Lecture 4
Numerical Computation

Chee Yap
Courant Institute of Mathematical Sciences
New York University

Overview

We introduce some basic concepts of numerical computation. Ultimately, our goal is to do algebraic computation using numerical methods. The reason is that purely algebraic methods is not as efficient or adaptive as numerical approaches. Here, the work of Brent is our starting point.

- I. Floating Point Arithmetic
- II. Brent's Work
- III. Newton Iteration with Error
- IV. Approximate Zeros (Smale's work)

Introduction

- We assume bigfloats in all numerical computation
  - Unlike numerical analysis
  - It is our work-horse – cf. Expr in CORE
- Why floats?
  - Large number range, compared to fixed precision
  - Fast, compared to rational numbers
  - Basically, its speed is integer computation + small overhead
- The price for the above advantages?
  - Uneven gaps between representable numbers
  - Harder error analysis (cf. von Neumann)
- Two modes of using bigfloats

Introduction

- We assume bigfloats in all numerical computation
  - Unlike numerical analysis
  - It is our work-horse – cf. Expr in CORE
- Why floats?
  - Large number range, compared to fixed precision
  - Fast, compared to rational numbers
  - Basically, its speed is integer computation + small overhead
- The price for the above advantages?
  - Uneven gaps between representable numbers
  - Harder error analysis (cf. von Neumann)
- Two modes of using bigfloats

Introduction

- We assume bigfloats in all numerical computation
  - Unlike numerical analysis
  - It is our work-horse – cf. Expr in CORE
- Why floats?
  - Large number range, compared to fixed precision
  - Fast, compared to rational numbers
  - Basically, its speed is integer computation + small overhead
- The price for the above advantages?
  - Uneven gaps between representable numbers
  - Harder error analysis (cf. von Neumann)
- Two modes of using bigfloats

Introduction

- We assume bigfloats in all numerical computation
  - ∗ Unlike numerical analysis
  - ∗ It is our work-horse – cf. Expr in CORE
- Why floats?
  - ∗ Large number range, compared to fixed precision
  - ∗ Fast, compared to rational numbers
  - ∗ Basically, its speed is integer computation + small overhead
- The price for the above advantages?
  - ∗ Uneven gaps between representable numbers
  - ∗ Harder error analysis (cf. von Neumann)
- Two modes of using bigfloats

Introduction

- We assume bigfloats in all numerical computation
  - ∗ Unlike numerical analysis
  - ∗ It is our work-horse – cf. Expr in CORE
- Why floats?
  - ∗ Large number range, compared to fixed precision
  - ∗ Fast, compared to rational numbers
  - ∗ Basically, its speed is integer computation + small overhead
- The price for the above advantages?
  - ∗ Uneven gaps between representable numbers
  - ∗ Harder error analysis (cf. von Neumann)
- Two modes of using bigfloats

Introduction

- We assume bigfloats in all numerical computation
  - ∗ Unlike numerical analysis
  - ∗ It is our work-horse – cf. Expr in CORE
- Why floats?
  - ∗ Large number range, compared to fixed precision
  - ∗ Fast, compared to rational numbers
  - ∗ Basically, its speed is integer computation + small overhead
- The price for the above advantages?
  - ∗ Uneven gaps between representable numbers
  - ∗ Harder error analysis (cf. von Neumann)
- Two modes of using bigfloats

Introduction

- We assume bigfloats in all numerical computation
  - ∗ Unlike numerical analysis
  - ∗ It is our work-horse – cf. Expr in CORE
- Why floats?
  - ∗ Large number range, compared to fixed precision
  - ∗ Fast, compared to rational numbers
  - ∗ Basically, its speed is integer computation + small overhead
- The price for the above advantages?
  - ∗ Uneven gaps between representable numbers
  - ∗ Harder error analysis (cf. von Neumann)
- Two modes of using bigfloats

* Weak mode: generalized IEEE standard
* Strong mode: we actively control precision

- Work of Brent in the 1970's
  * Basic conclusion: all the common elementary functions has local complexity $O(M(n)\log n)$
  * Shanks: elementary function is complex function that is a finite composition of constants, field operations, algebraic functions, exponential and logarithmic functions
  * Brent shows that multiplication is reducible to $\exp(x)$ and $\sin(x)$, and so $M(s)$ is lower bound

∗ Weak mode: generalized IEEE standard

∗ Strong mode: we actively control precision

- Work of Brent in the 1970's

∗ Basic conclusion: all the common elementary functions has local complexity $O(M(n)\log n)$

∗ Shanks: elementary function is complex function that is a finite composition of constants, field operations, algebraic functions, exponential and logarithmic functions

∗ Brent shows that multiplication is reducible to $\exp(x)$ and $\sin(x)$, and so $M(s)$ is lower bound

∗ Weak mode: generalized IEEE standard

∗ Strong mode: we actively control precision

- Work of Brent in the 1970's

∗ Basic conclusion: all the common elementary functions has local complexity $O(M(n)\log n)$

∗ Shanks: **elementary function** is complex function that is a finite composition of constants, field operations, algebraic functions, exponential and logarithmic functions

∗ Brent shows that multiplication is reducible to $\exp(x)$ and $\sin(x)$, and so $M(s)$ is lower bound

Floating Point Representation

- Crisis in the 1980's: proliferation of hardware fp
  - FP computation used to require a co-processor!
  - Problem of irreproducible results
  - Kahan's Turing award: contributions to IEEE Standard

- If $f \in \mathbb{Z}$, write $\langle f \rangle$ for $f2^{-\lfloor \lg |f| \rfloor}$
  - Call $\langle f \rangle$ the normalized value of f
  - E.g., $\langle 1 \rangle = \langle 2 \rangle = \langle 4 \rangle = 1$, $\langle 3 \rangle = \langle 6 \rangle = 1.5$, $\langle 5 \rangle = 1.25$, $\langle 7 \rangle = 1.75$, etc

- Alternatively, $\langle f \rangle = \pm(b_0.b_1b_2 \cdots b_t)_2$
  - where $(b_0b_1 \cdots b_t)_2$ is the binary notation for f
  - THUS: $|\langle f \rangle| \in [1, 2)$ for $f \neq 0$
  - Also, $\langle 2^k f \rangle = \langle f \rangle$ for all $k \in \mathbb{N}$

Floating Point Representation

- Crisis in the 1980's: proliferation of hardware fp
  - $*$ FP computation used to require a co-processor!
  - $*$ Problem of irreproducible results
  - $*$ Kahan's Turing award: contributions to IEEE Standard

- If $f \in \mathbb{Z}$, write $\langle f \rangle$ for $f2^{-\lfloor \lg |f| \rfloor}$
  - $*$ Call $\langle f \rangle$ the normalized value of f
  - $*$ E.g., $\langle 1 \rangle = \langle 2 \rangle = \langle 4 \rangle = 1$, $\langle 3 \rangle = \langle 6 \rangle = 1.5$, $\langle 5 \rangle = 1.25$, $\langle 7 \rangle = 1.75$, etc

- Alternatively, $\langle f \rangle = \pm(b_0.b_1b_2 \cdots b_t)_2$
  - $*$ where $(b_0b_1 \cdots b_t)_2$ is the binary notation for f
  - $*$ THUS: $|\langle f \rangle| \in [1,2)$ for $f \neq 0$
  - $*$ Also, $\langle 2^k f \rangle = \langle f \rangle$ for all $k \in \mathbb{N}$

Floating Point Representation

- Crisis in the 1980's: proliferation of hardware fp
  - FP computation used to require a co-processor!
  - Problem of irreproducible results
  - Kahan's Turing award: contributions to IEEE Standard

- If $f \in \mathbb{Z}$, write $\langle f \rangle$ for $f 2^{-\lfloor \lg |f| \rfloor}$
  - Call $\langle f \rangle$ the **normalized value** of f
  - E.g., $\langle 1 \rangle = \langle 2 \rangle = \langle 4 \rangle = 1$, $\langle 3 \rangle = \langle 6 \rangle = 1.5$, $\langle 5 \rangle = 1.25$, $\langle 7 \rangle = 1.75$, etc

- Alternatively, $\langle f \rangle = \pm(b_0.b_1 b_2 \cdots b_t)_2$
  - where $(b_0 b_1 \cdots b_t)_2$ is the binary notation for f
  - THUS: $|\langle f \rangle| \in [1, 2)$ for $f \neq 0$
  - Also, $\langle 2^k f \rangle = \langle f \rangle$ for all $k \in \mathbb{N}$

Floating Point Representation

- Crisis in the 1980's: proliferation of hardware fp
  - * FP computation used to require a co-processor!
  - * Problem of irreproducible results
  - * Kahan's Turing award: contributions to IEEE Standard

- If $f \in \mathbb{Z}$, write $\langle f \rangle$ for $f 2^{-\lfloor \lg |f| \rfloor}$
  - * Call $\langle f \rangle$ the **normalized value** of f
  - * E.g., $\langle 1 \rangle = \langle 2 \rangle = \langle 4 \rangle = 1$, $\langle 3 \rangle = \langle 6 \rangle = 1.5$, $\langle 5 \rangle = 1.25$, $\langle 7 \rangle = 1.75$, etc

- Alternatively, $\langle f \rangle = \pm (b_0.b_1 b_2 \cdots b_t)_2$
  - * where $(b_0 b_1 \cdots b_t)_2$ is the binary notation for f
  - * THUS: $|\langle f \rangle| \in [1, 2)$ for $f \neq 0$
  - * Also, $\langle 2^k f \rangle = \langle f \rangle$ for all $k \in \mathbb{N}$

- A (binary) bigfloat has form $n2^m$ where $n, m \in \mathbb{Z}$
  - Can also written as $\langle n \rangle 2^m$ for some m, n

- WRITE: $\langle e, f \rangle$ for $\langle f \rangle 2^e$
  - Call e the exponent and $\langle f \rangle$ the fraction
  - The representation $\langle e, f \rangle$ is normalized if $e = f = 0$ or if f is odd

- Local BigFloat Computation: all numbers comes from a bounded range $[a, b]$
  - Alternatively, all $\langle e, f \rangle$ has bounded e

- A (binary) bigfloat has form $n2^m$ where $n, m \in \mathbb{Z}$
  * Can also written as $\langle n \rangle 2^m$ for some m, n

- WRITE: $\langle e, f \rangle$ for $\langle f \rangle 2^e$
  * Call e the exponent and $\langle f \rangle$ the fraction
  * The representation $\langle e, f \rangle$ is normalized if $e = f = 0$ or if f is odd

- Local BigFloat Computation: all numbers comes from a bounded range $[a, b]$
  * Alternatively, all $\langle e, f \rangle$ has bounded e

- A (binary) bigfloat has form $n2^m$ where $n, m \in \mathbb{Z}$
  - * Can also written as $\langle n \rangle 2^m$ for some m, n

- WRITE: $\langle e, f \rangle$ for $\langle f \rangle 2^e$
  - * Call e the **exponent** and $\langle f \rangle$ the **fraction**
  - * The representation $\langle e, f \rangle$ is normalized if $e = f = 0$ or if f is odd

- Local BigFloat Computation: all numbers comes from a bounded range $[a, b]$
  - * Alternatively, all $\langle e, f \rangle$ has bounded e

- A (binary) bigfloat has form $n2^m$ where $n, m \in \mathbb{Z}$
  - $*$ Can also written as $\langle n \rangle 2^m$ for some m, n

- WRITE: $\langle e, f \rangle$ for $\langle f \rangle 2^e$
  - $*$ Call e the **exponent** and $\langle f \rangle$ the **fraction**
  - $*$ The representation $\langle e, f \rangle$ is normalized if $e = f = 0$ or if f is odd

- Local BigFloat Computation: all numbers comes from a bounded range $[a, b]$
  - $*$ Alternatively, all $\langle e, f \rangle$ has bounded e

IEEE Floating Point Standard

- This is the resolution of the crisis in the 1980's
  - Does it solve the non-robustness problem?
  - Slightly, because of rational design
  - It ensures portable code and predictable results: if it fails on one machine, it should fail on others!
  - Official Name: IEEE Standard 754 for Binary Floating-Point Arithmetic (1987)
- Floating Point System $FP(2,t)$:
  - All numbers of the form $\langle e, f\rangle$ where $|f| < 2^t$
  - DENOTE by $FP(2,t,e_{\min},e_{\max})$ if, in addition, $e_{\min} \leq e \leq e_{\max}$
- The IEEE Standard for double precision is $FP(2,53,-1023,1023)$
  - Bit allocation: $64 = 1 + 11 + 53$

IEEE Floating Point Standard

- This is the resolution of the crisis in the 1980's
  - Does it solve the non-robustness problem?
  - Slightly, because of rational design
  - It ensures portable code and predictable results: if it fails on one machine, it should fail on others!
  - Official Name: IEEE Standard 754 for Binary Floating-Point Arithmetic (1987)

- Floating Point System $FP(2, t)$:
  - All numbers of the form $\langle e, f \rangle$ where $|f| < 2^t$
  - DENOTE by $FP(2, t, e_{\min}, e_{\max})$ if, in addition, $e_{\min} \le e \le e_{\max}$

- The IEEE Standard for double precision is $FP(2, 53, -1023, 1023)$
  - Bit allocation: $64 = 1 + 11 + 53$

IEEE Floating Point Standard

- This is the resolution of the crisis in the 1980's
  - ∗ Does it solve the non-robustness problem?
  - ∗ Slightly, because of rational design
  - ∗ It ensures portable code and predictable results: if it fails on one machine, it should fail on others!
  - ∗ Official Name: IEEE Standard 754 for Binary Floating-Point Arithmetic (1987)
- Floating Point System $FP(2, t)$:
  - ∗ All numbers of the form $\langle e, f \rangle$ where $|f| < 2^t$
  - ∗ DENOTE by $FP(2, t, e_{\min}, e_{\max})$ if, in addition, $e_{\min} \leq e \leq e_{\max}$
- The IEEE Standard for double precision is $FP(2, 53, -1023, 1023)$
  - ∗ Bit allocation: $64 = 1 + 11 + 53$

IEEE Floating Point Standard

- This is the resolution of the crisis in the 1980's

  ∗ Does it solve the non-robustness problem?

  ∗ Slightly, because of rational design

  ∗ It ensures portable code and predictable results: if it fails on one machine, it should fail on others!

  ∗ Official Name: IEEE Standard 754 for Binary Floating-Point Arithmetic (1987)

- Floating Point System $FP(2, t)$:

  ∗ All numbers of the form $\langle e, f \rangle$ where $|f| < 2^t$

  ∗ DENOTE by $FP(2, t, e_{\min}, e_{\max})$ if, in addition, $e_{\min} \leq e \leq e_{\max}$

- The IEEE Standard for double precision is $FP(2, 53, -1023, 1023)$

  ∗ Bit allocation: $64 = 1 + 11 + 53$

* For quadruple precision: $FP(2, 113, -16382, 16383)$
* For single precision: $FP(2, 24, -127, 127)$

- Rounding Modes
  * $round(x) \in \{\lfloor x \rfloor, \lceil x \rceil\}$
  * Rounding Modes: ceiling, floor, to-zero, away-zero, to-even
  * Round to nearest: with tie-breaker from above modes
  * Default mode: nearest/to-even

- Unit Round Off, $\mathbf{u}$
  * For double precision, $\mathbf{u} = 2^{-53}$
  * For single precision, $\mathbf{u} = 2^{-24}$
  * If $x \in \mathbb{R}$ (in range), then $round(x) = x(1 \pm 2\mathbf{u})$
  * When rounding to nearest, then $round(x) = x(1 \pm \mathbf{u})$

- Approximate Arithmetic Model

  * For quadruple precision: $FP(2, 113, -16382, 16383)$
  * For single precision: $FP(2, 24, -127, 127)$

- Rounding Modes
  * $round(x) \in \{\lfloor x \rfloor, \lceil x \rceil\}$
  * Rounding Modes: ceiling, floor, to-zero, away-zero, to-even
  * Round to nearest: with tie-breaker from above modes
  * Default mode: nearest/to-even

- Unit Round Off, $\mathbf{u}$
  * For double precision, $\mathbf{u} = 2^{-53}$
  * For single precision, $\mathbf{u} = 2^{-24}$
  * If $x \in \mathbb{R}$ (in range), then $round(x) = x(1 \pm 2\mathbf{u})$
  * When rounding to nearest, then $round(x) = x(1 \pm \mathbf{u})$

- Approximate Arithmetic Model

  * For quadruple precision: $FP(2, 113, -16382, 16383)$
  * For single precision: $FP(2, 24, -127, 127)$

- Rounding Modes
  * $round(x) \in \{\lfloor x \rfloor, \lceil x \rceil\}$
  * Rounding Modes: ceiling, floor, to-zero, away-zero, to-even
  * Round to nearest: with tie-breaker from above modes
  * Default mode: nearest/to-even

- Unit Round Off, $\mathbf{u}$
  * For double precision, $\mathbf{u} = 2^{-53}$
  * For single precision, $\mathbf{u} = 2^{-24}$
  * If $x \in \mathbb{R}$ (in range), then $round(x) = x(1 \pm 2\mathbf{u})$
  * When rounding to nearest, then $round(x) = x(1 \pm \mathbf{u})$

- Approximate Arithmetic Model

* For quadruple precision: $FP(2, 113, -16382, 16383)$
* For single precision: $FP(2, 24, -127, 127)$

- Rounding Modes
  * $round(x) \in \{\lfloor x \rfloor, \lceil x \rceil\}$
  * Rounding Modes: ceiling, floor, to-zero, away-zero, to-even
  * Round to nearest: with tie-breaker from above modes
  * Default mode: nearest/to-even

- Unit Round Off, $\mathbf{u}$
  * For double precision, $\mathbf{u} = 2^{-53}$
  * For single precision, $\mathbf{u} = 2^{-24}$
  * If $x \in \mathbb{R}$ (in range), then $round(x) = x(1 \pm 2\mathbf{u})$
  * When rounding to nearest, then $round(x) = x(1 \pm \mathbf{u})$

- Approximate Arithmetic Model

- ∗ For quadruple precision: $FP(2, 113, -16382, 16383)$
- ∗ For single precision: $FP(2, 24, -127, 127)$

- Rounding Modes
  - ∗ $round(x) \in \{\lfloor x \rfloor, \lceil x \rceil\}$
  - ∗ Rounding Modes: ceiling, floor, to-zero, away-zero, to-even
  - ∗ Round to nearest: with tie-breaker from above modes
  - ∗ Default mode: nearest/to-even

- Unit Round Off, $\mathbf{u}$
  - ∗ For double precision, $\mathbf{u} = 2^{-53}$
  - ∗ For single precision, $\mathbf{u} = 2^{-24}$
  - ∗ If $x \in \mathbb{R}$ (in range), then $round(x) = x(1 \pm 2\mathbf{u})$
  - ∗ When rounding to nearest, then $round(x) = x(1 \pm \mathbf{u})$

- Approximate Arithmetic Model

* Let $\circ \in \{+, -, \times, \div\}$ and $\circ'$ be the approximate version
* Strict model: $x \circ' y = round(x \circ y)$
* Standard model: $x \circ' y = (x \circ y)(1 \pm \mathbf{u})$
* Difference: if $x \circ y \in F$, then strict model requires $x \circ' y = x \circ y$

- What else from IEEE Standard?
  * Subnormal numbers
  * Special values $NaN, \pm\infty, \pm 0$
  * Unambiguous comparisons: $+0 = -0$, NaN incomparable, etc
  * 5 Exceptions: invalid, overflow, divide by 0, underflow, inexact

- EXERCISE:
  * In Core Library, under progs/ieee, you see some programs manipulating IEEE formats

- $*$ Let $\circ \in \{+, -, \times, \div\}$ and $\circ'$ be the approximate version
- $*$ Strict model: $x \circ' y = round(x \circ y)$
- $*$ Standard model: $x \circ' y = (x \circ y)(1 \pm \mathbf{u})$
- $*$ Difference: if $x \circ y \in F$, then strict model requires $x \circ' y = x \circ y$

- What else from IEEE Standard?
  - $*$ Subnormal numbers
  - $*$ Special values $NaN, \pm\infty, \pm 0$
  - $*$ Unambiguous comparisons: $+0 = -0$, NaN incomparable, etc
  - $*$ 5 Exceptions: invalid, overflow, divide by 0, underflow, inexact

- EXERCISE:
  - $*$ In Core Library, under progs/ieee, you see some programs manipulating IEEE formats

∗ Let $\circ \in \{+, -, \times, \div\}$ and $\circ'$ be the approximate version

∗ Strict model: $x \circ' y = round(x \circ y)$

∗ Standard model: $x \circ' y = (x \circ y)(1 \pm \mathbf{u})$

∗ Difference: if $x \circ y \in F$, then strict model requires $x \circ' y = x \circ y$

- What else from IEEE Standard?

∗ Subnormal numbers

∗ Special values $NaN, \pm\infty, \pm 0$

∗ Unambiguous comparisons: $+0 = -0$, NaN incomparable, etc

∗ 5 Exceptions: invalid, overflow, divide by 0, underflow, inexact

- EXERCISE:

∗ In Core Library, under progs/ieee, you see some programs manipulating IEEE formats

* Let $\circ \in \{+, -, \times, \div\}$ and $\circ'$ be the approximate version
* Strict model: $x \circ' y = round(x \circ y)$
* Standard model: $x \circ' y = (x \circ y)(1 \pm \mathbf{u})$
* Difference: if $x \circ y \in F$, then strict model requires $x \circ' y = x \circ y$

- What else from IEEE Standard?
  * Subnormal numbers
  * Special values $NaN, \pm\infty, \pm 0$
  * Unambiguous comparisons: $+0 = -0$, NaN incomparable, etc
  * 5 Exceptions: invalid, overflow, divide by 0, underflow, inexact

- EXERCISE:
  * In Core Library, under progs/ieee, you see some programs manipulating IEEE formats

Complexity Model for Bigfloats

- Let $M(n)$ be the complexity of multiplying two n-bit binary integers
  - $*$ Which model? Usually, Turing machines. Then $M(n) = O(n \lg n \lg \lg n)$ by Schonhage-Strassen
  - $*$ Extend $M(n)$ to real arguments: if $x \leq 0$, $M(x) = 0$, else, $M(x) = M(\lceil x \rceil)$
  - $*$ We prefer Schonhage's Pointer machines

- We write $M(n)$ as a parameter in complexity statements
  - $*$ This way, the results remain valid even for other multiplication algorithms such as Karatsuba's

- Anecdote: what is $M(n)$ in Java's bigInteger?
  - $*$ Karatsuba's algorithm: $T(n) = 3T(n/2) + n$

Complexity Model for Bigfloats

- Let $M(n)$ be the complexity of multiplying two n-bit binary integers
  - $*$ Which model? Usually, Turing machines. Then $M(n) = O(n \lg n \lg \lg n)$ by Schonhage-Strassen
  - $*$ Extend $M(n)$ to real arguments: if $x \leq 0$, $M(x) = 0$, else, $M(x) = M(\lceil x \rceil)$
  - $*$ We prefer Schonhage's Pointer machines
- We write $M(n)$ as a parameter in complexity statements
  - $*$ This way, the results remain valid even for other multiplication algorithms such as Karatsuba's
- Anecdote: what is $M(n)$ in Java's bigInteger?
  - $*$ Karatsuba's algorithm: $T(n) = 3T(n/2) + n$

Complexity Model for Bigfloats

- Let $M(n)$ be the complexity of multiplying two n-bit binary integers

  $*$ Which model? Usually, Turing machines. Then $M(n) = O(n \lg n \lg\lg n)$ by Schonhage-Strassen

  $*$ Extend $M(n)$ to real arguments: if $x \le 0$, $M(x) = 0$, else, $M(x) = M(\lceil x \rceil)$

  $*$ We prefer Schonhage's Pointer machines

- We write $M(n)$ as a parameter in complexity statements

  $*$ This way, the results remain valid even for other multiplication algorithms such as Karatsuba's

- Anecdote: what is $M(n)$ in Java's bigInteger?

  $*$ Karatsuba's algorithm: $T(n) = 3T(n/2) + n$

Complexity Model for Bigfloats

- Let $M(n)$ be the complexity of multiplying two n-bit binary integers

  $*$ Which model? Usually, Turing machines. Then $M(n) = O(n \lg n \lg \lg n)$ by Schonhage-Strassen

  $*$ Extend $M(n)$ to real arguments: if $x \leq 0$, $M(x) = 0$, else, $M(x) = M(\lceil x \rceil)$

  $*$ We prefer Schonhage's Pointer machines

- We write $M(n)$ as a parameter in complexity statements

  $*$ This way, the results remain valid even for other multiplication algorithms such as Karatsuba's

- Anecdote: what is $M(n)$ in Java's bigInteger?

  $*$ Karatsuba's algorithm: $T(n) = 3T(n/2) + n$

- Axioms for $M(n)$
  - 1. Superlinear: $M(n) \geq n$
  - 2. Monotone: $M(n)$ is monotone nondecreasing
  - 3. Regularity: $M(n)$ satisfies the "regularity condition" below

- Regularity: for all $\alpha \in (0,1)$
  - $\alpha^2 M(n) \leq M(\alpha n) \leq \alpha M(n)$ (ev. n) It is satisfied by $M(n) = O(n^2)$ or Karatsuba $M(n) = O(n^{\lg 3})$

- EXERCISE:
  - Implement Karatsuba's algorithm in Core Library using its bigInteger (actually from gmp). ONLY RULE: Only call the addition routine of bigInteger, but not multiplication, division or reciprocal

- Axioms for $M(n)$
  - $*$ 1. Superlinear: $M(n) \geq n$
  - $*$ 2. Monotone: $M(n)$ is monotone nondecreasing
  - $*$ 3. Regularity: $M(n)$ satisfies the "regularity condition" below

- Regularity: for all $\alpha \in (0, 1)$
  - $*$ $\alpha^2 M(n) \leq M(\alpha n) \leq \alpha M(n)$ (ev. n) It is satisfied by $M(n) = O(n^2)$ or Karatsuba $M(n) = O(n^{\lg 3})$

- EXERCISE:
  - $*$ Implement Karatsuba's algorithm in Core Library using its bigInteger (actually from gmp). ONLY RULE: Only call the addition routine of bigInteger, but not multiplication, division or reciprocal

- Axioms for $M(n)$
  - $*$ 1. Superlinear: $M(n) \geq n$
  - $*$ 2. Monotone: $M(n)$ is monotone nondecreasing
  - $*$ 3. Regularity: $M(n)$ satisfies the "regularity condition" below

- Regularity: for all $\alpha \in (0, 1)$
  - $*$ $\alpha^2 M(n) \leq M(\alpha n) \leq \alpha M(n)$ (ev. n) It is satisfied by $M(n) = O(n^2)$ or Karatsuba $M(n) = O(n^{\lg 3})$

- EXERCISE:
  - $*$ Implement Karatsuba's algorithm in Core Library using its bigInteger (actually from gmp). ONLY RULE: Only call the addition routine of bigInteger, but not multiplication, division or reciprocal

- Axioms for $M(n)$
  - $*$ 1. Superlinear: $M(n) \geq n$
  - $*$ 2. Monotone: $M(n)$ is monotone nondecreasing
  - $*$ 3. Regularity: $M(n)$ satisfies the "regularity condition" below

- Regularity: for all $\alpha \in (0, 1)$
  - $*$ $\alpha^2 M(n) \leq M(\alpha n) \leq \alpha M(n)$ (ev. n) It is satisfied by $M(n) = O(n^2)$ or Karatsuba $M(n) = O(n^{\lg 3})$

- EXERCISE:
  - $*$ Implement Karatsuba's algorithm in Core Library using its bigInteger (actually from gmp). ONLY RULE: Only call the addition routine of bigInteger, but not multiplication, division or reciprocal

Error Notations

- "The art of error analysis consists of a good notation"

- Meta-notation

  * The symbol "$\pm$" mus always be rewritten as "$+\theta$" where θ is a variable satisfying $|\theta| \le 1$
  * E.g., $x \pm u$ is rewritten as $x + \theta u$
  * E.g., $(x \pm u)(y \pm u)$ is the same as $(x + \theta u)(y + \theta' u)$

- Let "$[x]_n$" be a short hand for "$x(1 \pm 2^{-n})$"

  * E.g., Write $[x+y]_n$, $[x-y]_n$, $[xy]_n, \ldots$ for the truncated relative precision arithmetic
  * E.g., $[x \circ y]_n = (x \circ y)(1 \pm 2^{-n})$.

- Similar notation for absolute error:

  * $\{x\}_n = x \pm 2^{-n} = x + \theta 2^{-n}$

Error Notations

- "The art of error analysis consists of a good notation"

- Meta-notation
  - $*$ The symbol "$\pm$" mus always be rewritten as "$+\theta$" where θ is a variable satisfying $|\theta| \leq 1$
  - $*$ E.g., $x \pm u$ is rewritten as $x + \theta u$
  - $*$ E.g., $(x \pm u)(y \pm u)$ is the same as $(x + \theta u)(y + \theta' u)$

- Let "$[x]_n$" be a short hand for "$x(1 \pm 2^{-n})$"
  - $*$ E.g., Write $[x+y]_n$, $[x-y]_n$, $[xy]_n, \ldots$ for the truncated relative precision arithmetic
  - $*$ E.g., $[x \circ y]_n = (x \circ y)(1 \pm 2^{-n})$.

- Similar notation for absolute error:
  - $*$ $\{x\}_n = x \pm 2^{-n} = x + \theta 2^{-n}$

Error Notations

- "The art of error analysis consists of a good notation"

- Meta-notation

  ∗ The symbol "$\pm$" mus always be rewritten as "$+\theta$" where θ is a variable satisfying $|\theta| \le 1$

  ∗ E.g., $x \pm u$ is rewritten as $x + \theta u$

  ∗ E.g., $(x \pm u)(y \pm u)$ is the same as $(x + \theta u)(y + \theta' u)$

- Let "$[x]_n$" be a short hand for "$x(1 \pm 2^{-n})$"

  ∗ E.g., Write $[x + y]_n$, $[x - y]_n$, $[xy]_n, \ldots$ for the truncated relative precision arithmetic

  ∗ E.g., $[x \circ y]_n = (x \circ y)(1 \pm 2^{-n})$.

- Similar notation for absolute error:

  ∗ $\{x\}_n = x \pm 2^{-n} = x + \theta 2^{-n}$

Error Notations

- "The art of error analysis consists of a good notation"

- Meta-notation
  - * The symbol "$\pm$" mus always be rewritten as "$+\theta$" where θ is a variable satisfying $|\theta| \leq 1$
  - * E.g., $x \pm u$ is rewritten as $x + \theta u$
  - * E.g., $(x \pm u)(y \pm u)$ is the same as $(x + \theta u)(y + \theta' u)$

- Let "$[x]_n$" be a short hand for "$x(1 \pm 2^{-n})$"
  - * E.g., Write $[x+y]_n$, $[x-y]_n$, $[xy]_n, \ldots$ for the truncated relative precision arithmetic
  - * E.g., $[x \circ y]_n = (x \circ y)(1 \pm 2^{-n})$.

- Similar notation for absolute error:
  - * $\{x\}_n = x \pm 2^{-n} = x + \theta 2^{-n}$

- $*$ E.g., $\{x+y\}_n, \quad \{x-y\}_n, \quad \{xy\}_n, \ldots.$

- CLAIM: To compute xy to precision n, suffices to compute $[x]_{n+4}$ and $[y]_{n+4}$, and then multiply them together with precision $n+2$. Proof:

$$\begin{aligned}
[[x]_{n+4}[y]_{n+4}]_{n+1} &= [x(1 \pm 2^{-n-4})y(1 \pm 2^{-n-4})]_{n+2} \\
&= [xy(1 \pm 2^{-n-2})]_{n+2} \\
&= xy(1 \pm 2^{-n-2})(1 \pm 2^{-n-2}) \\
&= xy(1 \pm 2^{-n}).
\end{aligned}$$

- THEOREM: Let x, y be bounded bigfloats
  - $*$ (I) We can compute $[x]_n$ in $O(n)$ time
  - $*$ (II) We can compute $[xy]_n$ in $O(M(n))$ time
  - $*$ (III) We can compute $[x \pm y]_n$ in time $O(n_x + n_y)$ where n_x, n_y are the precision of x, y

* E.g., $\{x+y\}_n$, $\{x-y\}_n$, $\{xy\}_n, \ldots$.

- CLAIM: To compute xy to precision n, suffices to compute $[x]_{n+4}$ and $[y]_{n+4}$, and then multiply them together with precision $n+2$. Proof:

$$
\begin{aligned}
[[x]_{n+4}[y]_{n+4}]_{n+1} &= [x(1\pm 2^{-n-4})y(1\pm 2^{-n-4})]_{n+2} \\
&= [xy(1\pm 2^{-n-2})]_{n+2} \\
&= xy(1\pm 2^{-n-2})(1\pm 2^{-n-2}) \\
&= xy(1\pm 2^{-n}).
\end{aligned}
$$

- THEOREM: Let x, y be bounded bigfloats
  * (I) We can compute $[x]_n$ in $O(n)$ time
  * (II) We can compute $[xy]_n$ in $O(M(n))$ time
  * (III) We can compute $[x\pm y]_n$ in time $O(n_x+n_y)$ where n_x, n_y are the precision of x, y

* E.g., $\{x+y\}_n, \quad \{x-y\}_n, \quad \{xy\}_n, \ldots.$

- CLAIM: To compute xy to precision n, suffices to compute $[x]_{n+4}$ and $[y]_{n+4}$, and then multiply them together with precision $n+2$. Proof:

$$
\begin{aligned}
[[x]_{n+4}[y]_{n+4}]_{n+1} &= [x(1 \pm 2^{-n-4})y(1 \pm 2^{-n-4})]_{n+2} \\
&= [xy(1 \pm 2^{-n-2})]_{n+2} \\
&= xy(1 \pm 2^{-n-2})(1 \pm 2^{-n-2}) \\
&= xy(1 \pm 2^{-n}).
\end{aligned}
$$

- THEOREM: Let x, y be bounded bigfloats
  * (I) We can compute $[x]_n$ in $O(n)$ time
  * (II) We can compute $[xy]_n$ in $O(M(n))$ time
  * (III) We can compute $[x \pm y]_n$ in time $O(n_x + n_y)$ where n_x, n_y are the precision of x, y

∗ E.g., $\{x+y\}_n$, $\{x-y\}_n$, $\{xy\}_n, \ldots$.

- CLAIM: To compute xy to precision n, suffices to compute $[x]_{n+4}$ and $[y]_{n+4}$, and then multiply them together with precision $n+2$. Proof:

$$\begin{aligned}
[[x]_{n+4}[y]_{n+4}]_{n+1} &= [x(1 \pm 2^{-n-4})y(1 \pm 2^{-n-4})]_{n+2} \\
&= [xy(1 \pm 2^{-n-2})]_{n+2} \\
&= xy(1 \pm 2^{-n-2})(1 \pm 2^{-n-2}) \\
&= xy(1 \pm 2^{-n}).
\end{aligned}$$

- THEOREM: Let x, y be bounded bigfloats
  ∗ (I) We can compute $[x]_n$ in $O(n)$ time
  ∗ (II) We can compute $[xy]_n$ in $O(M(n))$ time
  ∗ (III) We can compute $[x \pm y]_n$ in time $O(n_x + n_y)$ where n_x, n_y are the precision of x, y

- Proof (I): Let $x = \langle e_x, f_x \rangle$

  $*$ To compute $[x]_n$, we truncate f_x to at most $(n+1)$-bits Do repeated decrements of a binary counter with initial value n. With each decrement, we copy the next bit of f_x. The bits are copied starting from the most significant bit. We stop when we reach the least significant bit of f_x or when the counter reaches 0. The complexity of decrementing the counter is $O(n)$. We return at most $n+1$ bits of f_x that have been copied.

  $*$ NOTE: this bound is sublinear, and works only in pointer model!

- EXERCISE:

  $*$ (1) Verify the regularity condition when $M(n) = Cn \lg n \lg\lg n$

  $*$ (2) Verify for $M(n) = Cn^c$ (you must determine the

- Proof (I): Let $x = \langle e_x, f_x \rangle$

  $*$ To compute $[x]_n$, we truncate f_x to at most $(n+1)$-bits Do repeated decrements of a binary counter with initial value n. With each decrement, we copy the next bit of f_x. The bits are copied starting from the most significant bit. We stop when we reach the least significant bit of f_x or when the counter reaches 0. The complexity of decrementing the counter is $O(n)$. We return at most $n+1$ bits of f_x that have been copied.

  $*$ NOTE: this bound is sublinear, and works only in pointer model!

- EXERCISE:

  $*$ (1) Verify the regularity condition when $M(n) = Cn \lg n \lg\lg n$

  $*$ (2) Verify for $M(n) = Cn^c$ (you must determine the

- Proof (I): Let $x = \langle e_x, f_x \rangle$

  $*$ To compute $[x]_n$, we truncate f_x to at most $(n+1)$-bits Do repeated decrements of a binary counter with initial value n. With each decrement, we copy the next bit of f_x. The bits are copied starting from the most significant bit. We stop when we reach the least significant bit of f_x or when the counter reaches 0. The complexity of decrementing the counter is $O(n)$. We return at most $n+1$ bits of f_x that have been copied.

  $*$ NOTE: this bound is sublinear, and works only in pointer model!

- EXERCISE:

  $*$ (1) Verify the regularity condition when $M(n) = Cn \lg n \lg\lg n$

  $*$ (2) Verify for $M(n) = Cn^c$ (you must determine the

- Proof (I): Let $x = \langle e_x, f_x \rangle$

  $*$ To compute $[x]_n$, we truncate f_x to at most $(n+1)$-bits Do repeated decrements of a binary counter with initial value n. With each decrement, we copy the next bit of f_x. The bits are copied starting from the most significant bit. We stop when we reach the least significant bit of f_x or when the counter reaches 0. The complexity of decrementing the counter is $O(n)$. We return at most $n+1$ bits of f_x that have been copied.

  $*$ NOTE: this bound is sublinear, and works only in pointer model!

- EXERCISE:

  $*$ (1) Verify the regularity condition when $M(n) = Cn \lg n \lg\lg n$

  $*$ (2) Verify for $M(n) = Cn^c$ (you must determine the

range of the constant c in this case)

* (3) Redo the above theorem for unbounded bigfloats

range of the constant c in this case)

* (3) Redo the above theorem for unbounded bigfloats

Reciprocal

- Warm up, consider how to efficiently compute the reciprocal of a bigfloat number $c \neq 0$

- LEMMA: For $\alpha \in (0,1)$, we have $\sum_{j=0}^{\infty} M(\alpha^j n) = O(M(n))$.
  - Note: Regularity removes a factor of $\log n$

- THEOREM: For a bounded bigfloat c, we can compute $[1/c]_n$ in time $O(M(n))$
  - Let $f(x) = \frac{1}{x} - c$. Then Newton's iterator is $N(x) = x - f(x)/f'(x) = x(2 - cx)$.

  STANDARD PROOF: Our iteration is

$$x_{i+1} = x_i(2 - cx_i). \tag{1}$$

Reciprocal

- Warm up, consider how to efficiently compute the reciprocal of a bigfloat number $c \neq 0$

- LEMMA: For $\alpha \in (0,1)$, we have $\sum_{j=0}^{\infty} M(\alpha^j n) = O(M(n))$.
  - Note: Regularity removes a factor of $\log n$

- THEOREM: For a bounded bigfloat c, we can compute $[1/c]_n$ in time $O(M(n))$
  - Let $f(x) = \frac{1}{x} - c$. Then Newton's iterator is $N(x) = x - f(x)/f'(x) = x(2 - cx)$.

  STANDARD PROOF: Our iteration is

$$x_{i+1} = x_i(2 - cx_i). \tag{1}$$

Reciprocal

- Warm up, consider how to efficiently compute the reciprocal of a bigfloat number $c \neq 0$

- LEMMA: For $\alpha \in (0,1)$, we have $\sum_{j=0}^{\infty} M(\alpha^j n) = O(M(n))$.
  - Note: Regularity removes a factor of $\log n$

- THEOREM: For a bounded bigfloat c, we can compute $[1/c]_n$ in time $O(M(n))$
  - Let $f(x) = \frac{1}{x} - c$. Then Newton's iterator is $N(x) = x - f(x)/f'(x) = x(2 - cx)$.

  STANDARD PROOF: Our iteration is

$$x_{i+1} = x_i(2 - cx_i). \tag{1}$$

Reciprocal

- Warm up, consider how to efficiently compute the reciprocal of a bigfloat number $c \neq 0$

- LEMMA: For $\alpha \in (0,1)$, we have $\sum_{j=0}^{\infty} M(\alpha^j n) = O(M(n))$.
  - Note: Regularity removes a factor of $\log n$

- THEOREM: For a bounded bigfloat c, we can compute $[1/c]_n$ in time $O(M(n))$
  - Let $f(x) = \frac{1}{x} - c$. Then Newton's iterator is $N(x) = x - f(x)/f'(x) = x(2 - cx)$.

  STANDARD PROOF: Our iteration is

$$x_{i+1} = x_i(2 - cx_i). \tag{1}$$

If $x_i = (1 - \varepsilon_i)/c$ or $cx_i = 1 - \varepsilon_i$ then substituting into (1) gives $x_{i+1} = (1 - \varepsilon_i^2)/c$. Hence $\varepsilon_{i+1} = \varepsilon_i^2$. Assuming $|\varepsilon_0| < 1/2$, we conclude that $|\varepsilon_i| < 2^{-2^i}$ for all $i \geq 0$. Let $y = 1/c$ and $\widetilde{y} = x_k$ where $k = \lceil \lg n \rceil$. Then $|y - \widetilde{c}| = \varepsilon_k/c < 2^{-n}/c = O(2^{-n})$ (since c is bounded).

- The analysis assumes error-free operations!

  $*$ To include error in each iteration, the last step needs (at least) precision n

  $*$ Exploit self-correction property of Newton iteration: the previous step only requires about precision $n/2$, etc

  $*$ Suppose that the i-th iteration is carried out with precision 2^{i+1}.

  $*$ Complexity is $\sum_{i=0}^{\lg n} M(n2^{-i}) = O(M(n))$

- Here is the algorithm for approximate operations

If $x_i = (1-\varepsilon_i)/c$ or $cx_i = 1-\varepsilon_i$ then substituting into (1) gives $x_{i+1} = (1-\varepsilon_i^2)/c$. Hence $\varepsilon_{i+1} = \varepsilon_i^2$. Assuming $|\varepsilon_0| < 1/2$, we conclude that $|\varepsilon_i| < 2^{-2^i}$ for all $i \geq 0$. Let $y = 1/c$ and $\widetilde{y} = x_k$ where $k = \lceil \lg n \rceil$. Then $|y - \widetilde{c}| = \varepsilon_k/c < 2^{-n}/c = O(2^{-n})$ (since c is bounded).

- The analysis assumes error-free operations!

  $*$ To include error in each iteration, the last step needs (at least) precision n

  $*$ Exploit self-correction property of Newton iteration: the previous step only requires about precision $n/2$, etc

  $*$ Suppose that the i-th iteration is carried out with precision 2^{i+1}.

  $*$ Complexity is $\sum_{i=0}^{\lg n} M(n2^{-i}) = O(M(n))$

- Here is the algorithm for approximate operations

If $x_i = (1 - \varepsilon_i)/c$ or $cx_i = 1 - \varepsilon_i$ then substituting into (1) gives $x_{i+1} = (1 - \varepsilon_i^2)/c$. Hence $\varepsilon_{i+1} = \varepsilon_i^2$. Assuming $|\varepsilon_0| < 1/2$, we conclude that $|\varepsilon_i| < 2^{-2^i}$ for all $i \geq 0$. Let $y = 1/c$ and $\widetilde{y} = x_k$ where $k = \lceil \lg n \rceil$. Then $|y - \widetilde{c}| = \varepsilon_k/c < 2^{-n}/c = O(2^{-n})$ (since c is bounded).

- The analysis assumes error-free operations!

  $*$ To include error in each iteration, the last step needs (at least) precision n

  $*$ Exploit self-correction property of Newton iteration: the previous step only requires about precision $n/2$, etc

  $*$ Suppose that the i-th iteration is carried out with precision 2^{i+1}.

  $*$ Complexity is $\sum_{i=0}^{\lg n} M(n2^{-i}) = O(M(n))$

- Here is the algorithm for approximate operations

ITERATION $\widetilde{x}_i \longrightarrow \widetilde{x}_{i+1}$:

0. $\widetilde{c} \leftarrow [c]_{1+2^{i+1}}$
1. $y_i \leftarrow [\widetilde{c}\widetilde{x}_i]_{1+2^{i+1}}$
2. $z_i \leftarrow 2 - y_i$
3. $\widetilde{x}_{i+1} \leftarrow [\widetilde{x}_i z_i]_{2^{i+1}}$

∗ Each step is $O(M(2^{i+1}))$. So the overall complexity, by above lemma, is $O(M(n))$ to compute $[1/c]_n$

- Analysis: initially only pay attention to the first order terms (underline the second order terms)
  - ∗ Let $c\widetilde{x}_i = 1 \pm \delta_i$ where $\delta_i = 2^{2^i-1}3^{-2^i} = \frac{1}{2}(2/3)^{2^i}$
- EXERCISE:
  - ∗ (1) Extend the above result to an unbounded bigfloat c

$$\begin{array}{|l|}\hline \\ \textsc{Iteration}\ \widetilde{x}_i \longrightarrow \widetilde{x}_{i+1}: \\ \text{0.} \quad \widetilde{c} \leftarrow [c]_{1+2^{i+1}} \\ \text{1.} \quad y_i \leftarrow [\widetilde{c}\widetilde{x}_i]_{1+2^{i+1}} \\ \text{2.} \quad z_i \leftarrow 2 - y_i \\ \text{3.} \quad \widetilde{x}_{i+1} \leftarrow [\widetilde{x}_i z_i]_{2^{i+1}} \\ \hline \end{array}$$

* Each step is $O(M(2^{i+1}))$. So the overall complexity, by above lemma, is $O(M(n))$ to compute $[1/c]_n$

- Analysis: initially only pay attention to the first order terms (underline the second order terms)
  * Let $c\widetilde{x}_i = 1 \pm \delta_i$ where $\delta_i = 2^{2^i-1}3^{-2^i} = \frac{1}{2}(2/3)^{2^i}$

- EXERCISE:
  * (1) Extend the above result to an unbounded bigfloat c

ITERATION $\widetilde{x}_i \longrightarrow \widetilde{x}_{i+1}$:

0. $\widetilde{c} \leftarrow [c]_{1+2^{i+1}}$
1. $y_i \leftarrow [\widetilde{c\widetilde{x}_i}]_{1+2^{i+1}}$
2. $z_i \leftarrow 2 - y_i$
3. $\widetilde{x}_{i+1} \leftarrow [\widetilde{x}_i z_i]_{2^{i+1}}$

∗ Each step is $O(M(2^{i+1}))$. So the overall complexity, by above lemma, is $O(M(n))$ to compute $[1/c]_n$

- Analysis: initially only pay attention to the first order terms (underline the second order terms)
  - ∗ Let $c\widetilde{x}_i = 1 \pm \delta_i$ where $\delta_i = 2^{2^i - 1} 3^{-2^i} = \frac{1}{2}(2/3)^{2^i}$
- EXERCISE:
  - ∗ (1) Extend the above result to an unbounded bigfloat c

$*$ (2) Give an analogous analysis Newton's iteration for square root of a bounded bigfloat c

$*$ (3) Extend question (2) to unbounded c

$*$ (4) We said that to implement the approximate Newton iteration for computing $[1/c]_n$, we must begin with x_0 such that $|x_0-(1/c)| < 2/9$. How can you achieve this in practice? In theory?

$*$ (2) Give an analogous analysis Newton's iteration for square root of a bounded bigfloat c

$*$ (3) Extend question (2) to unbounded c

$*$ (4) We said that to implement the approximate Newton iteration for computing $[1/c]_n$, we must begin with x_0 such that $|x_0 - (1/c)| < 2/9$. How can you achieve this in practice? In theory?

Approximate Zeros

- Let $N_f(x) = x - f(x)/f'(x)$ be Newton iterator
  - $*$ When does Newton converge?
  - $*$ All numerical analysis books say: when close enough to a zero

- THEOREM [Yap-Fundamental-bk, Chapter 6]:
  - $*$ Let $f(X) \in \mathbb{Z}[X]$ be square-free of degree m and height H.
  - $*$ If $|x_0 - x^*| \leq (m^{3m+9}(2+H)^{6m})^{-1}$ then Newton will converge quadratically from x_0 to a root x^*

- Application to approximate real roots
  - $*$ 1. Use Sturm until root is isolated
  - $*$ 2. Use bisection until read the bound above

Approximate Zeros

- Let $N_f(x) = x - f(x)/f'(x)$ be Newton iterator
  - $*$ When does Newton converge?
  - $*$ All numerical analysis books say: when close enough to a zero

- THEOREM [Yap-Fundamental-bk, Chapter 6]:
  - $*$ Let $f(X) \in \mathbb{Z}[X]$ be square-free of degree m and height H.
  - $*$ If $|x_0 - x^*| \leq (m^{3m+9}(2+H)^{6m})^{-1}$ then Newton will converge quadratically from x_0 to a root x^*

- Application to approximate real roots
  - $*$ 1. Use Sturm until root is isolated
  - $*$ 2. Use bisection until read the bound above

Approximate Zeros

- Let $N_f(x) = x - f(x)/f'(x)$ be Newton iterator
  - $*$ When does Newton converge?
  - $*$ All numerical analysis books say: when close enough to a zero

- THEOREM [Yap-Fundamental-bk, Chapter 6]:
  - $*$ Let $f(X) \in \mathbb{Z}[X]$ be square-free of degree m and height H.
  - $*$ If $|x_0 - x^*| \leq (m^{3m+9}(2+H)^{6m})^{-1}$ then Newton will converge quadratically from x_0 to a root x^*

- Application to approximate real roots
  - $*$ 1. Use Sturm until root is isolated
  - $*$ 2. Use bisection until read the bound above

* 3. Use Newton
* NOTE: In Core Library, we try to skip step 2!

- Point Estimates of Smale: introduce notation
  * $\gamma(f,x) := \sup_{k\geq 2} \left|\frac{f^{(k)}(x)}{k!f'(x)}\right|^{1/(k-1)}$
  * $\beta(f,x) := \left|\frac{f(x)}{f'(x)}\right|$
  * $\alpha(f,x) := \beta(f,x)\gamma(f,x)$

- Also:
  * $\psi(x) := 1 - 4x + 2x^2$. The roots are $(2 \pm \sqrt{2})/2$.
  * $u(z,w) := \gamma(f,w)|z-w|$.
  * When $z = z^*$, a root of f, we write $u_w := u(z^*,w) = \gamma(f,z^*)|z^*-w|$.

- Initial intuition:
  * $|N_f'(x)| = \left|\frac{f(x)f''(x)}{f'(x)^2}\right| = 2\left|\frac{f(x)}{f'(x)} \cdot \frac{f''(x)}{2!f'(x)}\right| \leq 2\alpha(f,x)$.

  - ∗ 3. Use Newton
  - ∗ NOTE: In Core Library, we try to skip step 2!

- Point Estimates of Smale: introduce notation
  - ∗ $\gamma(f,x) := \sup_{k\ge 2} \left|\frac{f^{(k)}(x)}{k!f'(x)}\right|^{1/(k-1)}$
  - ∗ $\beta(f,x) := \left|\frac{f(x)}{f'(x)}\right|$
  - ∗ $\alpha(f,x) := \beta(f,x)\gamma(f,x)$

- Also:
  - ∗ $\psi(x) := 1 - 4x + 2x^2$. The roots are $(2 \pm \sqrt{2})/2$.
  - ∗ $u(z,w) := \gamma(f,w)|z-w|$.
  - ∗ When $z = z^*$, a root of f, we write $u_w := u(z^*,w) = \gamma(f,z^*)|z^*-w|$.

- Initial intuition:
  - ∗ $|N_f'(x)| = \left|\frac{f(x)f''(x)}{f'(x)^2}\right| = 2\left|\frac{f(x)}{f'(x)}\cdot\frac{f''(x)}{2!f'(x)}\right| \le 2\alpha(f,x)$.

- 3. Use Newton
- NOTE: In Core Library, we try to skip step 2!

- Point Estimates of Smale: introduce notation
  - $\gamma(f,x) := \sup_{k\geq 2}\left|\frac{f^{(k)}(x)}{k!f'(x)}\right|^{1/(k-1)}$
  - $\beta(f,x) := \left|\frac{f(x)}{f'(x)}\right|$
  - $\alpha(f,x) := \beta(f,x)\gamma(f,x)$

- Also:
  - $\psi(x) := 1 - 4x + 2x^2$. The roots are $(2 \pm \sqrt{2})/2$.
  - $u(z,w) := \gamma(f,w)|z-w|$.
  - When $z = z^*$, a root of f, we write $u_w := u(z^*,w) = \gamma(f,z^*)|z^*-w|$.

- Initial intuition:
  - $|N_f'(x)| = \left|\frac{f(x)f''(x)}{f'(x)^2}\right| = 2\left|\frac{f(x)}{f'(x)}\cdot\frac{f''(x)}{2!f'(x)}\right| \leq 2\alpha(f,x)$.

- ∗ 3. Use Newton
- ∗ NOTE: In Core Library, we try to skip step 2!

- Point Estimates of Smale: introduce notation
  - ∗ $\gamma(f,x) := \sup_{k\ge 2} \left|\frac{f^{(k)}(x)}{k!f'(x)}\right|^{1/(k-1)}$
  - ∗ $\beta(f,x) := \left|\frac{f(x)}{f'(x)}\right|$
  - ∗ $\alpha(f,x) := \beta(f,x)\gamma(f,x)$

- Also:
  - ∗ $\psi(x) := 1 - 4x + 2x^2$. The roots are $(2 \pm \sqrt{2})/2$.
  - ∗ $u(z,w) := \gamma(f,w)|z-w|$.
  - ∗ When $z = z^*$, a root of f, we write $u_w := u(z^*,w) = \gamma(f,z^*)|z^*-w|$.

- Initial intuition:
  - ∗ $|N_f'(x)| = \left|\frac{f(x)f''(x)}{f'(x)^2}\right| = 2\left|\frac{f(x)}{f'(x)} \cdot \frac{f''(x)}{2!f'(x)}\right| \le 2\alpha(f,x)$.

* This suggest that if $\alpha(f,x)$ is sufficiently small, then N_f is a contraction map (Lipschitz constant < 1)
* It seems that $\alpha(f,x) < 1/2$ is a necessary condition

- Notion of Approximate Zero:
  * Let $x = x_0$ and $x_{i+1} = N_f(x_i)$
  * Call x_0 an approximate zero of f if $|x_n - x^*| \leq \left(\frac{1}{2}\right)^{2^n - 1} |x_0 - x^*|, \qquad n \geq 0.$

- THEOREM (Smale)
  * If $\alpha(f,x) < 0.04$ then x is an approximate zero of f

- This is called a point estimate
  * Compare: Kantorovich's approach

- EXERCISE (open)
  * (1) Give a simple proof of Smale's theorem. (2)

* This suggest that if $\alpha(f,x)$ is sufficiently small, then N_f is a contraction map (Lipschitz constant < 1)
* It seems that $\alpha(f,x) < 1/2$ is a necessary condition

- Notion of Approximate Zero:
  * Let $x = x_0$ and $x_{i+1} = N_f(x_i)$
  * Call x_0 an approximate zero of f if $|x_n - x^*| \leq \left(\frac{1}{2}\right)^{2^n-1} |x_0 - x^*|, \qquad n \geq 0.$

- THEOREM (Smale)
  * If $\alpha(f,x) < 0.04$ then x is an approximate zero of f

- This is called a point estimate
  * Compare: Kantorovich's approach

- EXERCISE (open)
  * (1) Give a simple proof of Smale's theorem. (2)

* This suggest that if $\alpha(f,x)$ is sufficiently small, then N_f is a contraction map (Lipschitz constant < 1)
* It seems that $\alpha(f,x) < 1/2$ is a necessary condition

- Notion of Approximate Zero:
  * Let $x = x_0$ and $x_{i+1} = N_f(x_i)$
  * Call x_0 an approximate zero of f if $|x_n - x^*| \le \left(\frac{1}{2}\right)^{2^n-1} |x_0 - x^*|, \qquad n \ge 0.$

- THEOREM (Smale)
  * If $\alpha(f,x) < 0.04$ then x is an approximate zero of f

- This is called a point estimate
  * Compare: Kantorovich's approach

- EXERCISE (open)
  * (1) Give a simple proof of Smale's theorem. (2)

* This suggest that if $\alpha(f,x)$ is sufficiently small, then N_f is a contraction map (Lipschitz constant < 1)
* It seems that $\alpha(f,x) < 1/2$ is a necessary condition

- Notion of Approximate Zero:
  * Let $x = x_0$ and $x_{i+1} = N_f(x_i)$
  * Call x_0 an approximate zero of f if $|x_n - x^*| \leq \left(\frac{1}{2}\right)^{2^n-1} |x_0 - x^*|, \qquad n \geq 0.$

- THEOREM (Smale)
  * If $\alpha(f,x) < 0.04$ then x is an approximate zero of f

- This is called a point estimate
  * Compare: Kantorovich's approach

- EXERCISE (open)
  * (1) Give a simple proof of Smale's theorem. (2)

* This suggest that if $\alpha(f, x)$ is sufficiently small, then N_f is a contraction map (Lipschitz constant < 1)
* It seems that $\alpha(f, x) < 1/2$ is a necessary condition

- Notion of Approximate Zero:
  * Let $x = x_0$ and $x_{i+1} = N_f(x_i)$
  * Call x_0 an approximate zero of f if $|x_n - x^*| \leq \left(\frac{1}{2}\right)^{2^n - 1} |x_0 - x^*|, \qquad n \geq 0.$

- THEOREM (Smale)
  * If $\alpha(f, x) < 0.04$ then x is an approximate zero of f

- This is called a point estimate
  * Compare: Kantorovich's approach

- EXERCISE (open)
  * (1) Give a simple proof of Smale's theorem. (2)

∗ This suggest that if $\alpha(f,x)$ is sufficiently small, then N_f is a contraction map (Lipschitz constant < 1)

∗ It seems that $\alpha(f,x) < 1/2$ is a necessary condition

- Notion of Approximate Zero:

  ∗ Let $x = x_0$ and $x_{i+1} = N_f(x_i)$

  ∗ Call x_0 an approximate zero of f if $|x_n - x^*| \leq \left(\frac{1}{2}\right)^{2^n-1} |x_0 - x^*|, \qquad n \geq 0.$

- THEOREM (Smale)

  ∗ If $\alpha(f,x) < 0.04$ then x is an approximate zero of f

- This is called a point estimate

  ∗ Compare: Kantorovich's approach

- EXERCISE (open)

  ∗ (1) Give a simple proof of Smale's theorem. (2)

Improve the point estimate of Smale.

Robust Approximate Zeros

- Let $N_{f,i,C}(z) := \{N_f(z)\}_{2^i+C}$

- A robust sequence relative to C is $(\widetilde{z}_i)_{i\geq 0}$ such that for $i \geq 1$,

$$\widetilde{z}_i = N_{f,i,C}(\widetilde{z}_{i-1}).$$

- REMARK: if $\widetilde{z}_{i-1} = \infty$ or $f'(\widetilde{z}_{i-1}) = 0$, then $\widetilde{z}_i = \infty$. Say the sequence is finite if $\widetilde{z}_i \neq \infty$ for all i.

- DEFINITION: z_0 is a robust approximate zero if there exists z^* such that for all C satisfying

$$2^{-C} \leq |\widetilde{z}_0 - z^*|,$$

every robust sequence $(\widetilde{z}_i)_{i\geq 0}$ of z_0 relative to C is finite and satisfies

$$|\widetilde{z}_i - z^*| \leq 2^{1-2^i}|z_0 - z^*|.$$

Comparison to Malajovich's Work

- Malajovich has the only work on error analysis in Smale's setting
- He treats the multi-variate Newton setting, but when specialized to univariate, the complexity results are not as strong as ours.
-

Robust Point Estimate

- Smale: if z^* is simple zero of f and z_0 satisfy

$$|z_0 - z^*| \le \frac{3-\sqrt{7}}{2\gamma(f,z^*)}$$

then z_0 is an approximate zero.

- THEOREM: if z^* is simple zero of f and z_0 satisfy

$$|z_0 - z^*| \le \frac{4-\sqrt{14}}{2\gamma(f,z^*)}$$

then z_0 is a robust approximate zero.

- THEOREM: If $\alpha(f, z_0) < 0.02$ then z_0 is a robust approximate zero of f.

  * We can estimate the associated zero z^* as within distance $0.07/\gamma(f, z_0)$ from z_0.

How to Approximate the Newton Iterator

- How to compute $\{N_f(\widetilde{z}_i)\}_{2^i+C}$ for given $\widetilde{z}_i$, i and C?

- THEOREM: To compute $\{N_f(\widetilde{z}_i)\}_{2^i+C}$, it suffices to
  (1) evaluate $f(\widetilde{z}_i)$ to $\kappa + 2^{i+1} + 4 + C$ absolute bits,
  (2) evaluate $f'(\widetilde{z}_i)$ to $\kappa' + 2^i + 3 + C$ absolute bits,
  (3) perform the division to $\kappa'' + 2^i + 1 + C$ relative bits,
  where
  (1)' $\kappa \geq -\lg|f'(z_0)|$,

(2)' $\kappa' \geq -\lg |f'(z_0)|\gamma(f, z_0)$,
(3)' $\kappa'' \geq -\lg |f'(z_0)| + 3$.

Estimating Distance to Associated Root

- We still need to compute C satisfying $C \geq -\lg|z_0 - z^*|$ or

$$0 \leq C + \lg|z_0 - z^*|.$$

- Kalantari: $|z_0 - z^*| \geq \frac{1}{2\gamma_2(f,z_0)}$ where

$$\gamma_2(f,z) := \sup_{k\geq 1}\left|\frac{f^{(k)}(z)}{k!f'(z)}\right|^{1/k}.$$

- Wanted: C such that

$$0 \leq C + \lg|z_0 - z^*| = O(1).$$

- We could use Turan's proximity test [Pan] to estimate $|z_0 - z^*|$ (but it does not exploit z_0 as an approximate zero, and applies to only to polynomials).

- LEMMA: If z_0 satisfies

$$u := \gamma(f, z^*)|z_0 - z^*| \leq 1 - \frac{1}{\sqrt{2}}$$

and z^* is a simple zero of f then

$$|z_0 - z^*|(1 - 2u)(1 - u) \leq \left|\frac{f(z_0)}{f'(z_0)}\right| |z_0 - z^*| \frac{(1 - u)}{\psi(u)}.$$

REMARK: $\psi(u)$ is bounded away from 0 and u is close to 0 so $\left|\frac{f(z_0)}{f'(z_0)}\right|$ is equal to $|z_0 - z^*|$ up to a constant factor.

- ALGORITHM D: approximate $\left|\frac{f(z_0)}{f'(z_0)}\right|$ up to a constant multiplicative factor.

  This is our estimate of C!

- REMARK: Asano-Kirkpatrick-Yap gives a general scheme for converting absolute approximation to relative approximation. This can be used here.

Complexity of Approximate Zeros of a Polynomial

- Starting from an approximate zero $\widetilde{z}_0$ with associated zero z^*, how expensive is it to compute n-bit approximation of z^*?

- Let $f(z)$ be square-free, degree d, with L-bit integer coefficients.
  * ASSUME z^* is real, and z_0 a bigfloat.

- THEOREM: If $\alpha(f, z_0) < 0.02$ and exponent has size s, then we can compute n-bit absolute precision approximations of z^* in time

$$O(\ dM\ \left[n + d^2(L + \lg d)\lg(n + L)\right]\ ).$$

- Corollary (Brent): If d, L is fixed, then it takes $O(M(n))$.

Conclusions

- Why is IEEE Arithmetic important for EGC?

  * Because it is FAST, and we implement our filtered arithmetic here!

- BigInteger Arithmetic is our base line for speed

  * BigFloats is essentially BigInteger speed + small overhead
  * BigRat is no good, really

- Brent's fundamental work is our starting point

- Several extensions are necessary:

  * (1) Extended to global complexity
  * (2) Incorporate inexact operations (3) Avoid asymptotic notation for error analysis

Conclusions

- Why is IEEE Arithmetic important for EGC?
  - Because it is FAST, and we implement our filtered arithmetic here!
- BigInteger Arithmetic is our base line for speed
  - BigFloats is essentially BigInteger speed + small overhead
  - BigRat is no good, really
- Brent's fundamental work is our starting point
- Several extensions are necessary:
  - (1) Extended to global complexity
  - (2) Incorporate inexact operations (3) Avoid asymptotic notation for error analysis

Conclusions

- Why is IEEE Arithmetic important for EGC?
  * Because it is FAST, and we implement our filtered arithmetic here!

- BigInteger Arithmetic is our base line for speed
  * BigFloats is essentially BigInteger speed + small overhead
  * BigRat is no good, really

- Brent's fundamental work is our starting point

- Several extensions are necessary:
  * (1) Extended to global complexity
  * (2) Incorporate inexact operations (3) Avoid asymptotic notation for error analysis

Conclusions

- Why is IEEE Arithmetic important for EGC?
  - ∗ Because it is FAST, and we implement our filtered arithmetic here!
- BigInteger Arithmetic is our base line for speed
  - ∗ BigFloats is essentially BigInteger speed + small overhead
  - ∗ BigRat is no good, really
- Brent's fundamental work is our starting point
- Several extensions are necessary:
  - ∗ (1) Extended to global complexity
  - ∗ (2) Incorporate inexact operations (3) Avoid asymptotic notation for error analysis

Conclusions

- Why is IEEE Arithmetic important for EGC?
  - ∗ Because it is FAST, and we implement our filtered arithmetic here!
- BigInteger Arithmetic is our base line for speed
  - ∗ BigFloats is essentially BigInteger speed + small overhead
  - ∗ BigRat is no good, really
- Brent's fundamental work is our starting point
- Several extensions are necessary:
  - ∗ (1) Extended to global complexity
  - ∗ (2) Incorporate inexact operations (3) Avoid asymptotic notation for error analysis

- For Newton iteration, Smale's work must also be extended to incorporate inexact operations

- For Newton iteration, Smale's work must also be extended to incorporate inexact operations

REFERENCE

- Chapter 2 of [Mehlhorn-Yap]
- Paper of Brent

"A rapacious monster lurks within every computer, and it dines exclusively on accurate digits."
– B.D. McCullough (2000)

THE END