CPNM Lecture 9 - Approximations and Errors Associated with Numerical Methods

Mridul Sankar Barik

Jadavpur University

2022

Introduction

- Computers use binary arithmetic; each number is represented as a binary number
- Some numbers can be represented exactly
  - Example: $2.125 = 2^1 + 2^{-3}$
- Numbers that cannot be represented exactly are approximated
  - Example: $3.1 \approx 2^1 + 2^0 + 2^{-4} + 2^{-5} + 2^{-8} + \ldots$
  - Numbers like π does not have any finite representation in either decimal or binary number system

Arithmetic Operations

- Two types of arithmetic
  - Integer (without fractional part)
  - Real or Floating Point (with fractional part)
- Word size: Number of bits processed by a processor at one go (using a single instruction); usually 32/64 bit
- All operands in arithmetic operations have finite number of digits (bits)

Fixed Point Representation I

- Example: 32 bits - divided into 2 parts, one part to represent an integer part of the number and the other the fractional part
- | One Sign bit | 23 bits integral part | 8 bits fraction part |
- Largest: $11\ldots1.11111111 = 1677215.998046875$
- Smallest: $00\ldots0.00000001 = 0.00390625$

Floating Point Representation

- On a computer, real numbers are represented in the floating-point form.
- **Floating-Point Form**: Let x be a non-zero real number. An n-digit floating-point number in base β has the form

$$fl(x) = (-1)^s \times (.d_1 d_2 \ldots d_n)_\beta \times \beta^e$$

where,

$$(.d_1 d_2 \ldots d_n)_\beta = \frac{d_1}{\beta} + \frac{d_2}{\beta^2} + \ldots + \frac{d_n}{\beta^n}$$

is a β-fraction called the **mantissa** or **significand**, $s = 1$ or 0 is called the sign and e is an integer called the **exponent**. The number β is also called the **radix** and the point preceding d_1 in is called the **radix point**.

Normalization

- A floating-point number is said to be normalized if either $d_1 \neq 0$ or $d_1 = d_2 = \ldots = d_n = 0$.
- **Example**:
  - The real number $x = 6.238$ can be represented as $6.238 = (-1)^0 \times 0.6238 \times 10^1$, here $s = 0$, $\beta = 10$, $e = 1$, $d_1 = 6$, $d_2 = 2$, $d_3 = 3$ and $d_4 = 8$.

Overflow and Underflow

- The exponent e is limited to a range $m < e < M$
- During the calculation, if some computed number has an exponent $e > M$ then we say, the memory overflow or if $e < m$, we say the memory underflow.

IEEE Standard - Single Precision

- The IEEE (Institute of Electrical and Electronics Engineers) standard for floating-point arithmetic (IEEE 754)
  - Introduced in 1985, augmented in 2008
- Floating-point representation for a binary number x is given by

$$fl(x) = (-1)^s \times (1.a_1 a_2 \ldots a_n)_2 \times 2^e$$

  where $a_1, a_2, \ldots a_n$ are either 1 or 0.
- The IEEE **single precision** floating-point format uses 4 bytes (32 bits) to store a number.
  $| \, (sign) b_1 \, | \, (exponent) b_2 b_3 \ldots b_9 \, | \, (mantissa) b_{10} b_{11} \ldots b_{32} \, |$
  - It has a precision of 24 binary digits
  - Exponent e is limited by $-126 \leq e \leq 127$
- Note here that there are only 23 bits used for mantissa. This is because, the digit 1 before the binary point is not stored in the memory and will be inserted at the time of calculation.
- Instead of the exponent e, we store the non-negative integer $E = (b_2 b_3 \ldots b_9)_2$ and define $e = E - 127$; E is called the biased exponent and $0 \leq E \leq 255$

Example I

- Example
  $2.5 \xrightarrow{\textit{binary}} 10.1 \xrightarrow{\textit{normalized}} 1.01 \times 2^1$
  $e = 1$
  $\therefore E = 128 \text{ as } e = E - 127$
  $0 \mid 10000000 \mid 01000000000000000000000$

- Example
  $3.75 \xrightarrow{\textit{binary}} 11.11 \xrightarrow{\textit{normalized}} 1.111 \times 2^1$
  $e = 1$
  $\therefore E = 128 \text{ as } e = E - 127$
  $0 \mid 10000000 \mid 11100000000000000000000$

Example II

- Example
  $10.125 \xrightarrow{binary} 1010.001 \xrightarrow{normalized} 1.010001 \times 2^3$
  $e = 3$
  $\therefore E = 130$ as $e = E - 127$
  $0 \mid 10000010 \mid 01000100000000000000000$

- Example
  $52.21875 \xrightarrow{binary} 110100.00111 \xrightarrow{normalized} 1.1010000111 \times 2^5$
  $e = 5$
  $\therefore E = 132$ as $e = E - 127$
  $0 \mid 10000100 \mid 10100001110000000000000$

Representation of Zero and Infinity

- Representation of Zero
  - sign = 0 or 1
  - biased exponent = all 0's
  - mantissa = all 0's
  - $+0 = 0 \mid 00000000 \mid 00000000000000000000000$
  - -0 = $1 \mid 00000000 \mid 00000000000000000000000$
- Representation of infinity
  - sign = 0 or 1
  - biased exponent = all 1's
  - mantissa = all 0's
  - $+\infty = 0 \mid 11111111 \mid 00000000000000000000000$
  - $-\infty = 1 \mid 11111111 \mid 00000000000000000000000$

Invalid Numbers I

- Representation of non numbers: arises when result of an arithmetic operation is not mathematically valid [also called **NaN - Not a Number**]
  - Quiet NaN: (ex. 0/0, $\sqrt{-1}$; normally carried over in the computation)
    - sign = 0 or 1
    - biased exponent = all 1's
    - mantissa = a 0 as the left-most bit and at least one 1 in the rest
    - 0/1 | 11111111 | 00010000000000000000000
  - Signaling NaN: (underflow/overflow; used to give an error message)
    - sign = 0 or 1
    - biased exponent = all 1's
    - mantissa = a 1 as the left-most bit and any combination in the rest
    - 0/1 | 11111111 | 10010000000000000000000

Largest and Smallest Numbers

- Largest Positive Number
  - 0 | 11111110 | 11111111111111111111111
  - mantissa $= 1.11111111111111111111111 = 1 + (1 - 2^{-23}) = 2 - 2^{-23}$
  - exponent $= 254 - 127 = 127$
  - Largest number $= (2 - 2^{-23}) \times 2^{127} \approx 3.403 \times 10^{38}$
- Smallest Positive Number
  - 0 | 00000001 | 00000000000000000000000
  - mantissa $= 1.00000000000000000000000$
  - exponent $= 1 - 127 = -126$
  - Smallest normalized number $= 2^{-126} \approx 1.17549435 \times 10^{-38}$

Double Precision Floating Point Numbers

- IEEE **double precision** floating point format uses 8 bytes (64 bits) to store a number
  - It has a precision of 53 binary digits
  - Exponent e is limited by $-1022 \leq e \leq 1023$

Chopping and Rounding Error I

- Any real number x can be represented exactly as (infinite storage)

$$x = (-1)^s \times (.d_1 d_2 \ldots d_n d_{n+1} \ldots)_\beta \times \beta^e$$

- With finite storage a real number x is approximated by $fl(x)$.
- There are two ways to produce $fl(x)$ from x as defined below.
- The chopped machine approximation of x is given by

$$fl(x) = (-1)^s \times (.d_1 d_2 \ldots d_n)_\beta \times \beta^e$$

- The rounded machine approximation of x is given by

$$fl(x) = \begin{cases} (-1)^s \times (.d_1 d_2 \ldots d_n)_\beta \times \beta^e, 0 \leq d_{n+1} < \frac{\beta}{2} \\ (-1)^s \times (.d_1 d_2 \ldots (d_n + 1))_\beta \times \beta^e, \frac{\beta}{2} \leq d_{n+1} < \beta \end{cases}$$

Different Types of Errors

- The approximate representation (x_a) of a real number differs from the actual/true number (x_t), whose difference is called an **error** (ε)

$$\varepsilon = x_t - x_a$$

- The **absolute error** (ε_a) is the absolute value of the error

$$\varepsilon_a = | x_t - x_a | = | \varepsilon |$$

- The **relative error** (ε_r) is a measure of the error in relation to the true value

$$\varepsilon_r = \left| \frac{\varepsilon}{x_t} \right|$$

- The **percentage error** is defined as 100 times the relative error

$$\varepsilon_p = \varepsilon_r * 100$$

Different Types of Errors - Example

- Suppose that you have the task of measuring the lengths of a bridge and a rivet and come up with 9999 and 9 cm, respectively. If the true values are 10,000 and 10 cm, respectively, compute (a) the true error and (b) the true percent relative error for each case.
- Solution:
  - The error for measuring the bridge is (10,000 - 9999) cm $=$ 1 cm, and for the rivet is (10 - 9) cm $=$ 1 cm
  - The percent relative error for the bridge is $\frac{1}{10{,}000} \times 100\% =$ 0.01%, and for the rivet it is $\frac{1}{10} \times 100\% = 10\%$
  - Thus, although both measurements have an error of 1 cm, the relative error for the rivet is much greater.

Approximate Errors

- In numerical methods true value is not known a priori

$$RelativeError = \frac{ApproximateError}{ApproximateValue}$$

- For numerical methods that use iterative approach

$$RelativeError = \frac{CurrentApproximation - PreviousApproximation}{CurrentApproximation}$$

Significant Digits I

- Significant Digits of a number are those that can be used with confidence
  - Example: Suppose we seek a numerical solution to have an accuracy of 10^{-3} and obtain as solution $y = 23.40657231$. Here the solution is reliable only up to the first three decimal places i.e $y = 23.406$ or the solution has five significant digits 23406
- Some thumb rules on the significant digits
  - All non-zero digits are significant
  - All zeros occurring between non-zero digits are significant
  - Trailing zeros following a decimal point are significant (Ex.: 4.50, 65.0, 0.230 have three significant digits)
  - Zeros between the decimal point and preceding a non-zero digit are not significant (Ex.: $0.0002341 = 2341 \times 10^{-7}$, $0.002341 = 2341 \times 10^{-6}$, $0.02341 = 2341 \times 10^{-5}$, have four significant digits)
  - Trailing zeros in large numbers without the decimal point are not significant (Ex.: $54000 = 54 \times 10^3$ has only two signifiant digits)
- Formally stated: If x_A is an approximation to x, then we say that x_A approximates x to r significant β-digits if

$$| x - x_A | \leq \frac{1}{2} \beta^{s-r+1}$$

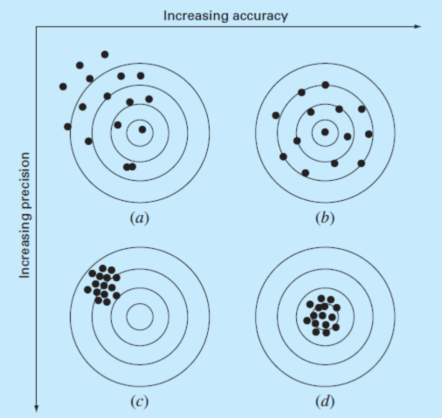
with s the largest integer such that $\beta^s \leq | x |$.

Significant Digits II

- Example: $x = \frac{1}{3}$ and $x_A = 0.333$
  - $\mid x - x_A \mid = \mid \frac{1}{3} - 0.333 \mid = \mid 0.333333... - 0.333 \mid \approx 0.00033 < 0.0005 = 0.5 \times 10^{-3} \Rightarrow s - r + 1 = -3$
  - $\mid x \mid = 0.33333 \geq 10^{-1} \Rightarrow s = -1$
  - So, $r = 3 \Rightarrow x_A$ is correct upto three significant digits

- Example: $x = 0.02138$ and $x_A = 0.02144$
  - $\mid x - x_A \mid \approx 0.00006 < 0.0005 = 0.5 \times 10^{-3} \Rightarrow s - r + 1 = -3$
  - $\mid x \mid = 0.02138 \geq 0.01 = 10^{-2} \Rightarrow s = -2$
  - So, $r = 2 \Rightarrow x_A$ is correct upto two significant digits

Accuracy and Precision



- **Accuracy** refers to how closely a computed or measured value agrees with the true value
- **Precision** refers to how closely individual computed or measured values agree with each other