Introduction to Scientific Computing Languages

Prof. **Paolo Bientinesi**

pauldj@aices.rwth-aachen.de

Numerical Representation

Numbers

- $\frac{123}{29} =$ (first 40 digits)
  $4.241379310344827586206896551724137931034\ldots$
- $\pi =$
  $3.141592653589793238462643383279502884197\ldots$
- In general: **Infinite** number of digits

Numerical Representation

Numbers

- $\frac{123}{29} =$ (first 40 digits)
  $4.241379310344827586206896551724137931034\ldots$
- $\pi =$
  $3.141592653589793238462643383279502884197\ldots$
- In general: **Infinite** number of digits

Computers

- **Finite** memory

Computers: Inexact Numbers

- **Infinite** numbers vs. **finite** memory
  $\Rightarrow$ **Approximated numbers**

Computers: Inexact Numbers

- **Infinite** numbers vs. **finite** memory
  $\Rightarrow$ **Approximated numbers**

How many digits? A pre-determined amount

π= 3.14159265358979323846264338327950288419 7...

Computers: Inexact Numbers

- **Infinite** numbers vs. **finite** memory
  $\Rightarrow$ **Approximated numbers**

How many digits? A pre-determined amount

π= **3.141**59265358979323846264338327950288419 7...

Using 4 digits: $\pi = 3.141$

- Modern computers: normally 8 or 16 digits, single / double precision.

Computers: Inexact Numbers

- **Infinite** numbers vs. **finite** memory
  $\Rightarrow$ **Approximated numbers**

How many digits? A pre-determined amount

π= **3.141**59265358979323846264338327950288419 7...

Using 4 digits: $\pi = 3.141$

- Modern computers: normally 8 or 16 digits, single / double precision.

Alternatively?

- Extended precision
- Variable precision
- Symbolic representation

Computers: Approximated Computations

4-digit representation

Inexact Arithmetic

$$\begin{array}{rl} 123.4 & + \\ .5678 & = \\ \hline 123.9678 & = \end{array}$$

Computers: Approximated Computations

4-digit representation

Inexact Arithmetic

$$\begin{array}{rc} 123.4 & + \\ .5678 & = \\ \hline 123.9\color{red}{678} & = \end{array}$$

Computers: Approximated Computations

4-digit representation

Inexact Arithmetic

	123.4	+
	.5678	=
	123.9678	=
Truncated	123.9	
Rounded	124.0	

Computers: Approximated Computations

4-digit representation

Inexact Arithmetic

	123.4	+
	.5678	=
	123.9678	=
Truncated	123.9	
Rounded	124.0	

Associativity?

- Exact arithmetic:
  (123.4 + .5678) + .5432 =
  123.4 + (.5678 + .5432)

Computers: Approximated Computations

4-digit representation

Inexact Arithmetic

	123.4	+
	.5678	=
	123.9678	=
Truncated	123.9	
Rounded	124.0	

Associativity?

- Exact arithmetic:
  (123.4 + .5678) + .5432 =
  123.4 + (.5678 + .5432)
- Inexact arithmetic:
  (123.4 + .5678) + .5432 = 124.4

Computers: Approximated Computations

4-digit representation

Inexact Arithmetic

	123.4	+
	.5678	=
	123.9678	=
Truncated	123.9	
Rounded	124.0	

Associativity? No!

- Exact arithmetic:
  (123.4 + .5678) + .5432 =
  123.4 + (.5678 + .5432)

- Inexact arithmetic:
  (123.4 + .5678) + .5432 = 124.4
  123.4 + (.5678 + .5432) = 124.5

Error Analysis

$$f : A \to B, \quad y = f(x)$$

Es.: $f(x) = x^2 + \sin(2 * x)$

$$x = \frac{\pi}{123}, \qquad f(x) = ?$$

Error Analysis

$$f : A \to B, \quad y = f(x)$$

Es.: $f(x) = x^2 + \sin(2 * x)$

$$x = \frac{\pi}{123}, \qquad f(x) = ?$$

- Exact arithmetic:

$$\left(\frac{\pi}{123}\right)^2 + \sin\left(2 * \frac{\pi}{123}\right) = ...$$

Error Analysis

$$f : A \to B, \quad y = f(x)$$

Es.: $f(x) = x^2 + \sin(2 * x)$

$$x = \frac{\pi}{123}, \qquad f(x) = ?$$

- Exact arithmetic:

$$\left(\frac{\pi}{123}\right)^2 + \sin\left(2 * \frac{\pi}{123}\right) = ...$$

- Inexact arithmetic:
  $x \rightsquigarrow \hat{x}, \quad f \rightsquigarrow \hat{f} \qquad \hat{f}(\hat{x})$ instead of $f(x)$

Known Disasters

Known Disasters

- Patriot Missile
  Scud launched from Iraq againt US military base in South Arabia.
  US Patriot's missile missed the incoming Scud.
  28 casualties.

  Cancellation

Known Disasters

- Patriot Missile
  Scud launched from Iraq againt US military base in South Arabia.
  US Patriot's missile missed the incoming Scud.
  28 casualties.

  Cancellation

- Spaceship Ariane 5
  launched in 1996,
  destroyed 37 seconds after liftoff.

  Overflow

`http://ta.twi.tudelft.nl/users/vuik/wi211/disasters.html`

Floating Point Numbers

$$y = \pm .d_1 d_2 \ldots d_t \times \beta^e$$

Floating Point Numbers

$$y = \pm .d_1 d_2 \ldots d_t \times \beta^e$$

- β: base (radix)

Floating Point Numbers

$$y = \pm .d_1 d_2 \ldots d_t \times \beta^e$$

- β: base (radix)
- t: precision = # slots for the mantissa $(d_1 d_2 \ldots d_t)$

Floating Point Numbers

$$y = \pm .d_1 d_2 \ldots d_t \times \beta^e$$

- β: base (radix)
- t: precision = # slots for the mantissa ($d_1 d_2 \ldots d_t$)
- $0 \leq d_i \leq \beta - 1$: digits

Floating Point Numbers

$$y = \pm .d_1 d_2 \dots d_t \times \beta^e$$

- β: base (radix)
- t: precision = # slots for the mantissa $(d_1 d_2 \dots d_t)$
- $0 \leq d_i \leq \beta - 1$: digits
- $e_{\min} \leq e \leq e_{\max}$: exponent range

Floating Point Numbers

$$y = \pm .d_1 d_2 \ldots d_t \times \beta^e$$

- β: base (radix)
- t: precision = # slots for the mantissa $(d_1 d_2 \ldots d_t)$
- $0 \leq d_i \leq \beta - 1$: digits
- $e_{\min} \leq e \leq e_{\max}$: exponent range
- Normalization: $d_1 = 1$; $\quad d_1$ used for the sign $\pm$

Floating Point Numbers

$$y = \pm .d_1 d_2 \ldots d_t \times \beta^e$$

- β: base (radix)
- t: precision = # slots for the mantissa $(d_1 d_2 \ldots d_t)$
- $0 \leq d_i \leq \beta - 1$: digits
- $e_{\min} \leq e \leq e_{\max}$: exponent range
- Normalization: $d_1 = 1$; d_1 used for the sign $\pm$

Arithmetic	β	t	$e_{\min}$	$e_{\max}$
Single precision	2	24	-125	128
Double precision	2	53	-1021	1024

Floating Point Numbers

$$y = \pm .d_1 d_2 \dots d_t \times \beta^e$$

- β: base (radix)
- t: precision = # slots for the mantissa ($d_1 d_2 \dots d_t$)
- $0 \le d_i \le \beta - 1$: digits
- $e_{\min} \le e \le e_{\max}$: exponent range
- Normalization: $d_1 = 1$; d_1 used for the sign $\pm$

Arithmetic	β	t	$e_{\min}$	$e_{\max}$
Single precision	2	24	-125	128
Double precision	2	53	-1021	1024
HP calculator	10	12	-499	499
IBM 3090	16	14	-63	64
Setun	3			
Quadruple prec.	2	113	-16381	16384

Floating Point Numbers (2)

- Floating point numbers are non-equidistant
- What are the nonnegative points in $\beta = 2, t = 3, e_{\min} = -1, e_{\max} = 3$?

Floating Point Numbers (2)

- Floating point numbers are non-equidistant
- What are the nonnegative points in $\beta = 2, t = 3, e_{\min} = -1, e_{\max} = 3$?
- How to represent 0?

Floating Point Numbers (2)

- Floating point numbers are non-equidistant
- What are the nonnegative points in $\beta = 2, t = 3, e_{\min} = -1, e_{\max} = 3$?
- How to represent 0?
- underflow? overflow? NaN?

Floating Point Numbers (2)

- Floating point numbers are non-equidistant
- What are the nonnegative points in $\beta = 2, t = 3, e_{\min} = -1, e_{\max} = 3$?
- How to represent 0?
- underflow? overflow? NaN?
- x and y are floating point numbers, $x \neq y$;
  $z := \frac{|x - y|}{max(|x|, |y|)}$ how small can z be?
- $\log_{10}(2^{52}) = 15.65 \approx 16$

Smallest number

Smallest number

x and y are floating point numbers, $x \neq y$;
how small can z be?

$$z := |x - y|$$

Smallest number

x and y are floating point numbers, $x \neq y$;
how small can z be?

$$z := |x - y| \quad \text{vs.} \quad z := \frac{|x - y|}{max(|x|, |y|)}$$

Smallest number

x and y are floating point numbers, $x \neq y$;
how small can z be?

$$z := |x - y| \quad \text{vs.} \quad z := \frac{|x - y|}{max(|x|, |y|)}$$

$z := |x - y|$

1) $.000 \cdots 1 \times 2^0 \quad = 2^{-t}$
2) $.000 \cdots 1 \times 2^{e_{\min}} \quad = 2^{e_{\min} - t}$
3) $.000 \cdots 1 \times 2^{e_{\min} + t} = 2^{e_{\min}}$

Smallest number

x and y are floating point numbers, $x \neq y$;
how small can z be?

$$z := |x - y| \quad \text{vs.} \quad z := \frac{|x-y|}{max(|x|, |y|)}$$

$z := |x - y|$

1) $.000 \cdots 1 \times 2^0 = 2^{-t}$
2) $.000 \cdots 1 \times 2^{e_{\min}} = 2^{e_{\min}-t}$
3) $.000 \cdots 1 \times 2^{e_{\min}+t} = 2^{e_{\min}}$

$z := \frac{|x-y|}{max(|x|, |y|)}$

1) $.000 \cdots 1 \times 2^0 = 2^{-t}$
2) $.000 \cdots 1 \times 2^{e_{\min}} = 2^{e_{\min}-t}$
3) $.000 \cdots 1 \times 2^{e_{\min}+t} = 2^{e_{\min}}$

Smallest number

x and y are floating point numbers, $x \neq y$;
how small can z be?

$$z := |x - y| \quad \text{vs.} \quad z := \frac{|x-y|}{max(|x|,|y|)}$$

$z := |x - y|$

1) $.000 \cdots 1 \times 2^0 = 2^{-t}$
2) $.000 \cdots 1 \times 2^{e_{\min}} = 2^{e_{\min}-t}$
3) $.000 \cdots 1 \times 2^{e_{\min}+t} = 2^{e_{\min}}$

$z := \frac{|x-y|}{max(|x|,|y|)}$

1) $.000 \cdots 1 \times 2^0 = 2^{-t}$
2) $.000 \cdots 1 \times 2^{e_{\min}} = 2^{e_{\min}-t}$
3) $.000 \cdots 1 \times 2^{e_{\min}+t} = 2^{e_{\min}}$

Case Studies

Bisection

- Recursive implementation
- Base case?

Case Studies

Bisection

- Recursive implementation
- Base case?

$\Pi^2/6$

$$\sum_{i=1}^{10000} \frac{1}{i^2} \qquad \text{vs.} \qquad \sum_{i=10000}^{1} \frac{1}{i^2}$$

Case Studies

Bisection

- Recursive implementation
- Base case?

$\Pi^2/6$

$$\sum_{i=1}^{10000} \frac{1}{i^2} \qquad \text{vs.} \qquad \sum_{i=10000}^{1} \frac{1}{i^2}$$

$Ax = b$

$$A = \begin{bmatrix} \epsilon & -1 \\ 1 & 1 \end{bmatrix}$$

Case Studies

Bisection

- Recursive implementation
- Base case?

$\Pi^2/6$

$$\sum_{i=1}^{10000} \frac{1}{i^2} \qquad \text{vs.} \qquad \sum_{i=10000}^{1} \frac{1}{i^2}$$

$Ax = b$

$$A = \begin{bmatrix} \epsilon & -1 \\ 1 & 1 \end{bmatrix}$$

`sqsqrt` vs. `sqrtsq`

Errors

Representation Errors

Roundoff Errors

Algorithmic Errors

Machine Precision

Machine Precision **u**

- Smallest positive number such that $[1 + \mathbf{u}] \neq 1$

Machine Precision

Machine Precision **u**

- Smallest positive number such that $[1 + \mathbf{u}] \neq 1$
- Largest positive number such that $[1 + \mathbf{u}] = 1$

Machine Precision

Machine Precision **u**

- Smallest positive number such that [1 + **u**] $\neq$ 1
- Largest positive number such that [1 + **u**] = 1
- $\frac{1}{2}\beta^{1-t}$

Machine Precision

Machine Precision **u**

- Smallest positive number such that [1 + **u**] $\neq$ 1
- Largest positive number such that [1 + **u**] = 1
- $\frac{1}{2}\beta^{1-t}$
- Distance between 1 and the next floating point number
- "Machine epsilon", ϵ_{M}, **u**

Representation Error

$f_{\min}$ = smallest positive floating point number

$f_{\max}$ = largest positive floating point number

$\bar{x} = [x]$ = floating point representation of x

Theorem:

Let $x \in \mathbb{R}$ and $x \in [f_{\min}, f_{\max}]$

Then $\bar{x} = x(1 + \delta_1)$ where $|\delta_1| \leq$ **u**

Representation Error

$f_{\min}$ = smallest positive floating point number

$f_{\max}$ = largest positive floating point number

$\bar{x} = [x]$ = floating point representation of x

Theorem:

Let $x \in \mathbb{R}$ and $x \in [f_{\min}, f_{\max}]$

Then $\bar{x} = x(1 + \delta_1)$ where $|\delta_1| \leq$ **u**

Also, $\bar{x} = x/(1 + \delta_2)$ where $|\delta_2| \leq$ **u**

Note: δ_1 and δ_2 are functions of x

Roundoff Error

Notation: $[exp]$ denotes the evaluation of *exp* in floating point arithmetic.

Assuming a left-to-right evaluation, it holds

$$[x + y + z/w] = \Big[\big[[x] + [y] \big] + \big[[z]/[w] \big] \Big]$$

Roundoff Error

Notation: $[exp]$ denotes the evaluation of *exp* in floating point arithmetic.

Assuming a left-to-right evaluation, it holds

$$[x + y + z/w] = \Big[\big[[x] + [y]\big] + \big[[z]/[w]\big]\Big]$$

Floating Point Arithmetic

Theorem: (Standard and Alternative Computational Models)

Let x and y be floating point numbers

Then
$[x \text{ op } y] = (x \text{ op } y)(1 + \epsilon_1)$, where $|\epsilon_1| \leq \mathbf{u}$, and $\text{op} \in \{+, -, *, /\}$

Roundoff Error

Notation: $[exp]$ denotes the evaluation of *exp* in floating point arithmetic.

Assuming a left-to-right evaluation, it holds

$$[x + y + z/w] = \Big[[[x] + [y]] + [[z]/[w]] \Big]$$

Floating Point Arithmetic

Theorem: (Standard and Alternative Computational Models)

Let x and y be floating point numbers

Then
$[x \text{ op } y] = (x \text{ op } y)(1 + \epsilon_1)$, where $|\epsilon_1| \leq \mathbf{u}$, and $\text{op} \in \{+, -, *, /\}$

Also,
$[x \text{ op } y] = \dfrac{x \text{ op } y}{(1 + \epsilon_2)}$, where $|\epsilon_2| \leq \mathbf{u}$, and $\text{op} \in \{+, -, *, /\}$

Note: ϵ_1 and ϵ_2 are functions of x, y and op

Example: Dot Product

$x, y \in \mathbb{R}^n; \quad \kappa := x^T y$

$$\kappa := \Big(\big((\chi_0 \psi_0 + \chi_1 \psi_1) + \cdots \big) + \chi_{n-2}\psi_{n-2} \Big) + \chi_{n-1}\psi_{n-1}$$

$$\begin{aligned} \check{\kappa} &= \Bigg(\Big(\big(\chi_0 \psi_0 (1 + \epsilon_*^{(0)}) + \chi_1 \psi_1 (1 + \epsilon_*^{(1)}) \big) (1 + \epsilon_+^{(1)}) + \cdots \Big) (1 + \epsilon_+^{(n-2)}) \\ &\quad + \chi_{n-1}\psi_{n-1}(1 + \epsilon_*^{(n-1)}) \Bigg) (1 + \epsilon_+^{(n-1)}) \\ &= \sum_{i=0}^{n-1} \left(\chi_i \psi_i (1 + \epsilon_*^{(i)}) \prod_{j=i}^{n-1} (1 + \epsilon_+^{(j)}) \right) \end{aligned}$$

where $\epsilon_+^{(0)} = 0$ and $|\epsilon_*^{(0)}|, |\epsilon_*^{(j)}|, |\epsilon_+^{(j)}| \leq \mathbf{u}$ for $j = 1, \ldots, n-1$

Backward Stability

Let $f : \mathcal{D} \to \mathcal{R}$ be a map from the domain $\mathcal{D}$ to the range $\mathcal{R}$.
Let $\hat{f} : \mathcal{D} \to \mathcal{R}$ represent the execution in floating point arithmetic of a given algorithm $\mathcal{A}$ that computes f.

$\mathcal{A}$ is said to be **backward stable** if
for all $x \in \mathcal{D}$ there exists a perturbed input $\bar{x} \in \mathcal{D}$, close to x, such that $\hat{f}(x) = f(\bar{x})$.

Backward Stability

Let $f : \mathcal{D} \to \mathcal{R}$ be a map from the domain $\mathcal{D}$ to the range $\mathcal{R}$.
Let $\hat{f} : \mathcal{D} \to \mathcal{R}$ represent the execution in floating point arithmetic of a given algorithm $\mathcal{A}$ that computes f.

$\mathcal{A}$ is said to be **backward stable** if
for all $x \in \mathcal{D}$ there exists a perturbed input $\bar{x} \in \mathcal{D}$, close to x, such that $\hat{f}(x) = f(\bar{x})$.

I.e., the result computed in floating point arithmetic ($\hat{f}(x)$) equals the result obtained when the mathematically exact function (f) is applied to slightly perturbed data ($\bar{x}$).

The difference between $\bar{x}$ and x, is the perturbation to the original input x.

References

- IEEE 754-1985 and IEEE 754-2008:
  "`Standard for Floating-Point Arithmetic`"
- Book: "`Accuracy and Stability of Numerical Algorithms`", by Nick Higham
- Article: "`What every computer scientist should know about floating-point arithmetic`", by David Goldberg
- Book: "`Numerical Computing with IEEE Floating Point Arithmetic`", by Michael Overton