Lecture 4 (10/2/2019)

We have discussed how to estimate error of Taylor approximation.

$$\underbrace{|f(x)}_{\text{exact}} - \underbrace{p_n(x)|}_{\text{appr.}} = \underbrace{|R_n(x)|}_{\text{error}} \overset{\text{Lagrange thm}}{=\!=} \underbrace{\frac{|f^{(n+1)}(c_x)|}{(n+1)!} |x-x_0|^{n+1}}_{}$$

bound this term from above by a quantity independent of x (possibly depending on n). n is used to control the size of error. We typically increase n to get smaller error.

This is the error coming from our approximation method. We typically can control this type of error by mathematical analysis (using Lagrange theorem for example).

Another type of error comes from using digital machine to do calculations (also called machine representation error). Note that a computer can store only finitely many digits. Number π or e or even $1/3 = 0.333\dots$ are not stored exactly in the memory.

Also, it is more convenient for computer to work well binary numbers than decimal numbers.

* How to convert a decimal number to a binary number:

$169 = (?)_2$

	169	
divide by 2	84	1
divide by 2	42	0
divide by 2	21	0
divide by 2	10	1
divide by 2	5	0
divide by 2	2	1
divide by 2	1	0
divide by 2	0	1

$\uparrow$ read from bottom to top

stop when we reach 0

$169 = (10101001)_2$

* How about fractions?

$0.3 = (?)_2$

what is $(0.1010)_2$ in decimal base?

$(0.1010)_2 = 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} + 0 \cdot 2^{-4} = \frac{1}{2} + \frac{1}{8} = 0.625$

- Algorithm :

$0.3 \times 2 = 0.6 \longrightarrow 0$ (the whole number of 0.6)

$0.6 \times 2 = 1.2 \longrightarrow 1$ (" " " " 1.2)

$\underbrace{0.2}_{}\times 2 = 0.4 \longrightarrow 0$ (" " 0.4)

the fractional part of 1.2

$0.4 \times 2 = 0.8 \longrightarrow 0$ (" " " 0.8)

$0.8 \times 2 = 1.6 \longrightarrow 1$ (" " " 1.6)

$\underbrace{0.6}_{}\times 2 = 1.2 \longrightarrow 1$ (" " " 1.2)

fractional part of 1.6

...... (from here the digits are periodic)

$$0.3 = (0.01\underbrace{0011}\ \underbrace{10011}\ \underbrace{10011}\ \ldots)_2$$

Suppose a number is stored in a computer by 64 bits.

$$\boxed{a_1}\ \boxed{a_2}\ \cdots\cdots\ \boxed{a_{64}}$$

where each a_i is either 0 or 1.

- If the combination $a_1, a_2 \cdots, a_{64}$ represents the binary number $(a_1 a_2 \ldots a_{64})_2$

  then what is the maximum number that can be represented?

  $$2^{64} - 1 \quad (\text{when } a_1 = \cdots = a_{64} = 1)$$

  What is the minimum number?

  $$0 \quad (\text{when } a_1 = \cdots = a_{64} = 1)$$

Advantage: this format can represent any integer from 0 to $2^{64}-1 \approx 10^{19}$.

Disavantage: narrow range of number $[0, 2^{64}-1]$. This range is not sufficient to include big physical constants such as

Avogadro's constant .... 6.022×10^{23}

or small constants such as

Gravitational constant ..... 6.674×10^{-11}

Electron mass ..... 9.109×10^{-31}

- IEEE 754-1985 standard (also known as double-precision floating-point format).

  This is a format standardized by IEEE in 1985, commonly used in computer software (Matlab, Maple, ...)

  $$x = \sigma \cdot \bar{x} \cdot 2^e$$

  σ is the sign (± 1)

  $\bar{x}$ is called significand or mantissa

  e is called exponent.

In the bit sequence:

$$\underbrace{c_0}_{\text{Sign}} \quad \underbrace{c_1 \cdots c_{11}}_{E} \quad \underbrace{a_1 \cdots a_{52}}_{\bar{x}} \qquad (*)$$

$c_0 = 0$ if the number is ≥ 0, $c_0 = 1$ if the number is < 0.

$\bar{x}$ is called significand or mantissa.

The format $(*)$ represents the following number:

- $\sigma (1.a_1 a_2 \cdots a_{52})_2 \, 2^{E-1023}$ if $1 \leq E \leq 2046$
- $\sigma (0.a_1 a_2 \cdots a_{52}) \, 2^{-1022}$ if $E = 0$
- $\pm\infty$ if $E = 2047$ and $a_1 = \cdots = a_{52} = 0$

  $\uparrow$ sign determined by σ
- NaN if $E = 2047$ and at least one of $a_1, a_2, \ldots, a_{52}$ is equal to 1

(see Table 2.4 on page 37)