

2 Iterative solutions and other tools

WHAT NEEDS THIS ITERATION, WOMAN?

— SHAKESPEARE'S *Othello* (1604).

Throughout much of this book, we need only a moderate amount of mathematics, mostly at the level that you learned in high school, with an occasional excursion into first-year college calculus. If you find calculus unfamiliar, or have simply forgotten most of what you learned about it, rest assured that the intent of this book is to use just enough mathematics to understand how to solve the problems that we address. Theorems and proofs and clever tricks in mathematics have their place, but we do not need them in this book.

Most of the functions that we deal with are venerable ones that have received extensive study, and their properties that lead to effective computational algorithms are tabulated in mathematical handbooks, or can be recovered relatively easily with expert systems known as symbolic-algebra packages. We shall see that the more difficult problem is to turn well-known mathematical recipes into computationally stable, fast, and reliable algorithms that can be implemented in computer programs adapted to the limitations of the *finite precision* and *limited range* of floating-point arithmetic.

In this chapter, we review series expansions of functions, and show how they can lead to iterative solutions of equations that often cannot be solved explicitly. Moreover, with a good starting guess for the solution, we shall see that convergence to an accurate answer can be astonishingly fast.

2.1 Polynomials and Taylor series

The computation of some of the elementary functions can sometimes be reduced to finding a root, y , of an equation $f(y) = 0$. For example, suppose that we want to compute the exponential function, $y = \exp(x)$, for a given x , and that we have an accurate implementation of the logarithm. Take the logarithm of each side to get $\ln(y) = x$. Then the y that we seek is the solution of $f(y) = \ln(y) - x = 0$.

If $f(y)$ is a polynomial of degree less than five, then explicit formulas are known for its roots, and you can probably recall from grade-school arithmetic what the roots of linear and quadratic equations are. If you wonder why *five* is significant, see *The Equation That Couldn't Be Solved* [Liv05] for the interesting story of how mathematicians searched for hundreds of years for general formulas for the roots of polynomials, only to have a brilliant young Norwegian mathematician, Niels Henrik Abel,¹ prove in 1826 that no formulas using only addition, subtraction, multiplication, division, and extraction of roots exist for arbitrary polynomials of degree five or higher. An English translation of Abel's paper is available in a collection of mathematical classics [Cal95, pages 539–541].

In most cases of interest for the computation of the elementary functions, $f(y)$ is *not* a polynomial, and the only practical way to find a root is to make a sequence of intelligent guesses, in the hope of converging to that root. Systematic procedures for finding the root can be derived by expanding the function in a Taylor² series (1715), truncating the series to a manageable size, and then solving the now-polynomial equation for a root. We examine that approach further in the next few sections.

The Taylor series of a function $f(y)$ about a point h is given by

$$\begin{aligned} f(y) &= \sum_{n=0}^{\infty} \frac{f^{(n)}(h)}{n!} (y-h)^n \\ &= f(h) + f'(h)(y-h) + f''(h)(y-h)^2/2! + f'''(h)(y-h)^3/3! + \cdots, \end{aligned}$$

¹Niels Henrik Abel (1802–1829) also contributed to the theory of functions, and his name gave the adjective *abelian* used in some areas of mathematics. There is a crater on the Moon named after him.

²Brook Taylor (1685–1731) was an English mathematician who is credited with the *Taylor series* and *Taylor's theorem*. He also worked on the calculus of finite differences and optical refraction.

where $f^{(n)}(h)$ is the n -th derivative (see [Section 4.1](#) on page 61) evaluated at $y = h$. The factorial notation, $n!$, means $n \times (n-1) \times (n-2) \cdots \times 2 \times 1$, with the convention that $0! = (-1)! = 1$. The capital Greek letter *sigma*, Σ , is the summation operator, and is often adorned with lower and upper index limits, written below and above the operator when there is sufficient space, and otherwise, written as subscripts and superscripts. Primes indicate the order of the first few derivatives, and ∞ is the symbol for infinity.

As long as $y - h$ is small and less than one in magnitude, and provided that the derivatives do not grow excessively, the decreasing values of powers of small numbers in the numerators, and the factorially increasing denominators, ensure that the Taylor series converges rapidly. When y is close to h , summing the first few terms of the series is often a computationally effective route to finding the function value. Otherwise, we can sometimes replace the series by a few leading terms, plus a correction term that represents the remainder of the sum, but is easier to compute.

Calculus textbooks discuss the conditions on $f(y)$ for which the Taylor series exists, but all that we need in this book is to know that almost all of the functions that we show how to compute *do* have a well-behaved series. General methods for determining whether infinite series converge (sum to a finite value) or diverge (sum grows without limit) are discussed at length in *Mathematical Methods for Physicists* [AW05, Chapter 5] and [AWH13, Chapter 1].

2.2 First-order Taylor series approximation

As a first approximation, truncate the Taylor series to the first two terms, and solve for the root:

$$\begin{aligned} f(y) &= 0 \\ &\approx f(h) + f'(h)(y - h), \\ y &\approx h - f(h)/f'(h). \end{aligned}$$

Here, the operator \approx means *approximately equal*. That solution suggests an iterative procedure starting with an initial guess, y_0 :

$$y_{n+1} = y_n - f(y_n)/f'(y_n).$$

That formula is so important that we highlight it in bold type. It is known as the *Newton–Raphson method*, from work by Isaac Newton³ in 1665 and Joseph Raphson in 1690, or sometimes just the *Newton method*. It predates the Taylor series, because it was originally derived by graphical arguments and binomial expansions. Special cases of the iteration for finding square roots and cube roots were known to the Babylonians (ca. 1900 BCE), and independently, in India (ca. 1000 BCE) and China (ca. 100 BCE).

Ypma [Ypm95] gives an interesting historical review of the Newton–Raphson method, and suggests that Newton–Raphson–Simpson is a more accurate name, considering the later contributions of Thomas Simpson (1710–1761).

With a sufficiently good starting guess, Newton–Raphson convergence is *quadratic*, doubling the number of correct digits each iteration. Importantly, the iterations are *self-correcting*: computational errors made in one iteration do not carry over to the next iteration, because that iteration is just a fresh start with an improved guess.

The drawbacks of the Newton–Raphson iteration are that it does not necessarily converge if the initial guess is too far from the root, and that it requires evaluation of both the function and its first derivative, which might be computationally expensive, or even impossible.

If the first derivative is replaced by an approximate numerical derivative, $f'(y_n) \approx (f(y_n) - f(y_{n-1})) / (y_n - y_{n-1})$, the iteration is called the *secant method*, and its convergence is less than quadratic: the error decreases by powers of the *golden ratio*, $\phi = (1 + \sqrt{5})/2 \approx 1.618$. Here, ϕ is the Greek letter *phi* that is commonly used for that ratio. The story of that remarkable number, which turns up in many different areas of mathematics, is told in *A Mathematical History of Golden Number* [HF98] and *The Golden Ratio* [Liv02].

³The English scientist Isaac Newton (1643–1727) of Cambridge University is one of the most famous in history, with important contributions to astronomy, mathematics, optics, philosophy, and physics. He and the German polymath Gottfried Leibniz (1646–1716) are jointly credited with the development of calculus; see *The Calculus Wars* [Bar06] for the interesting history of that work. Newton’s 1687 masterpiece book *Philosophiæ Naturalis Principia Mathematica* described gravitation and the laws of motion of classical mechanics. He invented the reflecting telescope, which gave a three-fold increase in magnification. He was President of the Royal Society for more than two decades, and in his last 30 years, Newton was Warden and Master of the Royal Mint, charged with improving coinage and eliminating counterfeiting [Lev09]. Leibniz invented the binary number system, and about 1671, designed the first mechanical calculator that could add, subtract, multiply, and divide.

Despite the drawbacks, Newton–Raphson iteration is widely used in numerical computations, and we refer to it often in this book. As long as we have an accurate starting value, the first-order iteration is an effective computational technique for many of the functions that we treat. Nevertheless, it can sometimes be useful to consider more complicated algorithms that we derive in the next two sections.

2.3 Second-order Taylor series approximation

The next approximation for finding a solution of $f(y) = 0$ is to include one more term of the Taylor series, and then regroup to form a *quadratic equation*:

$$\begin{aligned} f(y) &= 0 \\ &\approx f(h) + f'(h)(y - h) + f''(h)(y - h)^2/2 \\ &\approx Ay^2 + By + C. \end{aligned}$$

The coefficients are given by these formulas:

$$\begin{aligned} A &= f''(h)/2, \\ B &= f'(h) - f''(h)h, \\ C &= f(h) - f'(h)h + (f''(h)/2)h^2. \end{aligned}$$

The solution of the quadratic equation is the desired value, y . Developing a computer program for the robust determination of the roots of such equations from schoolbook formulas is more difficult than would appear, however, so we delay treatment of that problem until **Section 16.1** on page 465.

When the second derivative is zero, we have $A = 0$, and $y = -C/B = h - f(h)/f'(h)$, which is just the Newton–Raphson formula, as it must be, because it corresponds to dropping the third and higher terms of the Taylor series.

2.4 Another second-order Taylor series approximation

Another way to handle the three-term truncated Taylor series is to partly solve for y :

$$\begin{aligned} 0 &\approx f(h) + f'(h)(y - h) + f''(h)(y - h)^2/2! \\ &\approx f(h) + (y - h)[f'(h) + f''(h)(y - h)/2], \\ y &\approx h - f(h)/[f'(h) + f''(h)(y - h)/2]. \end{aligned}$$

Although that form does not immediately seem helpful, replacing y on the right-hand side by the value predicted by the Newton–Raphson iteration, $h - f(h)/f'(h)$, produces this result:

$$\begin{aligned} y &\approx h - f(h)/[f'(h) + f''(h)((h - f(h)/f'(h)) - h)/2] \\ &\approx h - f(h)/[f'(h) - f''(h)(f(h)/f'(h))/2] \\ &\approx h - 2f(h)f'(h)/[2(f'(h))^2 - f(h)f''(h)]. \end{aligned}$$

That suggests an iterative scheme that we highlight like this:

$$y_{n+1} = y_n - 2f(y_n)f'(y_n)/[2(f'(y_n))^2 - f(y_n)f''(y_n)]$$

That is called *Halley's method*, after Edmund Halley (1656–1742), whose name is also attached to a famous comet that orbits our Sun with a period of approximately 76 years, and was last close to the Earth in 1986. When the second derivative is zero, the method reduces to the Newton–Raphson iteration. We show an application of Halley's method in **Section 8.3** on page 227.

Taylor series of tan:

$$x + \frac{1}{3}x^3 + \frac{2}{15}x^5 + \frac{17}{315}x^7 + O(x^9)$$

Taylor series of cot:

"does not have a taylor expansion, try series()"

Taylor series of arcsin:

$$x + \frac{1}{6}x^3 + \frac{3}{40}x^5 + \frac{5}{112}x^7 + O(x^9)$$

Taylor series of arccos:

$$\frac{\pi}{2} - x - \frac{1}{6}x^3 - \frac{3}{40}x^5 - \frac{5}{112}x^7 + O(x^9)$$

Taylor series of arctan:

$$x - \frac{1}{3}x^3 + \frac{1}{5}x^5 - \frac{1}{7}x^7 + O(x^9)$$

Taylor series of sinh:

$$x + \frac{1}{6}x^3 + \frac{1}{120}x^5 + \frac{1}{5040}x^7 + O(x^9)$$

Taylor series of cosh:

$$1 + \frac{1}{2}x^2 + \frac{1}{24}x^4 + \frac{1}{720}x^6 + \frac{1}{40320}x^8 + O(x^9)$$

Taylor series of tanh:

$$x - \frac{1}{3}x^3 + \frac{2}{15}x^5 - \frac{17}{315}x^7 + O(x^9)$$

In those expansions, the *big-oh* notation, $\mathcal{O}(x^n)$, is read *order of* x^n . It means that the magnitude of the first omitted term is bounded by $|cx^n|$, where c is a constant, x is arbitrary, and the vertical bars mean absolute value.

We have already suggested that the square-root function can be computed from the Newton–Raphson iteration, so the fact that Maple is unable to report its Taylor series does not matter. Because Cody and Waite effectively compute $\cot(x)$ from $1/\tan(x)$, Maple's inability to find a Taylor series for $\cot(x)$ does not matter either. Maple can still provide a series for it, like this:

```
> Order := 9;
```

```
> series(cot(x), x = 0);
```

$$x^{-1} - \frac{1}{3}x - \frac{1}{45}x^3 - \frac{2}{945}x^5 - \frac{1}{4725}x^7 + O(x^9)$$

Thus, we can move the reciprocal term to the left, and get a Taylor series like this:

```
> taylor(cot(x) - 1/x, x = 0, 9);
```

$$- \frac{1}{3}x - \frac{1}{45}x^3 - \frac{2}{945}x^5 - \frac{1}{4725}x^7 + O(x^9)$$

The logarithm goes to negative infinity as $x \rightarrow +0$, so it is not surprising that there is no Taylor series there. Instead, we just ask for the series near $x = 1$:

```
> taylor(log(x), x = 1, 9);
```

$$x - 1 - \frac{1}{2}(x - 1)^2 + \frac{1}{3}(x - 1)^3 - \frac{1}{4}(x - 1)^4 + \frac{1}{5}(x - 1)^5 - \frac{1}{6}(x - 1)^6 + \frac{1}{7}(x - 1)^7 - \frac{1}{8}(x - 1)^8 + 0((x - 1)^9)$$

Notice that $\exp()$ and $\log()$ are the only ones with both even and odd powers in the Taylor series. All of the others have only odd terms, or only even terms. Cody and Waite exploit that by rearranging the series to get the first one or two terms exactly, and then using a low-order polynomial approximation to represent the remaining infinite series with accuracy sufficient for the machine precision.

2.7 Continued fractions

THE THEORY OF CONTINUED FRACTIONS DOES NOT RECEIVE
THE ATTENTION THAT IT DESERVES.

— ANDREW M. ROCKETT AND PETER SZÜSZ (1992)

Mathematical handbooks often tabulate function expansions known as *continued fractions*. They take the recursive form $F(x) = \text{constant} + a/(b + c)$, where c itself is a fraction $d/(e + f)$, and so on. At least one of the terms in each numerator or denominator contains x . Such a fraction can be written out like this:

$$F(x) = b_0 + \frac{a_1}{b_1 + \frac{a_2}{b_2 + \frac{a_3}{b_3 + \frac{a_4}{b_4 + \frac{a_5}{b_5 + \frac{a_6}{b_6 + \frac{a_7}{b_7 + \frac{a_8}{b_8 + \dots}}}}}}}$$

Because that display takes a lot of space, a more compact notation is common:

$$F(x) = b_0 + \frac{a_1}{b_1 +} \frac{a_2}{b_2 +} \frac{a_3}{b_3 +} \frac{a_4}{b_4 +} \frac{a_5}{b_5 +} \frac{a_6}{b_6 +} \frac{a_7}{b_7 +} \frac{a_8}{b_8 +} \dots$$

It is not immediately evident how to evaluate a continued fraction, because each denominator involves another denominator whose value we do not yet know. There are two common ways to do the evaluation:

- In *backward evaluation*, we start at term n and arbitrarily set $a_{n+1} = 0$, cutting off the infinite fraction. The resulting finite fraction, denoted $F_n(x)$, is called the n -th *convergent* of the infinite continued fraction.

In practice, we are interested in a particular range of x , and numerical experiments over that range, and with increasing values of n , let us determine how many terms are needed to achieve the desired accuracy. Aside from the evaluation of each of the terms a_k and b_k ($k = 1, 2, \dots, n$), the computation requires n adds and n divides.

- From the structure of successive convergents, Euler derived in 1737 a recurrence relation that allows *forward evaluation*. It takes the form

$$\begin{aligned} A_{-1} &= 1, & B_{-1} &= 0, \\ A_0 &= 0, & B_0 &= 1, \\ & & \text{for } k = 1, 2, \dots, n, \text{ compute} \\ A_k &= b_k A_{k-1} + a_k A_{k-2}, & B_k &= b_k B_{k-1} + a_k B_{k-2}, \\ & & F_n(x) &= b_0 + A_n / B_n. \end{aligned}$$

Apart from the terms a_k and b_k , the computation requires $2n + 1$ adds and $4n$ multiplies, but just *one* divide.

It sometimes happens that either $a_k = 1$ or $b_k = 1$, in which case, the multiplies can be reduced from $4n$ to $2n$.

Arrays of dimension n are *not* required, because only three successive entries of each of A and B are needed at a time. Instead, they can each be stored in three scalars whose values are shuffled down on each iteration.

On most modern machines, division is comparatively expensive, often costing the time for 3 to 15 multiplications. Thus, even though the operation count is smaller for backward evaluation, it might be more expensive than forward evaluation because of the $n - 1$ extra divides. However, the code for backward evaluation is simpler to program than that for forward evaluation, and has fewer memory references. On modern systems, memory traffic is costly, with a load or store taking up to 500 times longer than a floating-point copy inside the CPU. If the value is available in the intermediate memory cache, its access time may be much lower, of the order of 3 to 10 times slower than an in-CPU copy. A good optimizing compiler, and a register-rich architecture, may make it possible for many of the scalars to be held in CPU registers,⁴ eliminating most of the expensive memory accesses.

With backward evaluation, the number of terms, n , is fixed in advance by numerical experiment, even though for some x values, fewer terms would suffice. With series evaluation, as long as successive terms decrease in magnitude, it is possible to exit the summation loop as soon as the term just added does not change the sum. With forward evaluation of a continued fraction, early loop exit is possible as well, as soon as $\text{fl}(F_n(x)) = \text{fl}(F_{n-1}(x))$ to machine precision. Here, $\text{fl}(\text{expr})$ means the floating-point computation of expr : it is not a library function, but rather, an indicator of a transition from mathematics to finite-precision arithmetic. Unfortunately, rounding errors in the computed convergents may prevent that equality from ever being satisfied, so a loop limit is still needed, and early exit cannot be guaranteed.

In *exact* arithmetic, both forward and backward evaluation produce identical values of $F_n(x)$. However, error analysis for the floating-point computation of continued fractions [Bla64, JT74] shows that backward evaluation is usually numerically more stable than forward evaluation. Also, even though the convergents may be of modest magnitude, the terms A_k and B_k may become so large, or so small, that they can no longer be represented in the limited exponent range of a floating-point number. It is then necessary to introduce intermediate rescaling of A_k and B_k , complicating the computation significantly. Alternate forms of forward evaluation exist that eliminate the need for scaling, but they increase the work in each iteration [GST07, pages 181–185]. We present them in **Section 2.8** on page 17.

In summary, it is usually not possible to predict which of the two evaluation methods is faster, except by making numerical experiments with both, and the results of the experiments depend on the CPU, on the compiler, and on the compiler's optimization options. Error analysis favors backward evaluation, unless the speed penalty is too severe.

Because of those difficulties, continued fractions are often not even mentioned in books on numerical analysis, and they are not commonly used in computer algorithms for the evaluation of elementary functions. Nevertheless, they can lead to independent implementations of elementary functions that are of use in software testing, because agreement of function values computed by completely different algorithms makes software blunders unlikely. In addition, continued fractions have three advantages over conventional series representations:

- The domain of convergence (that is, the allowable argument range) of a continued fraction of an elementary function may be much larger than that for its Taylor series.
- Fewer terms may be needed to achieve a given level of accuracy with continued fractions than with Taylor series.
- Little code is needed to compute a continued fraction, especially by backward evaluation, making it easier to get it right on the first try.

The special case where the numerators of the continued fraction are all equal to one, $a_k = 1$, is called a *simple continued fraction*, and is sometimes given a separate notation as a bracketed list of coefficients:

$$F(x) = b_0 + \frac{1}{b_1 + \frac{1}{b_2 + \frac{1}{b_3 + \frac{1}{b_4 + \frac{1}{b_5 + \frac{1}{b_6 + \frac{1}{b_7 + \frac{1}{b_8 + \dots}}}}}}}} \\ = [b_0; b_1, b_2, b_3, b_4, b_5, b_6, b_7, b_8, \dots].$$

⁴A register is a storage area inside the CPU that provides much faster data access than external memory does. There are usually separate register sets for integer and floating-point arithmetic, and sometimes additional sets for status flags. Most CPU designs since the mid-1980s have 16 to 128 floating-point registers, sometimes with particular registers defined to be read-only, and hardwired to hold the commonly needed constants 0.0 and 1.0.

Some books, and the Maple symbolic-algebra system, use a comma instead of a semicolon after the first list entry.

All rational numbers can be uniquely written as *finite* simple continued fractions with integer coefficients, provided that, if the last coefficient is 1, it is discarded and absorbed into the next-to-last coefficient:

$$[b_0; b_1, b_2, b_3, \dots, b_n, 1] \rightarrow [b_0; b_1, b_2, b_3, \dots, b_n + 1].$$

Every *irrational*⁵ number has a unique *infinite* simple continued fraction.

Four properties of simple continued fractions are of interest for computer arithmetic:

$$\begin{aligned} 1/F(x) &= \begin{cases} [0; b_0, b_1, b_2, b_3, b_4, b_5, b_6, b_7, b_8, \dots], & \text{if } b_0 \neq 0, \\ [b_1; b_2, b_3, b_4, b_5, b_6, b_7, b_8, \dots], & \text{if } b_0 = 0, \end{cases} \\ c + F(x) &= [c + b_0; b_1, b_2, b_3, b_4, b_5, b_6, b_7, b_8, \dots], & \text{for constant } c, \\ cF(x) &= [cb_0; b_1/c, cb_2, b_3/c, cb_4, b_5/c, cb_6, b_7/c, cb_8, \dots], \\ -F(x) &= [-b_0; -b_1, -b_2, -b_3, -b_4, -b_5, -b_6, -b_7, -b_8, \dots], \\ &= [-b_0 - 1; 1, b_1 - 1, b_2, b_3, b_4, b_5, b_6, b_7, b_8, \dots]. \end{aligned}$$

That is, forming the reciprocal of a simple continued fraction requires just a left or right shift in the coefficient list, adding a constant affects only the first coefficient, and scaling a simple continued fraction by a constant alternately multiplies and divides the coefficients by that constant. That scaling is particularly easy, and exact, if c is a power of the floating-point base. Negation is a special case of scaling, and can be done either by negating all of the coefficients, or else by altering the first two as indicated, and inserting a 1 between them.

We can illustrate those properties for particular x values in Maple, whose conversion function takes an additional argument that gives the number of coefficients to be reported. Some of these examples use the Greek letter π , ω , π , the famous ratio of the circumference of a circle to its diameter [Bar92, Bec93, Bar96, Bla97, BBB00, AH01, EL04b, Nah06], and others involve the *golden ratio* [HF98, Liv02]:

```
% maple
> convert(Pi, confrac, 15);
[3, 7, 15, 1, 292, 1, 1, 1, 2, 1, 3, 1, 14, 2, 1]

> convert(1 / Pi, confrac, 15);
[0, 3, 7, 15, 1, 292, 1, 1, 1, 2, 1, 3, 1, 14, 2]

> convert(2 * Pi, confrac, 15);
[6, 3, 1, 1, 7, 2, 146, 3, 6, 1, 1, 2, 7, 5, 5]

> convert(-Pi, confrac, 15);
[-4, 1, 6, 15, 1, 292, 1, 1, 1, 2, 1, 3, 1, 14, 2]

> convert(Pi - 3, confrac, 15);
[0, 7, 15, 1, 292, 1, 1, 1, 2, 1, 3, 1, 14, 2, 1]

> convert(1 / (Pi - 3), confrac, 15);
[7, 15, 1, 292, 1, 1, 1, 2, 1, 3, 1, 14, 2, 1, 1]

> phi := (1 + sqrt(5)) / 2:      # the celebrated golden ratio

> convert(phi, confrac, 15);
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

⁵An irrational number is a number that cannot be represented as the ratio of two whole numbers. For example, about 500BC, the Greeks proved by geometrical arguments that $\sqrt{2}$ is irrational. We can show that with simple algebra. Assume that $\sqrt{2}$ is rational, so we can write $\sqrt{2} = p/q$, where p and q are whole numbers that have no factors in common. That means that at least one of those numbers must be odd, for if they were both even, they would have a factor 2 in common. Square the equation and rearrange to find $2q^2 = p^2$. That means that p^2 is even, and because only even numbers have even squares, p must be even. Thus, we can replace it by $2r$, where r is another integer. We then have $2q^2 = 4r^2$, or $q^2 = 2r^2$, so q^2 , and thus, q , must be even. However, having both p and q even contradicts the original assertion, which must therefore be false. Thus, $\sqrt{2}$ must be irrational.


```

> convert(1 / phi, confrac, 15);
[0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

> convert(2 * phi, confrac, 15);
[3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4]

> convert(-phi, confrac, 15);
[-2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

```

Notice in each case that the first coefficient in the simple continued fraction of x is just the nearest integer that is not greater than x , a function that we treat later in [Section 6.7](#) on page 136.

The particularly simple continued fraction for the golden ratio is due to its definition as the solution of $\phi = 1 + 1/\phi$: repeated substitutions of ϕ on the right-hand side show that $b_k = 1$ for all $k \geq 0$. Integral powers of the golden ratio have a surprising closed form: $\phi^n = F_n\phi + F_{n-1}$, where the F_n are the famous *Fibonacci numbers* introduced in 1202 CE [Sig02, Bee04b, PL07, Dev11] in the same book that brought the digit zero to the Western World, and introduced the words *algorithm* and *algebra*, derived from the name of a Ninth Century Persian mathematician and that of his book. Continued fractions of odd powers of ϕ have constant b_k values, whereas those of even powers of ϕ have only two different b_k values after b_0 :

```

> for k from 1 to 20 by 2 do
>   printf("%2d %30a %2d %30a\n", k, convert(phi^k, confrac, 5),
>                                     k+1, convert(phi^(k+1), confrac, 5))
> end do:
1           [1, 1, 1, 1, 1] 2           [2, 1, 1, 1, 1]
3           [4, 4, 4, 4, 4] 4           [6, 1, 5, 1, 5]
5           [11, 11, 11, 11, 11] 6           [17, 1, 16, 1, 16]
7           [29, 29, 29, 29, 29] 8           [46, 1, 45, 1, 45]
9           [76, 76, 76, 76, 76] 10          [122, 1, 121, 1, 121]
11          [199, 199, 199, 199, 199] 12          [321, 1, 320, 1, 320]
13          [521, 521, 521, 521, 521] 14          [842, 1, 841, 1, 841]
15 [1364, 1364, 1364, 1364, 1364] 16          [2206, 1, 2205, 1, 2205]
17 [3571, 3571, 3571, 3571, 3571] 18          [5777, 1, 5776, 1, 5776]
19 [9349, 9349, 9349, 9349, 9349] 20          [15126, 1, 15125, 1, 15125]

```

The Fibonacci numbers arise in a problem of rabbit populations, and satisfy the recurrence $F_{n+2} = F_{n+1} + F_n$, with starting conditions $F_0 = 0$ and $F_1 = 1$. As n increases, the ratio F_{n+1}/F_n approaches the limit of $\phi \approx 1.618$, showing that unchecked population growth is exponential. The Fibonacci numbers show up in an astonishing variety of places in mathematics, and even in vegetable gardens, where some plants exhibit the Fibonacci number sequence in their growth patterns. An entire journal, the *Fibonacci Quarterly*,⁶ is devoted to their study, which still remains active, more than 800 years after they were first discovered.

Continued fractions for irrational numbers have the nice property of providing the *best rational approximation*, in the sense that if t is irrational, its k -th convergent is the rational number a/b , where a and b have no common factors, and no other rational number c/d with $|d| < |b|$ is closer to t . All floating-point numbers are rational numbers, and that property can be exploited to find function arguments x for which $f(x)$ is almost exactly halfway between two adjacent floating-point numbers. Such arguments provide challenging tests for software that attempts to guarantee correct rounding of function values. They also offer information about how much numerical precision is needed for argument reduction, an important topic that we treat in [Chapter 9](#).

Successive convergents of irrational numbers are alternately lower and upper bounds. For example, some of the approximations derived from the continued fraction for π may be familiar from schoolbooks:

$$\begin{aligned}
 \pi_0 &= 3 \\
 &\approx \pi - 0.1416, \\
 \pi_1 &= 3 + 1/7 \\
 &= 22/7
 \end{aligned}$$

⁶See <http://www.math.utah.edu/pub/tex/bib/index-table-f.html#fibquart>.

$$\begin{aligned}
&\approx \pi + 1.264 \times 10^{-3}, \\
\pi_2 &= 3 + 1/(7 + 1/15) \\
&= 333/106 \\
&\approx \pi - 8.322 \times 10^{-5}, \\
\pi_3 &= 3 + 1/(7 + 1/(15 + 1/1)) \\
&= 355/113 \\
&\approx \pi + 2.668 \times 10^{-7}.
\end{aligned}$$

With an additional argument to record the convergents, the Maple `convert()` function can easily produce many more approximations to π :

```

> convert(Pi, confrac, 15, cvgts):
> cvgts;
      333  355  103993  104348  208341  312689  833719
[3, 22/7, ---, ---, ---, ---, ---, ---, ---,
 106  113  33102   33215   66317   99532   265381

 1146408  4272943  5419351  80143857  165707065  245850922
-----, -----, -----, -----, -----, -----]
364913    1360120  1725033  25510582  52746197   78256779

```

We can then check the bounding and convergence properties like this:

```

> for k to 15 do printf("%2d % .3e\n", k, op(k,cvgts) - Pi) end do;
1  -1.416e-01
2   1.264e-03
3  -8.322e-05
4   2.668e-07
5  -5.779e-10
6   3.316e-10
7  -1.224e-10
8   2.914e-11
9  -8.715e-12
10  1.611e-12
11 -4.041e-13
12  2.214e-14
13 -5.791e-16
14  1.640e-16
15 -7.820e-17

```

Some common irrational or *transcendental*⁷ constants have easy-to-program simple continued fractions that allow them to be computed at run time to arbitrary precision without the need to retrieve their coefficients from tables. Here we generate them in Mathematica, which displays simple continued fractions as braced lists:

```

% math
In[1]:= ContinuedFraction[Sqrt[2], 15]
Out[1]= {1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2}

In[2]:= ContinuedFraction[Sqrt[3], 15]
Out[2]= {1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2}

In[3]:= ContinuedFraction[(1 + Sqrt[5])/2, 15]
Out[3]= {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1}

```

⁷A transcendental number is an irrational number that cannot be the root of any finite polynomial with integer coefficients. The transcendental property of e was proved by Hermite in 1873, and of π by von Lindemann in 1882. Most numbers on the real axis are transcendental, but there are many open problems. For example, we do not know whether $e + \pi$, e^π , or π^e are transcendental, and we cannot yet prove that the digits of π are random.

```
In[4]:= ContinuedFraction[Exp[1], 18]
Out[4]= {2, 1, 2, 1, 1, 4, 1, 1, 6, 1, 1, 8, 1, 1, 10, 1, 1, 12}

In[5]:= ContinuedFraction[Tan[1/2], 18]
Out[5]= {0, 1, 1, 4, 1, 8, 1, 12, 1, 16, 1, 20, 1, 24, 1, 28, 1, 32}
```

Binary floating-point numbers can be converted to and from simple continued fractions with practical hardware, and continued fractions for the computation of certain elementary functions have attracted the interest of a few hardware designers [FO01, Chapter 12].

Continued fractions are not just for evaluation of irrational numbers, however. One of the oldest-known continued fractions is that for the trigonometric tangent function (Lambert, about 1760), which we treat later in [Section 11.2](#) on page 302:

$$\begin{aligned} \tan(x) &= 0 + \frac{x}{1 + \frac{x^2}{-3 + \frac{x^2}{5 + \frac{x^2}{-7 + \frac{x^2}{9 + \frac{x^2}{-11 + \dots}}}}} \quad x \neq (2n+1)\pi/2, \\ b_0 &= 0, \quad b_k = (-1)^{k+1}(2k-1), \quad \text{for } k > 0, \\ a_1 &= x, \quad a_k = x^2, \quad \text{for } k > 1. \end{aligned}$$

The coefficients are much simpler than those of the Taylor series, and convergence is rapid for small x . For example, with $x = 1/100$, $\tan(x)$ and the truncated continued fraction $x/(1 + x^2/(-3 + x^2/(5 + x^2/(-7))))$ agree to better than 21 decimal digits. Lambert was able to prove that if x is a nonzero rational number, then $\tan(x)$ is irrational; thus, because $\tan(\pi/4) = 1$, $\pi/4$ and π cannot be rational, so they must be irrational.

2.8 Summation of continued fractions

We promised in the preceding section to show alternative ways of computing continued fractions from forward summations that allow early exit on convergence.

The first of those handles the particular case of a continued fraction where all but the first of the b_k are equal to one, and there are subtractions, instead of additions, in the denominator:

$$F(x) = b_0 + \frac{a_0}{1 - \frac{a_1}{1 - \frac{a_2}{1 - \frac{a_3}{1 - \frac{a_4}{1 - \frac{a_5}{1 - \frac{a_6}{1 - \frac{a_7}{\dots}}}}}}}$$

A fraction of that type occurs in the incomplete gamma function that we consider later in this book in [Section 18.4](#) on page 560. The continued fraction can be evaluated in the forward direction as a sum with easily computed term recurrences:

$$\begin{aligned} F(x) &= b_0 + \sum_{k=0}^{\infty} t_k, \\ r_0 &= 0, \quad t_0 = a_0, \\ r_k &= \frac{a_k(1 + r_{k-1})}{1 - a_k(1 + r_{k-1})}, \quad t_k = r_k t_{k-1}, \quad k > 0. \end{aligned}$$

The n -th partial sum of the terms t_k , plus b_0 , is mathematically identical to the n -th convergent of the continued fraction, $F_n(x)$.

The second of those summation formulas was discovered by a mathematical physics researcher [BFSG74], and is called *Steed's algorithm*. A textbook description is available elsewhere [GST07, pages 181–183], so we merely exhibit a code template that can be readily adapted in just *six statements* to evaluate any particular continued fraction:

```
static fp_t
eval_cf_steed(fp_t x)
{
    fp_t a1, ak, b0, b1, bk, Ck, Ckm1, Ckm2, Dk, Dkm1, Ek, Ekm1;
    int k;
```

```

b0 =                                /* TO BE SUPPLIED */;
b1 =                                /* TO BE SUPPLIED */;
a1 =                                /* TO BE SUPPLIED */;
Ckm2 =                              /* TO BE SUPPLIED */;
Dkm1 = ONE / b1;
Ekm1 = a1 * Dkm1;
Ckm1 = Ckm2 + Ekm1;

for (k = 2; k <= MAXITER; ++k)
{
    ak =                            /* TO BE SUPPLIED */;
    bk =                            /* TO BE SUPPLIED */;
    Dk = ONE / (Dkm1 * ak + bk);
    Ek = (bk * Dk - ONE) * Ekm1;
    Ck = Ckm1 + Ek;

    if (Ck == Ckm1)                 /* converged */
        break;

    Ckm1 = Ck;
    Dkm1 = Dk;
    Ekm1 = Ek;
}

return (Ck);
}

```

The value of MAXITER limits the number of iterations, but its value needs to be determined by numerical experiment for each particular choice of coefficients, and for each numerical precision.

Steed's algorithm implicitly includes scaling of intermediate terms to reduce the likelihood of out-of-range numbers. Nevertheless, numerical experiments must verify that.

If the denominator $D_{km1} * ak + bk$ in any iteration can become tiny, or zero, Steed's algorithm may be unsatisfactory. That brings us to the final recipe in this section, the *modified Lentz algorithm* [TB86]. It handles rescaling differently, and may be less sensitive to small denominators, although it requires a fudge factor that may need adjustment for a particular continued fraction. A textbook treatment provides some background [GST07, pages 183–185], so once again, we just present a code template:

```

static fp_t
eval_cf_lentz(fp_t x)
{
    fp_t ak, b0, bk, Ck, Ckm1, Dk, Dkm1, Ek, Ekm1, eps, Hk;
    int k;

    b0 =                            /* TO BE SUPPLIED */;
    eps = FP(1.0e10) / FP_T_MAX; /* may need to fudge the numerator */
    Ckm1 = b0;

    if (b0 == ZERO)
        Ckm1 = eps;

    Ekm1 = Ckm1;
    Dkm1 = ZERO;

    for (k = 1; k <= MAXITER; ++k)
    {
        ak =                        /* TO BE SUPPLIED */;
        bk =                        /* TO BE SUPPLIED */;
        Dk = bk + ak * Dkm1;

```

```

    if (Dk == ZERO)
        Dk = eps;

    Ek = bk + ak / Ekm1;

    if (Ek == ZERO)
        Ek = eps;

    Dk = ONE / Dk;
    Hk = Ek * Dk;
    Ck = Ckm1 * Hk;

    if ((ONE + (Hk - ONE)) == ONE) /* converged */
        break;

    Ckm1 = Ck;
    Dkm1 = Dk;
    Ekm1 = Ek;
}

return (Ck);
}

```

Here, `FP_T_MAX` is the largest representable floating-point number, and `eps`, which replaces zero values that later appear as divisors, must be chosen as a tiny value whose reciprocal is still a finite floating-point number. The value `FP(1.0e10)` may therefore need to be adjusted for some historical floating-point systems. As with Steed's algorithm, the iteration limit `MAXITER` must be suitably adjusted for each application.

The three summation algorithms given in this section are not yet widely known, and are absent from the few textbook treatments of numerical evaluation of continued fractions. The only other textbook presentations of the Steed and Lentz algorithms known to this author are recent editions of the *Numerical Recipes* series [PTVF07, pages 206–209].

In some later chapters of this book, we use those summation algorithms for evaluation of continued fractions, but we do not repeat their code, because only the few lines that define the coefficients a_k and b_k need to be altered.

To learn more about the mathematics of continued fractions, see a recent reprint of the original 1948 edition of Wall's classic book, *Analytic Theory of Continued Fractions* [Wal00], two short monographs [RS92, KE97], and six advanced books [JT84, Bre91, LW92, Hen06, Khr08, LW08]. The second last of those books shows interesting relations of continued fractions to the computation of mathematical constants, to the design of accurate calendar systems, and to the construction of Eastern and Western musical scales.

For their application to some of the functions covered in this book, see the *Handbook of Continued Fractions for Special Functions* [CPV⁺08]. The authors of that book used symbolic-algebra systems to verify all of their formulas, and in doing so, found and corrected many errors in earlier books and research articles. Computers cannot replace, but can often help, mathematicians.

2.9 Asymptotic expansions

Functions can sometimes be formally represented by a series in inverse powers of the form

$$f(x) \asymp c_0 + c_1/x + c_2/x^2 + \cdots + c_n/x^n + \cdots.$$

Here, the relational operator \asymp indicates an *asymptotic expansion*. It has the property that partial sums to arbitrarily high order *diverge*. However, the first few terms decrease in magnitude, and then increase. A partial sum up to the term of smallest magnitude differs from $f(x)$ by an error that is about the size of the first term omitted.

There is a long mathematical history of asymptotic expansions and their use in the evaluation of elementary and special functions; see, for example, *Asymptotics and Special Functions* [Olv74].

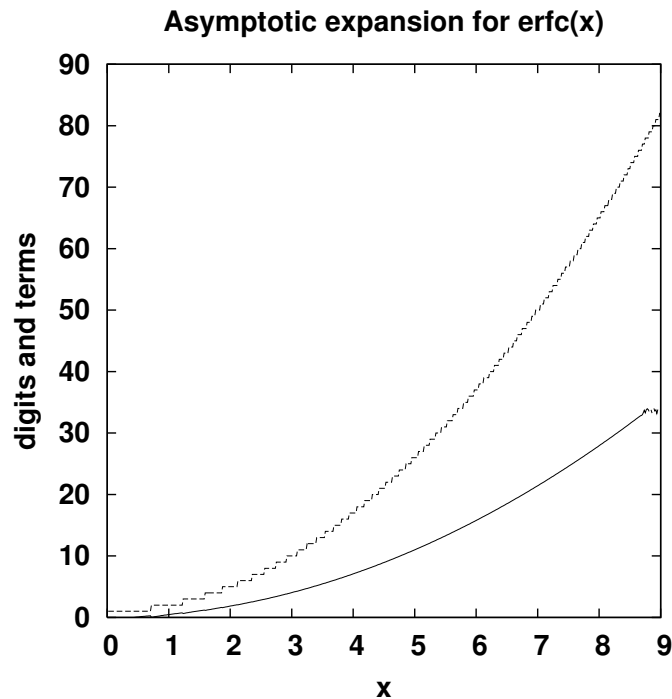


Figure 2.1: Accuracy of the asymptotic expansion for $\text{erfc}(x)$. The solid lower curve shows the number of correct decimal digits produced in 128-bit IEEE 754 arithmetic, and the dashed upper curve shows the number of terms summed.

The problem with asymptotic expansions is that they cannot be used to evaluate a function to arbitrary accuracy just by summing more and more terms. Instead, evaluation must stop as soon as the magnitude of the next term increases. For arguments of large magnitude, high accuracy may be possible, but for arguments of small magnitude, accuracy drops. **Figure 2.1** illustrates that phenomenon for the asymptotic expansion of a function that we treat later in **Section 19.1** on page 593.

In numerical software that is designed to produce high accuracy for a range of floating-point precisions from a common code base, as we do in the `mathcw` library, asymptotic expansions are usually inadvisable. Nevertheless, they can be useful in software development and testing, by providing an alternate, and independent, computational route to a function, just as we saw for continued fractions. For numerical work, they can sometimes be replaced by shorter polynomial approximations, a topic that we address in **Chapter 3**.

Despite the caveat in the preceding paragraph, we use asymptotic expansions later in this book for large-argument computation of a few functions, including gamma, log-gamma, and psi functions (**Chapter 18**), the cumulative distribution functions for the standard normal distribution (**Section 19.4**), and some of the Bessel functions (**Chapter 21**). In each case, in the regions for which the asymptotic series are summed, the arguments are big enough that full machine precision is attainable.

2.10 Series inversion

If we have a representation of a function as the series

$$f(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + \dots$$

with known coefficients a_k , then for a given x , we can straightforwardly evaluate the right-hand side to find the function value.

However, suppose that we know the value of the left-hand side, $y = f(x)$, and we wish to determine the x value that produces y . That problem is known as *series inversion*. That is, we seek an expansion

$$x = b_0 + b_1y + b_2y^2 + b_3y^3 + \cdots$$

where the coefficients b_k are to be determined. To find them, substitute the expansion of x in the series for y , and then rearrange the result to find coefficients of powers of y . The result is an equation of the form

$$y = [\text{mess}]_0 + [\text{bad mess}]_1y + [\text{horrid mess}]_2y^2 + [\text{ghastly mess}]_3y^3 + \cdots$$

Because that must hold for arbitrary y , all but one of the right-hand side coefficients must be zero, and the remaining factor, $[\text{bad mess}]_1$, must be one. Each coefficient therefore leads to an equation, and those equations can be solved in the forward direction starting with the zeroth, $[\text{mess}]_0 = 0$.

Series inversion is tedious to work out by hand, but symbolic-algebra systems make it easy. Here is an example in Mathematica:

```
In[1]:= Replace[InverseSeries[Series[Sin[x], {x, 0, 11}], x], x->y, 1]
```

$$\text{Out[1]} = y + \frac{y^3}{6} + \frac{3y^5}{40} + \frac{5y^7}{112} + \frac{35y^9}{1152} + \frac{63y^{11}}{2816} + 0[y^{12}]$$

For comparison, we can ask for the expansion of the inverse of the sine function, known as the *arcsine*:

```
In[2]:= Series[ArcSin[y], {y, 0, 11}]
```

$$\text{Out[2]} = y + \frac{y^3}{6} + \frac{3y^5}{40} + \frac{5y^7}{112} + \frac{35y^9}{1152} + \frac{63y^{11}}{2816} + 0[y^{12}]$$

The two outputs are identical, as expected.

Maple can find the inverse series with its general equation solver, but we have to tell the `series()` function the order of the highest term that we want in the output display:

```
> Order := 13;
> solve(series(sin(x), x = 0, 11) = y, x);
```

$$y + \frac{1}{6}y^3 + \frac{3}{40}y^5 + \frac{5}{112}y^7 + \frac{35}{1152}y^9 + \frac{63}{2816}y^{11} + 0(y^{13})$$

The syntax in the MuPAD algebra system is similar to that of Maple, but MuPAD has a separate function for inverting series expansions:

```
>> subs(revert(series(sin(x), x = 0, 11)), x = y);
```

$$y + \frac{y^3}{6} + \frac{3y^5}{40} + \frac{5y^7}{112} + \frac{35y^9}{1152} + \frac{63y^{11}}{2816} + 0(y^{13})$$

Many of the functions in this book have a companion function that is their inverse: square root and square, exponential and logarithm, sine and arcsine, and so on, and you can easily find them on hand calculators and in software libraries. However, the inverse functions are sometimes less well-known, and we consider some examples later in **Chapter 19**.

2.11 Summary

Iterative solutions of nonlinear equations with the first- and second-order procedures described in this chapter have broad applicability in numerical computation. Importantly, the algorithms can be generalized to find numerical solutions of problems involving more than the single variable that we generally consider in this book.

Representations of functions with Taylor series, asymptotic expansions, and continued fractions are essential tools for developing practical computational algorithms for function evaluation. Although traditional textbooks in numerical analysis often rely on mathematical encyclopedias, handbooks, and tables as sources of such expansions, we have seen in this chapter that symbolic-algebra languages offer more accessible, and more powerful, ways for finding, and then manipulating, such material. Those languages also permit numerical experimentation with arithmetic of arbitrary precision, so they can be invaluable for prototyping algorithms before implementing them in more conventional programming languages. We use symbolic-algebra systems often in this book, and we need more than one of them, because none yet supplies all of the capabilities that we require.

The famed computer scientist Richard Hamming is widely cited for the quote *The purpose of computing is insight, not numbers*. Mathematics, symbolic-algebra systems, and graphical displays of data and functions, are helpful tools for developing that insight.

The Mathematical-Function Computation Handbook
Programming Using the MathCW Portable Software
Library

Beebe, N.H.F.

2017, XXXVI, 1115 p., Hardcover

ISBN: 978-3-319-64109-6