Introduction
Computer Arithmetic

732A90
Computational Statistics

Maryna Prus
(maryna.prus@liu.se)

Slides originally by Krzysztof Bartoszek

November 2, 2021
Department of Computer and Information Science
Linköping University

Teaching staff for course

Me: Maryna Prus

1. Course coordination
2. Lectures

Filip Ekström
Joel Oskarsson
Martynas Lukosevicius
Shashi Nagarajan
Yifan Ding

1. Labs
2. Marking of reports
3. Support

Lesson structure, examination

- Lesson structure
  - Lectures
  - Computer labs

    you work in groups
  - Seminars

    same groups as for computer labs
- Examination
  - Lab reports
  - Seminars

    presentation or opposition
  - Final exam

    computer based

  Allowed aids:

    printed books and own PDF document (max 100 pages)

  Exam points:

A: $[18, \infty)$, B: $[16, 18)$, C: $[14, 16)$, D: $[12, 14)$, E: $[10, 12)$, F: $[0, 10)$

Course materials, software

- Course materials
  - Lecture slides
  - Books
    - James E. Gentle "Computational Statistics", Springer, 2009
    - Geof H. Givens, Jennifer A. Hoeting "Computational Statistics", Wiley, 2013
    - ...
  - Googling...
- Software
  - R

Course contents

1. Computer Arithmetic (JG pages 85–105)
2. Optimization (JG pages 241–272, handouts)
3. Random Number Generation (JG pages 305–312, 325–328, handouts)
4. Monte Carlo Methods (JG pages 312–318, 328 417–429, handouts)
5. Numerical Model Selection and Hypothesis Testing (JG pages 52–56, 424, 435–467, handouts)
6. Expectation Maximization Algorithm and Stochastic Optimization (JG pages 275–284, 296–298, 480–483, handout)

Pages are recommended reading for each lecture,
bot not exact lecture content.

Computer Arithmetic: Example

$$\frac{x}{x+y} + \frac{y}{x+y} = 1 \quad \textbf{?}$$

```
x<-0.5^1000;
y<-0.4^1000;
x/(x+y)+y/(x+y)
1
x<-0.5^10000;
y<-0.4^10000;
x/(x+y)+y/(x+y)
NaN
x<-0.1^1000;
y<-0.2^1000;
x/(x+y)+y/(x+y)
NaN
```

⇒ *Computer arithmetic is not the same as "usual" arithmetic*

⇒ *Computations can be affected by magnitudes of numbers*

Computer storage

- Computers store information in binary form
  0 1 0 1 1 0 0 1
- 1Byte=8bits (typical counting unit)
- 1KB=1024bytes
- 1MB=1024KB
- and so on

Storage unit (or *word*) - basic grouping of bits in a computer

Typically, length of one *storage unit* - 32 or 64 bits

Fixed–point system (integers)

- We use the base 10 (decimal) system, e.g.

$$1234 = 1 \cdot 10^3 + 2 \cdot 10^2 + 3 \cdot 10^1 + 4 \cdot 10^0$$

- Computers use base 2 (binary) system
- Positive integers
  - a - positive integer. a can be represented as

$$a = a_0 \cdot 2^0 + a_1 \cdot 2^1 + a_2 \cdot 2^2 + \ldots,\ a_i \in \{0, 1\},\ \forall\, i$$

  - Code for a

$$a_k\ \ldots\ a_2\, a_1\, a_0$$

    k - number of bits per unit
  - Example: code for 5

$$0 \ldots 0\,1\,0\,1$$

Fixed–point system

- Negative integers - *twos-complement representation*
  - b - negative integer. Then $a = -b$ positive. Code for a

$$a_k \;\ldots\; a_2\, a_1\, a_0$$

  - Code for b: sign bit + opposite of a
  - Sign bit for b: $0\ldots 0\,0\,1$ (same for all negative integers)
  - Opposite of a

$$c_k \;\ldots\; c_2\, c_1\, c_0$$

where

$$c_i = \begin{cases} 1, & a_i = 0 \\ 0, & a_i = 1 \end{cases}, \forall\, i$$

  - Example: code for -5

$$1\ldots 1\,1\,0\,1\,1$$

  - Range (approximately): from -2^{k-1} to 2^{k-1}

Fixed–point system: arithmetic operations

- **Addition**, **multiplication**:
  as "usually" but for base 2 instead of base 10
- **Subtraction**: $a - b = a + (-b)$
  $-b$ is integer $\Rightarrow$ works well
- **Division**: $a/b = a \cdot 1/b$
  but $1/b$ not integer!

  $\Rightarrow$ more complicated, another approach needed for division

- **Other problems: Overflow**
  - k bits available
  - Operation results in a too large number
    $\Rightarrow$ more than k bits needed for code
  - Sign bit is missing or not correctly interpreted (dependent on computer architecture)
    $\Rightarrow$ adding two large (positive) numbers can result in a negative number (**!**)

$\Rightarrow$ Even addition and multiplication are not always "safe"

Floating–point system (rational, “real”)

- Parameters for encoding
  - Base: b, usually $b = 2$, sometimes 10 or even 16
  - Sign: $+/-$
  - Exponent: e, integer
  - Number of digits for *mantissa* (or *significand*): p
  - Mantissa or significand: $d_1, d_2, \ldots, d_p$, d_i integer, $0 \leq d_i < b$, $i = 1, \ldots, p$
- Representation of a real number a

$$a \approx \pm 0.d_1 d_2 \ldots d_p \cdot b^e$$

- For 64 bits, $b = 2$, $p = 52$

sign (1bit)	Exponent (11bits)	Mantissa (52 bits)

Range: $\approx [-10^{300}, 10^{300}] \approx [-b^{e_{max}}, b^{e_{max}}]$

Floating–point system

- Example: Base $b = 10$, $p = 5$ (mantissa has 5 digits):

$$1.2345 = +0.12345 \cdot 10^1$$

works well (p large enough)

$$4.0000567 = +0.40000 \cdot 10^1$$

Problem: p too small (also for other values of base b)

- For $b = 2$, $p = 52$

```
options(digits=22) #max possible
0.1
0.1000000000000000055511
```

Reason: Rounding towards the nearest computer float

Note that for $b = 10$ no problem: $0.1 = 0.1 \cdot 10^0$

Floating–point system: special “numbers”

- Special numbers
  - $\pm$Inf: exponent is $e_{\max} + 1$, mantissa is 0
  - NaN: exponent is $e_{\max} + 1$, mantissa is $\neq 0$
- Overflow: number larger than can be represented
- Underflow: number smaller than can be represented using k bits (close to 0)
  $\Rightarrow$ loss of significant digits
  $\Rightarrow$ rounding to 0
- Examples

```
10^200*10^200  =   Inf
10^400/10^400  =   NaN
10^(-200)/10^200  =   0
```

Floating–point system: arithmetic operations

Floats are rounded so usual mathematical laws do not hold — floating point arithmetic:

- $x + y$, $x \cdot y$ can display overflow, underflow
- $a \neq b$ but $x + a = b + x$
- $a + x = x$ but $a + y \neq y$
- $a + x = x$ but $x - x \neq a$
- $x = \sqrt{y}$ but $x \cdot x \neq y$
- ...

Example

```
options(digits=22)
x<-sqrt(2)
x*x
2.000000000000000444089
(x*x)==2
FALSE
```

Summation

Underflow problems can occur with any summation
Example:

```
options(digits=22)
x<-1:1000000; sum(1/x); sum(1/rev(x))
[1] 14.39272672286572252176
[1] 14.39272672286572429812
```

Solution A:

1. Sort the numbers in order of increasing magnitude
2. Sum in this order

Solution B: If numbers have roughly same magnitude

1. Sum numbers pairwise, from n obtain $n/2$ numbers
   Choose the pairs so that the resulting sums are also of roughly same magnitude
2. Continue until 1 number left

More on summing

Example: Computing exponent using Taylor series

$$e^x = 1 + x + x^2/2 + x^3/6 + \ldots = \sum_{i=0}^{\infty} \frac{x^i}{i!}$$

```
options(digits=22)
fTaylor<-function(x,N){1+sum(sapply(1:N,
   function(i,x){x^i/prod(1:i)},x=x,simplify=
   TRUE))}
exp(20) #fine
485165195.4097902774811
fTaylor(20,100)
485165195.4097902774811
fTaylor(20,100)-exp(20)
0
```

More on summing

Example: Computing exponent using Taylor series

$$e^x = 1 + x + x^2/2 + x^3/6 + \ldots = \sum_{i=0}^{\infty} \frac{x^i}{i!}$$

```
exp(-20)  #problem
2.0611536224385578699942e-09
fTaylor(-20,100)
-3.8538772173524193931 37e-10
fTaylor(-20,200)
-3.8538772173524193931 37e-10
```

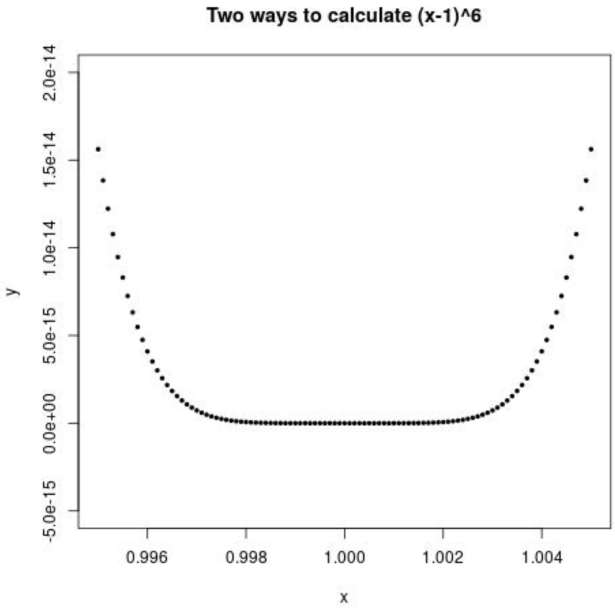
Reason: Varying sign of terms
⇒ adding two numbers of almost equal magnitude but of opposite sign because of rounding may result into smaller numbers than "really"
⇒ this effect is called *cancellation*

Can you explain why?

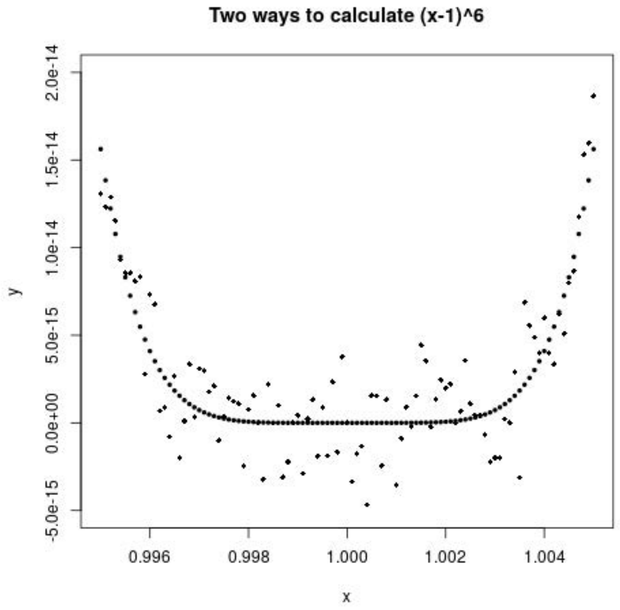
$$(x-1)^6 = 1 - 6x + 15x^2 - 20x^3 + 15x^4 - 6x^5 + x^6$$

```
## Example due to Thomas Ericsson in his
   Numerical Analysis course at Chalmers
f1<-function(x){(x-1)^6}
f2<-function(x){1-6*x+15*x^2-20*x^3+15*x^4-6*x
   ^5+x^6}

x<-seq(from=0.995,to=1.005,by=0.0001)
y1<-f1(x);y2<-f2(x)

plot(x,y1,pch=19,cex=0.5,ylim=c(-5*10^(-15),20
   *10^(-15)),main="Two ways to calculate (x
   -1)^6",xlab="x",ylab="y")
points(x,y2,pch=18,cex=0.8)
```

Can you explain why?



Can you explain why?



Thank you for attention!