30. MONTE CARLO TECHNIQUES

Revised July 1995 by S. Youssef (SCRI, Florida State University).

Monte Carlo techniques are often the only practical way to evaluate difficult integrals or to sample random variables governed by complicated probability density functions. Here we describe an assortment of methods for sampling some commonly occurring probability density functions.

30.1. Sampling the uniform distribution

Most Monte Carlo sampling or integration techniques assume a "random number generator" which generates uniform statistically independent values on the half open interval $[0, 1)$. Although such a generator is, strictly speaking, impossible on a finite digital computer, generators are nevertheless available which pass extensive batteries of tests for statistical independence and which have periods which are so long that, for practical purposes, values from these generators can be considered to be uniform and statistically independent. In particular, the lagged-Fibonacci based generator introduced by Marsaglia, Zaman, and Tsang [1] is efficient, has a period of approximately 10^{43}, produces identical sequences on a wide variety of computers and, passes the extensive "DIEHARD" battery of tests [2]. Many commonly available congruential generators fail these tests and often have sequences (typically with periods less than 2^{32}) which can be easily exhausted on modern computers and should therefore be avoided [3].

30.2. Inverse transform method

If the desired probability density function is $f(x)$ on the range $-\infty < x < \infty$, its cumulative distribution function (expressing the probability that $x \leq a$) is given by Eq. (28.1). If a is chosen with probability density $f(a)$, then the integrated probability up to point a, $F(a)$, is itself a random variable which will occur with uniform probability density on $[0, 1]$. If x can take on any value, and ignoring the endpoints, we can then find a unique x chosen from the p.d.f. $f(s)$ for a given u if we set

$$u = F(x) \ , \tag{30.1}$$

provided we can find an inverse of F, defined by

$$x = F^{-1}(u) \ . \tag{30.2}$$

This method is shown in Fig. 30.1a.

For a discrete distribution, $F(x)$ will have a discontinuous jump of size $f(x_k)$ at each allowed $x_k, k = 1, 2, \cdots$. Choose u from a uniform distribution on (0,1) as before. Find x_k such that

$$F(x_{k-1}) < u \leq F(x_k) \equiv \text{Prob}\ (x \leq x_k) = \sum_{i=1}^{k} f(x_i) \ ; \tag{30.3}$$

then x_k is the value we seek (note: $F(x_0) \equiv 0$). This algorithm is illustrated in Fig. 30.1b.

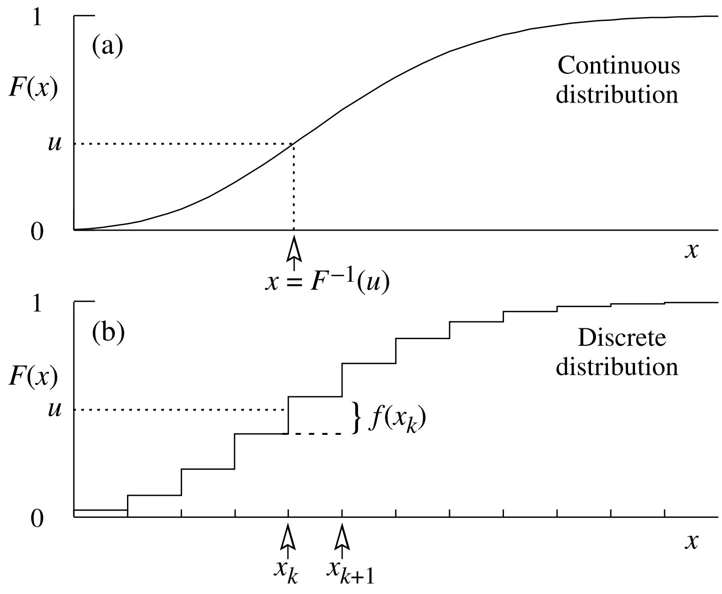

Figure 30.1: Use of a random number u chosen from a uniform distribution (0,1) to find a random number x from a distribution with cumulative distribution function $F(x)$.

30.3. Acceptance-rejection method (Von Neumann)

Very commonly an analytic form for $F(x)$ is unknown or too complex to work with, so that obtaining an inverse as in Eq. (30.2) is impractical. We suppose that for any given value of x the probability density function $f(x)$ can be computed and further that enough is known about $f(x)$ that we can enclose it entirely inside a shape which is C times an easily generated distribution $h(x)$ as illustrated in Fig. 30.2.

Frequently $h(x)$ is uniform or is a normalized sum of uniform distributions. Note that both $f(x)$ and $h(x)$ must be normalized to unit area and therefore the proportionality constant $C > 1$. To generate $f(x)$, first generate a candidate x according to $h(x)$. Calculate $f(x)$ and the height of the envelope $C\,h(x)$; generate u and test if $uC\,h(x) \leq f(x)$. If so, accept x; if not reject x and try again. If we regard x and $uC\,h(x)$ as the abscissa and ordinate of a point in a two-dimensional plot, these points will populate the entire area $C\,h(x)$ in a smooth manner; then we accept those which fall under $f(x)$. The efficiency is the ratio of areas, which must equal $1/C$; therefore we must keep C as close as possible to 1.0. Therefore we try to choose $C\,h(x)$ to be as close to $f(x)$ as convenience dictates, as in the lower part of Fig. 30.2. This practice is called importance sampling, because we generate more trial values of x in the region where $f(x)$ is most important.

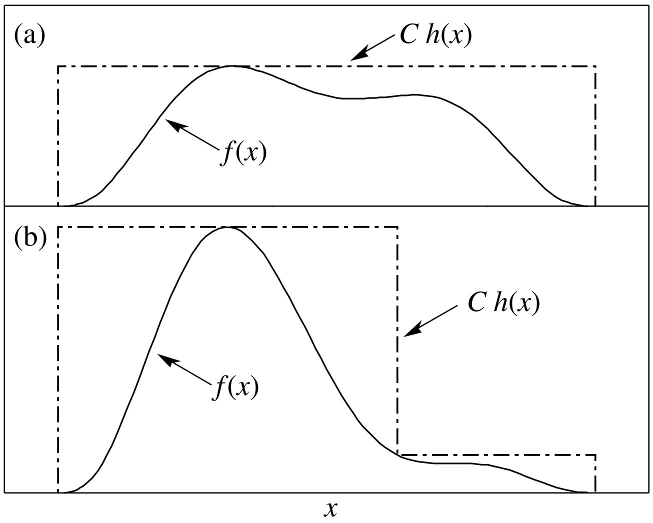

Figure 30.2: Illustration of the acceptance-rejection method. Random points are chosen inside the upper bounding figure, and rejected if the ordinate exceeds $f(x)$. Lower figure illustrates importance sampling.

30.4. Algorithms

Algorithms for generating random numbers belonging to many different distributions are given by Press [4], Ahrens and Dieter [5], Rubinstein [6], Everett and Cashwell [7], Devroye [8], and Walck [9]. For many distributions alternative algorithms exist, varying in complexity, speed, and accuracy. For time-critical applications, these algorithms may be coded in-line to remove the significant overhead often encountered in making function calls. Variables named "u" are assumed to be independent and uniform on (0,1).

In the examples given below, we use the notation for the variables and parameters given in Table 28.1.

30.4.1. *Sine and cosine of random angle*:

Generate u_1 and u_2. Then $v_1 = 2u_1 - 1$ is uniform on $(-1,1)$, and $v_2 = u_2$ is uniform on (0,1). Calculate $r^2 = v_1^2 + v_2^2$. If $r^2 > 1$, start over. Otherwise, the sine (S) and cosine (C) of a random angle are given by

$$S = 2v_1v_2/r^2 \quad \text{and} \quad C = (v_1^2 - v_2^2)/r^2 \ . \tag{30.4}$$

30.4.2. *Gaussian distribution*:

If u_1 and u_2 are uniform on (0,1), then

$$z_1 = \sin 2\pi u_1 \sqrt{-2\ln u_2} \quad \text{and} \quad z_2 = \cos 2\pi u_1 \sqrt{-2\ln u_2} \tag{30.5}$$

are independent and Gaussian distributed with mean 0 and $\sigma = 1$.

There are many faster variants of this basic algorithm. For example, construct $v_1 = 2u_1 - 1$ and $v_2 = 2u_2 - 1$, which are uniform on $(-1,1)$. Calculate $r^2 = v_1^2 + v_2^2$, and if $r^2 > 1$ start over. If $r^2 < 1$, it is uniform on (0,1). Then

$$z_1 = v_1 \sqrt{\frac{-2\ln r^2}{r^2}} \quad \text{and} \quad z_2 = v_2 \sqrt{\frac{-2\ln r^2}{r^2}} \tag{30.6}$$

are independent numbers chosen from a normal distribution with mean 0 and variance 1. $z_i' = \mu + \sigma z_i$ distributes with mean μ and variance σ^2.

For a multivariate Gaussian, see the algorithm in Ref. 10.

30.4.3. $\chi^2(n)$ distribution:

For n even, generate $n/2$ uniform numbers u_i; then

$$y = -2\ln\left(\prod_{i=1}^{n/2} u_i\right) \quad \text{is} \quad \chi^2(n)\ . \tag{30.7}$$

For n odd, generate $(n-1)/2$ uniform numbers u_i and one Gaussian z as in Sec. 30.4.2; then

$$y = -2\ln\left(\prod_{i=1}^{(n-1)/2} u_i\right) + z^2 \quad \text{is} \quad \chi^2(n)\ . \tag{30.8}$$

For $n \gtrsim 30$ the much faster Gaussian approximation for the χ^2 may be preferable: generate z as in Sec. 30.4.2 and use $y = \left[z + \sqrt{2n-1}\right]^2/2$; if $z < -\sqrt{2n-1}$ reject and start over.

30.4.4. *Gamma distribution*:

All of the following algorithms are given for $\lambda = 1$. For $\lambda \neq 1$, divide the resulting random number x by λ.

- If $k = 1$ (the *exponential* distribution), accept $x = -(\ln u)$.
- If $0 < k < 1$, initialize with $v_1 = (e + k)/e$ (with $e = 2.71828...$ being the natural log base). Generate u_1, u_2. Define $v_2 = v_1 u_1$.

Case 1: $v_2 \leq 1$. Define $x = v_2^{1/k}$. If $u_2 \leq e^{-x}$, accept x and stop, else restart by generating new u_1, u_2.
Case 2: $v_2 > 1$. Define $x = -\ln([v_1 - v_2]/k)$. If $u_2 \leq x^{k-1}$, accept x and stop, else restart by generating new u_1, u_2. Note that, for $k < 1$, the probability density has a pole at $x = 0$, so that return values of zero due to underflow must be accepted or otherwise dealt with.

- Otherwise, if $k > 1$, initialize with $c = 3k - 0.75$. Generate u_1 and compute $v_1 = u_1(1 - u_1)$ and $v_2 = (u_1 - 0.5)\sqrt{c/v_1}$. If $x = k + v_2 - 1 \leq 0$, go back and generate new u_1; otherwise generate u_2 and compute $v_3 = 64v_1^3u_2^2$. If $v_3 \leq 1 - 2v_2^2/x$ **or** if $\ln v_3 \leq 2\{[k-1]\ln[x/(k-1)] - v_2\}$, accept x and stop; otherwise go back and generate new u_1.

30.4.5. *Binomial distribution*:

If $p \leq 1/2$, iterate until a successful choice is made: begin with $k = 1$; compute $P_k = q^n$ [for $k \neq 1$ use $P_k \equiv f(r_k; n, p)$, and store P_k into B; generate u. If $u \leq B$ accept $r_k = k - 1$ and stop; otherwise increment k by 1 and compute next P_k and add to B; generate a new u and repeat. If we arrive at $k = n + 1$, stop and accept $r_{n+1} = n$. If $p > 1/2$ it will be more efficient to generate r from $f(r; n, q)$, *i.e.*, with p and q interchanged, and then set $r_k = n - r$.

30.4.6. *Poisson distribution*:

Iterate until a successful choice is made: Begin with $k = 1$ and set $A = 1$ to start. Generate u. Replace A with uA; if now $A < \exp(-\mu)$, where μ is the Poisson parameter, accept $n_k = k - 1$ and stop. Otherwise increment k by 1, generate a new u and repeat, always starting with the value of A left from the previous try. For large $\mu(\gtrsim 10)$ it may be satisfactory (and much faster) to approximate the Poisson distribution by a Gaussian distribution (see our Probability chapter, Sec. 28.3.3) and generate z from $f(z;0,1)$; then accept $x = \max(0, [\mu + z\sqrt{\mu} + 0.5])$ where [ ] signifies the greatest integer $\leq$ the expression.

30.4.7. *Student's t* distribution:

For $n > 0$ degrees of freedom (n not necessarily integer), generate x from a Gaussian with mean 0 and $\sigma^2 = 1$ according to the method of 30.4.2. Next generate y, an independent gamma random variate with $k = n/2$ degrees of freedom. Then $z = x\sqrt{2n}/\sqrt{y}$ is distributed as a t with n degrees of freedom.

For the special case $n = 1$, the Breit-Wigner distribution, generate u_1 and u_2; set $v_1 = 2u_1 - 1$ and $v_2 = 2u_2 - 1$. If $v_1^2 + v_2^2 \leq 1$ accept $z = v_1/v_2$ as a Breit-Wigner distribution with unit area, center at 0.0, and FWHM 2.0. Otherwise start over. For center M_0 and FWHM Γ, use $W = z\Gamma/2 + M_0$.

References:

1. G. Marsaglia, A. Zaman, and W.W. Tsang, *Towards a Universal Random Number Generator*, Supercomputer Computations Research Institute, Florida State

University technical report FSU-SCRI-87-50 (1987). This generator is available as the CERNLIB routine RANMAR.

2. Much of DIEHARD is described in: G. Marsaglia, *A Current View of Random Number Generators*, keynote address, *Computer Science and Statistics: 16th Symposium on the Interface*, Elsevier (1985).
3. New generators with periods even longer than the lagged-Fibonacci based generator are described in G. Marsaglia and A. Zaman, *Some Portable Very-Long-Period Random Number Generators*, Compt. Phys. **8**, 117 (1994). The Numerical Recipes generator **ran2** [W.H. Press and S.A. Teukolsky, *Portable Random Number Generators*, Compt. Phys. **6**, 521 (1992)] is also known to pass the DIEHARD tests.
4. W.H. Press *et al.*, *Numerical Recipes* (Cambridge University Press, New York, 1986).
5. J.H. Ahrens and U. Dieter, Computing **12**, 223 (1974).
6. R.Y. Rubinstein, *Simulation and the Monte Carlo Method* (John Wiley and Sons, Inc., New York, 1981).
7. C.J. Everett and E.D. Cashwell, *A Third Monte Carlo Sampler*, Los Alamos report LA-9721-MS (1983).
8. L. Devroye, *Non-Uniform Random Variate Generation* (Springer-Verlag, New York, 1986).
9. Ch. Walck, *Random Number Generation*, University of Stockholm Physics Department Report 1987-10-20 (Vers. 3.0).
10. F. James, Rept. on Prog. in Phys. **43**, 1145 (1980).