35. Interval arithmetic II

Last time

- Calculating with **sets** instead of numbers
- Fundamental idea of interval arithmetic:

  enclosure of **range** of function
- Non-existence of roots

Goals for today

- Directed rounding
- Dependency problem
- Finding all roots: Branch-and-bound algorithms
- Interval Newton method
- Global optimization

Correct rounding

- Think about operation like $\exp(x)$ on float x

Correct rounding

- Think about operation like exp(x) on float x
- Recall: a float x is a special (dyadic) *rational* number

Correct rounding

- Think about operation like $\exp(x)$ on float x
- Recall: a float x is a special (dyadic) *rational* number
- $\exp(x)$ will produce a non-float real

Correct rounding

- Think about operation like $\exp(x)$ on float x
- Recall: a float x is a special (dyadic) *rational* number
- $\exp(x)$ will produce a non-float real
- **Correct rounding**: Return *closest* float to true result

Correct rounding

- Think about operation like exp(x) on float x
- Recall: a float x is a special (dyadic) *rational* number
- exp(x) will produce a non-float real
- **Correct rounding**: Return *closest* float to true result
- *Difficult* to do in general: CRlibm library

Correct rounding

- Think about operation like exp(x) on float x
- Recall: a float x is a special (dyadic) *rational* number
- exp(x) will produce a non-float real
- **Correct rounding**: Return *closest* float to true result
- *Difficult* to do in general: CRlibm library
- IEEE-754 standard for floating-point arithmetic
- Requires correct rounding for `+`, `-`, `*`, `/`, `sqrt`

Correct rounding

- Think about operation like $\exp(x)$ on float x
- Recall: a float x is a special (dyadic) *rational* number
- $\exp(x)$ will produce a non-float real
- **Correct rounding**: Return *closest* float to true result
- *Difficult* to do in general: CRlibm library
- IEEE-754 standard for floating-point arithmetic
- Requires correct rounding for `+`, `-`, `*`, `/`, `sqrt`
- Use additional internal bits of precision to calculate

Directed rounding

- We do not know in *which* direction rounding occurred

Directed rounding

- We do not know in *which* direction rounding occurred
- For interval arithmetic, need to **bound** this **error**

Directed rounding

- We do not know in *which* direction rounding occurred
- For interval arithmetic, need to **bound** this **error**
- Desired result: interval **guaranteed** to contain **true** value

Directed rounding

- We do not know in *which* direction rounding occurred
- For interval arithmetic, need to **bound** this **error**
- Desired result: interval **guaranteed** to contain **true** value
- “**Enclosure**”

Directed rounding II

- Various possible techniques to do this:
    1. Control processor's rounding direction

Directed rounding II

- Various possible techniques to do this:
  1. Control processor's rounding direction
     - Possible but technically difficult and slow, not thread-safe

Directed rounding II

- Various possible techniques to do this:
  1. Control processor's rounding direction
     - Possible but technically difficult and slow, not thread-safe
  2. Artificially move the result **outwards**:
     - move left endpoint down (towards $-\infty$)
     - move right endpoint up (towards $+\infty$)

Directed rounding II

- Various possible techniques to do this:
  1. Control processor's rounding direction
     - Possible but technically difficult and slow, not thread-safe
  2. Artificially move the result **outwards**:
     - move left endpoint down (towards $-\infty$)
     - move right endpoint up (towards $+\infty$)
     - In Julia: `prevfloat` and `nextfloat`

Directed rounding II

- Various possible techniques to do this:
  1. Control processor's rounding direction
     - Possible but technically difficult and slow, not thread-safe
  2. Artificially move the result **outwards**:
     - move left endpoint down (towards $-\infty$)
     - move right endpoint up (towards $+\infty$)
     - In Julia: `prevfloat` and `nextfloat`
     - Result is 2ulps wide instead of 1ulp (unit in last place)

Simple implementation

- Can implement this easily:

```
struct SimpleInterval
    inf::Float64
    sup::Float64
end

import Base: +
+(x::SimpleInterval, y::SimpleInterval) =
    SimpleInterval( prevfloat(x.inf + y.inf),
                    nextfloat(x.sup + y.sup) )

x = SimpleInterval(0.1, 0.3)
y = SimpleInterval(0.2, 0.4)

x + y
```

IntervalArithmetic.jl

- Julia package tuned for efficiency
- Almost compliant with IEEE-1788 standard

IntervalArithmetic.jl

- Julia package tuned for efficiency
- Almost compliant with IEEE-1788 standard

```julia
using IntervalArithmetic

x = 0.1..0.3   # shorthand for `interval(0.1, 0.3)`
y = 0.2..0.4

x + y
```

IntervalArithmetic.jl

- Julia package tuned for efficiency
- Almost compliant with IEEE-1788 standard

```
using IntervalArithmetic

x = 0.1..0.3   # shorthand for `interval(0.1, 0.3)`
y = 0.2..0.4

x + y
```

- Compare

```
x = SimpleInterval(0.1, 0.3)
x + x

y = interval(0.1, 0.3)
```

IntervalArithmetic.jl

- Recall the example of excluding roots
- Let's see how to do this with `IntervalArithmetic.jl`

```
X = 3..4
f(x) = x^2 - 2

0 in f(X)   # returns false
```

- $f(X)$ is obtained by substituting X instead of x everywhere in definition of f
- Called **natural interval extension**

Dependency problem

- What is $X - X$ for the interval $X := [0..1]$?

Dependency problem

- What is $X - X$ for the interval $X := [0..1]$?
- It should be $\{x - x : x \in X\} = 0$

Dependency problem

- What is $X - X$ for the interval $X := [0..1]$?
- It should be $\{x - x : x \in X\} = 0$
- But we actually get $\{x - y : x, y \in X\} = [-1..1]$

Dependency problem

- What is $X - X$ for the interval $X := [0..1]$?
- It should be $\{x - x : x \in X\} = 0$
- But we actually get $\{x - y : x, y \in X\} = [-1..1]$
- We “cannot tell” that it is the same X both times

Dependency problem

- What is $X - X$ for the interval $X := [0..1]$?
- It should be $\{x - x : x \in X\} = 0$
- But we actually get $\{x - y : x, y \in X\} = [-1..1]$
- We “cannot tell” that it is the same X both times
- **Dependency problem** of interval arithmetic

Dependency problem

- What is $X - X$ for the interval $X := [0..1]$?
- It should be $\{x - x : x \in X\} = 0$
- But we actually get $\{x - y : x, y \in X\} = [-1..1]$
- We “cannot tell” that it is the same X both times
- **Dependency problem** of interval arithmetic
- Serious impediment to using interval arithmetic more widely
- Partial solution: Affine arithmetic

Finding a single root?

- We can exclude a root by the **inclusion test** $0 \in f(X)$

Finding a single root?

- We can exclude a root by the **inclusion test** $0 \in f(X)$
- If $0 \notin f(X)$ then there is no root – theorem

Finding a single root?

- We can exclude a root by the **inclusion test** $0 \in f(X)$
- If $0 \notin f(X)$ then there is no root – theorem
- However, if $0 \in f(X)$ we *cannot conclude anything*

Finding a single root?

- We can exclude a root by the **inclusion test** $0 \in f(X)$
- If $0 \notin f(X)$ then there is no root – theorem
- However, if $0 \in f(X)$ we *cannot conclude anything*
- Overestimation from dependency problem may lead to

  $0 \notin \text{range}(f; X)$

  but $0 \in f(X)$

Finding *all* roots

- Given an interval X, how can we find *all* roots in X

Finding *all* roots

- Given an interval X, how can we find *all* roots in X
- So far: know how to *exclude* roots from single interval

Finding *all* roots

- Given an interval X, how can we find *all* roots in X
- So far: know how to *exclude* roots from single interval
- **Idea**: Split interval into **pieces**

Finding *all* roots

- Given an interval X, how can we find *all* roots in X
- So far: know how to *exclude* roots from single interval
- **Idea**: Split interval into **pieces**
- Simplest: Equal-sized pieces – **mincing**

Finding *all* roots

- Given an interval X, how can we find *all* roots in X
- So far: know how to *exclude* roots from single interval
- **Idea**: Split interval into **pieces**
- Simplest: Equal-sized pieces – **mincing**
- Theorem: Over-estimation of range decreases as $\mathcal{O}(w)$
- w is width of each piece

Branch and prune

- How can we improve on this?

Branch and prune

- How can we improve on this?
- **Idea**: (Spatial) branch and prune

Branch and prune

- How can we improve on this?
- **Idea**: (Spatial) branch and prune
- Branch: Bisect
- Prune: Check each piece and throw away if no root

Branch and prune

- How can we improve on this?
- **Idea**: (Spatial) branch and prune
- Branch: Bisect
- Prune: Check each piece and throw away if no root
- Effectively builds a **binary tree** in an efficient way

Branch and prune

- How can we improve on this?
- **Idea**: (Spatial) branch and prune
- Branch: Bisect
- Prune: Check each piece and throw away if no root
- Effectively builds a **binary tree** in an efficient way
- Exhaustive search of the space up to some tolerance

Proving existence and uniqueness of roots

- Start from initial box X_0
- What does branch and prune produce?

Proving existence and uniqueness of roots

- Start from initial box X_0
- What does branch and prune produce?
- List of intervals X^i whose **union** contains all roots in X_0

Proving existence and uniqueness of roots

- Start from initial box X_0
- What does branch and prune produce?
- List of intervals X^i whose **union** contains all roots in X_0
- i.e. if x is a root of f then x is in some X^i

Proving existence and uniqueness of roots

- Start from initial box X_0
- What does branch and prune produce?
- List of intervals X^i whose **union** contains all roots in X_0
- i.e. if x is a root of f then x is in some X^i
- But still don't know if there *are* roots or how many

First solution

- Suppose we have reached a small interval X^i where we they may be a root
- So $0 \in f(X)$

First solution

- Suppose we have reached a small interval X^i where we they may be a root
- So $0 \in f(X)$
- Calculus: sufficient condition for unique root to exist:

  $f'(x) > 0$ for all $x \in X$

First solution

- Suppose we have reached a small interval X^i where we they may be a root
- So $0 \in f(X)$
- Calculus: sufficient condition for unique root to exist:

  $f'(x) > 0$ for all $x \in X$
- How can we check this?

First solution

- Suppose we have reached a small interval X^i where we they may be a root
- So $0 \in f(X)$
- Calculus: sufficient condition for unique root to exist:

  $f'(x) > 0$ for all $x \in X$
- How can we check this?
- **Idea**: Use algorithmic differentiation and interval arithmetic!

Second solution: Interval Newton operator

- Extend standard Newton method to interval context

Second solution: Interval Newton operator

- Extend standard Newton method to interval context
- Define $\tilde{m}(X)$:= midpoint of interval X

Second solution: Interval Newton operator

- Extend standard Newton method to interval context
- Define $\tilde{m}(X)$:= midpoint of interval X
- Need thin interval version: $m(X) := [\tilde{m}(x)..\tilde{m}(x)]$

Second solution: Interval Newton operator

- Extend standard Newton method to interval context
- Define $\tilde{m}(X)$:= midpoint of interval X
- Need thin interval version: $m(X) := [\tilde{m}(x)..\tilde{m}(x)]$
- Newton operator is
- $\mathcal{N}_f(X) := m(X) - \frac{f(m(X))}{f'(X)}$

Interval Newton II

- **Idea**: Through $(m, f(m))$ take *all* straight lines whose slope is some number in the interval $f'(X)$

Interval Newton II

- **Idea**: Through $(m, f(m))$ take *all* straight lines whose slope is some number in the interval $f'(X)$
- Here $f'(X)$ is the natural extension *of the derivative function f'*

Interval Newton II

- **Idea**: Through $(m, f(m))$ take *all* straight lines whose slope is some number in the interval $f'(X)$
- Here $f'(X)$ is the natural extension *of the derivative function* f'
- Can calculate using algorithmic differentiation!

Interval Newton II

- **Idea**: Through $(m, f(m))$ take *all* straight lines whose slope is some number in the interval $f'(X)$
- Here $f'(X)$ is the natural extension *of the derivative function* f'
- Can calculate using algorithmic differentiation!

Interval Newton III

- Theorem:
  - Any root in X lies in $\mathcal{N}_f(X)$
  - So if $\mathcal{N}_f(X) \cap X = \emptyset$ then there is no root
  - If $\mathcal{N}_f(X) \subseteq X$ then there is a unique root in X

Higher dimensions

- Simplest sets in higher dimensions:
- Cartesian products of intervals, $X_1 \times X_2$

Higher dimensions

- Simplest sets in higher dimensions:
- Cartesian products of intervals, $X_1 \times X_2$
- Branch and prune and interval Newton extend directly

Global optimization: Ground state of atomic cluster

- N atoms at positions $\mathbf{x}_i \in \mathbb{R}^{3\star}$
- **Interaction potential** $V(r_{ij})$ between pairs
- Ground state: Minimize **potential energy**

$$V(\mathbf{x}_1, \ldots, \mathbf{x}_N) := \sum_{i=1}^{N} \sum_{j>i} V(r_{ij})$$

- $r_{ij} := \|\mathbf{x}_i - \mathbf{x}_j\|$ is distance between atoms i and j

Lennard-Jones potential

- Standard model of interaction between argon atoms:

$$V(r) := 4\left(\frac{1}{r^{12}} - \frac{1}{r^6}\right)$$

- Problem: There are *lots* of local minima
- Estimated to grow like $\mathcal{O}(e^N)$
- To find the ground state we need **global optimization**

Global optimization

- Similar algorithm to branch and prune is

  *branch and **bound***

Global optimization

- Similar algorithm to branch and prune is

  *branch and **bound***

- Suppose have upper bound m for global minimum value
- If $\underline{f(X)} > m$ then X cannot contain global minimum

Global optimization

- Similar algorithm to branch and prune is

  *branch and **bound***

- Suppose have upper bound m for global minimum value
- If $\underline{f(X)} > m$ then X cannot contain global minimum
- Use this in similar branching algorithm

Constraint propagation

- Interval arithmetic may be applied to **contract** a box X
- By eliminating parts of X that don't satisfy a **constraint**

Constraint propagation

- Interval arithmetic may be applied to **contract** a box X
- By eliminating parts of X that don't satisfy a **constraint**
- e.g. Constraint $x^2 + y^2 \leq 1$
- Initial box $X = (-\infty..\infty) \times (-\infty..\infty)$

Constraint propagation

- Interval arithmetic may be applied to **contract** a box X
- By eliminating parts of X that don't satisfy a **constraint**
- e.g. Constraint $x^2 + y^2 \leq 1$
- Initial box $X = (-\infty..\infty) \times (-\infty..\infty)$
- **Idea**: Forwards–backwards algorithm:
  1. Propagate forwards
  2. Apply constraint
  3. Propagate backwards

Review

- Applications of intervals
- Branch and prune for excluding roots
- Interval Newton for proving existence and uniqueness
- Global optimization