1
Basic Numerical Analysis Techniques

1.1 Introduction

All the topics discussed in this chapter are standard topics covered in most undergraduate texts and reference books on numerical analysis. We do not expect that the present volume will actually be a text for topics such as interpolation or quadrature. However, these methods are all used extensively in statistical computing work, and it is essential that any volume on statistical computing provides adequate discussion to clearly outline the basic nature and use of these methods. As these are standard mathematical tools which are used by scientists in many disciplines for different purposes, we have mostly kept our discussion general although the statistical angle may be clear in some cases; some instances of specific applications in appropriate statistical contexts may be presented in the following chapters. In general we have primarily concentrated on the more popular and standard methods (such as Newton's forward difference interpolation formula in interpolation and the Newton Cotes formulae in quadrature) and provided only the brief ideas for the more complex methods with appropriate suggestions for further reading.

The material presented here has borrowed from many well known books on numerical analysis. As there is significant overlap between them, we have presented all our sources in the list of references, but not necessarily in the text. Perhaps the book by Press, Flannery, Teukolsky and Vetterling (1986) deserves special mention as it includes FORTRAN subroutines for many of the methods discussed in this chapter.

We begin with a basic description of the nature of error in floating point computations. Although it is not a "numerical analysis" technique, the concept of error in computation applies equally to statistical computations as to other disciplines. Also it is unlikely to be touched upon by the following, more specific, chapters. Other topics covered here are interpolation, quadrature, and numerical root finding techniques.

1.2 Error in Floating Point Computations

Normally we expect that a logical sequence of correct operations in a computer will lead to a "correct" set of results through appropriate computations. However, any floating point computation in the computer can lead to results which contain many different kinds of errors. Users should be aware of the extent and nature of these errors so as to have an overall idea of the level of inaccuracy.

In measuring or approximating the error in computation, we often make a distinction between the absolute error and the relative error. By definition, the absolute error is the difference between the computed value and the true value. On the other hand, the relative error is the ratio of the absolute error to the true value (provided the latter is nonzero). Often one is more interested in the relative degree of inaccuracy rather than its absolute magnitude, and in such cases the relative error is the more important quantity.

The results produced by the computer can be erroneous because of any one or more of the following three reasons.

1. It may be the result of a mistake such as a bug in the program or incorrect recording/reading of data.

2. It may be the result of a mathematical algorithm which gives approximate results.

3. It may be due to an approximation inherent in the functioning of the computer.

Generally errors of the first type can be controlled by the user and can be rectified through proper scrutiny of the codes and the data. However the user only has partial control over the other two types of errors. All the topics discussed in the remaining part of this chapter are subject to errors of the second type. One can, for example, increase the precision of a numerical root solving technique by setting a more stringent convergence criterion, but one cannot eliminate it altogether. The third kind of error is due to the computers inability to fully represent floating point numbers and its application of truncation and rounding to accommodate them. One can, for example, change the degree of operations and storage from single precision to double precision, but improving the precision cannot be done indefinitely.

See Sterbenz (1974) and Kennedy and Gentle (1980) for more detailed discussions on the idea of error in computations. Also see Moore (1966) for some work on computing error bounds.

1.3 Interpolation

Interpolation is one of the core topics of classical numerical analysis, which tries to determine the approximate value of an unknown function between two tabulated points. Although the need for sophisticated interpolation techniques for the sake of interpolation itself has decreased with the growth of computers, they are still useful in many applications. In addition interpolation formulae are the starting point of many other areas of classical numerical analysis such as numerical differentiation and quadrature.

Suppose that we have a function $f(x)$ and the value of the function is known and tabulated for a particular set of points (to be called tabulated points) $x_0, x_1, \ldots, x_n$. Our intention is to approximate the function $f(x)$ by a function $g(x)$ which matches $f(x)$ at each of the tabulated points. We will restrict ourselves to the case where $g(x)$ is a polynomial, and try to develop it using the known values of the function at the tabulated points. Our interpolation formula has the form

$$g(x) = \sum_{j=0}^{n} l_j(x) f(x_j), \quad f(x) = g(x) + e(x), \tag{1.1}$$

and we want to determine the coefficients $l_j(x)$ so that the error $e(x)$ vanishes at the tabulated points, i.e.

$$e(x_j) = 0, \quad j = 0, 1, 2, \ldots, n, \tag{1.2}$$

independent of the function $f(x)$. We hope to construct an accurate interpolation formula satisfying (1.1) for which an error bound can be determined for nontabulated points, where $e(x) \neq 0$ in general.

1.3.1 Lagrangian Interpolation

Consider the case where there are no restrictions on the spacing of the points. Essentially we find the polynomial of maximal degree n which passes through the points $(x_0, f(x_0)), (x_1, f(x_1)), \ldots, (x_n, f(x_n))$. Notice that conditions (1.1) and (1.2) imply that

$$l_j(x_k) = \delta_{jk}, \ j, k = 0, \ldots, n, \tag{1.3}$$

where δ_{jk} is the Kronecker delta (being equal to 1 when $j = k$, and 0 otherwise). Since $l_j(x)$ is a polynomial which vanishes at all the tabulated points except x_j where it equals 1, we have

$$l_j(x) = \frac{(x - x_0)(x - x_1) \ldots (x - x_{j-1})(x - x_{j+1}) \ldots (x - x_n)}{(x_j - x_0)(x_j - x_1) \ldots (x_j - x_{j-1})(x_j - x_{j+1}) \ldots (x_j - x_n)} \tag{1.4}$$

which is the only polynomial of degree n satisfying (1.3). Notice that one can write

$$l_j(x) = \frac{p(x)}{(x - x_j)p'(x_j)}, \ p'(x_j) = \left.\frac{dp(x)}{dx}\right|_{x=x_j}, \ p(x) = \prod_{i=0}^{n} (x - x_i). \tag{1.5}$$

To find an expression for $e(x)$ note that the function

$$F(z) = f(z) - g(z) - [f(x) - g(x)]\frac{p(z)}{p(x)} \tag{1.6}$$

as a function of z has has $n+2$ zeroes at the points $x_0, x_1, \ldots, x_n$ and x in the interval $(\min_i(x_i, x), \max_i(x_i, x))$. Therefore, by applying Rolle's Theorem $n+1$ times,

$$F^{(n+1)}(z) = f^{(n+1)}(z) - g^{(n+1)}(z) - \frac{[f(x) - g(x)]}{p(x)}(n+1)! \tag{1.7}$$

has at least one zero at $z = \xi$ in the above interval. Since $g^{(n+1)}(z) = 0$, we get

$$e(x) = f(x) - g(x) = \frac{p(x)}{(n+1)!}f^{(n+1)}(\xi). \tag{1.8}$$

This error bound requires the assumption that f is $(n+1)$ times continuously differentiable. This condition can be weakened somewhat, but then we also have to be satisfied with weaker error bounds. On the other hand, the error estimate cannot be improved in general by assuming more differentiability. See Hämmerlin and Hoffman (1991) for a more detailed discussion on this subject.

Equation (1.1) with the $l_j(x)$ given by (1.4) and and $e(x)$ by (1.8) is called the Lagrangian interpolation formula. When $n = 1$, $g(x)$ is the familiar formula for linear interpolation

$$g(x) = \frac{x - x_1}{x_0 - x_1}f(x_0) + \frac{x - x_0}{x_1 - x_0}f(x_1). \tag{1.9}$$

1.3.2 Divided Differences and the Method of Newton

Suppose that the interpolating polynomial $g(x)$ in equation (1.1) be written as

$$g(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \ldots + a_n(x - x_0)\ldots(x - x_{n-1}). \tag{1.10}$$

The coefficients can be successively computed by the interpolation conditions $e(x_j) = 0$, $j = 0, \ldots, n$. By the uniqueness of the polynomial of maximal degree n satisfying the interpolation conditions, this is the same polynomial as in equation (1.1). The coefficients $a_0, a_1, \ldots$, can be successively computed as

$$\begin{aligned} a_0 &= f(x_0) \\ a_1 &= \frac{f(x_1) - f(x_0)}{x_1 - x_0}, \text{ etc.} \end{aligned}$$

This method has the advantage that if additional points are added, the degree of the interpolating polynomial increases, but the old coefficients $a_0, a_1, \ldots, a_n$ remain unchanged. The form of the interpolation polynomial in equation (1.10) is called the Newton form.

Notice that the coefficient a_1 in the Newton form has the nature of a difference quotient. We will call this a divided difference of the first order and denote it by the symbol

$$[x_1, x_0] = \frac{f(x_1) - f(x_0)}{x_1 - x_0}. \tag{1.11}$$

Higher order divided differences may be defined successively by extending the above idea, and the m-th order divided difference, $2 \le m \le n$, is defined as

$$[x_m, x_{m-1}, \ldots, x_0] = \frac{[x_m, \ldots, x_1] - [x_{m-1}, \ldots, x_0]}{x_m - x_0}. \tag{1.12}$$

The divided differences for the function f at a point x not equal to any of the tabulated points x_j, $j = 0, \ldots, n$, will be defined as:

$$\begin{aligned}
[x_0, x] &= \frac{f(x_0) - f(x)}{x_0 - x} \\
[x_1, x_0, x] &= \frac{[x_1, x_0] - [x_0, x]}{x_1 - x} \\
&\vdots \\
[x_n, x_{n-1}, \ldots, x_0, x] &= \frac{[x_n, \ldots, x_0] - [x_{n-1}, \ldots, x]}{x_n - x}
\end{aligned}$$

Wherever necessary, to avoid confusion, we will write $[x_m, \ldots, x_0]f$ instead of $[x_m, \ldots, x_0]$ to indicate that the divided differences correspond to the given function f. Starting with $[x_n, x_{n-1}, \ldots, x_0, x]$ and by repeated substitution of the above formula, we obtain the Newton identity

$$\begin{aligned}
f(x) &= f(x_0) + [x_1, x_0](x - x_0) + [x_2, x_1, x_0](x - x_0)(x - x_1) + \ldots \\
&\quad + [x_n, \ldots, x_0](x - x_0) \ldots (x - x_{n-1}) + [x_n, \ldots, x](x - x_0) \ldots (x - x_n)
\end{aligned} \tag{1.13}$$

Notice that

$$e(x) = f(x) - g(x) = [x_n, \ldots, x](x - x_0) \ldots (x - x_n) \tag{1.14}$$

and the identity (1.13) decomposes the function f into two terms: the interpolating polynomial $g(x)$, and the error term $e(x)$.

Comparing with the Newton formula above, we see that $a_0 = f(x_0)$, and $a_m = [x_m, \ldots, x_1, x_0]$, $1 \le m \le n$, and the remainder term e can be written in terms of the data polynomial $p(x)$ as

$$e(x) = [x_n, \ldots, x]p(x). \tag{1.15}$$

The divided differences also have the nice property that they do not depend on the order of the points $x_0, \dots, x_m$.

1.3.3 Equidistant Interpolation Points

Suppose that $n+1$ equidistant points are given, and they are indexed as $x_i = x_0 + ih$, $i = 0, 1, \dots, n$, for some fixed step size $h > 0$. The m-th forward difference is defined as follows.

$$\begin{aligned} \Delta^0 f(x_i) &= f(x_i) \\ \Delta^m f(x_i) &= \Delta^{m-1} f(x_{i+1}) - \Delta^{m-1} f(x_i), \quad \text{for } m \geq 1. \end{aligned}$$

Notice that in terms of the divided difference notation this implies $\Delta f(x_0) = h[x_1, x_0]f$. It may be easily proved, proceeding by induction, that $\Delta^m f(x_0) = m! h^m [x_m, x_{m-1}, \dots, x_0]f$, $m \geq 1$.

Coefficients of the interpolating polynomial can now be computed by setting up the following *difference table*:

$$\begin{array}{cccccc}
x_0 & f(x_0) & & & \\
 & & \Delta f(x_0) & & \\
x_1 & f(x_1) & & \Delta^2 f(x_0) & \cdot \\
 & & \Delta f(x_1) & \cdot & \\
x_2 & f(x_2) & & \Delta^2 f(x_1) & \cdot \\
 & & \Delta f(x_2) & \cdot & \\
x_3 & f(x_3) & & \Delta^2 f(x_2) & \cdot \\
 & & \cdot & & \cdot \\
\cdot & \cdot & & \cdot & \cdot
\end{array}$$

The differences of different orders for the same index are located on the same vertical sloping diagonal line.

The most basic formula for interpolation with equal intervals is Newton's forward difference (or descending or advancing difference) formula. In this case the interpolation formula is built up using the leftmost point at each stage. Newton's interpolation formulae are sometimes also referred to as the Newton-Gregory formulae. With respect to $n+1$ distinct points $x_0, x_1, \dots, x_n$, let $g(x)$ be the interpolation polynomial for $f(x)$ of degree at most n. Thus let

$$g(x) = a_0 + a_1(x - x_0) + \ldots + a_n(x - x_0)(x - x_1)\ldots(x - x_{n-1}). \quad (1.16)$$

The unknown coefficients can be determined by successively replacing the values x_0, x_1, etc. in the equation (where the functional values are known) and thus solving for the constants. This leads to the following set of coefficients:

$$\begin{aligned} a_0 &= f(x_0) \\ a_m &= \frac{\Delta^m f(x_0)}{m! h^m} \quad m = 1, 2, \dots, n \end{aligned} \quad (1.17)$$

The above may also be directly deduced by using the form of the coefficients obtained in the previous subsection, and writing the divided differences in terms of the forward differences. The name of the formula indicates that values along the top diagonal of the difference table are used. For $f \in C^{n+1}[a,b]$, the remainder term e in $f = g + e$ for $x, x_j \in [a,b]$ becomes

$$e(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}(x-x_0)\dots(x-x_n), \ \xi \in (\min(x,x_0), \max(x,x_n)),$$

where $C^{n+1}[a,b]$ is the set of all funcitons on $[a,b]$ that are $n+1$ times continuously differentiable. The equation (1.16) with coefficients given by (1.17) represents Newton's forward difference interpolation formula.

The interpolation formula may be more easily represented with the following set of transformations. Let $t = \dfrac{x-x_0}{h}$, and denote the transformed functions in terms of t as: $f(x) = f^*(t)$, $g(x) = g^*(t)$, $e(x) = e^*(t)$. Then Newton's forward interpolation formula becomes

$$g^*(t) = f(x_0) + \Delta f(x_0)\binom{t}{1} + \Delta^2 f(x_0)\binom{t}{2} + \dots + \Delta^n f(x_0)\binom{t}{n} \quad (1.18)$$

$$e^*(t) = \binom{t}{n+1}\left.\frac{d^{n+1}f^*(t)}{dt^{n+1}}\right|_{t=\xi^*}, \quad \xi^* \in (\min(t,0), \max(t,n)) \quad (1.19)$$

On the other hand, we may want the interpolating polynomial to be expanded using the rightmost points at each step, and in this case we will take $x_{n-j} = x_n - jh$, $0 \le j \le n$, and introduce the backward differences as

$$\begin{array}{rcl} \nabla^0 f(x_{n-j}) & = & f(x_{n-j}) \\ \nabla^m f(x_{n-j}) & = & \nabla^{m-1} f(x_{n-j}) - \nabla^{m-1} f(x_{n-j-1}), \quad \text{for } m \ge 1. \end{array} \quad (1.20)$$

These backward differences lead to the difference table

$$\begin{array}{ccccc}
\cdot & \cdot & & \cdot & \cdot \\
 & & \cdot & & \cdot \\
x_{n-3} & f(x_{n-3}) & & \nabla^2 f(x_{n-2}) & \cdot \\
 & & \nabla f(x_{n-2}) & \cdot & \\
x_{n-2} & f(x_{n-2}) & & \nabla^2 f(x_{n-1}) & \cdot \\
 & & \nabla f(x_{n-1}) & \cdot & \\
x_{n-1} & f(x_{n-1}) & & \nabla^2 f(x_n) & \cdot \\
 & & \nabla f(x_n) & & \\
x_n & f(x_n) & & &
\end{array}$$

In this case Newton's backward or ascending difference interpolation formula is given by

$$g(x) = a_0 + a_1(x-x_n) + \dots + a_n(x-x_n)(x-x_{n-1})\dots(x-x_1) \quad (1.21)$$

where the coefficients are now

$$\begin{aligned} a_0 &= f(x_n) \\ a_m &= \frac{\nabla^m f(x_n)}{m!h^m} \quad m = 1, 2, \ldots, n \end{aligned} \tag{1.22}$$

The transformation $t = \dfrac{x - x_n}{h}$ leads to the following simpler form

$$g^*(t) = f(x_n) + \nabla f(x_n)\binom{t}{1} + \nabla^2 f(x_n)\binom{t+1}{2} + \ldots + \nabla^n f(x_n)\binom{t+n-1}{n} \tag{1.23}$$

where $f(x) = f^*(t)$ and $g(x) = g^*(t)$, with remainder

$$e^*(t) = \binom{t+n}{n+1} \left.\frac{d^{n+1} f^*(t)}{dt^{n+1}}\right|_{t=\xi^*} \quad \xi^* \in (\min(t, -n), \max(t, 0)) \tag{1.24}$$

One can also develop interpolation polynomials with the reference for interpolation being at the center. In this case $x_j = x_0 + jh$, $j = 0, \pm 1, \ldots, \pm k$. The central differences are now defined as

$$\begin{aligned} \delta^0 f(x_j) &= f(x_j) \\ \delta^m f(x_{j+\frac{1}{2}}) &= \delta^{m-1} f(x_{j+1}) - \delta^{m-1} f(x_j), \quad \text{for } m \geq 1 \text{ and odd} \\ \delta^m f(x_j) &= \delta^{m-1} f(x_{j+\frac{1}{2}}) - \delta^{m-1} f(x_{j-\frac{1}{2}}), \quad \text{for } m \geq 2 \text{ and even} \end{aligned} \tag{1.25}$$

The difference table then has the following form

$$\begin{array}{ccccc}
\cdot & \cdot & & \cdot & \cdot \\
 & & \cdot & & \cdot \\
x_{-1} & f(x_{-1}) & & \delta^2 f(x_{-1}) & \cdot \\
 & & \delta f(x_{-\frac{1}{2}}) & \cdot & \\
x_0 & f(x_0) & & \delta^2 f(x_0) & \cdot \\
 & & \delta f(x_{\frac{1}{2}}) & \cdot & \\
x_1 & f(x_1) & & \delta^2 f(x_1) & \cdot \\
 & & \cdot & & \\
\cdot & \cdot & & \cdot &
\end{array}$$

Using the mean difference $\bar{\delta}^m f(x_0) = \frac{1}{2}(\delta^m f(x_{1/2}) + \delta^m f(x_{-1/2}))$ for $m \geq 1$ (and odd) and using the transformation $t = \dfrac{x - x_0}{h}$, one gets the Stirling interpolation formula

$$\begin{aligned} g^*(t) = f(x_0) + \bar{\delta} f(x_0) t + \delta^2 f(x_0) \frac{t^2}{2!} \\ + \bar{\delta}^3 f(x_0) \frac{t(t^2-1)}{3!} + \ldots + \delta^{2k} f(x_0) \frac{t^2(t^2-1)\ldots(t^2-(k-1)^2)}{(2k)!} \end{aligned} \tag{1.26}$$

where $f(x) = f^*(t)$, $g(x) = g^*(t)$ with remainder term

$$e^*(t) = \binom{t+k}{2k+1} \frac{d^{2k+1} f^*(t)}{dt^{n+1}}\bigg|_{t=\xi^*} \quad \xi^* \in (\min(t,-k), \max(t,k)) \qquad (1.27)$$

1.4 Quadrature Methods

The numerical evaluation of one dimensional integrals, or quadrature, is another important technique which has important uses in statistical inference and data analysis. These are techniques that have to be applied to get approximate answers when analytical computation of definite integrals are not possible or difficult – which is sometimes true even for apparently simple looking functions.

1.4.1 Newton Cotes Methods

Newton Cotes formulae are a family of approximate integration rules and represent some of the best known quadrature methods. These formulae are based on $n+1$ equidistant values $a = x_0,\ x_1, \ldots x_{n-1},\ x_n = b$ of the argument x, for $n = 1, 2, 3 \ldots$, which covers the range of integration (a, b) and where the functional values are known. The derivation of Newton-Cotes formulae represent the integration of the appropriate number of the terms of Newton's forward difference formulae. In effect this replaces the integrand with polynomials which agree with $f(x)$ on the grid of points. Let $h = (b-a)/n$, so that $x_i = x_0 + ih$, $i = 1, \ldots, n$.

The simplest formula is based on two points (x_0 and x_1) and is computed by replacing the integrand with the corresponding Newton's forward interpolation formula up to two terms. This yields (with $h = x_1 - x_0$)

$$\begin{aligned}
\int_{x_0}^{x_1} f(x)dx &= \int_{x_0}^{x_1} \left[f(x_0) + \frac{x - x_0}{h} \Delta f(x_0) \right] dx \\
&= hf(x_0) + \frac{h}{2} \Delta f(x_0) \\
&= \frac{h}{2} \left[f(x_0) + f(x_1) \right]
\end{aligned}$$

The above rule is commonly called the *trapezoidal rule*. The name reflects the fact that the area under $f(x)$ for each of these subintervals are approximated by trapezoids. The basic trapezoidal rule

$$\int_{x_0}^{x_1} f(x)dx = \frac{h}{2} \left[f(x_0) + f(x_1) \right] \qquad (1.28)$$

is exact for linear functions. By dividing the the interval $[a, b]$ into n subintervals $[x_k, x_{k+1}]$ of width $h = (b-a)/n$, applying the above formula to each of these intervals, and summing these approximate values over all subintervals produce the overall approximate integral may be expressed as:

$$\int_{x_0}^{x_n} f(x)dx = \frac{h}{2}\left[f(x_0) + 2f(x_1) + 2f(x_2) + \ldots + 2f(x_{n-1}) + f(x_n)\right] \tag{1.29}$$

Equation (1.29), which may be called the composite trapezoidal rule replaces the integrand by piecewise linear approximations.

The Newton Cotes formula based on three points ($n = 2$) is called *Simpson's one-third rule*, and is derived by carrying Newton's forward difference formula to three terms. For three points x_0, $x_1 = x_0 + h$, $x_2 = x_0 + 2h$, the integral $\int_{x_0}^{x_2} f(x)dx$ is approximated by

$$\int_{x_0}^{x_2} [f(x_0) + \frac{\Delta f(x_0)}{h}(x - x_0) + \frac{\Delta^2 f(x_0)}{2h^2}(x - x_0)(x - x_1)]dx$$

which leads to the basic Simpson's one-third rule

$$\int_{x_0}^{x_2} f(x)dx = \frac{h}{3}[f(x_0) + 4f(x_1) + f(x_2)] \tag{1.30}$$

For $n + 1$ equidistant points $x_0, x_1, \ldots, x_n$, where n is a multiple of 2, the corresponding composite form is given by

$$\int_{x_0}^{x_n} f(x)dx =$$

$$\frac{h}{3}\left[f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + \ldots + 4f(x_{n-1}) + f(x_n)\right] \tag{1.31}$$

Similarly, the Newton-Cotes formula based on four points ($n = 3$) is called *Simpson's three-eighths rule* and is derived by carrying Newton's forward difference formula to four terms. The composite version of this rule (applicable when n is a multiple of 3) is given by

$$\int_{x_0}^{x_n} f(x)dx =$$

$$\frac{3h}{8}\left[f(x_0) + 3f(x_1) + 3f(x_2) + 2f(x_3) + 3f(x_4) + \ldots + 3f(x_{n-1}) + f(x_n)\right] \tag{1.32}$$

The trapezoidal rule gives exact results for first degree polynomials. Figure 1.1 gives an example of the trapezoidal estimate of the integral of a nonlinear function based on five points. Similarly, the three eighths rule is found to exactly reproduce the integrals for third degree polynomials. However, Simpson's one-thirds rule is found to exactly reproduce the integral of a third degree polynomial as well, although its derivation is carried only

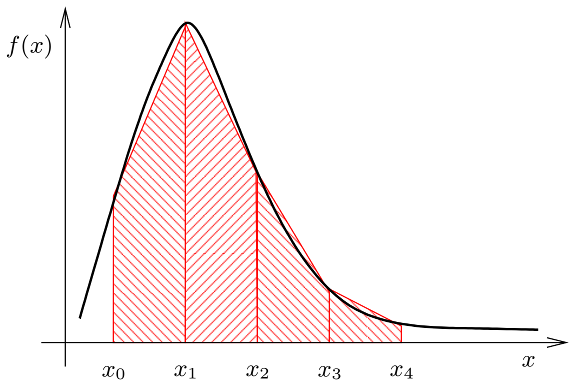

Figure 1.1: Trapezoidal estimate of an integral.

up to the second differences. Thus it is able to gain an additional degree of accuracy, and as a result is a very popular method in practice. Actually it can be shown that this is true for any symmetrical integration formula involving an odd number of terms.

The Newton Cotes rules described above are all examples of *closed* rules, in that the endpoints of each interval are among the points at which the integrand is evaluated. Closed basic rules can be naturally combined to get composite rules. Open Newton Cotes rules, which are based on equally spaced points placed on the interior of the integration interval, are sometimes used to handle integrands which exhibit singularities at either or both ends. However, they cannot be combined to produce composite rules, and except special cases are generally of limited applicability.

1.4.2 Romberg's Algorithm

A clever manipulation which can reduce the order of error in estimating integrands numerically is the *Romberg algorithm.* To illustrate this algorithm, consider the trapezoidal for which the error in estimation is $O(h^2)$, and hence of order $O(n^{-2})$ since $h = (b-a)/n$ (see Thisted, 1988, or Jacques and Judd, 1987, for a discussion of the error rates of the Newton Cotes rules). Using the Euler summation formula for an integral, where all indicated derivatives exist, we get

$$\int_{x_0}^{x_n} f(x)dx = h\sum_{i=0}^{n-1} f(x_i) + \frac{h}{2}(f(x_n) - f(x_0))$$
$$-\sum_{j=1}^{k} \frac{B_{2j}h^{2j}}{(2j)!}(f^{(2j-1)}(x_n) - f^{(2j-1)}(x_0)) + R_{2k}. \qquad (1.33)$$

The coefficients B_{2j} are the Bernoulli numbers, which are the coefficients in the McLaurin series expansion of $t/(e^t - 1)$. The first two terms give the trapezoidal approximation of the integral. In the remaining part of the formula, there are no odd terms. Letting I_j represent the trapezoidal estimate of the integral based on 2^j+1 terms, it is easily seen that $4/3I_{j+1} - 1/3I_j$ is an estimate for the integral with error of the order of n^{-4}, i.e. the error term of order n^{-2} is eliminated by the above linear combination. Extending this idea, one can take weighted sums of k successive trapezoidal estimates to get rid of higher order terms up to but not including the $O(n^{-2k})$. See Thisted (1988), Lange (1998), Henrici (1982), and Jacques and Judd (1987) for more details on Romberg's algorithm. Romberg's algorithm is also a special case of *Richardson's deferred approach to the limit*, where a numerical algorithm is performed for various values of a parameter h, and then extrapolated to the limiting case for $h \to 0$.

1.4.3 Gaussian Quadrature

The basic Newton Cotes integration formula based on $n+1$ equally spaced points can exactly integrate polynomials of degree n or less (and hence belong to the class of n-th order integration rules). In this case the abscissa where the integrand are evaluated are given, while the user has the freedom to select the weighting constants. In Gaussian quadrature the method gives the user the freedom to select the weighting constants as well as the abscissa where the integrand is evaluated. In effect this puts twice the number of degrees of freedom at the disposal of the user. Thus for the Gaussian quadrature formulae one can essentially achieve twice the order of the Newton-Cotes formulae with the same number of function evaluations.

Gaussian quadrature in fact allows exact evaluation of more general integrands which are in the form of a polynomial times some known weight function, i.e. of the type

$$\int_a^b w(x)f(x)dx \tag{1.34}$$

where $w(x)$ is a (nonnegative) weight function and either limits of the integral are allowed to be infinite. The most commonly used family of Gaussian quadrature rules corresponds to $w(x) = 1$, and when the integrand is defined on a finite interval the corresponding set of integration rules are called Gauss-Legendre integration rules. Without loss of generality one can restrict attention to the interval $(-1, 1)$ in this case, with a transformation of the integrand if necessary.

A weight function is called admissible when

$$\int_a^b w(x)dx > 0 \text{ and } \int_a^b x^k w(x)dx < \infty$$

for all $k \geq 0$. if the weight function is standardized so that $\int_a^b w(x)dx = 1$, the admissible weight functions are simply probability densities with all moments finite. Statistically, one can then view equation (1.34) as representing the expectation of the function $f(x)$.

The theory behind Gaussian quadratures is very closely related to that of orthogonal polynomials. Given a weight function $w(\cdot)$, we define the scalar product of two different square integrable functions f and g as

$$\langle f, g\rangle = \int_a^b w(x)f(x)g(x)dx.$$

The functions f and g are orthogonal with respect to this inner product if $\langle f, g\rangle = 0$. Given a weight function w, we can find a set of polynomials that includes exactly one polynomial of order j, denoted $\psi_j(x)$, for each $j = 0, 1, 2 \ldots$ such that all of them are mutually orthogonal with respect to the inner product defined by w.

For Gaussian quadrature formulae on $k + 1$ points it may be shown that the abscissa to be used in this case are precisely the roots of the orthogonal polynomial $\psi_{k+1}(x)$ given the corresponding interval and the weight function. Once the abscissa points are known the weights w_i may be obtained as the solution of the system of linear equations

$$\begin{bmatrix} \psi_0(x_0) & \ldots & \psi_0(x_k) \\ \psi_1(x_0) & \ldots & \psi_1(x_k) \\ \vdots & & \vdots \\ \psi_k(x_0) & \ldots & \psi_k(x_k) \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_k \end{bmatrix} = \begin{bmatrix} \int_a^b w(x)\psi_0(x)dx \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{1.35}$$

It can also be easily shown that with these weights, the integral is exact for all polynomials of degree $2k + 1$ or less.

See Thisted (1988), Press et al. (1986), or Lange (1998) for more details on Gaussian quadrature.

1.5 Solution of Nonlinear Equations

Many of the optimization techniques in statistics are invariably linked to the determination of the minima or maxima of functions through appropriate root solving techniques for the corresponding derivative functions. For complex nonlinear equations, closed form analytical solutions are often very hard or even impossible to determine. In this section we discuss some common numerical root solving techniques which determine approximate solutions of the roots via successive approximations and are used extensively in statistical work. In order to get a proper understanding of the basics of the method, we discuss the one variable problem in detail. Some of these methods have simple generalizations to the higher dimensional problems.

For the rest of this section we assume that no solution in closed form can be found for the equations in question, making approximate numerical solutions necessary. For all the methods we will see that the continuity of the function $f(x)$ is a minimum requirement.

1.5.1 Bisection Method

The bisection method is one of the simplest methods of finding solutions to an equation $f(x) = 0$ in one unknown. Admittedly it is not one of the faster methods, but it neither requires the evaluation of or conditions on the derivative function, and under fairly minimal conditions is guaranteed to converge to some root in a given interval. We assume that the function is continuous, and suppose that there exists an interval $[a, b]$ such that $f(a)$ and $f(b)$ are of opposite signs. Since the function is continuous and f has different signs at a and b, by the intermediate value theorem there must exist a point in the interval $[a, b]$ where the function vanishes. Successive iterations of the bisection algorithm determine the next iterate by choosing the mid point of the current interval. Thus, at the first step one chooses the point $c = (a + b)/2$. If $f(c) = 0$, the process stops there. Otherwise, either $f(a)$ and $f(c)$ are of opposite sign in which case $[a, c]$ contains a root, or $f(c)$ and $f(b)$ are of opposite signs and $[c, b]$ brackets a root. We replace $[a, b]$ by the appropriate subinterval where the function has opposite signs at the two end points and continue. If we bisect $[a, b]$ a total of n times, the bracketing interval has length $2^{-n}(b - a)$. For n large enough, we can stop and approximate the bracketed root by the midpoint of the final bracketing interval.

Statistical applications of the method of bisection in finding the quantiles of a distribution or confidence intervals of shortest width are considered in Lange (1998).

1.5.2 Method of Iteration

The method of iteration, or the method of functional iteration is another method which determines the root of an equation $f(x) = 0$ without requiring the derivatives of the function $f(x)$. Suppose that the equation $f(x) = 0$ can be equivalently written as $x = F(x)$. This can be achieved, for example, if one chooses $F(x) = f(x) + x$. The iterative procedure is then begun with some appropriate starting value $x^{(0)}$. At the i-th stage, the next iterate is determined from the relation $x^{(i+1)} = F(x^{(i)})$. One continues the process until some appropriate convergence criterion in met. A root of $f(x)$ is called a fixed point of $F(x)$. The above algorithm is also called the Fixed-Point algorithm.

The convergence of the Fixed-Point algorithm to a root of the equation $f(x) = 0$ is not automatic, and the conditions that guarantee it are discussed below. When the method converges, the rate of convergence is linear (see discussion below).

In order to facilitate the comparison of the method of iteration with the Newton-Raphson algorithm described later, we define the concept of the rate of convergence. To cover the general case with several unknowns, let x be a vector in $\mathbb{R}^p$, and let $||x||$ represent its Euclidean norm (the restriction to the Euclidean norm is not necessary, but we stick to it to keep a clear focus here. For the univariate problem it is simply the absolute value of x.) Let α be the true root of equation to which convergence is desired. Consider a sequence $\{x^{(i)} : i = 0, 1, 2, \ldots\}$ with $x^{(0)}$ sufficiently close to α. We will say that $\{x^{(i)}$ converges linearly to α if there exists a constant $c \in (0, 1)$ such that

$$||x^{(i+1)} - \alpha|| \leq c||x^{(i)} - \alpha||. \tag{1.36}$$

The sequence converges quadratically if for a sufficiently close starting value there exists a positive constant c such that

$$||x^{(i+1)} - \alpha|| \leq c||x^{(i)} - \alpha||^2. \tag{1.37}$$

The sequence converges superlinearly if

$$\limsup ||x^{(i+1)} - \alpha||/||x^{(i)} - \alpha|| = 0. \tag{1.38}$$

In general we will say that the sequence $x^{(0)}, x^{(1)}, \ldots,$ exhibits convergence of order β if $\lim_{i\to\infty} ||x^{(i+1)} - \alpha|| = c||x^{(i)} - \alpha||^\beta$, for some nonzero constant c. Quadratic convergence corresponds to $\beta = 2$, and $\beta = 1$ leads to linear convergence.

Consider the univariate problem in the context of the Fixed point algorithm. Let α be the true solution. Assume that $F(x)$ has a continuous derivative in an interval around α. Let $x^{(i)}$ be the i-the iterate, so that

$$\alpha - x^{(i+1)} = F(\alpha) - F(x^{(i)}) = (\alpha - x^{(i)})F'(\xi^{(i)}) \tag{1.39}$$

where $\xi^{(i)}$ is a point between α and $x^{(i)}$, and when $F'(\xi^{(i)})$ is smaller than 1 in absolute magnitude, the following iterate is closer to the true solution. Sufficient closeness of $x^{(i)}$ to α will mean that $|\alpha - x^{(i+1)}|$ will be approximately described by $|F'(\alpha)(\alpha - x^{(i)})|$. Thus, when F is continuously differentiable at α and $|F'(\alpha)| < 1$, the iteration is asymptotically stable at α and the method converges, at a linear rate, for a suitable starting value close to α. In particular the error $(x^{(i)} - \alpha)$ tends to be described approximately by $a[F'(\alpha)]^i$, where a is a constant. The successive approximations tend to α from the same direction if $0 < F'(\alpha) < 1$, and oscillate about α with decreasing amplitude if $-1 < F'(\alpha) < 0$. However, the deviation may become unbounded if $|F'(\alpha)| > 1$, in which case the iteration is asymptotically unstable at α. Figure 1.2 provides illustrations of oscillatory convergence, monotone convergence and divergence of the fixed point method. The bold curve represents the $y = F(x)$ line and the true solution α corresponds to the abcissa of the point where the line $y = x$ and the curve $y = F(x)$ intersect.

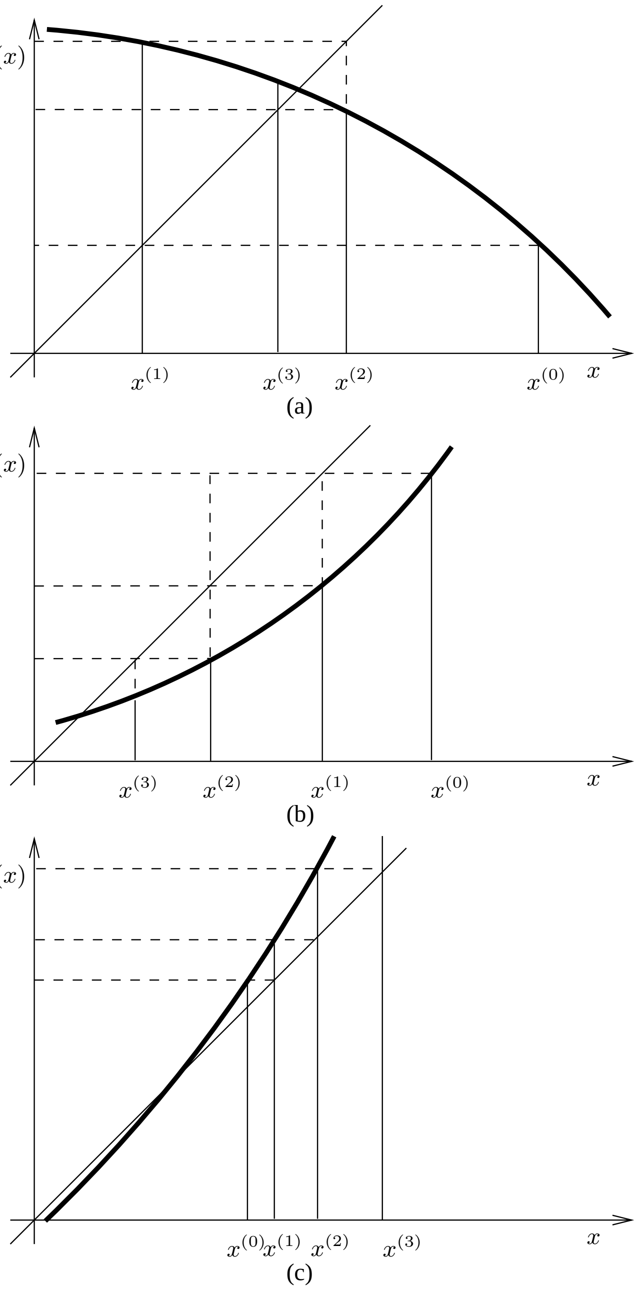

Figure 1.2: (a) Oscillatory convergence of the fixed point method. (b) Monotone convergence of the fixed point method. (c) Nonconvergence of the fixed point method.

In the general case with p unknowns, the error $(\alpha - x^{(i)})$ now tends to be described by $B^{(i)}z$, where z is a constant vector, and $B = F'(\alpha)$ is the $p \times p$ Jacobian matrix. In this case the necessary condition for convergence is $\rho(B) = \max_i |\lambda_i| < 1$, where $\lambda_i, i = 1, \ldots, p$ are the eigen values of B. When convergence occurs, the rate of convergence in linear.

An interesting point to note is that the linear convergence of the fixed-point method becomes quadratic convergence when $|F'(\alpha)| = 0$, provided $F''(\alpha) \neq 0$. (For the p-variate case this requires that the Jacobian is a nilpotent matrix).

We end the section with the following proposition from Lange (1998) which gives sufficient conditions for the existence of a fixed point and convergence to it.

Proposition: Suppose that the function $F(x)$ defined on a closed interval I satisfying the conditions

1. $f(x) \in I$ whenever $x \in I$.
2. $|f(x) - f(y)| \leq \lambda |y - x|$ for any two points x and y in I.

Then, provided the Lipschitz constant λ is in $[0, 1)$, $F(x)$ has a unique fixed point $\alpha \in I$, and the functional iterates $x^{(i+1)} = F(x^{(i)})$ converge to α regardless of their starting point $x^{(0)} \in I$. In addition, the relation

$$|x^{(i)} - \alpha| = \frac{\lambda^n}{1-\lambda} |x^{(1)} - x^{(0)}|$$

gives the precise error estimate. □

In a broader sense, the functional iteration method can be extended to m previous iterates to get the m point iteration formula

$$X^{(i+1)} = F(x^{(i)}, x^{(i-1)}, \ldots, x^{(i-n+1)}). \tag{1.40}$$

Even more generally, the iteration function itself may depend i, in which case the iteration function will be called nonstationary.

See Thisted (1988), Hildebrand (1974) and Conte and De Boor (1980) for more discussion of the fixed point method. Also see Thisted for a general discussion of the parallel chord method.

1.5.3 Newton Raphson Algorithm

The Newton-Raphson algorithm is one of the most popular root solving techniques in many applications of the problem. Assume that the function $f(x)$ admits of a continuous derivative. Let $x^{(i)}$ be the current approximation to the root α of the equation $f(x) = 0$. Expanding $f(x)$ in a Taylor series about the current iterate gives

$$f(x) = f(x^{(i)}) + (x - x^{(i)}) f'(x^{(i)}) + (x - x^{(i)})^2 2 f''(x^*)$$

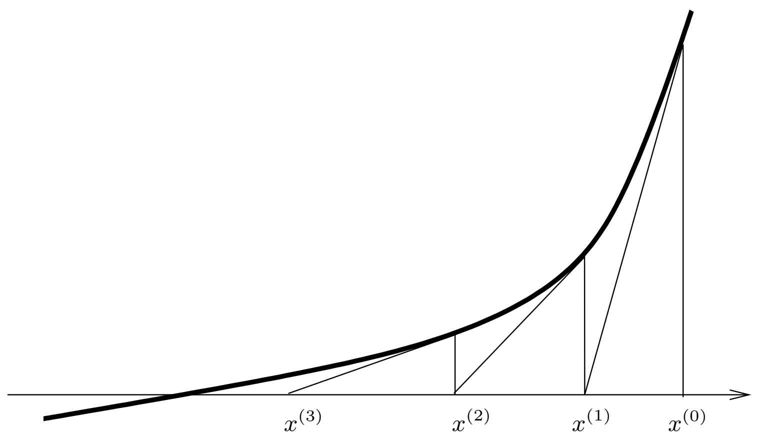

Figure 1.3: Convergence of the Newton-Raphson method.

where x^* is between x and $x^{(i)}$. Replacing $x = \alpha$, one gets

$$0 = f(\alpha) = f(x^{(i)}) + (\alpha - x^{(i)})f'(x^{(i)}) + (\alpha - x^{(i)})^2 2f''(x^*).$$

When $x^{(i)}$ is sufficiently close to α, the remainder term is neglected which leads to the approximation $\alpha \equiv x^{(i)} - f(x^{(i)})/f'(x^{(i)})$, and hence the next iterate is obtained as

$$x^{(i+1)} = x^{(i)} - \frac{f(x^{(i)})}{f'(x^{(i)})}.$$

Notice that from the perspective of functional iteration, the Newton-Raphson method may be rephrased as $x^{(i+1)} = F(x^{(i)})$, where $F(x) = x - f(x)/f'(x)$. Notice that at the true solution α this satisfies $F'(\alpha) = 0$, so that the method is quadratically convergent, one major reason for its wide popularity. Newton's method also requires that $f'(x) \neq 0$ at the true solution α. However it appears that the conditions necessary for the convergence of the Newton-Raphson algorithm are more commonly satisfied compared to the functional iteration method where the Lipschitz condition restricts the variability of the function near $f(x) = 0$. Figure 1.3 gives an example of the convergence of the Newton-Raphson method. The solid curve represents the $y = f(x)$ function.

1.5.4 Secant Method

Although the Newton-Raphson method is very efficient and is widely used, it requires the use of the first derivative $f'(x)$ of the function $f(x)$. When this derivative is difficult to determine, an efficient alternative would be to replace the derivative by a finite difference. Thus we can write

$$f'(x^i) = \frac{f(x^{(i)}) - f(x^{(i-1)})}{x^{(i)} - x^{(i-1)}}.$$

This iteration is called the *secant method* since it approximates the function $f(x)$ by a secant line through the two last iterates, rather than by the tangent at the very last iterate. The method does not require the use of the derivative of the function and hence is fairly easy to program. Given the two latest iterates $x^{(i-1)}$ and $x^{(i)}$, the secant method produces the next iterate as

$$x^{(i+1)} = x^{(i)} - f(x^{(i)}) \frac{x^{(i)} - x^{(i-1)}}{f(x^{(i)}) - f(x^{(i-1)})}.$$

Basically one can think of the next iterate in the secant method to be the root of the equation of the line obtained by joining the two current iterates. A natural extension may be to take the three latest iterates, fit a quadratic curve through them, and choose the root of the quadratic equation closest to the most recent iterate as the next iterate. This is called Muller's method. It may be shown (Muller, 1956) that the order of convergence of the ordinary secant method is 1.618, while that for Muller's method is 1.839. However, in most applications the added complexity for the Muller's method makes it unattractive. See Thisted (1988) for a general discussion.

Some variants of the secant method have been called the *regula falsi* method (or the method of false position). Unfortunately, the term has been used by various authors to refer to several slightly different (although similar) versions. We will use the name regula falsi to denote the method where two approximations $x^{(0)}$ and $x^{(1)}$ are known to bracket a root of $f(x)$ (so that $f(x^{(0)})f(x^{(1)}) < 0$). The chord joining $y^{(0)} = f(x^{(0)})$ and $y^{(1)} = f(x^{(1)})$ intersects the x axis at the point $x^{(2)}$. One then chooses $x^{(2)}$ and $x^{(i)}$, $i = 0$ or 1, such that $f(x^{(i)})f(x^{(2)}) < 0$, and continues the process. It is easily seen that the method will converge to a root between $x^{(0)}$ and $x^{(1)}$ for any continuous function $f(x)$. The method of regula falsi can also be viewed, using inverse interpolation, as a nonstationary functional iteration method. See Ralston and Rabinowitz (1978) for more details.

References

Abramowitz, M. and Stegun, I. A. (1964). *Handbook of Mathematical Functions*, Applied Mathematics Series, Vol. 55, Washington National Bureau of Standards (Reprinted 1968 by Dover Publications, New York).

Acton, F. S. (1970). *Numerical Methods That Work*, Harper and Row, New York.

Conte, S. and de Boor, C. (1980). *Elementary Numerical Analysis: An Algorithmic Approach*, McGraw-Hill, New York.

Dahlquist, G. and Bjorck, A. (1974). *Numerical Methods*, Prentice-Hall, Englewood Cliffs, New Jersey.

Hämmerlin, G and Hoffman, K.-H. (1991). *Numerical Mathematics*, Springer-Verlag, New York.

Henrici, P. (1982). *Essentials of Numerical Analysis with Pocket Calculator Demonstrations*, John Wiley, New York.

Hildebrand, F. B. (1974). *Introduction to Numerical Analysis*, 2nd ed., McGraw-Hill, New York.

Isaacson, E. and Keller, H. B. (1966). *Analysis of Numerical Methods*, John Wiley & Sons, New York

Jacques, I., and Judd, C. (1987). *Numerical Analysis*, Chapman and Hall, London.

Johnson, L. W., and Reiss, R. D. (1982). *Numerical Analysis*, 2nd Ed. , Addison Wesley, Reading, Massachusetts.

Kellison, S. G. (1975). *Fundamentals of Numerical Analysis.* Richard D. Irwin, Homewood, Illinois.

Kennedy, W. J., and Gentle, J. E. (1980). *Statistical Computing*, Mercel Dekker, New York.

Knuth, Donald E. (1997). *Seminumerical Algorithms*, 3rd. Ed., Vol 2. of *The Art of Computer Programming*, Addison-Wesley, Reading, Massachusetts.

Lange, K. (1998). *Numerical Analysis for Statisticians*, Springer-Verlag, New York.

Moore, R. (1966). *Interval Analysis*, Prentice Hall, Englewood Cliffs, New Jersey.

Muller, D. E. (1956). A method for solving algebraic equations using an automated computer, *Mathematical Tables and Aids Computation*, **10**, 208–215.

Ortega, J. M., (1990). *Numerical Analysis : A Second Course*, Society for Industrial and Applied Mathematics, Philadelphia.

Press, W. H., Flannery, B. P., Teukolsky, S. A., and Vetterling, W. T. (1986). *Numerical Recipes: The Art of Scientific Computing*, Cambridge University Press, Cambridge, UK.

Ralston, A. and Rabinowitz, P. (1978). *First Course in Numerical Analysis*, McGraw-Hill, New York.

Sterbenz, P. H. (1974). *Floating-Point Computation*, Prentice Hall, Englewood Cliffs, New Jersey.

Stoer, J. and Bulirsch, R. (1980). *Introduction to Numerical Analysis*, Springer-Verlag, New York.

Thisted, R. (1988). *Elements of Statistical Computing: Numerical Computation*, Chapman and Hall, New York.