Principles of Scientific Computing
Monte Carlo methods

Jonathan Goodman

last revised April 20, 2006

Monte Carlo means using random numbers in scientific computing. More precisely, it means using random numbers as a tool to compute something that is not random. For example[1], let X be a random variable and write its expected value as $A = E[X]$. If we can generate $X_1, \ldots, X_n$, n independent random variables with the same distribution, then we can make the approximation

$$A \approx \widehat{A}_n = \frac{1}{n} \sum_{k=1}^{n} X_k \ .$$

The *strong law of large numbers* states that $\widehat{A}_n \to A$ as $n \to \infty$. The X_k and $\widehat{A}_n$ are random and (depending on the seed, see Section 2) could be different each time we run the program. Still, the target number, A, is not random.

We emphasize this point by distinguishing between Monte Carlo and *simulation*. Simulation means producing random variables with a certain distribution just to look at them. For example, we might have a model of a random process that produces clouds. We could simulate the model to generate cloud pictures, either out of scientific interest or for computer graphics. As soon as we start asking quantitative questions about, say, the average size of a cloud or the probability that it will rain, we move from pure simulation to Monte Carlo.

The reason for this distinction is that there may be other ways to define A that make it easier to estimate. This process is called *variance reduction*, since most of the error in $\widehat{A}$ is *statistical*. Reducing the variance of $\widehat{A}$ reduces the statistical error.

We often have a choice between Monte Carlo and deterministic methods. For example, if X is a one dimensional random variable with probability density $f(x)$, we can estimate $E[X]$ using a panel integration method, see Section **??**. This probably would be more accurate than Monte Carlo because the Monte Carlo error is roughly proportional to $1/\sqrt{n}$ for large n, which gives it order of accuracy roughly $\frac{1}{2}$. The worst panel method given in Section **??** is first order accurate. The general rule is that deterministic methods are better than Monte Carlo in any situation where the determinist method is practical.

We are driven to resort to Monte Carlo by the "curse of dimensionality". The curse is that the work to solve a problem in many dimensions may grow exponentially with the dimension. Suppose, for example, that we want to compute an integral over ten variables, an integration in ten dimensional space. If we approximate the integral using twenty points in each coordinate direction, the total number of integration points is $20^{10} \approx 10^{13}$, which is on the edge of what a computer can do in a day. A Monte Carlo computation might reach the same accuracy with only, say, 10^6 points. People often say that the number of points needed for a given accuracy in Monte Carlo does not depend on the dimension, and there is some truth to this.

One favorable feature of Monte Carlo is that it is possible to estimate the order of magnitude of statistical error, which is the dominant error in most Monte Carlo computations. These estimates are often called "error bars" because of

[1] Section **??** has a quick review of the probability we use here.

the way they are indicated on plots of Monte Carlo results. Monte Carlo error bars are essentially statistical confidence intervals. Monte Carlo practitioners are among the avid consumers of statistical analysis techniques.

Another feature of Monte Carlo that makes academics happy is that simple clever ideas can lead to enormous practical improvements in efficiency and accuracy (which are basically the same thing). This is the main reason I emphasize so strongly that, while A is given, the algorithm for estimating it is not. The search for more accurate alternative algorithms is often called "variance reduction". Common variance reduction techniques are importance sampling, antithetic variates, and control variates.

Many of the examples below are somewhat artificial because I have chosen not to explain specific applications. The techniques and issues raised here in the context of toy problems are the main technical points in many real applications. In many cases, we will start with the probabilistic definition of A, while in practice, finding this is part of the problem. There are some examples in later sections of choosing alternate definitions of A to improve the Monte Carlo calculation.

1 Quick review of probability

This is a quick review of the parts of probability needed for the Monte Carlo material discussed here. Please skim it to see what notation we are using and check Section 6 for references if something is unfamiliar.

Probability theory begins with the assumption that there are *probabilities* associated to *events*. Event B has probability $\Pr(B)$, which is a number between 0 and 1 describing what fraction of the time event B would happen if we could repeat the *experiment* many times. The exact meaning of probabilities is debated at length elsewhere.

An event is a set of possible outcomes. The set of all possible outcomes is called Ω and particular outcomes are called ω. Thus, an event is a subset of the set of all possible outcomes, $B \subseteq \Omega$. For example, suppose the experiment is to toss four coins (a penny, a nickle, a dime, and a quarter) on a table and record whether they were face up (*heads*) or face down (*tails*). There are 16 possible outcomes. The notation $THTT$ means that the penny was face down (tails), the nickle was up, and the dime and quareter were down. The event "all heads" consists of a single outcome, $HHHH$. The event $B =$ "more heads than tails" consists of the five outcomes: $B = \{HHHH, THHH, HTHH, HHTH, HHHT\}$.

The basic set operations apply to events. For example the intersection of events B and c is the set of outcomes both in B and in C: $\omega \in B \cap C$ means $\omega \in B$ and $\omega \in C$. For that reason, $B \cap C$ represents the event "B and C". For example, if B is the event "more heads than tails" above and C is the event "then dime was heads", then C has 8 outcomes in it, and $B \cap C = \{HHHH, THHH, HTHH, HHHT\}$. The set union, $B \cup C$ is $\omega \in B \cup C$ if $\omega \in B$ or $\omega \in C$, so we call it "B or C". One of the axioms of probability is

$$\Pr(B \cup C) = \Pr(B) + \Pr(C) \quad , \quad \text{if} \quad B \cap C \text{ is empty.}$$

Another axiom is that

$$0 \le \Pr(B) \le 1 \ , \ \text{ for any event } B.$$

These have many intuitive consequences, such as

$$B \subset C \Longrightarrow \Pr(B) \le \Pr(C) \ .$$

A final axiom is $\Pr(\Omega) = 1$; for sure something will happen. This really means that Ω includes every possible outcome.

The *conditional* "probability of B given C" is given by Bayes' formula:

$$\Pr(B \mid C) = \frac{\Pr(B \cap C)}{\Pr(C)} = \frac{\Pr(B \text{ and } C)}{\Pr(C)} \tag{1}$$

Intuitively, this is the probability that a C outcome also is a B outcome. If we do an experiment and $\omega \in C$, $\Pr(B \mid C)$ is the probability that $\omega \in B$. The right side of (1) represents the fraction of C outcomes that also are in B. We often know $\Pr(C)$ and $\Pr(B \mid C)$, and use (1) to calculate $\Pr(B \cap C)$. Of course, $\Pr(C \mid C) = 1$. Events B and C are *independent* if $\Pr(B) = \Pr(B \mid C)$, which is the same as $\Pr(B \mid C) = \Pr(B)\Pr(C)$, so B being independent of C is the same as C being independent of B.

The *probability space* Ω is finite if it is possible to make a finite list[2] of all the outcomes in Ω: $\Omega = \{\omega_1, \omega_2, \ldots, \omega_n\}$. The space is *countable* if it is possible to make a possibly infinite list of all the elements of Ω: $\Omega = \{\omega_1, \omega_2, \ldots, \omega_n, \ldots\}$. We call Ω *discrete* in both cases. When Ω is discrete, we can specify the probabilities of each outcome

$$f_k = \Pr(\omega = \omega_k) \ .$$

Then an event B has probability

$$\Pr(B) = \sum_{\omega_k \in B} \Pr(\omega_k) = \sum_{\omega_k \in B} f_k \ .$$

A discrete random variable[3] is a number, $X(\omega)$, that depends on the random outcome, ω. In the coin tossing example, $X(\omega)$ could be the number of heads. The *expected value* is (defining $x_k = X(\omega_k)$)

$$E[X] = \sum_{\omega \in \Omega} X(\omega)\Pr(\omega) = \sum_{\omega_k} x_k f_k \ .$$

The *probability distribution* of a *continuous* random variable is described by a *probability density function*, or *PDF*, $f(x)$. If $X \in R^n$ is an n component random vector and $B \subseteq R^n$ is an event, then

$$\Pr(B) = \int_{x \in B} f(x)dx \ .$$

[2]Warning: this list is impractically large even in common simple applications.

[3]Warning: sometimes ω is the random variable and X is a function of a random variable.

In one dimension, we also have the *cumulative distribution function*, or[4] *CDF*: $F(x) = \Pr(X \leq x) = \int_{-\infty}^{x} f(x)dx$. For $a < b$ we have $\Pr(a \leq x \leq b) = F(b) - F(a)$. Also $f(x) = \frac{d}{dx}F(x)$, so we may write, informally,

$$\Pr(x \leq X \leq x + dx) = F(x + dx) - F(x) = f(x)dx \ . \tag{2}$$

The expected value is

$$\mu = E[X] = \int_{R^n} xf(x)dx \ .$$

in more than one dimension, both sides are n component vectors with components

$$\mu_k = E[X_k] = \int_{R^n} x_k f(x)dx \ .$$

In one dimension, the *variance* is

$$\sigma^2 = \mathrm{var}(X) = E\left[(X - \mu)^2\right] = \int_{-\infty}^{\infty} (x - \mu)^2 f(x)dx \ .$$

It is helpful not to distinguish between discrete and continuous random variables in general discussions. For example, we write $E[X]$ for the expected value in either case. The probability distribution, or probability *law*, of X is its probability density in the continuous case or the discrete probabilities in the discrete case. We write $X \sim f$ to indicate that f is the probability density, or the discrete probabilities, of X. We also write $X \sim X'$ to say that X and X' have the same probability distribution. If X and X' also are independent, we say they are iid, for *independent and identically distributed*. The goal of a simple sampler (Section 3 below) is generating a sequence $X_k \sim X$. We call these *samples* of the distribution X.

In more than one dimension, there is the symmetric $n \times n$ *variance/covariance* matrix (usually called *covariance matrix*),

$$C = E\left[(X - \mu)(X - \mu)^*\right] = \int_{R^n} (x - \mu)(x - \mu)^* f(x)dx \ , \tag{3}$$

whose entries are the individual covariances

$$C_{jk} = E\left[(X_j - \mu_j)(X_k - \mu_k)\right] = \mathrm{cov}\left[X_j, X_k\right] \ .$$

The covariance matrix is positive semidefinite in the sense that for any $y \in R^n$, $y^*Cy \geq 0$. This follows from (3):

$$\begin{aligned} y^*Cy &= y^* \ \left(E\left[(X - \mu)(X - \mu)^*\right]\right) \ y \\ &= E\left[\left(y^*(X - \mu)\right)\left((X - \mu)^* y\right)\right] \\ &= E\left[W^2\right] \geq 0 \ , \end{aligned}$$

[4] A common convention is to use capital letters for random variables and the corresponding lower case letter to represent values of that variable.

where $W = y^*(X - \mu)$. In order for C not to be positive definite, there must be a $y \neq 0$ that has $E\left[W^2\right] = 0$, which means that[5] W is identically zero. This means that the random vector X always lies on the hyperplane in R^n defined by $y^*x = y^*\mu$.

Let $Z = (X, Y)$ be an $n + m$ dimensional random variable with X being the first n components and Y being the last m, with probability density $f(z) = f(x, y)$. The *marginal* density for X is $g(x) = \int_{y \in R^m} f(x, y)dy$, and the marginal for Y is $h(y) = \int_x f(x, y)dx$. The random variables X and Y are *independent* if $f(x, y) = g(x)h(y)$. This is the same as saying that any event depending only on X is independent of any event depending only on[6] Y. The *conditional probability density* is the probability density for X alone once the value of Y is known:

$$f(x \mid y) = \frac{f(x, y)}{\int_{x' \in R^n} f(x', y)dx'} \ . \tag{4}$$

For $n = m = 1$ the informal version of this is the same as Bayes' rule (1). If B is the event $x \leq X \leq x + dx$ and C is the event $y \leq Y \leq y + dy$, then

$$\begin{aligned} f(x \mid y)dx &= \Pr(B \mid Y = y) \\ &= \Pr(B \mid y \leq Y \leq y + dy) \\ &= \frac{\Pr(B \text{ and } y \leq Y \leq y + dy)}{\Pr(y \leq Y \leq y + dy)} \end{aligned}$$

The numerator in the last line is $f(x, y)dxdy$ and the denominator is $\int_{x'} f(x', y)dx'dy$, which gives (4).

Suppose X is an n component random variable, $Y = u(X)$, and $g(y)$ is the probability density of Y. We can compute the expected value of Y in two ways

$$E[Y] = \int_y yg(y)dy = E[u(X)] = \int_x u(x)f(x)dx \ .$$

The one dimensional informal version is simple if u is one to one (like e^x or $1/x$, but not x^2). Then $y + dy$ corresponds to $u(x + dx) = u(x) + u'(x)dx$, so

$$\begin{aligned} g(y)dy &= Pr(y \leq Y \leq y + dy) \\ &= Pr(y \leq u(X) \leq y + dy \\ &= Pr(x \leq X \leq x + dx) \\ &= f(x)dx \quad \text{where } dx = \frac{dy}{u'(x)} \\ &= \frac{f(x)}{u'(x)}dy \ . \end{aligned}$$

[5] If W is a random variable with $E\left[W^2\right] = 0$, then the $\Pr(W \neq 0) = 0$. Probabilists say that $W = 0$ *almost surely* to distinnguish between events that don't exist (like $W^2 = -1$) and events that merely never happen.

[6] If $B \subseteq R^n$ and $C \subseteq R^m$, then $\Pr(X \in B \text{ and } Y \in C) = \Pr(X \in B)\Pr(Y \in C)$.

This shows that if $y = u(x)$, then $g(y) = f(x)/u'(x)$.

Three common continuous random variables are *uniform*, *exponential*, and *gaussian* (or *normal*). In each case there is a *standard* version and a general version that is easy to express in terms of the standard version. The standard uniform random variable, U, has probability density.

$$f(u) = \begin{cases} 1 & \text{if } 0 \le u \le 1 \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

Because the density is constant, U is equally likely to be anywhere within the unit interval, $[0, 1]$. From this we can create the general random variable uniformly distributed in the interval $[a, b]$ by $Y = (b - a)U + a$. The PDF for Y is

$$g(y) = \begin{cases} \frac{1}{b-a} & \text{if } a \le y \le b \\ 0 & \text{otherwise.} \end{cases}$$

The exponential random variable, T, with *rate constant* $\lambda > 0$, has PDF

$$f(t) = \begin{cases} \lambda e^{-\lambda t} & \text{if } 0 \le t \\ 0 & \text{if } t < 0. \end{cases} \tag{6}$$

The exponential is a model for the amount of time something (e.g. a light bulb) will work before it breaks. It is characterized by the *Markov property*, if it has not broken by time t, it is as good as new. Let λ characterize the probability density for breaking right away, which means $\lambda dt = \Pr(0 \le T \le dt)$. The random time T is exponential if all the conditional probabilities are equal to this:

$$\Pr(t \le T \le t + dt \mid T \ge t) = \Pr(T \le dt) = \lambda dt \ .$$

Using Bayes' rule (1), and the observation that $\Pr(T \le t + dt \text{ and } T \ge t) = f(t)dt$, this becomes

$$\lambda dt = \frac{f(t)dt}{1 - F(t)} \ ,$$

which implies $\lambda\,(1 - F(t)) = f(t)$, and, by differentiating, $-\lambda f(t) = f'(t)$. This gives $f(t) = Ce^{-\lambda t}$ for $t > 0$. We find $C = \lambda$ using $\int_0^\infty f(t)dt = 1$. Independent exponential *inter arrival* times generate the *Poisson arrival processes*. Let T_k, for $k = 1, 2, \ldots$ be independent exponentials with rate λ. The k^{th} *arrival time* is

$$S_k = \sum_{j \le k} T_j \ .$$

The expected number of arrivals in interval $[t_1, t_2]$ is $\lambda(t_2 - t_1)$ and all arrivals are independent. This is a fairly good model for the arrivals of telephone calls at a large phone bank.

We denote the *standard normal* by Z. The standard normal has PDF

$$f(z) = \frac{1}{\sqrt{2\pi}} e^{-z^2/2} \ . \tag{7}$$

The general normal with mean μ and variance σ^2 is given by $X = \sigma Z + \mu$ and has PDF

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(x-\mu)^2/2\sigma^2} \ . \tag{8}$$

We write $X \sim \mathcal{N}(\mu, \sigma^2)$ in this case. A standard normal has distribution $\mathcal{N}(0,1)$. An n component random variable, X, is a *multivariate normal* with mean μ and covariance C it has probability density

$$f(x) = \frac{1}{Z} \exp\left((x-\mu)^* H (x-\mu)/2\right) \ , \tag{9}$$

where $H = C^{-1}$ and the *normalization constant* is (we don't use this) $Z = 1/\sqrt{(2\pi)^n \det(C)}$. The reader should check that this is the same as (8) when $n = 1$.

The class of multivariate normals has the *linear transformation property*. Suppose L is an $m \times n$ matrix with rank m (L is a linear transformation from R^n to R^m that is onto R^m). If X is an n dimensional multivariate normal, then Y is an m dimensional multivariate normal. The covariance matrix for Y is given by

$$C_Y = L C_X L^* \ . \tag{10}$$

We derive this, taking $\mu = 0$ without loss of generality (why?), as follows:

$$C_Y = E\left[YY^*\right] = E\left[(LX)\left((LX)^*\right]\right] = E\left[LXX^*L^*\right] = LE\left[XX^*\right]L^* = LC_X L^* \ .$$

The two important theorems for simple Monte Carlo are the *law of large numbers* and the *central limit theorem*. Suppose $A = E[X]$ and X_k for $k = 1, 2, \ldots$ is an iid sequence of samples of X. The approximation of A is

$$\widehat{A}_n = \frac{1}{n} \sum_{k=1}^{n} X_k \ .$$

The error is $R_n = \widehat{A}_n - A$. The law of large numbers states[7] that $\widehat{A}_n \to A$ as $n \to \infty$. Statisticians call estimators with this property *consistent*. The central limit theorem states that if $\sigma^2 = \mathrm{var}(X)$, then $R_n \approx \mathcal{N}(0, \sigma^2/n)$. It is easy to see that

$$E\left[R_n\right] = E\left[\widehat{A}_n\right] - A = 0 \ ,$$

and that (recall that A is not random)

$$\mathrm{var}(R_n) = \mathrm{var}(\widehat{A}_n) = \frac{1}{n}\mathrm{var}(X) \ .$$

The first property makes the estimator *unbiased*, the second follows from independence of the X_k. The deep part of the central limit theorem is that for large n, R_n is approximately normal, regardless of the distribution of X (as long as $E\left[X^2\right] < \infty$).

[7] The *Kolmogorov strong law* of large numbers is the theorem that $\lim_{n\to\infty} \widehat{A}_n = A$ almost surely, i.e. that the probability of the limit not existing or being the wrong answer is zero. More useful for us is the *weak law*, which states that, for any $\epsilon > 0$, $P(|R_n| > \epsilon) \to 0$ as $n \to \infty$. These are closely related but not the same.

2 Random number generators

The random variables used in Monte Carlo are generated by a (pseudo) random number generator. The procedure `double rng()` is a perfect random number generator if

```
for( k=0; k<n; k++ )  U[k] = rng();
```

produces an array of iid standard uniform random variables. The best available random number generators are perfect in this sense for nearly all practical purposes. The native C/C++ procedure `random()` is good enough for most Monte Carlo (I use it).

Bad ones, such as the native `rand()` in C/C++ and the procedure in *Numerical Recipies* give incorrect results in common simple cases. If there is a random number generator of unknown origin being passed from person to person in the office, do not use it (without a condom).

The computer itself is not random. A pseudo random number generator simulates randomness without actually being random. The *seed* is a collection of m integer variables: `int seed[m];`. Assuming standard C/C++ 32 bit integers, the number of bits in the seed is $32 \cdot m$. There is a *seed update* function $\Phi(s)$ and an *output* function $\Psi(s)$. The update function produces a new seed: $s' = \Phi(s)$. The output function produces a floating point number (check the precision) $u = \Psi(s) \in [0, 1]$. One call `u = rng();` has the effect

$$s \longleftarrow \Phi(s) \; ; \quad \texttt{return } u = \Psi(s) \; ; \; .$$

The random number generator should come with procedures `s = getSeed();` and `setSeed(s);`, with obvious functions. Most random number generators set the initial seed to the value of the system clock as a default if the program has no `setSeed(s);` command. We use `setSeed(s)` and `getSeed()` for two things. If the program starts with `setSeed(s);`, then the sequence of seeds and "random" numbers will be the same each run. This is helpful in debugging and reproducing results across platforms. The other use is *checkpointing*. Some Monte Carlo runs take so long that there is a real chance the computer will crash during the run. We avoid losing too much work by storing the state of the computation to disk every so often. If the machine crashes, we restart from the most recent checkpoint. The random number generator seed is part of the checkpoint data.

The simplest random number generators use linear congruences. The seed represents an integer in the range $0 \leq s < c$ and Φ is the linear congruence (a and b positive integers) $s' = \Phi(s) = (as + b)_{\text{mod } c}$. If $c > 2^{32}$, then we need more than one 32 bit integer variable to store s. Both `rand()` and `random()` are of this type, but `rand()` has $m = 1$ and `random()` has $m = 4$. The output is $u = \Psi(s) = s/c$. The more sophisticated random number generators are of a similar computational complexity.

3 Sampling

A *simple sampler* is a procedure that produces an independent sample of X each time it is called. The job of a simple sampler is to turn iid standard uniforms into samples of some other random variable. A large Monte Carlo computation may spend most of its time in the sampler and it often is possible to improve the performance by paying attention to the details of the algorithm and coding. Monte Carlo practitioners often are amply rewarded for time stpnt tuning their samplers.

3.1 Bernoulli coin tossing

A *Bernoulli* random variable with parameter p, or a *coin toss*, is a random variable, X, with $\Pr(X = 1) = p$ and $\Pr(X = 0) = 1 - p$. If U is a standard uniform, then $p = \Pr(U \leq p)$. Therefore we can sample X using the code fragment

```
X=0; if ( rng() <= p) X=1;
```

Similarly, we can sample a random variable with finitely many values $\Pr(X = x_k) = p_k$ (with $\sum_k p_k = 1$) by dividing the unit interval into disjoint sub intervals of length p_k. This is all you need, for example, to simulate a simple random walk or a finite state space Markov chain.

3.2 Exponential

If U is standard uniform, then

$$T = \frac{-1}{\lambda} \ln(U) \tag{11}$$

is an exponential with rate parameter λ. Before checking this, note first that $U > 0$ so $\ln(U)$ is defined, and $U < 1$ so $\ln(U)$ is negative and $T > 0$. Next, since λ is a rate, it has units of 1/Time, so (11) produces a positive number with the correct units. The code `T = -(1/lambda)*log( rng() );` generates the exponential.

We verify (11) using the informal probability method. Let $f(t)$ be the PDF of the random variable of (11). We want to show $f(t) = \lambda e^{-\lambda t}$ for $t > 0$. Let B be the event $t \leq T \leq t + dt$. This is the same as

$$t \leq \frac{-1}{\lambda} \ln(U) \leq t + dt \ ,$$

which is the same as

$$-\lambda t - \lambda dt \leq \ln(U) \leq -\lambda t \quad \text{(all negative)},$$

and, because e^x is an increasing function of x,

$$e^{-\lambda t - \lambda dt} \leq U \leq e^{-\lambda t} \ .$$

Now, $e^{-\lambda t-\lambda dt} = e^{-\lambda t}e^{-\lambda dt}$, and $e^{-\lambda dt} = 1-\lambda dt$, so this is

$$e^{-\lambda t} - \lambda dt e^{-\lambda t} \le U \le e^{-\lambda t} \ .$$

But this is an interval within $[0,1]$ of length $\lambda\, dt\, e^{-\lambda t}$, so

$$\begin{aligned} f(t)dt &= \Pr(t \le T \le t+dt) \\ &= \Pr(e^{-\lambda t} - \lambda dt e^{-\lambda t} \le U \le e^{-\lambda t}) \\ &= \lambda\, dt\, e^{-\lambda t} \ , \end{aligned}$$

which shows that (11) gives an exponential.

3.3 Using the distribution function

In principle, the CDF provides a simple sampler for any one dimensional probability distribution. If X is a one component random variable with probability density $f(x)$, the cumulative distrubution function is $F(x) = \Pr(X \le x) = \int_{x' \le x} f(x'dx'$. Of course $0 \le F(x) \le 1$ for all x, and for any $u \in [0,1]$, there is an x with $F(x) = u$. Moreover, if $X \sim f$ and $U = F(X)$, and $u = F(x)$, $\Pr(U \le u) = \Pr(X \le x) = F(x) = u$. But $\Pr(U \le u) = u$ says that U is a standard uniform. Conversely, we can reverse this reasoning to see that if U is a standard uniform and $F(X) = U$, then X has probability density $f(x)$. The simple sampler is, first take `U  rng();`, then find X with $F(X) = U$. The second step may be difficult in applications where $F(x)$ is hard to evaluate or the equation $F(x) = u$ is hard to solve.

For the exponential random variable,

$$F(t) = \Pr(T \le t) = \int_{t'=0}^{t} \lambda e^{-\lambda t'} dt' = 1 - e^{-\lambda t} \ .$$

Solving $F(t) = u$ gives $t = \frac{-1}{\lambda}\ln(1-u)$. This is the same as the sampler we had before, since $1-u$ also is a standard uniform.

For the standard normal we have

$$F(z) = \int_{z'=-\infty}^{z} \frac{1}{\sqrt{2\pi}} e^{-z'^2/2} dz' = N(z) \ . \tag{12}$$

There is no *elementary*[8] formula for the *cumulative normal*, $N(z)$, but there is good software to evaluate it to nearly double precision accuracy, both for $N(z)$ and for the inverse cumulative normal $z = N^{-1}(u)$. In many applications,[9] this is the best way to make standard normals. The general $X \sim \mathcal{N}(\mu, \sigma^2)$ may be found using $X = \sigma Z + \mu$.

[8] An elementary function is one that can be expressed using exponentials, logs, trigonometric, and algebraic functions only.

[9] There are applications where the relationship between Z and U is important, not only the value of Z. These include sampling using normal copulas, and quasi (low discrepency sequence) Monte Carlo.

3.4 The Box Muller method

The *Box Muller* algorithm generates two independent standard normals from two independent standard uniforms. The formulas are

$$\begin{aligned} R &= \sqrt{-2\ln(U_1)} \\ \Theta &= 2\pi U_2 \\ Z_1 &= R\cos(\Theta) \\ Z_2 &= R\sin(\Theta)\ . \end{aligned}$$

We can make a thousand independent standard normals by making a thousand standard uniforms then using them in pairs to generate five hunderd pairs of independent standard normals.

3.5 Multivariate normals

Let $X \in R^n$ be a multivariate normal random variable with mean zero and covariance matrix C. We can sample X using the Choleski factorization of C, which is $C\ LL^T$, where L is lower triangular. Note that L exists because C is symmetric and positive definite. Let $Z \in R^n$ be a vector of n independent standard normals generated using Box Muller or any other way. The covariance matrix of Z is I (check this). Therefore, if

$$X = LZ\ , \tag{13}$$

then X is multivariate normal (because it is a linear transformation of a multivariate normal) and has covariance matrix (see (10)

$$C_X = LIL^t = C\ .$$

If we want a multivariate normal with mean $\mu \in R^n$, we simply take $X = LZ+\mu$.

3.6 Rejection

The *rejection* algorithm turns samples from one density into samples of another. One ingredient is a simple sampler for the *trial distribution*, or *proposal distribution*. Suppose `gSamp()` produces iid samples from the PDF $g(x)$. The other ingredient is an *acceptance probability*, $p(x)$, with $0 \le p(x) \le 1$ for all x. The algorithm generates a trial $X \sim g$ and accepts this trial value with probability $p(X)$. The process is repeated until the first acceptance. All this happens in

```
while ( rng() > p( X = gSamp() ) );
```
(14)

We accept X is $U \le p(X)$, so $U > p(X)$ means reject and try again. Each time we generate a new X, which must be independent of all previous ones.

The X returned by (14) has PDF

$$f(x) = \frac{1}{Z}p(x)g(x)\ , \tag{15}$$

where Z is a normalization constant that insures that $\int f(x)dx = 1$:

$$Z = \int_{x \in R^n} p(x)g(x)dx \ . \tag{16}$$

This shows that Z is the probability that any given trial will be an acceptance. The formula (15) shows that rejection goes from g to f by thinning out samples in regions where $f < g$ by rejecting some of them. We verify it informally in the one dimensional setting:

$$\begin{aligned} f(x)dx &= \Pr(\text{ accepted } X \in (x, x+dx)) \\ &= \Pr(X \in (x, x+dx) \mid \text{ acceptance }) \\ &= \frac{\Pr(X \in (x, x+dx) \text{ and accepted})}{\Pr(\text{ accepted })} \\ &= \frac{g(x)dxp(x)}{Z} \ . \end{aligned}$$

An argument like this also shows the correctness of (15) also for multivariate random variables.

We can use rejection to generate normals from exponentials. Suppose $g(x) = e^{-x}$ for $x > 0$, corresponding to a standard exponential, and $f(x) = \frac{2}{\sqrt{2\pi}} e^{-x^2/2}$ for $x > 0$, corresponding to the positive half of the standard normal distribution. Then (15) becomes

$$\begin{aligned} p(x) &= Z\frac{f(x)}{g(x)} \\ &= Z \cdot \frac{2}{\sqrt{2\pi}} \cdot \frac{e^{-x^2/2}}{e^{-x}} \\ p(x) &= Z \cdot \frac{2}{\sqrt{2\pi}} e^{x - x^2/2} \ . \end{aligned} \tag{17}$$

This would be a formula for $p(x)$ if we know the constant, Z.

We maximize the efficiency of the algorithm by making Z, the overall probability of acceptance, as large as possible, subject to the constraint $p(x) \leq 1$ for all x. Therefore, we find the x that maximizes the right side:

$$e^{x - x^2/2} = \max \implies x - \frac{x^2}{2} = \max \implies x_{\max} = 1 \ .$$

Choosing Z so that the maximum of $p(x)$ is one gives

$$1 = p_{\max} = Z \cdot \frac{2}{\sqrt{2\pi}} e^{x_{\max} - x_{\max}^2/2} = Z\frac{2}{\sqrt{2\pi}} e^{1/2} \ ,$$

so

$$p(x) = \frac{1}{\sqrt{e}} e^{x - x^2/2} \ . \tag{18}$$

It is impossible to go the other way. If we try to generate a standard exponential from a positive standard normal we get accetpance probability related to the recripocal to (17):

$$p(x) = Z\frac{\sqrt{2\pi}}{2}e^{x^2/2-x} \; .$$

This gives $p(x) \to \infty$ as $x \to \infty$ for any $Z > 0$. The normal has thinner[10] *tails* than the exponential. It is possible to start with an exponential and thin the tails using rejection to get a Gaussian (Note: (17) has $p(x) \to 0$ as $x \to \infty$.). However, rejection cannot fatten the tails by more than a factor of $\frac{1}{Z}$. In particular, rejection cannot fatten a Gaussian tail to an exponential.

The *efficiency* of a rejection sampler is the expected number of trials needed to generate a sample. Let N be the number of samples to get a success. The efficiency is

$$E[N] = 1 \cdot \Pr(N = 1) + 2 \cdot \Pr(N = 2) + \cdots$$

We already saw that $\Pr(N = 1) = Z$. To have $N = 2$, we need first a rejection then an acceptance, so $\Pr(N = 2) = (1 - Z)Z$. Similarly, $\Pr(N = k) = (1 - Z)^{k-1}Z$. Finally, we have the geometric series formulas for $0 < r < 1$:

$$\sum_{k=0}^{\infty} r^k = \frac{1}{1-r} \quad , \quad \sum_{k=1}^{\infty} kr^{k-1} = \sum_{k=0}^{\infty} kr^{k-1} = \frac{d}{dr}\sum_{k=0}^{\infty} r^k = \frac{1}{(1-r)^2} \; .$$

Applying these to $r = 1 - Z$ gives $E[N] = \frac{1}{Z}$. In generating a standard normal from a standatd exponential, we get

$$Z = \sqrt{\frac{\pi}{2e}} \approx .76 \; .$$

The sampler is efficient in that more than 75% of the trials are successes.

Rejection samplers for other distributions, particularly in high dimensions, can be much worse. We give a rejection algorithm for finding a random point uniformly distributed inside the unit ball n dimensions. The algorithm is correct for any n in the sense that it produces at the end of the day a random point with the desired probability density. For small n, it even works in practice and is not such a bad idea. However, for large n the algorithm is very inefficient. In fact, Z is an exponentially decreasing function of n. It would take more than a century on any present computer to generate a point uniformly distributed inside the unit ball in $n = 100$ dimensions this way. Fortunately, there are better ways.

A point in n dimensions is $x = (x_1, \ldots, x_n)$. The *unit ball* is the set of points with $\sum_{k=1}^{n} x_k^2 \leq 1$. We will use a trial density that is uniform inside the smallest (hyper)cube that contains the unit ball. This is the cube with $-1 \leq x_k \leq 1$ for each k. The uniform density in this cube is

$$g(x_1, \ldots, x_n) = \begin{cases} 2^{-n} & \text{if } |x_k| \leq 1 \text{ for all } k = 1, \ldots, n \\ 0 & \text{otherwise.} \end{cases}$$

[10]The tails of a probability density are the parts for large x, where the graph of $f(x)$ gets thinner, like the tail of a mouse.

This density is a product of the one dimensional uniform densities, so we can sample it by choosing n independent standard uniforms:

```
for( k = 0; k < n; k++) x[k] = 2*rng() - 1; // unif in [-1,1].
```

We get a random point inside the unit ball if we simply reject samples outside the ball:

```
while(1) {     // The rejection loop (possibly infinite!)

   for( k = 0; k < n; k++) x[k] = 2*rng() - 1; // Generate a trial vector
                                               // of independent uniforms
                                               // in [-1,1].
   ssq = 0;  // ssq means "sum of squares"
   for( k = 0; k < n; k++ ) ssq+= x[k]*x[k];
   if ( ssq <= 1.) break ; // You accepted and are done with the loop.
                           // Otherwise go back and do it again.
 }
```

The probability of accepting in a given trial is equal to the ratio of the volume (or area in 2D) of the ball to the cube that contains it. In 2D this is

$$\frac{area(\text{disk})}{area(\text{square})} = \frac{\pi}{4} \approx .79 \ ,$$

which is pretty healthy. In 3D it is

$$\frac{vol(\text{ball})}{vol(\text{cube})} = \frac{\frac{4\pi}{3}}{8} \approx .52 \ ,$$

Table 1 shows what happens as the dimension increases. By the time the dimension reaches $n = 10$, the expected number of trials to get a success is about $1/.0025 = 400$, which is slow but not entirely impractical. For dimension $n = 40$, the expected number of trials has grown to about 3×10^{20}, which is entirely impractical. Monte Carlo simulations in more than 40 dimensions are common. The last column of the table shows that the acceptance probability goes to zero faster than any exponential of the form e^{-cn}, because the numbers that would be c, listed in the table, increase with n.

3.7 Histograms and testing

Any piece of scientific software is presumed wrong until it proves itself correct in tests. We can test a one dimensional sampler using a *histogram*. Divide the x axis into neighboring *bins* of length Δx centered about bin centers $x_j = j\Delta x$. The corresponding bins are $B_j = [x_j - \frac{\Delta x}{2}, x_j + \frac{\Delta x}{2}]$. With n samples, the *bin counts* are[11] $N_j = \#\{X_k \in B_j, 1 \leq k \leq n\}$. The probability that a given sample lands in B_j is $\Pr(B_j) = \int_{x \in B_j} f(x)dx \approx \Delta x f(x_j)$. The expected bin count is $E[N_j] \approx n\Delta x f(x_j)$, and the standard deviation (See Section 4) is

[11] Here $\#\{\cdots\}$ means the number of elements in the set $\{\cdots\}$.

Table 1: Acceptence fractions for producing a random point in the unit ball in n dimensions by rejection.

dimension	vol(ball)	vol(cube)	ratio	$-ln$(ratio)/dim
2	π	4	.79	.12
3	$4\pi/3$	8	.52	.22
4	$\pi^2/2$	16	.31	.29
10	$2\pi^{n/2}/(n\Gamma(n/2))$	2^n	.0025	.60
20	$2\pi^{n/2}/(n\Gamma(n/2))$	2^n	2.5×10^{-8}	.88
40	$2\pi^{n/2}/(n\Gamma(n/2))$	2^n	3.3×10^{-21}	1.2

$\sigma_{N_j} \approx \sqrt{n\Delta x}\sqrt{f(x_j)}$. We generate the samples and plot the N_j and $E[N_j]$ on the same plot. If $E[N_j] \gg \sigma_{N_j}$, then the two curves should be relatively close. This condition is

$$\frac{1}{\sqrt{n\Delta x}} \ll \sqrt{f(x_j)} \; .$$

In particular, if f is of order one, $\Delta x = .01$, and $n = 10^6$, we should have reasonable agreement if the sampler is correct. If Δx is too large, the approximation $\int_{x \in B_j} f(x)dx \approx \Delta x f(x_j)$ will not hold. If Δx is too small, the histogram will not be accurate.

It is harder to test higher dimensional random variables. We can test two and possibly three dimensional random variables using multidimensional histograms. We can test that various one dimensional functions of the random X have the right distributions. For example, the distributions of $R^2 = \sum X_k^2 = \|X\|_{l^2}^2$ and $Y = \sum a_k X_k = a \cdot X$ are easy to figure out if X is uniformly distributed in the ball.

4 Error bars

It is relatively easy to estimate the order of magnitude of the error in most Monte Carlo computations. Monte Carlo computations are likely to have large errors (see Chapter **??**). Therefore, all Monte Carlo computations (except possibly those for senior management) should report error estimates.

Suppose X is a scalar random variable and we approximate $A = E[X]$ by

$$\widehat{A}_n = \frac{1}{n} \sum_{k=1}^{n} X_k \; .$$

The central limit theorem states that

$$R_n = \widehat{A}_n - A \approx \sigma_n Z \; , \tag{19}$$

where σ_n is the *standard deviation* of $\widehat{A}_n$ and $Z \sim \mathcal{N}(0,1)$. A simple calculation shows that $\sigma_n = \frac{1}{\sqrt{n}}\sqrt{\sigma^2}$, where $\sigma^2 = \text{var}(X) = E[(X-A)^2]$. We estimate σ^2

using[12]

$$\widehat{\sigma_n^2} = \frac{1}{n}\sum_{k=1}^{n}\left(X_k - \widehat{A}_n\right)^2 \ , \tag{20}$$

then take

$$\widehat{\sigma}_n = \frac{1}{\sqrt{n}}\sqrt{\widehat{\sigma_n^2}} \ .$$

Since Z in (19) will be of order one, R_n will be of order $\widehat{\sigma}_n$.

We typically report Monte carlo data in the form $A = \widehat{A}_n \pm \widehat{\sigma}_n$. Graphically, we plot $\widehat{A}_n$ as a circle (or some other symbol) and $\widehat{\sigma}_n$ using a bar of length $2\widehat{\sigma}_n$ with $\widehat{A}_n$ in the center. This is the *error bar*.

We can think of the error bar as the interval $\left[\widehat{A}_n - \widehat{\sigma}_n, \widehat{A}_n + \widehat{\sigma}_n\right]$. More generally, we can consider k standard deviation error bars $\left[\widehat{A}_n - k\widehat{\sigma}_n, \widehat{A}_n + k\widehat{\sigma}_n\right]$. In statistics, these intervals are called *confidence intervals* and the confidence is the probability that A is within the conficence interval. The central limit theorem (and numerical computations of integrals of gaussians) tells us that one standard deviation error bars have have confidence

$$\Pr\left(A \in \left[\widehat{A}_n - \widehat{\sigma}_n, \widehat{A}_n + \widehat{\sigma}_n\right]\right) \approx 66\% \ ,$$

and and two standard deviation error bars have

$$\Pr\left(A \in \left[\widehat{A}_n - 2\widehat{\sigma}_n, \widehat{A}_n + 2\widehat{\sigma}_n\right]\right) \approx 95\% \ ,$$

It is the custom in Monte Carlo practice to plot and report one standard deviation error bars. This requires the consumer to understand that the exact answer is outside the error bar about a third of the time. Plotting two or three standard deviation error bars would be safer but would give an inaccurate picture of the probable error size.

5 Software: performance issues

Monte Carlo methods raises many performance issues. Naive coding following the text can lead to poor performance. Two significant factors are frequent branching and frequent procedure calls.

6 Resources and further reading

There are many good books on the probability background for Monte Carlo, the book by Sheldon Ross at the basic level, and the book by Sam Karlin and Gregory Taylor for more the ambitious. Good books on Monte Carlo include

[12] Sometimes (20) is given with $n-1$ rather than n in the denominator. This can be a serious issue in practical statistics with small datasets. But Monte Carlo datasets should be large enough that the difference between n and $n-1$ is irrelevent.

the still surprisingly useful book by Hammersley and Handscomb, the physicists' book (which gets the math right) by Mal Kalos and Paula Whitlock, and the broader book by George Fishman. I also am preparing a book on Monte Carlo, whth many parts already posted.

Markov Chain Monte Carlo, or *MCMC*, is the most important topic left out here. Most multivariate random variables not discussed here cannot be sampled in a practical way by the *direct* sampling methods, but do have indirect *dynamic* samplers. The ability to sample essentially arbitrary distributions has led to an explosion of applications in statistics (sampling Bayesian posterior distributions, Monte Carlo calibration of statistical tests, etc.), and operations research (improved rare event sampling, etc.).

Choosing a good random number generator is important yet subtle. The native C/C++ function `rand()` is suitable only for the simplest applications because it cycles after only a billion samples. The function `random()` is much better. The random number generators in Matlab are good, which cannot be said for the generators in other scientific computing and visualization packages. Joseph Marsaglia has a web site with the latest and best random number generators.

7 Exercises

1. What is wrong with the following piece of code?

```
for ( k = 0; k < n; k++ ) {
   setSeed(s)
   U[k] = rng();
  }
```

2. Calculate the distribution function for an exponential random variable with rate constant λ. Show that the sampler using the distribution function given in Section 3.3 is equivalent to the one given in Section 3.2. Note that if U is a standard uniform, then $1 - U$ also is standard uniform.

3. If S_1 and S_2 are independent standard exponentials, then $T = S_1 + S_2$ has PDF $f(t) = te^{-t}$.

   (a) Write a simple sampler of T that generates S_1 and S_2 then takes $T = S_1 + S_2$.

   (b) Write a simpler sampler of T that uses rejection from an exponential trial. The trial density must have $\lambda < 1$. Why? Look for a value of λ that gives reasonable efficiency. Can you find the optimal λ?

   (c) For each sampler, use the histogram method to verify its correctness.

   (d) Program the Box Muller algorithm and verify the results using the histogram method.

4. A *Poisson* random walk has a position, $X(t)$ that starts with $X(0) = 0$. At each time T_k of a Poisson process with rate λ, the position moves (jumps) by a $\mathcal{N}(0, \sigma^2)$, which means that $X(T_k + 0) = X(T_k - 0) + \sigma Z_k$ with iid standard normal Z_k. Write a program to simulate the Poisson random walk and determine $A = \Pr(X(T) > B)$. Use (but do not hard wire) two parameter sets:

   (a) $T = 1$, $\lambda = 4$, $\sigma = .5$, and $B = 1$.

   (b) $T = 1$, $\lambda = 20$, $\sigma = .2$, and $B = 2$.

   Use standard error bars. In each case, choose a sample size, n, so that you calculate the answer to 5% relative accuracy and report the sample size needed for this.