Approximations and Round-Off Errors

Chapter 3

Error Definitions

True error: E_t = True value – Approximation (+/-)

$$\textbf{True percent relative error: } \varepsilon_{\mathbf{t}} = \left| \frac{\textbf{True value} - \textbf{Approximation}}{\textbf{True value}} \right| \times 100\%$$

Approximate Error

- For numerical methods, the true value will be known only when we deal with functions that can be solved *analytically*.
- In real world applications, we usually do not know the answer a priori.

Approximate Error = CurrentApproximation(i) – PreviousApproximation(i-1)

$$\textbf{Approximate Relative Error: } \varepsilon_{\mathbf{a}} = \left| \frac{\textbf{Approximate error}}{\textbf{Approximation}} \right| \times 100\%$$

Iterative approaches

$$\text{Approx. Relative Error:} \quad \varepsilon_a = \left| \frac{(\text{Current Approx.}) - (\text{Previous Approx.})}{\text{Current Approx.}} \right| \times 100\%$$

Computations are repeated until stopping criterion is satisfied

$$|\varepsilon_a| \langle \ \varepsilon_s$$ ⟵ Pre-specified % tolerance based on your knowledge of the solution. (Use absolute value)

If ε_s is chosen as:

$$\varepsilon_s = (0.5 \times 10^{(2-n)})\%$$

Then the result is correct to *at least* n significant figures *(Scarborough 1966)*

EXAMPLE 3.2: *Maclaurin series expansion*

$$e^x = 1 + x + \frac{x^2}{2} + \frac{x^3}{3!} + \ldots + \frac{x^n}{n!}$$

Calculate $e^{0.5}$ (= 1.648721…) up to 3 significant figures. During the calculation process, compute the *true* and *approximate* percent relative errors at each step

Error tolerance $\longrightarrow$ $\varepsilon_s = (0.5 \times 10^{(2-3)})\% = 0.05\%$

Terms
1
1+(0.5)
$1+(.5)+(.5)^2/2$
$1+(.5)+(.5)^2/2+(.5)^3/6$

Count	Result	ε_t (%) True	ε_a (%) Approx.
1	1	39.3	
2	1.5	9.02	33.3
3	1.625	1.44	7.69
4	1.6458333	0.175	1.27
5	1.6484375	0.0172	0.158
6	1.648697917	0.00142	0.0158

Round-off and Chopping Errors

- Numbers such as **π, e**, or **$\sqrt{7}$** cannot be expressed by a fixed number of significant figures. Therefore, they can not be represented exactly by a computer which has a **fixed word-length**

$$\pi = 3.1415926535\ldots.$$

- Discrepancy introduced by this omission of significant figures is called ***round-off*** **or** ***chopping*** **errors**.

- If π is to be stored on a base-10 system carrying 7 significant digits,

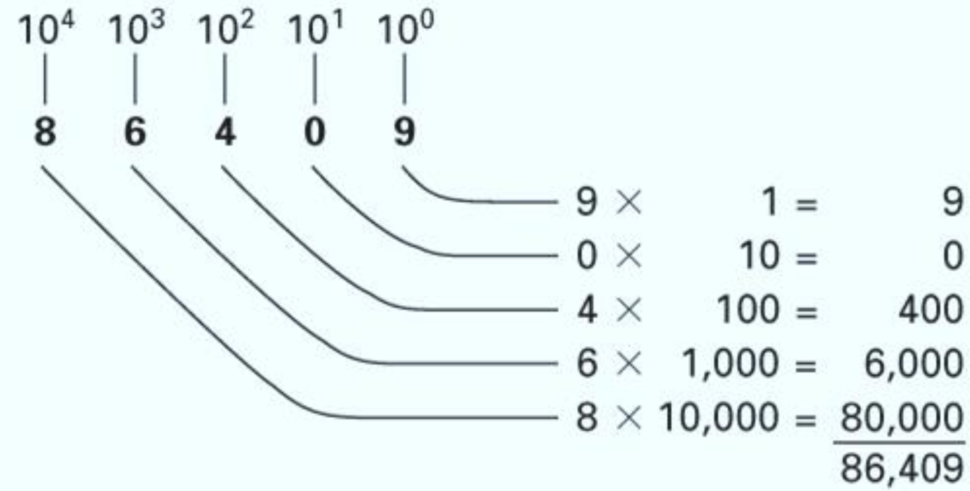
  chopping : $\pi=3.141592$ error: $\varepsilon_t=0.00000065$

  round-off: $\pi=3.141593$ error: $\varepsilon_t=0.00000035$

- Some machines use ***chopping***, because ***rounding*** has additional computational overhead.

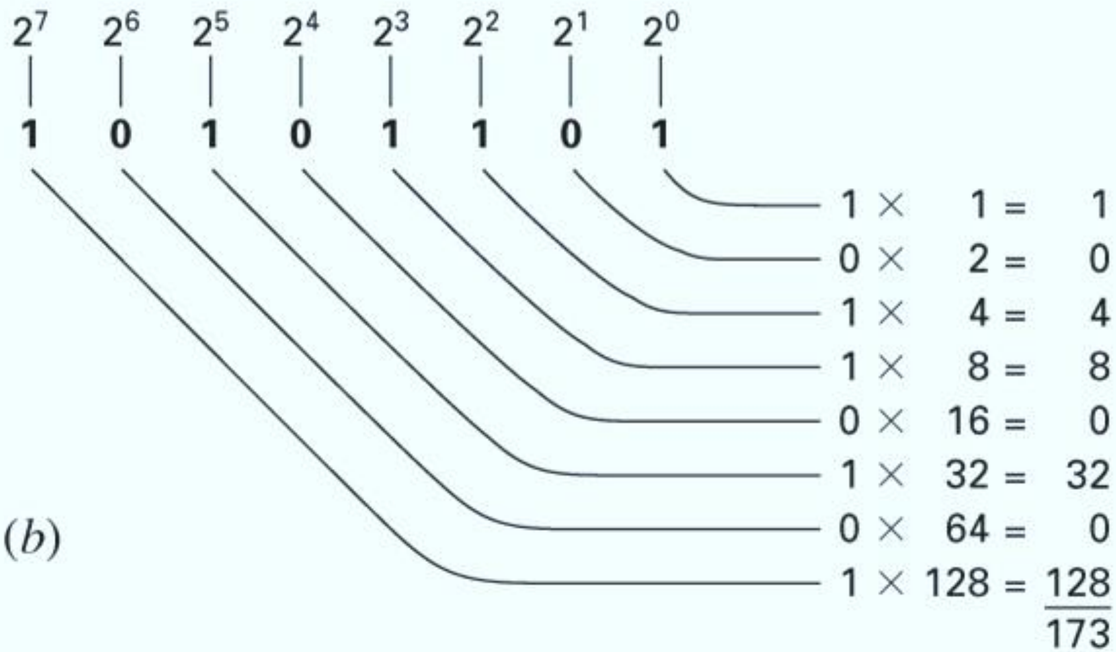
Number Representation

86409
in Base-10



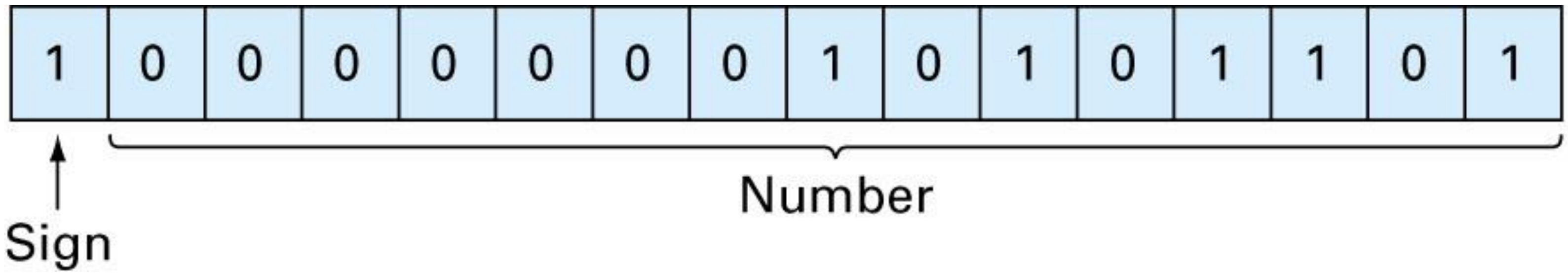
(a)

173
in Base-2

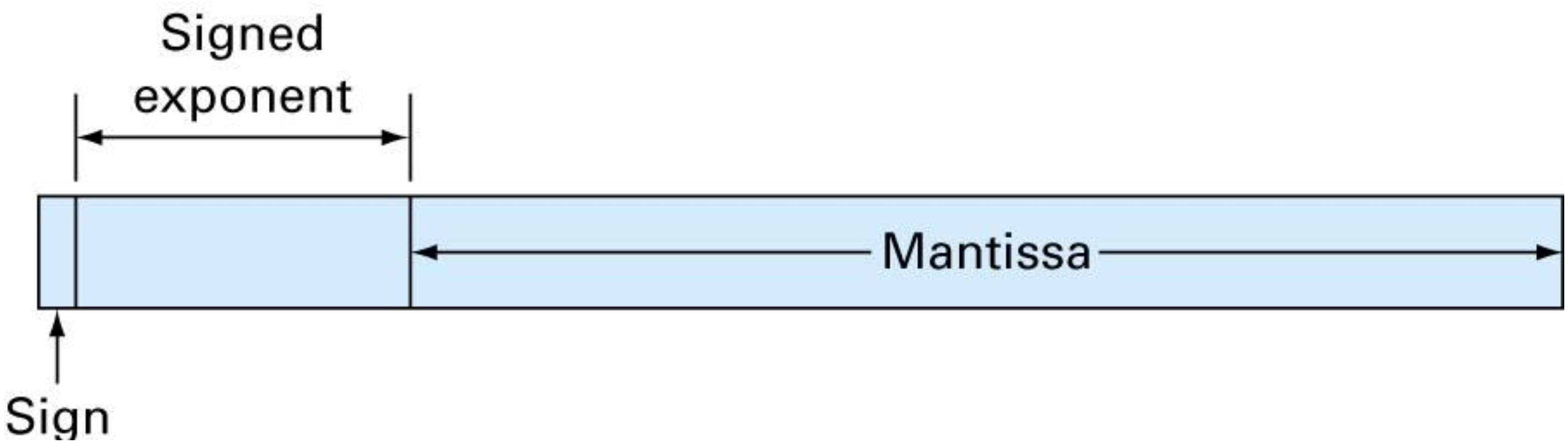


(b)

The representation of -173 on a 16-bit computer using the *signed magnitude method*



Computer representation of a floating-point number



exponent

$$m.b^e$$

mantissa

Base of the number system used

156.78 ⏩ ⏩ ⏩ 0.15678×10^3

(in a floating point base-10 system)

$$\frac{1}{34} = 0.029411765$$

Suppose only 4
decimal places to be stored

$$0.0294 \times 10^0$$

- ***Normalize* ➔ remove the leading zeroes.**
- Multiply the mantissa by 10 and lower the exponent by 1

$0.294\underline{1} \times 10^{-1}$

Additional significant figure is retained

- Due to ***Normalization***, absolute value of m is limited:

$$\frac{1}{b} \leq m < 1$$

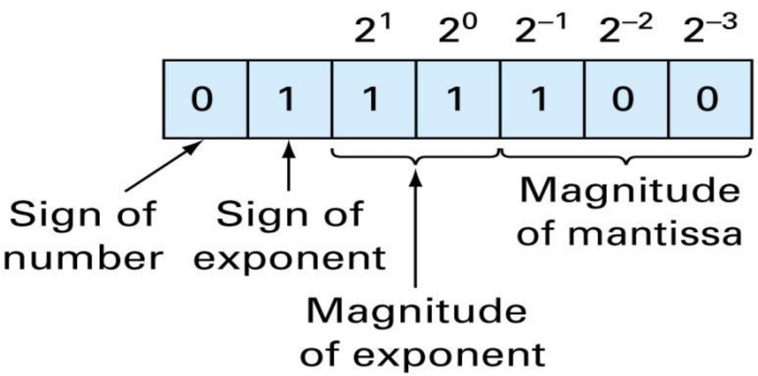
  for ***base-10*** system: **0.1 ≤ *m* < 1**

  for ***base-2*** system: **0.5 ≤ *m* < 1**

- Floating point representation allows both fractions and very large numbers to be expressed on the computer. However,
  - Floating point numbers take up more room
  - Take longer to process than integer numbers.

Q: What is the smallest positive floating point number that can be represented using a 7-bit word (3-bits reserved for mantissa).

What is the number?

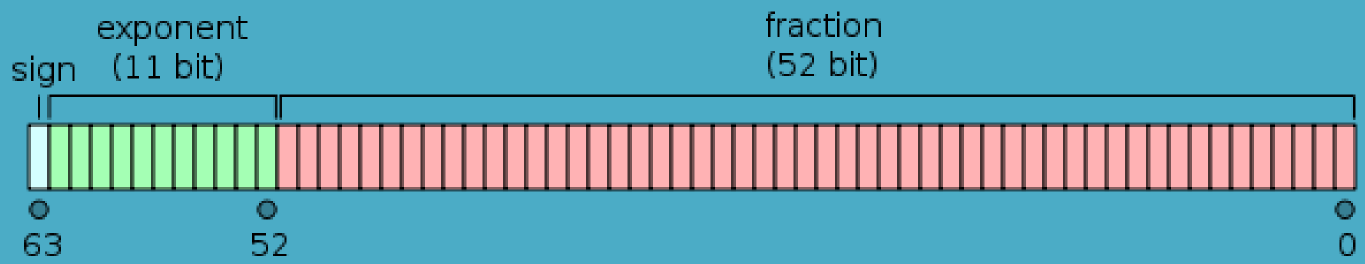


Another Exercise: What is the largest positive floating point number that can be represented using a 7-bit word (3-bits reserved for mantissa).

IEEE 754 double-precision binary floating-point format: binary64

This is a commonly used format on PCs.

- Sign bit: 1 bit
- Exponent width: 11 bits
- Significand precision: 53 bits (52 explicitly stored)



This gives from 15–17 significant decimal digits precision. If a decimal string with at most 15 significant digits is converted to IEEE 754 double precision representation and then converted back to a string with the same number of significant digits, then the final string should match the original.

Notes on floating point numbers:

- Overflow / Underflow

very small and very large numbers can not be represented using a fixed-length mantissa/exponent representation, therefore overflow and underflow can occur while doing arithmetic with these numbers.

- The interval between representable numbers increases as the numbers grow in magnitude and similarly, the round-off error.