Significant Figure:

Lecture 2 - Recalls

- Represent the reliability of a numerical value.
- The significant digits of a number are those that can be used with confidance.
- They correspond to the number of certain digits plus one estimated digit.
- Computers retain only a finite number of significant figures, that is why such numbers can never be represented exactly. The omission of the remaining significant figures is called round-off error.

3.2 Accuracy and precision:

Accuracy → how closely a computed or measured value agrees with the true value.

precision → how closely individual computed or measured values agree with each other.

Inaccuracy → also called "bias" → systematic deviation from truth.

Imprecision → " " "uncertainity" → magnitude of scatter.

Numerical methods should be sufficiently accurate or unbiased to meet requirements of a particular engineering problem.

3.3 Error Definition.

Sources of errors → Round off (number representation) & Trunction → Approximation of mathematical procedur

- Numerical errors arise from the use of approximations to represent exact mathematical operations & quantities.

Truncation error: result when approximations are used to represent exact mathematical procedures

Round_off error: result when numbers having limited significant figures are used to represent exact number.

True value = approximation + error $\qquad E_t$ = true value − approximation.

True functional relative error $= \dfrac{\text{True error}}{\text{True value}}$ $\qquad \varepsilon_t = \begin{pmatrix}\text{True percent}\\ \text{relative error}\end{pmatrix} = \dfrac{\text{true error}}{\text{true value}}$ 100

Ex. Length of bridge = 9999 cm
" " rivet = 9 cm

a) true error = ?
b) True percent relative error = ?

True value of bridge = 10,000
" " " rivet = 10 cm.

$E_t = 10{,}000 - 9999 = 1$ cm (bridge)
$E_t = 10 - 9 = 1$ cm (rivet)

true error for both are the Same.

The percent relative error

t signifies that error is normalized to the true value

for bridge => $\varepsilon_t = \frac{1}{10{,}000} \times 100\% = 0.01\%$

" rivet => $\varepsilon_t = \frac{1}{10} \times 100\% = 10\%$

Although true errors are equal, relative error for rivet is much greater.

Normalizing Error using the best available estimate of the true value, that is to the approximation itself:

$$\varepsilon_a = \frac{\text{approximate error}}{\text{approximation}} \times 100\%$$

a signifies that error is normalized to approximate value.

Challenge: Determining error estimates in the absence of knowledge regarding true value.

Iterative approach → to compute answers → present approximation is made on the basis of previous approximation. → Repeat process to compute better & better approximations:

$$\text{percent Relative Error} = \frac{\text{current approximation} - \text{previous approximation}}{\text{current approximation}} \times 100$$

Not interested often in sign of error → interested whether absolute value is lower than a specified percent tolerance ε_s $|\varepsilon_a| < \varepsilon_s$ (Stopping error criterion)

If this relationship holds, our result is assumed to be within the prespecified acceptable level ε_s. Relating errors to number of significant figure in approximation.

If $\varepsilon_s = (0.5 \times 10^{2-n})\%$ => we can be assured that the result is correct at least "n" significant figures.

Scarborough 1966

Example (lecture 2)

Functions can be represented in math by infinite series, e.g.

$$\boxed{\boxed{e^x = 1} + x} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots + \frac{x^n}{n!}$$ (Mac Laurin Series expansion)

previous approximation (considering one term only)

Start with $e^x = 1$, add terms one at a time to estimate $e^{0.5}$.

After each new term is added, compute "true" & "approximate" percent relative errors.

$$\begin{cases} \varepsilon_t = \dfrac{\text{True error}}{\text{true value}} \times 100\% \\ \varepsilon_a = \dfrac{\text{Current approximation} - \text{previous approx.}}{\text{current approx.}} \times 100\% \end{cases}$$

Solution → First determin the error criterion that ensure a result is correct to at least 3 significant figures.

Stopping Error Criterion ← $\varepsilon_s = (0.5 \times 10^{2-n})\% = (0.5 \times 10^{2-3})\% = 0.05\%$

Thus, we will add terms to the series until ε_a falls below this level.

$e^x = 1 + x$ or for $x = 0.5 \rightarrow e^{0.5} = 1 + 0.5 = 1.5$

True value, approximation

$$\varepsilon_t = \frac{e^{0.5} - 1.5}{e^{0.5}} = \frac{1.648721 - 1.5}{1.648721} \times 100\% = 9.02\%$$ true percent relative error

$$\varepsilon_a = \frac{1.5 - 1}{1.5} \times 100\% = 33.3\%$$ ← approximate percent relative error

We see that ε_a is not less than required value of ε_s ($|\varepsilon_a| \not< \varepsilon_s$)

We need to continue computation by adding another term $\frac{x^2}{2!}$ & repeat the error calculation. until $\varepsilon_a < \varepsilon_s$ → we can calculate & see that after 6 terms are included, approximation falls below $\varepsilon_s = 0.05\%$ & reach to 0.0158% & computation is terminated. (page 62)

* Many numerical methods involve iterative calculations ~~like~~ this. This entails solving a mathemical problem by computing successive approximation to the solution starting from an initial guess.

(5)

3.4. Round_off Errors → Error created due to approximate representation of numbers.

numbers such as π, e, $\sqrt{7}$ cannot be expressed by a fixed number of significant figures → cannot be represented by computers exactly.

* computers use a base-2 representation ⟹ they cannot precisely repeat certain exact base-10 numbers. → Discrepancy introduced by this omission of significant figures is called round-off error.

exp. → $\frac{1}{3} \simeq 0.333333$

Round off-error $= \frac{1}{3} - 0.333333 = 0.000000333\ldots$

3.4.1. Computer Representation of numbers:

Number system, word, positional notation,

"word" → is fundamental unit whereby information is represented., this consists of a string of "binary digits" or "bits" → Numbers are usually stored in one or more "words"

Number system: Decimal → Base-10 → uses 10-digits 0,1,2,3,4,5,6,7,8,9

86,409 → 8 groups of 10,000 +
6 " " 1000 +
4 " " 100 +
0 " " 10 +
9 units → result in the number → 86,409

Base → The number used as the reference for constructing the system. base-10, or 2 (10 digits) Visually is represented:

Positional Notation.

10^4	10^3	10^2	10	1
8	6	4	0	9

9 x 1 = 9 +
0 x 10 = 0 +
4 x 100 = 400 +
6 x 1000 = 6000 +
8 x 10,000 = 80,000 +

86,409

How the decimal (base-10) system work ↑

This representation is called "positional notation"

We are 10-fingered humans → use decimal or base-10 system

if it was 8 → were using octal or base-8 system (0,1,2,3,4,5,6,7)

Computers are 2-fingered animals → using binary or base-2 system. (0, 1)

↳ Relates to primary logic units of digital computers are on/off electronic components.

$11 = (1 \times 2^1) + (1 \times 2^0) = 2 + 1 = 3$ in decimal system.

↓
in binary

Example: what is the binary number 10101101 is equivant to in decimal?

positional Notation

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	0	1	0	1	1	0	1

$1 \times 1 = 1$ +
$0 \times 2 = 0$ +
$1 \times 4 = 4$ +
$1 \times 8 = 8$ +
$0 \times 16 = 0$ +
$1 \times 32 = 32$ +
$0 \times 64 = 0$ +
$1 \times 128 = 128$ +

173

How integers are represented on a computer?

Using "Signed Magnitude Method" → method employs the first bit of a word to indicate the sign
- 1 → − (negative)
- 0 → + (positive)

The remaining bits are used to store the number. Ex. −173 in 16-bit computer.

	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	0	0	0	0	0	0	0	1	0	1	0	1	1	0	1

For sign

16-bit

Range of Integers in base-10 that can be represented on a 16-bit computer?

First bit hold the sign → Remaining 15 bits → hold from 0 to $\overbrace{111111111111111}^{15}$

The upper limit is:

$$(1\times2^{14})+(1\times2^{13})+(1\times2^{12})+(1\times2^{11})+(1\times2^{10})+\cdots+(1\times2^{1})+(1\times2^{0})=32,767$$

This expression can simply be evaluated as $2^{15}-1$.

Then 16-bit computer word can store decimal integers ranging from:

$-32,767$ to $32,767$.

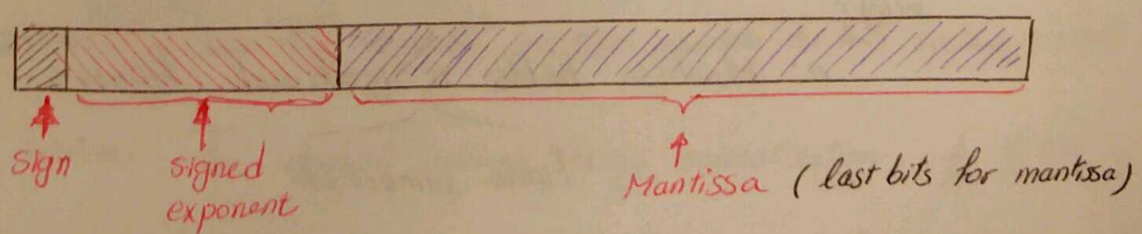
Floating-Point Representation

We learned about computer representation of integers; how about the fractional quantities → we use floating-point representation.

Divide it into two sections → fractional part → mantissa or Significand
→ Integer part → exponent or characteristic

in $\{m \cdot b^{e}\}$ →
- m → the mantissa
- b → the base of the number system being used.
- e → the exponent

Example: The number 156.78 could be represented as 0.15678×10^{3} in a floating-point base-10 system. The manner in which a floating-point number is stored in a word: (one of the ways)



Normalization of Mantissa: to remove leading zero digits.

Example: suppose $\frac{1}{34} = 0.029411765....$ was stored in a floating-point base-10 system that allowed only 4 decimal places to be stored:

4 decimal places

0.0294×10^{0}

inclusion of useless zero to the right of decimal

The number can be normalized by multiplying mantissa by 10 & lowering the exponent by 1 : 0.2941×10^{-1} → lowered the exponent by 1

Removed leading zero digit

Consequence of Normalization → absolute value of "m" is limited.

in mb^{e} → $\frac{1}{b} \leq m < 1$ b: base of the number system

m: mantissa value

Example: for base-10 system → $\frac{1}{10} \leq m < 1$ → $0.1 \leq m < 1$

" base-2 " → $\frac{1}{2} \leq m < 1$ → $0.5 \leq m < 1$

Question: what do you think, are the disadvantages of floating-point representation?

Advantage → allows both fractions & very large numbers to be expressed by comp

Dis " → 1) Make up more room.

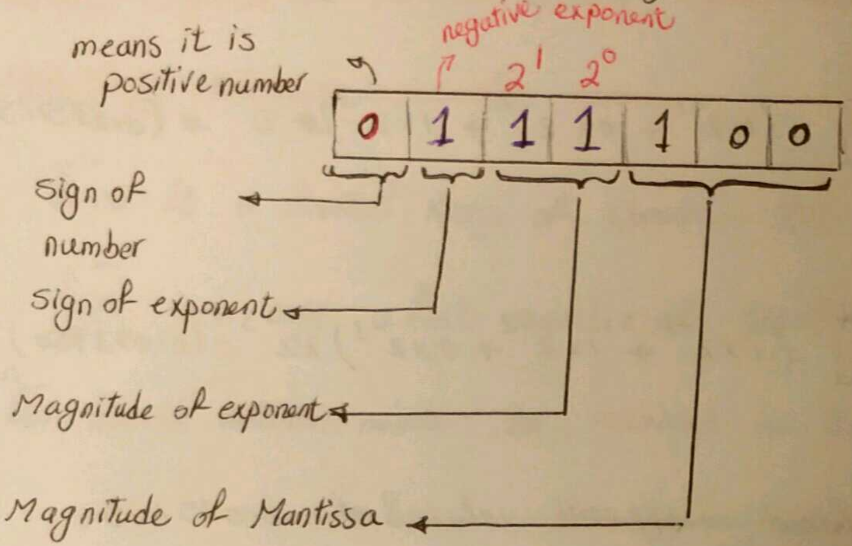
2) Take longer to process than integer numbers

3) Their use introduces a source of error → what type of error? why?

↓ Round-off

A: Mantissa holds $\underbrace{\text{a significant}}_{\text{finite number of}}$ figures

Example: 7-bit words → storing a floating point



First → Define if number is + or − $\left\{ \begin{array}{l} \to 1 \; - \\ \to 0 \; + \end{array} \right.$ → it is +

Second → " " exponent is + or − → it is −

Third → " the value of exponent $1 \times 2^1 + 1 \times 2^0 = 2 + 1 = 3$

Then exponent is −3

Then we are expecting to have $2^{-1}, 2^{-2}, 2^{-3}$ when calculing magnitude of mantissa.

Fourth → Calculate value of magnitude of mantissa:

$$(1 \times 2^{-1}) + (0 \times 2^{-2}) + (0 \times 2^{-3}) = 0.5 + 0 + 0 = 0.5$$

This will be the smallest possible positive number in 7-bit system.

* Note: Smaller mantissa possible (000, 001, 011) but 100 used due to limit imposed by normalization. ? → The smallest positive number for this system is $+0.5 \times 2^{-3} = 0.0625$ in base-10 system. The smallest value is set by normalization, $\frac{1}{b} \leq m$

The next highest numbers are developed by increasing the mantissa:

0	1	1	1	1	0	1

$(1\times2^{-1}+0\times2^{-2}+1\times2^{-3}) * 2^{-3}=(0.078125)_{10}$

0	1	1	1	1	1	0

$(1\times2^{-1}+1\times2^{-2}+0\times2^{-3})\times2^{-3}=(0.093750)_{10}$

⋮

Base 10 equivalents will be spaced evenly with intervals of 0.015625.

To continue increasing, we must ~~i~~decrease the exponent to 10 (one-zero)

$\rightarrow \quad 1\times2^{1}+0\times2^{0}=2$

The mantissa decrease to smallest value of 100

Then : next number 0110100 $(1\times2^{-1}+0\times2^{-2}+0\times2^{-3})\times2^{-2}=(0.125000)_{10}$

⋮

The pattern is repeated as larger quantities is formulated until largest value is reached.

00 11111 $=(1\times2^{-1}+1\times2^{-2}+1\times2^{-3})\times2^{-3}=(7)_{10}$ largest.

Several aspects of floating-point representation that have significance regarding computer round-off errors:

1. There is a limited Range of Quantities that may be Represented.
2. There are only a finite number of quantities that can be represented within the range
3. The interval between numbers, ΔX, increases as the numbers grow in magnitude.

---

1. There are large positive & negative numbers that cannot be represented. Attempts to employ numbers outside the acceptable range will result in an "overflow error".

Very small numbers cannot be represented due to limitation in floating-point representation.

→ Underflow "hole" between zero & the first positive number

2. limited precision; irrational numbers cannot be represented exactly. → The error is "quantizing errors" → Approximation is done in 2 ways: { - chopping (omit higher orders) or - Rounding

Ex. $\pi = 3.14159265358\ldots$ if it is stored on base-10 carrying 7 significant fig.

→ $\pi = 3.141592$

3. The quantizing errors will be proportional to the magnitude of the number being represented.

→ proportionality

$$\frac{|\Delta X|}{|X|} \leqslant \xi \quad \text{chopping is employed}$$

$$\frac{|\Delta X|}{|X|} \leqslant \frac{\xi}{2} \quad \text{Rounding " "}$$

ξ → machine epsilon ⇒ $\xi = b^{1-t}$ → number of significant digits.

number base