Chapter 2

Computer Arithmetic

In this chapter, we explain the floating-point number system and develop basic facts about roundoff errors. Some topics such as loss of significance, stability of numerical algorithms, and condition of problems will be introduced with examples. Error analysis of computer arithmetic will be brifely discussed.

2.1 Floating-Point Number and Roundoff Error

In general, a nonzero real number x in the decimal number system can be written as

$$x = \pm r \times 10^n,$$

where

$$\frac{1}{10} \leq r < 1,$$

and n is an integer (positive, negative, or zero). This representation is the called normalized scientific notation, r is called the mantissa and n is the exponent. Note that the leading digit in the fraction is not zero (except when the number involved is zero). For example,

$$\begin{aligned} 42.965 &= 4 \times 10^1 + 2 \times 10^0 + 9 \times 10^{-1} + 6 \times 10^{-2} + 5 \times 10^{-3} \\ &= 0.42965 \times 10^2, \\ -0.00234 &= -0.234 \times 10^{-2}. \end{aligned}$$

Likewise, we can use the scientific notation for binary number system to express the number x as

$$x = \pm q \times 2^m$$

with

$$\frac{1}{2} \leq q < 1,$$

and some integer m. Both q and m will be expressed in terms of binary numbers. For example,

$$\begin{aligned} 1001.1101 &= 1\times 2^3 + 1\times 2^0 + 1\times 2^{-1} + 1\times 2^{-2} + 1\times 2^{-4} \\ &= 0.10011101 \times 2^{100} \\ &= (9.8125)_{10} \end{aligned}$$

Allmost all microcomputers uses binary number system, but not necessary mainframe computers.

Example 2.1 *What is the binary representation of $\frac{2}{3}$?*

Sol: To determine the binary representation for $\frac{2}{3}$, we write

$$\frac{2}{3} = (0.a_1a_2a_3\ldots)_2.$$

Multiply by 2 to obtain

$$\frac{4}{3} = (a_1.a_2a_3\ldots)_2.$$

Therefore, we get $a_1 = 1$ by taking the integer part of both sides. Substracting 1, we have

$$\frac{1}{3} = (0.a_2a_3a_4\ldots)_2.$$

Repeating the previous step, we arrive at

$$\frac{2}{3} = (0.101010\ldots)_2.$$

■

Most computers work internally in binary system and output decimal system for human users. A number that has a terminating exapnsion in one base may have a nonterminating exapnsion in another. The conversion procedure frequently involves a small error as shown in the following example.

Example 2.2 *What is the binary representation of $\frac{1}{10}$?*

Sol:

$$\frac{1}{10} = (0.1)_{10} = (0.0001\ 1001\ 1001\ 1001\ \ldots)_2.$$

■

Typically, only a relatively small subset of the real number system is used for the representation of all the real numbers. This subset, which are called the *floating-point numbers*, contains only rational numbers, both positive and negative. When a number can not be represented exactly with the fixed finite number of digits in a computer, a near-by floating-point number is chosed for approximate representation.

For any real number x, let

$$x = \pm 0.a_1 a_2 \cdots a_t a_{t+1} a_{t+2} \cdots \times 2^m, \quad a_1 \neq 0, \tag{2.1}$$

denote the normalized scientific binary representation of x. Note that the leading digit in mantissa $a_1 \neq 0$, hence $a_1 = 1$. If x is within the numerical range of the machine, the floating-point form of x, denoted $fl(x)$, is obtained by terminating the mantissa of x at t digits for some integer t. There are two ways of preforming this termination.

1. **chopping:** simply discard the excess bits $a_{t+1}, a_{t+2}, \ldots$ to obtain

$$fl(x) = \pm 0.a_1 a_2 \cdots a_t \times 2^m. \tag{2.2}$$

2. **rounding up:** add $2^{-(t+1)} \times 2^m$ to x and then chop the excess bits to obtain a number of the form

$$fl(x) = \pm 0.\delta_1 \delta_2 \cdots \delta_t \times 2^m. \tag{2.3}$$

   In this method, if $a_{t+1} = 1$, we add 1 to a_t to obtain $fl(x)$, and if $a_{t+1} = 0$, we merely chop off all but the first t digits.

Since computers can only store real numbers using fixed finite numbers of digits, this places a restriction on the precision with which real numbers can be represented. The error results from replacing a number with its floating-point form is called *roundoff error* or *rounding error*, regardless of whether the rounding or chopping method is used. The following definition describes two methods for measuring approximation errors.

Definition 2.1 (Absolute Error and Relative Error) *If x is an approximation to the exact value $x^\star$, the absolute error is $|x^\star - x|$ and the relative error is $\frac{|x^\star - x|}{|x^\star|}$, provided that $x^\star \neq 0$.*

Remark 2.1 *As a measure of accuracy, the absolute error may be misleading and the relative error more meaningful.*

If the floating-point representation $fl(x)$ for the number x is obtained by using t digits and chopping procedure, then the relative error is

$$\frac{|x - fl(x)|}{|x|} = \frac{|0.00 \cdots 0 a_{t+1} a_{t+2} \cdots \times 2^m|}{|0.a_1 a_2 \cdots a_t a_{t+1} a_{t+2} \cdots \times 2^m|} = \frac{|0.a_{t+1} a_{t+2} \cdots|}{|0.a_1 a_2 \cdots a_t a_{t+1} a_{t+2} \cdots|} \times 2^{-t}.$$

Since $a_1 \neq 0$, the minimal value of the denominator is $\frac{1}{2}$. The numerator is bounded above by 1. As a consequence

$$\left| \frac{x - fl(x)}{x} \right| \leq 2^{-t+1}. \tag{2.4}$$

In a similar manner, if t-digit rounding arithmetic is used and $a_{t+1} = 0$, then $fl(x) = \pm 0.a_1 a_2 \cdots a_t \times 2^m$. A bound for the relative error is

$$\frac{|x - fl(x)|}{|x|} = \frac{|0.a_{t+1}a_{t+2}\cdots|}{|0.a_1a_2\cdots a_ta_{t+1}a_{t+2}\cdots|} \times 2^{-t} \leq 2^{-t}.$$

since the numerator is bounded above by $\frac{1}{2}$. But if $a_{t+1} = 1$, then $fl(x) = \pm(0.a_1a_2\cdots a_t + 2^{-t}) \times 2^m$, the upper bound for relative error becomes

$$\frac{|x - fl(x)|}{|x|} = \frac{|1 - 0.a_{t+1}a_{t+2}\cdots|}{|0.a_1a_2\cdots a_ta_{t+1}a_{t+2}\cdots|} \times 2^{-t} \leq 2^{-t}.$$

since the numerator is bounded by $\frac{1}{2}$ due to $a_{t+1} = 1$. Therefore the relative error for rounding arithmetic is

$$\left|\frac{x - fl(x)}{x}\right| \leq 2^{-t} = \frac{1}{2} \times 2^{-t+1}. \tag{2.5}$$

The number $\epsilon_M \equiv 2^{-t+1}$ is referred to as the *unit roundoff error* or *machine epsilon.* The floating-point representation, $fl(x)$, of x can be expressed as

$$fl(x) = x(1+\delta), \qquad |\delta| \leq \varepsilon_M. \tag{2.6}$$

In 1985, the IEEE (Institute for Electrical and Electronic Engineers) published a report called *Binary Floating Point Arithmetic Standard 754-1985.* In this report, formats were specified for single, double, and extended precisions, and these standards are generally followed by microcomputer manufactures using floating-point hardware.

The single precision IEEE standard floating-point format allocates 32 bits for the normalized floating-point number $\pm q \times 2^m$ as shown in Figure 2.1. The first bit is a sign indicator, denoted s. This is followed by an 8-bit exponent c and a 23-bit mantissa f. The base for the exponent and mantissa is 2, and the actual exponent is $c - 127$. The value of c is restricted by the inequality $0 < c < 255$. The values 0 and 255 are reserved for special cases including ± 0 and $\pm\infty$, respectively. Hence the actual exponent of the number is restricted by the inequality $-126 \leq c - 127 \leq 127$. In addition, a normalization is imposed that requires that the leading digit in fraction be 1, and this digit is not stored as part of the 23-bit mantissa. Using this format gives a floating-point number of the form

$$(-1)^s \times (1.f)_2 \times 2^{c-127}.$$

Since the mantissa f actually corresponds to 24 binary digits (i.e., precision $t = 24$), the machine epsilon is

$$\epsilon_M = 2^{-24+1} = 2^{-23} \approx 1.192 \times 10^{-7}. \tag{2.7}$$

This approximately corresponds to 6 accurate decimal digits. And the first single precision floating-point number greater than 1 is $1 + 2^{-23}$.

The largest number that can be represented by the single precision format is approximately $2^{128} \approx 3.403 \times 10^{38}$, and the smallest positive number is $2^{-126} \approx 1.175 \times 10^{-38}$. This

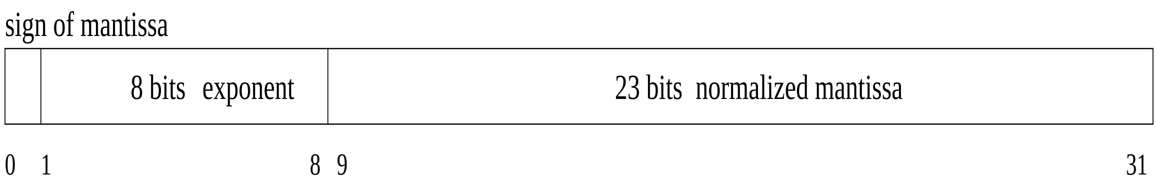

Figure 2.1: 32-bit single precision.

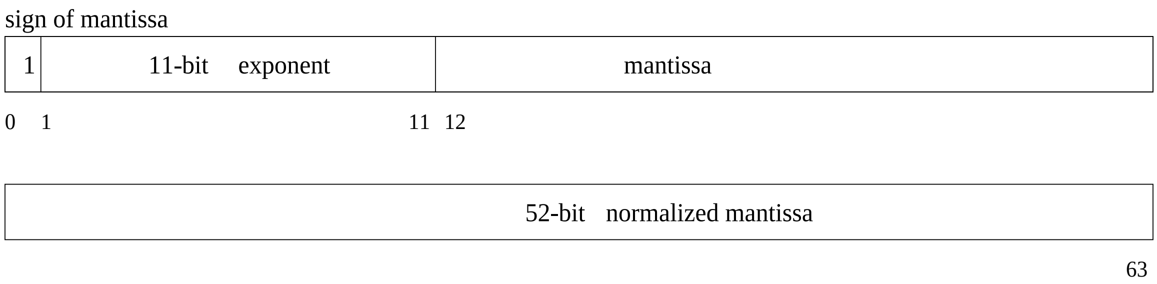

Figure 2.2: 64-bit double precision.

range is in general not sufficient for some scientific calculation. For this and other reason, we need "double precision" arithmetic.

A floating point number in double precision IEEE standard format uses two words (64 bits) to store the number as shown in Figure 2.2. The first bit is a sign indicator, denoted s, same as the single precision format. This is followed by an 11-bit exponent c and a 52-bit mantissa f. The actual exponent is $c-1023$. Using this format gives a floating-point number of the form

$$(-1)^s \times (1.f)_2 \times 2^{c-1023}.$$

which provides between 15 and 16 decimal digits of accruacy, since the machine epsilon

$$\epsilon_M = 2^{-52} \approx 2.220 \times 10^{-16},$$

and a range of approximately $2^{-1022} \approx 2.225 \times 10^{-308}$ to $2^{1024} \approx 1.798 \times 10^{308}$. The first double precision floating-point number greater than 1 is $1 + 2^{-52}$.

Table 2.1 summarizes some characteristics of IEEE standard floating-point representations. Clearly, for the most accuracy, computations should be done using double precision floating-point numbers, however, the execution time is much higher.

In the IEEE floating-point standard, the *round to nearest* or *correctly rounded* value of the real number x, denoted $round(x)$, is defined as follows. First, let x_+ be the closest foating-point number greater than x and x_- be the closest one less than x. Then $round(x)$ is either x_+ or x_-, whichever is nearer to x. If x is a floating-point number, then $round(x) = x$. This method is almost always used since it is the most useful and gives the floating-point number closest to x. Other rounding modes could be (1) *round towards 0*: $round(x)$ is either x_+ or x_-, whichever is between 0 and x; (2) *round towards* $-\infty$/*round dwon*: $round(x) = x_-$; (3) *round towards* ∞/*round up*: $round(x) = x_+$.

	single precision	double precision
ϵ_M	$2^{-23} \approx 1.192 \times 10^{-7}$	$2^{-52} \approx 2.220 \times 10^{-16}$
smallest positive number	$2^{-126} \approx 1.175 \times 10^{-38}$	$2^{-1022} \approx 2.225 \times 10^{-308}$
largest number	$2^{128} \approx 3.403 \times 10^{38}$	$2^{1024} \approx 1.798 \times 10^{308}$
decimal precision	6	15

Table 2.1: Some characteristics of IEEE standard floating-point numbers

If a number $x = \pm q \times 2^m$ with m outside the computer's possible range (too large or two small), then we say that an *overflow* or an *underflow* has occurred. Generally, an overflow results in a fatal error (or exception), and the normal excution of the program stops. An underflow, however, is usually treated automatically by setting x to zero without any interruption of the program but with a warning message in most computers.

There are some useful special numbers in the IEEE standard. For example, instead of terminating with an overflow when dividing a nonzero number by 0, the machine representation for ∞, Inf, is stored, which is the mathematically sensible thing to do. There are two different representations, +Inf and −Inf, that correspond to two quite different numbers, $+\infty$ and $-\infty$. A NaN stands for Not a Number and is an error pattern rather than a number. Table 2.2 lists the IEEE exception handling standard.

big*big	± Inf	overflow
number/0.0	± Inf	division
0.0/0.0	NaN	invalid
small/big	subnormal number	underflow
2.0/3.0	rounded	

Table 2.2: IEEE exception handling.

Because of the hidden bit representation, a special technique for storing zero is necessary. Note that all zero bits in the mantissa field represents the significant 1.0 rather than 0.0. Moreover, there are two different representations for the same number zero; namely +0 and −0.

For single precision integers, 31 bits are allocated for the integer, because onley 1 bit is needed for the sign. Consequently, the range for integers is from $-(2^{31} - 1)$ to $(2^{31} - 1) = 2147483647$. Double precision integers, on the other hand, use 63-bit storage and 1 bit for sign. Pure integer calculation is not common in numerical algorithms.

2.2 Loss of Significance, Stability, and Conditioning

Roundoff errors are inevitable and difficult to control. Other types of errors which occur in computation may be under our control. The subject of numerical analysis is largely preoccupied with understanding and controlling errors of various kinds. Here in this section we examine some of them.

2.2.1 Loss of Significance

Suppose that x is a real number expressed in normalized scientific notation in the decimal system

$$x = \pm a_1 a_2 a_3 \cdots \times 10^n \qquad a_1 \neq 0.$$

The digits $a_1, a_2, a_3, \ldots,$ used to express the fraction part of x do not all have the same significance because they represent different power of 10. Thus, we say that a_1 is the most significant digit, and the significance of the digits diminishes from left to right.

One of the most common error-producing calculations involves the cancellation of significant digits due to the subtraction of nearly equal numbers (or the addition of one very large number and one very small number). Assume that two nearly equal numbers x and y, with $x > y$, have the t-digit floating-point representations

$$fl(x) = 0.d_1 d_2 \cdots d_p \alpha_{p+1} \alpha_{p+2} \cdots \alpha_t \times 10^n,$$

and

$$fl(x) = 0.d_1 d_2 \cdots d_p \beta_{p+1} \beta_{p+2} \cdots \beta_t \times 10^n.$$

Then the floating-point form of $x - y$ is

$$fl(fl(x) - fl(y)) = 0.\sigma_{p+1} \sigma_{p+2} \cdots \sigma_t \times 10^n,$$

where

$$0.\sigma_{p+1} \sigma_{p+2} \cdots \sigma_t = 0.\alpha_{p+1} \alpha_{p+2} \cdots \alpha_t - 0.\beta_{p+1} \beta_{p+2} \cdots \beta_t.$$

As a result, the floating-point number used to represent $x - y$ has at most $t - p$ digits of significance. However, in most computers, $x - y$ will be assigned t digits, with the last p digits being either zero or randomly assigned. Any further calculations involving the results of $x - y$ retain the problem of having only $t - p$ digits of significance, since a chain of calculations can not be expected to be more accurate than its weakest portion. The phenomenon can be illustrated with the following example.

Example 2.3 *If $x = 0.3721478693$ and $y = 0.3720230572$, what is the relative error in the computation of $x - y$ using five decimal digits of accuracy?*

Sol: In exact computation using ten decimal digits of accuracy,

$$x - y = 0.0001248121.$$

But both x and y will be rounded to five decimal digits before subtraction. Thus

$$\begin{aligned} fl(x) &= 0.37215 \\ fl(y) &= 0.37202 \\ fl(x) - fl(y) &= 0.00013 = 0.13000 \times 10^{-3} \end{aligned}$$

Therefore the relative error is

$$\frac{(x-y)-(fl(x)-fl(y))}{x-y} \approx 0.04 = 4\%.$$

■

An interesting question arised immediately would be how many significant binary bits are lost in the subtraction when x is close to y?

Theorem 2.1 *If $x \geq 0$ and $y \geq 0$ are normalized floating-point binary numbers such that $x > y$ and*

$$2^{-q} \leq 1 - \frac{y}{x} \leq 2^{-p},$$

then at most q and at least p significant binary digits are lost in the subtraction $x - y$.

Proof: Write

$$x = r \times 2^n, \qquad \frac{1}{2} \leq r < 1$$

and

$$y = s \times 2^m, \qquad \frac{1}{2} \leq s < 1.$$

Since $x > y$, we must sift the decimal digits of y to the right

$$y = (s \times 2^{m-n}) \times 2^n.$$

Then

$$x - y = (r - s \times 2^{m-n}) \times 2^n = r\left(1 - \frac{s \times 2^m}{r \times 2^n}\right) \times 2^n = r\left(1 - \frac{y}{x}\right) \times 2^n.$$

By assumption $2^{-q} \leq 1 - \frac{y}{x} \leq 2^{-p}$, hence

$$r\left(1 - \frac{y}{x}\right) < 1 \cdot 2^{-p} = 2^{-p}.$$

This means that to normalize the result $x - y$, a shift of at least p bits to the left is required. Similarly,

$$r\left(1 - \frac{y}{x}\right) \geq \frac{1}{2} \cdot 2^{-q} = 2^{-(q+1)},$$

and a shift of at most q bits to the right is required. ■

Sometimes, loss of significance can be avoided by using double precision. This at least doubles the number of bits in mantissa. But when both operands are already in double precision, this means quadruple precision would be needed, and this is not available in many machines and languages, or at least not cheaply. In some cases, rewriting the mathematical formula is an alternative and can make the difference.

Example 2.4 *Consider the two equivalent functions*

$$f(x) = x(\sqrt{x+1} - \sqrt{x}) \quad \textit{and} \quad g(x) = \frac{x}{\sqrt{x+1} + \sqrt{x}}.$$

Compare the function evaluation of $f(500)$ and $g(500)$ using 6 digits and rounding.

Sol:

$$\begin{aligned} f(500) &= 0.500000 \times 10^3 \times (\sqrt{501} - \sqrt{500}) \\ &= 0.500000 \times 10^3 \times (0.223830 \times 10^2 - 0.223607 \times 10^2) \\ &= 0.500000 \times 10^3 \times 0.223000 \\ &= 0.111500 \times 10^3 \end{aligned}$$

and

$$\begin{aligned} g(500) &= \frac{500}{\sqrt{501} + \sqrt{500}} \\ &= \frac{0.500000 \times 10^3}{0.223830 \times 10^2 + 0.223607 \times 10^2} \\ &= \frac{0.500000 \times 10^3}{0.447437 \times 10^2} \\ &= 0.111748 \times 10^2 \end{aligned}$$

If more digits are used, we can calculated

$$\begin{aligned} f(500) &= 500 \times (\sqrt{501} - \sqrt{500}) \\ &= 500 \times (22.38302929 - 22.36067977) \\ &= 500 \times 0.022349516 \\ &= 11.1747553 \end{aligned}$$

Hence it can be argued that the formulation $g(x)$ is better. ■

Example 2.5 *The quadratic formulas for computing the roots of $ax^2 + bx + c = 0$, when $a \neq 0$, are*

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \qquad \textit{and} \qquad x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}.$$

Consider the quadratic equation $x^2 + 62.10x + 1 = 0$ and discuss the numerical results.

Sol: Using the quadratic formula and 8-digit rounding arithmetic, one can obtain the roots

$$x_1 = -0.01610723 \qquad \text{and} \qquad x_2 = -62.08390.$$

We will use these values as “exact solutions”. Now we perform the calculations with 4-digit rounding arithmethic. First we have

$$\sqrt{b^2 - 4ac} = \sqrt{62.10^2 - 4.000} = \sqrt{3856 - 4.000} = \sqrt{3852} = 62.06,$$

and

$$fl(x_1) = \frac{-62.10 + 62.06}{2.000} = \frac{-0.04000}{2.000} = -0.02000.$$

The relative error in computing x_1 is

$$\frac{|fl(x_1) - x_1|}{|x_1|} = \frac{|-0.02000 + 0.01610723|}{|-0.01610723|} = \frac{0.00389277}{0.01610723} \approx 0.2417.$$

In calculating x_2,

$$fl(x_2) = \frac{-62.1062.06}{2.000} = \frac{-124.2}{2.000} = -62.10,$$

and the relative error in computing x_2 is

$$\frac{|fl(x_2) - x_2|}{|x_2|} = \frac{|-62.10 + 62.08390|}{|-62.08390|} = \frac{0.0161}{62.08390} \approx 0.259 \times 10^{-3}.$$

In this equation, $b^2 = 62.10^2$ is much larger than $4ac = 4$. Hence b and $\sqrt{b^2 - 4ac}$ become two nearly equal numbers. The calculation of x_1 involves the subtraction of two nearly equal numbers, in contrast, the calculation of x_2 involves the addition of the nearly equal numbers which will not cause serious loss of significance.

To obtain a more accurate 4-digit rounding approximation for x_1, we change the formulation by rationalizing the numerator, that is,

$$x_1 = \frac{-2c}{b + \sqrt{b^2 - 4ac}}.$$

Then

$$fl(x_1) = \frac{-2.000}{62.10 + 62.06} = \frac{-2.000}{124.2} = -0.01610.$$

The relative error in computing x_1 is now reduced to 0.62×10^{-3}. However, if we rationalize the numberator in x_2 to get

$$x_2 = \frac{-2c}{b - \sqrt{b^2 - 4ac}}.$$

The use of this formula results not only involve the subtraction of two nearly equal numbers but also the division by the small number. This would cause degrade in accuracy.

$$fl(x_2) = \frac{-2.000}{62.10 - 62.06} = \frac{-2.000}{0.04000} = -50.00.$$

The relative error in x_2 becomes 0.19. ■

The above example also shows that if a finite-precision representation or calculation introduces an error, further enlargement of the error occurs when dividing by a number with small magnitude, or, equivalently, when multiplying by a number with large magnitude.

There is another situation in which a drastic loss of significant digits will occur. This is in the evaluation of polynomials and certain functions.

Example 2.6 *Evaluate*

$$f(x) = x^3 - 6x^2 + 3x - 0.149$$

at $x = 4.71$ *using 3-digit arithmetic.*

Sol: The following table gives the intermediate results in the calculation.

	x	x^2	x^3	$6x^2$	$3x$
Exact	4.71	22.1841	104.487111	133.1046	14.13
3-digit chopping	4.71	22.1	104	132	14.1
3-digit rounding	4.71	22.2	105	133	14.1

The exact function value is

$$f(4.71) = -14.636489.$$

The result from 3-digit chopping arithmetic is

$$\hat{f}(4.71) = -14.0,$$

and 3-digit rounding

$$\tilde{f}(4.71) = -14.0.$$

Hence relative error for both calculation is

$$\frac{|-14.636489 + 14.0|}{|-14.636489|} \approx 0.4 \times 10^{-1}.$$

Alternatively, if we rewrite the formulation of $f(x)$ in nested manner as

$$f(x) = ((x-6)x + 3)x - 0.149.$$

Then the 3-digit chopping arithmetic gives

$$\hat{f}(4.17) = -14.5$$

with relative error

$$\frac{|-14.636489 + 14.5|}{|-14.636489|} \approx 0.93 \times 10^{-2},$$

and 3-digit rounding

$$\tilde{f}(4.17) = -14.6$$

with relative error

$$\frac{|-14.636489 + 14.6|}{|-14.636489|} \approx 0.25 \times 10^{-2}.$$

■

Example 2.7 *Let*

$$\begin{aligned} p(x) &= ((x^3 - 3x^2) + 3x) - 1, \\ q(x) &= ((x - 3)x + 3)x - 1. \end{aligned}$$

Compare the function values at $x = 2.19$.

Sol: Use 3-digit and rounding for $p(2.19)$ and $q(2.19)$.

$$\begin{aligned} \hat{p}(2.19) &= ((2.19^3 - 3 \times 2.19^2) + 3 \times 2.19) - 1 \\ &= ((10.5 - 14.4) + 3 \times 2.19) - 1 \\ &= (-3.9 + 6.57) - 1 \\ &= 2.67 - 1 \\ &= 1.67 \end{aligned}$$

$$\begin{aligned} \hat{q}(2.19) &= ((2.19 - 3) \times 2.19 + 3) \times 2.19 - 1 \\ &= (-0.81 \times 2.19 + 3) \times 2.19 - 1 \\ &= (-1.77 + 3) \times 2.19 - 1 \\ &= 1.23 \times 2.19 - 1 \\ &= 2.69 - 1 \\ &= 1.69 \end{aligned}$$

With more digits, one can have

$$p(2.19) = g(2.19) = 1.685159$$

Hence the absolute errors are

$$|p(2.19) - \hat{p}(2.19)| = 0.015159$$

and

$$|q(2.19) - \hat{q}(2.19)| = 0.004841,$$

respectively. One can observe that the evaluation formula $q(x)$ is better than $p(x)$. ■

In fact, polynomials should always be expressed in nested form before performing an evaluation, since this form minimizes the number of required arithmetic calculations. One way to reduce roundoff error is to reduce the number of error-producing computations.

Example 2.8 *How to evaluate*

$$y = x - \sin x$$

when x is small?

Sol: Since $x \approx \sin x$ for small x, the computation will cause loss of significance. Alternatively, use Taylor series for $\sin x$ so that

$$\begin{aligned} y &= x - \left(x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \cdots\right) \\ &= \frac{x^3}{3!} - \frac{x^5}{5!} + \frac{x^7}{7!} - \frac{x^9}{9!} + \cdots \\ &= \frac{x^3}{6} - \frac{x^5}{6 \times 20} + \frac{x^7}{6 \times 20 \times 42} - \frac{x^9}{6 \times 20 \times 42 \times 72} \cdots \\ &= \frac{x^3}{6}\left(1 - \frac{x^2}{20}\left(1 - \frac{x^2}{42}\left(1 - \frac{x^2}{72}(\cdots)\right)\right)\right) \end{aligned}$$

■

Finally, floating-point arithmetic is not always associative. This can be illustrated with the following example.

Example 2.9 *Performing the calculation with 3-decimal mantissa and chopping arithmetic will result*

$$fl(fl(10^{-3} + 1) - 1) = 0$$

and

$$fl(fl(10^{-3} + fl(1 - 1))) = 10^{-3}.$$

■

2.2.2 Numerical Stability

Another theme that occurs repeatedly in numerical analysis is the distiction between numerical algorithms that are stable and those that are not. Informally speaking, a numerical process is unstable if small errors made at one stage of the process are magnified and propagated in subsequent stages and seriously degrade the accuracy of the overall calculation. Whether a process is stable or unstable should be decided on the basis of relative error.

Example 2.10 *Consider the following recurrence algorithm*

$$\begin{cases} x_0 = 1, \qquad x_1 = \frac{1}{3} \\ x_{n+1} = \frac{13}{3}x_n - \frac{4}{3}x_{n-1} \end{cases}$$

for computing the sequence of $\{x_n = (\frac{1}{3})^n\}$. *This algorithm is unstable.*

Sol: A computer implemention of the recurrence algorithm gives the following result.

n	x_n	n	x_n	n	x_n	n	x_n
0	1.0000000	4	0.0123466	8	0.0003757	12	0.0571502
1	0.3333333	5	0.0041187	9	0.0009437	13	0.2285939
2	0.1111112	6	0.0013857	10	0.0035887	14	0.9143735
3	0.0370373	7	0.0005153	11	0.0142927	15	3.6574934

The error present in x_n is multiplied by $\frac{13}{3}$ in computing x_{n+1}. For example, the error will be propagated with a factor of $\left(\frac{13}{3}\right)^{14}$ in computing x_{15}. Additional roundoff errors in computing $x_2, x_3, \ldots$ may also be propagated and added to that of x_{15}. ■

2.2.3 Conditioning

The words condition and conditioning are used informally to indicate how sensitive the solution of a problem may be to small relative changes in the input data. A problem is ill-conditioned if small changes in the data can produce large changes in the resluts. For certain types of problems, a condition number can be defined. If that number is large, it indicates an ill-conditioned problem. In contrast, if the number is modest, the problem is recognized as a well-conditioned problem.

For example, the condition number for a nonsingular square matrix A is defined as

$$\kappa(A) = \|A\|\|A^{-1}\|,$$

with respect to some matrix norm. The matrix A is said to be ill-conditioned if $\kappa(A)$ is large, and well-conditioned when $\kappa(A)$ is modest. For a general rectangular matrix, the singular values are used to characterize the condition number in 2-norm

$$\kappa_2(A) = \frac{\sigma_{max}(A)}{\sigma_{min}(A)},$$

where $\sigma_{max}(A)$ is the largest singular value and $\sigma_{min}(A)$ the smallest singular value of A.

A well-known ill-conditioned matrix is the Hilbert matrix

$$H_n = [h_{ij}] \in \mathbb{R}^{n\times n}, \qquad \text{where} \qquad h_{ij} = \frac{1}{i+j-1}.$$

In general, ill-conditioning is not easy to detect.

In solving a system of linear equations $Ax = b$ in which A is ill-conditioned, small perturbation in b will cause large perturbation in x.

Example 2.11 *Consider solving the following system of linear equations*

$$\begin{bmatrix} 20514 & 4424 & 987 & 224 \\ 4424 & 987 & 224 & 54 \\ 987 & 224 & 54 & 14 \\ 224 & 54 & 14 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 20514 \\ 4424 \\ 987 \\ 224 \end{bmatrix}.$$

Sol: The numerical solution obtained by using Matlab command $A\backslash b$, where A is the coefficient matrix and b is the right-hand-side vector, is

$$x = \begin{bmatrix} 1.00000000000002 \\ -0.00000000000003 \\ -0.00000000000055 \\ 0.00000000000125 \end{bmatrix}.$$

However, if b_1 is changed from 20514 to 20515, then the numerical solution becomes

$$x = \begin{bmatrix} 0.99672906602257 \\ -0.00528381642514 \\ 0.16978663445968 \\ -0.33974939613341 \end{bmatrix}.$$

One may check that the 2-norm condition number of the coefficient is $\kappa_2(A) \approx 6.1191 \times 10^5$.
■

2.3 Floating-Point Error Analysis

In addition to inaccurate representation of numbers, the arithmetic performed in a computer is not exact. The arithmetic involves manipulating binary digits by various shifting, or logical, operations. We now consider the errors that are produced in the course of elementary arithmetic operations.

Let $\odot$ stand for any one of the four basic arithmetic operators $+$, $-$, $\star$, $\div$. Whenever two machine numbers x and y are to be combined arithmetically, the computer will produce $fl(x \odot y)$ instead of $x \odot y$. In the operation, $x \odot y$ is first correctly formed, normalized, and then rounded to become a machine number. Under this assumption and (2.6), the relative error of $fl(x \odot y)$ satisfies

$$fl(x \odot y) = (x \odot y)(1 + \delta), \quad \delta \le \epsilon_M, \tag{2.8}$$

where ϵ_M is the unit roundoff. But if x, y are not machine numbers, then they must first rounded to floating-point format before the arithmetic operation and the resulting relative error becomes

$$fl(fl(x) \odot fl(y)) = (x(1 + \delta_1) \odot y(1 + \delta_2))(1 + \delta_3), \quad \delta_i \le \epsilon_M, \quad i = 1, 2, 3. \tag{2.9}$$

Thus, the machine unit roundoff gives an upper bound for the relative error in any single basic arithmetic operation.

The analysis (2.8) can be extended to arithmetic operations on three floating-point numbers. For example,

$$\begin{aligned} fl(x(y+z)) &= (x \cdot fl(y+z))(1+\delta_1) \\ &= (x(y+z)(1+\delta_2))(1+\delta_1) \\ &= x(y+z)(1+\delta_1+\delta_2+\delta_1\delta_2) \\ &\approx x(y+z)(1+\delta_1+\delta_2) \\ &= x(y+z)(1+\delta_3) \end{aligned}$$

The generalization to the sum of n floating-point numbers is as follows.

Theorem 2.2 *Let $x_0, x_1, \ldots, x_n$ be positive floating-point numbers. Then the relative error in computing $\sum_{i=0}^{n} x_i$ in the usual way is at most $(1+\epsilon_M)^n - 1 \approx n\epsilon_M$, where ϵ_M is the machine unit roundoff.*

Proof: Let $S_k = x_0 + x_1 + \cdots + x_k$ and $S_k^\star = fl(S_k)$, and be computed in the usual way

$$\begin{cases} S_0 = x_0 \\ S_{j+1} = S_j + x_{j+1} \end{cases} \qquad \text{and} \qquad \begin{cases} S_0^\star = fl(x_0) \\ S_{j+1}^\star = fl(S_j^\star + x_{j+1}) \end{cases}$$

Define

$$\rho_k = \frac{S_k^* - S_k}{S_k} \qquad \text{and} \qquad \delta_k = \frac{S_{k+1}^* - S_{k+1}}{S_k^* + x_{k+1}},$$

where each $|\delta_k| \leq \epsilon_M$. Then

$$\begin{aligned}
\rho_{k+1} &= \frac{S_{k+1}^* - S_{k+1}}{S_{k+1}} \\
&= \frac{(S_k^* + x_{k+1})(1+\delta_k) - (S_k + x_{k+1})}{S_{k+1}} \\
&= \frac{(S_k(1+\rho_k) + x_{k+1})(1+\delta_k) - (S_k + x_{k+1})}{S_{k+1}} \\
&= \frac{(S_k + \rho_k S_k + x_{k+1})(1+\delta_k) - (S_k + x_{k+1})}{S_{k+1}} \\
&= \frac{S_k + \rho_k S_k + x_{k+1} + \delta_k S_k + \delta_k \rho_k S_k + \delta_k x_{k+1} - S_k - x_{k+1}}{S_{k+1}} \\
&= \frac{\rho_k S_k + \delta_k S_k + \delta_k \rho_k S_k \delta_k x_{k+1}}{S_{k+1}} \\
&= \frac{\delta_k (S_k + x_{k+1}) + \rho_k (S_k + \delta_k S_k)}{S_{k+1}} \\
&= \frac{\delta_k S_{k+1} + \rho_k S_k (1+\delta_k)}{S_{k+1}} \\
&= \delta_k + \rho_k \left(\frac{S_k}{S_{k+1}} \right) (1+\delta_k).
\end{aligned}$$

Since x_i are all positive, $S_k < S_{k+1}$, and

$$|\rho_{k+1}| \leq \epsilon_M + |\rho_k|(1+\epsilon_M) = \epsilon_M + \theta|\rho_k|,$$

where $\theta = 1 + \epsilon_M$. Thus

$$\begin{aligned}
|\rho_0| &= 0 \\
|\rho_1| &\leq \epsilon_M \\
|\rho_2| &\leq \epsilon_M + \epsilon_M \theta \\
|\rho_3| &\leq \epsilon_M + \epsilon_M \theta + \epsilon \theta^2 \\
&\vdots
\end{aligned}$$

In general,

$$
\begin{aligned}
|\rho_n| &\le \epsilon_M + \epsilon_M\theta + \epsilon\theta^2 + \cdots + \epsilon\theta^{n-1} \\
&= \epsilon\left(1 + \theta + \theta^2 + \cdots + \theta^{n-1}\right) \\
&= \epsilon\left(\frac{\theta^n - 1}{\theta - 1}\right) \\
&= \epsilon\left(\frac{(1+\epsilon_M)^n - 1}{1 + \epsilon_M - 1}\right) \\
&= (1+\epsilon_M)^n - 1 \\
&= 1 + \binom{n}{1}\epsilon_M + \binom{n}{2}\epsilon_M^2 + \cdots - 1 \\
&\approx \binom{n}{1}\epsilon_M \\
&= n\epsilon_M.
\end{aligned}
$$

■