Unit 1

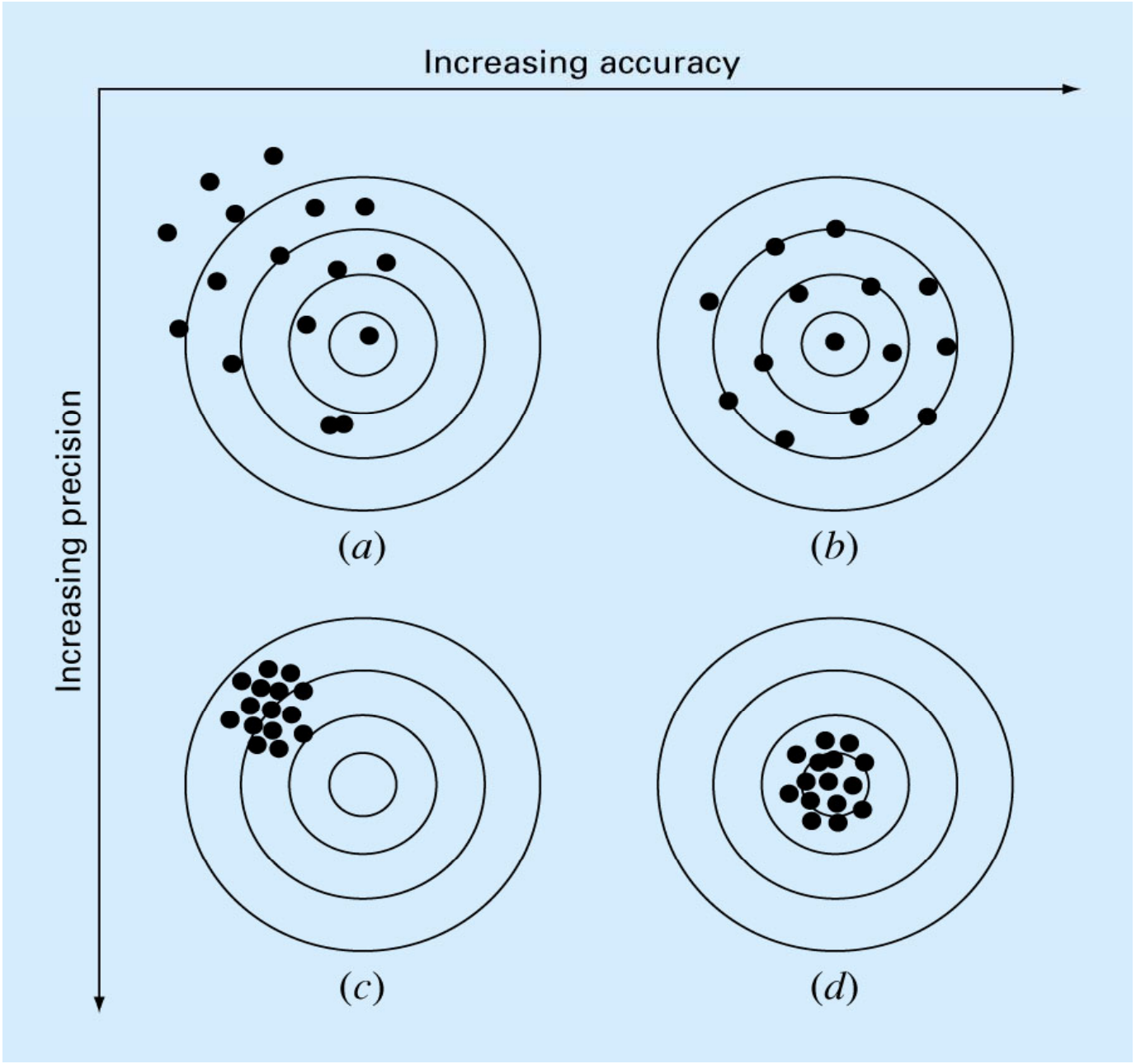
Approximations and Round-Off Errors

Chapter 3

- For many engineering problems, we cannot obtain analytical solutions.
- Numerical methods yield approximate results, results that are close to the exact analytical solution. We cannot exactly compute the errors associated with numerical methods.
  - Only rarely given data are exact, since they originate from measurements. Therefore there is probably error in the input information.
  - Algorithm itself usually introduces errors as well, e.g., unavoidable round-offs, etc …
  - The output information will then contain error from both of these sources.
- How confident we are in our approximate result?
- The question is *"how much error is present in our calculation and is it tolerable?"*

- Accuracy. How close is a computed or measured value to the true value
- Precision (or *reproducibility*). How close is a computed or measured value to previously computed or measured values.
- Inaccuracy (or *bias*). A systematic deviation from the actual value.
- Imprecision (or *uncertainty*). Magnitude of scatter.

Fig. 3.2



Significant Figures

- Number of significant figures indicates precision. Significant digits of a number are those that can be *used* with *confidence,* e.g., the number of certain digits plus one estimated digit.

53,800 How many significant figures?

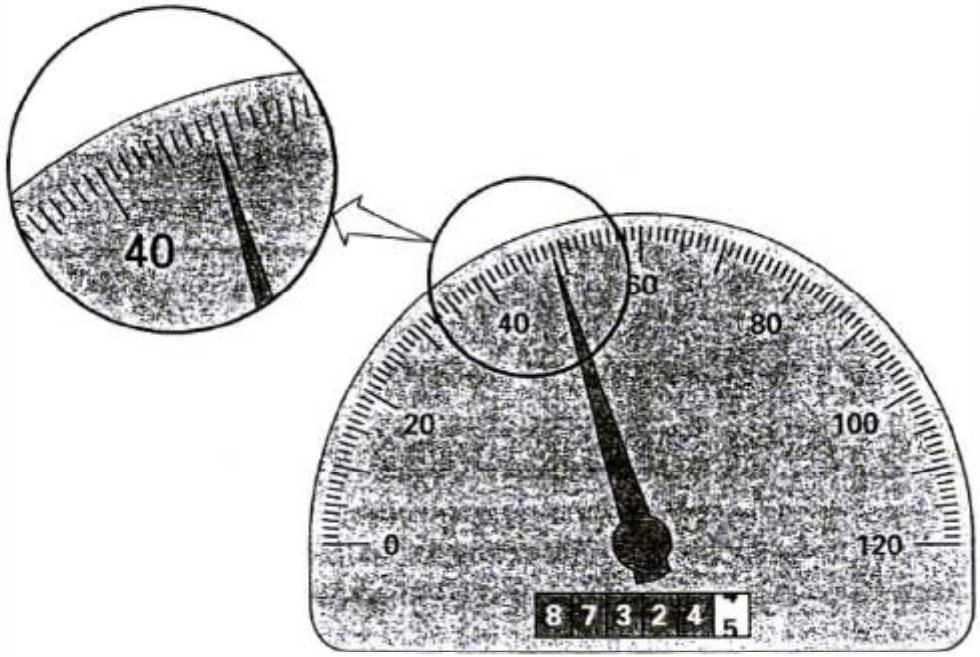
5.38×10^4	3
5.380×10^4	4
5.3800×10^4	5

Zeros are sometimes used to locate the decimal point not significant figures.

0.00001753	4
0.0001753	4
0.001753	4

FIGURE 3.1

An automobile speedometer and odometer illustrating the concept of a significant figure.



Error Definitions

True Value = Approximation + Error

E_t = True value – Approximation (+/-)

True error

$$\text{True fractional relative error} = \frac{\text{true error}}{\text{true value}}$$

$$\text{True percent relative error, } \varepsilon_t = \frac{\text{true error}}{\text{true value}} \times 100\%$$

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

$$\varepsilon_a = \frac{\text{Approximate error}}{\text{Approximation}} \times 100\%$$

- *Iterative approach*, example Newton's method

$$\varepsilon_a = \frac{\text{Current approximation - Previous approximation}}{\text{Current approximation}} \times 100\%$$

(+ / -)

- Use absolute value.
- Computations are repeated until stopping criterion is satisfied.

$$|\varepsilon_a| < \varepsilon_s$$

Pre-specified % tolerance based on the knowledge of your solution

- If the following criterion is met

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

you can be sure that the result is correct to at least n significant figures.

Calculation of Errors

Problem Statement. 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.

(a) The error for measuring the bridge is [Eq. (3.2)]

$$E_t = 10{,}000 - 9999 = 1 \text{ cm}$$

and for the rivet it is

$$E_t = 10 - 9 = 1 \text{ cm}$$

(b) The percent relative error for the bridge is [Eq. (3.3)]

$$\varepsilon_t = \frac{1}{10{,}000} 100\% = 0.01\%$$

and for the rivet it is

$$\varepsilon_t = \frac{1}{10} 100\% = 10\%$$

Thus, although both measurements have an error of 1 cm, the relative error for the rivet is much greater. We would conclude that we have done an adequate job of measuring the bridge, whereas our estimate for the rivet leaves something to be desired.

Error Estimates for Iterative Methods

Problem Statement. In mathematics, functions can often be represented by infinite series. For example, the exponential function can be computed using

$$e^x = 1 + x + \frac{x^2}{2} + \frac{x^3}{3!} + \cdots + \frac{x^n}{n!} \qquad \text{(E3.2.1)}$$

Thus, as more terms are added in sequence, the approximation becomes a better and better estimate of the true value of e^x. Equation (E3.2.1) is called a *Maclaurin series expansion.*

Starting with the simplest version, $e^x = 1$, add terms one at a time to estimate $e^{0.5}$. After each new term is added, compute the true and approximate percent relative errors with Eqs. (3.3) and (3.5), respectively. Note that the true value is $e^{0.5} = 1.648721\ldots$. Add terms until the absolute value of the approximate error estimate ε_a falls below a prespecified error criterion ε_s conforming to three significant figures.

Solution. First, Eq. (3.7) can be employed to determine the error criterion that ensures a result is correct to at least three significant figures:

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

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

The first estimate is simply equal to Eq. (E3.2.1) with a single term. Thus, the first estimate is equal to 1. The second estimate is then generated by adding the second term, as in

$$e^x = 1$$

or for $x = 0.5$,

$$e^{0.5} = 1 + 0.5 = 1.5$$

This represents a true percent relative error of [Eq. (3.3)]

$$\varepsilon_t = \frac{1.648721 - 1.5}{1.648721} 100\% = 9.02\%$$

Equation (3.5) can be used to determine an approximate estimate of the error, as in

$$\varepsilon_a = \frac{1.5 - 1}{1.5} 100\% = 33.3\%$$

Because ε_a is not less than the required value of ε_s, we would continue the computation by adding another term, $x^2/2!$, and repeating the error calculations. The process is continued until $\varepsilon_a < \varepsilon_s$. The entire computation can be summarized as

Terms	Result	ε_t (%)	ε_a (%)
1	1	39.3	
2	1.5	9.02	33.3
3	1.625	1.44	7.69
4	1.645833333	0.175	1.27
5	1.648437500	0.0172	0.158
6	1.648697917	0.00142	0.0158

```
FUNCTION IterMeth(val, es, maxit)
iter = 1
sol = val
ea = 100
DO
  solold = sol
  sol = ...
  iter = iter + 1
  IF sol ≠ 0 ea=abs((sol − solold)/sol)*100
  IF ea ≤ es OR iter ≥ maxit EXIT
END DO
IterMeth = sol
END IterMeth
```

FIGURE 3.3
Pseudocode for a generic iterative calculation.

Round-off Errors

- Numbers such as π, e, or $\sqrt{7}$ cannot be expressed by a fixed number of significant figures.
- Computers use a base-2 representation, they cannot precisely represent certain exact base-10 numbers.
- Fractional quantities are typically represented in computer using “floating point” form, e.g.,

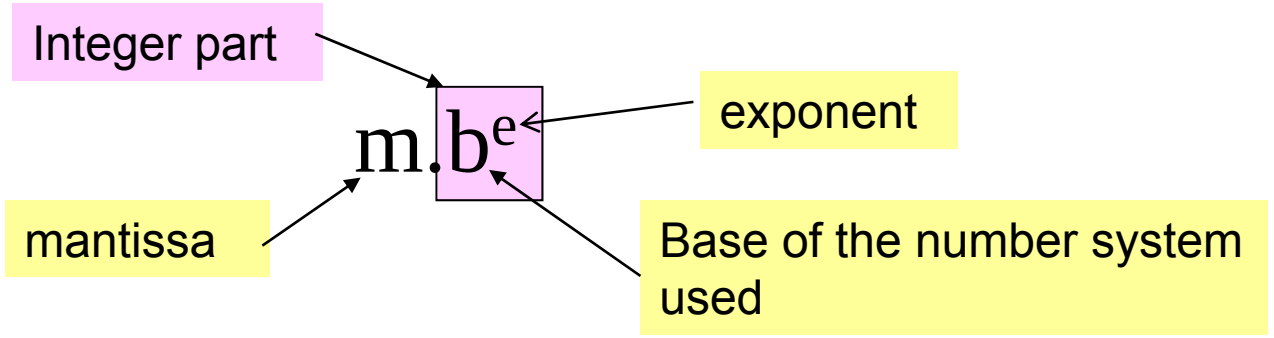


Figure 3.3

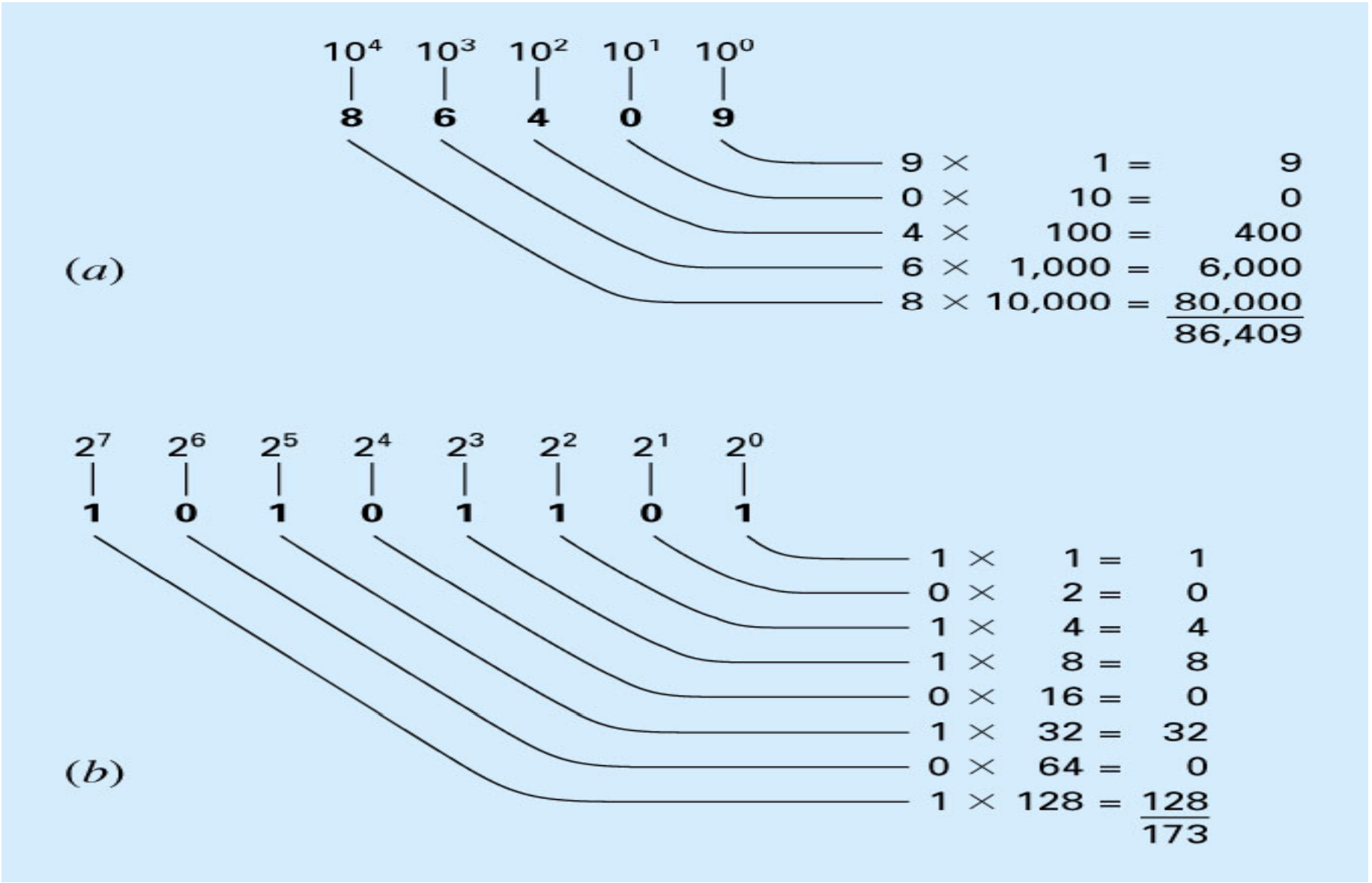
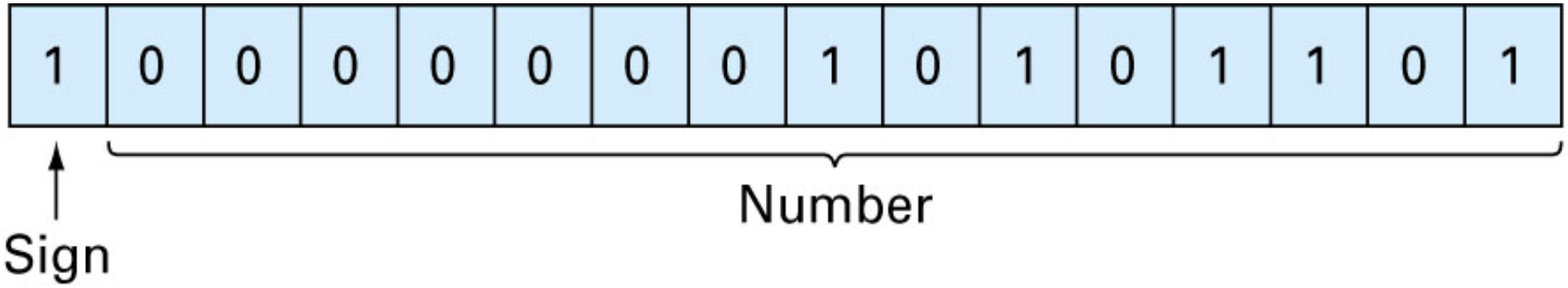


Figure 3.4



Range of Integers

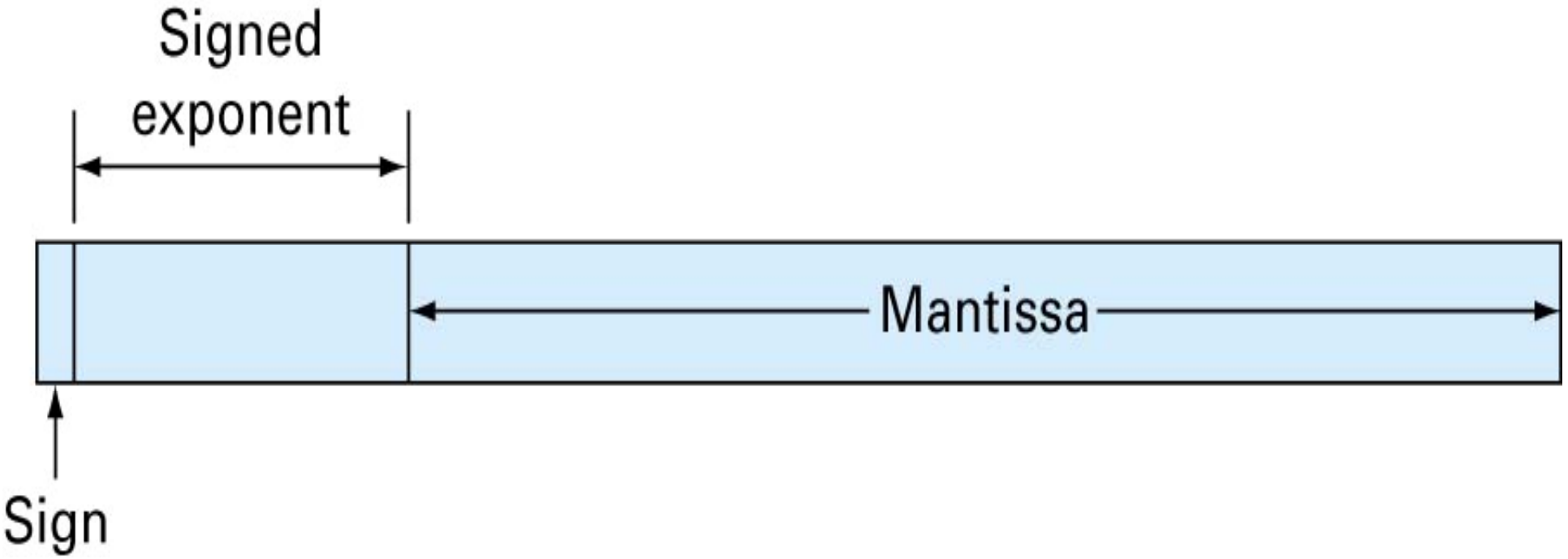
Problem Statement. Determine the range of integers in base-10 that can be represented on a 16-bit computer.

Solution. Of the 16 bits, the first bit holds the sign. The remaining 15 bits can hold binary numbers from 0 to 111111111111111. The upper limit can be converted to a decimal integer, as in

$$(1 \times 2^{14}) + (1 \times 2^{13}) + \cdots + (1 \times 2^{1}) + (1 \times 2^{0})$$

which equals 32,767 (note that this expression can be simply evaluated as $2^{15} - 1$). Thus, a 16-bit computer word can store decimal integers ranging from −32,767 to 32,767. In addition, because zero is already defined as 0000000000000000, it is redundant to use the number 1000000000000000 to define a "minus zero." Therefore, it is usually employed to represent an additional negative number: −32,768, and the range is from −32,768 to 32,767.

Figure 3.5



For instance, the number 156.78 could be represented as 0.15678×10^3 in a floating-point base-10 system.

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

0.0294×10^0

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

- Normalized to remove the leading zeroes. Multiply the mantissa by 10 and lower the exponent by 1

0.2941×10^{-1}

Additional significant figure is retained

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

Therefore

for a base-10 system 0.1 ≤m<1

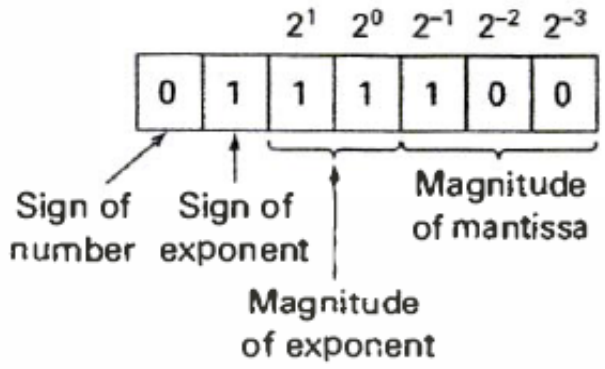
for a base-2 system 0.5 ≤m<1

- 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.
  - Round-off errors are introduced because mantissa holds only a finite number of significant figures.

Hypothetical Set of Floating-Point Numbers

Problem Statement. Create a hypothetical floating-point number set for a machine that stores information using 7-bit words. Employ the first bit for the sign of the number, the next three for the sign and the magnitude of the exponent, and the last three for the magnitude of the mantissa (Fig. 3.8).

FIGURE 3.8
The smallest possible positive floating-point number from Example 3.5.



Solution. The smallest possible positive number is depicted in Fig. 3.8. The initial 0 indicates that the quantity is positive. The 1 in the second place designates that the exponent has a negative sign. The 1's in the third and fourth places give a maximum value to the exponent of

$$1 \times 2^1 + 1 \times 2^0 = 3$$

Therefore, the exponent will be -3. Finally, the mantissa is specified by the 100 in the last three places, which conforms to

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

Although a smaller mantissa is possible (e.g., 000, 001, 010, 011), the value of 100 is used because of the limit imposed by normalization [Eq. (3.8)]. Thus, the smallest possible positive number for this system is $+0.5 \times 2^{-3}$, which is equal to 0.0625 in the base-10 system. The next highest numbers are developed by increasing the mantissa, as in

$$0111101 = (1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3}) \times 2^{-3} = (0.078125)_{10}$$
$$0111110 = (1 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3}) \times 2^{-3} = (0.093750)_{10}$$
$$0111111 = (1 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3}) \times 2^{-3} = (0.109375)_{10}$$

Notice that the base-10 equivalents are spaced evenly with an interval of 0.015625.

At this point, to continue increasing, we must decrease the exponent to 10, which gives a value of

$$1 \times 2^1 + 0 \times 2^0 = 2$$

The mantissa is decreased back to its smallest value of 100. Therefore, the next number is

$$0110100 = (1 \times 2^{-1} + 0 \times 2^{-2} + 0 \times 2^{-3}) \times 2^{-2} = (0.125000)_{10}$$

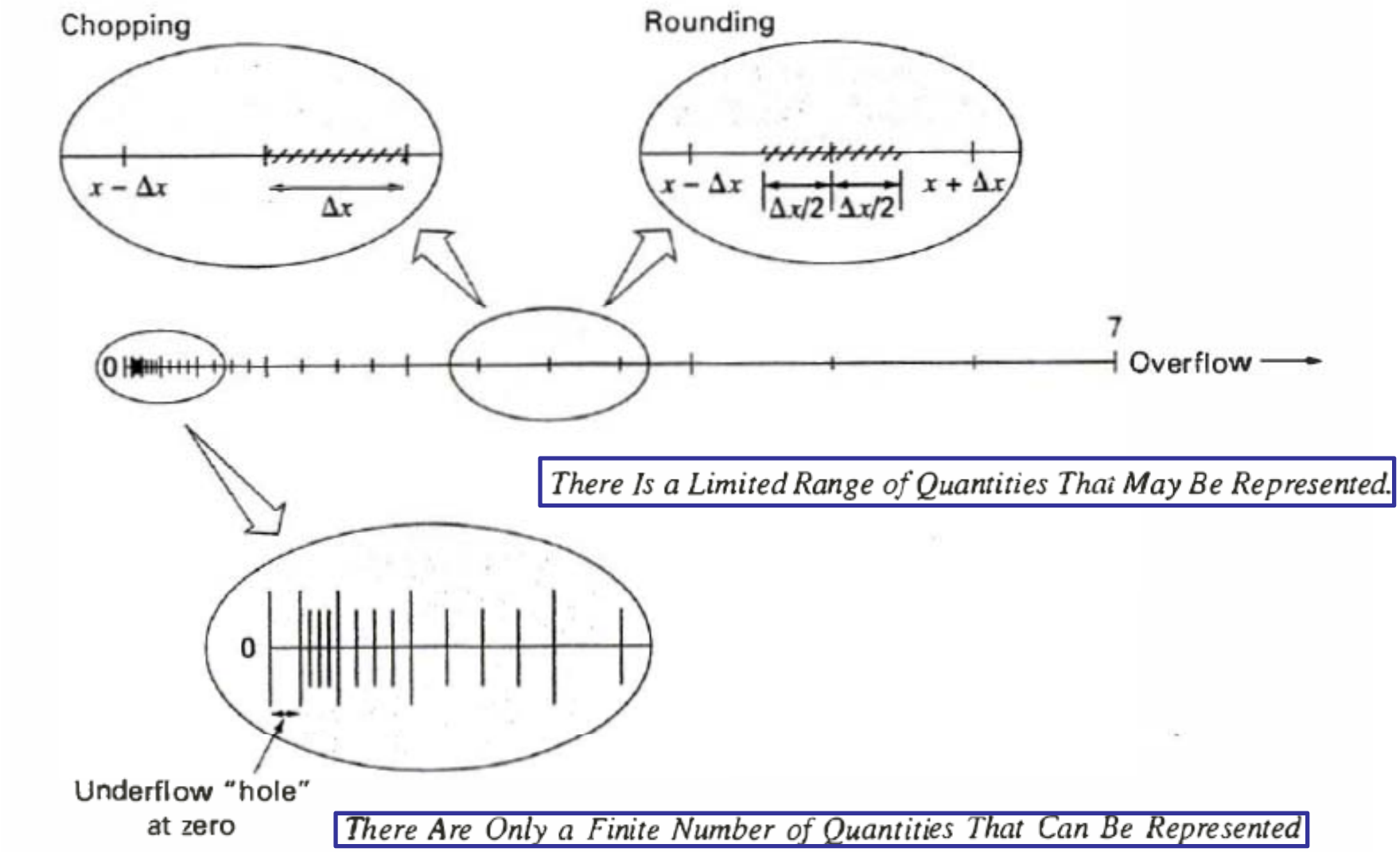
This still represents a gap of $0.125000 - 0.109375 = 0.015625$. However, now when higher numbers are generated by increasing the mantissa, the gap is lengthened to 0.03125,

$$0110101 = (1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3}) \times 2^{-2} = (0.156250)_{10}$$
$$0110110 = (1 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3}) \times 2^{-2} = (0.187500)_{10}$$
$$0110111 = (1 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3}) \times 2^{-2} = (0.218750)_{10}$$

This pattern is repeated as each larger quantity is formulated until a maximum number is reached,

$$0011111 = (1 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3}) \times 2^{3} = (7)_{10}$$

The Interval between Numbers, Δx, Increases as the Numbers Grow in Magnitude.



Chopping

Example:

π=3.14159265358 to be stored on a base-10 system carrying 7 significant digits.

π=3.141592 chopping error ε_t=0.00000065

If rounded

π=3.141593 ε_t=0.00000035

- Some machines use chopping, because rounding adds to the computational overhead. Since number of significant figures is large enough, resulting chopping error is negligible.