TWO ROOTS GOOD, ONE ROOT BETTER

R. Baker Kearfott

Department of Mathematics

University of Southwestern Louisiana

Lafayette, LA

email rbk@ucs.usl.edu

and Vladik Kreinovich

Department of Computer Science

University of Texas at El Paso

El Paso, TX

email vladik@cs.ep.utexas.edu

Main objective of this talk:

- to explain the following experimental facts:
  - *it is harder to find a root if it is not unique*
  - *it is harder to find a point where a function f attains its maximum*
- Two possible explanations:
  - this is a drawback of the existing methods, so for other methods, finding non-unique roots is as easy as finding unique ones
  - it actually, *is* harder to find a non-unique root

Example when a similar problem turned out to be a method's drawback:

- *Problem:*

  Newton's method works:

  - *faster* and *better* if the root is in the middle of the domain, and
  - *worse* if the root is close to the border

- *Solution:* Other methods found near-the-border roots as easily as the roots in the middle of the domain

MAIN RESULT

CASE 1: UNIQUE SOLUTION

THEOREM 1. *There exists an algorithm that is applicable to an arbitrary computably constructive function f from a computable compact K to R that has a unique root, and returns that root.*

THEOREM 2. *There exists an algorithm that is applicable to an arbitrary computably constructive function f from a computable compact K to R that attains its maximum at exactly one point, and returns that point.*

Comment. We still need to define what we mean by an algorithm, what is a computable functions, etc. We will do that in a minute.

MAIN RESULT
CASE 2: NON-UNIQUE SOLUTION

THEOREM 3. *No algorithm is possible that is applicable to any polynomial function $f(x)$ with exactly two roots, and returns these two roots.*

THEOREM 4. *No algorithm is possible that is applicable to any polynomial function that attains maximum at exactly two points, and returns these two points.*

SO, WE HAVE SOLVED THE ORIGINAL PROBLEM BUT A NEW PROBLEM ARISES:

- We have thus solved the original problem:

  it *is* easier to find a unique root.

  How easy is it?

- The algorithm that we describe is exp –time, so it is not feasible in the following sense:

  For input of length n, it takes $\geq 2^n$ computational steps.

  This is an *exhaustive search.*

  For $n \approx 300$, it takes longer than the lifetime of the Universe.

- **New problem:** Can we have a *feasible* algorithm that finds all unique roots? (at least for polynomial f)?

- **Our answer:** *No.*

  Strictly speaking, no unless P=NP, i.e., unless *all* problems can be solved in feasible time by a single algorithm.

First: what do we mean by an algorithm?

- *An algorithm is a well-defined sequence of precise steps.*

  Comment. In other words, something that can be immediately programmed.

- An example of a *not yet* algorithm:

  Newton's method:

  1) Take any x_1 (*how?*)

  2) $x_{n+1} = x_n - f(x_n)/f'(x_n)$

  3) stop, e.g., when x_{n+1} is close to x_n (*when exactly should we stop? what does this method guarantee?*)

Computable real numbers

- **Problem:**
  - We are interested in algorithms that work with real numbers
  - In the computer, we can only store an *approximation* to a real number.

Definition.

- We say that an algorithm $\mathcal{U}$ *computes* a real number x if for every natural number k, it generates a rational number r_k such that $|r_k - x| \le 2^{-k}$.
- We say that we have a *computable* real number if we have an algorithm $\mathcal{U}$ that computes it.

Computable functions

- **Idea:** $f : R \to R$ is computable if we can compute $f(x)$ for any desired accuracy.

  In reality, we can only have x_n such that $|x_n - x| \le 2^{-n}$. So, we must know what n to choose.

- **Definitions.**

  - We say that an algorithm $\mathcal{V}$ *computes* a function $f : R \to R$ if $\mathcal{V}$ includes calls to an (unspecified) algorithm $\mathcal{U}$ so that when we take as $\mathcal{U}$ an algorithm that computes a real number x, $\mathcal{V}$ will compute a real number $f(x)$.

  - We say that we have a *computable* real function $f(x)$ if we have an algorithm that computes this function.

- *Comments.*

  1. This algorithm $\mathcal{V}$ takes k as input, and generates a rational number s_k such that $|s_k - f(x)| \le 2^{-k}$. In course of computations, it may generate an auxiliary number l, and ask $\mathcal{U}$ for a value r_l that is 2^{-l}–close to x.

  2. In a similar manner, one can define a constructive function of n real variables: it just calls n programs $\mathcal{U}_i$, $1 \le i \le n$.

Constructive metric space and constructive compact

- *Constructive metric space:*

  $X,\, d : X \times X \to R$

- *Constructive compact:*

  $K,\, d,\, U : n \to 2^{-n}$–net for X.

Constructively continuous function

Definition. We say that a computable function $f : A \to B$ is *constructively continuous* on a metric space A if there exists an algorithm, that for every $\varepsilon > 0$, generates $\delta > 0$ such that if $d_A(x, y) \leq \delta$, then $d_B(f(x), f(y)) \leq \varepsilon$.

Known properties of constructive functions

- **Supremum is computable**
  - $f : K \to R$
  - K is a constructive compact
  - f is constructively continuous
  - To compute $\sup f$ with accuracy ε, we do the following:

1) Find δ that corresponds to this ε;

2) Find a δ–net $x_1, ..., x_n$.

3) Compute $f(x_1), ..., f(x_n)$.

4) Find $\max(f(x_1), ..., f(x_n))$.

- **Level sets:** $\forall f\, \forall r > s > 0\, \exists t$ such that $s < t < r$ and $\{x | f(x) \le t\}$ is a constructive compact.

An algorithm that computes a unique root

- $\exists! x_0 (f(x) = 0)$
- We want to compute x_0 with accuracy 2^{-k}.
- $\exists r : 2^{-(k+1)} < r < 2^{-k}$ for which $K = \{(x, y) | (x, y) \geq r\}$ is a constructive compact.
- Compute

$$\delta = \inf_{(x,y) \in K} [\max\{|f(x)|, |f(y)|\}].$$

- If $\max\{|f(x)|, |f(y)|\} < \delta$, then $d(x, y) < r$.
- So, if $|f(y)| < \delta$, then $d(x, x_0) < r < 2^{-k}$.
- So, we try all the points from all $\varepsilon-$nets until we get y for which $|f(y)| < \delta$.

Comment. This is an exhaustive search.

There exists an algorithm that computes a unique maximum location

- **Main idea:**

$$f(x) \to \max$$

is equivalent to

$$g(x) = 0,$$

where

$$g(x) = f(x) - \max_y f(y).$$

Known negative result that we will use

- **Theorem.** $\neg\exists$ *an algorithm that decides whether* $x = 0$ *or* $x \neq 0$.
- **Idea of the proof:** else, we would be able to solve *all* mathematical problems.
  - Example: $x^n + y^n = z^n$
  - $x_k = 2^{-k}$ if $\exists x, y, z, n > 2(x^n + y^n = z^n)$ and $x_k = 0$ else.
  - $x = 0$ iff Fermat's Last Theorem is true.
- Using Gödel's theorem completes the proof.

Proof that no algorithm is possible that finds a root if there are exactly two of them

- Take $f_\alpha(x) = (x-1-\alpha^2)(x-1+\alpha^2)((x+1)^2+\alpha^2)$, where α is a constructive real number.
- For every α, f_α has exactly two roots:
  - If $\alpha = 0$, then $x = \pm 1$.
  - If $\alpha \neq 0$, then $x = 1 \pm \alpha^2$.
- *Comments.*
  1. For maximum, we take
     $f_\alpha(x) = -(x-1-\alpha^2)^2(x-1+\alpha^2)^2((x+1)^2+\alpha^2)$.
  2. If the roots are separated, i.e., if we know the lower bound for the distance between the roots, then we can find them algorithmically.

Even for unique roots, finding the solution is as hard as NP-hard

- *3–CNF:*

  $F = (x_1 \vee x_2 \vee \neg x_3)\&(...)$

- $\tilde{F} = x_1 x_2 (1 - x_3) + (...) + ... = 0,\ x_i \in [0, 1].$

- **Lemma.** *F has a unique solution iff $\tilde{F}$ has a unique root.*