Rep. Prog. Phys., Vol. 43, 1980. Printed in Great Britain

Monte Carlo theory and practice

F JAMES

Data Handling Division, CERN, Geneva, Switzerland

Abstract

The Monte Carlo method has long been recognised as a powerful technique for performing certain calculations, generally those too complicated for a more classical approach. Since the use of high-speed computers became widespread in the 1950s, a great deal of theoretical investigation has been undertaken and practical experience has been gained in the Monte Carlo approach. The aim of this review is, first, to lay a theoretical basis for both the 'traditional' Monte Carlo and quasi-Monte Carlo methods, and, secondly, to present some practical aspects of when and how to use them. An important theme of this review will be the comparison of Monte Carlo, quasi-Monte Carlo and numerical quadrature for the integration of functions, especially in many dimensions.

This review was received in February 1980.

0034–4885/80/091145+45 $06.50

Contents

	Page
1. Introduction and definitions	1147
1.1. Definition	1147
1.2. Simulation	1147
1.3. Integration	1148
2. Mathematical foundation for Monte Carlo integration	1148
2.1. Random variables and distributions	1148
2.2. Independence of random variables	1149
2.3. Expectation, variance, covariance	1149
2.4. The law of large numbers	1150
2.5. Convergence	1151
2.6. The central limit theorem	1151
2.7. Résumé: mathematical properties of the Monte Carlo method	1153
3. From Buffon's needle to variance-reducing techniques	1153
3.1. Buffon's needle: hit-or-miss Monte Carlo	1153
3.2. Integration: crude Monte Carlo	1154
3.3. Classical variance-reducing techniques	1155
3.4. Adaptive variance-reducing techniques	1158
4. Comparison with numerical quadrature	1160
4.1. One-dimensional quadrature	1160
4.2. Multidimensional quadrature	1162
4.3. The Monte Carlo paradox	1166
5. Random and pseudo-random numbers	1167
5.1. Truly random numbers	1167
5.2. Pseudo-random numbers	1169
6. Quasi-Monte Carlo	1175
6.1. The quasi-random philosophy	1175
6.2. The theoretical basis of quasi-Monte Carlo	1176
6.3. Quasi-random number generators	1177
7. Non-uniform random numbers	1179
7.1. Gaussian generators	1180
7.2. All other known distributions	1182
7.3. Empirical distributions	1182
8. Applications	1183
8.1. The uncertainty of a weighted average	1183
8.2. Integration over a triangle	1184
8.3. Programs for real-life calculations	1185
8.4. Splitting and killing in sequential simulations	1186
8.5. Multiparticle phase space	1187
8.6. Sampling from a finite population	1187
References	1188

1. Introduction and definitions

1.1. Definition

A Monte Carlo technique is any technique making use of random numbers to solve a problem. (We assume for the moment that the reader understands what a random number is, although this is by no means a trivial point and will be treated later in some detail.)

The above definition should be supplemented by a somewhat narrower but more enlightening definition as given by Halton (1970): the Monte Carlo method is defined as representing the solution of a problem as a parameter of a hypothetical population, and using a random sequence of numbers to construct a sample of the population, from which statistical estimates of the parameter can be obtained.

Let us express the solution of the problem as a result F, which may be a real number, a set of numbers, a yes/no decision, etc. The Monte Carlo estimate of F will be a function of, among other things, the random numbers used in the calculation. The introduction of randomness into an otherwise well-defined problem produces solutions with rather special properties which, as we shall see, are sometimes surprisingly good.

1.2. Simulation

Historically, the first large-scale calculations to make use of the Monte Carlo method were studies of neutron scattering and absorption, random processes for which it is quite natural to employ random numbers. Such calculations, a subset of Monte Carlo calculations, are known as direct simulation, since the 'hypothetical population' of the narrower definition above corresponds directly to the real population being studied. However, as those involved were well aware, the numerical results obtained were perfectly 'deterministic' and, in principle, obtainable by classical computational techniques (in fact, integration). Whether or not the Monte Carlo method can be applied to a given problem does not depend on the stochastic nature of the system being studied, but only on our ability to formulate the problem in such a way that random numbers may be used to obtain the solution. This can be seen by inverting the neutron scattering problem and considering first the classical solution in terms of a complicated multidimensional integral. The value of this integral is quite non-random, but happens also to be the solution of a problem involving random processes. The Monte Carlo method may be applied wherever it is possible to establish equivalence between the desired result and the expected behaviour of a stochastic system.

The problem to be solved may already be of a probabilistic or statistical nature, in which case its Monte Carlo formulation will usually be a straightforward simulation, or it may be of a deterministic or analytic nature, in which case an appropriate Monte Carlo formulation may require some imagination and may appear contrived or artificial. In any case, the suitability of the method chosen will depend on its mathematical properties and not on its superficial resemblance to the problem to be solved. We shall see how Monte Carlo techniques may be compared with other methods of solution of the same physical problem.

1.3. Integration

At least in a formal sense, all Monte Carlo calculations are equivalent to integrations. This follows from the definition of a Monte Carlo calculation as producing a result F which is a function of random numbers r_i. Let us assume for simplicity the usual case that the r_i are uniformly distributed between zero and one. Then the Monte Carlo result $F = F(r_1, r_2, \ldots, r_n)$ is an unbiased estimator of the multi-dimensional integral

$$I = \int_0^1 \ldots \int_0^1 F(x_1, x_2, \ldots, x_n)\, \mathrm{d}x_1\, \mathrm{d}x_2 \ldots \mathrm{d}x_n$$

or, stated another way, the expectation of F is the integral I. (When the problem to be solved is explicitly the problem of integrating a function f, the F above is not to be identified with f but rather the Monte Carlo estimate of its integral.) This formal equivalence will allow us to lay a firm theoretical justification for Monte Carlo techniques and will also lead us to many results of practical importance.

2. Mathematical foundation for Monte Carlo integration

In this section we will define some basic statistical terms and invoke some of the important results of mathematical statistics to lay a formal foundation for the validity of Monte Carlo calculations. The results of this section will be important to the later sections, so we will try to make it complete, but since many readers will already be familiar with this material, no attempt is made to be mathematically rigorous. Those who wish a more detailed treatment are urged to consult an independent text, such as Eadie *et al* (1971). Those who still remember their elementary statistics are advised to skip directly to §2.6.

2.1. Random variables and distributions

A *random variable* is a variable that can take on more than one value (generally a continuous range of values), and for which any particular value that will be taken cannot be predicted in advance. Even though the value of the variable is unpredictable, the *distribution* of the variable may well be known. The distribution of a random variable gives the probability of a given value (or infinitesimal range of values). Since we will usually be working with continuous variables, we define

$$g(u)\, \mathrm{d}u = P[u < u' < u + \mathrm{d}u].$$

The function $g(u)$ is the *probability density function* of u and gives the probability of finding the random variable u' within $\mathrm{d}u$ of a given value u. This is the most usual way for physicists to express the way u' is distributed, although it is sometimes more convenient mathematically to use the *integrated distribution function* defined as the definite integral of g from minus infinity to u:

$$G(u) = \int_{-\infty}^{u} g(x)\, \mathrm{d}x$$

$$g(u) = \mathrm{d}G(u)/\mathrm{d}u.$$

Note that $G(u)$ is a monotonically non-decreasing function taking on values from zero to one, and that g is always normalised so that its integral over all u is one.

A function of a random variable is, of course, itself a random variable, although

it will in general be distributed differently from its argument. The functions $G(u)$ and $g(u)$ defined above are, however, not to be considered as random variables since they are functions of the variable u rather than the random variable u'.

2.2. Independence of random variables

Let us consider two random variables u' and v'. In order to specify completely the distribution of u' and v', we now require a function of two variables, say $h(u, v)$, and the ensuing mathematics becomes considerably more complicated. However, an important special case is when the function $h(u, v)$ can be factored exactly into a product of two functions, each of which depends only on one variable, $h(u, v) = p(u)q(v)$. In this case we say that u' and v' are stochastically *independent* since the distribution of u' does not depend on the value of v' and vice versa.

When more than two variables are considered, the concept of independence becomes more complicated, and it is no longer sufficient to consider only the dependence of pairs of variables. Indeed, it is possible to have all pairs of variables independent and still have dependence among triplets or higher combinations of variables. For example, let r and s be two independent random variables, each uniformly distributed between zero and one, and consider the three new variables:

$$x = r$$

$$y = s$$

$$z = (r + s) \bmod 1.$$

Now each of the three random variables x, y, z is also uniformly distributed between zero and one, and all pairs (x, y), (y, z) and (x, z) are independent (knowledge about the value of one member of a pair gives no information about the value of the other member). However, the three are clearly dependent, since knowledge of any two determines the third completely.

2.3. Expectation, variance, covariance

The mathematical *expectation of a function* $f(u')$ is defined as the average or mean value of the function

$$E(f) = \int f(u)\, \mathrm{d}G(u) = \int f(u)\, g(u)\, \mathrm{d}u$$

where $G(u)$ is a distribution function giving the distribution of the independent variable u'. Usually the u' will be uniformly distributed between a and b: $\mathrm{d}G = \mathrm{d}u/(b-a)$, so that the expectation becomes

$$E(f) = \frac{1}{b-a} \int_a^b f(u)\, \mathrm{d}u.$$

Similarly the *expectation of a variable* u' is the average value of u:

$$E(u') = \int u\, \mathrm{d}G(u) = \int u\, g(u)\, \mathrm{d}u.$$

The *variance* of a function or variable is the average of the squared deviation from its expectation and is most conveniently defined in terms of the expectation:

$$V(f) = E[f - E(f)]^2 = \int [f - E(f)]^2\, \mathrm{d}G.$$

Note that calculating the expectation requires one integration and the variance involves one more integration.

The square root of the variance is called the *standard deviation.* It is more physically meaningful than the variance since it has the same dimensions as its argument but the square root makes it more clumsy to manipulate mathematically. The standard deviation can most easily be interpreted as the root-mean-square deviation from the mean.

Considering expectation and variance as operators, we may verify some simple rules for applying these operators to linear combinations of variables. Let x and y be random variables and c be a constant. Then

$$E(cx+y)=cE(x)+E(y) \tag{2.1}$$

$$V(cx+y)=c^2V(x)+V(y)+2cE[(y-E(y))(x-E(x))]. \tag{2.2}$$

Expectation is therefore a linear operator, whereas variance is not linear. The last term in the above expression for the variance is called the *covariance between x and y* and is zero if x and y are *independent.* If this term is positive, x and y are said to be positively correlated, and if negative, x and y are negatively correlated. Note that x and y may be uncorrelated (i.e. their covariance may be zero) even if they are not independent, but if they are independent they must also be uncorrelated. Note also that even though the variance operator is not linear, the following relationship holds if x and y are independent variables:

$$V(x+y)=V(x)+V(y) \qquad x, y \text{ uncorrelated.}$$

2.4. The law of large numbers

The law of large numbers concerns the behaviour of sums of large numbers of random variables. Let us choose n numbers u_i randomly with probability density uniform on the interval from a to b, and for each u_i evaluate the function $f(u_i)$. This law says that the sum of these function values, divided by n, will converge to the expectation of the function f. That is, as n becomes very large,

$$\frac{1}{n}\sum_{i=1}^{n} f(u_i) \to \frac{1}{b-a}\int_a^b f(u)\,\mathrm{d}u. \tag{2.3}$$

In statistical language, the left-hand side of (2.3) is a *consistent estimator* of the integral on the right-hand side, since (under certain conditions) it converges to the exact value of the integral as n approaches infinity. The 'certain conditions' involve the behaviour of the function f, since it must of course be integrable, and we will generally require that it be everywhere finite and at least piecewise continuous (it may have a finite number of discontinuities in the interval under consideration).

Since the left-hand side of (2.3) is just the Monte Carlo estimate of the integral on the right-hand side, the law of large numbers can be interpreted as a statement that the Monte Carlo estimate of an integral is, under 'certain conditions', a consistent estimate, i.e. it converges to the correct answer as the random sample size becomes very large.

2.5. Convergence

It is worthwhile discussing at this point the meaning of convergence in the statistical context, since it is considerably more complex than the more familiar convergence of calculus. We recall that in calculus, the sequence $\{A\}$ is said to converge to B if for any arbitrarily small positive quantity δ, an element of $\{A\}$ can be found such that all the succeeding elements of $\{A\}$ are guaranteed to be within δ of B.

In the statistical context, the 'guarantee' must be replaced by a statement of probability, so that the corresponding definition becomes: $A(n)$ is said to converge to B as n goes to infinity if for any probability $P[0<P<1]$, and any positive quantity δ, a k can be found such that for all $n>k$ the probability that $A(n)$ will be within δ of B is greater than P. Note that this is quite weak, in that no matter how big n is, $A(n)$ can never be guaranteed to be within a given distance of B.

This risk, that convergence is only given with a certain probability, is inherent in Monte Carlo calculations and is the reason why this technique was named after the world's most famous gambling casino. Indeed, the name is doubly appropriate because the style of gambling in the Monte Carlo casino, not to be confused with the noisy and tasteless gambling houses of Las Vegas and Reno, is serious and sophisticated. The apparent contradiction between the unpredictability of the gambling process and the seriousness of the results is one of the fascinating aspects of the Monte Carlo method which has been responsible for a great deal of the interest shown in the method but has also resulted in considerable confusion and misunderstanding. This point will come up again, especially in our discussion of random numbers.

2.6. The central limit theorem

Whereas the law of large numbers tells us that the Monte Carlo estimate of an integral is correct for 'infinite' n, the central limit theorem tells us approximately how that estimate is distributed for large but finite n. This very important theorem says essentially that the sum of a large number of independent random variables is always normally distributed (i.e. a Gaussian distribution), no matter how the individual random variables are distributed, provided they have finite expectations and variances and provided n is 'large enough'. How large n has to be depends, of course, on the individual distributions, but in practice the convergence to the Gaussian distribution is surprisingly fast, even when the underlying distributions are, for example, uniform, as we shall see in an example in the following section.

The Gaussian distribution is completely specified by giving its expectation a and variance s^2. We denote by $N(a, s^2)$ the distribution whose density is Gaussian with mean a and variance s^2:

$$f(x)=\frac{1}{s\sqrt{2\pi}}\exp\left[-(x-a)^2/2s^2\right]$$

we can complete the statement of the central limit theorem by giving the expectation and variance of the (Gaussian) distribution resulting from summing a (large) number of independent random variables. This expectation and variance will, of course, depend on the expectations and variances of the individual distributions and can be calculated immediately using (2.1) and (2.2). Let the n independent random

variables x_i have distributions with finite expectations e_i and variances v_i. Then $S=\Sigma x_i$ will have expectation $E(S)=\Sigma e_i$ and variance $V(S)=\Sigma v_i$. This is an exact result even for finite n, which follows from (2.1) and (2.2). The fact that the distribution of S is asymptotically Gaussian is the important part of the theorem which enables us to turn our knowledge of $E(S)$ and $V(S)$ into statements of probability about the value of S for a given trial.

2.6.1. Example: Gaussian random number generator. The central limit theorem allows us to construct a Gaussian random number generator, given any other kind of random number generator, simply by taking sums of random numbers. Let us see how this works in practice, using a uniform random number generator which we assume for the moment to be given. We will denote the sum of n uniform random numbers as R_n, so that R_1 will be a random number distributed uniformly (between zero and one). Then R_2 will be distributed as in figure 1(*b*), i.e. with a density function which is a triangle. This kind of distribution is familiar to gamblers using dice, where the outcome is the sum of two numbers uniformly distributed between one and six. The extreme values of the sum (2 and 12) are the most unlikely, and the middle value (7) is the most probable. R_3 is distributed as shown in figure 1(*c*), i.e. a parabolic spline function with knots at 1 and 2 (i.e. three different parabolas joined at the points $x=1$ and $x=2$, with the first derivative continuous at these points), which is beginning to look like the well-known bell-shaped Gaussian curve. R_4 is a cubic spline function, and higher sums are higher-order spline functions which

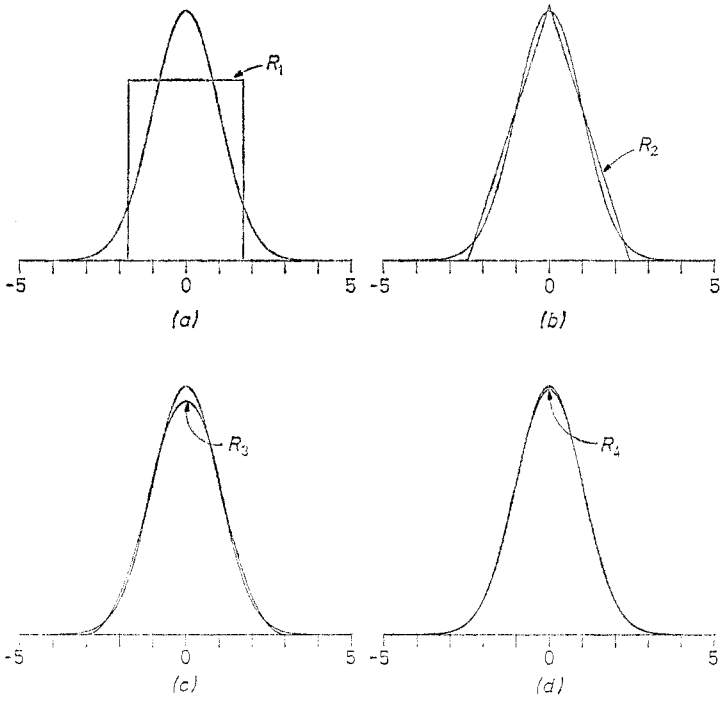

Figure 1. Distributions of sums of uniform random numbers, each compared with the normal distribution. (*a*) R_1, the uniform distribution. (*b*) R_2, the sum of two uniformly distributed numbers. (*c*) R_3, the sum of three uniformly distributed numbers. (*d*) R_{12}, the sum of twelve uniformly distributed numbers.

approximate more and more closely the Gaussian distribution. After R_5 or R_6 the distribution is almost indistinguishable from a true Gaussian by eye, except for the extreme tails which are of course of finite length whereas the true Gaussian tails go to infinity in both directions. The area under these tails is extremely small, so the discrepancy in probability content is negligible for many applications, but care must be taken since the tails may be the most important feature.

Since the expectation and variance of the uniform distribution are, respectively, $\frac{1}{2}$ and $\frac{1}{12}$ (by straightforward calculation from the definitions of expectation and variance), we have

$$E(R_n)=n/2 \qquad V(R_n)=n/12.$$

Usually we want a standard Gaussian distribution, i.e. with mean zero and variance one. We therefore take

$$\frac{R_n-n/2}{n/12}\rightarrow N(0,1).$$

A convenient choice for a practical Gaussian random number generator is $n=12$, which reduces simply to $R_{12}-6$. The properties of this generator will be discussed below in §7.

2.7. Résumé: mathematical properties of the Monte Carlo method

Let us consider again (2.3), where the left-hand side is the n-point Monte Carlo estimate of the integral on the right-hand side, the u_i being truly random numbers uniformly distributed between the integration limits a and b. The mathematical properties of this estimate are rather general properties of numerical results of Monte Carlo calculations, which we outline here.

(i) If the variance of f is finite, the Monte Carlo estimate is *consistent*, i.e. it converges to the true value of the integral for very large n.

(ii) The Monte Carlo estimate is *unbiased* for all n, i.e. the expectation of the Monte Carlo estimate is the true value of the integral. This follows directly from the linearity of the expectation operator.

(iii) The Monte Carlo estimate is asymptotically *normally distributed* (approaches a Gaussian density).

(iv) The *standard deviation* of the Monte Carlo estimate is given by $\sigma=\sqrt{V(f)}/\sqrt{n}$. This result is true for all n but is only useful insofar as the estimate is Gaussian-distributed (true only for 'large' n).

3. From Buffon's needle to variance-reducing techniques

In this section we present one of the earliest real Monte Carlo calculations, that of Buffon's needle, and examine some of its properties. We will see that its most important and worst property is its slow convergence (low efficiency). We then present a series of techniques known collectively as 'variance-reduction', designed to improve this efficiency.

3.1. Buffon's needle: hit-or-miss Monte Carlo

Although it is hard to imagine nowadays doing Monte Carlo calculations without a high-speed computer, the technique was first investigated and used long before

the existence of electronics. One such early calculation, known as Buffon's needle (Buffon 1777), was used to calculate the value of π. It is a good example of the use of the Monte Carlo method to solve a problem which has no immediate statistical interpretation and which we are accustomed to attacking with more traditional mathematical tools.

The 'calculation' proceeds as follows. Lay out on the floor a pattern of parallel lines separated by a distance d (the stripes of an American flag will do). Repeatedly throw 'randomly' a needle of length d onto this striped pattern. Each time the needle lands in such a way as to cross the boundary between two stripes, count a 'hit'. When the needle does not cross a boundary, count a 'miss'. After a given (large) number of tries, estimate π by twice the number of tries (hits + misses) divided by the number of hits.

The above recipe is based on the fact that the probability of a hit is $2/\pi$. This can be calculated very easily as follows. Let the angle between the needle and the perpendicular to the stripes be equal to a, then the projection of the needle onto this perpendicular is of length $d|\cos(a)|$ and the distance between stripes is d. For a given angle a, the probability of a hit is clearly the ratio of these two lengths, $d|\cos(a)|/d = |\cos(a)|$. Since all angles are equally likely, the average value of $|\cos(a)|$ can be calculated by integrating $|\cos(a)|$ over its range and dividing by the range. By symmetry it is sufficient to integrate over one quadrant, say from 0 to $\pi/2$, where the integral is just one, and the probability is therefore $2/\pi$.

Estimating this probability by the actual ratio of hits to random tries is called *hit-or-miss Monte Carlo* and is in general the least efficient Monte Carlo method. Let us calculate the expected accuracy after n tries. The number of hits follows a binomial distribution with expectation np (where p is the probability of a hit, $2/\pi$) and variance $np(1-p)$ (Eadie *et al* 1971, p44). The variance of $2/\pi$ is therefore $p(1-p)/n$ and the standard deviation is the square root of this. Converting this to the standard deviation on π gives $2{\cdot}37/\sqrt{n}$. (We have to know π to calculate this result, but it could also be estimated from the data.) This means that the uncertainty on the value of π is

after 100 tries: 0·2374
after 10 000 tries: 0·0237
after 1 000 000 tries: 0·0024.

These uncertainties are intolerably high compared with those of almost any other method of calculating π. In addition, physical biases are difficult to eliminate, as will be discussed below in connection with the generation of truly random numbers. We can therefore conclude that Buffon's needle represents an amusing exercise and a good example of the application of the Monte Carlo method in an unexpected domain unrelated to stochastic phenomena, but that it should not be used in practice to calculate π. Now let us see how to improve upon it, still within the general framework of the Monte Carlo method.

3.2. Integration: crude Monte Carlo

Consider doing the Buffon needle calculation on a computer. We would choose a random angle a and a random distance x from the edge of the stripe pattern along the direction perpendicular to the stripes (the outcome is clearly independent of translations along the direction of the stripes). On these (a, x) axes, figure 2 shows

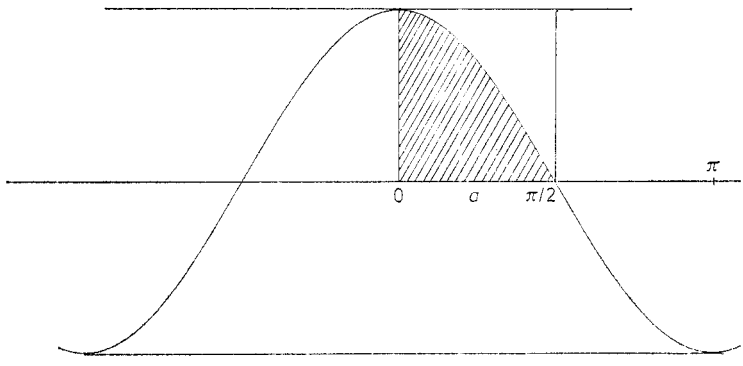

Figure 2. Buffon's needle as an integration problem.

the region corresponding to a hit, namely the area between the a axis and the curve cos (a). The calculation is equivalent to the integration of cos (a).

Let us therefore perform this integration using *crude Monte Carlo* instead of the hit-or-miss variety, by straightforward application of the method of §2, choosing randomly values of a and averaging the values of $|\cos(a)|$. It is easily verified that this results in a standard deviation smaller by a factor of 0·82. This is a general result: crude Monte Carlo is always more efficient than hit-or-miss Monte Carlo, since hit-or-miss can be considered as crude Monte Carlo on a step function taking on only values zero or one, and of all functions bounded between zero and one with a given expectation, the step function has the largest variance.

Another way of looking at the comparison between crude and hit-or-miss is the following. For a given angle a, the probability of a hit is $|\cos(a)|$. Instead of finding the expectation of this value by direct averaging (crude Monte Carlo), we take it as the probability of actually generating a hit. In order to make a hit with probability $|\cos(a)|$, generate another random number x, $0<x<1$, and call it a hit whenever $x<|\cos(a)|$. This is less efficient, but it does mean that all the values entering into the average are equal to one (or zero), which may be advantageous in some situations. In many practical calculations it may correspond to using 'unweighted' rather than 'weighted' events, by taking the weight as the probability of accepting the event. In terms of pure Monte Carlo efficiency, this unweighting procedure is always disadvantageous, but it may improve the efficiency of other parts of the calculation, as we shall see later.

3.3. Classical variance-reducing techniques

From the results of §2, the square of the uncertainty on a Monte Carlo integral is

$$s^2 = V(f)/n.$$

This uncertainty can be decreased by increasing n, but this improves (converges) very slowly. Another way is to try to decrease the effective variance $V(f)$. We have already seen one example of changing the variance in comparing crude with hit-or-miss Monte Carlo. In this subsection we introduce the most important techniques for variance-reduction.

3.3.1. Stratified sampling. We may feel intuitively that the reason why Monte Carlo integration has such a large uncertainty is that the points are chosen unevenly, and

that if the points were more uniformly distributed the fluctuations would be smaller. Intuition is not always right as we shall see in §4, but there is at least one way to make the point distribution more uniform which we can show will produce in general an improvement in the variance. Since it is a special case of a more general technique of controlling the distribution of points, let us first present the general technique.

Mathematically, stratified sampling is based on the fundamental property of the Riemann integral:

$$I = \int_0^1 f(u)\,\mathrm{d}u$$
$$= \int_0^a f(u)\,\mathrm{d}u + \int_a^1 f(u)\,\mathrm{d}u \qquad 0 < a < 1.$$

The splitting up of the integral into pieces is a common technique in adaptive numerical quadrature, but the properties of this technique in the framework of Monte Carlo integration are somewhat different. The technique consists, in the general case, of dividing the full integration interval (or space) into sub-intervals (sub-spaces), and choosing n_j points in the jth sub-interval, whose length (volume) we will denote by $\{j\}$. Then, instead of adding the contributions from all points directly, partial sums are formed over each interval, and the partial sums are added, weighted proportionally to $\{j\}$ and inversely to n_j. This yields a result with the variance

$$s^2 = \sum_j \frac{\{j\}}{n_j} \int_{\{j\}} f^2(x)\,\mathrm{d}x - \sum_j \frac{1}{n_j} \left| \int_{\{j\}} f(x)\,\mathrm{d}x \right|^2$$

which is of course just the sum of the variances of the individual pieces. If the intervals $\{j\}$ and the numbers of points n_j are chosen carefully, this can lead to a dramatic reduction in the variance compared with crude Monte Carlo, but it can also lead to a *larger* variance, so something must be known about the function in order to use this technique most advantageously.

Suppose we do not know anything about the function and simply divide the space into equal volumes $\{j\}$, choosing in each volume equal numbers of points n_j (uniform stratification). It is easily verified from the above formula, using the triangle inequality, that uniform stratification cannot increase the variance and will in general decrease it if the expectation of the function is different in the different sub-regions. In particular, if the stratification is into just two equal regions $\{1\}$ and $\{2\}$, the improvement in variance is

$$D(s^2) = \frac{1}{n} \left| \int_{\{1\}} f(x)\,\mathrm{d}x - \int_{\{2\}} f(x)\,\mathrm{d}x \right|^2.$$

Since this cannot be negative, uniform stratification can be seen to be a safe method but the improvement in variance may be arbitrarily small.

In real calculations, additional complications may arise. In many-dimensional integration, for example, it may not be at all straightforward to divide the integration region into sub-regions of known volume. Computational overheads in time and memory space may also be prohibitive.

3.3.2. Importance sampling. We have seen that a large variation in the value of the function f leads to a large uncertainty in the Monte Carlo estimate of its integral. Conversely, Monte Carlo calculations will be most efficient when each point (event) has nearly the same function value (weight). This can be arranged by choosing a

large number of points in regions of the sampling space where the function value is largest and compensating for this overpopulation by reducing the function values in these regions. In this way the reweighted function values become more nearly constant and the effective variance is reduced.

Mathematically, importance sampling corresponds to a change of integration variable:

$$f(x)\,\mathrm{d}x \to f(x)\,\mathrm{d}G(x)/g(x).$$

Points are chosen according to $G(x)$ instead of uniformly, and f is weighted inversely by $g(x)=\mathrm{d}G(x)/\mathrm{d}x$. The relevant variance is now $V(f/g)$, which will be small if g has been chosen to be close to f in shape.

To apply importance sampling to a function f, a function g must be found such that:

(i) $g(x)$ is a probability density function, i.e. it is everywhere non-negative and is normalised so that its integral over the sampling space is unity.

(ii) $G(x)$, the integral of g, is known analytically. This is an integrated distribution function, which increases monotonically as a function of x, from zero to one.

(iii) Either the function $G(x)$ can be inverted (solved for x) analytically or, alternatively, a g-distributed random number generator is available.

(iv) The ratio $f(x)/g(x)$ is as nearly constant as possible, so that the variance $V(f/g)$ is small compared with $V(f)$.

Importance sampling then proceeds as follows. Choose values of G randomly and uniformly between zero and one: for each G, solve for x, and evaluate $f(x)/g(x)$, taking the sum of these ratios as the result.

Although importance sampling is undoubtedly one of the most basic and useful Monte Carlo techniques, it suffers in practice from a number of drawbacks:

(i) The class of functions g which are integrable and of which the integral can be inverted analytically, is small: essentially the trigonometric functions, exponentials, and polynomials of very low degree, and some combinations of these. Of course the inversion can be done numerically, but this is usually slow and somewhat clumsy or else inaccurate.

(ii) True multidimensional importance sampling is extremely clumsy for all but the simplest functions, so that it is usually used one-dimension-at-a-time in multidimensional problems.

(iii) It is *unstable* in the sense that if the function g becomes very small, f/g becomes very large and in general its variance also. In particular, if g goes to zero somewhere where f is not zero, $V(f/g)$ may be infinite and the usual technique of estimating the variance from the sample points may not detect this fact if the region where $g=0$ is small. It is therefore dangerous to choose functions g which go through zero, or which approach zero quickly (such as Gaussian functions).

On the positive side, importance sampling is the only general method for removing infinite singularities in the integrand f, by using a sampling function g with a similar singularity in the same place.

3.3.3. Control variates. The control variate method is similar to importance sampling in that one again seeks an integrable function g which approximates the function to be integrated f, but this time the two functions are subtracted rather than divided. Mathematically, this technique is based on the linearity of the integral operator:

$$\int f(x)\,\mathrm{d}x = \int\{f(x)-g(x)\}\,\mathrm{d}x + \int g(x)\,\mathrm{d}x.$$

Now, if the definite integral of g over the entire interval is known, the only uncertainty comes from the integral of $(f-g)$, which will have a smaller variance than f if g has been chosen carefully.

The method of control variates is more stable than importance sampling, since zeros in g cannot induce singularities in $(f-g)$. Another advantage over importance sampling is that the integral of the 'approximating function' g need not be inverted analytically.

3.3.4. Antithetic variates. Usually Monte Carlo calculations make use of random numbers (points) which are *independent* of each other, at least in principle. The method of antithetic variates deliberately makes use of correlated points, taking advantage of the fact that such correlation may be negative as well as positive. We recall from (2.2) that the variance of the sum of two function values f' and f'' is just the sum of the individual variances when the random points where the function is evaluated are chosen independently, but that in the general case an additional term is present:

$$V(f'+f'')=V(f')+V(f'')+2\,\mathrm{cov}\,(f',f'').$$

If we can arrange to choose points such that f' and f'' are negatively correlated, a substantial reduction in variance may be realised. This requires knowledge of the function f, and it is not easy to give general methods for accomplishing this negative correlation. Hammersley and Handscomb (1964, pp60–5) discuss this in some detail and give further references. For our purposes it will suffice to give a simple example to see how the technique works in general.

Suppose that it is known that $f(x)$ is a monotonically increasing function of x. Then choose x_i randomly and independently as usual, uniformly distributed between the integration limits (say, 0 to 1), but instead of forming the sum of $f(x_i)$ we take one-half of the sum of $\{f(x_i)+f(1-x_i)\}$. Then each time x_i is small, resulting in a small value of $f(x_i)$, $1-x_i$ and thus $f(1-x_i)$ will be large, and vice versa. The partial sums $\{f(x_i)+f(1-x_i)\}$ will therefore be more constant than the individual function values and have a lower variance. Looked at in another way, we are taking the average of the estimate of the integral of $f(x)$ and the estimate of the integral of $f(1-x)$ using the same points x, and since these two functions are highly (negatively) correlated, the variance of the sum is less than the sum of the variances.

3.4. Adaptive variance-reducing techniques

With the possible exception of uniform stratification, all the variance-reduction methods described above require some advance knowledge of the behaviour of the function, and if misapplied may easily lead to a degradation of the Monte Carlo efficiency rather than an improvement, not to mention the additional labour factor involved in the application of the variance-reduction. A natural extension is toward *adaptive* techniques which learn about the function as they proceed, preferably requiring no *a priori* knowledge about the function. Similarly inspired techniques abound in numerical quadrature where it is probably safe to say that most automatic function integration is done using adaptive methods. Truly adaptive methods for Monte Carlo integration are less common, perhaps because they are rather difficult to realise (and easy to misinterpret). We shall consider three examples which should serve to illustrate the problems involved and ideas that have proved to be useful.

The programs I shall describe here are all designed for multidimensional integration of general functions, especially badly behaved functions with spikes and large variances.

3.4.1. Sheppey and Lautrup's RIWIAD. The program RIWIAD of Sheppey and Lautrup is one of the earliest to be used with success on difficult multivariate functions on the hypercube. It first divides the full hypercube evenly into a number of sub-hypercubes and estimates the integral and its variance in each hypercube by crude Monte Carlo (uniform stratification). Based on the values found in each sub-volume, it then adjusts the boundaries to form new hyper-rectangles such that sub-volumes are smaller where the function is larger, and the process is continued. At each step, an estimate of the integral and its uncertainty is made in each sub-volume, and the interval boundaries are modified to improve the next stratification. A running weighted average of the integral estimates and uncertainty estimates is maintained, and the procedure stops when the desired uncertainty is achieved.

RIWIAD has several drawbacks. The stratification boundaries are always parallel to the original parameter axes and always run along the whole length of the hypercube, dividing all the volumes through which they go, even if the previous results indicated that some of these sub-volumes did not have to be divided. Worst of all, the weighted average of partial results produces a bias due to the correlation between the estimate of the expectation and the estimate of the variance. Suppose, for example, that the function has a narrow spike, and that on the first step no point falls in the spike. Both the integral and its variance will be estimated too low. Then on the next step, a point hits the spike; this time the estimates are both about right, but since the variance is large the value gets a low weight and the overall estimate remains too low. The program never recovers from such an incident since it never forgets an early value even if later experience shows it to be a bad estimate.

3.4.2. Friedman's adaptive importance sampling. A more recent program of J Friedman (unpublished, superseded by his more recent effort described immediately below) uses a quite different approach. The program is divided into an exploratory phase and an evaluation phase, and none of the function values found in the exploratory phase are used explicitly in the evaluation. This avoids the bias due to the way the exploratory points are chosen, at a modest cost in efficiency. The exploratory phase is used to establish a control function which will be used for the importance sampling of the evaluation phase. The control function is a sum of Cauchy (Breit–Wigner) peaks, whose positions and shapes correspond to those of the function to be integrated, as determined respectively by a peak search using a function-minimising routine, and an eigenvector analysis of the covariance of the function around each peak. Cauchy-shaped peaks are used because they tend to zero more slowly than Gaussian peaks, helping to avoid the instability problem mentioned above.

Although this program is an improvement over RIWIAD for most functions, it also has several drawbacks in practice and is unsuitable for functions which cannot be approximated by a small number of peaks.

3.4.3. Friedman's DIVONNE2 *with recursive partitioning.* A more recent offering of Friedman, called DIVONNE2 (Friedman 1977a, b), represents a synthesis of the ideas seen to be most valuable in the above programs, together with some more modern

ideas in multidimensional data structures. It consists of two separate programs, the first of which performs a recursive multidimensional partitioning (stratification) of the function parameter space, and the second does a stratified-sampling Monte Carlo integration based on this partitioning.

The goal of the partitioning is to produce sub-volumes in which the *range* of function values, as determined by function-minimisation techniques, is as small as possible. The partitioning program retains the drawback of RIWIAD that partition boundaries must be parallel to the parameter axes, but since the partitioning is recursive (only one sub-volume is split in two at each step, not a whole row), the algorithm eventually tends to liberate itself from the orientation of the axes.

The partitioning algorithm has other applications than integration and can be used, for example, in conjunction with a specially designed random number generator to generate points in the parameter space distributed according to the function f (see the subsection below on generating random numbers according to empirical distributions).

The actual integration need not be performed using Monte Carlo. Other methods are offered as options in the program, but in practice this choice does not seem to make much difference in the accuracy obtained, and Monte Carlo is usually used because it gives a reasonably accurate uncertainty estimate.

4. Comparison with numerical quadrature

In order to decide whether a Monte Carlo method should be applied to a given problem, it is reasonable to see how it compares with other available methods. In the case of integration, alternative numerical techniques have been the subject of extensive studies for centuries, and the widespread use of computers has led to considerable practical experience in this field. The current section is a brief review of the properties of numerical quadrature as it is commonly practised today, for the purposes of comparison with Monte Carlo. This is not intended to be a complete or detailed account of any quadrature techniques but is intended only to give the properties of most use in deciding whether to use quadrature at all.

4.1. One-dimensional quadrature

Unless otherwise stated, numerical quadrature is always done in one dimension. Some of the reasons for this will appear later, but certainly a prime motive for sticking with one dimension is the beauty and elegance of the methods that have been developed for one dimension.

All quadrature formulae approximate the value of the integral by a linear combination of function values:

$$I_Q = \sum_{i=1}^{n} w_i f(x_i).$$

Different formulae correspond to different choices of the points x and the weights w. Crude Monte Carlo could be considered a quadrature formula with unit weights and points chosen uniformly but randomly.

4.1.1. Trapezoidal rule. This simplest of all rules consists of dividing the total interval into n sub-intervals and approximating the integral over each sub-interval

by the area of the trapezoid inscribed under (or over) the curve to be integrated. The sum of these approximations reduces to the average of the $n+1$ function values multiplied by the length of the interval (in fact, the end points must be added with a factor one-half, but this important detail can be considered as a boundary correction and is not relevant to our arguments here). For large n, we can think of the function expressed as a Taylor series expansion about each of the n points: then the constant terms and the first derivative (linear) terms will be integrated exactly by the trapezoidal rule, and to the extent that higher-order terms are of decreasing importance, the largest contribution to the error will come from the second derivative (constant curvature) terms. This error is proportional to the sagittas of the curve segments over each band, and these sagittas will each be proportional to the square of the distance between successive points. Therefore if the function is evaluated at n equally spaced points, the uncertainty on the integral should be proportional to $1/n^2$ for large n.

Recall that for Monte Carlo integration, the convergence was only like the square root of n, so that where increasing n by a factor of 100 only buys you one more decimal digit with Monte Carlo, you get *four* digits with the trapezoidal rule. This is especially interesting because the two methods are so similar. Indeed, the methods are identical except that points are chosen equally spaced in one case and randomly in the other, and the randomness apparently causes us to lose a factor of four in convergence rate (decimal digits per factor of 100 increase in n). Before seeing what randomness gives us in return for this disastrous convergence rate, let us consider still more impressive convergence rates of other quadrature methods.

4.1.2. Higher-order quadrature. By choosing the points and weights appropriately, it is possible to integrate exactly polynomials of higher degree and therefore achieve higher convergence rates. The next step after the trapezoidal rule is Simpson's rule which requires three points on a given interval and integrates exactly all polynomials of degree three. The highest possible degree for a given number of points is achieved with Gauss quadrature formulae which integrate exactly all polynomials of degree $2n-1$ (or less) with n carefully chosen points and n corresponding weights. The numerical values of these points and weights, as well as the basic properties of Gaussian quadrature, are given by Stroud and Secrest (1966).

The theoretical convergence rate for Gauss quadrature is enormously higher than for Monte Carlo, but some of its other properties are not so nice. The uncertainty is not easy to estimate, error-bound formulae being given in terms of the values of higher derivatives of the function over the interval, which are much harder to calculate than the integral itself, so are essentially useless in practice. In addition, the validity of the error-bound formulae depends on continuity properties of the function and its derivatives, which may not be known. In practice, one is forced to use 'overkill', aiming at a precision much higher than that required, and uncertainties, if estimated at all, are usually estimated by comparing the results of more than one different Gauss rule on the same interval. Unfortunately, the nature of these rules is such that the best way to combine the results of two different Gauss rules over the same interval is to throw away the lower-order result and keep only the higher. Practical experience indicates also that there is no advantage in going to extremely high orders, and that beyond about 12 or 15 points it is usually better to split the interval and apply a lower-order rule several times. This indication of the breakdown of the polynomial philosophy is discussed below.

4.1.3. Adaptive quadrature. The quadrature rules described above are all fixed-point rules, i.e. the points and weights are fixed in advance. Adaptive quadrature, on the other hand, is an attempt to attain a prescribed accuracy by adapting the quadrature method to the function. The most common class of adaptive methods consists in using a fixed-point rule and an error-bound estimate, then dividing the interval into two or more pieces, usually of equal length, if the error-bound estimate exceeds the required value. The same procedure is then applied recursively to each sub-interval until all sub-intervals satisfy the error bounds, or until the sum of all estimated uncertainties reaches an acceptable level. The most common strategies are compared by Malcolm and Simpson (1975).

Most computer centres offer one or more 'automatic integration' programs based on adaptive quadrature of the above type. These programs differ mainly in the fixed-point rule used and in the method of obtaining an estimate of uncertainty which, as we have seen, is not always straightforward. Because of problems in obtaining reliable estimates of uncertainty, the better programs aim for a certain amount of overkill, but may be unreliable nonetheless. For example, a spline function which appears smooth to the eye has discontinuous higher-order derivatives which tend to produce poor results with high-order Gauss rules and consequently adaptive quadrature based on them. Other problems with adaptive quadrature are discussed by Lyness and Kaganove (1976).

4.2. Multidimensional quadrature

Numerical quadrature formulae are based on the study of orthogonal polynomials, which are well understood in one dimension. For higher dimensionalities the mathematical basis is not as well understood, and practical studies are much more recent and less extensive. We outline here briefly the current situation.

4.2.1. Multidimensional region boundaries. In one dimension, only three 'different' regions of integration need to be considered: finite, semi-infinite and infinite. Choosing one particular interval in each class, all other intervals can be mapped onto one of the three by a linear mapping, which conserves all the convergence properties of any integration method. In general in this review, we consider only the finite interval. Simple non-linear transformations are available to transform semi-infinite and infinite intervals into the unit interval, and this is a standard way to perform integration over infinite intervals, but these transformations do modify the properties of quadrature rules. For Monte Carlo integration, these transformations do not affect the n dependence of the convergence, but the function whose variance determines the uncertainty of the estimate is, of course, the transformed function.

In more than one dimension, the situation is quite different. Already in two dimensions, and restricting ourselves to finite regions, there are an infinite number of 'different' regions which cannot be transformed into each other by linear transformations. For example, a circle is fundamentally different from a square, in the sense that a quadrature formula for a square will not have the same properties when applied to a circle.

The standard Monte Carlo technique for dealing with odd-shaped regions is to embed the region in the smallest hyper-rectangle that will surround it and integrate over the hyper-rectangle, throwing away the points that fall outside the inner region. This leads to some inefficiency of course, due to the rejected points, but is capable

of dealing in a straightforward way with essentially any finite region. Such a general technique does not work for numerical quadrature methods, since it introduces discontinuities on the boundary of the inner region, thus destroying some of the nice convergence properties.

The ability of Monte Carlo to integrate over complicated multidimensional regions (albeit not always very efficiently) is one of its most valuable properties, since it is often the only known technique capable of handling such problems. Purists may be right in saying that this only expresses our ignorance of better methods, but for people with real problems to be solved, it does represent a way out.

4.2.2. Extension of one-dimensional rules. For rectangular regions, which are after all the most common, multidimensional quadrature rules can be formed by straightforward extension of one-dimensional rules. Such rules, known as product rules, generally preserve the properties of the one-dimensional rules of which they are extensions, but only at the cost of increasing the number of points exponentially with the dimensionality. Thus a product rule requiring n function evaluations in one dimension will require n^2 evaluations in two dimensions, n^3 in three dimensions, and so on. This slows down the effective convergence rate in d dimensions by a factor $1/d$ in the exponent as shown in the table below.

Uncertainty as a function of number of points n	In one dimension	In d dimensions
Monte Carlo	$n^{-1/2}$	$n^{-1/2}$
Trapezoidal rule	n^{-2}	$n^{-2/d}$
Simpson's rule	n^{-4}	$n^{-4/d}$
Gauss rule	n^{-2m+1}	$n^{-(2m-1)/d}$

Since the convergence of Monte Carlo is independent of dimensionality, there is always some d above which Monte Carlo converges faster than any fixed quadrature rule. Thus Simpson's rule in more than eight dimensions converges more slowly than Monte Carlo and a ten-point Gauss rule converges more slowly than Monte Carlo in more than 38 dimensions, even assuming that the function has the nice continuity properties required by these higher-order rules.

But suppose we actually try to apply a ten-point Gauss rule in 38 dimensions. This requires at least 10^{38} function evaluations, which is clearly unfeasible. This brings up two new points.

(i) *The feasibility limit* is the largest number of function evaluations we can afford to make. Depending on the computer resources available, the feasibility limit will usually be between 10^5 and 10^{10} points for functions which can be evaluated reasonably fast. This limits the use of a ten-point Gauss rule to five dimensions for someone with moderate computer resources, or ten dimensions for someone with 'unlimited' computer resources. Figure 3 shows that, except for very-low-order rules, the feasibility limit is reached long before the crossover point where Monte Carlo converges faster than quadrature, so that the theoretical convergence rates for high-order rules in high dimensionalities will remain purely theoretical.

(ii) *The growth rate* is the smallest number of *additional* function evaluations needed to improve the current estimate. Monte Carlo estimates can be improved by adding a single point, but at the other extreme Gauss rule estimates can only

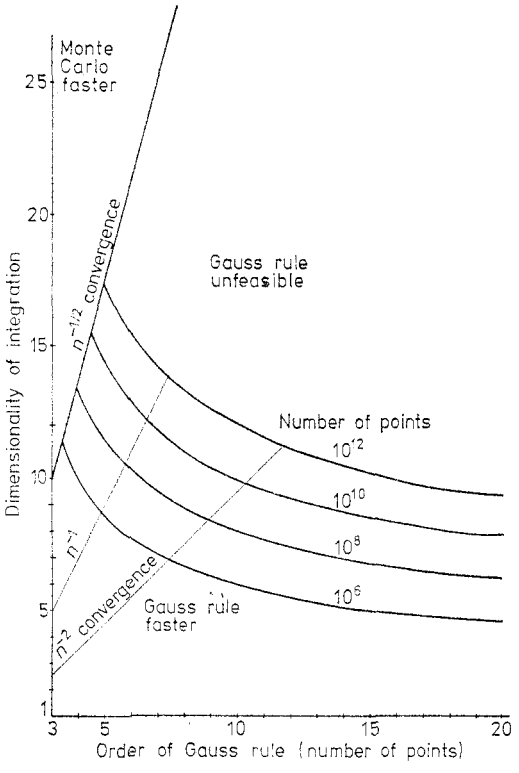

Figure 3. Comparison of Monte Carlo integration and numerical quadrature in many dimensions.

be improved by going to a higher-order rule, requiring $(m+1)^d$ additional points or by sub-dividing the space, which requires at least $2m^d$ additional points even for the simplest partitioning. In both cases all the previous Gauss points must be thrown away.

One way to get around the problems of feasibility limit and growth rate has been suggested by Tsuda (1973). He uses a rule with far too many points actually to evaluate, and then applies the standard Monte Carlo technique of sampling the resulting terms randomly. He reports good results for this combination of quadrature and Monte Carlo, but the reasons behind this success are not clear to me. It could be that the use of points of a quadrature rule guards against any two points being too close, and therefore ensures a certain uniformity of distribution even if only a random subset of these points is actually used (this explanation was suggested to me by J Friedman).

4.2.3. Multidimensional rules. The situation is greatly improved if truly multi-dimensional quadrature rules are used instead of product rules. Unfortunately, good quadrature rules are not known for many regions, dimensionalities and orders. The situation is well described in the article of Haber (1970) and in the book of Stroud (1971), of which we summarise some of the more important results here.

As in one dimension, multidimensional formulae can be found which will integrate exactly any polynomial of degree less than or equal to some degree r. In addition we may require the formulae to satisfy two important criteria.

(i) That all the weights be positive. This is important in order to avoid numerical instabilities arising from cancellation of large terms of opposite sign, and seems also to make the formulae more robust with respect to the validity of the polynomial assumption.

(ii) That all the points used lie within the region of integration. This seems such an elementary requirement that one is surprised to discover that many formulae in d dimensions do not possess it, even for convex regions.

If we restrict ourselves to formulae satisfying the above requirements, very few generally applicable formulae have been found. Even for the simplest region, the hypercube, the only known formulae valid for all dimensionalities and even reasonably close to the theoretical efficiency limit are of degree 2 and 3, as summarised in the table below. We see that there exist low-order multidimensional formulae con-

Degree r	Best known n	n (Gauss)
2	$2d+1$	
3	$2d$	2^d
5	$O(d^5)$, not found	3^d
>5	?	$\left(\frac{r+1}{2}\right)^d$, r odd

siderably better than the Gauss rule, and in fact some higher-order formulae of comparable theoretical efficiency are known, but they do not have all positive weights for all r. Stroud has shown that a formula of degree $r=5$ exists, with n of the order of d^5 and positive weights, but to my knowledge no one has as yet found it.

4.2.4. Adaptive multidimensional quadrature. Like non-adaptive quadrature, adaptive quadrature is much better developed in one dimension than in many dimensions, since the problems mentioned above for multidimensional quadrature, in general, clearly make adaptivity difficult too. Nevertheless, several attempts have been made, of which we will mention a few that have been published. They appear to be reasonably successful, at least for small dimensionalities (up to 6).

(i) Van Dooren and de Ridder (1976) have published an algorithm not too different from Friedman's DIVONNE2 (1977a, b), except that the former use multidimensional extensions of one-dimensional Gauss rules instead of Monte Carlo for the basic integration technique and their sub-division of regions is always into two equal parts.

(ii) Genz (1972) presents an algorithm especially interesting for its use of extrapolation methods, but the multidimensional adaptivity does not seem to result in a great improvement in efficiency.

(iii) Kahaner and Wells (1979) use an interesting technique based on simplices rather than hypercubes. Their basic thesis is that the lack of good adaptive quadrature procedures in many dimensions is mainly due to problems in organisation of multidimensional data structures. Their work is as much an exercise in programming as numerical analysis and it presents many interesting ideas in both areas. It probably points in the direction where we can expect the most significant advances. From a practical point of view, their program is not of much interest since it is written in a language (MADCAP) not generally available.

4.3. The Monte Carlo paradox

Some of the conclusions to be drawn from the comparison of Monte Carlo integration with numerical quadrature are somewhat surprising and call for deeper consideration.

(i) In one dimension, the perfectly 'regular' trapezoidal rule converges much faster than the identical rule with randomly distributed points, but in many dimensions a random distribution leads to faster convergence than the perfectly regular grid.

(ii) Just to confuse matters further, the random distribution which is superior to the regular distribution in many dimensions can nevertheless be improved by making it more uniform, either by stratified sampling as we have already seen, or through quasi-Monte Carlo which is discussed below.

The explanation of this paradox is that our intuitive feeling for what constitutes 'uniformity' in distribution, based on one-dimensional knowledge, is not quite right for higher dimensions. For example, consider the projection of the point distribution onto one axis, for the hyper-rectangular grid of points. Great spikes appear in this projection whenever we come to a 'hyper-row' of points, which no longer looks very uniform; the projections of a random distribution are more uniform in this sense. (See Sobol (1979) for a simple and convincing example of this.) In the section on quasi-Monte Carlo, we will define and discuss a more precise measure of uniformity (or non-uniformity) called discrepancy, which will explain this paradox.

Furthermore, the volume of multidimensional space is always very big, so that points are always far apart, which negates the very basis of quadrature rules.

4.3.1. The 'polynomial hangup'. Let us look more carefully at the theoretically fast convergence rate of high-order quadrature rules. This is related to the 'polynomial hypothesis' dear to the hearts of quadrature experts. For low orders, it is hard to find fault with the polynomial hypothesis; the zero-degree polynomial is certainly the simplest function and it is reasonable to expect a good integration method to be able to integrate it exactly. (Even Monte Carlo does that, by the way!) Similarly, a first-degree (straight-line) polynomial naturally comes next in the scale of complicatedness as perceived by the human eye, but who is to say that a parabola is simpler or smoother than, for example, a sine function or an exponential? Is there a justification for seeking methods that integrate exactly polynomials of degree r, when the function to be integrated is not a polynomial?

We may seek such a justification in Taylor's theorem. This theorem states that under certain conditions any function can be expressed as a polynomial of degree r, plus a remainder term. The conditions are that the function and all its derivatives should be continuous; the coefficients of the polynomial are given in terms of these derivatives evaluated at the point where the independent variable is equal to zero. The usefulness of the theorem comes from the cases where the remainder term becomes very small as r increases, but the theorem says nothing about when this can be expected to be true.

Indeed, there is nothing special about the polynomial in this respect. Other theorems give conditions under which general functions can be expressed as other infinite series (e.g. trigonometric series, Fourier series) and conditions under which the series can be truncated with a given remainder. The property that makes the Taylor series special is that under very general conditions the higher-order terms

can indeed be neglected *in the neighbourhood of zero* (the point about which the expansion is performed). Unfortunately, this has very little to do with the macroscopic properties of the function which are important for integration over a large region, especially a multidimensional region which is always large.

In practice, polynomials are notoriously bad at approximating functions over large intervals, so we should not be surprised that related quadrature rules sometimes give unsatisfactory results. Experience shows spline functions to be good at approximating a wider class of functions, and although spline functions are piecewise polynomials they are not polynomials, and indeed have discontinuous derivatives of some degree at the knots. Integrating spline functions is, of course, easy if you know where the knots are, although Gauss rules generally fail without this knowledge.

5. Random and pseudo-random numbers

In principle, a random number is simply a particular value taken on by a random variable (which was defined above). However, in Monte Carlo studies, one often uses the word 'random' with various other, quite different, meanings. Here it is usually applied to sequences of numbers which, once they have been determined, are not at all random in the statistical sense but may have some properties which are similar to the properties of a truly random sequence. To be precise one must distinguish three different types of sequences: *truly random*, *pseudo-random* and *quasi-random*. (The first two of these are described in this section, and the third in the section on quasi-Monte Carlo.)

Unfortunately, it is common to confuse the *randomness* properties of a sequence with its *distribution*. This is unnecessary, since the two are quite independent. A perfectly random sequence may have any distribution (uniform, Gaussian, etc), whereas a perfectly uniformly distributed sequence may be not at all random.

5.1. Truly random numbers

A sequence of truly random numbers is unpredictable and therefore unreproducible. Such a sequence can only be generated by a random physical process, for example radioactive decay, thermal noise in electronic devices, cosmic ray arrival times, etc. If such a physical process is used (properly) to generate the random numbers for a Monte Carlo calculation, there is no theoretical problem, since the theory outlined above is sufficient justification, provided there is no physical defect in the apparatus.

In practice, however, it turns out to be very difficult to construct physical generators which are fast enough (one needs typically hundreds of floating-point numbers per second) and at the same time accurate and unbiased. Faced with these practical difficulties, very few large-scale calculations have been made using such generators.

One important exception is the work of Frigerio and Clark (1975) and Frigerio *et al* (1978). They used a radioactive alpha-particle source and a high-resolution counter turned on for periods of 20 ms, during which time they counted, on average, 24·315 decays. Whenever the count was odd, they recorded a zero-bit, and when even, a one-bit, all written to magnetic tape. A careful correction was made to eliminate the bias due to the fact that the probability of an odd count is not exactly one-half

(the bias could have been removed without even knowing this probability, using the method given in the next subsection). Their apparatus yielded about 6000 31-bit truly random numbers per hour. These numbers have been stored on magnetic tape, subjected to a number of tests for 'randomness' and used in Monte Carlo calculations. (Copies of the tape, containing 2·5 million truly random numbers, are available from the Argonne National Laboratory Code Center, Argonne, Illinois 60439, USA.)

To illustrate the practical problems of physical bias in truly random generators, let us again consider the Buffon needle experiment. First of all, the width of the stripes must be constant and equal to the length of the needle to within the accuracy ultimately desired for the final result, which is not so hard if we only want one or two figure accuracy, but will clearly prevent us from going much further. Also, an unbiased decision procedure must be found for the cases when the needle almost crosses a boundary. Thirdly, we must ensure that the actual distribution of angle and position of the needle is uniform. The angular distribution may be made uniform by spinning the needle very fast as it is thrown, provided the surface is very flat and of homogeneous friction properties. The distribution of needle position will not be uniform but may be expected to follow some Gaussian distribution about the point where the thrower aims. In practice, one would determine the width of this distribution experimentally and carry out a rather complicated correction of the type performed by Frigerio *et al* as described above.

5.1.1. Bias removal technique. It often happens when generating truly random numbers, as in the example just above, that the major problem is in determining the exact distribution (i.e. the bias of the apparatus), whereas the 'truly random' property is guaranteed by the nature of the physical process used. In these circumstances, a very useful trick to eliminate the bias is the following.

Suppose we are given a truly random sequence of zeros and ones, but where the probabilities $P(0)$ and $P(1)$ may not be exactly one-half. Using this original sequence, we produce a second sequence in the following way. Consider *pairs* of bits in the sequence, and if the two bits in the pair are the same, reject both bits; if the two bits are different, accept the second bit (always rejecting the first of each pair). The new sequence thus formed is guaranteed to have zeros and ones with equal probability as long as there was no correlation between the bits of the original sequence. This can be seen easily by calculating $P'(0)$ and $P'(1)$, the probabilities of zero and one in the new sequence, in terms of $P(0)$ and $P(1)$, the original probabilities. Since a zero can only come from a one followed by a zero, $P'(0)=P(1)P(0)$ and similarly $P'(1)=P(0)P(1)$. These probabilities must therefore be equal no matter what $P(0)$ and $P(1)$ are. Unfortunately $P'(0)$ and $P'(1)$ do not add up to one because the probability of rejecting a pair entirely is $P^2(0)+P^2(1)$, which must be greater than or equal to one-half. In addition, half the bits are lost because a pair yields at most one bit, so the efficiency of the procedure is at most 25% but it allows the use of a basic generator which is of unknown bias, as long as this bias is nearly constant in time. (Any method using an explicit correction for bias must also know the exact time dependence of this bias.)

The efficiency of this method is easily seen to be $P(0)P(1)$, which is equal to $P(1-P)$, where P is either $P(0)$ or $P(1)$. This means that, for heavily biased original sequences, the efficiency is approximately equal to the probability of the less probable bit.

5.2. Pseudo-random numbers

The random numbers most often used in real calculations are those known as pesudo-random, which are generated according to a strict mathematical formula and therefore reproducible and not at all random in the mathematical sense but are supposed to be indistinguishable from a sequence generated truly randomly. That is, someone who does not know the formula is not supposed to be able to tell that a formula was used rather than a physical process. The theory outlined in §2 is generally assumed to hold for Monte Carlo results calculated with pseudo-random numbers as well as with truly random numbers.

Unfortunately, there is no way to generate such numbers, which are both truly random and not truly random. This has not prevented people from using pseudo-random sequences (often with considerable success), closing one eye to the theoretical impossibility of it all. In this subsection we discuss how this is done in practice.

5.2.1. From mid-squares to multiplicative generators. Perhaps the earliest pseudo-random number generator was that of Von Neumann known as 'mid-squares'. Given a starting number of r digits, the first 'random' number is the middle $r/2$ digits of this number. Then the first 'random' number is squared (forming another number of r digits) and the middle $r/2$ digits of this square are the second 'random' number, etc. The digits may be decimal, octal, binary or in any other base. If the original number is chosen carefully, this method can yield a reasonably long string of numbers which appear random, but the properties of this generator, to the extent that they are known at all, are not very good, and it is not used any more. First of all, this generator is characterised by a *period*, since if any number reappears the entire sequence from the first appearance to the second will reappear. This is a rather general property of pseudo-random generators, including those commonly used today. Also, certain numbers reproduce themselves immediately (for example, zero), which means that those numbers can never appear unless the period is one.

It may appear that the mid-squares method cannot possibly be very good because it is not complicated enough. The naive approach then consists in 'improving' the unacceptable method by making it more complicated. An excellent example of how one might do this is given by Knuth (1969, pp4–6). His 'super-random' generator is so complicated that one could never hope to understand its properties, and turns out nevertheless to be very bad. The lesson to be learned is that a simple generator, whose properties (and weaknesses) are known, is always to be preferred to a complicated generator of unknown properties. A corollary of this lesson is that it is not easy to 'improve' a bad pseudo-random generator by making it more complicated. Such an exercise cannot add any true randomness and usually serves only to shorten the period by using up several numbers to produce one. Exceptions to this are the shuffling technique discussed below in connection with quasi-random numbers and the Dieter–Ahrens generator also discussed below.

Indeed the pseudo-random generator most widely used is even somewhat simpler than mid-squares; it is the method attributed to D H Lehmer, known as *multiplicative congruential* or *linear congruential.* Given a modulus m, a multiplier a and a starting value r_0, the method generates successive pseudo-random numbers by the formula

$$r_i = ar_{i-1} \pmod{m}.$$

A variation known as the *mixed congruential* generator requires, in addition, an additive

constant b:

$$r_i = ar_{i-1} + b \pmod{m}.$$

The two generators have very similar properties and will be considered together. For both generators, m is invariably chosen as 2^t, where t is the number of bits in the representation of an integer on the computer being used, so that in practice the algorithm consists of multiplying two numbers of t bits each, yielding a number of $2t$ bits, of which the *lower* (least significant) t bits are retained as the next 'random' number. These integers are then converted to floating-point numbers in the range zero to one by dividing by m.

5.2.2. The early approach: maximum period. It turns out to be a relatively easy problem in number theory to give the conditions for a congruential generator to attain the maximum period, which is generally of length $m/4$. Early theoretical results therefore concerned primarily this aspect with very little progress on other properties. This gave rise to a large number of generators with long periods, which were then subjected to 'tests for randomness' and the ones for which no 'non-random' behaviour could be discovered were used. Often these generators were later found to be unacceptable but continued to be used by those who had not yet stumbled upon the unfortunate properties†.

The 1960s may be termed the 'dark ages' of pseudo-random generators, characterised by an enormous number of articles (mostly unpublished) purporting to show, on the basis of 'tests' as described below, that one pseudo-random generator was better or worse than another.

5.2.3. Testing pseudo-random generators. Since there was in the early days no good theory about the behaviour of pseudo-random number generators, it was necessary to resort to 'tests of randomness' in order to certify a given generator as 'good'. These tests usually consist of forming some function of a given string of pseudo-random numbers and comparing the value of this function with the expected value of the same function of truly random numbers. For example, the simplest test would be to take the average of the first n numbers from a pseudo-random generator, which should be close to 0·5, the expectation of the average of truly random numbers uniformly distributed between zero and one. The variance of the average for truly random numbers being $n/12$, the square root of this quantity is the expected standard deviation, so we expect that 95% of the strings of n numbers will have an average within two such standard deviations of 0·5. If our pseudo-random generator yields an average which falls outside this range, we say that it fails that test at the 5% level. Of course, even a truly random sequence would fail such a test 5% of the time, but that is just too bad.

In practice, one uses somewhat more complicated tests, based on more complicated functions. These tests have names such as the runs test, poker test, etc. Some tests are felt to be more sensitive than others, but since one does not in principle know what kind of 'non-randomness' to look for, it is not possible to measure the power of a test in any precise way. The most common tests are described abundantly in the literature (e.g. Ahrens *et al* 1970) and summarised in Knuth (1969).

† The best example of this is RANDU which was distributed by IBM with their 360 series and was found almost immediately to be very poor. One can still find articles being published today by people just getting around to making this painful discovery.

Since there is an infinite number of possible functions that could be applied to each of the possible sequences coming from a pseudo-random generator, no generator can be 'tested' thoroughly. The most interesting such function is just the calculation for which the pseudo-random numbers are needed, and the (unknown) correct answer to this problem provides yet another test of the generator—indeed, the only test we really care about. The philosophy of pseudo-Monte Carlo could therefore be stated in these words: if a pseudo-random number generator has passed a certain number of tests, then it will pass the next one, where the next one is the answer to our problem. It is, of course, not known in general why it should pass this next test, except for the fact that it is not known why it should not.

A somewhat different kind of test was used by J Lach (1962, unpublished) who was suspicious because results using the IBM 709 pseudo-random generator produced fluctuations greater than expected. He simply plotted the random number distribution on a cathode ray display and observed the 'non-randomness' by eye. Taking pairs of numbers as (x, y) coordinates of points, no obvious correlations were seen, but when triplets (x, y, z) were considered and (x, y) were plotted only for $z < 0{\cdot}1$, the resulting point distribution showed a structure of slanting bands, with all the space between the bands completely empty of points. The pseudo-random generator was later corrected by changing the multiplier so that the particular effect observed by Lach disappeared, but what Lach had observed was later showed by Marsaglia to be a defect inherent in all generators of this type (see next subsection).

My personal feeling about testing is that it is best to avoid it through a deeper theoretical understanding of the generator. (In the case of the multiplicative congruential generator, the important properties are now known exactly; see below.) If testing must be done, I prefer visual tests of the type used by Lach, since these tests are not only rather sensitive to the kinds of 'non-randomness' we are interested in, but may also give some insight into the properties of the generator.

5.2.4. The Marsaglia effect. In his classic paper *Random numbers fall mainly in the planes* Marsaglia (1968) finally brought some genuine understanding into the occult art of pseudo-random number generation. He showed that if successive d-tuples from a multiplicative congruential generator are taken as coordinates of points in d-dimensional space, all the points will lie on a certain finite number of parallel hyperplanes, this number always being not greater than a certain function of d and the bit length of integer arithmetic on the machine. We give some values of this function in the table below.

Maximum number of hyperplanes $= (d!\, 2^t)^{1/d}$				
Number of bits(t)	$d=3$	$d=4$	$d=6$	$d=10$
16	73	35	19	13
32	2953	566	120	41
36	7442	1133	191	54
48	119086	9065	766	126
60	1905376	72520	3064	290

Furthermore, it is usually the case that the points lie on more than one such set of hyperplanes, making an extremely regular pattern rather than the 'random'

distribution desired. (Of course, it is true that any points must lie on some set of hyperplanes, but truly random points would lie on a much larger number of such planes.) We can use the table above to decide the maximum dimensionality for which we care to use such random numbers to perform, for example, numerical integration, based on the word length of our machine. For machines with long words, the limit is probably beyond anything we would be likely to need, but with integers of 36 bits and less, care must be taken.

Note that Marsaglia completely explained the effect observed earlier by Lach, and which was 'corrected' by changing the multiplier of the generator. Lach was observing the hyperplanes in three-dimensional space and taking a slice in one of the dimensions produced the bands when projected onto the other two dimensions. Changing the multiplier may have increased the number of planes, and certainly changed their orientation, so that the effect then appeared to go away. Lach was using a computer with 36-bit integers, so that it should have been possible to get a good distribution in only three dimensions.

5.2.5. The Dieter–Ahrens solution. About the same time as Marsaglia was dicovering the hyperplanes, he and others were investigating multiplicative generators in more detail (Marsaglia 1972) and found ways to determine, for example, the exact distribution of pairs of numbers (Dieter 1971), and the autocorrelation function (Dieter and Ahrens 1971). The result of all this work is a good understanding of both the good and bad properties of such generators, as well as how to find good multipliers. Dieter and Ahrens (1979)† show that the way around the Marsaglia hyperplane problem is to use compound multiplicative congruential generators of the form

$$r_i = (ar_{i-1} + br_{i-2}) \pmod{m}$$

which will increase the number of hyperplanes by a factor $2^{(t/d)}$ provided the constants a and b are chosen carefully. The hyperplanes do not go away but their number may be increased arbitrarily by adding more terms as above.

5.2.6. Good pseudo-random generators. On a computer with integer length t bits, the best simple multiplicative generator is probably that proposed by Ahrens *et al* (1970), where the multiplier is

$$a = 2^{t-2}\,\tfrac{1}{2}(\sqrt{5}-1).$$

(You may recognise the famous 'golden section' constant here.) In practice, the constant a is determined for a given value of the integer length t by multiplying 2^{t-2} into a very precise value of the golden section constant ($=0{\cdot}618\,033\,988\,749\,894\,848\,204\,5868$) and rounding to the nearest integer congruent to 5 (mod 8). This will yield a generator with period 2^{t-2} and good distribution properties.

On CDC 6000, 7000 and Cyber machines, it is unfortunately not easy to take advantage of the full 60-bit words, since integer multiplication is performed only on 48 bits (for compatibility with floating-point numbers which have 48-bit mantissas). For such computers, the value $t=48$ is therefore appropriate, and the constant a is

$$a = (1170\,673\,633\,457\,725)_8 = (43\,490\,275\,647\,445)_{10}$$

which has a period of $2^{46} = 70\,368\,744\,177\,664$.

† We are grateful to the authors of this book for providing a pre-publication version of the first seven chapters.

On IBM 370 and IBM-compatible computers, the 32-bit integer arithmetic makes simple generators somewhat risky for large calculations. With only 31 significant bits available, the maximum period is 2^{29} or about 500 million. Since it is dangerous to come anywhere close to exhausting the period (exhausting the period would give a perfectly uniform distribution since all numbers would be generated) it is not too difficult to imagine calculations where a better generator is needed. In this case I recommend using the McGill University package 'Super-duper' (available from Professor G Marsaglia, School of Computer Science, McGill University, PO Box 6070, Montreal, Canada). The basic generator of this package combines two methods to give a period as long as one would expect from a 64-bit machine.

5.2.7. Machine-independent pseudo-generators. It is sometimes convenient to have a random number generator which produces exactly the same numbers on any computer. Assuming that we want floating-point numbers between zero and one, we therefore choose the precision of the lowest-precision machine we are likely to use and simulate that precision on other computers. (On computers with longer words, the lower bits will be zero.) Such a generator will, in general, not be optimal on any machine, either in terms of period or of speed, but we will show here that it can be implemented, in FORTRAN, on most larger computers. It can then be used to test programs and compare and continue calculations across changes of computer.

If we choose IBM 32-bit words as our minimum precision, such a generator, called RN32 (CERN Program Library†), has been implemented as follows. As default starting integer use the value 65 539. Multiply the previous (or starting) integer ('seed') by 69 069. Keep only the lower 31 bits of the result. This 31-bit integer becomes the seed for the next number. We get a floating-point pseudo-random number from the seed by masking off the lower 8 bits to assure exact floating-point representation of the integer, floating it, and multiplying the result by the exact floating representation of 2^{-31}.

Differences in FORTRAN and floating-point representations require slightly different implementation on different machines. We show on p1174 as examples the CDC and IBM versions.

With the default seed shown, the first two numbers produced by these generators are approximately‡:

$$R1 = 0{\cdot}107\ 915\ 04\ldots$$

$$R2 = 0{\cdot}587\ 475\ 06\ldots.$$

5.2.8. Practical computing considerations. The usage of random number generators from FORTRAN programs requires some special considerations of a practical nature. Perhaps the most important of these stems from the fact that most pseudo-random generators, like the one above, are coded as FORTRAN functions rather than subroutines. Strictly speaking, this is not in accordance with the rules of FORTRAN, since random number generators are not functions of their arguments only, they

† Programs in this library are made generally available. Further information may be obtained from: Program Library, Division DD, CERN, 1211 Geneva 23, Switzerland.

‡ The numbers produced by different computers are exactly the same if represented as binary fractions, but the exact decimal representation requires many more digits than we reproduce here and more than your computer is likely to give in a printout.

```
      FUNCTION RN32(IDUMMY)
C                                  CDC VERSION,  F.JAMES, 1978
C                IY IS THE SEED, CONS=2**-31
      DATA IY/65539/
      DATA CONS /1661400000000000000B/
      DATA MASK31/17777777777B/
      IY = IY * 69069
C         KEEP ONLY LOWER 31 BITS
      IY = IY .AND. MASK31
C         SET LOWER 8 BITS TO ZERO TO ASSURE EXACT FLOAT
      JY = IY .AND. 07777777777777777400B
      YFL = JY
      RN32 = YFL*CONS
      RETURN
C         ENTRY TO INPUT SEED
      ENTRY RN32IN
      IY = IDUMMY
      RETURN
C         ENTRY TO OUTPUT SEED
      ENTRY RN32OT
      IDUMMY = IY
      RETURN
      END


      FUNCTION RN32(DUMMY)
C                                  IBM VERSION,  F.JAMES, 1978
C                IY IS THE SEED, CONS=2**-31
      DATA IY/65539/
      DATA CONS/Z39200000/
      IY = IY * 65539
C         ASSURE LEFTMOST BIT ZERO (POSITIVE INTEGER)
      IF (IY .GT. 0)  GO TO 6
      IY = IY + 2147483647 + 1
    6 CONTINUE
C         SET LOWER 8 BITS TO ZERO TO ASSURE EXACT FLOAT
      JY = (IY/256)*256
      YFL = JY
      RN32 = YFL*CONS
      RETURN
C         ENTRY TO INPUT SEED
      ENTRY RN32IN(IX)
      IY = IX
      RETURN
C         ENTRY TO OUTPUT SEED
      ENTRY RN32OT(IX)
      IX = IY
      RETURN
      END
```

have 'side effects', namely they set up the next number. Since they are functions, the FORTRAN compilers reserve the right to optimise them out of existence by replacing each function evaluation by the constant value of the function. For example

$$X = \text{RANDOM}(1) + \text{RANDOM}(1)$$

could be compiled as if it were

$$X = 2{\cdot}0 * \text{RANDOM}(1)$$

which is of course not the same thing at all. The well-known way around this is to do something like

$$X = \text{RANDOM}(I) + \text{RANDOM}(I+1)$$

in order to fool the compiler into thinking the two calls have different arguments and must therefore be called twice. Similar problems may arise when calls to random number generators appear in DO loops. Of course, the proper way around this is to use random number generators coded as subroutines rather than as functions. This may be somewhat clumsier to use, but is much safer.

In many applications, the actual time taken to generate the random numbers may be important. In earlier days this was usually the case, and it is still a point of great pride among programmers to chop half a microsecond off the generation time, even though it may be quite negligible compared with the rest of the calculation. In cases where generation time is important, several tricks may be used.

One is, of course, to code the generator in assembler, which is often done anyway since the operations needed may be easier to code in assembler. Even better is to code the generator 'in-line' in the calling program to avoid the overhead of a subroutine call, which is usually the greater part of the time spent in getting a random number. The standard CDC FORTRAN function RANF causes the compiler to produce fast in-line code. Although the multiplier used by RANF is not the best, the generous effective word-length of 48 bits still produces random numbers good enough for most applications, so I would advise CDC users to call RANF whenever speed of generation is an important consideration.

Often a calculation requires n-tuples of random numbers, in which case it is much more efficient to use a subroutine that returns n random numbers at a time rather than calling a single generator n times, because of the overhead in the call. As an example, the CERN Program Library subroutine NRAN (V105) generates n random numbers in one call on the CDC 7600 about seven times as fast as n calls to RNDM (V104), for large n, even though the two routines use exactly the same method of generation (with different multipliers for 'independence').

Sometimes it is desirable to have exactly the same sequence of random numbers in one calculation as you had in the previous calculation, and sometimes it is equally important that the sequence be different. Many generators therefore offer different ways of initiating the sequence. Most generators use a default starting value (like RN32 above) and therefore always produce the same sequence unless requested otherwise. Such generators often allow inputting and outputting the seed value, so that at the end of a run the current seed value can be output and read back in at the beginning of the next run to continue the sequence (this is the case with RN32). In this way, different sequences can be forced by inputting different starting seeds. Still other generators use 'random' starting seeds obtained by using the time of day and date from the system clock and transforming that into an appropriate integer. This removes all control from the user and even adds some element of truly random unpredictability.

6. Quasi-Monte Carlo

The theoretical difficulties and practical success of pseudo-random numbers have given rise to another type of sequence known as quasi-random. (In English usage, 'pseudo-' means false, and 'quasi-' means almost, but in the technical context of random numbers their meanings are somewhat different and much more precise.) Quasi-random sequences are not even intended to appear random but only to give the right answer to the problem at hand. Thus they are more satisfactory since they are not based on an illusion, but on the other hand they must in principle be tailored to the problem at hand. Since this problem can often be reduced to multiple integration, the tailoring becomes ready-to-wear in practice and the theory applicable to most cases.

6.1. The quasi-random philosophy

The concept of quasi-random numbers arises from the realisation that the mathematical randomness of pseudo-random numbers is neither attainable in theory nor necessary in practice, and it is more meaningful to assure that the 'random'

sequence has the necessary properties to produce the desired result. For example, in multiple integration and in most simulation studies, each multidimensional point or simulated event is considered independently of the others and the order in which they appear is immaterial. That is, correlations between successive points (events) is usually of no importance—this aspect of randomness can safely be abandoned for most calculations. Another aspect which can be abandoned is the degree of fluctuation about uniformity for certain distributions—in many cases a *super-uniform* distribution is, in fact, more desirable than a truly random distribution with uniform probability density.

Since we have now dropped all pretense of randomness, the reader may object at this point to retaining the name Monte Carlo. Strictly speaking he is right, but it is probably more justified to enlarge the concept of Monte Carlo to include the use of quasi-random sequences. Quasi-Monte Carlo is indeed rather a downward (in dimensionality) extension of Monte Carlo than an upward extension of one-dimensional quadrature, since it retains some fundamental properties of Monte Carlo such as applicability to spaces of very high dimensionality, performance nearly independent of dimensionality, very small growth rate, even for high dimensionalities, and robustness with respect to the continuity properties of the function. In addition, the theory of quasi-Monte Carlo outlined below is much closer to that of true Monte Carlo than to that of quadrature.

6.2. The theoretical basis of quasi-Monte Carlo

6.2.1. The discrepancy of a point set. Let us here introduce a measure of non-uniformity valid for any dimensionality, called *discrepancy* (see Weyl (1916), or secondary references Zaremba (1968, 1972) or Stroud (1971)). Consider the unit hypercube in d dimensions, with each coordinate of $\boldsymbol{x}$ varying from zero to one, and we are given a set of n points, the ith point having coordinates $\boldsymbol{x}_i$. The function $v(\boldsymbol{x})$ gives the integrated number of points, from the origin to the point $\boldsymbol{x}$ (the empirical distribution function). The corresponding volume from the origin to the point $\boldsymbol{x}$ is just given by the product of the coordinates of the point $\boldsymbol{x}$, and the local discrepancy g at $\boldsymbol{x}$ is defined as the difference between the number of points in this volume and the expected number based on the volume:

$$g(\boldsymbol{x}) = \frac{v(\boldsymbol{x})}{n} - x_1 x_2 \ldots x_d.$$

One can then define various measures of global discrepancy by taking different norms of the function g. The most common are the *extreme discrepancy* given by the maximum of the absolute value of g for all $\boldsymbol{x}$, and the *mean square discrepancy* given by the integral of the square of g over all $\boldsymbol{x}$. The general term 'discrepancy' is sometimes loosely applied also to the global measures.

Since we will use discrepancy to *test the hypothesis* of uniformity of a point distribution, it is not surprising that this measure is already well-known to statisticians, who will recognise extreme discrepancy as the Kolmogorov statistic and mean-square discrepancy as the Smirnov–Cramer–Von Mises statistic for testing compatibility of distributions (see Eadie *et al* 1971, pp268–70).

If the extreme discrepancy of a point set approaches zero as the number of points approaches infinity, the (infinite) set of points is said to be *uniform*. We refer to this as uniformity in the sense of Weyl, to distinguish it from more common meanings

of the word. A truly random point set in a finite-dimensional space can be shown to be uniform in this sense. In quasi-Monte Carlo we will use non-random points which are also uniform (for infinite sets) or which have low discrepancy (for finite sets).

6.2.2. The convergence of quasi-Monte Carlo integration. The theorems given in this subsection concern the approximation of a multidimensional definite integral by an unweighted sum of function values over a set of points. The function to be integrated will be assumed to be of finite *variation.* A precise definition of variation is not very enlightening and is beyond the scope of this article (see Zaremba 1968, Stroud 1971); we give here only a rough idea sufficient for an understanding of the results presented below. The variation in quasi-Monte Carlo theory plays the role of variance in true Monte Carlo, being also a measure of the non-constancy of the function. For a differentiable function of d variables, the variation can be thought of as an average of the absolute values of the dth mixed partial derivatives. Integrable functions of interest to physicists (with at most a finite number of discontinuities) have a finite variation.

The following theorems form the mathematical basis for integration by quasi-Monte Carlo.

(i) (Weyl 1916). If a definite integral is estimated by an unweighted sum of function values over a set of points, the estimate will converge to the true value of the integral as the number of points approaches infinity if and only if the point set is uniform in the sense of Weyl. This theorem is the equivalent of the law of large numbers for true Monte Carlo, and gives the conditions under which the quasi-Monte Carlo estimate is consistent.

(ii) (Hlawka, see Zaremba 1968). If a definite integral is approximated by an unweighted sum of function values over a finite set of points, the resulting error will be bounded by the product of the discrepancy of the point set and the variation of the function.

(iii) (Roth and others, see Kuipers and Niederreiter 1974, Zaremba 1968). The discrepancy of a point set cannot be made smaller than a certain value, which depends on the number of points n and the dimensionality d. Attempts to find point sets which achieve this fundamental lower limit have been successful only in a small number of cases.

(iv) (Korobov, see Stroud 1971). The discrepancy of the first n points of an infinite point set cannot decrease as a function of n any faster than $1/n$ for large n.

The second theorem above implies that the estimate of the integral will converge to the correct answer as fast as the discrepancy of the point set converges to zero, and the fourth theorem gives us hope that this could be as fast as $1/n$, compared with the much slower square root of n for true Monte Carlo. Unfortunately, it is not generally known how to generate points which attain the lower discrepancy bound, but one can at least generate points with considerably lower discrepancy than the expectation of a truly random set.

After reading the above theorems, we should not be surprised to learn that the expected value of the extreme discrepancy of a set of n truly random points decreases with n like $1/\sqrt{n}$ for large n in any number of dimensions.

6.3. Quasi-random number generators

Because theorem (iv) of the last subsection applies only to infinite sequences,

we must distinguish here between finite quasi-random sequences of n numbers where n is fixed in advance, and the first n numbers of an infinite sequence. The latter will clearly be more convenient to use since it can be extended if necessary, but the above theorems indicate that we might be able to get a better discrepancy if we fix n.

6.3.1. Good lattice points. Optimal points for function integration are generated by fixing n (and the dimensionality d) and actually minimising the extreme discrepancy of the n points with respect to their positions. The computational complexity of such a calculation being overwhelming, only an approximate minimum-discrepancy solution can be found for anything but a very small point set. A considerable amount of theoretical work has been done on d-dimensional lattices (Kuipers and Niederreiter 1974, Zaremba 1972) but this approach has not yet produced techniques of great interest for large calculations, except for the Korobov sequences described below.

6.3.2. Finite Korobov sequences. Korobov considered sets of points restricted to belong to certain families characterised by different expressions for the coordinates, with each expression containing some free parameters. The values of these parameters were then optimised by requiring a minimum extreme discrepancy. Probably the most successful Korobov family is the parallelepiped lattice, where successive points $\boldsymbol{x}$ are given by:

$$\boldsymbol{x}_k = \left.\frac{ak}{N}\right|_{\text{mod } 1}, \left.\frac{bk}{N}\right|_{\text{mod } 1}, \ldots, \left.\frac{dk}{N}\right|_{\text{mod } 1}, \qquad k = 1, N$$

where $a, b, \ldots, d$ are coefficients to be determined in order to optimise the discrepancy for the given value of the number of points N and the dimensionality. Discussion of Korobov sequences and references to the original Russian articles can be found in Stroud (1971) and Zakrzewska *et al* (1978). The latter article describes a program for multiple integration using Korobov sequences. These sequences can also be used as an option in DIVONNE2 (Friedman 1977a), and extensive tables of optimal coefficients for generating Korobov sequences are given in Keast (1972).

6.3.3. The Richtmyer generator. This generator is the equivalent of the Korobov parallelepiped family described just above, but for infinite N and 'any' d. Since one can no longer optimise the coefficients, it is apparently sufficient to use 'irrational' numbers in order to avoid a short period. Since truly irrational numbers cannot be represented in computers, it has been suggested to use the square roots of the first few prime numbers. Thus one gets the simple formula for the jth coordinate of the ith quasi-random point:

$$x_{ij} = iS_j, \text{ mod } 1$$

where S_j is the square root of the jth prime number.

In theory this generator is supposed to have very good properties for an infinite number of points, and its discrepancy should decrease like $1/n$ for very large n. The problem is then to make it behave well for small n (which may still be very large in practice) without destroying the asymptotic behaviour. This is done, first of all, by observing the two-dimensional distributions of the first few thousand numbers of two of the coordinates. When a pair is seen to be badly distributed, one of the

corresponding S values is dropped from the table and replaced by a higher root prime. Of course, this observed distribution would in principle improve with larger n, but one does not know how large, so it is better in practice to be careful.

The second method for improvement of short-term behaviour of such quasi-random generators is the *shuffling* technique, which assures that all the numbers from the generator will be used, but not quite in the order in which they are generated. Usually another (pseudo-)random generator is used for the shuffling, which is performed using a buffer (usually 10 or 20 words per dimension), and selecting the next quasi-random number pseudo-randomly from the buffer of the appropriate coordinate, filling the used location in the buffer with the next quasi-random number in the corresponding sequence. This yields points different from those of the unshuffled generator but preserves the super-uniform distribution of each of the coordinate values.

6.3.4. The van der Corput generator. The formula of van der Corput corresponds to expressing the integers in a system of base P, reversing the digits, putting a point in front, and interpreting the resulting sequence as fractions in the base P. P is any prime, so the ith coordinate is generated using this formula with P being the ith prime number. For example, for $P=2$ this gives the results shown in the table below.

Decimal integer	Binary integer	Binary fraction	Decimal fraction
$j=1$	1	0·1	0·5
2	10	0·01	0·25
3	11	0·11	0·75
4	100	0·001	0·125
5	101	0·101	0·625
6	110	0·011	0·375
7	111	0·111	0·875
8	1000	0·0001	0·0625

This generator has properties similar to those of the Richtmyer generator, except that it seems to behave much better for smaller n. In spite of the apparent computational complexity, it can be made fast, thanks to a relatively simple algorithm for implementing it, due to Halton (1960).

As with the Richtmyer generator, this method can be improved by shuffling. A particularly effective scrambling technique, based on explicit minimisation of the discrepancy for this generator, is given by Braaten and Weller (1979).

7. Non-uniform random numbers

Up to now we have been almost exclusively concerned with uniformly distributed random numbers, either with uniformly distributed probability of occurrence or for quasi-random sets, a distribution as uniform as possible (sometimes called 'super-uniform', since it is more uniform than a truly random set with uniform probability density). In this section we discuss the problem of generating random numbers such that the probability of obtaining a number in a given range is not uniform but follows some other distribution.

Generating non-uniform distributions is very important in many applications, where the physical phenomena being simulated are known to follow certain other distributions. The most important of these are the Gaussian (or normal) and exponential distributions for continuous variables, and the Poisson and binomial distributions for discrete variables. Many other distributions may be required for special applications, and many different techniques are known for generating them. We present here only a brief review of the most important methods with some indication of where to look for more. It is assumed throughout this section that an appropriate generator of uniformly distributed random numbers is available for use in generating the non-uniform distributions.

7.1. Gaussian generators

The Gaussian distribution is one of the most important in statistical and physical calculations and also one of the richest in terms of different methods proposed for generating random numbers.

7.1.1. Using the central limit theorem. This method has already been described above in §2.6.1. It is not exact, although it may be good enough for many purposes, and the absence of points in the extreme tails may even be desirable in some cases. It is also not especially fast, but may be faster than some other methods when a good generator of arrays of uniform numbers is available. (Note that this method, like most of those given in this section, must not make use of a quasi-random uniform number generator, since serial correlations in the uniform generator lead to distortions in the distribution of the output random numbers.)

As a word of warning, I should point out an interesting mistake sometimes made in connection with this generator. It arises from the realisation that the central limit theorem of course works for differences as well as sums, so that taking the sum of six uniform numbers minus the sum of six other uniform numbers would be as good as taking the sum of twelve uniform numbers and subtracting six. Some clever people decide therefore to use twelve uniform numbers to generate *two* random Gaussian deviates, once using a sum and once with differences. It is certainly true that this gives two (approximately) Gaussian numbers, but they are unfortunately highly correlated. Correlation has also been the source of some concern about the simple generator of §2.6.1, since any correlations in the uniform generator would produce deviations from the Gaussian distribution of the sum.

7.1.2. The transformation method. Since the Gaussian probability function cannot be integrated in terms of the usually available functions, it is not straightforward to apply a transformation from uniform to Gaussian-distributed variables. There is, however, a clever method of transforming two independent uniform variables u and v into two independent Gaussian variables x and y:

$$x=(-2\ln u)^{1/2}\cos(2\pi v)$$

$$y=(-2\ln u)^{1/2}\sin(2\pi v).$$

This method is exact and easy to program but is not quite as fast as it may appear, since it requires calculation of a logarithm, square root, sine and cosine, all of which are reasonably time-consuming operations.

An improvement on the above method is the polar method of Marsaglia:

(i) Generate uniform random numbers u and v.

(ii) Calculate $w=(2u-1)^2+(2v-1)^2$.

(iii) If $w>1$, go back to (i).

(iv) Return $x=uz$ and $y=vz$, where $z=(-2\ln w/w)^{1/2}$.

This variation eliminates the sine and cosine at the slight expense of $\simeq 21\%$ rejection in step (iii) and a few more arithmetic operations.

7.1.3. The Forsythe–Von Neumann method. This is an ingenious method for generating random numbers in any distribution of the form:

$$f(x)=c\exp[-G(x)] \qquad 0<G(x)<1,\ a<x<b$$

based on the fact that if you:

(i) choose u_0 uniformly between a and b;

(ii) calculate $t=G(u_0)$;

(iii) generate uniformly $u_1, u_2, \ldots, u_k$, $0<u_i<1$ where k is determined by the condition:

$$t>u_1>u_2>\ldots u_{k-1}<u_k$$

then the probability that k is odd is $P(t)=e^{-t}$.

Therefore, whenever k is even, reject that value of u_0 and go back to (i). When k is odd, accept that u_0 as a member of a sample from f. Unfortunately, the fact that the range of G must be from zero to one requires some fiddling to use this technique for generating from the Gaussian distribution, but some good methods are based on it (see Ahrens and Dieter 1973).

7.1.4. Compound methods. Many other techniques have been proposed for generating Gaussian random numbers, and the best (fastest exact) methods are composed by combining several of these techniques. The general idea is to use a fast approximate method most of the time, and then with a carefully calculated (small) probability, one draws from a 'corrective' distribution which just makes up for the approximation in the first technique. In addition, different regions under the Gaussian curve are attacked using different techniques, with the region first being chosen using an auxiliary random number. Such methods are often somewhat complicated to program and require a table of constants used to choose regions, methods, corrections, etc. A detailed account of a good compound method is given in Dieter and Ahrens (1973), and a summary of many good methods, both simple and compound for the Gaussian distribution, is given in Ahrens and Dieter (1972).

7.1.5. Generating correlated Gaussians. The above subsections deal only with the generation of one-dimensional Gaussians, which can be used directly for multi-dimensional Gaussian distributions only when the different variables (dimensions) are uncorrelated (i.e. when the covariance matrix is diagonal). For the general case of multidimensional Gaussian variables with a general covariance matrix V, uncorrelated standard Gaussian variables may be used when transformed as indicated here. Let $\boldsymbol{z}$ be a standard normal random vector (i.e. independent Gaussian-distributed components with zero mean and unit variance), then a unique lower-triangular matrix C exists such that

$$\boldsymbol{x}=C\boldsymbol{z}+\boldsymbol{m}$$

and $(\boldsymbol{x}-\boldsymbol{m})$ has the covariance matrix

$$V = CC'$$

where C' is the transpose of C.

Given V, the matrix C can be calculated by using the following recursive formulae (the 'square root' method):

$$c_{i1} = v_{i1}/\sqrt{v_{11}} \qquad 1 \leqslant i \leqslant m$$

$$c_{ii} = \left| v_{ii} - \sum_{k=1}^{i-1} c_{ik}^2 \right|^{1/2} \qquad 1 < i \leqslant m$$

$$c_{ij} = \frac{v_{ij} - \sum_{k=1}^{i-1} c_{ik}c_{jk}}{c_{jj}} \qquad 1 < j < i \leqslant m.$$

In practice, one usually wants a large set of random vectors all generated with the same covariance matrix V, so the matrix C is computed once at the beginning of the program and then used each time a random Gaussian vector is wanted.

7.2. All other known distributions

A vast number of transformations, tricks and formulae are known for generating random numbers according to different distributions. For example, given two uniform numbers, their sum is distributed according to a triangular distribution, and the largest of the two is distributed like $\sqrt{u}$. An extraordinarily complete and very dense collection of such techniques is given in Everett and Cashwell (1972).

7.3. Empirical distributions

It often happens that one wants to generate random numbers distributed according to some probability density f which is not any of the usual distributions but may, for example, have been determined empirically from measurements on a particular complex system.

7.3.1. The rejection (hit-or-miss) method. One can, of course, always use the hit-or-miss method if the probability density f is bounded and its upper bound is known. In this method, one simply chooses points randomly and uniformly in the space, using the function value at each point (divided by the maximum function value) as the probability of accepting the point. A point is then accepted if and only if $f/f_{\max}$ is greater than a uniform random number chosen between zero and one. This well-known technique becomes very inefficient when the variance of f is large, in which case nearly all the points are rejected. For this reason it is usually better to use one of the methods described below.

7.3.2. Distribution given as histogram. A distribution in the form of a histogram is usually represented as a vector of frequencies, where the first value is the relative frequency of points desired in the first bin, etc. These frequencies must first be normalised so that their sum is unity, then it is usually convenient to form the cumulative distribution, where the ith number in the cumulative distribution vector is the sum from one to i of the numbers in the corresponding density vector. (The last number in the cumulative vector is therefore always equal to one.) To generate a

random number according to the histogram, one first generates a uniform number u_0 and then looks for the first position in the cumulative distribution vector where the value is greater than u_0. This is the bin in which that random number should be generated. It may of course be very inefficient to do this search sequentially (at least for long vectors), and a better method would be to do it by a binary search technique. (The CERN library program HISRAN uses this method.)

A still faster method, although much more complicated, is that of the Marsaglia tables, described in Ahrens and Dieter (1972).

7.3.3. Distribution given as function. To randomly sample according to a one-dimensional distribution given as a smooth function, the usual technique is first to determine the percentiles of this distribution, i.e. the points on the independent variable axis where the integral of the function takes on given values (called percentiles because they are chosen so that the integral over each interval is a given percentage, often 1%, of the total). This is the inversion of the cumulative distribution function. The result of this relatively time-consuming operation is a set of x values which can then be used to generate random numbers very rapidly by direct interpolation in the table of x now considered as a function of F. The CERN library program FUNRAN uses this method with four-point polynomial interpolation in a table of 100 values.

7.3.4. Multidimensional distributions. Multidimensional distributions given as histograms may, of course, be treated exactly as for one dimension. However, when the desired distribution is given as a smooth function, the method outlined above cannot be extended in a straightforward manner, and would anyway require multidimensional tables and multidimensional interpolation, which either consume considerable time and space or are quite inaccurate, especially when the function involved has a large variance.

The problem of randomly sampling a space of high-dimensionality is closely related to that of multidimensional integration, so it is reasonable to look at integration methods for indications on how to proceed. Indeed the recursive partitioning method of Friedman (1977b) is directly applicable and DIVONNE2 (Friedman 1977a) has as an option the generation of points according to the function. This is because the aim of the partitioning algorithm is to delimit regions in which the function variance is small, after which one can efficiently apply hit-or-miss generation or simply produce weighted points.

8. Applications

In Monte Carlo calculation, the step from theoretical understanding to correct results is often far from trivial. Unlike analytical calculations where gross errors usually produce results which are obviously absurd, subtle bugs in Monte Carlo 'reasoning' easily give rise to answers which are completely wrong but still appear sufficiently reasonable to go unnoticed. If only for this reason, it is indispensable to consider a few examples, particularly those which illustrate the most notorious traps for the unwary.

8.1. The uncertainty of a weighted average

The results given here can be derived easily from the definitions of mean and

variance but are included here because they are of such central importance in real calculations. We suppose that (as is the usual case) the result of our calculation is an average over a set of terms which we will call weights w_i. We further assume that this average is Gaussian-distributed in accordance with the central limit theorem and wish to determine the standard deviation of this distribution. In order to estimate the average and its standard distribution it is necessary to accumulate:

(i) the sum of the weights, W;
(ii) the sum of the squares of the weights, Q;
(iii) the total number of entries, N.

Then it follows from §2 that the best estimate of the average is just W/N and that the standard deviation of this is $D=(1/N)(Q-W^2/N)^{1/2}$.

For the important case when most of the weights are zero (for example, for one bin of a histogram when most of the events go into other bins), the second term under the square root in the expression for D is negligible compared with the first term and the result is simplified considerably.

In the other limit, when all weights are equal (and non-zero), the two terms under the square root cancel and the standard deviation is of course zero. In practice it may not appear to be zero because of rounding error in the computer, which is especially serious for this particular calculation. For this reason, the sums Q and W should be accumulated in double precision, and it is necessary to test that rounding has not caused the argument of the square root to become negative.

8.2. Integration over a triangle

One of the fundamental advantages of the Monte Carlo method is the ability to easily handle problems with awkward integration regions (inter-dependent integration limits). However, as this example shows, there are a variety of different ways to handle these problems and not all of them are correct.

Consider the integration of the function g over the two-dimensional region specified as

$$I=\int_{x=0}^{1}\int_{y=0}^{x} g(x, y)\,\mathrm{d}y\,\mathrm{d}x.$$

We give four ways of estimating this integral by Monte Carlo.

8.2.1. The obvious way.

(*a*) Choose a random number x_i between zero and one.
(*b*) Choose another random number y_i between zero and x_i.
(*c*) Take the sum of $g(x_i, y_i)$ repeating steps (*a*) and (*b*).

A simple graphical representation of this method shows that it gives the wrong answer. While it is true that this procedure would yield points only in the allowed region (the lower triangle in figure 4), it would give the same expected number of points along each vertical line in the figure, producing a much higher density of points on the left-hand side than on the right.

8.2.2. The rejection method.

(*a*) Choose a random number x_i between zero and one.
(*b*) Choose another random number y_i also between zero and one.
(*c*) If $y_i > x_i$, reject the point and return to (*a*).
(*d*) Accumulate the sum of $g(x_i, y_i)$ for the remaining points.

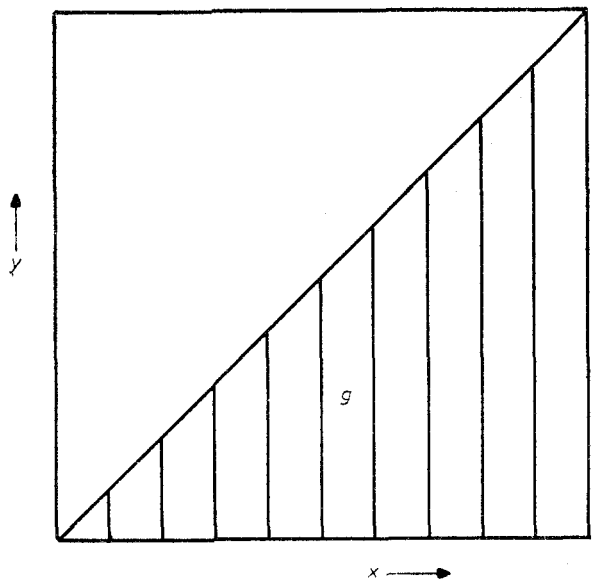

Figure 4. A triangular integration region.

This method, although correct, has the disadvantage of using only half the points generated, i.e. it is equivalent to integrating over the whole square, but considering the function to be zero on the upper triangle.

8.2.3. The folding method (a trick).

(*a*) Choose *two* independent random numbers r_1 and r_2, each between zero and one.
(*b*) Set x_i = larger of (r_1, r_2).
(*c*) Set y_i = smaller of (r_1, r_2).
(*d*) Sum up $g(x_i, y_i)$ as before.

This method is equivalent to choosing points r over the whole square, then folding the square about the diagonal so that all points x, y fall in the lower triangle. It is clear that this gives a constant point density without any rejection, and is therefore correct and more efficient than the rejection method.

8.2.4. The weighting method.

(*a*) Choose a random number x_i between zero and one.
(*b*) Choose another random number y_i between zero and x_i.
(*c*) Take the sum of $2x_i g(x_i, y_i)$, repeating the steps above.

In this method, the points are chosen 'incorrectly' as in the obvious method, but the bias is corrected by applying the weighting function which happens to be just $2x$ in this case. This method may or may not be more efficient than folding, depending on the function g. In particular it will be more efficient whenever the variance of xg is smaller than the variance of g. If nothing is known *a priori* about g, it is usual to avoid weighting if possible.

8.3. Programs for real-life calculations

At this point the reader should already be convinced that the possibilities for

undetected gross errors in Monte Carlo calculations are numerous. Of course, there is nothing special about Monte Carlo in this respect; complex systems lead to complex calculations and errors can be made on many levels, from the logical understanding of the problem and method all the way down to typing errors in the programs and data. In fact, the Monte Carlo method offers unique opportunities to verify the results of complicated calculations, especially in the case of simulations.

The basic principle is to output not only the number you are interested in but also as many other intermediate and accessory results as possible, especially those for which you know in advance what answer to expect. Even if you are only interested in the global average of some quantity, print out a histogram of the quantity as a function of some other interesting quantity. This generally costs little or nothing extra in a big calculation, and may give considerable insight into the system being studied (if the expected distribution is not known in advance) or allow a powerful check of the correctness of the computation (if the expected distribution is known). I find it convenient to use a general histogramming package such as the generally available HBOOK (CERN Program Library) which allows one to look at an entire one- or two-dimensional empirical distribution in very readable format with only two or three simple lines of FORTRAN. The quantities which you should look at will of course depend on the problem, but a general rule is to examine the quantity of interest in one more dimension than is required, if possible.

8.4. Splitting and killing in sequential simulations

In this subsection we consider simulation calculations in which each 'event' (member of the hypothetical population) consists of a sequence of elementary interactions. Examples of such calculations would be:

(i) Simulation of the traffic flow in a city, where elementary interactions would be car turning left, turning right, parking, breaking down, having an accident, etc.

(ii) Simulation of neutrons or charged particles traversing matter, where elementary interactions would be scattering, decay, absorption, etc.

In these calculations it may be necessary to assign to each elementary interaction a weight proportional to the probability of that interaction. The weight of an entire event is then the product of the weights of its component interactions and the final results of the simulation will be averages over these total weights. As we have seen, the uncertainties of these averages are minimised when the weights are equal. The efficiency of the calculation can therefore be improved by using the following techniques for reducing the variance of the weight distribution.

(i) *Splitting*. After each elementary interaction, compare the accumulated product of weights with the average product at that point for the other events. If it is significantly greater than the average, split the event into two (or more) events from that point on, each one having half (or less) of the above-mentioned accumulated product. In practice, this may be complicated to implement using programming languages which do not explicitly support recursiveness.

(ii) *Killing*. Compare the accumulated product as above, and if it is significantly *less* than the average, either kill (reject) the whole event before finishing it, or continue with the weight increased to the average. The probability of killing the event should be $1-r$, where r is the ratio of the accumulated product weight to the average accumulated product at that point.

It should be clear that it is of no use to apply the killing technique after the entire

event has been generated, but only during intermediate steps to avoid the rest of the calculation. Splitting may be performed after the entire event has been generated if this is more convenient, but the decision to split should be made on the basis of the accumulated product weight at the point at which the event is to be split.

8.5. Multiparticle phase space

One of the richest areas of Monte Carlo calculations has been the integration of the relativistic phase space of multiparticle reactions in high energy and nuclear physics. For a reaction with k outgoing particles, the phase-space volume element is basically the $3k$-dimensional momentum space element, but the true dimensionality is reduced to $3k-4$ by a four-dimensional delta function expressing the conservation of energy and momentum. Whenever k is greater than four or five, the complexity of these integrals becomes overwhelming and they can only be performed by numerical techniques, usually only by Monte Carlo. Unfortunately, this interesting problem is much too vast to be treated here and we will merely point to the most important references on the subject.

(i) The classic work on the subject is the monograph of Hagedorn (1964).

(ii) A more recent and extensive treatment, also much more oriented toward practical Monte Carlo calculations, is the book of Byckling and Kajantie (1973).

(iii) The most recent techniques for enriching the region of low momentum transfer are summarised in the review article of Carey and Drijard (1978), which could be considered as an update to the book of Byckling and Kajantie. The techniques reviewed in this paper are very important since one finds, in practice, that in high-energy collisions only a small part of phase space is actually populated, namely that corresponding to peripheral or low-momentum-transfer events.

8.6. Sampling from a finite population

In many fields, particularly in astronomy, plasma physics, fluid dynamics, etc, it is a common problem to simulate the behaviour of a large but finite number of objects (stars, electrons, molecules, etc) which interact with one another. A typical step in such a simulation is the calculation of the force or potential at one object by summing the contributions due to all the other objects. Although the number of objects is finite it may be so large that it is not possible to perform the entire sum, and some approximation must then be made using a smaller sample of objects. Three possible approaches are:

(i) *A fixed-point rule.* Based on some additional knowledge of the physics or the geometry of the problem, it may be possible to average over some fixed set of points. Such a formula would be highly problem-dependent, and the uncertainty of the result would depend on the distributions involved, perhaps in a very complicated way.

(ii) *Random sampling with replacement.* In this method, objects are chosen randomly and one does not 'remember' which objects were already chosen, so that some may be taken more than once. The population thus becomes infinite, and the theory developed earlier applies just as if it were any other Monte Carlo calculation: the uncertainty on the potential is the standard deviation of the individual contributions, divided by the square root of the sample size.

(iii) *Sampling without replacement.* This method resembles (ii), except that

one explicitly avoids taking the contribution from any one object more than once. The final convergence must be better than (ii), since one eventually reaches zero error when all contributions have been taken, but since by definition we cannot consider all contributions, it is the convergence rate in the early part of the sequence that matters. This convergence rate starts out equal to that of (ii), only improving slowly as the number of contributions taken becomes a significant fraction of the total. The price paid for this small improvement is having to remember which contributions were already chosen. Also the improvement may not be usable if it is too hard to calculate.

References

Ahrens J H and Dieter U 1972 *Commun. ACM* **15** 873–82
—— 1973 *Math. Comp.* **27** 927–37
Ahrens J H, Dieter U and Grube A 1970 *Computing* **6** 121–38
Braaten E and Weller G 1979 *J. Comp. Phys.* **33** 249–58
Buffon 1777 *Essai d'arithmetique morale*
Byckling E and Kajantie K 1973 *Particle Kinematics* (New York: Wiley)
Carey D and Drijard D 1978 *J. Comp. Phys.* **28** 327–56
Dieter U 1971 *Math. Comp.* **25** 855–83
Dieter U and Ahrens J H 1971 *Numer. Math.* **17** 101–23
—— 1973 *Computing* **11** 137–46
—— 1979 *Pseudo-Random Numbers* (New York: Wiley)
Eadie W T, Drijard D, James F E, Roos M and Sadoulet B 1971 *Statistical Methods in Experimental Physics* (Amsterdam: North-Holland)
Everett C J and Cashwell E D 1972 *A Monte Carlo Sampler. Los Alamos Scientific Laboratory Informal Rep.* LA-5061-MS
Friedman J 1977a *DIVONNE2, a program for multiple integration and adaptive importance sampling. SLAC Computation Research Group Tech. Memo* CGTM No 188
—— 1977b *Trans. Math. Software* submitted
Frigerio N A and Clark N 1975 *Trans. Am. Nucl. Soc.* **22** 283–4
Frigerio N A, Clark N and Tyler S 1978 *Toward Truly Random Numbers. Argonne National Laboratory Rep.* ANL/ES-26 Part 4
Genz A 1972 *Comp. Phys. Commun.* **4** 11–5
Haber S 1970 *Siam Rev.* **12** 4
Hagedorn R 1964 *Relativistic Kinematics* (New York: Benjamin)
Halton J H 1960 *Numer. Math.* **2** 84–90
—— 1970 *Siam Rev.* **12** 1–63
Hammersley J M and Handscomb D C 1964 *Monte Carlo Methods* (London: Methuen)
Kahaner D K and Wells B 1979 *ACM Trans. Math. Software* **5** 86–96
Keast P 1972 *Department of Computer Science, University of Toronto, Toronto, Canada. Tech. Rep.* 40
Knuth D 1969 *The Art of Computer Programming* vol 2 (Reading, Mass.: Addison-Wesley) pp1–160
Kuipers L and Niederreiter H 1974 *Uniform Distribution of Sequences* (New York: Wiley)
Lyness J N and Kaganove J J 1976 *ACM Trans. Math. Software* **2** 65–81
Malcolm M A and Simpson R B 1975 *ACM Trans. Math. Software* **1** 141–6
Marsaglia G 1968 *Proc. Nat. Acad. Sci.* **61** 25–8
—— 1972 *Applications of Number Theory to Numerical Analysis* (New York: Academic) pp249–85
Sobol I M 1979 *Siam J. Numer. Anal.* **16** 790–3 (Erratum **16** 1080)
Stroud A H 1971 *Approximate Calculation of Multiple Integrals* (Englewood Cliffs, NJ: Prentice-Hall)

Stroud A H and Secrest D 1966 *Gaussian Quadrature Formulas* (Englewood Cliffs, NJ: Prentice-Hall)
Tsuda T 1973 *Numer. Math.* **20** 377–91
van Dooren P and de Ridder L1976 *J. Comp. Appl. Math.* **2** 207–10
Weyl H 1916 *Math. Ann.* **77** 313–52
Zakrzewska K, Dudek J and Nazarewicz N 1978 *Comp. Phys. Commun.* **14** 299–309
Zaremba S K 1968 *Siam. Rev.* **10** 303–14
Zaremba S K (ed) 1972 *Applications of Number Theory to Numerical Analysis* (*Proc. Symp. Centre for Research in Mathematics, University of Montreal, 9–14 September 1971*) (New York: Academic)