28. The opening example to this chapter described a physical experiment involving the temperature of a gas under pressure. In this application, we were given $P = 1.00$ atm, $V = 0.100$ m^3, $N = 0.00420$ mol, and $R = 0.08206$. Solving for T in the ideal gas law gives

$$T = \frac{PV}{NR} = \frac{(1.00)(0.100)}{(0.00420)(0.08206)} = 290.15 \text{ K} = 17°\text{C}.$$

In the laboratory, it was found that T was 15°C under these conditions, and when the pressure was doubled and the volume halved, T was 19°C. Assume that the data are rounded values accurate to the places given, and show that both laboratory figures are within the bounds of accuracy for the ideal gas law.

1.3 Algorithms and Convergence

Throughout the text we will be examining approximation procedures, called *algorithms*, involving sequences of calculations. An **algorithm** is a procedure that describes, in an unambiguous manner, a finite sequence of steps to be performed in a specified order. The object of the algorithm is to implement a procedure to solve a problem or approximate a solution to the problem.

The use of an algorithm is as old as formal mathematics, but the name derives from the Arabic mathematician Muhammad ibn-Mŝâ al-Khwarârizmî (c. 780–850). The Latin translation of his works begins with the words "Dixit Algorismi" meaning "al-Khwarârizmî says."

We use a **pseudocode** to describe the algorithms. This pseudocode specifies the form of the input to be supplied and the form of the desired output. Not all numerical procedures give satisfactory output for arbitrarily chosen input. As a consequence, a stopping technique independent of the numerical technique is incorporated into each algorithm to avoid infinite loops.

Two punctuation symbols are used in the algorithms:

- a period (.) indicates the termination of a step,
- a semicolon (;) separates tasks within a step.

Indentation is used to indicate that groups of statements are to be treated as a single entity.

Looping techniques in the algorithms are either counter-controlled, such as,

For $i = 1, 2, \ldots, n$

Set $x_i = a + i \cdot h$

or condition-controlled, such as

While $i < N$ do Steps 3–6.

To allow for conditional execution, we use the standard

If . . . then or If . . . then

else

constructions.

The steps in the algorithms follow the rules of structured program construction. They have been arranged so that there should be minimal difficulty translating pseudocode into any programming language suitable for scientific applications.

The algorithms are liberally laced with comments. These are written in italics and contained within parentheses to distinguish them from the algorithmic statements.

Illustration The following algorithm computes $x_1 + x_2 + \cdots + x_N = \sum_{i=1}^{N} x_i$, given N and the numbers $x_1, x_2, \ldots, x_N$.

INPUT $N, x_1, x_2, \ldots, x_n$.

OUTPUT $SUM = \sum_{i=1}^{N} x_i$.

Step 1 Set $SUM = 0$. (*Initialize accumulator.*)

Step 2 For $i = 1, 2, \ldots, N$ do
set $SUM = SUM + x_i$. (*Add the next term.*)

Step 3 OUTPUT (SUM);
STOP. □

Example 1 The Nth Taylor polynomial for $f(x) = \ln x$ expanded about $x_0 = 1$ is

$$P_N(x) = \sum_{i=1}^{N} \frac{(-1)^{i+1}}{i}(x-1)^i,$$

and the value of $\ln 1.5$ to eight decimal places is 0.40546511. Construct an algorithm to determine the minimal value of N required for

$$|\ln 1.5 - P_N(1.5)| < 10^{-5},$$

without using the Taylor polynomial remainder term.

Solution From calculus we know that if $\sum_{n=1}^{\infty} a_n$ is an alternating series with limit A whose terms decrease in magnitude, then A and the Nth partial sum $A_N = \sum_{n=1}^{N} a_n$ differ by less than the magnitude of the $(N+1)$st term; that is,

$$|A - A_N| \leq |a_{N+1}|.$$

The following algorithm uses this bound.

INPUT value x, tolerance TOL, maximum number of iterations M.
OUTPUT degree N of the polynomial or a message of failure.

Step 1 Set $N = 1$;
$y = x - 1$;
$SUM = 0$;
$POWER = y$;
$TERM = y$;
$SIGN = -1$. (*Used to implement alternation of signs.*)

Step 2 While $N \leq M$ do Steps 3–5.

Step 3 Set $SIGN = -SIGN$; (*Alternate the signs.*)
$SUM = SUM + SIGN \cdot TERM$; (*Accumulate the terms.*)
$POWER = POWER \cdot y$;
$TERM = POWER/(N+1)$. (*Calculate the next term.*)

Step 4 If $|TERM| < TOL$ then (*Test for accuracy.*)
OUTPUT (N);
STOP. (*The procedure was successful.*)

Step 5 Set $N = N + 1$. (*Prepare for the next iteration.*)

Step 6 OUTPUT ('Method Failed'); (*The procedure was unsuccessful.*)
STOP.

The input for our problem is $x = 1.5$, $TOL = 10^{-5}$, and perhaps $M = 15$. This choice of M provides an upper bound for the number of calculations we are willing to perform, recognizing that the algorithm is likely to fail if this bound is exceeded. Whether the output is a value for N or the failure message depends on the precision of the computational device. ■

Characterizing Algorithms

We will be considering a variety of approximation problems throughout the text, and in each case we need to determine approximation methods that produce dependably accurate results for a wide class of problems. Because of the differing ways in which the approximation methods are derived, we need a variety of conditions to categorize their accuracy. Not all of these conditions will be appropriate for any particular problem.

One criterion we will impose on an algorithm whenever possible is that small changes in the initial data produce correspondingly small changes in the final results. An algorithm that satisfies this property is called **stable**; otherwise it is **unstable**. Some algorithms are stable only for certain choices of initial data, and are called **conditionally stable**. We will characterize the stability properties of algorithms whenever possible.

The word *stable* has the same root as the words stand and standard. In mathematics, the term stable applied to a problem indicates that a small change in initial data or conditions does not result in a dramatic change in the solution to the problem.

To further consider the subject of round-off error growth and its connection to algorithm stability, suppose an error with magnitude $E_0 > 0$ is introduced at some stage in the calculations and that the magnitude of the error after n subsequent operations is denoted by E_n. The two cases that arise most often in practice are defined as follows.

Definition 1.17 Suppose that $E_0 > 0$ denotes an error introduced at some stage in the calculations and E_n represents the magnitude of the error after n subsequent operations.

- If $E_n \approx CnE_0$, where C is a constant independent of n, then the growth of error is said to be **linear**.
- If $E_n \approx C^nE_0$, for some $C > 1$, then the growth of error is called **exponential**. ■

Linear growth of error is usually unavoidable, and when C and E_0 are small the results are generally acceptable. Exponential growth of error should be avoided, because the term C^n becomes large for even relatively small values of n. This leads to unacceptable inaccuracies, regardless of the size of E_0. As a consequence, an algorithm that exhibits linear growth of error is stable, whereas an algorithm exhibiting exponential error growth is unstable. (See Figure 1.12.)

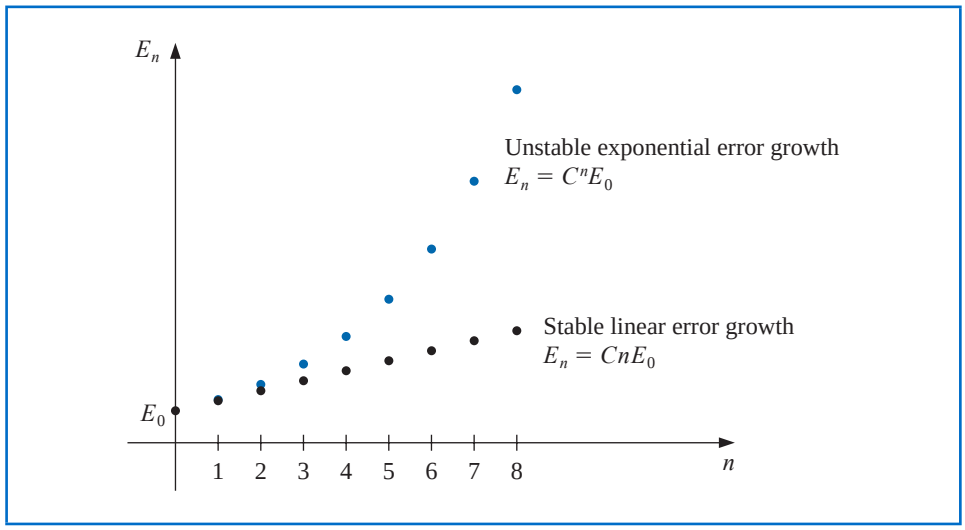
Illustration For any constants c_1 and c_2,

$$p_n = c_1\left(\frac{1}{3}\right)^n + c_2 3^n,$$

is a solution to the recursive equation

$$p_n = \frac{10}{3}p_{n-1} - p_{n-2}, \quad \text{for } n = 2, 3, \ldots.$$

Figure 1.12



This can be seen by noting that

$$\begin{aligned}\frac{10}{3}p_{n-1} - p_{n-2} &= \frac{10}{3}\left[c_1\left(\frac{1}{3}\right)^{n-1} + c_2 3^{n-1}\right] - \left[c_1\left(\frac{1}{3}\right)^{n-2} + c_2 3^{n-2}\right] \\ &= c_1\left(\frac{1}{3}\right)^{n-2}\left[\frac{10}{3}\cdot\frac{1}{3} - 1\right] + c_2 3^{n-2}\left[\frac{10}{3}\cdot 3 - 1\right] \\ &= c_1\left(\frac{1}{3}\right)^{n-2}\left(\frac{1}{9}\right) + c_2 3^{n-2}(9) = c_1\left(\frac{1}{3}\right)^{n} + c_2 3^{n} = p_n.\end{aligned}$$

Suppose that we are given $p_0 = 1$ and $p_1 = \frac{1}{3}$. This determines unique values for the constants as $c_1 = 1$ and $c_2 = 0$. So $p_n = \left(\frac{1}{3}\right)^n$ for all n.

If five-digit rounding arithmetic is used to compute the terms of the sequence given by this equation, then $\hat{p}_0 = 1.0000$ and $\hat{p}_1 = 0.33333$, which requires modifying the constants to $\hat{c}_1 = 1.0000$ and $\hat{c}_2 = -0.12500 \times 10^{-5}$. The sequence $\{\hat{p}_n\}_{n=0}^{\infty}$ generated is then given by

$$\hat{p}_n = 1.0000\left(\frac{1}{3}\right)^n - 0.12500 \times 10^{-5}(3)^n,$$

which has round-off error,

$$p_n - \hat{p}_n = 0.12500 \times 10^{-5}(3^n),$$

This procedure is unstable because the error grows *exponentially* with n, which is reflected in the extreme inaccuracies after the first few terms, as shown in Table 1.5 on page 36.

Now consider this recursive equation:

$$p_n = 2p_{n-1} - p_{n-2}, \quad \text{for } n = 2, 3, \ldots.$$

It has the solution $p_n = c_1 + c_2 n$ for any constants c_1 and c_2, because

$$\begin{aligned}2p_{n-1} - p_{n-2} &= 2(c_1 + c_2(n-1)) - (c_1 + c_2(n-2)) \\ &= c_1(2-1) + c_2(2n - 2 - n + 2) = c_1 + c_2 n = p_n.\end{aligned}$$

Table 1.5

n	Computed $\hat{p}_n$	Correct p_n	Relative Error
0	0.10000×10^{1}	0.10000×10^{1}	
1	0.33333×10^{0}	0.33333×10^{0}	
2	0.11110×10^{0}	0.11111×10^{0}	9×10^{-5}
3	0.37000×10^{-1}	0.37037×10^{-1}	1×10^{-3}
4	0.12230×10^{-1}	0.12346×10^{-1}	9×10^{-3}
5	0.37660×10^{-2}	0.41152×10^{-2}	8×10^{-2}
6	0.32300×10^{-3}	0.13717×10^{-2}	8×10^{-1}
7	-0.26893×10^{-2}	0.45725×10^{-3}	7×10^{0}
8	-0.92872×10^{-2}	0.15242×10^{-3}	6×10^{1}

If we are given $p_0 = 1$ and $p_1 = \frac{1}{3}$, then constants in this equation are uniquely determined to be $c_1 = 1$ and $c_2 = -\frac{2}{3}$. This implies that $p_n = 1 - \frac{2}{3}n$.

If five-digit rounding arithmetic is used to compute the terms of the sequence given by this equation, then $\hat{p}_0 = 1.0000$ and $\hat{p}_1 = 0.33333$. As a consequence, the five-digit rounding constants are $\hat{c}_1 = 1.0000$ and $\hat{c}_2 = -0.66667$. Thus

$$\hat{p}_n = 1.0000 - 0.66667n,$$

which has round-off error

$$p_n - \hat{p}_n = \left(0.66667 - \frac{2}{3}\right) n.$$

This procedure is stable because the error grows grows *linearly* with n, which is reflected in the approximations shown in Table 1.6. □

Table 1.6

n	Computed $\hat{p}_n$	Correct p_n	Relative Error
0	0.10000×10^{1}	0.10000×10^{1}	
1	0.33333×10^{0}	0.33333×10^{0}	
2	-0.33330×10^{0}	-0.33333×10^{0}	9×10^{-5}
3	-0.10000×10^{1}	-0.10000×10^{1}	0
4	-0.16667×10^{1}	-0.16667×10^{1}	0
5	-0.23334×10^{1}	-0.23333×10^{1}	4×10^{-5}
6	-0.30000×10^{1}	-0.30000×10^{1}	0
7	-0.36667×10^{1}	-0.36667×10^{1}	0
8	-0.43334×10^{1}	-0.43333×10^{1}	2×10^{-5}

The effects of round-off error can be reduced by using high-order-digit arithmetic such as the double- or multiple-precision option available on most computers. Disadvantages in using double-precision arithmetic are that it takes more computation time and the growth of round-off error is not entirely eliminated.

One approach to estimating round-off error is to use interval arithmetic (that is, to retain the largest and smallest possible values at each step), so that, in the end, we obtain

an interval that contains the true value. Unfortunately, a very small interval may be needed for reasonable implementation.

Rates of Convergence

Since iterative techniques involving sequences are often used, this section concludes with a brief discussion of some terminology used to describe the rate at which convergence occurs. In general, we would like the technique to converge as rapidly as possible. The following definition is used to compare the convergence rates of sequences.

Definition 1.18 Suppose $\{\beta_n\}_{n=1}^{\infty}$ is a sequence known to converge to zero, and $\{\alpha_n\}_{n=1}^{\infty}$ converges to a number α. If a positive constant K exists with

$$|\alpha_n - \alpha| \le K|\beta_n|, \quad \text{for large } n,$$

then we say that $\{\alpha_n\}_{n=1}^{\infty}$ converges to α with **rate, or order, of convergence** $O(\beta_n)$. (This expression is read "big oh of β_n".) It is indicated by writing $\alpha_n = \alpha + O(\beta_n)$. ■

Although Definition 1.18 permits $\{\alpha_n\}_{n=1}^{\infty}$ to be compared with an arbitrary sequence $\{\beta_n\}_{n=1}^{\infty}$, in nearly every situation we use

$$\beta_n = \frac{1}{n^p},$$

for some number $p > 0$. We are generally interested in the largest value of p with $\alpha_n = \alpha + O(1/n^p)$.

Example 2 Suppose that, for $n \ge 1$,

$$\alpha_n = \frac{n+1}{n^2} \quad \text{and} \quad \hat{\alpha}_n = \frac{n+3}{n^3}.$$

Both $\lim_{n\to\infty} \alpha_n = 0$ and $\lim_{n\to\infty} \hat{\alpha}_n = 0$, but the sequence $\{\hat{\alpha}_n\}$ converges to this limit much faster than the sequence $\{\alpha_n\}$. Using five-digit rounding arithmetic we have the values shown in Table 1.7. Determine rates of convergence for these two sequences.

Table 1.7

n	1	2	3	4	5	6	7
α_n	2.00000	0.75000	0.44444	0.31250	0.24000	0.19444	0.16327
$\hat{\alpha}_n$	4.00000	0.62500	0.22222	0.10938	0.064000	0.041667	0.029155

There are numerous other ways of describing the growth of sequences and functions, some of which require bounds both above and below the sequence or function under consideration. Any good book that analyzes algorithms, for example [CLRS], will include this information.

Solution Define the sequences $\beta_n = 1/n$ and $\hat{\beta}_n = 1/n^2$. Then

$$|\alpha_n - 0| = \frac{n+1}{n^2} \le \frac{n+n}{n^2} = 2 \cdot \frac{1}{n} = 2\beta_n$$

and

$$|\hat{\alpha}_n - 0| = \frac{n+3}{n^3} \le \frac{n+3n}{n^3} = 4 \cdot \frac{1}{n^2} = 4\hat{\beta}_n.$$

Hence the rate of convergence of $\{\alpha_n\}$ to zero is similar to the convergence of $\{1/n\}$ to zero, whereas $\{\hat{\alpha}_n\}$ converges to zero at a rate similar to the more rapidly convergent sequence $\{1/n^2\}$. We express this by writing

$$\alpha_n = 0 + O\left(\frac{1}{n}\right) \quad \text{and} \quad \hat{\alpha}_n = 0 + O\left(\frac{1}{n^2}\right). \qquad \blacksquare$$

We also use the O (*big oh*) notation to describe the rate at which functions converge.

Definition 1.19 Suppose that $\lim_{h\to 0} G(h) = 0$ and $\lim_{h\to 0} F(h) = L$. If a positive constant K exists with

$$|F(h) - L| \le K|G(h)|, \quad \text{for sufficiently small } h,$$

then we write $F(h) = L + O(G(h))$. $\blacksquare$

The functions we use for comparison generally have the form $G(h) = h^p$, where $p > 0$. We are interested in the largest value of p for which $F(h) = L + O(h^p)$.

Example 3 Use the third Taylor polynomial about $h = 0$ to show that $\cos h + \frac{1}{2}h^2 = 1 + O(h^4)$.

Solution In Example 3(b) of Section 1.1 we found that this polynomial is

$$\cos h = 1 - \frac{1}{2}h^2 + \frac{1}{24}h^4 \cos\tilde{\xi}(h),$$

for some number $\tilde{\xi}(h)$ between zero and h. This implies that

$$\cos h + \frac{1}{2}h^2 = 1 + \frac{1}{24}h^4 \cos\tilde{\xi}(h).$$

Hence

$$\left|\left(\cos h + \frac{1}{2}h^2\right) - 1\right| = \left|\frac{1}{24}\cos\tilde{\xi}(h)\right| h^4 \le \frac{1}{24}h^4,$$

so as $h \to 0$, $\cos h + \frac{1}{2}h^2$ converges to its limit, 1, about as fast as h^4 converges to 0. That is,

$$\cos h + \frac{1}{2}h^2 = 1 + O(h^4). \qquad \blacksquare$$

Maple uses the O notation to indicate the form of the error in Taylor polynomials and in other situations. For example, at the end of Section 1.1 the third Taylor polynomial for $f(x) = \cos(x)$ was found by first defining

$f := \cos(x)$

and then calling the third Taylor polynomial with

$taylor(f, x = 0, 4)$

Maple responds with

$$1 - \frac{1}{2}x^2 + O(x^4)$$

to indicate that the lowest term in the truncation error is x^4.

EXERCISE SET 1.3

1. **a.** Use three-digit chopping arithmetic to compute the sum $\sum_{i=1}^{10}(1/i^2)$ first by $\frac{1}{1}+\frac{1}{4}+\cdots+\frac{1}{100}$ and then by $\frac{1}{100}+\frac{1}{81}+\cdots+\frac{1}{1}$. Which method is more accurate, and why?

b. Write an algorithm to sum the finite series $\sum_{i=1}^{N} x_i$ in reverse order.

2. The number e is defined by $e=\sum_{n=0}^{\infty}(1/n!)$, where $n!=n(n-1)\cdots 2\cdot 1$ for $n\neq 0$ and $0!=1$. Use four-digit chopping arithmetic to compute the following approximations to e, and determine the absolute and relative errors.

a. $e\approx\sum_{n=0}^{5}\frac{1}{n!}$ **b.** $e\approx\sum_{j=0}^{5}\frac{1}{(5-j)!}$

c. $e\approx\sum_{n=0}^{10}\frac{1}{n!}$ **d.** $e\approx\sum_{j=0}^{10}\frac{1}{(10-j)!}$

3. The Maclaurin series for the arctangent function converges for $-1<x\leq 1$ and is given by

$$\arctan x=\lim_{n\to\infty}P_n(x)=\lim_{n\to\infty}\sum_{i=1}^{n}(-1)^{i+1}\frac{x^{2i-1}}{2i-1}.$$

a. Use the fact that $\tan\pi/4=1$ to determine the number of n terms of the series that need to be summed to ensure that $|4P_n(1)-\pi|<10^{-3}$.

b. The C++ programming language requires the value of π to be within 10^{-10}. How many terms of the series would we need to sum to obtain this degree of accuracy?

4. Exercise 3 details a rather inefficient means of obtaining an approximation to π. The method can be improved substantially by observing that $\pi/4=\arctan\frac{1}{2}+\arctan\frac{1}{3}$ and evaluating the series for the arctangent at $\frac{1}{2}$ and at $\frac{1}{3}$. Determine the number of terms that must be summed to ensure an approximation to π to within 10^{-3}.

5. Another formula for computing π can be deduced from the identity $\pi/4=4\arctan\frac{1}{5}-\arctan\frac{1}{239}$. Determine the number of terms that must be summed to ensure an approximation to π to within 10^{-3}.

6. Find the rates of convergence of the following sequences as $n\to\infty$.

a. $\lim_{n\to\infty}\sin\frac{1}{n}=0$ **b.** $\lim_{n\to\infty}\sin\frac{1}{n^2}=0$

c. $\lim_{n\to\infty}\left(\sin\frac{1}{n}\right)^2=0$ **d.** $\lim_{n\to\infty}[\ln(n+1)-\ln(n)]=0$

7. Find the rates of convergence of the following functions as $h\to 0$.

a. $\lim_{h\to 0}\frac{\sin h}{h}=1$ **b.** $\lim_{h\to 0}\frac{1-\cos h}{h}=0$

c. $\lim_{h\to 0}\frac{\sin h-h\cos h}{h}=0$ **d.** $\lim_{h\to 0}\frac{1-e^h}{h}=-1$

8. **a.** How many multiplications and additions are required to determine a sum of the form

$$\sum_{i=1}^{n}\sum_{j=1}^{i}a_ib_j?$$

b. Modify the sum in part (a) to an equivalent form that reduces the number of computations.

9. Let $P(x)=a_nx^n+a_{n-1}x^{n-1}+\cdots+a_1x+a_0$ be a polynomial, and let x_0 be given. Construct an algorithm to evaluate $P(x_0)$ using nested multiplication.

10. Equations (1.2) and (1.3) in Section 1.2 give alternative formulas for the roots x_1 and x_2 of $ax^2+bx+c=0$. Construct an algorithm with input a,b,c and output x_1, x_2 that computes the roots x_1 and x_2 (which may be equal or be complex conjugates) using the best formula for each root.

11. Construct an algorithm that has as input an integer $n\geq 1$, numbers $x_0,x_1,\ldots,x_n$, and a number x and that produces as output the product $(x-x_0)(x-x_1)\cdots(x-x_n)$.