```
    psdes(&lword,&irword);                    "Pseudo-DES" encode the words.
    itemp=jflone | (jflmsk & irword);         Mask to a floating number between 1 and
    ++(*idum);                                    2.
    return (*(float *)&itemp)-1.0;            Subtraction moves range to 0. to 1.
}
```

The accompanying table gives data for verifying that `ran4` and `psdes` work correctly on your machine. We do not advise the use of `ran4` unless you are able to reproduce the hex values shown. Typically, `ran4` is about 4 times slower than `ran0` (§7.1), or about 3 times slower than `ran1`.

Values for Verifying the Implementation of `psdes`

`idum`	before `psdes` call		after `psdes` call (hex)		`ran4(idum)`	
	`lword`	`irword`	`lword`	`irword`	VAX	PC
–1	1	1	604D1DCE	509C0C23	0.275898	0.219120
99	1	99	D97F8571	A66CB41A	0.208204	0.849246
–99	99	1	7822309D	64300984	0.034307	0.375290
99	99	99	D7F376F0	59BA89EB	0.838676	0.457334

Successive calls to `psdes` with arguments -1, 99, -99, and 1, should produce exactly the `lword` and `irword` values shown. Masking conversion to a returned floating random value is allowed to be machine dependent; values for VAX and PC are shown.

CITED REFERENCES AND FURTHER READING:

Data Encryption Standard, 1977 January 15, Federal Information Processing Standards Publication, number 46 (Washington: U.S. Department of Commerce, National Bureau of Standards). [1]

Guidelines for Implementing and Using the NBS Data Encryption Standard, 1981 April 1, Federal Information Processing Standards Publication, number 74 (Washington: U.S. Department of Commerce, National Bureau of Standards). [2]

Validating the Correctness of Hardware Implementations of the NBS Data Encryption Standard, 1980, NBS Special Publication 500–20 (Washington: U.S. Department of Commerce, National Bureau of Standards). [3]

Meyer, C.H. and Matyas, S.M. 1982, *Cryptography: A New Dimension in Computer Data Security* (New York: Wiley). [4]

Knuth, D.E. 1973, *Sorting and Searching*, vol. 3 of *The Art of Computer Programming* (Reading, MA: Addison-Wesley), Chapter 6. [5]

Vitter, J.S., and Chen, W-C. 1987, *Design and Analysis of Coalesced Hashing* (New York: Oxford University Press). [6]

7.6 Simple Monte Carlo Integration

Inspirations for numerical methods can spring from unlikely sources. "Splines" first were flexible strips of wood used by draftsmen. "Simulated annealing" (we shall see in §10.9) is rooted in a thermodynamic analogy. And who does not feel at least a faint echo of glamor in the name "Monte Carlo method"?

Suppose that we pick N random points, uniformly distributed in a multidimensional volume V. Call them $x_1, \ldots, x_N$. Then the basic theorem of Monte Carlo integration estimates the integral of a function f over the multidimensional volume,

$$\int f\, dV \approx V \langle f \rangle \pm V \sqrt{\frac{\langle f^2 \rangle - \langle f \rangle^2}{N}} \tag{7.6.1}$$

Here the angle brackets denote taking the arithmetic mean over the N sample points,

$$\langle f \rangle \equiv \frac{1}{N} \sum_{i=1}^{N} f(x_i) \qquad \langle f^2 \rangle \equiv \frac{1}{N} \sum_{i=1}^{N} f^2(x_i) \tag{7.6.2}$$

The "plus-or-minus" term in (7.6.1) is a one standard deviation error estimate for the integral, not a rigorous bound; further, there is no guarantee that the error is distributed as a Gaussian, so the error term should be taken only as a rough indication of probable error.

Suppose that you want to integrate a function g over a region W that is not easy to sample randomly. For example, W might have a very complicated shape. No problem. Just find a region V that *includes* W and that *can* easily be sampled (Figure 7.6.1), and then define f to be equal to g for points in W and equal to zero for points outside of W (but still inside the sampled V). You want to try to make V enclose W as closely as possible, because the zero values of f will increase the error estimate term of (7.6.1). And well they should: points chosen outside of W have no information content, so the effective value of N, the number of points, is reduced. The error estimate in (7.6.1) takes this into account.

General purpose routines for Monte Carlo integration are quite complicated (see §7.8), but a worked example will show the underlying simplicity of the method. Suppose that we want to find the weight and the position of the center of mass of an object of complicated shape, namely the intersection of a torus with the edge of a large box. In particular let the object be defined by the three simultaneous conditions

$$z^2 + \left(\sqrt{x^2 + y^2} - 3\right)^2 \le 1 \tag{7.6.3}$$

(torus centered on the origin with major radius $= 4$, minor radius $= 2$)

$$x \ge 1 \qquad y \ge -3 \tag{7.6.4}$$

(two faces of the box, see Figure 7.6.2). Suppose for the moment that the object has a constant density ρ.

We want to estimate the following integrals over the interior of the complicated object:

$$\int \rho\, dx\, dy\, dz \qquad \int x\rho\, dx\, dy\, dz \qquad \int y\rho\, dx\, dy\, dz \qquad \int z\rho\, dx\, dy\, dz \tag{7.6.5}$$

The coordinates of the center of mass will be the ratio of the latter three integrals (linear moments) to the first one (the weight).

In the following fragment, the region V, enclosing the piece-of-torus W, is the rectangular box extending from 1 to 4 in x, -3 to 4 in y, and -1 to 1 in z.

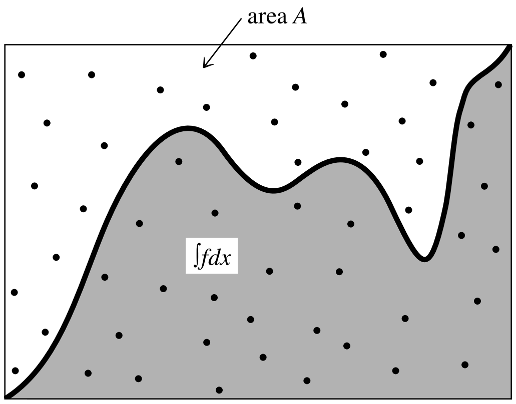

Figure 7.6.1. Monte Carlo integration. Random points are chosen within the area A. The integral of the function f is estimated as the area of A multiplied by the fraction of random points that fall below the curve f. Refinements on this procedure can improve the accuracy of the method; see text.

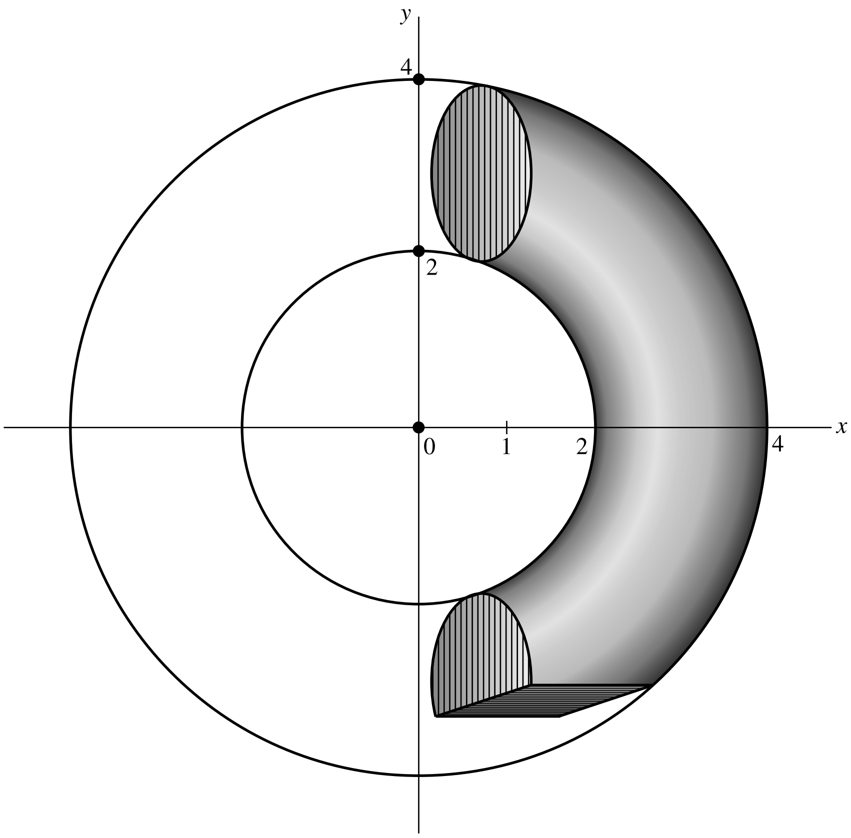

Figure 7.6.2. Example of Monte Carlo integration (see text). The region of interest is a piece of a torus, bounded by the intersection of two planes. The limits of integration of the region cannot easily be written in analytically closed form, so Monte Carlo is a useful technique.

```
#include "nrutil.h"
...
n=...                                     Set to the number of sample points desired.
den=...                                   Set to the constant value of the density.
sw=swx=swy=swz=0.0;                       Zero the various sums to be accumulated.
varw=varx=vary=varz=0.0;
vol=3.0*7.0*2.0;                          Volume of the sampled region.
for(j=1;j<=n;j++) {
    x=1.0+3.0*ran2(&idum);                Pick a point randomly in the sampled re-
    y=(-3.0)+7.0*ran2(&idum);                 gion.
    z=(-1.0)+2.0*ran2(&idum);
    if (z*z+SQR(sqrt(x*x+y*y)-3.0) < 1.0) {      Is it in the torus?
        sw += den;                        If so, add to the various cumulants.
        swx += x*den;
        swy += y*den;
        swz += z*den;
        varw += SQR(den);
        varx += SQR(x*den);
        vary += SQR(y*den);
        varz += SQR(z*den);
    }
}
w=vol*sw/n;                               The values of the integrals (7.6.5),
x=vol*swx/n;
y=vol*swy/n;
z=vol*swz/n;
dw=vol*sqrt((varw/n-SQR(sw/n))/n);        and their corresponding error estimates.
dx=vol*sqrt((varx/n-SQR(swx/n))/n);
dy=vol*sqrt((vary/n-SQR(swy/n))/n);
dz=vol*sqrt((varz/n-SQR(swz/n))/n);
```

A change of variable can often be extremely worthwhile in Monte Carlo integration. Suppose, for example, that we want to evaluate the same integrals, but for a piece-of-torus whose density is a strong function of z, in fact varying according to

$$\rho(x, y, z) = e^{5z} \tag{7.6.6}$$

One way to do this is to put the statement

```
den=exp(5.0*z);
```

inside the `if (...)` block, just before `den` is first used. This will work, but it is a poor way to proceed. Since (7.6.6) falls so rapidly to zero as z decreases (down to its lower limit -1), most sampled points contribute almost nothing to the sum of the weight or moments. These points are effectively wasted, almost as badly as those that fall outside of the region W. A change of variable, exactly as in the transformation methods of §7.2, solves this problem. Let

$$ds = e^{5z}dz \qquad \text{so that} \qquad s = \frac{1}{5}e^{5z}, \quad z = \frac{1}{5}\ln(5s) \tag{7.6.7}$$

Then $\rho dz = ds$, and the limits $-1 < z < 1$ become $.00135 < s < 29.682$. The program fragment now looks like this

```
#include "nrutil.h"
...
n=...                                  Set to the number of sample points desired.
sw=swx=swy=swz=0.0;
varw=varx=vary=varz=0.0;
ss=0.2*(exp(5.0)-exp(-5.0))            Interval of s to be random sampled.
vol=3.0*7.0*ss                         Volume in x,y,s-space.
for(j=1;j<=n;j++) {
    x=1.0+3.0*ran2(&idum);
    y=(-3.0)+7.0*ran2(&idum);
    s=0.00135+ss*ran2(&idum);          Pick a point in s.
    z=0.2*log(5.0*s);                  Equation (7.6.7).
    if (z*z+SQR(sqrt(x*x+y*y)-3.0) < 1.0) {
        sw += 1.0;                     Density is 1, since absorbed into definition
        swx += x;                          of s.
        swy += y;
        swz += z;
        varw += 1.0;
        varx += x*x;
        vary += y*y;
        varz += z*z;
    }
}
w=vol*sw/n;                            The values of the integrals (7.6.5),
x=vol*swx/n;
y=vol*swy/n;
z=vol*swz/n;
dw=vol*sqrt((varw/n-SQR(sw/n))/n);     and their corresponding error estimates.
dx=vol*sqrt((varx/n-SQR(swx/n))/n);
dy=vol*sqrt((vary/n-SQR(swy/n))/n);
dz=vol*sqrt((varz/n-SQR(swz/n))/n);
```

If you think for a minute, you will realize that equation (7.6.7) was useful only because the part of the integrand that we wanted to eliminate (e^{5z}) was both integrable analytically, and had an integral that could be analytically inverted. (Compare §7.2.) In general these properties will not hold. Question: What then? Answer: Pull out of the integrand the "best" factor that *can* be integrated and inverted. The criterion for "best" is to try to reduce the remaining integrand to a function that is as close as possible to constant.

The limiting case is instructive: If you manage to make the integrand f *exactly* constant, and if the region V, of known volume, *exactly* encloses the desired region W, then the average of f that you compute will be exactly its constant value, and the error estimate in equation (7.6.1) will exactly vanish. You will, in fact, have done the integral exactly, and the Monte Carlo numerical evaluations are superfluous. So, backing off from the extreme limiting case, *to the extent* that you are able to make f approximately constant by change of variable, and *to the extent* that you can sample a region only slightly larger than W, you will increase the accuracy of the Monte Carlo integral. This technique is generically called *reduction of variance* in the literature.

The fundamental disadvantage of simple Monte Carlo integration is that its accuracy increases only as the square root of N, the number of sampled points. If your accuracy requirements are modest, or if your computer budget is large, then the technique is highly recommended as one of great generality. In the next two sections we will see that there are techniques available for "breaking the square root of N barrier" and achieving, at least in some cases, higher accuracy with fewer function evaluations.

CITED REFERENCES AND FURTHER READING:
Hammersley, J.M., and Handscomb, D.C. 1964, *Monte Carlo Methods* (London: Methuen).
Shreider, Yu. A. (ed.) 1966, *The Monte Carlo Method* (Oxford: Pergamon).
Sobol', I.M. 1974, *The Monte Carlo Method* (Chicago: University of Chicago Press).
Kalos, M.H., and Whitlock, P.A. 1986, *Monte Carlo Methods* (New York: Wiley).

7.7 Quasi- (that is, Sub-) Random Sequences

We have just seen that choosing N points uniformly randomly in an n-dimensional space leads to an error term in Monte Carlo integration that decreases as $1/\sqrt{N}$. In essence, each new point sampled adds linearly to an accumulated sum that will become the function average, and also linearly to an accumulated sum of squares that will become the variance (equation 7.6.2). The estimated error comes from the square root of this variance, hence the power $N^{-1/2}$.

Just because this square root convergence is familiar does not, however, mean that it is inevitable. A simple counterexample is to choose sample points that lie on a Cartesian grid, and to sample each grid point exactly once (in whatever order). The Monte Carlo method thus becomes a deterministic quadrature scheme — albeit a simple one — whose fractional error decreases at least as fast as N^{-1} (even faster if the function goes to zero smoothly at the boundaries of the sampled region, or is periodic in the region).

The trouble with a grid is that one has to decide *in advance* how fine it should be. One is then committed to completing all of its sample points. With a grid, it is not convenient to "sample *until*" some convergence or termination criterion is met. One might ask if there is not some intermediate scheme, some way to pick sample points "at random," yet spread out in some self-avoiding way, avoiding the chance clustering that occurs with uniformly random points.

A similar question arises for tasks other than Monte Carlo integration. We might want to search an n-dimensional space for a point where some (locally computable) condition holds. Of course, for the task to be computationally meaningful, there had better be continuity, so that the desired condition will hold in some finite n-dimensional neighborhood. We may not know *a priori* how large that neighborhood is, however. We want to "sample *until*" the desired point is found, moving smoothly to finer scales with increasing samples. Is there any way to do this that is better than uncorrelated, random samples?

The answer to the above question is "yes." Sequences of n-tuples that fill n-space more uniformly than uncorrelated random points are called *quasi-random sequences*. That term is somewhat of a misnomer, since there is nothing "random" about quasi-random sequences: They are cleverly crafted to be, in fact, *sub*-random. The sample points in a quasi-random sequence are, in a precise sense, "maximally avoiding" of each other.

A conceptually simple example is *Halton's sequence* [1]. In one dimension, the jth number H_j in the sequence is obtained by the following steps: (i) Write j as a number in base b, where b is some prime. (For example $j = 17$ in base $b = 3$ is 122.) (ii) Reverse the digits and put a radix point (i.e., a decimal point base b) in