems. Let us now apply the interval Newton's method, with the help of *Mathematica*, to the function h in the Introduction.

Example 2: $h(x) = \sqrt{[3]}xe^{-x^2}$ and

$$h'(x) = \frac{1}{3e^{x^2}x^{2/3}} - \frac{2x^{4/3}}{e^{x^2}}.$$

When we start the interval Newton's method, for example, with $Interval[\{1., 5.\}]$ we get

$$\{Interval[\{1., 3.\}],\ Interval[\{3.00003, 5.\}]\}.$$

Thus the initial interval is mapped here into the union of two disjoint intervals, (because the derivative h' contains a zero in the interval). Iterating these two intervals and their descendants further ends up with an empty set after 4 steps, thus proving that h has no roots in $Interval[\{1., 5.\}]$. The same would be the case on any interval not containing zero. On the other hand, starting with $Interval[\{-3., 2.\}]$ produces

$$\{Interval[\{-3., -0.57278\}],\ Interval[\{-0.5, 2.\}]\}.$$

Ten more iterations give $Interval[\{-0.00195313, 0.000488281\}]$. Further iterations will get closer and closer to the origin. A similar thing would be the case when any other interval containing zero is used to start with.

The following two simpler examples will further illustrate more drastic differences between the ordinary and interval Newton's methods, [11] .

Example 3: Consider $f(x) = 2.001 - 3x + x^3$. We have $f'(x) = -3+3x^2$. We can use the interval extension $F'(X) = -3+3X^2$. If we take

$X_0 = Interval[\{-3, 3\}]$ and apply the interval Newton's iteration we obtain two disjoint intervals, $\{Interval[\{-3, -0.083375\}]$, $Interval[\{0.667, 3\}]\}$, (because $F'[Interval[\{-3, 3\}]] = Interval[\{-3, 24\}]$ contains a zero). From this point on we can continue to produce more sequences of intervals using the interval Newton's algorithm beginning with either of the intervals in the set above. If we apply the interval Newton's iteration to the second interval, the sequence of intervals generated terminates after 7 steps with the empty interval because there is no zero of $f(x)$ on the positive x-axis. If we start with the first interval, $Interval[\{-3, -0.083375\}]$, then, we find a nested sequence of intervals terminating with
$Interval[\{-2.00011, -2.00011\}]$ after 5 steps.

In contrast, the ordinary Newton's method, $x_{n+1} = x_n - f(x_n)/f'(x_n)$, generates a very erratic sequence of values unless the initial point x_0 is less than -1. For example, $x_0 = 0$ produce the sequence $\{0.667, 0.845187, 0.925925, 0.965774, 0.987941, 1.00789, 0.9829, 1.0013, 0.873019, 0.939346, 0.972823, 0.992691, 1.01924, \dots\}$. What has happened here is that the ordinary Newton's method got hung up, oscillating near a place where a local extremum with a very small nonzero value. On the other hand, the interval Newton's method will reject such a region as not containing a zero by producing an empty intersection after a finite number of steps.

Example 4: An even more drastic difference between the interval and ordinary Newton's methods is illustrated by the function $g(x) = x^{\frac{1}{3}}$. This is a simple enough expression for which symbolic interval computations can be carried out by hand. We have $g'(X) = \frac{X^{-\frac{2}{3}}}{3}$. For $X = Interval[\{a, b\}]$ we let $|X| = Maximum\{|a|, |b|\}$ and $width(X) = b - a$. We now have

$$\begin{aligned} X^{-\frac{2}{3}} &= Interval[\{a^{-2/3}, b^{-2/3}\}], && 0 < a \le b, \\ &= Interval[\{|X|^{-2/3}, -\infty\}], && a < 0 < b, \\ &= Interval[\{|a|^{-2/3}, |b|^{-2/3}\}], && a \le b < 0. \end{aligned}$$

Again, in case of division with intervals containing 0 we can use extended interval arithmetic. In case $a < 0 < b$ we find that

$$N(X) = middle(X) - \frac{middle(X)^{1/3}}{\frac{X^{-2/3}}{3}}$$

and this gives

$$X_1 = X_0 \cap (middle(X) - 3middle(X)^{1/3} Interval[\{0, |X|^{2/3}\}]).$$

We can distinguish three cases:

1. if $middle(X) > 0$, then $X_1 \subseteq Interval[\{a, middle(X)\}]$,
2. if $middle(X) < 0$, then $X_1 \subseteq Interval[\{middle(X), b\}]$,
3. if $middle(X) = 0$, then $X_1 = 0$.

In the first two cases, the width of X_1 is no more than half that of X. In any case, if X contains 0, then the sequence X_n will contain and converge to the solution $x = 0$. The width of X_n will be no more than $2^{-n} width(X)$. This proves that g has a root in any interval containing the origin and we are sure that interval Newton's iteration will converge to it by the 4th property of the interval Newton's algorithm.This is in strong contrast with the behavior of the ordinary Newton's method as the following computation shows:

$$\begin{aligned} x_{n+1} &= x_n - \frac{g(x_n)}{g'(x_n)} \\ &= x_n - 3x_n \\ &= -2x_n. \end{aligned}$$

Thus the ordinary Newton's method for g diverges from any x_0, except the solution itself.

A *Mathematica* Version: 2.2 implementation of the interval Newton's method can electronically be obtained from the authors.

References

1. Donovan, G.C., Miller, A.R. and Moreland, T.J. *Pathological functions for Newton's method.* The American Mathematical Monthly **100** (1) (1993), pp. 53–58.
2. Burden, R.L. and Faires, J.D. *Numerical analysis.* PWS-Kent, Boston, 1989.
3. Lanford, O.E. III. *Computer-assisted proofs in analysis.* In: "Proceedings of the International Congress of Mathematicians", Berkeley, 1986.
4. Moore, R.E. *Interval analysis.* Prentice-Hall, Englewood Cliffs, 1966.
5. Hansen, E. *Global optimization using interval analysis.* Marcel Dekker, New York, 1992.

6. Ratschek, H. and Rokne, J.G. *Computer methods for the range of functions.* Horwood, Chichester, 1984.

7. Alefeld, G. and Herzberger, J. *Introduction to interval computations.* Academic Press, New York, 1983.

8. Kulisch, U.W. and Miranker, W.L. *The arithmetic of the digital computer.* SIAM Rev. **28** (1) (1986), pp. 1–40.

9. Kearfott, R.B. *Interval arithmetic techniques in the computational solution of nonlinear systems of equations: introduction, examples, and comparisons.* In: "Computational Solution of Nonlinear Systems of Equations (Lectures in Applied Mathematics, Volume 26)", American Mathematical Society, Providence, 1990, pp. 337–358.

10. Akyildiz, Y. and Bartholomew-Biggs, M.C. *Roots of a function over an interval.* Mathematica Journal **3** (1) (1993), p. 21.

11. Moore, R.E. *Methods and application of interval analysis.* SIAM, Philadelphia, 1979.

Received: February 28, 1993

Revised version: May 16, 1993

Y. Akyildiz
Computer Science Department
King Fahd University of Petroleum
and Minerals, Box 1971
Dhahran 31261
Saudi Arabia
M.I. Al-Suwaiyel
KACST
PO Box: 6086
Riyadh 11442
Saudi Arabia