2 HOW TO OBTAIN AND ESTIMATE ACCURACY IN NUMERICAL CALCULATIONS

2.1. BASIC CONCEPTS IN ERROR ESTIMATION

2.1.1. Introduction

Approximation is a central concept in almost all the uses of mathematics. One must often be satisfied with approximate values of the quantities with which one works. Another type of approximation occurs when one ignores some quantities which are small compared to other quantities. Such approximations are often necessary to insure that the mathematical and numerical treatment of a problem does not become hopelessly complicated.

We shall now introduce some notations, useful in practice, though their definitions are not exact in a mathematical sense:

$a \ll b$ (or $b \gg a$) is read: "a is much smaller than b" (or "b is much greater than a"). What is meant by "much smaller" (or "much greater") depends on the context—among other things, on the desired precision. In a given instance it can be sufficient that $a < \frac{1}{2}b$; in another instance perhaps $a < b/100$ is necessary.

$a \approx b$ is read: "a is approximately equal to b" and means the same as $|a - b| \ll c$, where c is chosen appropriate to the context. We *cannot generally* say, for example, that $10^{-6} \approx 0$.

$a \lesssim b$ (or $b \gtrsim a$) is read: "a is less than or approximately equal to b" and means the same as "$a < b$ or $a \approx b$."

Occasionally we shall have use for the following more precisely defined mathematical concepts:

$f(x) = O(g(x))$ when $x \longrightarrow a$, which means that $|f(x)/g(x)|$ is bounded as $x \longrightarrow a$ (a can be finite, $+\infty$, or $-\infty$).

$f(x) = o(g(x))$ when $x \longrightarrow a$, which means that $\lim_{x \to a} f(x)/g(x) = 0$.

2.1.2. Sources of Error

Numerical results are influenced by many types of errors. Some sources of error are difficult to influence; others can be reduced or even eliminated by, for example, rewriting formulas or making other changes in the computational sequence.

A. Errors in Given Input Data. Input data can be the result of measurements which have been influenced by systematic errors or by temporary disturbances. Round-off errors occur, for example, whenever an irrational number is shortened ("rounded off") to a fixed number of decimals. Round-off errors can also occur when a decimal fraction is converted to the form used in the computer.

B. Round-off Errors During the Computations. If the calculating device which one is using cannot handle numbers which have more than, say, s digits, then the exact product of two s-digit numbers (which contains $2s$ or $2s - 1$ digits) cannot be used in the subsequent calculations; the product must be rounded off. The effect of such roundings can be quite noticeable in an extensive calculation, or in an algorithm which is numerically unstable (defined in Example 1.3.3).

C. Truncation Errors. These are errors committed when a limiting process is truncated (broken off) before one has come to the limiting value. Truncation occurs, for example, when an infinite series is broken off after a finite number of terms, or when a derivative is approximated with a difference quotient (although in this case the term **discretization error** is better). Another example is when a nonlinear function is approximated with a linear function. Observe the distinction between truncation error and round-off error.

D. Simplifications in the Mathematical Model. In most of the applications of mathematics, one makes **idealizations**. In a mechanical problem, for example, one might assume that a string in a pendulum has zero mass. In many other types of problems it is advantageous to consider a given body to be homogeneously filled with matter, instead of being built up of atoms. For a calculation in economics, one might assume that the rate of interest is constant over a given period of time. The effects of such sources of error are usually more difficult to estimate than the types named in *A*, *B*, and *C*.

E. "Human" Errors and Machine Errors. In all numerical work, one must expect that clerical errors, errors in hand calculation, and misunderstandings will occur. One should even be aware that printed tables, etc., may contain errors. When one uses computers, one can expect errors in the program itself, errors in the punched cards, operator errors, and machine errors.

Errors which are purely machine errors are responsible for only a very small part of the strange results which (occasionally with great publicity) are produced by computers year after year. Most of the errors depend on the so-called human factor. As a rule, the effect of this type of error source cannot be analyzed with the help of the theoretical considerations of this chapter! We take up these sources of error in order to emphasize that both the person who carries out a calculation and the person who guides the work of others can plan so that such sources of error are not damaging. One can reduce the risk for such errors by suitable adjustments in working conditions and routines. Stress and tiredness are common causes of such errors.

One should also carefully consider what kind of checks can be made, either in the final result or in certain stages of the work, to prevent the necessity of redoing a whole project for the sake of a small error in an early stage. One can often discover whether calculated values are of the wrong order of magnitude or are not sufficiently regular (see difference checks, Chap. 7). Occasionally one can check the credibility of several results at the same time by checking that certain relations are true. In linear problems, one often has the possibility of sum checks. In physical problems, one can check, for example, to see whether energy is conserved, although because of the error sources *A–D* one cannot expect that it will be exactly conserved. In some situations, it can be best to treat a problem in two independent ways, although one can usually (as intimated above) check a result with less work than this.

2.1.3. Absolute and Relative Errors

Let $\tilde{a}$ be an approximate value for a quantity whose exact value is a. We define:

The **absolute error** in $\tilde{a}$ is $\tilde{a} - a$.

The **relative error** in $\tilde{a}$ is $(\tilde{a} - a)/a$ if $a \neq 0$. The relative error is often given as a percentage—for example, 3 percent relative error means that the relative error is 0.03.

In some books the error is defined with opposite sign to that which we use here. It makes almost no difference which convention one uses, as long as one is consistent. Using our definition, then, $a - \tilde{a}$ is the *correction* which should be added to $\tilde{a}$ to get rid of the error, $\tilde{a} - a$. The correction and the error have, then, the same magnitude but different signs.

It is important to make a distinction between the error, which can be positive or negative, and a positive bound for the magnitude of the error, an **error bound.** We shall have reason to compute error bounds in many situations.

In the above definition of absolute error, a and $\tilde{a}$ need not be real numbers: they can also be vectors or matrices. (If we let $\|\cdot\|$ denote a vector norm (see Sec. 5.5.2), then the magnitude of the absolute and relative errors for the vector $\tilde{a}$ are defined by $\|\tilde{a} - a\|$ and $\|\tilde{a} - a\|/\|a\|$ respectively.)

The notation $a = \tilde{a} \pm \epsilon$ means, in this book, $|\tilde{a} - a| \leq \epsilon$. For example, $a = 0.5876 \pm 0.0014$ means $0.5862 \leq a \leq 0.5890$. In many applications, the same notation as above denotes the "standard error" (see Sec. 2.2.2) or some other measure of deviation of a statistical nature.

2.1.4. Rounding and Chopping

When one gives the *number of digits* in a numerical value one should not include zeros in the beginning of the number, as these zeros only help to denote where the decimal point should be. If one is counting the *number of decimals,* one should of course include leading zeros to the right of the decimal point.

Example

The number 0.00147 is given with three digits but has five decimals. The number 12.34 is given with four digits but has two decimals.

If the magnitude of the error in $\tilde{a}$ does not exceed $\frac{1}{2}\cdot 10^{-t}$, then $\tilde{a}$ is said to have t **correct decimals.** The *digits* in $\tilde{a}$ which occupy positions where the unit is greater than or equal to 10^{-t} are called, then, **significant digits** (any initial zeros are not counted).

Example

0.001234 ± 0.000004 has five correct decimals and three significant digits, while 0.001234 ± 0.000006 has four correct decimals and two significant digits.

The number of correct decimals gives one an idea of the magnitude of the absolute error, while the number of significant digits gives a rough idea of the magnitude of the relative error.

There are **two ways of rounding off** numbers to a given number (t) of decimals. In **chopping,** one simply leaves off all the decimals to the right of the tth. That way of abridging a number is *not recommended* since the error has, systematically, the opposite sign of the number itself. Also, the magnitude of the error can be as large as 10^{-t}. A surprising number of computers use chopping on the results of every arithmetical operation. This usually does not do so much harm, because the number of digits used in the operations is generally far greater than the number of significant digits in the data.

In **rounding** (sometimes called "correct rounding"), one chooses, among the numbers which can be expressed with t decimals, a number which is closest to the given number. Thus if the part of the number which stands to the right of the tth decimal is less than $\frac{1}{2} \cdot 10^{-t}$ in magnitude, then one should leave the tth decimal unchanged. If it is greater than $\frac{1}{2} \cdot 10^{-t}$, then one raises the tth decimal by 1. In the boundary case, when that which stands to the right of the tth decimal is exactly $\frac{1}{2} \cdot 10^{-t}$, one should raise the tth decimal if it is odd or leave it unchanged if it is even. In this way, the error is positive or negative about equally often. Most computers which perform rounding always, in the boundary case mentioned above, raise the number by $\frac{1}{2} \cdot 10^{-t}$ (or the corresponding operation in a base other than 10), because this is easier to realize technically. Whichever convention one chooses in the boundary case, the error in rounding will always lie in the interval $[-\frac{1}{2} \cdot 10^{-t}, \frac{1}{2} \cdot 10^{-t}]$.

Example

Shortening to three decimals.

0.2397	rounds to	0.240	(is chopped to	0.239),
−0.2397	rounds to	−0.240	(is chopped to	−0.239),
0.23750	rounds to	0.238	(is chopped to	0.237),
0.23650	rounds to	0.236	(is chopped to	0.236),
0.23652	rounds to	0.237	(is chopped to	0.236).

Observe that when one rounds off a numerical value one produces an error; thus it is occasionally wise to give more decimals than those which are correct. Take, for example, $a = 0.1237 \pm 0.0004$, which has three correct decimals according to the definition given previously. If one rounds to three decimals, one gets 0.124; here the third decimal is not correct, since the least possible value for a is 0.1233.

One consequence of these rounding conventions is that numerical results which are not followed by any error estimations should often, though not always, be considered as having an uncertainty of $\frac{1}{2}$ unit in the last decimal place.

In presenting numerical results, it is a good habit, if one does not want to go to the difficulty of presenting an error estimate with each result, to give explanatory remarks such as:

"All the digits given are thought to be significant."
"The data has an uncertainty of at most 3 units in the last digit."
"For an ideal two-atomed gas, $c_P/c_V = 1.4$ (exactly)."

REVIEW QUESTIONS

1. Clarify (with examples) the various types of error sources which occur in numerical work.
2. Define absolute error, relative error. What is meant by an *error bound*?

PROBLEM

Give π to four decimals using (a) chopping, (b) rounding.

2.2. PROPAGATION OF ERRORS

2.2.1. Simple Examples of Error Analysis

Example 2.2.1

If $x_1 = 2.31 \pm 0.02$ and $x_2 = 1.42 \pm 0.03$, what is the error bound for $x_1 - x_2$?

The greatest possible value of x_1 is 2.33, and the least possible value for x_2 is 1.39. Thus the greatest possible value for $x_1 - x_2$ is $2.33 - 1.39 = 0.94$. Similarly, the least possible value for $x_1 - x_2$ is $2.29 - 1.45 = 0.84$. Hence, $0.84 \leq x_1 - x_2 \leq 0.94$; that is,

$$x_1 - x_2 = 0.89 \pm 0.05.$$

More generally, if $x_1 = \tilde{x}_1 \pm \epsilon_1$, $x_2 = \tilde{x}_2 \pm \epsilon_2$, we have

$$\begin{aligned}
&\tilde{x}_1 - \epsilon_1 - (\tilde{x}_2 + \epsilon_2) \leq x_1 - x_2 \leq \tilde{x}_1 + \epsilon_1 - (\tilde{x}_2 - \epsilon_2) \\
&\tilde{x}_1 - \tilde{x}_2 - (\epsilon_1 + \epsilon_2) \leq x_1 - x_2 \leq \tilde{x}_1 - \tilde{x}_2 + (\epsilon_1 + \epsilon_2) \\
&x_1 - x_2 = \tilde{x}_1 - \tilde{x}_2 \pm (\epsilon_1 + \epsilon_2).
\end{aligned}$$

A similar calculation gives

$$x_1 + x_2 = \tilde{x}_1 + \tilde{x}_2 \pm (\epsilon_1 + \epsilon_2).$$

Using induction, one can show, for an arbitrary number of terms:

THEOREM 2.2.1

In **addition** *and* **subtraction**, *the bounds for the* **absolute** *error in the result are given by the sum of the bounds for the absolute errors of the operands.*

The error bound given in the above proposition can, for various reasons, be a coarse overestimate of the real error; we shall see this in Examples 2.2.8 and 2.2.11.

According to the definition of relative error (Sec. 2.1.3), we have the following relation between an exact value x, its estimate $\tilde{x}$, and the estimate's *real relative error* r:

$$\tilde{x} = x + xr = x(1 + r).$$

If $\tilde{x}_1$, $\tilde{x}_2$ have relative errors r_1 and r_2, respectively, then

$$\tilde{x}_1\tilde{x}_2 = x_1(1 + r_1)x_2(1 + r_2) = x_1x_2(1 + r_1)(1 + r_2).$$

Thus, the relative error in $\bar{x}_1\bar{x}_2$ is

$$(1 + r_1)(1 + r_2) - 1 = r_1 + r_2 + r_1r_2 \approx r_1 + r_2, \quad \text{if} \quad |r_1| \ll 1, |r_2| \ll 1. \tag{2.2.1}$$

For example, if $r_1 = 0.02$, $r_2 = -0.01$, then the relative error in the product is $0.0098 \approx 0.01$. In the same way, one finds for the relative error in the quotient, x_1/x_2, that:

$$\frac{1 + r_1}{1 + r_2} - 1 = \frac{r_1 - r_2}{1 + r_2} \approx r_1 - r_2, \quad \text{if} \quad |r_1| \ll 1, |r_2| \ll 1. \tag{2.2.2}$$

If the *bounds* (see Sec. 2.1.3) for the relative errors in x_1, x_2 are ρ_1 and ρ_2, respectively, then $\rho_1 + \rho_2$ is the best upper bound for $|r_1 + r_2|$ as well as $|r_1 - r_2|$. Thus:

THEOREM 2.2.2

In **multiplication** *and* **division,** *the bounds for the* **relative** *errors in the operands are added.* (As one can see from the derivation above, this theorem is only approximately valid; see Examples 2.2.5, 2.4.3.)

We note here that the approximations used in Eqs. (2.2.1) and (2.2.2) are very useful in many other situations.

Error analysis is more than just a means for judging the reliability of calculated results; it has an even more important function as a means for **planning a given calculation**—for example, in the choosing of an algorithm (see Example 1.3.3)—and in making certain **decisions** during a calculation. Examples of such decisions are the choice of step length during a numerical integration or, especially in hand calculation, the choice of the number of digits to be used in the various parts of a computation. Increased accuracy is often bought at the price of more time-consuming or more troublesome calculations.

Example 2.2.2

The following type of situation occurs quite often. Assume that a product or a quotient is to be calculated, $y = x_1x_2$ or $y = x_1/x_2$. The quantity x_1 is already known to a certain amount of accuracy and x_2 is to be calculated. How accurately should one compute x_2 (assuming that the work to calculate x_2 grows the more quickly as one demands higher accuracy)?

Since the limit for the relative error in y is equal to the sum of the bounds for the relative errors in x_1 and x_2, there is no use making the relative error in x_2 very much less than the relative error in x_1. To take along *one* more significant digit in x_2 than in x_1 can, however, often be wise.

The error-propagation formulas are also of great interest in the **planning and analysis of scientific experiments.** One can shed some light on certain *questions analogous to the previous example*—for example, to what degree it is advisable to obtain a new apparatus to improve the measurements of a given variable when the measurements of other variables are subject to error as well.

Cancellation. One very common reason for poor accuracy in the result of a calculation is that one has somewhere carried out a subtraction in which the difference between the operands is considerably less than either of the operands. This is called **cancellation of terms**. From Theorem 2.2.1 we see, if we denote the error in x_1 and x_2 by Δx_1 and Δx_2, respectively, that:

$$y = x_1 - x_2 \Rightarrow |\Delta y| \leq |\Delta x_1| + |\Delta x_2| \Rightarrow \left|\frac{\Delta y}{y}\right| \leq \frac{|\Delta x_1| + |\Delta x_2|}{|x_1 - x_2|}.$$

This shows that *there can be very poor relative accuracy in the difference between two nearly equal numbers.* For example, if $x_1 = 0.5764 \pm \frac{1}{2}\cdot 10^{-4}$ and $x_2 = 0.5763 \pm \frac{1}{2}\cdot 10^{-4}$, then $x_1 - x_2 = 0.0001 \pm 0.0001$—the error bound is just as large as the estimate of the result.

One should try to avoid cancellation by appropriate **rewriting of formulas**, or by other changes in the algorithm.

Example 2.2.3

The quadratic equation $x^2 - 56x + 1 = 0$ has the roots

$$x_1 = 28 - \sqrt{783} \approx 28 - 27.982 = 0.018 \pm \tfrac{1}{2}\cdot 10^{-3},$$
$$x_2 = 28 + \sqrt{783} = 55.982 \pm \tfrac{1}{2}\cdot 10^{-3}.$$

In spite of the fact that the square root is given to five digits, we get only two significant digits in x_1, while the relative error in x_2 is less than 10^{-5}. *It is worthwhile to notice that the subtraction itself, in the calculation of x_1, has been carried out exactly. The subtraction only gives an indication of the unhappy consequence of a loss of information in the past* due to the rounding of one of the operands.

Since $x_1x_2 = 1$, one can instead use $x_1 = 1/55.982 = 0.01786288$ with a relative error of less than 10^{-5}. Thus $x_1 = 0.0178629 \pm 0.0000002$; the same estimate of the square root gave x_1 to five significant digits instead of two because we also used information from the quadratic equation which was the source of the numerical data.

More generally, if $|\delta| \ll x$, then one should rewrite

$$\sqrt{x+\delta} - \sqrt{x} = \frac{x + \delta - x}{\sqrt{x+\delta} + \sqrt{x}} = \frac{\delta}{\sqrt{x+\delta} + \sqrt{x}}.$$

There are other exact ways of rewriting formulas which are as useful as the above; for example,

$$\cos(x + \delta) - \cos x = -2 \sin(\tfrac{1}{2}\delta) \sin(x + \tfrac{1}{2}\delta).$$

If one cannot find an exact way of rewriting a given expression of the form $f(x+\delta) - f(x)$, it is often advantageous to use one or more terms in the Taylor series

$$f(x+\delta) - f(x) = f'(x)\delta + \tfrac{1}{2}f''(x)\delta^2 + \dots .$$

In Example 2.2.3 we got a warning that cancellation would occur, since x_1 was found as the difference between two nearly equal numbers each of which was, relatively, much larger than the difference itself. In practice, one does not always get such a warning, for two reasons: first, in using a computer one has no direct contact with the individual steps of a calculation; secondly, cancellation can be spread over a great number of operations, as in the following example.

Example 2.2.4

Set $y_0 = 28$ and define y_n, for $n = 1, 2, \dots, 100$, by the recursion formula:

$$y_n = y_{n-1} - \tfrac{1}{100}\sqrt{783}.$$

As previously, we use the approximate value 27.982 for the square root. We then compute with five decimals in each operation in order to make the effect of further round-off errors negligible. We get the same bad value for y_{100} that we got for x_1 in the previous example. Of all the subtractions, only the last would lead one to suspect cancellation, $y_{100} = 0.29782 - 0.27982 = 0.01800$, but this result in itself gives one no reason to suspect that only two digits are significant. (With four significant digits, the result is 0.01786.)

2.2.2. The General Formula for Error Propagation; Maximum Error and Standard Error

Consider a function of one variable, $y(x)$, and suppose we want to estimate a bound for the magnitude of the error $\Delta y = y(\tilde{x}) - y(x^{(0)})$ in $y(x^{(0)})$, where $\tilde{x}$ is an approximate value for $x^{(0)}$. A natural way (see Fig. 2.2.1) to approximate Δy is with the differential of y. In this way, one locally approxi-

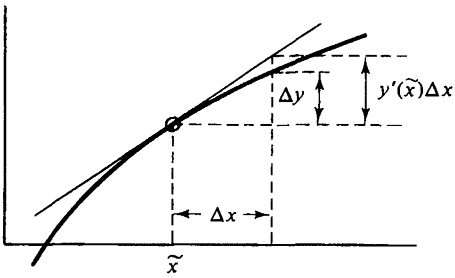


Fig. 2.2.1

mates $y(x)$ with a linear function. The quantity $|y'(x)|$ can be interpreted as a measure of **the sensitivity of $y(x)$ for disturbances in the argument** x.

Suppose that one knows approximate values $\tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_n$ for certain variables $x_1, x_2, \ldots, x_n$, whose exact values are $x_1^{(0)}, x_2^{(0)}, \ldots, x_n^{(0)}$. Let y be a function of the variables $x_1, x_2, \ldots, x_n$. We shall now estimate the error in the approximate value $y(\tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_n)$. This general problem is of great interest in experimental science and other areas where empirical data are put into mathematical models.

Introduce the vector notation

$$\tilde{x} = (\tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_n), \qquad x_0 = (x_1^{(0)}, x_2^{(0)}, \ldots, x_n^{(0)}),$$

and set

$$\Delta x_i = \tilde{x}_i - x_i^{(0)}, \qquad \Delta y = y(\tilde{x}) - y(x_0).$$

THEOREM 2.2.3 **General Error-Propagation Formula**

$$\Delta y \approx \sum_{i=1}^{n} \frac{\partial y}{\partial x_i}(\tilde{x}) \cdot \Delta x_i, \qquad \textit{hence} \quad |\Delta y| \lesssim \sum_{i=1}^{n} \left|\frac{\partial y}{\partial x_i}(\tilde{x})\right| \cdot |\Delta x_i|. \qquad (2.2.3)$$

The formula is derived as follows: one moves from x_0 to $\tilde{x}$ in n steps, by changing one coordinate at a time. Thus for $i = 1, 2, \ldots, n-1$, set

$$x_i = (\tilde{x}_1, \ldots, \tilde{x}_i, x_{i+1}^{(0)}, \ldots, x_n^{(0)}), \qquad x_n = \tilde{x}.$$

In moving from x_{i-1} to x_i, the ith coordinate is changed from $x_i^{(0)}$ to $\tilde{x}_i$, while the rest of the coordinates are left unchanged. Then, by the mean-value theorem, we have, for some point ξ_i between x_{i-1} and x_i, that

$$y(x_i) - y(x_{i-1}) = \frac{\partial y}{\partial x_i}(\xi_i) \cdot (\tilde{x}_i - \tilde{x}_i^{(0)}) \approx \frac{\partial y}{\partial x_i}(\tilde{x}) \cdot \Delta x_i.$$

The theorem now follows, since

$$\Delta y = y(x_n) - y(x_0) = \sum_{i=1}^{n} (y(x_i) - y(x_{i-1})).$$

As one can see from Theorem 2.2.3, Δy was approximated with the *total differential* of y. This means that, in a small neighborhood of $\tilde{x}$, containing x_0, one approximates y with a linear function.

In the practical use of the formula (2.2.3), one calculates $|\partial y / \partial x_i|$ at the point $\tilde{x}$, and then adds a certain marginal amount for safety. However, in order to get a strict error bound for $y(\tilde{x})$, one should really use the maximum absolute values of the derivatives in a neighborhood of $\tilde{x}$. In most practical situations, it suffices to use the values of the derivatives at $\tilde{x}$ and then to add a 5 to 10 percent margin of safety. But if the Δx_i are large or if the derivatives have a large relative variation in the neighborhood of $\tilde{x}$, then one must use the maximal values. (The latter situation occurs, for example, in a neighborhood of an extremal point for the function y.)

Example 2.2.5

The relative error in a quantity is approximately equal to the absolute error in its natural logarithm, since

$$\Delta(\ln y) \approx d(\ln y) = \frac{dy}{y} \approx \frac{\Delta y}{y}.$$

Using an exact formula, one has $\Delta(\ln y) = \ln(y+\Delta y) - \ln(y) = \ln(1+\Delta y/y)$. A comparison follows:

$\frac{\Delta y}{y} = 0.001$	0.01	0.1	0.2	-0.2	-0.1
$\Delta(\ln y) = 0.00100$	0.00995	0.095	0.182	-0.223	-0.105

For logarithms to base 10 ("common logarithms") one has

$$\Delta(\log_{10} y) = \log_{10} e \cdot \Delta(\ln y) \approx 0.434 \frac{\Delta y}{y}.$$

Example 2.2.6

Compute $y = x_1^2 - x_2$, for $x_1 = 1.03 \pm 0.01$, $x_2 = 0.45 \pm 0.01$.

$$\left|\frac{\partial y}{\partial x_1}\right| = |2x_1| \leq 2.1, \qquad \left|\frac{\partial y}{\partial x_2}\right| = |-1| = 1,$$

$$|\Delta y| \leq 2.1 \cdot 0.01 + 1 \cdot 0.01 = 0.031.$$

We find $y = 1.061 - 0.450 \pm 0.032 = 0.611 \pm 0.032$; the error bound has been raised 0.001 because of the rounding in the calculation of x_1^2. If one rounded the results to two decimals, one would write $y = 0.61 \pm 0.04$.

We shall now give a generalization of Theorem 2.2.2:

THEOREM 2.2.4

If $y = x_1^{m_1} x_2^{m_2} x_3^{m_3} \ldots x_n^{m_n}$, *and* $x_i \neq 0$, *then*

$$\left|\frac{\Delta y}{y}\right| \lesssim \sum_{i=1}^{n} |m_i| \left|\frac{\Delta x_i}{x_i}\right|.$$

Proof. $\ln y = m_1 \ln x_1 + m_2 \ln x_2 + \ldots + m_n \ln x_n$.
Differentiate! We get

$$\frac{1}{y}\frac{\partial y}{\partial x_i} = \frac{m_i}{x_i},$$

and the theorem is proved.

It is very seldom, in practice, that one is asked to give mathematically guaranteed error bounds, but it occasionally happens. More often, it is satisfactory to give an estimate of the *order of magnitude* of the anticipated error. The bound for $|\Delta y|$ obtained, for example, with Theorem 2.2.3 covers the worst possible cases, where the sources of error Δx_i contribute with the same sign and magnitudes equal to the error bounds for the individual variables. For this reason, the bound for $|\Delta y|$ thus obtained is called the **maximal error.** *In practice, the trouble with formula* (2.2.3) *is that it gives bounds which are too coarse.*

As a complement to the maximal error, which is often much too pessimistic when the number of variables is large, one uses the **standard error.** The standard error of an estimate of a given quantity is the same as the *standard deviation of its sampling distribution.* The theory of standard error is based on probability theory and will not be treated in detail here. However, we give without proof the following theorem, which a reader familiar with probability theory can derive using the first of the formulas (2.2.3):

THEOREM 2.2.5

Assume that the errors $\Delta x_1, \Delta x_2, \ldots, \Delta x_n$ are independent random variables with mean zero and standard deviations $\epsilon_1, \epsilon_2, \ldots, \epsilon_n$. Then the standard error ϵ for $y = y(x_1, x_2, \ldots, x_n)$ is given by the formula:

$$\epsilon \approx \left(\sum_{i=1}^{n} \left(\frac{\partial y}{\partial x_i}\right)^2 \epsilon_i^2\right)^{1/2}$$

Example 2.2.7

For the problem in Example 2.2.6 we get

$$\epsilon^2 \approx (2.1)^2 \cdot 10^{-4} + 1 \cdot 10^{-4} = 5.41 \cdot 10^{-4}.$$

Thus the standard error $\epsilon \approx 0.024$.

Example 2.2.8. *Maximum Error and Standard Error for Sums*

From the derivation of Theorem 2.2.1 it follows that:

$$y = x_1 + x_2 + \ldots + x_n \Rightarrow \Delta y = \Delta x_1 + \Delta x_2 + \ldots + \Delta x_n.$$

If each x_i has error bound δ, then the maximal error for y is $n\delta$. Thus, *the maximal error grows proportionally to n.* If n is large—for example, $n = 1{,}000$—then it is in fact highly improbable that the real error will be anywhere near $n\delta$, since that bound is attained only when every Δx_i has the same sign and the same maximal magnitude. Observe, though, that if one is adding positive numbers, each of which has been abridged to t decimals by "chopping," then each Δx_i has the same sign and a magnitude which is on the average $\frac{1}{2}\delta$, where $\delta = 10^{-t}$. Thus, using the above method (not to be recommended) for abridging the x_i, the real error is often about 500δ.

If the numbers are rounded (instead of chopped), and *if one can assume that the errors in the various terms are stochastically independent* with standard deviation ϵ, then the standard error in y becomes (using Theorem 2.2.5)

$$(\epsilon^2 + \epsilon^2 + \epsilon^2 + \ldots + \epsilon^2)^{1/2} = \epsilon \cdot \sqrt{n}.$$

Thus the growth of the standard error of the sum of n terms is only proportional to $\sqrt{n}$.

If $n \gg 1$, then the error in y is, under the assumptions made above, approximately **normally distributed** with standard deviation $\sigma = \epsilon \cdot \sqrt{n}$. This distribution is illustrated in Fig. 2.2.2; the curve shown there is also called the Gauss curve. The assumption that the error is normally distributed with standard deviation σ means, for example, that the statement "the magnitude of the error is less than 2σ" (see the shaded area of Fig. 2.2.2) is false in about only 5 percent of all cases (the clear areas under the curve). More generally: *the assertion that the magnitude of the error is less than σ, 2σ, and 3σ, respectively, is false in about (respectively) 32 percent, 5 percent, and 0.27 percent of all cases.*

The assumption that the errors are normally distributed is justified in many computational situations and scientific experiments where the error can be considered to have arisen from the addition of a large number of *independent* error sources of about the same order of magnitude.

One can show that if the individual terms have a uniform probability distribution in the interval $[-\frac{1}{2}\delta, \frac{1}{2}\delta]$, then the standard deviation of an individual term is $\delta/\sqrt{12}$. In only 5 percent of all cases, then, is the error in the sum of 1,000 terms greater than $2 \cdot \delta \cdot \sqrt{1000/12} \approx 18\delta$. This example shows that rounding can be far superior to chopping when a statistical interpretation (especially, the assumption of independence) can be given to the *principal sources of error.* Observe that, in the above, we have only considered the propagation of the errors which were present in the original data, and have ignored the effect of possible round-off errors in the additions themselves.

In science and technology, one generally is careful to discriminate between **systematic error** and **random error**. A systematic error can, for example, be produced by insufficiencies in the construction of an instrument; such an error is the same in each trial. Random errors depend on variations in the experimental environment which cannot be controlled; then the formula

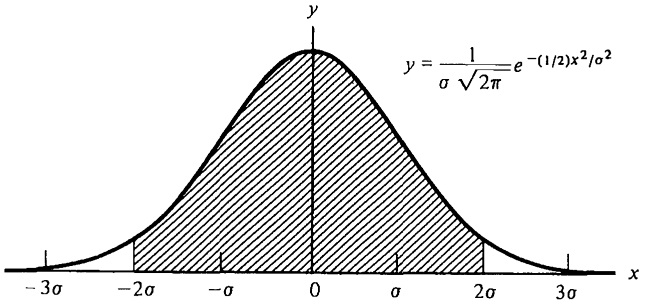


Fig. 2.2.2

for standard error is used. For systematic errors, however, the formula for maximal error (2.2.3) should be used.

2.2.3. On the Practical Application of Error Estimation

In hand calculation or calculation with a desk calculator, it is much easier to work with few digits than with many. On most computers, however, no simplification is gained if an operand contains fewer digits. (Many computers do not even make any shortcuts when one of the operands is zero.) An essential question, then, in hand calculation, is: how many decimals or how many digits should one use? On a computer, one often works either with a constant number of digits (computation with "floating" decimal or binary point) or else with a constant number of decimal or binary digits in the fractional part of a number (computation with "fixed" decimal or binary point). In hand computation, one can (and should) have a more flexible adaptation to the given situation.

Example 2.2.9

Compute $\sum_{n=1}^{6} n^{1/2}\pi^{-2n}$ with an error which is less than 10^{-5}. It is appropriate to compute the terms to six *decimals* (where the last decimal can have error 1). One can adapt the number of *digits* in $n^{1/2}$ and π^{-2n} to the number of digits which is needed in each term (compare Example 2.2.3).

n	π^{-2n}	$n^{1/2}$	Term
1	0.101321(2)	1	0.101321(2) π^{-2}
2	0.010266	1.4142(1)	0.014518
3	0.001040	1.732(0)	0.001801(3)
4	0.0001053	2	0.000211
5	0.000011	2.2	0.000024
6	0.0000011	2.4	0.000003
			$\sum = 0.117878 \pm 5\cdot10^{-6} \approx 0.11788$

The digits shown within parentheses have not been used, but they show that the error is low. In the computation of π^{-2n} one can use the (often convenient) formula

$$(1+\epsilon_1)(1+\epsilon_2) = 1 + (\epsilon_1+\epsilon_2) + \epsilon_1\epsilon_2,$$

where $\epsilon_1\epsilon_2$ can be computed to low accuracy or, in some cases, be completely neglected. As can be seen in the table, the terms of the series decrease so rapidly that the result given (including the error estimate) is valid for the corresponding infinite series. (It can be shown that one gains a decimal for each term; see Example 3.1.3) In the above example, we were unnecessarily stingy with decimals and terms; error analysis is often easier if one works with a slight surplus of digits and terms.

In the application of mathematics, there often occur situations where one "sees" (or is in some other way convinced) that a certain quantity can be ignored, but one cannot, without difficulty, give a strict proof of the fact. Of course, if the proof looks easy, then it is worthwhile, for the added security, to try to prove the fact. On the other hand, numerical methods are often used just because the problem is too difficult for analytic treatment. It can then be wise to drop the demand for a proof. Mathematical clarity is a good resource, but the concomitant demand for strictness can have a crippling effect. Small gaps in the argumentation are natural elements in the mathematical treatment of a difficult problem, but they should be openly presented. The more important and general the problem to be treated is, the more relevant becomes the demand for mathematical stringency. Thus some research in numerical analysis is concerned with developing methods which enable a computer to give strict error estimates for computed results in a wide range of typical problems.

A decision to forgo a proof should, though, be based on experience. One should know that the situation, in all its essential elements, is analogous to a simpler situation where a proof can be given. One should also keep a certain margin of safety; for example, one might use more terms or more digits than one anticipates is necessary.

The criteria which one adopts should not be too coarse; this comment is especially relevant in the case of slowly converging processes. For example, it is far from always true that the magnitude of the remainder of a series is less than the most recently used term, or that decimals which remain unchanged in several consecutive terms of a sequence are correct.

Example 2.2.10

In the series

$$\sum_{n=1}^{\infty} \frac{1}{n(n+1)}$$

the thousandth term is less than 10^{-6}, but the remainder is one thousand times as large, since

$$\sum_{n=1,001}^{\infty} \frac{1}{n(n+1)} = \sum_{1,001}^{\infty} \left(\frac{1}{n} - \frac{1}{n+1}\right)$$
$$= \frac{1}{1,001} - \frac{1}{1,002} + \frac{1}{1,002} - \frac{1}{1,003} + \cdots = \frac{1}{1,001}.$$

The sum of the 999, 1,000, 1,001, ..., 1,008 first terms is, to six decimals,

0.999000, 0.999001, 0.999002, ..., 0.999009.

The fifth decimal is unchanged in ten consecutive partial sums. In spite of this, the error is one unit in the third decimal.

Actually, one should, when one decides how many digits to take along in a quantity to be used later in the computation, give consideration to the entire context of the computations. It can in fact occur that the errors in many operands depend upon each other in such a way that they cancel each other. Such **cancellation of error** (a completely different phenomenon from the previously mentioned (p. 28) "cancellation of terms") is most common in larger problems, but will be illustrated here with a simple example.

Example 2.2.11

Compute to full accuracy $y = z_1 + z_2$, where

$$z_1 = (x^2 + 1)^{1/2}, \qquad z_2 = 200 - x, \qquad x = 100 \pm 1.$$

One finds

$$\frac{dz_1}{dx} = x(x^2 + 1)^{-1/2}, \qquad \frac{dz_2}{dx} = -1, \qquad z_1 = 100 \pm 1, \qquad z_2 = 100 \pm 1.$$

It is tempting to be satisfied with the result $y = 200 \pm 2$. But the errors in z_1 and z_2 cancel each other, since

$$\frac{\Delta y}{\Delta x} \approx \frac{dy}{dx} = \frac{dz_1}{dx} + \frac{dz_2}{dx} = x(x^2 + 1)^{-1/2} - 1$$
$$= \frac{x - \sqrt{x^2 + 1}}{\sqrt{x^2 + 1}} = \frac{x^2 - (x^2 + 1)}{\sqrt{x^2 + 1}(x + \sqrt{x^2 + 1})} \approx \frac{-1}{2x^2}.$$

Thus $|\Delta y| \lesssim \frac{1}{2} \cdot 10^{-4}$. z_1, and z_2 should then be computed to four decimals even though the last integer digits are uncertain! The result:

$$y = 200.0050 \pm \tfrac{1}{2} \cdot 10^{-4}.$$

In the above example one could (and should) avoid the calculation of the square root to many digits. By rewriting the expression for y,

$$y = (\sqrt{x^2 + 1} - x) + 200 = \frac{x^2 + 1 - x^2}{\sqrt{x^2 + 1} + x} + 200$$
$$= \frac{1}{x + \sqrt{x^2 + 1}} + 200,$$

one obtains the result even with low accuracy in the square root. In larger problems, such a cancellation of error can occur even though one cannot easily give a way to rewrite the expressions involved. The authors have seen examples where the final result was a sum of seven terms, and it was obtained correctly to eight decimals even though the terms (which were complicated functions of the solutions to a system of nonlinear equations with fourteen unknowns) had errors already in the second decimal!

2.2.4. The Use of Experimental Perturbations

In larger calculational problems, the relations between input data and output data are so complicated that it is diffcult to directly apply the general formulas for the propagation of error. One should then investigate the sensitivity of the output data for errors in the input data by means of an *experimental perturbational calculation:* one performs the calculations many times with perturbed input data and studies the relation between the changes (perturbations) in the input data and the changes in the output data.

Important data, such as the step length in a numerical integration or the parameter which determines when an iterative process is to be broken off, should be varied with all the other data left unchanged. If one can easily *vary the precision of the machine* in the arithmetic operations one can get an idea of the influence of *round-off errors.* It is generally not necessary to make a perturbational calculation for each and every input data component; one can instead *perturb many input data simultaneously*—for example, by using random numbers.

Such a perturbational calculation often gives not only an error estimate, but also greater insight into the problem. Occasionally, it can be difficult to interpret the perturbational data correctly, since the disturbances in the output data depend not only on the mathematical problem but also on the choice of numerical method and the details in the design of the algorithm. The round-off errors during the computation are not the same for the perturbed and the unperturbed problem. Thus if the output data reacts more sensitively than one had anticipated, it can be difficult to immediately point out the source of the error. It can then be profitable to plan a series of perturbation experiments with the help of which one can separate the effects of the various sources of error. If the dominant source of error is the method or the algorithm, then one should try another method or another algorithm.

It is beyond the scope of this book to give further comments on the planning of such experiments; imagination and the general insights regarding error analysis which this chapter is meant to give play a large role. Even in the special literature, the discussion of the planning of such numerical experiments is surprisingly meager.

2.2.5. Automatic Control of Accuracy

Efforts have been made to design the computational unit of a computer so that it gives, in every arithmetic operation, only those digits of the result which are judged to be significant (possibly with a fixed number of extra digits), so-called *unnormalized floating arithmetic.* This method reveals poor construction in algorithms, but in many other cases it gives a significant and unnecessary loss of accuracy. The mechanization of the rules which a knowledgeable and routined person uses for control of accuracy in hand calculation is not as free from problems as one might expect. As a complement to arithmetical operations of conventional type, the above type of arithmetic is of some interest. At present, it has only been used in a preliminary and experimental way on computers, and it is doubtful that its use will be much wider.

Another effort to automate the control of accuracy (even truncation error) is **interval analysis**, developed by R. E. Moore. Interval analysis is partly an automatization of calculation with maximal error bounds. The ordinary computational unit of the computer is used. The kernel of the method consists of programs for the four operations of arithmetic, with intervals as input data and output data. Let $I_1 = [a_1, \ b_1]$, and $I_2 = [a_2, \ b_2]$. Then, for example, $I_1 - I_2$ is defined as the shortest interval which contains all the numbers of the form $x_1 - x_2$, where $x_1 \in I_1$, $x_2 \in I_2$. Thus $I_1 - I_2 \doteq [a_1 - b_2, b_1 - a_2]$. If rounding off is needed, then the lower bound is always lowered and the upper bound is always raised. Interval analysis has been used successfully in, among other things, the numerical treatment of certain ordinary differential equations and giving strict error bounds for the solutions to systems of linear equations. Interval analysis often requires a refined design in the given algorithm in order to prevent the bounds for the intervals from becoming unacceptably coarse.

For both interval analysis and unnormalized floating arithmetic, the error bounds can be unnecessarily large when *cancellation between errors* occurs, as in Example 2.2.11.

Example 2.2.12

We shall treat the Example 1.3.3, Algorithm 2, using interval analysis. There it was pointed out that $0 < y_{10} < \frac{1}{60}$. Thus

$$y_{10} \in I_{10} = [0, 0.0167],$$

$$y_9 \in I_9 = \frac{1}{50} - \frac{I_{10}}{5} \subset [0.0200 - 0.0034, 0.0200] = [0.0166, 0.0200],$$

$$\begin{aligned} y_8 \in I_8 = \frac{1}{45} - \frac{I_9}{5} &\subset [0.0222 - 0.0040, 0.0223 - 0.0033] \\ &= [0.0182, 0.0190], \text{ etc.} \end{aligned}$$

Example 2.2.13

The recursion formulas

$$x_{n+1} = 2^{-1/2}(x_n - y_n)$$
$$y_{n+1} = 2^{-1/2}(x_n + y_n),$$

mean a series of 45-degree rotations in the xy-plane (see Fig. 2.4.5). By a two-dimensional interval one means a rectangle whose sides are parallel to the coordinate axes.

If (x_0, y_0) is given as some interval $|x_0 - 1| \leq \epsilon$, $|y_0| \leq \epsilon$ (see the dashed square, in the leftmost portion of Fig. 2.4.5), then (x_n, y_n) will, with *exact* performance of the transformations, also be a square with side 2ϵ, for all n (see the other squares in Fig. 2.4.5). If the computations are made using interval analysis, rectangles with sides parallel to the coordinate axes will, in each step, be circumscribed about the exact image of the interval one had in the previous step. Thus the interval is multiplied by $\sqrt{2}$ in each step. After forty steps, for example, the interval has been multiplied by more than 10^6 (since $2^{20} > 10^6$).

If one used, for example, polar coordinates instead of rectangular, there would not have been any difficulties of the above type. This illustrates that one occasionally, with the use of interval analysis, needs to consider the formulation of the algorithm in order to avoid unrealistic overestimates of the error.

REVIEW QUESTIONS

1. In the commentary to Theorem 2.2.1 (on the error bounds for addition and subtraction), it is mentioned that the bounds given there can "for various reasons" be a coarse overestimate of the real error. Give (preferably with examples) two such reasons.
2. Explain (and give an example of) what is meant by "cancellation of terms."
3. Explain the terms "maximum error" and "standard error." What statistical assumption is made about the terms of a sum in calculating the standard error in the sum due to round-off?

PROBLEMS

1. Compute, to three correct decimals:
   (a) $1.3134 \cdot \pi$; (b) $0.3761 \cdot e$; (c) $\pi \cdot e$.
   (Analyze first how many decimals must be taken along in each of the terms.)
2. Determine the maximum error and the relative error in the following results where $x = 2.00$, $y = 3.00$, and $z = 4.00$ are correctly rounded.
   (a) $f = 3x + y - z$; (b) $f = x \cdot \dfrac{y}{z}$; (c) $f = x \cdot \sin \dfrac{y}{40}$.
3. (a) Determine the maximum error for $y = x_1 x_2^2 / \sqrt{x_3}$ where $x_1 = 2.0 \pm 0.1$, $x_2 = 3.0 \pm 0.2$, $x_3 = 1.0 \pm 0.1$. Which of the variables contributes most to the error in y?
   (b) Compute the standard error using the same data as in (a), assuming that the error estimates for the x_i indicate standard deviations.
4. $$\begin{cases} 3x + ay = 10 \\ 5x + by = 20 \end{cases}$$
   $$a = 2.100 \pm 5 \cdot 10^{-4}$$
   $$b = 3.300 \pm 5 \cdot 10^{-4}.$$
   How accurately can $x + y$ be determined?
5. One wishes to compute the expression
   $$f = (\sqrt{2} - 1)^6, \qquad (1)$$

using the approximate value 1.4 for $\sqrt{2}$. One can then choose between substituting the approximate value in (1) or else in one of the following equivalent expressions:

$$\frac{1}{(\sqrt{2}+1)^6} \tag{2}$$

$$(3-2\sqrt{2})^3 \tag{3}$$

$$\frac{1}{(3+2\sqrt{2})^3} \tag{4}$$

$$99-70\sqrt{2} \tag{5}$$

$$\frac{1}{99+70\sqrt{2}} \tag{6}$$

Which alternative gives the best result?

6. The expression $\ln(x-\sqrt{x^2-1})$ is to be computed for $x = 30$. If the square root is obtained from a six-place table, how large does the error in the logarithm become? Give an expression which is mathematically equivalent to the above expression but better numerically, and compute the error in the logarithm.

7. One is making observations of a satellite in order to determine its speed. At the first observation, $R = 30{,}000 \pm 10$ km. Five seconds later, the distance has increased by $r = 125.0 \pm 0.5$ km and the change in angle was $\varphi = 0.00750 \pm 0.00002$ radians. What is the speed of the satellite, assuming that it moves in a straight line and with constant speed in the given time interval?

8. One has measured two sides and the included angle of a triangle to be $a = 100.0 \pm 0.1$, $b = 101.0 \pm 0.1$, and the angle $c = 1.00° \pm 0.01°$.
   (a) How accurately is it possible to give the third side c?
   (b) How accurately does one get c if one uses the cosine theorem and the value $\cos 1° = 0.9998$ from a four-place table?
   (c) Derive a formula with which it is possible to compute c to full accuracy with the help of a four-place table.

   Note: As a *convention* for hand computation, we define *full accuracy* to mean

$$|R_C + R_T| \leq 0.2|R_X|,$$

   where
   R_X denotes a bound for the absolute error due to uncertainty in the input data; it can occasionally be separated into R_{XX}, uncertainty in the argument, and R_{XF}, uncertainty in the values of the function;
   R_C denotes a bound for the absolute error from round-off errors made during the calculation; and
   R_T denotes truncation error (defined in Sec. 2.1.2).
   (In this problem, $R_T = 0$.)

9. In the statistical treatment of data, one often needs to compute the quantities

$$\bar{x} = n^{-1}\sum_{i=1}^{n} x_i, \qquad s^2 = n^{-1}\sum_{i=1}^{n}(x_i-\bar{x})^2.$$

(*Note:* If the numbers x_i are the results of statistically independent measurements of a quantity with expected value m, then $\bar{x}$ is an estimate of m, whose standard error is estimated by $(n-1)^{-1/2}s$.)

(a) Show that $s^2 = n^{-1}\sum_{i=1}^{n} x_i^2 - \bar{x}^2$.

(b) Let α be an arbitrary number and set $x_i' = x_i - \alpha$. (α is sometimes called a provisional mean. In practice, it should be chosen to be of the same order of magnitude as the x_i.) Show that

$$\bar{x} = \alpha + n^{-1}\sum_{i=1}^{n} x_i', \qquad s^2 = n^{-1}\sum_{i=1}^{n} (x_i')^2 - (\bar{x}-\alpha)^2.$$

(c) In sixteen measurements of a quantity x one got the following results:

i	x_i	i	x_i	i	x_i	i	x_i
1	546.85	5	546.81	9	546.96	13	546.84
2	546.79	6	546.82	10	546.94	14	546.86
3	546.82	7	546.88	11	546.84	15	546.84
4	546.78	8	546.89	12	546.82	16	546.84

Take $\alpha = 546.85$. Make a table with one column for $10^2 x_i'$ and one column for $(10^2 x_i')^2$, where the x_i' are defined according to (b). Compute $\bar{x}$ and s^2, s^2 to two significant digits.

(d) In the computations in (c), one never needed more than three digits. If one uses the formulas in (a), how many digits must one have in x_i^2 in order to get two significant digits in s^2? If one uses five digits throughout the computations, why is the cancellation with the use of the formulas in (a) more fatal than the cancellation in the subtraction $x_i' = x_i - \alpha$? (One can even get negative values for s^2!)

(e) Discuss how important (or unimportant) the choice of a good provisional mean is. Take into consideration quick estimates made without a calculating machine, work with a desk calculator, and calculation with a computer!

(f) What advantage do formulas of type (a) or (b) have on a computer as opposed to direct calculation of s according to definition, assuming that the amount of data is so great that it cannot be stored in the central memory?

(g) Set $\bar{y} = n^{-1}\sum_{i=1}^{n} y_i$, $\quad y_i' = y_i - \beta$. Derive a formula for computing $n^{-1}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})$, of the same type as the formula in (b).

10. Continue the calculations in Example 2.2.12.

11. One has an algorithm for computing the integral

$$I(a, b) = \int_0^1 \frac{e^{-bx}}{a + x^2}\, dx.$$

The physical quantities x and y have been measured to be

$$x = 0.400 \pm 0.003, \qquad y = 0.340 \pm 0.005.$$

Using the algorithm for various values of a and b one obtained:

a	b	I
0.39	0.34	1.425032
0.40	0.32	1.408845
0.40	0.34	1.398464
0.40	0.36	1.388198
0.41	0.34	1.372950

How large is the uncertainty in $I(x, y)$?

12. Show that if the errors in the quantities x_i, $i = 1, 2, \ldots, n$ are independent random variables with standard deviation ϵ, then the mean

$$\bar{x} = \frac{x_1 + x_2 + \ldots + x_n}{n}$$

has standard error $\epsilon/\sqrt{n}$.

2.3. NUMBER SYSTEMS; FLOATING AND FIXED REPRESENTATION

2.3.1. The Position System

In order to represent numbers, we use in daily life a **position system** with base 10 (the decimal system). Thus for the numbers we use ten different characters, and the magnitude with which the digit a contributes to a number's value depends on the digit's position in the number in such a way that, if the digit stands n steps to the left of the decimal point, the value contributed is $a \cdot 10^{n-1}$, or if it stands n steps to the right of the decimal point, the value contributed is $a \cdot 10^{-n}$. The sequence of digits 4711.303 means, then,

$$4 \cdot 10^3 + 7 \cdot 10^2 + 1 \cdot 10^1 + 1 \cdot 10^0 + 3 \cdot 10^{-1} + 0 \cdot 10^{-2} + 3 \cdot 10^{-3}.$$

Every real number has a unique representation in the above way, except for the possibility of infinite sequences of nines—for example, the infinite decimal fraction 0.3199999 . . . represents the same number as 0.32.

One can very well consider other position systems with base different from 10. Any natural number $B \geq 2$ can be used as base. One can show that every positive real number has (with exceptions analogous to the nines-sequences mentioned above) a unique representation of the form

$$a_n B^n + a_{n-1} B^{n-1} + \ldots + a_1 B + a_0 + a_{-1} B^{-1} + a_{-2} B^{-2} + \ldots,$$

where the coefficients a_i are the "digits" in the system with base B—that is, positive integers a_i such that $0 \leq a_i \leq B - 1$.

One of the greatest advantages of the position system is that one can give simple, general rules for the arithmetical operations. The smaller the base is, the simpler these rules become. For this reason, among others, most computers operate in base 2, the **binary number system**. In this system, the addition and multiplication tables take the following simple form:

$$\begin{aligned} &0+0=0; \quad &0+1=1+0=1; \quad &1+1=10; \\ &0\cdot0=0; \quad &0\cdot1 \simeq 1\cdot0 \quad =0; \quad &1\cdot1=1. \end{aligned}$$

In the binary system, the number seventeen, for example, becomes 10001, since

$$1\cdot2^4+0\cdot2^3+0\cdot2^2+0\cdot2^1+1\cdot2^0 = \text{sixteen} + \text{one} = \text{seventeen}.$$

Put another way $(10001)_2 = (17)_{10}$; i.e., we use an index to denote the base of the number system (in decimal representation). The numbers become longer written in the binary system; large integers become about 3.3 times as long, since N binary digits suffice to represent integers less than $2^N = 10^{N\cdot\log 2} \approx 10^{N/3.3}$.

Occasionally one groups together the binary digits in subsequences of three or four, which is equivalent to using 2^3 and 2^4, respectively, as base. These systems are called the **octal** and **hexadecimal** number systems, respectively. The octal system uses the digits from 0 to 7; in the hexadecimal system, the digits 0 through 9 and the letters A, B, C, D, E, F ("ten" through "fifteen") are used.

Example

$$\begin{aligned} &(17)_{10} = (10001)_2 = (21)_8 = (11)_{16}, \\ &(13.25)_{10} = (1101.01)_2 = (15.2)_8 = (D.4)_{16}, \\ &(0.1)_{10} = (0.19999\ldots)_{16}. \end{aligned}$$

The "points" used to separate the integer and fractional part of a number (corresponding to the decimal point) are called the binary point, octal point, and hexadecimal point. The digits in the binary system are called bits (= **binary digits**). One very seldom converts (translates) from one number system to another manually; even a binary computer is fed decimal data and prints results in decimal form—it performs the conversions itself. There are also tables for conversion between the various number systems.

We are so accustomed to the position system that we forget that it is built upon an ingenious idea. The reader can puzzle over how the rules for arithmetical operations would look if one used roman numerals, a number system without the position principle described above.

2.3.2. Floating and Fixed Representation

A computer is usually equipped with two types of arithmetic operations, calculation with **fixed point** and with **floating point**. "Point" means the decimal point if the base is 10, or the binary point if the base is 2, etc. For simplicity of expression in the discussion that follows, we shall treat only the decimal case. A few of the most important results, however, will be formulated for a general base.

Computation with **floating** decimal point means that one works with a constant number of **digits;** computation with **fixed** decimal point means that one works with a constant number of **decimals.** Commonly, in the output data (because of various sources of error) some digits will be insignificant, or, respectively, some decimals will be incorrect.

2.3.3. Floating Decimal Point

By **normalized** *floating-decimal representation* of a number a, we mean a representation in the form:

$$a = m \cdot 10^q, \qquad 0.1 \le |m| < 1, \qquad q \text{ an integer.}$$

Such a representation is possible for all numbers a, and unique if $a \neq 0$. The variable m is called the fractional part or **mantissa** and q is called the **exponent.**

In a computer, the number of digits for q and m is limited. This means that only a finite set of numbers can be represented in the machine. The numbers in this set (for a given q and m, and usually a base different from 10) are called **floating-point numbers.** The limited number of digits in the exponent implies that a is limited to an interval which is called the machine's *floating-point variable* **range.** With, for example, three digits in the exponent (plus one bit for the sign of the exponent), the range is

$$10^{-1,000} \le |a| < 10^{999}.$$

For $a = 0$ one sets $m = 0$, and it is also practical to make q as small as possible; in the above example one would then take $q = -999$. Minor deviations from the conventions mentioned here occur on certain machines, but such deviations are of little importance for the questions which are studied here.

In the computer, a is represented by the floating number

$$\bar{a} = \bar{m} \cdot 10^q$$

where $\bar{m}$ means the mantissa m, rounded off to t decimals. The **precision** of the machine is said, then, to be t decimal digits. (One should be careful to distinguish between the precision of the machine and the accuracy of the input or output data of a problem.) Then, according to Sec. 2.1.4, we have

$$|\bar{m} - m| \le \begin{cases} \frac{1}{2} \cdot 10^{-t} & \text{for rounding,} \\ 10^{-t} & \text{for chopping.} \end{cases}$$

(There is one exception. If $|m|$ after rounding should be raised to 1, then $|\bar{m}|$ is set equal to 0.1 and q is raised by 1.)

The magnitude of the relative error in $\bar{a}$ is, then, since $m \ge 0.1$, at the most equal to

$$\frac{\frac{1}{2}10^{-t}\cdot 10^q}{m\cdot 10^q} \leq \frac{1}{2}\cdot 10^{1-t} \quad \text{for rounding,}$$

and twice as large for chopping. By analogous reasoning for an arbitrary base B we get the following theorem:

THEOREM 2.3.1

Suppose that the floating numbers in a machine have base B and a mantissa with t digits. (The binary digit which gives the sign of the number is not counted.) Then, every real number in the floating-point range of the machine can be represented with a relative error which does not exceed the **machine unit (round-off unit)** *u which is defined by:*

$$u = \begin{cases} \frac{1}{2}\cdot B^{1-t} & \textit{if rounding is used,} \\ B^{1-t} & \textit{if chopping is used.} \end{cases}$$

In most machines, u lies between 10^{-15} and 10^{-6}. The quantity u is, in many contexts, a natural unit for relative changes and relative errors. We shall occasionally use terminology of the type "the quantity is perturbed by a few u."

Input and output data seldom are so large in magnitude that the range of the machine is not sufficient. In the rare cases where the above does not hold, one can, for example, use double (long) precision or else work with logarithms or some other transformation of the data. One should, however, keep in mind the risk that intermediate results in a calculation can produce *exponent spill*, i.e., an exponent which is too large (exponent overflow) or too small (underflow) for the machine representation. Different machines take different actions in such situations, as well as for division by zero. One should acquaint oneself with the conventions used by the particular machine one is working with and write one's programs so that no irritating occurrences of the above phenomena go overlooked.

Too small an exponent is usually, but not always, unprovoking. If the machine does not signal that a result has an exponent which is too small, but simply sets the result equal to zero, then one can—if there is a risk of annoying consequences—put in a test to see whether the quantity is zero and then have the program either print a message and stop the run or continue at some other appropriate point in the program. (Most machines give a signal when a division by zero occurs, and the program is then often terminated.) Occasionally, but not often, "unexplainable errors" in output data are the product of an "underflow" somewhere in the computations.

Example 2.3.1

Even simple programs can quickly give exponent spill. If

$$x_0 = 2, \qquad x_{n+1} = x_n^2,$$

then already $x_{13} = 2^{8,192}$ is larger than most machines permit. One should also be careful in computation with factorials, for example $200! > 10^{374}$.

2.3.4. Fixed Decimal Point

Computation with fixed decimal point means that all real numbers are shortened to t decimals. Since the length of a computer word is usually constant (say, s digits), then only numbers in the interval $I = [-10^{s-t}, 10^{s-t}]$ are permitted. This restriction is much narrower than the corresponding one with floating representation with division of the word into mantissa and exponent. Some common conventions in fixed point are $t = s$ (fraction convention) or $t = 0$ (integer convention). Occasionally, the limitation on I causes difficulty, since even if $x \in I$, $y \in I$, we can have, for example, that $x + y$ or x/y lies outside of I.

If one writes programs for a fixed-point machine, then one must see to it that all numbers (even uninteresting intermediate results) remain within I. This can be attained by multiplying the variables by appropriate factors, so-called **scale factors**, and then transforming the equations accordingly. One can interpret this as changing the units of measurement of the variables of the problem. This complicates preparatory work, since there is a risk of another type: if one chooses the scale factors carelessly, certain intermediate results can have many leading zeros. The number of significant digits then becomes low; this can lead to poor accuracy in the final results. On the other hand, addition in fixed point is usually much faster than addition in floating point. One can say that in floating-point calculation the machine itself chooses a scale factor in each operation.

Most of the algebraic programming languages (Algol, Fortran, etc.) assume that elementary arithmetical operations shall be performable so long as the values of the variables are not unusually large. Practically, this means that floating representation is assumed. As a consequence, fixed point is seldom used, except when a program will be used so much (and in unmodified form) that the savings in computer time become greater than the additional cost from the increase in programming. In certain problems, the choice of scale factors is trivial; in others, it may be impracticable.

In practice, hexadecimal or binary representation is more common than decimal, which we have talked about here in order to simplify the mode of expression. The reader will surely have no difficulty in modifying the discussion for the binary or hexadecimal cases. See also the problems at the end of this section.

2.3.5. Round-off Errors in Computation with Floating Arithmetic Operations

Even if the operands in an arithmetic operation are exactly represented in the machine, it is not certain that the *exact* result of the operation is a floating-point number. For example, the exact product of two floating-point t-digit numbers has $2t$ or $2t - 1$ digits.

Let x and y be two floating-point machine numbers. Denote by

$$fl(x + y), \quad fl(x - y), \quad fl(xy), \quad fl\left(\frac{x}{y}\right)$$

the results of floating addition, subtraction, multiplication, and division, which the machine stores in memory (after rounding or chopping). In many computers, these results are equal to the rounded or chopped value of the exact result of the operation. We assume that such is the case, and thus, according to Theorem 2.3.1, we have,

$$|fl(x \text{ op } y) - x \text{ op } y| \leq |x \text{ op } y| \cdot u, \tag{2.3.1}$$

where "op" stands for one of the four symbols for elementary operations; $+, -, \cdot, /$. *The operations fl (x op y) have, to some degree, other properties than the exact arithmetic operations.*

Example 2.3.2

Associativity does not, in general, hold for floating addition. Consider floating addition using seven decimals in the mantissa,

$$a = 0.1234567 \cdot 10^0, \qquad b = 0.4711325 \cdot 10^4, \qquad c = -b.$$

The following schema indicates how floating addition is performed.

$$fl(b + c) = 0, \qquad fl(a + fl(b + c)) = a = 0.1234567 \cdot 10^0$$

$$\begin{array}{rl|r}
a = & 0.0000123 & 4567 \cdot 10^4 \\
+b = & 0.4711325 & \cdot 10^4 \\
\hline
fl(a + b) = & 0.4711448 & \cdot 10^4 \\
c = & -0.4711325 & \cdot 10^4
\end{array}$$

The digits to the right of the vertical line are thrown away.

$$fl(fl(a + b) + c) = 0.0000123 \cdot 10^4 = 0.1230000 \cdot 10^0 \neq fl(a + fl(b + c)).$$

Example 2.3.3

Using a hexadecimal machine (floating representation, $t = 6$, with chopping, $u = 16^{-5} \approx 10^{-6}$) one computes

$$\sum_{n=1}^{10,000} n^{-2} \approx 1.644834$$

in two different orders. Using the natural ordering $((1 + \frac{1}{4}) + \frac{1}{9}) + \ldots$; the error was $1{,}317 \cdot 10^{-6}$, but summing in the opposite order, the error was only $2 \cdot 10^{-6}$. This was not unexpected. Each operation is an addition, where the sum s is increased by n^{-2}; thus, in each operation, one commits an error of about $s \cdot u$, and all of these errors are added. Using the first summation order, we have $1 \leq s \leq 2$ in every step, but using the other order of summation, we have $s < 10^{-2}$ in 9,900 of the 10,000 additions.

If the terms in a sum (of positive terms) are of varying orders of magnitude, one should add the terms in increasing order if one needs high accuracy in the result. It is seldom that one needs to think of this in practice, but important instances in which this idea can be applied do occur; see Chap. 8.

The previous example is mainly of interest as a simple illustration of the important thesis: *mathematically equivalent algorithms are not always numerically equivalent.* By **mathematical equivalence** of two algorithms we mean here that the algorithms give exactly the same results from the same input data, if the computations are made without round-off error ("with infinitely many decimals"). The one algorithm can then as a rule be derived from the other purely formally using the rules of algebra for real numbers (or else with the help of other mathematical identities). Two algorithms are **numerically equivalent** if their respective results, using the same input data, do not differ by more than what the problem's exact output data would be changed by if the input data were perturbed by a few u (u was defined in Theorem 2.3.1).

Example 2.3.4. *Increasing the Precision of Addition in Floating-Point Arithmetic*

In Example 2.3.3, when adding in the natural order, about the last *9,000* terms give *no* contribution to the sum because of the summation order, the fact that a computer has a fixed finite word length, and the way floating-point addition is performed (recall $u \approx 10^{-6}$). There the remedy was to sum the terms in the opposite order (smallest to largest). Often, however, one has no a priori information about the relative magnitudes of the terms to be summed. If there are a large number of such terms, then it can be time-consuming to sort them into order of increasing magnitude before adding. Also, the number of extra memory cells (equal to the number of terms in the sum) required for the sorting may not be available.

Consider the "usual" program segment for computing a sum $\sum_{i=1}^{N} X_i$:

```
      S = 0.0
      DO 33 I = 1,N
   33 S = S + X(I)
```

Let S_i denote the partial sum (value of the variable S) after adding the term X_i. Since for each i the additional error is bounded by $|S_i|\cdot u$, the total error is bounded approximately by $N\cdot(\sum_{i=1}^{N}|X_i|)\cdot u$ (see also the more exact expression given in the table in Sec. 2.4.1). This may be unsatisfactory for N large. We would like a way to compute sums with floating-point arithmetic which gives a small error bound independent of N.

To illustrate the basic idea which can be used, take $A = 0.1234567\cdot 10^0$, $B = 0.4711325\cdot 10^4$. Here $fl(A + B) = 0.4711448\cdot 10^4$ (denote this by S). Suppose we form

$$C = fl(fl(B - fl(A + B)) + A)$$

corresponding to the Fortran statements

```
S = A + B
C = (B - S) + A
```

Then

$$C = (-.1230000 \cdot 10^0) + (.1234567 \cdot 10^0)$$
$$= 0004567 \cdot 10^0 = 4567000 \cdot 10^{-3}.$$

Thus the variable C has picked up the information that was lost in the operation

```
S = A + B.
```

W. Kahan, at the 1971 IFIP Congress, gave the following program for computing $\sum_{J=1}^{N} X(J, \ldots)$ which implements the above idea and also contains some other sophistications to account for the differences in performance of floating-point operations in existing computers:

```
        S = 0.
        C = 0.
        DO 999  J=1, N
        Y = C + X(J,...)
        T = S + Y
        F = 0.
        IF(sign (Y) equals sign (S)) F = (0.46*T - T) + T
        C = ((S-F)-(T-F)) + Y
    999 S = T
        S = S + C
```

Kahan states that "on all current North American machines with floating point hardware," this program keeps $|\xi_j|$ less than $5u + O(Nu^2)$ in the expression for the *computed* sum of N terms,

$$S_N = \sum_{j=1}^{N} (1 + \xi_j)X_j.$$

(This formulation is a typical example of backward error analysis—see Section 2.4.)
Thus the above program gives a very small error bound in the sum, and the error bound is (essentially) independent of N.

We do not go into more detail here. For a comprehensive analysis of floating-point operations, and references to the literature, see Knuth [13].†

REVIEW QUESTIONS

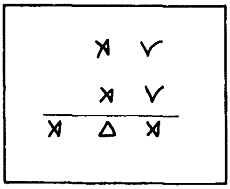
1. What are the binary, octal, and hexadecimal number systems?

†Bibliographical references are listed in Chapter 13, starting in Sec. 13.2.

2. What is meant by "floating-point representation"?
3. How large can the maximum relative error be in the floating-point representation of a number?
4. Give examples to show that some of the axioms for arithmetic with real numbers need not always hold for floating-point arithmetic.

PROBLEMS

1. (a) Which numbers can be expressed with a finite number of binary digits to the right of the binary point?
   (b) Give a number which can be expressed with a finite decimal fraction, but which in binary notation requires an infinite number of binary digits!
2. The (fictional) space probe *Venus 17* sent back pictures of a Venusian rock-carving, assumed to constitute an addition. Which number system do the inhabitants of Venus count in? How many fingers can one presume that they have on each hand? (Almost every culture on earth uses one of the numbers 5, 10, or 20 as base for its number system.)



3. Give in decimal representation: (a) $(10000)_2$; (b) $(100)_8$; (c) $(64)_{16}$; (d) $(FF)_{16}$; (e) $(0.11)_8$; (f) $(0.0631463146314\ldots)_8$; (g) the largest positive integer which can be written with thirty-one binary digits (answer with one significant digit).
4. In the fictional computer EDC4711, a memory cell has two binary positions for storing signs (of the mantissa and of the exponent) and eleven decimal positions for the exponent. The number π is, for example, stored in the following way:

$$\boxed{+}\boxed{3}\boxed{1}\boxed{4}\boxed{1}\boxed{5}\boxed{9}\boxed{2}\boxed{6}\boxed{5}\boxed{3}\boxed{6}\qquad\boxed{+}\boxed{0}\boxed{0}\boxed{1}$$

   With this representation, numbers in the interval $(-10^{999}, 10^{999})$ are stored to eleven significant digits. Show how the following numbers are stored: (a) 2.7182818285; (b) -1073741824; (c) 0.577216; (d) $-123\cdot 10^{-45}$.
5. How large (maximally) can the relative error be when a number is stored using the representation in Problem 4?
6. In the CDC 3200, a floating-point number is stored using forty-eight binary positions, in the following way:

1	11	36
s.	exp.	mantissa

A given number is written in the form $p = \pm p' 2^{p''}$, where the mantissa p' must satisfy $\frac{1}{2} \leq p' < 1$. The mantissa is stored as a (correctly rounded) binary fraction. The exponent, which must satisfy $|p''| < 2^{10}$, is represented by the binary integer $p'' + 2^{10}$ if $p'' \geq 0$, otherwise with $p'' + (2^{10} - 1)$. The sign bit is set equal to zero. If $p < 0$, then the ones-complement of the representation obtained above is taken—that is, in all forty-eight positions the ones are changed to zeros and the zeros to ones. (Thus, in particular, the sign bit is set to 1 if $p < 0$.) If $p = 0$, the number is stored as all zeros.

Show how the following numbers would be stored:

(a) 1.0; (b) −0.0625; (c) 250.25; (d) 0.1.

(e) Give (in decimal notation) the largest and smallest positive numbers which can be stored.

(f) What is the maximal relative error with which a number is stored?

7. In IBM system 360, every memory cell consists of thirty-two binary positions. A number p is stored in floating hexadecimal form in the following way:

One sets

$$p = p' \cdot 16^{(p''-64)},$$

where $p'' \geq 0$ and $\frac{1}{16} \leq |p'| < 1$; and one stores the mantissa p' with sign and the modified exponent p''.

In single-precision representation, one memory cell is used; p'' is allocated 7 bits (pos. 2–8); p' is given 1 bit for sign (pos. 1) and 24 bits for p' (pos. 9–32).

In double-precision representation, two memory cells are used; p'' is again allocated 7 bits (pos. 2–8, cell 1), and p' is allocated 1 bit for sign (pos. 1, cell 1), and 56 bits (pos. 9–32 in cell 1 and all of cell 2) thereafter.

How are the following numbers represented in single precision?

(a) 1.0; (b) 0.5; (c) −15.0

(d) Give (decimally) the largest and smallest positive numbers which can be stored.

(e) What is the maximal relative error with which a number can be stored in single and double precision, respectively? (Give the answers in decimal notation.)

2.4. BACKWARD ERROR ANALYSIS; CONDITION NUMBERS

2.4.1. Backward Error Analysis

Let us use the notation of Sec. 2.3.5 and let "op" denote one of the operations $+, -, \cdot, /$. By Eq. (2.3.1), it follows that there exists a number δ, $|\delta| \leq u$ (where δ depends on x and y) such that

$$fl(x \text{ op } y) = (x \text{ op } y)(1 + \delta), \quad (|\delta| \leq u). \tag{2.4.1}$$

This means, for example, that in multiplication, $fl(xy)$ is the exact product of x and $y(1 + \delta)$ for some δ, $|\delta| \leq u$. In the same way, the results using the three other operations can be interpreted as the result of exact operations where the operands have been perturbed by a relative amount which does not exceed u. To use **backward error analysis** means that one applies the above interpretation step by step backwards in an algorithm. By means of backward error analysis it has been shown, even for many quite complicated algorithms, that the output data which the algorithms produce (under the influence of round-off error) is the exact output data of a problem of the same type in which the input data has been changed (relatively) by a few u. That change, measured with u as a unit, is called the **condition number of the algorithm.** Clearly, the condition number has a small value for a good algorithm and a large value for a poor algorithm.

Using backward error analysis, one transfers the problem of estimating the effects of round-off errors during the computation back to the problem of estimating the effect of disturbances in input data. In this way, the general error-propagation formula (see Sec. 2.2.2) can then be used in the estimation of the effect of the disturbances. A graphical illustration of backward error analysis is given in Fig. 2.4.4.

Example 2.4.1

By repeatedly using formula (2.4.1) in the case of multiplication, one can show that $fl(x_1 x_2 \ldots x_n)$ is exactly equal to a product of the form

$$x_1x_2(1+\delta_2)x_3(1+\delta_3)\ldots x_n(1+\delta_n),$$

where

$$|\delta_i| \leq u, \quad i = 2, 3, \ldots, n.$$

From this, it follows that

$$|fl(x_1x_2\ldots x_n) - x_1x_2\ldots x_n| = \epsilon\,|x_1x_2\ldots x_n|,$$

where

$$\epsilon = |(1+\delta_2)(1+\delta_3)\ldots(1+\delta_n) - 1|.$$

Thus

$$\epsilon \leq (1+u)^{n-1} - 1.$$

For convenience, let $j = n - 1$ and make the (realistic) assumption that $(n-1)\cdot u < 0.1$. Then:

$$\ln(1+u)^j = j\ln(1+u) < ju, \quad (u < 1)$$

so

$$(1+u)^j < e^{ju}$$

$$(1+u)^j - 1 < (ju) + \frac{(ju)^2}{2!} + \frac{(ju)^3}{3!} + \ldots$$

$$(1+u)^j - 1 < ju\left(1 + \left(\frac{ju}{2}\right) + \left(\frac{ju}{2}\right)^2 + \left(\frac{ju}{2}\right)^3 + \ldots\right)$$

$$(1+u)^j - 1 < ju\left(1 + \frac{0.05}{1-0.05}\right) < 1.06\,ju$$

Thus $\epsilon < 1.06(n - 1)u$, for $(n - 1)u < 0.1$.

Example 2.4.2

Recall the notation (introduced in Problem 8 at the end of Sec. 2.2):

R_X = a bound for the part of the *absolute* error in a result which depends on the error in the input data.

R_C = a bound for the *absolute* error in a result which depends on round-off errors during the calculations.

If the magnitude of the relative error in the indata (x_i, y_i, a_i, x), $i = 0, 1, \ldots, n$ does not exceed r, then, using backward error analysis analogously to Example 2.4.1, one finds the following results for some commonly occurring expressions (assuming that $2nu \leq 0.1$, $nr \leq 0.1$):

	$\dfrac{R_X}{1.06r}$	$\dfrac{R_C}{1.06u}$
$\sum_{i=1}^{n} x_i$	$\sum \lvert x_i \rvert$	$\sum \lvert (n + 1 - i)x_i \rvert$
$\prod_{i=1}^{n} x_i$	$n\prod \lvert x_i \rvert$	$(n - 1) \prod \lvert x_i \rvert$
$\sum_{i=1}^{n} x_i y_i$	$2 \sum \lvert x_i y_i \rvert$	$\sum \lvert (n + 2 - i)x_i y_i \rvert$
$\sum_{i=0}^{n} a_i x^i = F(x)$	$\lvert xF'(x) \rvert + \sum \lvert a_i x^i \rvert$	$\sum \lvert (2i + 1)a_i x^i \rvert$

Here, the sum, the product, and the scalar product are assumed to be computed in the natural order, i.e., for the sum $(\ldots(((x_1 + x_2) + x_3) + x_4) + \ldots + x_n)$, and analogously for the products; the polynomial is assumed to be computed by Horner's rule, see Sec. 1.3.2. The reader is of course recommended to verify the results given above (Problem 1 at the end of this section).

2.4.2. Condition Numbers for Problems and Algorithms

There can be many reasons for poor accuracy in output data. For example, the algorithm can be poorly constructed. But poor accuracy can also depend on the problem itself; that is, the output data can be very sensitive to disturbances in the input data—independently of the choice of algorithm. In the first case, we say that the algorithm is ill-conditioned; in the second case, the problem is said to be **ill-conditioned.** One also says that the algorithm is **numerically unstable** or that the problem is (mathematically) unstable.

Consider a numerical problem P, with given (possibly error-afflicted) input data, and an algorithm A, which is used on a computer with round-off unit u (floating point). Since floating point is used, it is natural to look at relative errors.

The **condition number C_A for the algorithm** A was defined in Sec. 2.4.1. We shall consider the output data produced by algorithm A as the exact output data for the problem with the same structure as P, except that the input data has been changed. The condition number of the algorithm indicates a sufficient size for such a relative change using u as a unit of measurement.

The **condition number C_P for the problem** P with given input data is the largest relative change, measured with u as a unit, that the exact output data of the problem can have, if there is a relative disturbance in the input data of size u.

If r is a bound for the relative error in the input data, we have then

$$C_P(r + C_A u)$$

as *a bound for the relative error in the output data.*

The condition numbers for P and A depend on the input data. A problem or an algorithm can be well-conditioned (have a small condition number) for certain input data, and ill-conditioned (have a large condition number) for other input data. Furthermore, C_P and C_A are independent of each other; C_P can be small even if C_A is large, and vice versa. In principle, C_P can be estimated using the general error-propagation formula.

Example 2.4.3

Let P be the problem of computing the value $f(x)$ for the function f at the point x. Then, from the definition of C_P, we have that C_P is equal to the largest of the two values that

$$\left|\frac{(f(x+\Delta x) - f(x))/f(x)}{\Delta x/x}\right|$$

takes for $\Delta x = ux$ and $\Delta x = -ux$, respectively. Thus we find that (practically):

$$C_P = \left|\frac{f'(x)/f(x)}{1/x}\right|.$$

Example 2.4.4

Compute x from the system

$$x + \alpha y = 1$$
$$\alpha x + y = 0.$$

The solution is $x = (1 - \alpha^2)^{-1}$. The matrix is singular for $\alpha = 1$. The condition number for the problem of computing x, when α is the only input data value which has been rounded, is

$$C_P = \frac{x'(\alpha)/x(\alpha)}{1/\alpha} = \frac{2\alpha^2}{1-\alpha^2}.$$

Thus, the problem is **ill-conditioned** when $\alpha^2 \approx 1$, but very **well-conditioned** when $\alpha^2 \ll 1$. The condition number is thus related to how near the matrix of coefficients lies to a singular matrix. The problem of computing $y = -\alpha(1-\alpha^2)^{-1}$ is also **ill-conditioned** when $\alpha^2 \approx 1$.

On the other hand, the **problem** of computing

$$z = x + y = \frac{1}{1-\alpha^2} - \frac{\alpha}{1-\alpha^2} = \frac{1}{1+\alpha} \tag{2.4.2}$$

is **well-conditioned** even when $\alpha \approx 1$. (The condition number is then about 0.5.) But there would be a fatal cancellation if, when $\alpha \approx 1$, one first solved for x and y from the system of equations and then got z by adding x and y. For example, for $\alpha = 0.9900$ the condition number for solving for x and y is about 100. With floating-decimal arithmetic (using four digits), one gets

$$z = x + y = 0.5025 \cdot 10^2 - 0.4975 \cdot 10^2 = 0.5000,$$

while the correct answer, using Eq. (2.4.2), is $z = 0.5025$. In this case, it is the **algorithm** for computing z, with the use of rounded values for x and y, which is **ill-conditioned,** not the problem. In order to get four digits in z with that algorithm, one would need six digits in x and y. In comparison with what can happen in larger problems, however, this is a very mild degree of poor conditioning.

If the relative accuracy in the output data is unacceptably poor, and if this is because C_A is large, then one should first investigate whether a simple **rewriting of the computational sequence** can improve things (eliminate cancellations, change the order of summation, etc.). If this does not work, then one can investigate whether it is sufficient to use **higher-precision arithmetic** in a small part of the computation—that is, use more digits than normal. This is possible on most computers, but computational speed is reduced and there can also be a great deal of extra programming involved.

Notice, in addition, that C_A is related to the *relative accuracy* in a result which A gives. If one does not need high relative accuracy in a quantity with a small absolute magnitude, then it is not necessary to reject the algorithm, even if C_A is large. (Condition numbers can, on the other hand, be defined with respect to absolute accuracy.)

If the poor accuracy is caused by a large C_P, then one might at first think that nothing can be done about the situation; the difficulty lies, of course, with P. But *the difficulty can depend on the form one has chosen to represent the input and output data of the problem.*

Example 2.4.5

The polynomial P,

$$P(x) = (x - 10)^4 + 0.200(x - 10)^3 + 0.0500(x - 10)^2 \\ - 0.00500(x - 10) + 0.00100,$$

is identical with a polynomial Q which, if the coefficients are rounded to six digits, becomes

$$\tilde{Q}(x) = x^4 - 39.8000x^3 + 594.050x^2 - 3941.00x + 9805.05.$$

One finds that $P(10.11) = 0.0015 \pm 10^{-4}$, where only three digits are needed in the computation, while $\tilde{Q}(10.11) = -0.0481 \pm \frac{1}{2}\cdot 10^{-4}$, in spite of the fact that eight digits were used in the computation. The rounding to six digits of the coefficients of Q has thus caused an error in the polynomial's value at $x = 10.11$; the erroneous value is more than 30 times larger than the correct value $P(10.11)$. When the coefficients of Q are input data, the problem of computing the value of the polynomial for $x \approx 10$ is far more ill-conditioned than when the coefficients of P are input data.

2.4.3. Geometrical Illustrations of Error Analysis

The problem of error propagation can to some degree be illustrated geometrically. A numerical problem P means a mapping of the space X of possible input data onto the space Y of the output data. The dimensions of these spaces are usually quite large. In Fig. 2.4.1 we satisfy ourselves with two di-

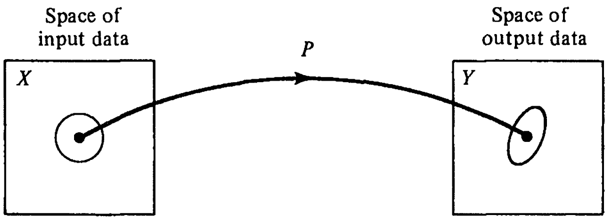


Fig. 2.4.1

mensions, and, since we are considering relative changes, we can consider that the coordinate axes are logarithmically scaled. A small circle of radius r is mapped onto an ellipse whose major axis is about $C_p r$.

In the construction of an algorithm for a given problem, one often breaks down the problem into a chain of subproblems, $P_1, P_2, \ldots, P_k$ for which algorithms $A_1, A_2, \ldots, A_k$ are known, in such a way that the output data from P_{i-1} is the input data to P_i (Fig. 2.4.2). Different ways of decomposing the problem give different algorithms, as a rule with different condition numbers. It is dangerous if the last part of such a chain—for example, the last link—is ill-conditioned. On the other hand, it need not be dangerous if the first link of such a decomposition of the problem is ill-conditioned, if the problem itself is well-conditioned.

In Fig. 2.4.3 we see two examples of a decomposition of the problem P into two subproblems. From X to X'' there is a strong contraction which is followed by an expansion about equally strong in the mapping from X'' to Y. The round-off errors which are made in X'' when the intermediate results are stored have as a consequence that one arrives somewhere in the surrounding circle, which is then transformed to a very large region in Y. The situation in Problem 9 in Sec. 2.2 corresponds approximately to this picture—but with other proportions.

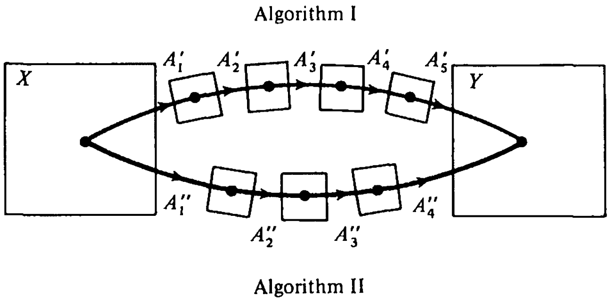


Fig. 2.4.2

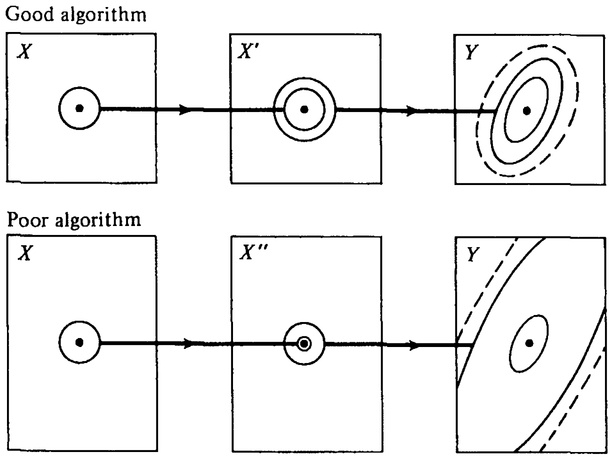


Fig. 2.4.3

Example 2.4.6

One can show that the problem P of determining the largest eigenvalue of a real symmetric matrix is always well-conditioned, $C_P \approx 1$. It seems natural to break down this problem into the two subproblems:

P_1: to compute the coefficients of the characteristic equation of the matrix.

P_2: to solve the characteristic equation.

But there is a matrix of order 40 where the condition number for P_2 is 10^{14}—in spite of the fact that the origin lies exactly between the largest and smallest eigenvalues, so that one cannot blame the high condition number on a difficulty of the same type as that encountered in Example 2.4.5. This is an extreme case of the situation which the lower picture in Fig. 2.4.3 is intended to illustrate.

A general method for solving the eigenvalue problem for symmetric matrices should, then, not be based on the above way of breaking down the problem. A possibility for getting around the difficulty is to use higher precision. But there are in fact more effective ways to attack the problem; see Chap. 5.

Pictures similar to Figs. 2.4.1–5 can help clarify many other relations in composite numerical processes. Figure 2.4.4 illustrates backward error analysis (Sec. 2.4.1); x is input data and x' is an intermediate result; y is the output data obtained by the numerical computation, which were influenced by rounding errors. The results of the backward analysis are the circles in X' and X, in which there are certainly points $\hat{x}'$ and $\hat{x}$ which, under the exact mappings from X' to Y and from X to X', are mapped onto the points y and $\hat{x}'$,

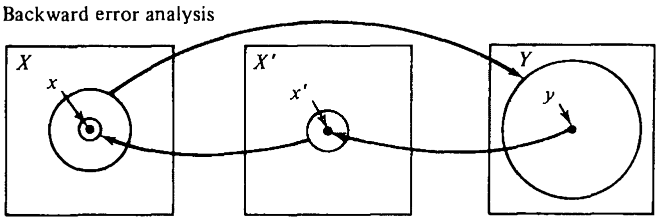


Fig. 2.4.4

respectively. The exact positions of $\hat{x}'$ and $\hat{x}$ are not known. After this backward analysis of the algorithm and a forward analysis of the mathematical problem, we know that x is mapped onto a point somewhere in the circle in Y. (The smaller circle in X is the exact inverse image of the circle in X'.) Backward error analysis is often more adequate than forward error analysis.

Finally, Fig. 2.4.5 gives an illustration of one difficulty which can occur with the use of interval analysis (see Example 2.2.13).

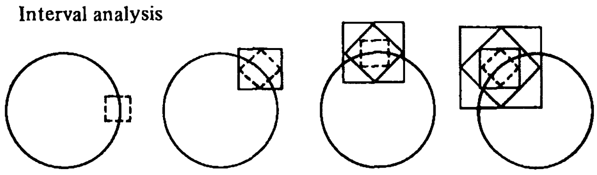


Fig. 2.4.5

REVIEW QUESTIONS

1. Define condition number for:
   (a) a numerical problem;
   (b) an algorithm.
2. Give examples of well-conditioned and ill-conditioned problems.
3. Give an example of an ill-conditioned algorithm for a well-conditioned problem, and vice versa.
4. What is backward error analysis? Give an example of its use.
5. A problem with condition number C_P is treated using an algorithm which has condition number C_A, on a computer with round-off unit (machine unit) u. Show that if the input data has a relative error whose absolute value is less than r, then the absolute value of the relative error in the output data is less than $C_P(r + C_A u)$.

PROBLEM

Verify some of the results given in the table in Example 2.4.2.