

where  $\epsilon_m$  is the machine precision,  $\epsilon \sim 10^{-7}$  for single precision and  $\epsilon \sim 10^{-15}$  for double precision (the standard for science). Because most scientific computations are done with doubles, we will assume double precision. We want to determine an  $N$  that minimizes the total error, that is, the sum of the approximation and round-off errors:

$$\epsilon_{\text{tot}} \simeq \epsilon_{\text{ro}} + \epsilon_{\text{approx}}. \quad (6.23)$$

This occurs, approximately, when the two errors are of equal magnitude, which we approximate even further by assuming that the two errors are equal:

$$\epsilon_{\text{ro}} = \epsilon_{\text{approx}} = \frac{\mathcal{E}_{\text{trap,simp}}}{f}. \quad (6.24)$$

To continue the search for optimum  $N$  for a general function  $f$ , we set the scale of function size and the lengths by assuming

$$\frac{f^{(n)}}{f} \simeq 1, \quad b - a = 1 \quad \Rightarrow \quad h = \frac{1}{N}. \quad (6.25)$$

The estimate (6.24), when applied to the **trapezoid rule**, yields

$$\sqrt{N}\epsilon_m \simeq \frac{f^{(2)}(b-a)^3}{fN^2} = \frac{1}{N^2}, \quad (6.26)$$

$$\Rightarrow N \simeq \frac{1}{(\epsilon_m)^{2/5}} = \left( \frac{1}{10^{-15}} \right)^{2/5} = 10^6, \quad (6.27)$$

$$\Rightarrow \epsilon_{\text{ro}} \simeq \sqrt{N}\epsilon_m = 10^{-12}. \quad (6.28)$$

The estimate (6.24), when applied to **Simpson's rule**, yields

$$\sqrt{N}\epsilon_m = \frac{f^{(4)}(b-a)^5}{fN^4} = \frac{1}{N^4}, \quad (6.29)$$

$$\Rightarrow N = \frac{1}{(\epsilon_m)^{2/9}} = \left( \frac{1}{10^{-15}} \right)^{2/9} = 2154, \quad (6.30)$$

$$\Rightarrow \epsilon_{\text{ro}} \simeq \sqrt{N}\epsilon_m = 5 \times 10^{-14}. \quad (6.31)$$

These results are illuminating in that they show how

- Simpson's rule is an improvement over the trapezoid rule.
- It is possible to obtain an error close to machine precision with Simpson's rule (and with other higher-order integration algorithms).
- Obtaining the best numerical approximation to an integral is not achieved by letting  $N \rightarrow \infty$  but with a relatively small  $N \leq 1000$ .

## 6.2.4 Algorithm: Gaussian Quadrature

It is often useful to rewrite the basic integration formula (6.3) such that we separate a weighting function  $W(x)$  from the integrand:

$$\int_a^b f(x) dx \equiv \int_a^b W(x)g(x) dx \simeq \sum_{i=1}^N w_i g(x_i). \quad (6.32)$$

In the Gaussian quadrature approach to integration, the  $N$  points and weights are chosen to make the approximation error vanish if  $g(x)$  were a  $(2N - 1)$ -degree polynomial. To obtain this incredible optimization, the points  $x_i$  end up having a specific distribution over  $[a, b]$ . In general, if  $g(x)$  is smooth or can be made smooth by factoring out some  $W(x)$  (Table 6.2.4), Gaussian algorithms will produce higher accuracy than the trapezoid and Simpson rules for the same number of points. Sometimes the integrand may not be smooth because it has different behaviors in different regions. In these cases it makes sense to integrate each region separately and then add the answers together. In fact, some “smart” integration subroutines decide for themselves how many intervals to use and what rule to use in each.

All the rules indicated in Table 6.2.4 are Gaussian with the general form (6.32). We can see that in one case the weighting function is an exponential, in another a Gaussian, and in several an integrable singularity. In contrast to the equally spaced rules, there is never an integration point at the extremes of the intervals, yet the values of the points and weights change as the number of points  $N$  changes. Although we will leave the derivation of the Gaussian points and weights to the references on numerical methods, we note here that for ordinary Gaussian (Gauss–Legendre) integration, the points  $y_i$  turn out to be the  $N$  zeros of the Legendre polynomials, with the weights related to the derivatives,  $P_N(y_i) = 0$ , and  $w_i = 2/[(1 - y_i^2)[P_N'(y_i)]^2]$ . Subroutines to generate these points and weights are standard in mathematical function libraries, are found in tables such as those in [A&S 72], or can be computed. The *gauss* subroutines we provide also scale the points to span specified regions. As a check that your points are correct, you may want to compare them to the four-point set in Table 6.1.

### Mapping Integration Points

Our standard convention (6.3) for the general interval  $[a, b]$  is

$$\int_a^b f(x) dx \simeq \sum_{i=1}^N f(x_i) w_i. \quad (6.33)$$

With Gaussian points and weights, the  $y$  interval  $-1 < y_i \leq 1$  must be *mapped* onto the  $x$  interval  $a \leq x \leq b$ . Here are some mappings we have found useful in our work. In all cases  $(y_i, w_i')$  are the elementary Gaussian points and weights for the interval  $[-1, 1]$ , and we want to scale to  $x$  with various ranges.

1.  $[-1, 1] \rightarrow [a, b]$  uniformly,  $(a + b)/2 = \mathbf{midpoint}$ :

$$x_i = \frac{b + a}{2} + \frac{b - a}{2} y_i, \quad w_i = \frac{b - a}{2} w_i', \quad (6.34)$$

$$\Rightarrow \int_a^b f(x) dx = \frac{b - a}{2} \int_{-1}^1 f[x(y)] dy. \quad (6.35)$$

Types of Gaussian Integration Rules

<i>Integral</i>	<i>Name</i>	<i>Integral</i>	<i>Name</i>
$\int_{-1}^1 f(y) dy$	Gauss	$\int_{-1}^1 \frac{F(y)}{\sqrt{1-y^2}} dy$	Gauss–Chebyshev
$\int_{-\infty}^{\infty} e^{-y^2} F(y) dy$	Gauss–Hermite	$\int_0^{\infty} e^{-y} F(y) dy$	Gauss–Laguerre
$\int_0^{\infty} \frac{e^{-y}}{\sqrt{y}} F(y) dy$	Associated Gauss–Laguerre		

Table 6.1 Points and Weights for four-point Gaussian Quadrature

$\pm y_i$	$w_i$
0.33998 10435 84856	0.65214 51548 62546
0.86113 63115 94053	0.34785 48451 37454

2.  $[0 \rightarrow \infty]$ ,  $a = \text{midpoint}$ :

$$x_i = a \frac{1 + y_i}{1 - y_i}, \quad w_i = \frac{2a}{(1 - y_i)^2} w'_i. \quad (6.36)$$

3.  $[-\infty \rightarrow \infty]$ , **scale set by  $a$** :

$$x_i = a \frac{y_i}{1 - y_i^2}, \quad w_i = \frac{a(1 + y_i^2)}{(1 - y_i^2)^2} w'_i. \quad (6.37)$$

4.  $[a \rightarrow \infty]$ ,  $a + 2b = \text{midpoint}$ :

$$x_i = \frac{a + 2b + ay_i}{1 - y_i}, \quad w_i = \frac{2(b + a)}{(1 - y_i)^2} w'_i. \quad (6.38)$$

5.  $[0 \rightarrow b]$ ,  $ab/(b + a) = \text{midpoint}$ :

$$x_i = \frac{ab(1 + y_i)}{b + a - (b - a)y_i}, \quad w_i = \frac{2ab^2}{(b + a - (b - a)y_i)^2} w'_i. \quad (6.39)$$

As you can see, even if your integration range extends out to infinity, there will be points at large but not infinite  $x$ . As you keep increasing the number of grid points  $N$ , the last  $x_i$  gets larger but always remains finite.

## 6.2.5 Implementation and Error Assessment

- Write a double-precision program to integrate an arbitrary function numerically using the trapezoid rule, the Simpson rule, and Gaussian quadrature. For our assumed **problem** there is an analytic answer:

$$\frac{dN(t)}{dt} = e^{-t} \quad \Rightarrow \quad N(1) = \int_0^1 e^{-t} dt = 1 - e^{-1}.$$

- Compute the relative error  $\epsilon = |(\text{numerical-exact})/\text{exact}|$  in each case. Present your data in the tabular form

$N$	$\epsilon_T$	$\epsilon_S$	$\epsilon_G$
2	...	...	...
10	...	...	...

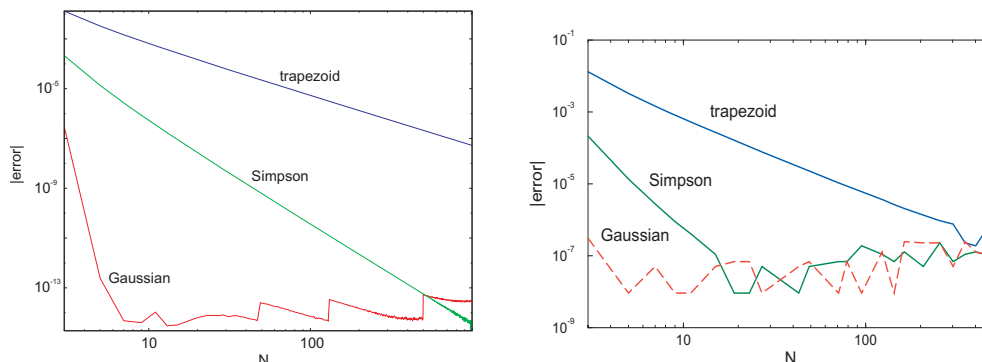
with spaces or tabs separating the fields. Try  $N$  values of 2, 10, 20, 40, 80, 160, .... (*Hint*: Even numbers may not be the assumption of every rule.)

- Make a log-log plot of relative error *versus*  $N$  (Figure 6.3). You should observe that

$$\epsilon \simeq CN^\alpha \quad \Rightarrow \quad \log \epsilon = \alpha \log N + \text{constant}.$$

This means that a power-law dependence appears as a straight line on a log-log plot, and that if you use  $\log_{10}$ , then the ordinate on your log-log plot will be the negative of the number of decimal places of precision in your calculation.

Figure 6.3 Log-log plots of the error in the integration of exponential decay using the trapezoid rule, Simpson's rule, and Gaussian quadrature *versus* the number of integration points  $N$ . Approximately 15 decimal places of precision are attainable with double precision (*left*), and 7 places with single precision (*right*). The algorithms are seen to stop converging when round-off error (the fluctuating and increasing part near the bottom) starts to dominate.



4. Use your plot or table to estimate the power-law dependence of the error  $\epsilon$  on the number of points  $N$  and to determine the number of decimal places of precision in your calculation. Do this for both the trapezoid and Simpson rules and for both the algorithmic and round-off error regimes. (Note that it may be hard to reach the round-off error regime for the trapezoid rule because the approximation error is so large.)

In Listing 6.2 we give a sample program that performs an integration with Gaussian points. The method **gauss** generates the points and weights and may be useful in other applications as well.

Listing 6.2 **IntegGauss.py** integrates the function  $f(x)$  via Gaussian quadrature. The points and weights are generated in the method **gauss**, which remains fixed for all applications. Note that the parameter **eps**, which controls the level of precision desired, should be set by the user, as should the value for **job**, which controls the mapping of the Gaussian points onto arbitrary intervals (they are generated for  $-1 \leq x \leq 1$ ).

```
# IntegGauss.py: Gaussian quadrature generator of pts & wts

from numpy import *
from sys import version
if int(version[0])>2:
    raw_input=input
max.in = 11
vmin = 0.; vmax = 1.
ME = 2.7182818284590452354E0
w = zeros( (2001), float)
x = zeros( (2001), float)

def f(x):
    return (exp( - x ) )

def gauss(npts, job, a, b, x, w):
    m = i = j = t = t1 = pp = p1 = p2 = p3 = 0.
    eps = 3.E-14
    m = (npts + 1)/2
    for i in range(1, m + 1):
        t = cos(math.pi*(float(i) - 0.25)/(float(npts) + 0.5) )
        t1 = 1
        while( (abs(t - t1) ) >= eps):
            p1 = 1.; p2 = 0.
            for j in range(1, npts + 1):
                p3 = p2; p2 = p1
                p1 = ((2.*float(j)-1)*t*p2 - (float(j)-1.)*p3)/(float(j))
            pp = npts*(t*p1 - p2)/(t*t - 1.)
            t1 = t; t = t1 - p1/pp
        x[i - 1] = - t; x[npts - i] = t
        w[i - 1] = 2./( (1. - t*t)*pp*pp)
        w[npts - i] = w[i - 1]
    if (job == 0):
        for i in range(0, npts):
```

```

        x[i] = x[i]*(b - a)/2. + (b + a)/2.
        w[i] = w[i]*(b - a)/2.
    if (job == 1):
        for i in range(0, npts):
            xi = x[i]
            x[i] = a*b*(1. + xi) / (b + a - (b - a)*xi)
            w[i] = w[i]*2.*a*b/( (b + a - (b-a)*xi)*(b + a - (b-a)*xi))
    if (job == 2):
        for i in range(0, npts):
            xi = x[i]
            x[i] = (b*xi + b + a + a) / (1. - xi)
            w[i] = w[i]*2.*(a + b)/( (1. - xi)*(1. - xi) )

def gaussint (no, min, max):
    quadra = 0.
    gauss (no, 0, min, max, x, w)          # Returns pts & wts
    for n in xrange(0, no):
        quadra += f(x[n]) * w[n]           # Calculate integral
    return (quadra)

for i in range(3, max_in + 1, 2):
    result = gaussint(i, vmin, vmax)
    print (" i ", i, " err ", abs(result - 1 + 1/ME))
print ("Enter and return any character to quit")
s = raw_input()

```

### 6.3 EXPERIMENTATION

Try two integrals for which the answers are less obvious:

$$F_1 = \int_0^{2\pi} \sin(100x) dx, \quad F_2 = \int_0^{2\pi} \sin^x(100x) dx. \quad (6.40)$$

Explain why the computer may have trouble with these integrals. |

### 6.4 HIGHER-ORDER RULES (ALGORITHM)

As in numerical differentiation, we can use the known functional dependence of the error on interval size  $h$  to reduce the integration error. For simple rules like the trapezoid and Simpson rules, we have the analytic estimates (6.24), while for others you may have to experiment to determine the  $h$  dependence. To illustrate, if  $A(h)$  and  $A(h/2)$  are the values of the integral determined for intervals  $h$  and  $h/2$ , respectively, and we know that the integrals have expansions with a leading error term proportional to  $h^2$ ,

$$A(h) \simeq \int_a^b f(x) dx + \alpha h^2 + \beta h^4 + \dots, \quad (6.41)$$

$$A\left(\frac{h}{2}\right) \simeq \int_a^b f(x) dx + \frac{\alpha h^2}{4} + \frac{\beta h^4}{16} + \dots. \quad (6.42)$$

Consequently, we make the  $h^2$  term vanish by computing the combination

$$\frac{4}{3}A\left(\frac{h}{2}\right) - \frac{1}{3}A(h) \simeq \int_a^b f(x) dx - \frac{\beta h^4}{4} + \dots. \quad (6.43)$$

Clearly this particular trick (Romberg's extrapolation) works only if the  $h^2$  term dominates the error and then only if the derivatives of the function are well behaved. An analogous extrapolation can also be made for other algorithms.

In Table 6.2.1 we gave the weights for several equal-interval rules. Whereas the Simpson rule used two intervals, the three-eighths rule uses three, and the Milne<sup>2</sup> rule four. (These

<sup>2</sup>There is, not coincidentally, a Milne Computer Center at Oregon State University, although there no longer is a central


are single-interval rules and must be strung together to obtain a rule *extended* over the entire integration range. This means that the points that end one interval and begin the next are weighted twice.) You can easily determine the number of elementary intervals integrated over, and check whether you and we have written the weights right, by summing the weights for any rule. The sum is the integral of  $f(x) = 1$  and must equal  $h$  times the number of intervals (which in turn equals  $b - a$ ):

$$\sum_{i=1}^N w_i = h \times N_{\text{intervals}} = b - a. \quad (6.44)$$

## 6.5 PROBLEM: MONTE CARLO INTEGRATION BY STONE THROWING

Imagine yourself as a farmer walking to your furthestmost field to add algae-eating fish to a pond having an algae explosion. You get there only to read the instructions and discover that you need to know the area of the pond in order to determine the correct number of the fish to add. Your **problem** is to measure the area of this irregularly shaped pond with just the materials at hand [G,T&C 06].

It is hard to believe that Monte Carlo techniques can be used to evaluate integrals. After all, we do not want to gamble on the values! While it is true that other methods are preferable for single and double integrals, Monte Carlo techniques are best when the dimensionality of integrations gets large! For our pond problem, we will use a *sampling* technique (Figure 6.4):

1. Walk off a box that completely encloses the pond and remove any pebbles lying on the ground within the box.
2. Measure the lengths of the sides in natural units like *feet*. This tells you the area of the enclosing box  $A_{\text{box}}$ .
3. Grab a bunch of pebbles, count their number, and then throw them up in the air in random directions.
4. Count the number of splashes in the pond  $N_{\text{pond}}$  and the number of pebbles lying on the ground within your box  $N_{\text{box}}$ .
5.  Assuming that you threw the pebbles uniformly and randomly, the number of pebbles falling into the pond should be proportional to the area of the pond  $A_{\text{pond}}$ . You determine that area from the simple ratio

$$\frac{N_{\text{pond}}}{N_{\text{pond}} + N_{\text{box}}} = \frac{A_{\text{pond}}}{A_{\text{box}}} \quad \Rightarrow \quad A_{\text{pond}} = \frac{N_{\text{pond}}}{N_{\text{pond}} + N_{\text{box}}} A_{\text{box}}. \quad (6.45)$$

### 6.5.1 Stone Throwing Implementation

Use sampling (Figure 6.4) to perform a 2-D integration and thereby determine  $\pi$ :

1. Imagine a circular pond enclosed in a square of side 2 ( $r = 1$ ).
2. We know the analytic area of a circle  $\oint dA = \pi$ .
3. Generate a sequence of random numbers  $-1 \leq r_i \leq +1$ .
4. For  $i = 1$  to  $N$ , pick  $(x_i, y_i) = (r_{2i-1}, r_{2i})$ .
5. If  $x_i^2 + y_i^2 < 1$ , let  $N_{\text{pond}} = N_{\text{pond}} + 1$ ; otherwise let  $N_{\text{box}} = N_{\text{box}} + 1$ .
6. Use (6.45) to calculate the area, and in this way  $\pi$ .

Figure 6.4 Throwing stones into a pond as a technique for measuring its area. The ratio of “hits” to total number of stones thrown equals the ratio of the area of the pond to that of the box.

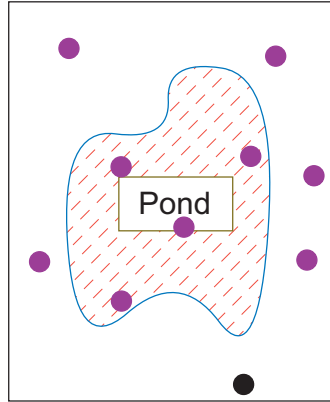
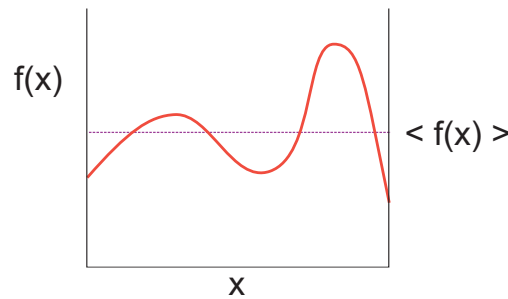


Figure 6.5 The area under the curve  $f(x)$  is the same as that under the horizontal line whose height  $y = \langle f \rangle$ .



7. Increase  $N$  until you get  $\pi$  to three significant figures (we don't ask much — that's only slide-rule accuracy).

### 6.5.2 Integration by Mean Value (Math)

The standard Monte Carlo technique for integration is based on the *mean value theorem* (presumably familiar from elementary calculus):

$$I = \int_a^b dx f(x) = (b - a) \langle f \rangle. \quad (6.46)$$

The theorem states the obvious if you think of integrals as areas: The value of the integral of some function  $f(x)$  between  $a$  and  $b$  equals the length of the interval  $(b - a)$  times the mean value of the function over that interval  $\langle f \rangle$  (Figure 6.5). The integration algorithm uses Monte Carlo techniques to evaluate the mean in (6.46). With a sequence  $a \leq x_i \leq b$  of  $N$  uniform random numbers, we want to determine the *sample mean* by *sampling* the function  $f(x)$  at these points:

$$\langle f \rangle \simeq \frac{1}{N} \sum_{i=1}^N f(x_i). \quad (6.47)$$

This gives us the very simple integration rule:

$$\int_a^b dx f(x) \simeq (b - a) \frac{1}{N} \sum_{i=1}^N f(x_i) = (b - a) \langle f \rangle. \quad (6.48)$$

Equation (6.48) looks much like our standard algorithm for integration (6.3) with the “points”  $x_i$  chosen randomly and with uniform weights  $w_i = (b - a)/N$ . Because no attempt has been made to obtain the best answer for a given value of  $N$ , this is by no means an optimized way to evaluate integrals; but you will admit it is simple. If we let the number of samples of  $f(x)$  approach infinity  $N \rightarrow \infty$  or if we keep the number of samples finite and take the average of infinitely many runs, the laws of statistics assure us that (6.48) will approach the correct answer, at least if there were no round-off errors.

For readers who are familiar with statistics, we remind you that the uncertainty in the value obtained for the integral  $I$  after  $N$  samples of  $f(x)$  is measured by the standard deviation  $\sigma_I$ . If  $\sigma_f$  is the standard deviation of the integrand  $f$  in the sampling, then for normal distributions we have

$$\sigma_I \simeq \frac{1}{\sqrt{N}} \sigma_f. \quad (6.49)$$

So for large  $N$ , the error in the value obtained for the integral decreases as  $1/\sqrt{N}$ .

## 6.6 HIGH-DIMENSIONAL INTEGRATION (PROBLEM)

Let’s say that we want to calculate some properties of a small atom such as magnesium with 12 electrons. To do that we need to integrate atomic wave functions over the three coordinates of each of 12 electrons. This amounts to a  $3 \times 12 = 36$ -D integral. If we use 64 points for each integration, this requires about  $64^{36} \simeq 10^{65}$  evaluations of the integrand. If the computer were fast and could evaluate the integrand a million times per second, this would take about  $10^{59}$  s, which is significantly longer than the age of the universe ( $\sim 10^{17}$  s).

Your **problem** is to find a way to perform multidimensional integrations so that you are still alive to savor the answers. Specifically, evaluate the 10-D integral

$$I = \int_0^1 dx_1 \int_0^1 dx_2 \cdots \int_0^1 dx_{10} (x_1 + x_2 + \cdots + x_{10})^2. \quad (6.50)$$

Check your numerical answer against the analytic one,  $\frac{155}{6}$ .

### 6.6.1 Multidimensional Monte Carlo

It is easy to generalize mean value integration to many dimensions by picking random points in a multidimensional space. For example,

$$\int_a^b dx \int_c^d dy f(x, y) \simeq (b - a)(d - c) \frac{1}{N} \sum_i^N f(\mathbf{x}_i) = (b - a)(d - c) \langle f \rangle. \quad (6.51)$$

### 6.6.2 Error in Multidimensional Integration (Assessment)

When we perform a multidimensional integration, the error in the Monte Carlo technique, being statistical, decreases as  $1/\sqrt{N}$ . This is valid even if the  $N$  points are distributed over  $D$  dimensions. In contrast, when we use these same  $N$  points to perform a  $D$ -dimensional integration as  $D$  1-D integrals using a rule such as Simpson’s, we use  $N/D$  points for each integration. For fixed  $N$ , this means that the number of points used for each integration decreases as the number of dimensions  $D$  increases, and so the error in each integration increases with