Statistical and Mathematical Methods for Artificial Intelligence
a.a. 2024-25

Elena Loli Piccolomini

Problem Solving Process

- Develop a Mathematical model;
- Develop Algorithms for the numerical solution
- Implement Algorithms in computer software
- Run the software to simulate the physical process numerically
- Graphical visualization of the computer results
- Interpret and validate the computed results.

Sources of Approximation

- Measure errors. Due to the measure instrument.
- Arithmetic errors. Due to the propagation of the rounding errors of each single operation in an algorithmic process.
- Truncation errors. Due to the truncation of an infinite prcodure to a finite procedure (e.g. a seires is approximated with a finite sum).
- Inherent errors. Due to the finite representation of the data of a problem.

Absolute Error and Relative Error

Suppose that $\tilde{x}$ is an approximation of x,

- Absolute error : $E_x = abs(\tilde{x} - x)$
- Relative error:

$$R_x = \frac{abs(\tilde{x} - x)}{abs(x)}, \quad x \neq 0$$

Example: Compute E_x and R_x in the following cases:

$$x = 3.141592, \quad \tilde{x} = 3.14, \quad E_x = 0.001592, \ R_x = 0.000507$$

$$x = 1.e + 6, \quad \tilde{x} = 999996, \quad E_x = 4, \ R_x = 4 \cdot 10^{-6}$$

$$x = 1.2 \cdot 10^{-5}, \quad \tilde{x} = 0.9 \cdot 10^{-5}, \quad E_x = 0.3 \cdot 10^{-5}, \ R_x = 0.25$$

Truncation Error and Rounding Error

- Truncation error : difference between true result (for actual input) and result produced by given algorithm using exact arithmetic.
  - Due to approximations such as truncating infinite series or terminating iterative sequence before convergence.
- Rounding error: difference between result produced by given algorithm using exact arithmetic and result produced by same algorithm using limited precision arithmetic
  - Due to inexact representation of real numbers and arithmetic operations upon them

Computational error is sum of truncation error and rounding error, but one of these usually dominates

Representation of a real number in base β

Given an integer $\beta > 1$, a real number $x \neq 0$ can be expressed in a unique way as:

$$x = sign(x)(d_1\beta^{-1} + d_2\beta^{-2} + \ldots)\beta^p = sign(x)m\beta^p$$

where $sign(x) = 1$ if $x < 0$, $sign(x) = -1$ if $x < 0$, p is an integer and the digits $d_1, d_2, d_3, \ldots$ satisfy the following conditions:

- $0 \leq d_i \leq \beta - 1$
- $d_1 \neq 0$ and d_i are not all equal to $\beta - 1$ from a certain index i on.

Representation of a real number in base β

- m: **mantissa** ($\frac{1}{\beta} \leq m < 1$)
- β^p **exponential part**

Normalized scientific representation :

$$x = \pm(0.d_1 d_2 d_3 \ldots)\beta^p$$

Mixed representation:

$$x = \pm d_1 d_2 \ldots d_p . d_{p+1} d_{p+2} \ldots \quad p > 0$$

$$x = \pm 0.0 \ldots 0 d_1 d_2 \ldots \quad p > 0$$

Floating-Point Numbers

Formally a system of floating point numbers $\mathcal{F}(\beta, t, L, U)$ depends on the parameters:

- β: base
- t: precision
- $[L, U]$: exponent range

Any floating point number $x \in \mathcal{F}(\beta, t, L, U)$ has the form:

$$x = \pm \left(d_1 \beta^{-1} + \cdots + d_t \beta^{-t} \right)^p, \quad L \leq p \leq U$$

where d_i is an integer s.t.

$$0 \leq d_i \leq \beta - 1, \quad i = 1, \ldots, t$$

- $m = (d_1 \ldots d_t)$ is called *mantissa*
- p is the *exponent or characteristic*

Normalized Floating Point Numbers

A floating point system is *normalized* when $d_1 \neq 0$. In this case

$$\frac{1}{\beta} \leq d_1 < 1$$

Reasons for normalization:

- Representation of each number unique.
- No digits wasted on leading zeros.
- Leading bit need not be stored (in binary system).

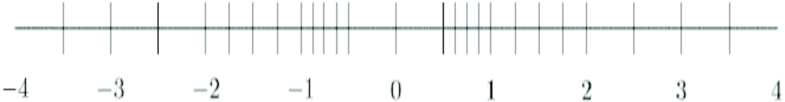
Properties of Floating Point Systems

- Floating-point number system is finite and discrete
- Total number of normalized floating-point numbers is:

$$2(\beta - 1)\beta^{t-1}(U - L + 1) + 1$$

- Smallest positive normalized number: $\text{UFL} = \beta^{L-1}$
- Largest floating-point number: $\text{OFL} = \beta^{U}(1 - \beta^{-t})$
- Floating-point numbers equally spaced only between successive powers of β
- Not all real numbers exactly representable; only *machine numbers* are exactly representable and are elements of $\mathcal{F}(\beta, t, L, U)$.

Example $\mathcal{F}(2, 3, -1, 1)$



- Number of elements: $2(2-1)2^2\,(1+1+1)+1=25$
- $\mathrm{UFL} = 2^{-2} = 0.25$
- $\mathrm{OFL} = 2^1(1-2^{-3}) = 1.75$

Rounding Rules

- *Rounding rules* refer to the approximation of real numbers x to the floating point numbers $fl(x) \in \mathcal{F}$.
- Two commonly used rounding rules
  - **chop** : truncate β base expansion of x after t-st digit; also called round toward zero.
  - **round to nearest** : $fl(x)$ is nearest floating-point number to x, using floating-point number whose last stored digit is even in case of tie; also called *round to even*;
- Round to nearest is most accurate, and is default rounding rule in IEEE systems.

Standard IEEE

Two types of floating point numbers:

- Single Precision 4 byte i.e. 32 bit.
  Word:

1 (sign)	23 (mantissa)	8 (exponent)

- Double Precision 8 byte i.e. 64 bit
  Word:

1 (sign)	52 (mantissa)	11 (exponent)

Due to normalized floating point representation the first bit is always equal to 1 and therefore it is *hidden*.

Machine Precision

- Accuracy of floating-point system characterized by unit roundoff (or machine precision or machine epsilon) denoted by ϵ_{mach}
  - With rounding by chopping, $\epsilon_{mach} = \beta^{1-t}$
  - With rounding to nearest, $\epsilon_{mach} = \frac{1}{2}\beta^{1-t}$
- Alternative definition is smallest number ϵ such that

$$fl(1+\epsilon) > 1$$

- Maximum relative error in representing real number x within range of floating-point system is given by

$$\boxed{\left|\frac{fl(x)-x}{x}\right| \leq \epsilon_{mach}}$$

Example

- Example System $\mathcal{F}(2, 3, -1, 1)$
  - Rounding by chopping:
  $$\epsilon_{mach} = 2^{1-3} = 2^{-2} \approx 0.25$$
  - Rounding to nearest:
  $$\epsilon_{mach} = \frac{1}{2}2^{1-3} = 2^{-3} \approx 0.125$$
- IEEE System
  - Single Precision machine epsilon:
  $$\epsilon_{mach} = \frac{1}{2}2^{1-23} = 2^{-23} \approx 1.1921e-07$$
  - Double Precision machine epsilon:
  $$\epsilon_{mach} = \frac{1}{2}2^{1-52} = 2^{-52} \approx 2.2204e-16$$

Exceptional Values

- IEEE floating-point standard provides special values to indicate two exceptional situations:
  - `Inf`, which stands for *infinity* results from dividing a finite number by zero, such as $1/0$,
  - `NaN`, which stands for *not a number* results from undefined or indeterminate operations such as $0/0$, $0 * \texttt{Inf}$ or $\texttt{Inf}/\texttt{Inf}$.
- `Inf` and `NaN` are implemented in IEEE arithmetic through special reserved values of exponent field.

Floating Point Arithmetic

- The computer canb execute operations only on finite numbers and the result of the operations must be itself a finite number.
- Hence the result of floating-point arithmetic operation may differ from result of corresponding real arithmetic operation on same operands
  - Addition or subtraction : Shifting of mantissa to make exponents match may cause loss of some digits of smaller number, possibly all of them.
  - Multiplication : Product of two t-digit mantissas contains up to $2t$ digits, so result may not be representable:
  - Division : Quotient of two t-digit mantissas may contain more than t digits, such as nonterminating binary expansion of $1/10$

Example: $\beta = 10$, $t = 6$

$x = 192.403, y = 0.635782, \quad fl(x) = 0.192403 \cdot 10^3, \ fl(y) = 0.635782 \cdot 10^0$

- $z = fl(x) + fl(y) = (0.192403 + 0.000635782) \cdot 10^3 = 0.193038782 \cdot 10^3$
  $fl(z) = 0.193039 \cdot 10^3$ The last two digits of y do not affect the result, and with even smaller exponent, y could have had no effect on the result.
- $w = fl(x) * fl(y) = (0.635782 \cdot 0.192403) \cdot 10^3 = 0.122326364146 \cdot 10^3$
  $fl(w) = 0.122326 \cdot 10^3$ which discards half of digits of true product.

Floating point arithmetic

Real result may also fail to be representable because its exponent p is beyond available range:

- $p > U$ Overflow
- $p < L$ Underflow

- Overflow is usually more serious than underflow because there is no good approximation to arbitrarily large magnitudes in floating-point system, whereas zero is often reasonable approximation for arbitrarily small magnitudes.
- On many computer systems overflow is fatal, but an underflow may be silently set to zero.

Floating point arithmetic (cont.)

- Arithmetic operation between real numbers $\cdot \ : \ \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$
- floating-point operation: $\odot : \mathbb{F} \times \mathbb{F} \rightarrow \mathbb{F}$

$$x \odot y = fl(x \cdot y)$$

- Each operation causes an error, called rounding error, which is very small:

$$\left| \frac{(x \odot y) - (x \cdot y)}{x \cdot y} \right| < eps$$

Floating point arithmetic (cont.)

Steps for the execution of a floating point arithmetic operation (the register has extended precision) $x \oplus y$

- Compute the exact results $z = x + y$
- Transform the result as floating point number $x \oplus y = fl(z)$

Example of floating point sums

$\beta = 10,\ t = 6$
$x = 192.403, y = 0.635782, \quad fl(x) = 0.192403 \cdot 10^3,\ fl(y) = 0.635782 \cdot 10^0$

- $z = fl(x) + fl(y) = (0.192403 + 0.000635782) \cdot 10^3 = 0.193038782 \cdot 10^3$
  $fl(z) = 0.193039 \cdot 10^3$ The last two digits of y do not affect the result, and with even smaller exponent, y could have had no effect on the result.
- $w = fl(x) * fl(y) = (0.635782 \cdot 0.192403) \cdot 10^3 = 0.122326364146 \cdot 10^3$
  $fl(w) = 0.122326 \cdot 10^3$ many digits of the exact result are lost

Example

- Ideally, $x \texttt{ flop } y = fl(x \texttt{ op } y)$, i.e., floating-point arithmetic operations produce correctly rounded results;
- Computers satisfying IEEE floating-point standard achieve this ideal as long as x op y is within range of floating-point system.
- Some familiar laws of real arithmetic are not necessarily valid in floating-point system.
- Floating-point addition and multiplication are commutative but not associative.
  Example:$\beta = 10$ e $t = 2$ $x = .11 \cdot 10^0$, $y = .13 \cdot 10^{-1}$, $z = .14 \cdot 10^{-1}$
  $(x + y) + z = 0.13 \cdot 10^0$ $x + (y + z) = 0.14 \cdot 10^0$

Cancellation

- Subtraction between two t-digit numbers having same sign and similar magnitudes yields result with fewer than t digits, so it is usually exactly representable.
- Reason is that leading digits of two numbers cancel (i.e., their difference is zero).
- For example $x = 1.92403 \cdot 10^2$, $y = 1.92275 \cdot 10^2$

$$fl(x) = 0.1.92403 \cdot 10^3, \quad fl(y) = 0.192275 \cdot 10^3$$

$$z = fl(x - y) = (0.1.92403 - 0.192275) \cdot 10^3 = 0.000128 \cdot 10^3$$

$$fl(z) = 0.128000 \cdot 10^3$$

  which is correct, and exactly representable, but has only three significant digits.
- Despite exactness of result, cancellation often implies serious loss of information.