Chapter 4. Integration of Functions

4.0 Introduction

Numerical integration, which is also called *quadrature*, has a history extending back to the invention of calculus and before. The fact that integrals of elementary functions could not, in general, be computed analytically, while derivatives *could* be, served to give the field a certain panache, and to set it a cut above the arithmetic drudgery of numerical analysis during the whole of the 18th and 19th Centuries.

With the invention of automatic computing, quadrature became just one numerical task among many, and not a very interesting one at that. Automatic computing, even the most primitive sort involving desk calculators and rooms full of "computers" (that were, until the 1950's, people rather than machines), opened to feasibility the much richer field of numerical integration of differential equations. Quadrature is merely the simplest special case: The evaluation of the integral

$$I = \int_a^b f(x)dx \tag{4.0.1}$$

is precisely equivalent to solving for the value $I \equiv y(b)$ the differential equation

$$\frac{dy}{dx} = f(x) \tag{4.0.2}$$

with the boundary condition

$$y(a) = 0 \tag{4.0.3}$$

Chapter 15 of this book deals with the numerical integration of differential equations. In that chapter, much emphasis is given to the concept of "variable" or "adaptive" choices of step-size. We will not, therefore, develop that material here. If the function that you propose to integrate is sharply concentrated in one or more peaks, or if its shape is not readily characterized

by a single length-scale, then it is likely that you should cast the problem in the form of (4.0.2) – (4.0.3) and use the methods of Chapter 15.

The quadrature methods in this chapter are based, in one way or another, on the obvious device of adding up the value of the integrand at a sequence of abscissas within the range of integration. The game is to obtain the integral as accurately as possible with the smallest number of function evaluations of the integrand. Just as in the case of interpolation (Chapter 3), one has the freedom to choose methods of various *orders*, with higher order sometimes, but not always, giving higher accuracy. "Romberg integration," which is discussed in §4.3 is a general formalism for making use of integration methods of a variety of different orders, and we recommend it highly.

Apart from the methods of this chapter and of Chapter 15, there are yet other methods for obtaining integrals. One important class is based on function approximation. We discuss explicitly the integration of Chebyshev-approximated functions in §5.7. Although not explicitly discussed here, you ought to be able to figure out how to do *cubic spline quadrature* using the output of the routine SPLINE in §3.3. (Hint: Integrate equation 3.3.3 over x analytically. See Forsythe *et al.*)

Multidimensional integrals are another whole multidimensional bag of worms. Section 4.6 is an introductory discussion in this chapter; the important technique of *Monte-Carlo integration* is treated in Chapter 7.

REFERENCES AND FURTHER READING:

Carnahan, Brice, Luther, H.A., and Wilkes, James O. 1969, *Applied Numerical Methods* (New York: Wiley), Chapter 2.

Isaacson, Eugene, and Keller, Herbert B. 1966, *Analysis of Numerical Methods* (New York: Wiley), Chapter 7.

Acton, Forman S. 1970, *Numerical Methods That Work* (New York: Harper and Row), Chapter 4.

Stoer, J., and Bulirsch, R. 1980, *Introduction to Numerical Analysis* (New York: Springer-Verlag), Chapter 3.

Ralston, Anthony, and Rabinowitz, Philip. 1978, *A First Course in Numerical Analysis*, 2nd ed. (New York: McGraw-Hill), Chapter 4.

Dahlquist, Germund, and Bjorck, Ake. 1974, *Numerical Methods* (Englewood Cliffs, N.J.: Prentice-Hall), §7.4.

Forsythe, George E., Malcolm, Michael A., and Moler, Cleve B. 1977, *Computer Methods for Mathematical Computations* (Englewood Cliffs, N.J.: Prentice-Hall), §5.2, p.89.

4.1 Classical Formulas for Equally-Spaced Abscissas

Where would any book on numerical analysis be without Mr. Simpson and his "rule"? The classical formulas for integrating a function whose value is known at equally-spaced steps have a certain elegance about them, and they are redolent with historical association. Through them, the modern

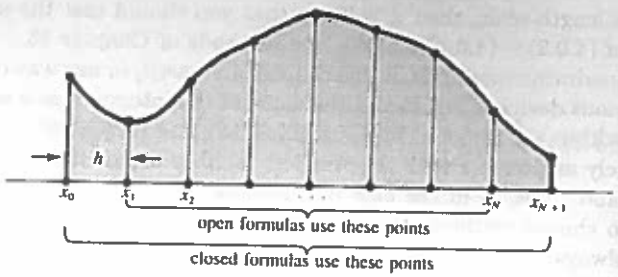

Figure 4.1.1. Quadrature formulas with equally spaced abscissas compute the integral of a function between x_0 and x_{N+1}. Closed formulas evaluate the function on the boundary points, while open formulas refrain from doing so (useful if the evaluation algorithm breaks down on the boundary points).

numerical analyst communes with the spirits of his or her predecessors back across the centuries, as far as the time of Newton, if not farther. Alas, times *do* change; with the exception of two of the most modest formulas ("extended trapezoidal rule," equation 4.1.11, and "extended midpoint rule," equation 4.1.19, see §4.2), the classical formulas are almost entirely useless. They are museum pieces, but beautiful ones.

Some notation: We have a sequence of abscissas, denoted $x_0, x_1, \ldots, x_N$, x_{N+1} which are spaced apart by a constant step h,

$$x_i = x_0 + ih \qquad i = 0, 1, \ldots, N+1 \tag{4.1.1}$$

A function $f(x)$ has known values at the x_i's,

$$f(x_i) \equiv f_i. \tag{4.1.2}$$

We want to integrate the function $f(x)$ between a lower limit a and an upper limit b, where a and b are each equal to one or the other of the x_i's. An integration formula that uses the value of the function at the endpoints, $f(a)$ or $f(b)$, is called a *closed* formula. Occasionally, we want to integrate a function whose value at one or both endpoints is difficult to compute (e.g., the computation of f goes to a limit of zero over zero there, or worse yet has an integrable singularity there). In this case we want an *open* formula, which estimates the integral using only x_i's strictly *between* a and b (see Figure 4.1.1).

The basic building blocks of the classical formulas are rules for integrating a function over a small number of intervals. As that number increases, we can find rules that are exact for polynomials of increasingly high order. (Keep in mind that higher order does not always imply higher accuracy in real cases.) A sequence of such closed formulas is now given.

Closed Newton-Cotes Formulas

Trapezoidal rule:

$$\int_{x_1}^{x_2} f(x)dx = h\left[\frac{1}{2}f_1 + \frac{1}{2}f_2\right] \quad + O(h^3 f'') \tag{4.1.3}$$

Here the error term $O(\,)$ signifies that the true answer differs from the estimate by an amount that is the product of some numerical coefficient times h^3 times the value of the functions's second derivative somewhere in the interval of integration. The coefficient is knowable, and it can be found in all the standard references on this subject. The point at which the second derivative is to be evaluated is, however, unknowable. If we knew it, we could evaluate the function there and have a higher-order method! Since the product of a knowable and an unknowable is unknowable, we will streamline our formulas and write only $O(\,)$, instead of the coefficient.

Equation (4.1.3) is a two-point formula (x_1 and x_2). It is exact for polynomials up to and including degree 1, i.e. $f(x) = x$. One anticipates that there is a three-point formula exact up to polynomials of degree 2. This is true; moreover, by a cancellation of coefficients due to left-right symmetry of the formula, the three-point formula is exact for polynomials up to and including degree 3, i.e. $f(x) = x^3$:

Simpson's rule:

$$\int_{x_1}^{x_3} f(x)dx = h\left[\frac{1}{3}f_1 + \frac{4}{3}f_2 + \frac{1}{3}f_3\right] \quad + O(h^5 f^{(4)}) \tag{4.1.4}$$

Here $f^{(4)}$ means the fourth derivative of the function f evaluated at an unknown place in the interval. Note also that the formula gives the integral over an interval of size $2h$, so the coefficients add up to 2.

There is no lucky cancellation in the four-point formula, so it is also exact for polynomials up to and including degree 3.

Simpson's $\frac{3}{8}$ rule:

$$\int_{x_1}^{x_4} f(x)dx = h\left[\frac{3}{8}f_1 + \frac{9}{8}f_2 + \frac{9}{8}f_3 + \frac{3}{8}f_4\right] \quad + O(h^5 f^{(4)}) \tag{4.1.5}$$

The five-point formula again benefits from a cancellation:

Bode's rule:

$$\int_{x_1}^{x_5} f(x)dx = h[\frac{14}{45}f_1 + \frac{64}{45}f_2 + \frac{24}{45}f_3 + \frac{64}{45}f_4 + \frac{14}{45}f_5] \quad + O(h^7 f^{(6)}) \quad (4.1.6)$$

This is exact for polynomials up to and including degree 5.

At this point the formulas stop being named after famous personages, so we will not go any further. Consult Abramowitz and Stegun for additional formulas in the sequence.

Extrapolative Formulas for a Single Interval

We are going to depart from historical practice for a moment. Most texts would give, at this point, a sequence of "Newton-Cotes Formulas of Open Type." Here is an example:

$$\int_{x_0}^{x_5} f(x)dx = h[\frac{55}{24}f_1 + \frac{5}{24}f_2 + \frac{5}{24}f_3 + \frac{55}{24}f_4] \quad + O(h^5 f^{(4)})$$

Notice that the integral from $a = x_0$ to $b = x_5$ is estimated, using only the interior points x_1, x_2, x_3, x_4. In our opinion, formulas of this type are not useful for the reasons that (i) they cannot usefully be strung together to get "extended" rules, as we are about to do with the closed formulas, and (ii) for all other possible uses they are dominated by the Gaussian integration formulas which we will introduce in §4.3.

Instead of the Newton-Cotes open formulas, let us set out the formulas for estimating the integral in the single interval from x_0 to x_1, using values of the function f at $x_1, x_2, \ldots$. These will be useful building blocks for the "extended" open formulas.

$$\int_{x_0}^{x_1} f(x)dx = h[f_1] \quad + O(h^2 f') \quad (4.1.7)$$

$$\int_{x_0}^{x_1} f(x)dx = h[\frac{3}{2}f_1 - \frac{1}{2}f_2] \quad + O(h^3 f'') \quad (4.1.8)$$

$$\int_{x_0}^{x_1} f(x)dx = h[\frac{23}{12}f_1 - \frac{16}{12}f_2 + \frac{5}{12}f_3] \quad + O(h^4 f^{(3)}) \quad (4.1.9)$$

$$\int_{x_0}^{x_1} f(x)dx = h[\frac{55}{24}f_1 - \frac{59}{24}f_2 + \frac{37}{24}f_3 - \frac{9}{24}f_4] \quad + O(h^5 f^{(4)}) \quad (4.1.10)$$

Perhaps a word here would be in order about how formulas like the above can be derived. There are elegant ways, but the most straightforward is to write down the basic form of the formula, replacing the numerical coefficients with unknowns, say p, q, r, s. Without loss of generality take $x_0 = 0$ and $x_1 = 1$, so $h = 1$. Substitute in turn for $f(x)$ (and for f_1, f_2, f_3, f_4) the functions $f(x) = 1$, $f(x) = x$, $f(x) = x^2$, and $f(x) = x^3$. Doing the integral in each case reduces the right-hand side to a number, and the left-hand side to a linear equation for the unknowns p, q, r, s. Solving the four equations produced in this way gives the coefficients.

Extended Formulas (Closed)

If we use equation (4.1.3) $N - 1$ times, to do the integration in the intervals $(x_1, x_2), (x_2, x_3), \ldots, (x_{N-1}, x_N)$, and then add the results, we obtain an "extended" or "composite" formula for the integral from x_1 to x_N.

Extended trapezoidal rule:

$$\int_{x_1}^{x_N} f(x)dx = h[\frac{1}{2}f_1 + f_2 + f_3 + \cdots + f_{N-1} + \frac{1}{2}f_N] \quad + O\left(\frac{(b-a)^3 f''}{N^2}\right) \quad (4.1.11)$$

Here we have written the error estimate in terms of the interval $b - a$ and the number of points N instead of in terms of h. This is clearer, since one is usually holding a and b fixed and wanting to know (e.g.) how much the error will be decreased by taking twice as many steps (in this case, it is by a factor of 4). In subsequent equations we will show *only* the scaling of the error term with the number of steps.

For reasons which will not become clear until §4.2, equation (4.1.11) is in fact the most important equation in this section, the basis for most practical quadrature schemes.

The *extended formula of order* $1/N^3$ is:

$$\int_{x_1}^{x_N} f(x)dx = h[\frac{5}{12}f_1 + \frac{13}{12}f_2 + f_3 + f_4 + \cdots + f_{N-2} + \frac{13}{12}f_{N-1} + \frac{5}{12}f_N] \quad + O\left(\frac{1}{N^3}\right) \quad (4.1.12)$$

(We will see in a moment where this comes from.)

If we apply equation (4.1.4) to successive, non-overlapping *pairs* of intervals, we get the *extended Simpson's rule:*

$$\int_{x_1}^{x_N} f(x)dx = h[\frac{1}{3}f_1 + \frac{4}{3}f_2 + \frac{2}{3}f_3 + \frac{4}{3}f_4 + \cdots + \frac{2}{3}f_{N-2} + \frac{4}{3}f_{N-1} + \frac{1}{3}f_N] + O\left(\frac{1}{N^4}\right) \tag{4.1.13}$$

Notice that the 2/3, 4/3 alternation continues throughout the interior of the evaluation. Many people believe that the wobbling alternation somehow contains information about the integral of their function which is not apparent to mortal eyes. In fact, the alternation is an artifact of using only the building block (4.1.4) and not (4.1.5), which is of the same order. If we take one three-interval step using (4.1.5) and then subsequent two-interval steps of (4.1.4), the alternation of 2/3, 4/3 will be exactly out of phase with that in (4.1.13). Averaging the result with (4.1.13) gives the *alternative extended Simpson's rule:*

$$\int_{x_1}^{x_N} f(x)dx = h[\frac{17}{48}f_1 + \frac{59}{48}f_2 + \frac{43}{48}f_3 + \frac{49}{48}f_4 + f_5 + f_6 + \cdots + f_{N-4} + \frac{49}{48}f_{N-3} + \frac{43}{48}f_{N-2} + \frac{59}{48}f_{N-1} + \frac{17}{48}f_N] + O\left(\frac{1}{N^4}\right) \tag{4.1.14}$$

This is a somewhat ugly formula because of the peculiar rational coefficients, but your computer won't care about that.

We can now tell you where equation (4.1.12) came from. It is Simpson's extended rule averaged with the sequence: one step of trapezoidal (4.1.3) followed by Simpson's extended rule. The trapezoidal step is *two* orders lower than Simpson's rule; however, its contribution to the integral goes down as an additional power of N (since it is used only twice, not N times). This makes the resulting formula of degree *one* less than Simpson. It is likewise true that, in constructing (4.1.14), we could have used a starting step of one degree *less* than Simpson, instead of using (4.1.5). However there *is* no step of degree one less than Simpson!

Extended Formulas (Open and Semi-open)

We can construct open and semi-open extended formulas by adding the closed formulas (4.1.11) - (4.1.14), evaluated for the second and subsequent steps, to the extrapolative open formulas for the first step, (4.1.7) - (4.1.10). As discussed immediately above, it is consistent to use an end step that is of

one order lower than the (repeated) interior step. The resulting formulas for an interval open at both ends are as follows:

Equations (4.1.7) and (4.1.11) give

$$\int_{x_1}^{x_N} f(x)dx = h[\frac{3}{2}f_2 + f_3 + f_4 + \cdots + f_{N-2} + \frac{3}{2}f_{N-1}] + O\left(\frac{1}{N^2}\right) \tag{4.1.15}$$

Equations (4.1.8) and (4.1.12) give

$$\int_{x_1}^{x_N} f(x)dx = h[\frac{23}{12}f_2 + \frac{7}{12}f_3 + f_4 + f_5 + \cdots + f_{N-3} + \frac{7}{12}f_{N-2} + \frac{23}{12}f_{N-1}] + O\left(\frac{1}{N^3}\right) \tag{4.1.16}$$

Equations (4.1.9) and (4.1.13) give

$$\int_{x_1}^{x_N} f(x)dx = h[\frac{27}{12}f_2 + 0 + \frac{13}{12}f_4 + \frac{4}{3}f_5 + \cdots + \frac{4}{3}f_{N-4} + \frac{13}{12}f_{N-3} + 0 + \frac{27}{12}f_{N-1}] + O\left(\frac{1}{N^4}\right) \tag{4.1.17}$$

The interior points alternate 4/3 and 2/3. If we want to avoid this alternation, we can combine equations (4.1.9) and (4.1.14), giving

$$\int_{x_1}^{x_N} f(x)dx = h[\frac{109}{48}f_2 - \frac{5}{48}f_3 + \frac{63}{48}f_4 + \frac{49}{48}f_5 + f_6 + f_7 + \cdots + f_{N-5} + \frac{49}{48}f_{N-4} + \frac{63}{48}f_{N-3} - \frac{5}{48}f_{N-2} + \frac{109}{48}f_{N-1}] + O\left(\frac{1}{N^4}\right) \tag{4.1.18}$$

We should mention in passing another extended open formula, for use where the limits of integration are located halfway between tabulated abscissas. This one is known as the *extended midpoint rule*, and is accurate to the

same order as (4.1.15):

$$\int_{x_1}^{x_N} f(x)dx = h[f_{3/2} + f_{5/2} + f_{7/2} + \cdots + f_{N-3/2} + f_{N-1/2}] + O\left(\frac{1}{N^2}\right) \tag{4.1.19}$$

There are also formulas of higher order for this situation, but we will refrain from giving them.

The *semi-open formulas* are just the obvious combinations of equations (4.1.11) through (4.1.14) with (4.1.15) through (4.1.18) respectively. At the closed end of the integration, use the weights from the former equations; at the open end use the weights from the latter equations. One example should give the idea, the formula with error term decreasing as $1/N^3$ which is closed on the right and open on the left:

$$\int_{x_1}^{x_N} f(x)dx = h[\frac{23}{12}f_2 + \frac{7}{12}f_3 + f_4 + f_5 + \cdots + f_{N-2} + \frac{13}{12}f_{N-1} + \frac{5}{12}f_N] + O\left(\frac{1}{N^3}\right) \tag{4.1.20}$$

REFERENCES AND FURTHER READING:

Abramowitz, Milton, and Stegun, Irene A. 1964, *Handbook of Mathematical Functions*, Applied Mathematics Series, vol. 55 (Washington: National Bureau of Standards; reprinted 1968 by Dover Publications, New York), §25.4.

Isaacson, Eugene, and Keller, Herbert B. 1966, *Analysis of Numerical Methods* (New York: Wiley), §7.1.

4.2 Elementary Algorithms

Our starting point is equation (4.1.11), the extended trapezoidal rule. There are two facts about the trapezoidal rule which make it the starting point for a variety of algorithms. One fact is rather obvious, while the second is rather "deep."

The obvious fact is that, for a fixed function $f(x)$ to be integrated between fixed limits a and b, one can double the number of intervals in the extended trapezoidal rule without losing the benefit of previous work. The coarsest implementation of the trapezoidal rule is to average the function at its endpoints a and b. The first stage of refinement is to add to this average the value of the function at the halfway point. The second stage of refinement is to add the values at the 1/4 and 3/4 points. And so on (see Figure 4.2.1).

Without further ado we can write a routine with this kind of logic in it:

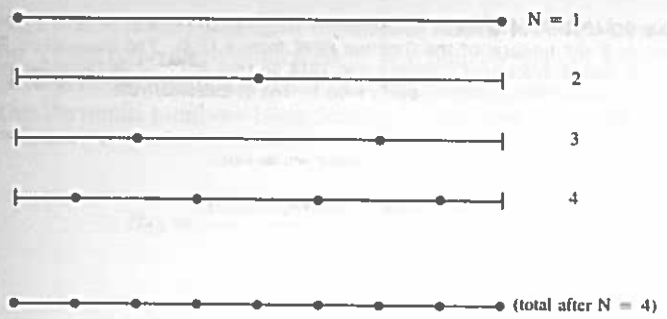

Figure 4.2.1. Sequential calls to the routine TRAPZD incorporate the information from previous calls and evaluate the integrand only at those new points necessary to refine the grid. The bottom line shows the totality of function evaluations after the fourth call. The routine QSIMP, by weighting the intermediate results, transforms the trapezoid rule into Simpson's rule with essentially no additional overhead.

```
      SUBROUTINE TRAPZD(FUNC,A,B,S,N)
        This routine computes the N'th stage of refinement of an extended trapezoidal rule. FUNC is
        input as the name of the function to be integrated between limits A and B, also input. When
        called with N=1, the routine returns as S the crudest estimate of ∫_a^b f(x)dx. Subsequent
        calls with N=2,3,... (in that sequential order) will improve the accuracy of S by adding
        2^(N-2) additional interior points. S should not be modified between sequential calls.
      IF (N.EQ.1) THEN
        S=0.5*(B-A)*(FUNC(A)+FUNC(B))
        IT=1                          IT is the number of points to be added on the next call.
      ELSE
        TNM=IT
        DEL=(B-A)/TNM                 This is the spacing of the points to be added.
        X=A+0.5*DEL
        SUM=0.
        DO 11 J=1,IT
          SUM=SUM+FUNC(X)
          X=X+DEL
        11 CONTINUE
        S=0.5*(S+(B-A)*SUM/TNM)       This replaces S by its refined value.
        IT=2*IT
      ENDIF
      RETURN
      END
```

The above routine (TRAPZD) is a work horse which can be harnessed in several ways. The simplest and crudest is to integrate a function by the extended trapezoidal rule where you know in advance (we can't imagine how!) the number of steps you want. If you want $2^M + 1$, you can accomplish this by the fragment

```
      DO J=1,M+1
        CALL TRAPZD(FUNC,A,B,S,J)
      ENDDO
```

with the answer returned in S.

Much better, of course, is to refine the trapezoidal rule until some specified degree of accuracy has been achieved:

```
      SUBROUTINE QTRAP(FUNC,A,B,S)
          Returns as S the integral of the function FUNC from A to B. The parameters EPS can be
          set to the desired fractional accuracy and JMAX so that 2**(JMAX-1) is the maximum allowed
          number of steps. Integration is performed by the trapezoidal rule.
      PARAMETER (EPS=1.E-6, JMAX=20)
      OLDS=-1.E30                    Any number that is unlikely to be the average of the function at its
      DO 11 J=1,JMAX                   endpoints will do here.
          CALL TRAPZD(FUNC,A,B,S,J)
          IF (ABS(S-OLDS).LT.EPS*ABS(OLDS)) RETURN
          OLDS=S
        11 CONTINUE
      PAUSE 'Too many steps.'
      END
```

Unsophisticated as it is, routine QTRAP is in fact a fairly robust way of doing integrals of functions that are not very smooth. Increased sophistication will usually translate into a higher order method whose efficiency will be greater only for sufficiently smooth integrands. QTRAP is the method of choice, e.g., for an integrand which is a function of a variable that is linearly interpolated between measured data points. Be sure that you do not require too stringent an EPS, however: if QTRAP takes too many steps in trying to achieve your required accuracy, accumulated roundoff errors may start increasing, and the routine may never converge. The value 10^{-6} used above is just on the edge of trouble for most 32-bit machines; it is achievable when the convergence is moderately rapid, but not otherwise.

We come now to the "deep" fact about the extended trapezoidal rule, equation (4.1.11). It is this: the error of the approximation, which begins with a term of order $1/N^2$ is in fact *entirely even* when expressed in powers of $1/N$. This follows directly from the *Euler-Maclaurin Summation Formula*,

$$\int_{x_1}^{x_N} f(x)dx = h\left[\frac{1}{2}f_1 + f_2 + f_3 + \cdots + f_{N-1} + \frac{1}{2}f_N\right] - \frac{B_2h^2}{2!}(f'_N - f'_1) - \cdots - \frac{B_{2k}h^{2k}}{(2k)!}(f_N^{(2k-1)} - f_1^{(2k-1)}) - \cdots \tag{4.2.1}$$

Here B_{2k} is a *Bernoulli number*, defined by the generating function

$$\frac{t}{e^t - 1} = \sum_{n=0}^{\infty} B_n \frac{t^n}{n!} \tag{4.2.2}$$

with the first few even values (odd values vanish except for $B_1 = -1/2$)

$$B_0 = 1 \quad B_2 = \frac{1}{6} \quad B_4 = -\frac{1}{30} \quad B_6 = \frac{1}{42}$$
$$B_8 = -\frac{1}{30} \quad B_{10} = \frac{5}{66} \quad B_{12} = \frac{691}{2730} \tag{4.2.3}$$

Equation (4.2.1) is not a convergent expansion, but rather only an asymptotic expansion whose error when truncated at any point is always less than twice the magnitude of the first neglected term. The reason that it is not convergent is that the Bernoulli numbers become very large, e.g.

$$B_{50} = \frac{495057205241079648212477525}{66}$$

The key point is that only even powers of h occur in the error series of (4.2.1). This fact is not, in general, shared by the higher order quadrature rules in §4.1. For example, equation (4.1.13) has an error series beginning with $\mathcal{O}(1/N^3)$, but continuing with all subsequent powers of N: $1/N^4$, $1/N^5$, etc.

Suppose we evaluate (4.1.11) with N steps, getting a result S_N, and then again with $2N$ steps, getting a result S_{2N}. (This is done by any two consecutive calls of TRAPZD.) The leading error term in the second evaluation will be 1/4 the size of the error in the first evaluation. Therefore the combination

$$S = \frac{4}{3}S_{2N} - \frac{1}{3}S_N \tag{4.2.4}$$

will cancel out the leading order error term. But there *is* no error term of order $1/N^3$, by (4.2.1). The surviving error is of order $1/N^4$, the same as Simpson's rule. In fact, it should not take long for you to see that (4.2.4) is *exactly* Simpson's rule (4.1.13), alternating 2/3's, 4/3's and all. This is the preferred method for evaluating that rule, and we can write it as a routine exactly analogous to QTRAP above:

```
      SUBROUTINE QSIMP(FUNC,A,B,S)
          Returns as S the integral of the function FUNC from A to B. The parameters EPS can be
          set to the desired fractional accuracy and JMAX so that 2**(JMAX-1) is the maximum allowed
          number of steps. Integration is performed by Simpson's rule.
      PARAMETER (EPS=1.E-6, JMAX=20)
      OST=-1.E30
      OS= -1.E30
      DO 11 J=1,JMAX
          CALL TRAPZD(FUNC,A,B,ST,J)
          S=(4.*ST-OST)/3.                Compare equation (4.2.4), above.
          IF (ABS(S-OS).LT.EPS*ABS(OS)) RETURN
          OS=S
          OST=ST
        11 CONTINUE
      PAUSE 'Too many steps.'
      END
```

The routine QSIMP will in general be more efficient than QTRAP (i.e., require fewer function evaluations) when the function to be integrated has a finite 4^{th} derivative (i.e., a continuous 3^{rd} derivative). The combination of QSIMP and its necessary work-horse TRAPZD is a good one for light-duty work where you have to type the routines into an unfamiliar machine.

REFERENCES AND FURTHER READING:

Stoer, J., and Bulirsch, R. 1980, *Introduction to Numerical Analysis* (New York: Springer-Verlag), §3.3.

Dahlquist, Germund, and Bjorck, Ake. 1974, *Numerical Methods* (Englewood Cliffs, N.J.: Prentice-Hall), §§7.4.1-7.4.2.

Forsythe, George E., Malcolm, Michael A., and Moler, Cleve B. 1977, *Computer Methods for Mathematical Computations* (Englewood Cliffs, N.J.: Prentice-Hall), §5.3.

4.3 Romberg Integration

We can view Romberg's method as the natural generalization of the routine QSIMP in the last section to integration schemes which are of higher order than Simpson's rule. The basic idea is to use the results from k successive refinements of the extended trapezoidal rule (implemented in TRAPZD) to remove all terms in the error series up to but not including $O(N^{2k})$. QSIMP is the case of $k = 2$. This is one example of a very general idea which goes by the name of *Richardson's deferred approach to the limit*: perform some numerical algorithm for various values of a parameter h, and then extrapolate the result to the continuum limit $h = 0$.

Equation (4.2.4), which subtracts off the leading error term, is a special case of polynomial extrapolation. In the more general Romberg case, we can use Neville's algorithm (see §3.1) to extrapolate the successive refinements to zero step-size. Neville's algorithm can in fact be coded very concisely within a Romberg integration routine. For clarity of the program, however, it seems better to do the extrapolation by subroutine call to POLINT, already given in §3.1.

```
      SUBROUTINE QROMB(FUNC,A,B,SS)
        Returns as S the integral of the function FUNC from A to B. Integration is performed by
        Romberg's method of order 2K, where, e.g., K=2 is Simpson's rule.
      PARAMETER (EPS=1.E-6, JMAX=20, JMAXP=JMAX+1, K=5, KM=K-1)
        Here EPS is the fractional accuracy desired, as determined by the extrapolation error
        estimate; JMAX limits the total number of steps; K is the number of points used in the
        extrapolation.
      DIMENSION S(JMAXP),H(JMAXP)  These store the successive trapezoidal approximations and their rel-
      H(1)=1.                                ative step-sizes.
      DO 11 J=1,JMAX
        CALL TRAPZD(FUNC,A,B,S(J),J)
        IF (J.GE.K) THEN
          CALL POLINT(H(J-KM),S(J-KM),K,0.,SS,DSS)
          IF (ABS(DSS).LT.EPS*ABS(SS)) RETURN
        ENDIF
        S(J+1)=S(J)
        H(J+1)=0.25*H(J)          This is a key step: The factor is 0.25 even though the step-size
      11 CONTINUE                 is decreased by only 0.5. This makes the extrapolation a
      PAUSE 'Too many steps.'     polynomial in $h^2$ as allowed by equation (4.2.1), not just a
      END                         polynomial in $h$.
```


The routine QROMB, along with its required TRAPZD and POLINT, is quite powerful for sufficiently smooth (e.g., analytic) integrands, integrated over intervals which contain no singularities, and where the endpoints are also nonsingular. QROMB, in such circumstances, takes many, *many* fewer function evaluations than either of the routines in §4.2. For example, the integral

$$\int_0^2 x^4 \log(x + \sqrt{x^2 + 1})dx$$

converges (with parameters as shown above) on the very first extrapolation, after just 5 calls to TRAPZD, while QSIMP requires 8 calls (8 times as many evaluations of the integrand) and QTRAP requires 13 calls (making 256 times as many evaluations of the integrand).

REFERENCES AND FURTHER READING:

Stoer, J., and Bulirsch, R. 1980, *Introduction to Numerical Analysis* (New York: Springer-Verlag), §§3.4-3.5.

Dahlquist, Germund, and Bjorck, Ake. 1974, *Numerical Methods* (Englewood Cliffs, N.J.: Prentice-Hall), §§7.4.1-7.4.2.

Ralston, Anthony, and Rabinowitz, Philip. 1978, *A First Course in Numerical Analysis*, 2nd ed. (New York: McGraw-Hill), §4.10-2.

4.4 Improper Integrals

For our present purposes, an integral will be "improper" if it has any of the following problems:

- its integrand goes to a finite limiting value at finite upper and lower limits, but cannot be evaluated *right on* one of those limits [e.g., $\sin(x)/x$ at $x = 0$]
- its upper limit is ∞ , or its lower limit is $-\infty$
- it has an integrable singularity at either limit (e.g., $x^{-1/2}$ at $x = 0$)
- it has an integrable singularity at a known place between its upper and lower limits

- it has an integrable singularity at an unknown place between its upper and lower limits

If an integral is infinite (e.g. $\int_1^\infty x^{-1}dx$), or does not exist in a limiting sense [e.g. $\int_{-\infty}^{\infty}\cos(x)dx$], we do not call it improper; we call it impossible. No amount of clever algorithmics will return a meaningful answer to an ill-posed problem.

In this section we will generalize the techniques of the preceding two sections to cover the first four problems on the above list. The fifth problem, singularity at unknown location, can really only be handled by the use of a variable-step-size differential equation integration routine, as will be given in Chapter 15.

We need a work-horse like the extended trapezoidal rule (equation 4.1.11), but one which is an *open* formula in the sense of §4.1., i.e. does not require the integrand to be evaluated at the endpoints. Equation (4.1.19), the extended midpoint rule, is the best choice. The reason is that (4.1.19) shares with (4.1.11) the "deep" property of having an error series that is entirely even in h. Indeed there is a formula, not as well known as it ought to be, called the *Second Euler-Maclaurin summation formula*,

$$\begin{aligned}\int_{x_1}^{x_N} f(x)dx = h[f_{3/2} + f_{5/2} + f_{7/2} + \cdots + f_{N-3/2} + f_{N-1/2}] \\ - \frac{B_2 h^2}{4}(f'_N - f'_1) - \cdots \\ - \frac{B_{2k}h^{2k}}{(2k)!}(1 - 2^{-2k+1})(f_N^{(2k-1)} - f_1^{(2k-1)})\end{aligned} \tag{4.4.1}$$

This equation can be derived by writing out (4.2.1) with step-size h, then writing it out again with step-size $h/2$, then subtracting twice the second from the first.

It is not possible to double the number of steps in the extended midpoint rule and still have the benefit of previous function evaluations (try it!). However, it is possible to *triple* the number of steps and do so. Shall we do this, or double and accept the loss? On the average, tripling does a factor $\sqrt{3}$ of unnecessary work, since the "right" number of steps for a desired accuracy criterion may in fact fall anywhere in the logarithmic interval implied by tripling. For doubling, the factor is only $\sqrt{2}$, but we lose an extra factor of 2 in being unable to use all the previous evaluations. Since $1.732 < 2 \times 1.414$, it is better to triple.

Here is the resulting routine, which is directly comparable to TRAPZD.

```
SUBROUTINE MIDPNT(FUNC,A,B,S,N)
    This routine computes the N'th stage of refinement of an extended midpoint rule. FUNC is
    input as the name of the function to be integrated between limits A and B, also input. When
    called with N=1, the routine returns as S the crudest estimate of ∫ᵃᵇ f(x)dx. Subsequent
    calls with N=2,3,... (in that sequential order) will improve the accuracy of S by adding
    (2/3)×3^(N-1) additional interior points. S should not be modified between sequential calls.
IF (N.EQ.1) THEN
    S=(B-A)*FUNC(0.5*(A+B))
```



```
    IT=1                                  2×IT points will be added on the next refinement.
ELSE
    TNM=IT
    DEL=(B-A)/(3.*TNM)
    DDEL=DEL+DEL                          The added points alternate in spacing between DEL and DDEL.
    X=A+0.5*DEL
    SUM=0.
    DO 11 J=1,IT
        SUM=SUM+FUNC(X)
        X=X+DDEL
        SUM=SUM+FUNC(X)
        X=X+DEL
    11 CONTINUE
    S=(S+(B-A)*SUM/TNM)/3.                The new sum is combined with the old integral to give a refined
    IT=3*IT                                   integral.
ENDIF
RETURN
END
```

The routine MIDPNT can exactly replace TRAPZD in a driver routine like QTRAP (§4.2); one simply changes CALL TRAPZD to CALL MIDPNT, and perhaps also decreases the parameter JMAX since $3^{\mathrm{JMAX}-1}$ (from step tripling) is a much larger number than $2^{\mathrm{JMAX}-1}$ (step doubling).

The open formula implementation analogous to Simpson's rule (QSIMP in §4.2) substitutes MIDPNT for TRAPZD and decreases JMAX as above, but now also changes the extrapolation step to be

```
    S=(9.*ST-OST)/8.
```

since, when the number of steps is tripled, the error decreases to $1/9^{th}$ its size, not $1/4^{th}$ as with step doubling.

Either the modified QTRAP or the modified QSIMP will fix the first problem on the list at the beginning of this section. Yet more sophisticated is to generalize Romberg integration in like manner:

```
SUBROUTINE QROMO(FUNC,A,B,SS,CHOOSE)
    Romberg integration on an open interval. Returns as SS the integral of the function FUNC
    from A to B, using any specified integrating subroutine CHOOSE and Romberg's method.
    Normally CHOOSE will be an open formula, not evaluating the function at the endpoints. It
    is assumed that CHOOSE triples the number of steps on each call, and that its error series
    contains only even powers of the number of steps. The routines MIDPNT, MIDINF, MIDSQL,
    MIDSQU, are possible choices for CHOOSE.
PARAMETER (EPS=1.E-6, JMAX=14, JMAXP=JMAX+1, K=5, KM=K-1)
    The parameters have the same meaning as in QROMB.
DIMENSION S(JMAXP),H(JMAXP)
H(1)=1.
DO 11 J=1,JMAX
    CALL CHOOSE(FUNC,A,B,S(J),J)
    IF (J.GE.K) THEN
        CALL POLINT(H(J-KM),S(J-KM),K,0.,SS,DSS)
        IF (ABS(DSS).LT.EPS*ABS(SS)) RETURN
    ENDIF
    S(J+1)=S(J)
    H(J+1)=H(J)/9.                        This is where the assumption of step tripling and an even error series
11  CONTINUE                                  is used.
PAUSE 'Too many steps.'
END
```

The differences between QROMO and QROMB (§4.3) are so slight that it is perhaps gratuitous to list QROMO in full. It, however, is an excellent driver routine for solving all the other problems of improper integrals in our first list (except the intractable fifth), as we shall now see.

The basic trick for improper integrals is to make a change of variables to eliminate the singularity, or to map an infinite range of integration to a finite one. For example, the identity

$$\int_a^b f(x)dx = \int_{1/b}^{1/a} \frac{1}{t^2} f\left(\frac{1}{t}\right) dt \qquad ab > 0 \tag{4.4.2}$$

can be used with *either* $b \to \infty$ and a positive, *or* with $a \to -\infty$ and b negative, and works for any function which decreases towards infinity at least as fast as $1/x^2$.

You can make the change of variable implied by (4.4.2) either analytically and then use (e.g.) QROMO and MIDPNT to do the numerical evaluation, *or* you can let the numerical algorithm make the change of variable for you. We prefer the latter method as being more transparent to the user. To implement equation (4.4.2) we simply write a modified version of MIDPNT, called MIDINF, which allows b to be infinite (or, more precisely, a very large number on your particular machine, such as 1×10^{30}), or a to be negative infinite.

```
      SUBROUTINE MIDINF(FUNK,AA,BB,S,N)
        This routine is an exact replacement for MIDPNT, i.e. returns as S the N^th stage of
        refinement of the integral of FUNK from AA to BB, except that the function is evaluated at
        evenly spaced points in 1/x rather than in x. This allows the upper limit BB to be as large
        and positive as the computer allows, or the lower limit AA to be as large and negative,
        but not both. AA and BB must have the same sign.
      FUNC(X)=FUNK(1./X)/X**2        This is a statement function which effects the change of variable.
      B=1./AA                              These two statements change the limits of integration ac-
      A=1./BB                              cordingly.
      IF (N.EQ.1) THEN               From this point on, the routine is exactly identical to MIDPNT.
        S=(B-A)*FUNC(0.5*(A+B))
        IT=1
      ELSE
        TNM=IT
        DEL=(B-A)/(3.*TNM)
        DDEL=DEL+DEL
        X=A+0.5*DEL
        SUM=0.
        DO 11 J=1,IT
          SUM=SUM+FUNC(X)
          X=X+DDEL
          SUM=SUM+FUNC(X)
          X=X+DEL
        11 CONTINUE
        S=(S+(B-A)*SUM/TNM)/3.
        IT=3*IT
      ENDIF
      RETURN
      END
```


If you need to integrate from a negative lower limit to positive infinity, you do this by breaking the integral into two pieces at some positive value, for example,

```
CALL QROMO(FUNK,-5.,2.,S1,MIDPNT)
CALL QROMO(FUNK,2.,1.E30,S2,MIDINF)
ANSWER=S1+S2
```

Where should you choose the breakpoint? At a sufficiently large positive value so that the function FUNK is at least beginning to approach its asymptotic decrease to zero value at infinity. The polynomial extrapolation implicit in the second call to QROMO deals with a polynomial in $1/x$, not in x.

To deal with an integral that has an integrable power-law singularity at its lower limit, one also makes a change of variable. If the integrand diverges as $(x-a)^\gamma$, $0 \le \gamma < 1$, near $x = a$, use the identity

$$\int_a^b f(x)dx = \frac{1}{1-\gamma} \int_0^{(b-a)^{1-\gamma}} t^{\frac{\gamma}{1-\gamma}} f(t^{\frac{1}{1-\gamma}} + a)dt \qquad (b > a) \tag{4.4.3}$$

If the singularity is at the upper limit, use the identity

$$\int_a^b f(x)dx = \frac{1}{1-\gamma} \int_0^{(b-a)^{1-\gamma}} t^{\frac{\gamma}{1-\gamma}} f(b - t^{\frac{1}{1-\gamma}})dt \qquad (b > a) \tag{4.4.4}$$

If there is a singularity at both limits, divide the integral at an interior breakpoint as in the example above.

Equations (4.4.3) and (4.4.4) are particularly simple in the case of inverse square-root singularities, a case that occurs frequently in practice:

$$\int_a^b f(x)dx = \int_0^{\sqrt{b-a}} 2t f(a + t^2)dt \qquad (b > a) \tag{4.4.5}$$

for a singularity at a, an

$$\int_a^b f(x)dx = \int_0^{\sqrt{b-a}} 2t f(b - t^2)dt \qquad (b > a) \tag{4.4.6}$$

for a singularity at b. Once again, we can implement these changes of variable transparently to the user by defining substitute routines for MIDPNT which make the change of variable automatically:

```
      SUBROUTINE MIDSQL(FUNK,AA,BB,S,N)
        This routine is an exact replacement for MIDPNT, except that it allows for an inverse
        square-root singularity in the integrand at the lower limit AA.
      FUNC(X)=2.*X*FUNK(AA+X**2)
      B=SQRT(BB-AA)
      A=0.
      IF (N.EQ.1) THEN
        The rest of the routine is exactly like MIDPNT and is omitted.
```

Exactly similarly,

```
      SUBROUTINE MIDSQU(FUNK,AA,BB,S,N)
        This routine is an exact replacement for MIDPNT, except that it allows for an inverse
        square-root singularity in the integrand at the upper limit BB.
      FUNC(X)=2.*X*FUNK(BB-X**2)
      B=SQRT(BB-AA)
      A=0.
      IF (N.EQ.1) THEN
        The rest of the routine is exactly like MIDPNT and is omitted.
```

One last example should suffice to show how these formulas are derived in general. Suppose the upper limit of integration is infinite, and the integrand falls off exponentially. Then we want a change of variable that maps $e^{-x}dx$ into $(\pm)dt$ (with the sign chosen to keep the upper limit of the new variable larger than the lower limit). Doing the integration gives by inspection

$$t = e^{-x} \qquad \text{or} \qquad x = -\log t \tag{4.4.7}$$

so that

$$\int_{x=a}^{x=\infty} f(x)dx = \int_{t=0}^{t=e^{-a}} f(-\log t)\frac{dt}{t} \quad . \tag{4.4.8}$$

The user-transparent implementation would be

```
      SUBROUTINE MIDEXP(FUNK,AA,BB,S,N)
        This routine is an exact replacement for MIDPNT, except that BB is assumed to be infi-
        nite (value passed not actually used). It is assumed that the function FUNK decreases
        exponentially rapidly at infinity.
      FUNC(X)=FUNK(-ALOG(X))/X
      B=EXP(-AA)
      A=0.
      IF (N.EQ.1) THEN
        The rest of the routine is exactly like MIDPNT and is omitted.
```

REFERENCES AND FURTHER READING:

Acton, Forman S. 1970, *Numerical Methods That Work* (New York: Harper and Row), Chapter 4.

Dahlquist, Germund, and Bjorck, Ake. 1974, *Numerical Methods* (Englewood Cliffs, N.J.: Prentice-Hall), §7.4.3, p.294.

Stoer, J., and Bulirsch, R. 1980, *Introduction to Numerical Analysis* (New York: Springer-Verlag), §3.7, p.152.

4.5 Gaussian Quadratures

In the formulas of §4.1, the integral of a function was approximated by the sum of its functional values at a set of equally spaced points, multiplied by certain aptly chosen weighting coefficients. We saw that as we allowed ourselves more freedom in choosing the coefficients, we could achieve integration formulas of higher and higher order. The idea of *Gaussian quadratures* is to give ourselves the freedom to choose not only the weighting coefficients, but also the location of the abscissas at which the function is to be evaluated: they will no longer be equally spaced. Thus, we will have *twice* the number of degrees of freedom at our disposal; it will turn out that we can achieve Gaussian quadrature formulas whose order is, essentially, twice that of the Newton-Cotes formula with the same number of function evaluations.

Does this sound too good to be true? Well, in a sense it is. The catch is a familiar one, which cannot be overemphasized: high order is not the same as high accuracy. High order translates to high accuracy only when the integrand is very smooth, in the sense of being "well-approximated by a polynomial."

There is, however, one additional feature of Gaussian quadrature formulas which adds to their usefulness: We can arrange the choice of weights and abscissas to make the integral exact for a class of integrands "polynomials times some known function $W(x)$" rather than for the usual class of integrands "polynomials." The function $W(x)$ can then be chosen to remove integrable singularities from the desired integral. Given $W(x)$, in other words, and given an integer N, we can find a set of weights w_i and abscissas x_i such that the approximation

$$\int_a^b W(x)f(x)dx \approx \sum_{i=1}^{N} w_i f(x_i) \tag{4.5.1}$$

is exact if $f(x)$ is a polynomial. For example, to do the integral

$$\int_{-1}^{1} \frac{\exp(-\cos^2 x)}{\sqrt{1-x^2}}dx \tag{4.5.2}$$

(not a very natural looking integral, it must be admitted), we might well be interested in a Gaussian quadrature formula based on the choice

$$W(x) = \frac{1}{\sqrt{1-x^2}} \tag{4.5.3}$$

in the interval $(-1, 1)$. (This particular choice is called *Gauss-Chebyshev integration*, for reasons that will become clear shortly.)

Notice that the integration formula (4.5.1) can also be written with the weight function $W(x)$ not overtly visible: Define $g(x) \equiv W(x)f(x)$ and $v_i \equiv w_i/W(x_i)$. Then (4.5.1) becomes

$$\int_a^b g(x)dx \approx \sum_{i=1}^{N} v_i g(x_i) \tag{4.5.4}$$

Where did the function $W(x)$ go? It is lurking there, ready to give high-order accuracy to integrands of the form polynomials times $W(x)$, and ready to *deny* high-order accuracy to integrands that are otherwise perfectly smooth and well-behaved. When you find tabulations of the weights and abscissas for a given $W(x)$, you have to determine carefully whether they are to be used with a formula in the form of (4.5.1), or like (4.5.4).

Here is an example of a quadrature routine which contains the tabulated abscissas and weights for the case $W(x) = 1$ and $N = 10$. Since the weights and abscissas are, in this case, symmetric around the midpoint of the range of integration, there are actually only five distinct values of each:

```
SUBROUTINE QGAUS(FUNC,A,B,SS)
    Returns as SS the integral of the function FUNC between A and B, by ten-point Gauss-
    Legendre integration: the function is evaluated exactly ten times at interior points in the
    range of integration.
DIMENSION X(5),W(5)              The abscissas and weights.
DATA X/.1488743389,.4333953941,.6794095682,.8650633666,.9739065285/
DATA W/.2955242247,.2692667193,.2190863625,.1494513491,.0666713443/
XM=0.5*(B+A)
XR=0.5*(B-A)
SS=0                             Will be twice the average value of the function, since the ten weights
DO 11 J=1,5                          (five numbers above each used twice) sum to 2.
    DX=XR*X(J)
    SS=SS+W(J)*(FUNC(XM+DX)+FUNC(XM-DX))
 11 CONTINUE
SS=XR*SS                         Scale the answer to the range of integration.
RETURN
END
```

The above routine illustrates that one can use Gaussian quadratures without necessarily understanding the theory behind them: one just locates tabulated weights and abscissas in a book (e.g. Abramowitz and Stegun). However, the theory is very pretty, and it will come in handy if you ever need to construct your own tabulation of weights and abscissas for an unusual choice of $W(x)$. We will therefore give, without any proofs, some useful results that will enable you to do this.

Finally, we will include a routine for computing the weights and abscissas for the most common and useful case, namely the simple choice $W(x) \equiv 1$. This special case is most accurately termed *Gauss-Legendre integration*, but it is often (confusingly) called simply *Gaussian integration*. These weights and abscissas were, e.g. used above in QGAUS.

The theory behind Gaussian quadratures is closely tied to that of orthogonal polynomials. Let us fix the interval of interest to (a, b). We can define the "scalar product of two functions f and g over a weight function W" as

$$\langle f|g\rangle \equiv \int_a^b W(x)f(x)g(x)dx \tag{4.5.5}$$

The scalar product is a number, not a function of x. Two functions are said to be *orthogonal* if their scalar product is zero. A function is said to be *normalized* if its scalar product with itself is unity. A set of functions that are all mutually orthogonal and also all individually normalized is called an *orthonormal* set.

We can find a set of polynomials (i) that includes exactly one polynomial of order j, called $p_j(x)$, for each $j = 0, 1, 2, \ldots$, and (ii) all of which are mutually orthogonal over the specified weight function $W(x)$. A constructive procedure for finding such a set is the recurrence relation

$$\begin{aligned} p_0(x) &\equiv 1 \\ p_{i+1}(x) &= \left[x - \frac{\langle xp_i|p_i\rangle}{\langle p_i|p_i\rangle}\right] p_i(x) - \left[\frac{\langle p_i|p_i\rangle}{\langle p_{i-1}|p_{i-1}\rangle}\right] p_{i-1}(x) \end{aligned} \tag{4.5.6}$$

plus the special rule that the second term of (4.5.6) be omitted for the case of $i = 0$ (there is no p_{-1}).

The polynomials defined by (4.5.6) are *monic*, i.e. the coefficient of their leading term [x^j for $p_j(x)$] is unity. If we divide each $p_j(x)$ by the constant $[\langle p_j|p_j\rangle]^{1/2}$ we can render the set of polynomials orthonormal. One also encounters orthogonal polynomials with various other normalizations.

The polynomial $p_j(x)$ can be shown to have exactly j distinct roots in the interval (a, b). Moreover, it can be shown that the roots of $p_j(x)$ "interleave" the $j - 1$ roots of $p_{j-1}(x)$, i.e. there is exactly one root of the former in between each two adjacent roots of the latter. This fact comes in handy if you need to find all the roots: you can start with the one root of $p_1(x)$ and then, in turn, bracket the roots of each higher j, pinning them down at each stage more precisely by Newton's rule or some other root-finding scheme (see Chapter 9).

Why would you ever want to find all the roots of an orthogonal polynomial $p_j(x)$? Because the abscissas of the N-point Gaussian quadrature formulas (4.5.1) and (4.5.4) with weighting function $W(x)$ in the interval (a, b) are precisely the roots of the orthogonal polynomial $p_N(x)$ for the same interval and weighting function. This is the fundamental theorem of Gaussian quadratures, and lets you find the abscissas for any particular case.

Once you know the abscissas $x_1 \dots x_N$, you need to find the weights w_i, $i = 1 \dots N$. One way to do this is to solve the set of linear equations

$$\begin{bmatrix} p_0(x_1) & \dots & p_0(x_N) \\ p_1(x_1) & \dots & p_1(x_N) \\ \vdots & & \vdots \\ p_{N-1}(x_1) & \dots & p_{N-1}(x_N) \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \end{bmatrix} = \begin{bmatrix} \int_a^b W(x)p_0(x)dx \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{4.5.7}$$

Equation (4.5.7) simply solves for those weights such that the quadrature (4.5.1) gives the correct answer for the integral of the first $N-1$ orthogonal polynomials. It can be shown that, with those weights, the integral of the *next* N polynomials is also exact, so that the quadrature is exact for all polynomials of degree $2N-1$ or less.

There is another, trickier, way to find the weights w_i, based on some deeper theorems. Let us define a new sequence of polynomials $\phi_j(x)$ by the following recurrence:

$$\begin{aligned} \phi_0(x) &\equiv 0 \\ \phi_1(x) &\equiv p_1' \int_a^b W(x)dx \\ \phi_{i+1} &= \left[x - \frac{\langle xp_i|p_i\rangle}{\langle p_i|p_i\rangle}\right]\phi_i(x) - \left[\frac{\langle p_i|p_i\rangle}{\langle p_{i-1}|p_{i-1}\rangle}\right]\phi_{i-1}(x) \end{aligned} \tag{4.5.8}$$

Here p_1' is the derivative of $p_1(x)$ with respect to x, a constant since p_1 is linear. Notice that (4.5.6) says that the ϕ_j's obey *exactly the same recurrence relation* as the p_j's, but with different starting values. Also notice that ϕ_j is of degree one lower than the corresponding p_j. If you know the coefficients of the recurrence relation that generates the p_j's (that is, you know the various scalar products of p_i's that appear in [4.5.6]), then you can easily evaluate $\phi_j(x)$ from (4.5.8) for any desired x and j, by recurrence. Now the wonderful result for the weights of the N-point Gaussian quadrature is

$$w_i = \frac{\phi_N(x_i)}{p_N'(x_i)} \qquad i = 1, \dots, N \tag{4.5.9}$$

Since any constant factor common to both p_N and ϕ_N will cancel in (4.5.9), that equation also holds when the recurrence for orthonormal polynomials, or any other normalization, is used instead of the recurrence for the corresponding monic polynomials, (4.5.6) or (4.5.8).

Here are some of the intervals, weighting functions, and recurrence relations which generate commonly used orthogonal polynomials and their corresponding Gaussian quadrature formulas:

(a,b)	$W(x)$	Recurrence	Gauss–
$(-1,1)$	1	$(i+1)P_{i+1} = (2i+1)xP_i - iP_{i-1}$	Legendre
$(-1,1)$	$(1-x^2)^{-1/2}$	$T_{i+1} = 2xT_i - T_{i-1}$	Chebyshev
$(0,\infty)$	$x^c e^{-x}$	$(i+1)L_{i+1}^c = (-x+2i+c+1)L_i^c - (i+c)L_{i-1}^c$	Laguerre $c = 0, 1, \dots$
$(-\infty,\infty)$	e^{-x^2}	$H_{i+1} = 2xH_i - iH_{i-1}$	Hermite

Tabulations of the abscissas and weights for these, and other, possibilities can be found in standard references.

Here is an example of a routine for calculating one set of abscissas and weights, those of Gauss-Legendre. The routine, due to G. Rybicki, has three departures from the general methods for arbitrary weight functions that we have just discussed: (i) Instead of bracketing the roots by their "interleaving" property, it uses a clever approximation to jump directly to the neighborhood of the desired root, where it converges by Newton's method (to be discussed in §9.4). (ii) Instead of using equation (4.5.9) to find the weights, it uses a special formula that holds for the Gauss-Legendre case,

$$w_i = \frac{2}{(1-x_i^2)[P_N'(x_i)]^2} \tag{4.5.10}$$

(iii) The routine scales the range of integration from (x_1, x_2) to $(-1, 1)$, and provides abscissas x_i and weights w_i for the Gaussian formula

$$\int_{x_1}^{x_2} f(x)dx = \sum_{i=1}^{N} w_i f(x_i) \tag{4.5.11}$$

```
      SUBROUTINE GAULEG(X1,X2,X,W,N)
        Given the lower and upper limits of integration X1 and X2, and given N, this routine returns
        arrays X and W of length N, containing the abscissas and weights of the Gauss-Legendre
        N-point quadrature formula.
      IMPLICIT REAL*8 (A-H,O-Z)      High precision is a good idea for this routine.
      REAL*4 X1,X2,X(N),W(N)
      PARAMETER (EPS=3.D-14)         Increase if you don't have this floating precision.
      M=(N+1)/2                      The roots are symmetric in the interval, so we only have to find half
      XM=0.5D0*(X2+X1)                     of them.
      XL=0.5D0*(X2-X1)
      DO 12 I=1,M                    Loop over the desired roots.
        Z=COS(3.141592654D0*(I-.25D0)/(N+.5D0))
        Starting with the above approximation to the Ith root, we enter the main loop of refinement
        by Newton's method.
1       CONTINUE
          P1=1.D0
          P2=0.D0
          DO 11 J=1,N                Loop up the recurrence relation to get the Legendre polynomial eval-
```

```
            P3=P2
            P2=P1
            P1=((2.D0*J-1.D0)*Z*P2-(J-1.D0)*P3)/J
          11 CONTINUE
      P1 is now the desired Legendre polynomial. We next compute PP, its derivative, by a
      standard relation involving also P2, the polynomial of one lower order.
            PP=N*(Z*P1-P2)/(Z*Z-1.D0)
            Z1=Z
            Z=Z1-P1/PP                  Newton's method.
          IF(ABS(Z-Z1).GT.EPS)GO TO 1
          X(I)=XM-XL*Z                  Scale the root to the desired interval,
          X(N+1-I)=XM+XL*Z              and put in its symmetric counterpart.
          W(I)=2.D0*XL/((1.D0-Z*Z)*PP*PP)     Compute the weight
          W(N+1-I)=W(I)                 and its symmetric counterpart.
        12 CONTINUE
      RETURN
      END
```


REFERENCES AND FURTHER READING:

Abramowitz, Milton, and Stegun, Irene A. 1964, *Handbook of Mathematical Functions*, Applied Mathematics Series, vol. 55 (Washington: National Bureau of Standards; reprinted 1968 by Dover Publications, New York), §25.4.

Stoer, J., and Bulirsch, R. 1980, *Introduction to Numerical Analysis* (New York: Springer-Verlag), §3.6.

Johnson, Lee W., and Riess, R. Dean. 1982, *Numerical Analysis*, 2nd ed. (Reading, Mass.: Addison-Wesley), §6.5.

Carnahan, Brice, Luther, H.A., and Wilkes, James O. 1969, *Applied Numerical Methods* (New York: Wiley), §§2.9-2.10.

Ralston, Anthony, and Rabinowitz, Philip. 1978, *A First Course in Numerical Analysis*, 2nd ed. (New York: McGraw-Hill), §§4.4-4.8.

4.6 Multidimensional Integrals

Integrals of functions of several variables, over regions with dimension greater than one, are *not easy*. There are two reasons for this. First, the number of function evaluations needed to sample an N-dimensional space increases as the N^{th} power of the number needed to do a one-dimensional integral. If you need 30 function evaluations to do a one-dimensional integral crudely, then you will likely need on the order of 30000 evaluations to reach the same crude level for a 3-dimensional integral. Second, the region of integration in N-dimensional space is defined by an $N-1$ dimensional boundary which can itself be terribly complicated: it need not be convex or simply connected, for example. By contrast, the boundary of a one-dimensional integral consists of two numbers, its upper and lower limits.

The first question to be asked, when faced with a multidimensional integral, is, "can it be reduced analytically to a lower dimensionality?" For

example, so-called *iterated integrals* of a function of one variable $f(t)$ can be reduced to one-dimensional integrals by the formula

$$\int_0^x dt_n \int_0^{t_n} dt_{n-1} \cdots \int_0^{t_3} dt_2 \int_0^{t_2} f(t_1)dt_1 = \frac{1}{(n-1)!}\int_0^x (x-t)^{n-1} f(t)dt \tag{4.6.1}$$

Alternatively, the function may have some special symmetry in the way it depends on its independent variables. If the boundary also has this symmetry, then the dimension can be reduced. In three dimensions, for example, the integration of a spherically-symmetric function over a spherical region reduces, in polar coordinates, to a one-dimensional integral.

The next questions to be asked will guide your choice between two entirely different approaches to doing the problem. The questions are: Is the shape of the boundary of the region of integration simple or complicated? Inside the region, is the integrand smooth and simple, or complicated, or locally strongly peaked? Does the problem require high accuracy, or does it require an answer accurate only to a percent, or a few percent?

If your answers are that the boundary is complicated, the integrand is *not* strongly peaked in very small regions, and relatively low accuracy is tolerable, then your problem is a good candidate for *Monte Carlo integration*. This method is also very straightforward to program, especially in its cruder forms. One needs only to know a region with simple boundaries that *includes* the complicated region of integration, plus a method of determining whether a random point is inside or outside the region of integration. Monte Carlo integration evaluates the function at a random sample of points, and estimates its integral based on that random sample. We will discuss it in more detail, and with more sophistication, in Chapter 7.

If the boundary is simple, and the function is very smooth, then the remaining approach, breaking up the problem into repeated one-dimensional integrals, will be effective and relatively fast. If you require high accuracy, this approach is in any case the *only* one available to you, since Monte Carlo methods are by nature asymptotically slow to converge.

For low accuracy, use repeated one-dimensional integration when the integrand is slowly varying and smooth in the region of integration, Monte Carlo when the integrand is oscillatory or discontinuous, but not strongly peaked in small regions.

If the integrand *is* strongly peaked in small regions, and you know where those regions are, break the integral up into several regions so that the integrand is smooth in each, and do each separately. If you don't know where the strongly peaked regions are, you might as well (at the level of sophistication of this book) quit: It is hopeless to expect an integration routine to search out unknown pockets of large contribution in a huge N-dimensional space.

If, on the basis of the above guidelines, you decide to pursue the repeated one-dimensional integration approach, here is how it works. For definiteness,

we will consider the case of a three-dimensional integral in x, y, z-space. Two dimensions, or more than three dimensions are entirely analogous.

The first step is to specify the region of integration by (i) its lower and upper limits in x, which we will denote x_1 and x_2; (ii) its lower and upper limits in y at a specified value of x, denoted $y_1(x)$ and $y_2(x)$; and (iii) its lower and upper limits in z at specified x and y, denoted $z_1(x,y)$ and $z_2(x,y)$. In other words, find the numbers x_1 and x_2, and the functions $y_1(x), y_2(x), z_1(x,y)$, and $z_2(x,y)$ such that

$$\begin{aligned} I &\equiv \int\int\int dx dy dz f(x,y,z) \\ &= \int_{x_1}^{x_2} dx \int_{y_1(x)}^{y_2(x)} dy \int_{z_1(x,y)}^{z_2(x,y)} dz\, f(x,y,z) \end{aligned} \tag{4.6.2}$$

For example, a two-dimensional integral over a circle of radius one centered on the origin becomes

$$\int_{-1}^{1} dx \int_{-\sqrt{1-x^2}}^{\sqrt{1-x^2}} dy\, f(x,y) \tag{4.6.3}$$

Now we can define a function $G(x,y)$ that does the innermost integral,

$$G(x,y) \equiv \int_{z_1(x,y)}^{z_2(x,y)} f(x,y,z) dz \tag{4.6.4}$$

and a function $H(x)$ that does the integral of $G(x,y)$,

$$H(x) \equiv \int_{y_1(x)}^{y_2(x)} G(x,y) dy \tag{4.6.5}$$

and finally our answer as an integral over $H(x)$

$$I = \int_{x_1}^{x_2} H(x) dx \tag{4.6.6}$$

To implement equations (4.6.4) – (4.6.6) in a program, one needs three separate copies of a basic one-dimensional integration routine (and of any subroutines called by it), one each for the x, y, and z integrations. If you try to make do with only one copy, then it will call itself recursively, since (e.g.) the function evaluations of H for the x integration will themselves call the integration routine to do the y integration (see Figure 4.6.1). In our example, let us suppose that we plan to use the one-dimensional integrator QGAUS of

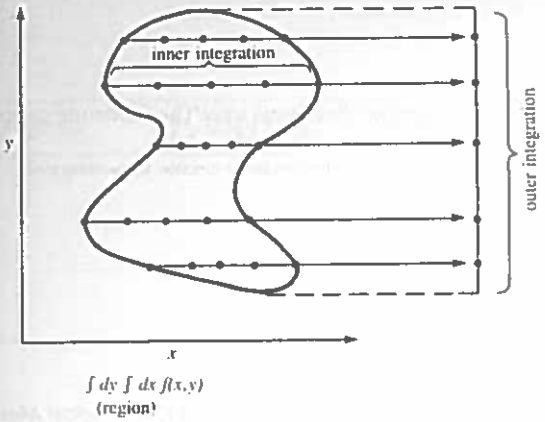

Figure 4.6.1. Function evaluations for a two-dimensional integral over an irregular region, shown schematically. The outer integration routine, in y, requests values of the inner, x, integral at locations along the y axis of its own choosing. The inner integration routine then evaluates the function at x locations suitable to it. This is more accurate in general than, e.g., evaluating the function on a Cartesian mesh of points.

§4.5. Then we make three identical copies and call them QGAUSX, QGAUSY, and QGAUSZ. The basic program for three-dimensional integration then is as follows:

```
SUBROUTINE QUAD3D(X1,X2,SS)
    Returns as SS the integral of a user-supplied function FUNC over a three-dimensional region
    specified by the limits X1, X2, and by the user-supplied functions Y1, Y2, Z1, and Z2,
    as defined in (4.6.2).
EXTERNAL H
CALL QGAUSX(H,X1,X2,SS)
RETURN
END

FUNCTION F(ZZ)                    Called by QGAUSZ. Calls FUNC.
COMMON /XYZ/ X,Y,Z
Z=ZZ
F=FUNC(X,Y,Z)
RETURN
END

FUNCTION G(YY)                    Called by QGAUSY. Calls QGAUSZ.
EXTERNAL F
COMMON /XYZ/ X,Y,Z
Y=YY
CALL QGAUSZ(F,Z1(X,Y),Z2(X,Y),SS)
G=SS
RETURN
END

FUNCTION H(XX)                    Called by QGAUSX. Calls QGAUSY.
EXTERNAL G
COMMON /XYZ/ X,Y,Z
X=XX
CALL QGAUSY(G,Y1(X),Y2(X),SS)
H=SS
```

```
      RETURN
      END
```

The necessary user-supplied functions have the following calling sequences:

```
      FUNCTION FUNC(X,Y,Z)          The 3-dimensional function to be integrated

      FUNCTION Y1(X)
      FUNCTION Y2(X)
      FUNCTION Z1(X,Y)
      FUNCTION Z2(X,Y)
```


REFERENCES AND FURTHER READING:

Dahlquist, Germund, and Bjorck, Ake. 1974, *Numerical Methods* (Englewood Cliffs, N.J.: Prentice-Hall), §7.7, p.318.

Johnson, Lee W., and Riess, R. Dean. 1982, *Numerical Analysis*, 2nd ed. (Reading, Mass.: Addison-Wesley), §6.2.5, p.307.

Abramowitz, Milton, and Stegun, Irene A. 1964, *Handbook of Mathematical Functions*, Applied Mathematics Series, vol. 55 (Washington: National Bureau of Standards; reprinted 1968 by Dover Publications, New York), equations 25.4.58 ff.

Chapter 5. Evaluation of Functions

5.0 Introduction

The purpose of this chapter is to acquaint you with a selection of the techniques which are frequently used in evaluating functions. In Chapter 6, we will apply and illustrate these techniques by giving routines for a variety of specific functions. The purposes of this chapter and the next are thus mostly in harmony, but there is nevertheless some tension between them: Routines that are clearest and most illustrative of the general techniques of this chapter are not always the methods of choice for a particular special function. By comparing this chapter to the next one, you should get some idea of the balance between "general" and "special" methods that occurs in practice.

Insofar as that balance favors general methods, this chapter should give you ideas about how to write your own routines for the evaluation of a function which, while "special" to you, is not so special as to be included in the Chapter 6 or the standard program libraries.

REFERENCES AND FURTHER READING:

Fike, C.T. 1968, *Computer Evaluation of Mathematical Functions* (Englewood Cliffs, N.J.: Prentice-Hall).

Lanczos, Cornelius. 1956, *Applied Analysis* (Englewood Cliffs, N.J.: Prentice-Hall), Chapter 7.