

Introduction to Scientific Computing Languages

Prof. **Paolo Bientinesi**

`pauldj@aices.rwth-aachen.de`



Deutsche
Forschungsgemeinschaft

DFG

Numbers

- $\frac{123}{29} =$ (first 40 digits)

4.241379310344827586206896551724137931034...

- $\pi =$

3.141592653589793238462643383279502884197...

- In general: **Infinite** number of digits

Numbers

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

Computers

- **Finite** memory

- **Infinite** numbers vs. **finite** memory
⇒ **Approximated numbers**

Computers: Inexact Numbers

- **Infinite** numbers vs. **finite** memory
⇒ **Approximated numbers**

How many digits? A pre-determined amount

$\pi = 3.141592653589793238462643383279502884197\dots$

Computers: Inexact Numbers

- **Infinite** numbers vs. **finite** memory
⇒ **Approximated numbers**

How many digits? A pre-determined amount

$\pi = 3.141592653589793238462643383279502884197\dots$

Using 4 digits: $\pi = 3.141$

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

Computers: Inexact Numbers

- **Infinite** numbers vs. **finite** memory
⇒ **Approximated numbers**

How many digits? A pre-determined amount

$\pi = 3.141592653589793238462643383279502884197\dots$

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

$$\begin{array}{r} 123.4 \quad + \\ \quad .5678 \quad = \\ \hline 123.9678 \quad = \end{array}$$

Computers: Approximated Computations

4-digit representation

Inexact Arithmetic

$$\begin{array}{r} 123.4 \quad + \\ \quad .5678 \quad = \\ \hline 123.9678 \quad = \end{array}$$

Computers: Approximated Computations

4-digit representation

Inexact Arithmetic

	123.4	+
	.5678	=
	<hr/>	
	123.9678	=
Truncated	123.9	
Rounded	124.0	

Computers: Approximated Computations

4-digit representation

Inexact Arithmetic

$$\begin{array}{r} 123.4 \quad + \\ \quad .5678 \quad = \\ \hline 123.9678 \quad = \\ \text{Truncated} \quad 123.9 \\ \text{Rounded} \quad 124.0 \end{array}$$

Associativity?

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

Computers: Approximated Computations

4-digit representation

Inexact Arithmetic

$$\begin{array}{r} 123.4 \quad + \\ \quad .5678 \quad = \\ \hline 123.9678 \quad = \\ \text{Truncated} \quad 123.9 \\ \text{Rounded} \quad 124.0 \end{array}$$

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

$$\begin{array}{r} 123.4 \quad + \\ \quad .5678 \quad = \\ \hline 123.9678 \quad = \\ \text{Truncated} \quad 123.9 \\ \text{Rounded} \quad 124.0 \end{array}$$

Associativity? No!

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

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

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

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

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

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

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

- Exact arithmetic:

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

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

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

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

- Exact arithmetic:

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

- Inexact arithmetic:

$$x \rightsquigarrow \hat{x}, \quad f \rightsquigarrow \hat{f} \quad \hat{f}(\hat{x}) \text{ instead of } f(x)$$

Representation Errors

Roundoff Errors

Algorithmic Errors

Known Disasters

- Patriot Missile, 1991
Scud launched from Iraq against US military base in South Arabia.
US Patriot's missile missed the incoming Scud.
28 casualties.

Cancellation

Known Disasters

- Patriot Missile, 1991
Scud launched from Iraq against 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

$$y = \pm d_0.d_1d_2 \dots d_{t-1} \times \beta^e$$

Floating Point Numbers

$$y = \pm d_0.d_1d_2 \dots d_{t-1} \times \beta^e$$

- β : base (radix)

Floating Point Numbers

$$y = \pm d_0.d_1d_2 \dots d_{t-1} \times \beta^e$$

- β : base (radix)
- t : precision = # slots for the mantissa ($d_0.d_1 \dots d_{t-1}$)

Floating Point Numbers

$$y = \pm d_0.d_1d_2 \dots d_{t-1} \times \beta^e$$

- β : base (radix)
- t : precision = # slots for the mantissa ($d_0.d_1 \dots d_{t-1}$)
- $0 \leq d_i \leq \beta - 1$: digits

Floating Point Numbers

$$y = \pm d_0.d_1d_2 \dots d_{t-1} \times \beta^e$$

- β : base (radix)
- t : precision = # slots for the mantissa ($d_0.d_1 \dots d_{t-1}$)
- $0 \leq d_i \leq \beta - 1$: digits
- $e_{\min} \leq e \leq e_{\max}$: exponent range

Floating Point Numbers

$$y = \pm d_0.d_1d_2 \dots d_{t-1} \times \beta^e$$

- β : base (radix)
- t : precision = # slots for the mantissa ($d_0.d_1 \dots d_{t-1}$)
- $0 \leq d_i \leq \beta - 1$: digits
- $e_{\min} \leq e \leq e_{\max}$: exponent range
- Normalization: $d_0 = 1$; d_0 used for the sign \pm

Floating Point Numbers

$$y = \pm d_0.d_1d_2 \dots d_{t-1} \times \beta^e$$

- β : base (radix)
- t : precision = # slots for the mantissa ($d_0.d_1 \dots d_{t-1}$)
- $0 \leq d_i \leq \beta - 1$: digits
- $e_{\min} \leq e \leq e_{\max}$: exponent range
- Normalization: $d_0 = 1$; d_0 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

$$y = \pm d_0.d_1d_2 \dots d_{t-1} \times \beta^e$$

- β : base (radix)
- t : precision = # slots for the mantissa ($d_0.d_1 \dots d_{t-1}$)
- $0 \leq d_i \leq \beta - 1$: digits
- $e_{\min} \leq e \leq e_{\max}$: exponent range
- Normalization: $d_0 = 1$; d_0 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)

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

Floating Point Numbers (2)

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

Floating Point Numbers (2)

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

Floating Point Numbers (2)

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

IEEE single precision – FYI

$$\boxed{\pm \quad a_1 a_2 a_3 \dots a_8 \quad b_1 b_2 b_3 \dots b_{23}}$$

If exponent bitstring $a_1 \dots a_8$ is	Then numerical value represented is
$(00000000)_2 = (0)_{10}$	$\pm(0.b_1 b_2 b_3 \dots b_{23})_2 \times 2^{-126}$
$(00000001)_2 = (1)_{10}$	$\pm(1.b_1 b_2 b_3 \dots b_{23})_2 \times 2^{-126}$
$(00000010)_2 = (2)_{10}$	$\pm(1.b_1 b_2 b_3 \dots b_{23})_2 \times 2^{-125}$
$(00000011)_2 = (3)_{10}$	$\pm(1.b_1 b_2 b_3 \dots b_{23})_2 \times 2^{-124}$
↓	↓
$(01111111)_2 = (127)_{10}$	$\pm(1.b_1 b_2 b_3 \dots b_{23})_2 \times 2^0$
$(10000000)_2 = (128)_{10}$	$\pm(1.b_1 b_2 b_3 \dots b_{23})_2 \times 2^1$
↓	↓
$(11111100)_2 = (252)_{10}$	$\pm(1.b_1 b_2 b_3 \dots b_{23})_2 \times 2^{125}$
$(11111101)_2 = (253)_{10}$	$\pm(1.b_1 b_2 b_3 \dots b_{23})_2 \times 2^{126}$
$(11111110)_2 = (254)_{10}$	$\pm(1.b_1 b_2 b_3 \dots b_{23})_2 \times 2^{127}$
$(11111111)_2 = (255)_{10}$	$\pm\infty$ if $b_1 = \dots = b_{23} = 0$, NaN otherwise

IEEE double precision – FYI

$$\pm \quad a_1 a_2 a_3 \dots a_{11} \quad b_1 b_2 b_3 \dots b_{52}$$

If exponent bitstring is $a_1 \dots a_{11}$	Then numerical value represented is
$(0000000000)_2 = (0)_{10}$	$\pm(0.b_1 b_2 b_3 \dots b_{52})_2 \times 2^{-1022}$
$(0000000001)_2 = (1)_{10}$	$\pm(1.b_1 b_2 b_3 \dots b_{52})_2 \times 2^{-1022}$
$(0000000010)_2 = (2)_{10}$	$\pm(1.b_1 b_2 b_3 \dots b_{52})_2 \times 2^{-1021}$
$(0000000011)_2 = (3)_{10}$	$\pm(1.b_1 b_2 b_3 \dots b_{52})_2 \times 2^{-1020}$
↓	↓
$(0111111111)_2 = (1023)_{10}$	$\pm(1.b_1 b_2 b_3 \dots b_{52})_2 \times 2^0$
$(1000000000)_2 = (1024)_{10}$	$\pm(1.b_1 b_2 b_3 \dots b_{52})_2 \times 2^1$
↓	↓
$(1111111100)_2 = (2044)_{10}$	$\pm(1.b_1 b_2 b_3 \dots b_{52})_2 \times 2^{1021}$
$(1111111101)_2 = (2045)_{10}$	$\pm(1.b_1 b_2 b_3 \dots b_{52})_2 \times 2^{1022}$
$(1111111110)_2 = (2046)_{10}$	$\pm(1.b_1 b_2 b_3 \dots b_{52})_2 \times 2^{1023}$
$(1111111111)_2 = (2047)_{10}$	$\pm\infty$ if $b_1 = \dots = b_{52} = 0$, NaN otherwise

Machine Precision u

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

Machine Precision u

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

Machine Precision \mathbf{u}

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

Machine Precision \mathbf{u}

- Smallest positive number such that $[1 + \mathbf{u}] \neq 1$
- Largest positive number such that $[1 + \mathbf{u}] = 1$
- $\frac{1}{2}\beta^{1-t}$
- Distance between 1 and the next floating point number
- “Machine epsilon”, ϵ_M , \mathbf{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 $|\delta_1| \leq \mathbf{u}$

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 $|\delta_1| \leq \mathbf{u}$

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

Note: δ_1 and δ_2 are functions of x

Smallest distance $\neq 0$

Smallest distance $\neq 0$

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

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

Smallest distance $\neq 0$

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 distance $\neq 0$

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

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

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

$$1) 0.0 \dots 01 \times 2^0 = 2^{1-t}$$

$$2) 0.0 \dots 01 \times 2^{e_{\min}} = 2^{e_{\min} - t + 1}$$

$$3) 0.0 \dots 01 \times 2^{e_{\min} + t - 1} = 2^{e_{\min}}$$

Smallest distance $\neq 0$

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

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

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

$$1) 0.0 \dots 01 \times 2^0 = 2^{1-t}$$

$$2) 0.0 \dots 01 \times 2^{e_{\min}} = 2^{e_{\min} - t + 1}$$

$$3) 0.0 \dots 01 \times 2^{e_{\min} + t - 1} = 2^{e_{\min}}$$

Smallest distance $\neq 0$

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) $0.0 \dots 01 \times 2^0 = 2^{1-t}$
- 2) $0.0 \dots 01 \times 2^{e_{\min}} = 2^{e_{\min} - t + 1}$
- 3) $0.0 \dots 01 \times 2^{e_{\min} + t - 1} = 2^{e_{\min}}$

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

- 1) $0.0 \dots 01 \times 2^0 = 2^{1-t}$
- 2) $0.0 \dots 01 \times 2^{e_{\min}} = 2^{e_{\min} - t + 1}$
- 3) $0.0 \dots 01 \times 2^{e_{\min} + t - 1} = 2^{e_{\min}}$

Smallest distance $\neq 0$

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) $0.0 \dots 01 \times 2^0 = 2^{1-t}$
- 2) $0.0 \dots 01 \times 2^{e_{\min}} = 2^{e_{\min} - t + 1}$
- 3) $0.0 \dots 01 \times 2^{e_{\min} + t - 1} = 2^{e_{\min}}$

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

- 1) $0.0 \dots 01 \times 2^0 = 2^{1-t}$
- 2) $0.0 \dots 01 \times 2^{e_{\min}} = 2^{e_{\min} - t + 1}$
- 3) $0.0 \dots 01 \times 2^{e_{\min} + t - 1} = 2^{e_{\min}}$

Roundoff Error

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

Assuming a left-to-right evaluation, it holds

$$[x + y + z/w] = \left[\left[[x] + [y] \right] + \left[[z]/[w] \right] \right]$$

Roundoff Error

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

Assuming a left-to-right evaluation, it holds

$$[x + y + z/w] = \left[\left[[x] + [y] \right] + \left[[z]/[w] \right] \right]$$

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

$$[x + y + z/w] = \left[\left[[x] + [y] \right] + \left[[z]/[w] \right] \right]$$

Floating Point Arithmetic

Theorem: (Standard and Alternative Computational Models)

Let x and y be floating point numbers

Then

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

Also,

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

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

Example: Dot Product

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

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

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

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

Bisection

- Recursive implementation
- Base case?

Bisection

- Recursive implementation
- Base case?

$\Pi^2/6$

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

Bisection

- Recursive implementation
- Base case?

$\Pi^2/6$

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

sqsqrt vs. sqrtsq

Bisection

- Recursive implementation
- Base case?

$\Pi^2/6$

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

sqsqrt vs. sqrtsq

$$Ax = b$$

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

Backward Stability

Let $f : \mathcal{D} \rightarrow \mathcal{R}$ be a map from the domain \mathcal{D} to the range \mathcal{R} .

Let $\hat{f} : \mathcal{D} \rightarrow \mathcal{R}$ 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 $f : \mathcal{D} \rightarrow \mathcal{R}$ be a map from the domain \mathcal{D} to the range \mathcal{R} .

Let $\hat{f} : \mathcal{D} \rightarrow \mathcal{R}$ 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 .

- 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