Scientific Computing
Chapter VII
Monte Carlo Methods

Jonathan Goodman
Courant Institute of Mathemaical Sciences

Last revised April 10, 2002

1 Introduction

Monte Carlo[1] methods use random numbers to estimate a number, A, whose value itself is not random. A Monte Carlo "run" will produce an estimate, $\widehat{A}$, whose value is random and possibly different[2] from run to run. The "expected value" of $\widehat{A}$ (in the technical sense, see below) may or may not be equal to A. If not, the "bias" is $E[\widehat{A}]-A$. The "statistical error" is $\widehat{A}-E[\widehat{A}]$. Taking a longer run should reduce both the bias and the statistical error. With some notable exceptions, the bias is much smaller than the statistical error.

Monte Carlo methods are the last resort, because they are usually expensive and inaccurate. Even if the definition of A seems to call for Monte Carlo, for example, if A is defined in terms of some random process, still it is prudent to seek a deterministic algorithm to compute A. For example, if we know the probability density of a random variable (definitions reviewed beloe), we would normally use a deterministic integration method such as the trapezoid rule, rather than Monte Carlo. In many cases, recursive algorithms such as the famous binomial tree method in mathematical finance, can compute expected values of quantities defined in terms of markov processes.

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

[1]Monte Carlo (Carl Mountain) is a famous gambling center in Europe. A search engine will reveal that it is also known for car racing.

[2]Whether we get a different estimate depends on how we choose the "seed" for the random number generator, see below.

of points needed for a given accuracy in Monte Carlo does not depend on the dimension, and there is some truth to this.

We wish to distinguish between Monte Carlo and simulation. In Monte Carlo, random numbers are a means to an answer. We could seek deterministic methods for find the same answer. Other computational tasks demand random results. An example of this is computer rendering of textures, such as trees or clouds. Someone planning a complex business deal might want to generate a number of random scenerios just to see what they look like. We call this "simulation" to distinguish it from true Monte Carlo. In simulation, the use of random (or pseudorandom) numbers is part of the problem specification. In Monte Carlo, randomness is at the creative discression of the algorithm designer. With these definitions, the commonly used phrase "Monte Carlo simulation" becomes an oxymoron, referring to an algorithm designer with too little initiative or time for any but the most routine implementation[3]

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 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.

2 Quick review of probability

This is a quick review of the parts of probability needed to discuss the Monte Carlo material discussed here. You may not be able to learn the stuff here, but at least you will get the notation and know what you need to look up.

Probability theory begins with the assumption that there are "probabilities"

[3]Please take this as a challenge, not an insult. There are situations in which no Monte Carlo methods better than 'brute force simulation" are known, and not for lack of very creative people trying.

associated to "events". Event[4] A has probability $Pr(A)$, which is a number between 0 and 1 describing what fraction of the time event A would happen if we could repeat the "experiment" many times. The exact meaning of probabilities is debated at length by philosophers and at the beginnings of thick dusty probability books.

Continuing with basic definitions, an event is defined as 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, $A \subset \Omega$. 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. One event, all heads, consists of a single outcome, $HHHH$. The event "more heads than tails" consists of the five outcomes $HHHH$, $THHH$, $HTHH$, $HHTH$, and $HHHT$. For now I will suppose any set of outcomes is an allowed event but this assumption will be revisited later.

Because events are sets (of outcomes), we can apply the basic set operations to events. For example the intersection of events A and B is the set of events both in A and in B: $\omega \in A \cap B$ means $\omega \in A$ and $\omega \in B$. For that reason, $A \cap B$ represents the event "A and B". For example, if A is the event "more heads than tails" above and B is the event "then dime was heads", then B has 8 outcomes in it (elements), and $A \cup B = \{HHHH, THHH, HTHH, HHHT\}$. Similarly, $A \cup B$ represents the event "A or B". With this notation, one of the axioms (basic assumptions) of probability is

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

Another axiom is that

$$0 \leq Pr(A) \leq 1 \;\; , \;\; \text{for any event } A.$$

These have many intuitive consequences, such as

$$A \subset B \Longrightarrow Pr(A) \leq Pr(B) \; .$$

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

We have the Bayes formula for conditional probability:

$$Pr(A \mid B) = \frac{\Pr(A \cup B)}{Pr(B)} = \frac{\Pr(A \text{ and } B)}{Pr(B)}$$

This formula has the interpretation that we restrict to outcomes in B and then ask what the probability of A is. In this formula, B plays the role of Ω, universe

[4]This conflicts with our previous use of A to represent the answer to some computational problem.

of all posible outcomes. We need to divide by $Pr(B)$ on the right so that $Pr(B \mid B) = 1$.

The "probability space" Ω is finite if it is possible to make a finite list 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 teh probabilities of each outcome[5]. Then, for any event, A,

$$Pr(A) = \sum_{\omega \in A} Pr(\omega) \; .$$

The set of all real numbers, R, is a simple example of a probability space that is not discrete: it is impossible to make a list of all real numbers.[6]

A "random variable" is a number (or vector of numbers) whose value is random. Usually this random number depends on the outcome of some experiment, so we sometimes write $X(\omega)$, indicating that the random variable X depends on[7] ω. A common convention is that events are represented be early capitol letters, random variables are represented by late capitol letters, and possible values of the random variable are represented by corresponding lower case late alphabet letters. For example, any (real valued) random variable has a "distribution function" (also called "cumulative distribution function" or CDF),

$$F(x) = Pr(X \leq x) \; . \tag{1}$$

For any value of x, the value of $F(x)$ is the probability of the event

$$A = \{\omega \in \Omega \text{ such that } X(\omega) \leq x\} \; .$$

The number x is not a random variable and the CDF $F(x)$ is not a random function. For example, if X is the number of tails among the four coins, then the event $X \leq 1$ is the A listed above and

$$F(1) = Pr(HHHH) + Pr(THHH) + Pr(HTHH) + Pr(HHTH) + Pr(HHHT) \; .$$

As for probability spaces, we say that a random variable X is discrete if its possible values can be listed with a finite or infinite list, x_1, x_2, …, x_n, …. For a discrete random variable we define probabilities $f_n = f(x_n) = Pr(X = x_n)$. The "expected value" of a discrete random variable is

$$E(X) = \sum_n x_n f_n \; .$$

[5]It is more correct to say the probability of each single outcome event.

[6]This theorem, which could be in an undergraduate analysis or theoretical advanced calculus course, was first proved by the German turn of the 20^{th} century mathematician who was among the founders of modern logic and set theory, Georg Cantor.

[7]Sometimes people call ω itself the random variable and call $X(\omega)$ a "function of a random variable". This might work for a while, but it makes it hard to say, for example, that X and Y are "correlated random variables".

A random variable that is not discrete may be described[8] by a "probability density function" (PDF), $f(x)$. These are often called "continuous" random variables. This means that X may take a "continuoum" of possible values, not that X is a continuous function of ω. If A is the event $x_1 \leq X \leq x_2$, then

$$Pr(A) = Pr(x_1 \leq X \leq x_2) = \int_{x_1}^{x_2} f(x)dx \ ,$$

and

$$E(X) \int_{-\infty}^{\infty} xf(x)dx \ .$$

More generally, the probability of any event A is given by

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

Notice that the notation for discrete and continuous random variables is not entirely consistent. For practical calculations we work with a slightly dimpler definition of the PDF:

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

Some pure mathematicians complain that this is not rigorous. However, nobody who legitimately makes this complaint would have any trouble rewording our discussion below to be rigorous. Any PDF has the properties $f(x) \geq 0$ for all x, and $\int_{-\infty}^{\infty} f(x)dx = 1$.

Interesting random variables often arise as functions of one or more other random variables. For example, if $y = u(x)$ is a function and X is a random variable, we can define a random variable Y by $Y = u(X)$. This means, get your random number X, then apply the function u to it. The formula (2) allows us to find the PDF for Y if we know the PDF for X or vice versa. For example, if u is one to one (like e^x or $1/x$, but not x^2), $g(y)$ is the PDF for Y, and $y = u(x)$, then $y + dy$ corresponds to $u(x + dx) = u(x) + u'(x)dx$, so

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

This shows that if $y = u(x)$, then $g(y) = f(x)/u'(x)$. There are two ways to evaluate $E(Y)$:

$$E(Y) = \int_{y=-\infty}^{\infty} yg(y)dy = \int_{x=-\infty}^{\infty} u(x)f(x)dx \ .$$

[8]Mathematicians recognize possibilities between discrete random variables and random given by a PDF. The serious measure theory used to describe them is not needed here.

Three commonly used 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, X is "uniformly distributed" in the interval $[0,1]$. It's PDF is

$$f(x) = \begin{cases} 1 & \text{if } 0 \leq x \leq 1 \\ 0 & \text{otherwise.} \end{cases}$$

From this we can create the general random variable uniformly distributed in the interval $[a,b]$ by $Y = aX + b$. The PDF for Y is

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

The standard exponential random variable has PDF

$$f(x) = \begin{cases} e^{-x} & \text{if } 0 \leq x \\ 0 & \text{if } x < 0. \end{cases}$$

The general exponential with rate λ is given by $Y = X/\lambda$ and has PDF

$$g(y) = \begin{cases} \lambda e^{-\lambda y} & \text{if } 0 \leq x \\ 0 & \text{if } y < 0. \end{cases}$$

The standard normal has PDF

$$f(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2} \; .$$

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

$$g(y) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{(-(y-\mu)^2/2\sigma^2)} \; .$$

We write $Y \sim \mathcal{N}(\mu, \sigma^2)$ in this case. A standard normal has distribution $\mathcal{N}(0,1)$.

If X and Y are two random variables we can consider their joint density, $h(x,y)$. For example, of we want to know the probability that $X^2 + Y^2 < 1$, we do the two dimensional integral

$$Pr(X^2 + Y^2 < 1) = \int\int_{x^2+y^2<1} h(x,y) dx dy \; .$$

If we do not need to know about Y, we can compute the "marginal" density for X from the formula

$$f(x) = \int_{y=-\infty}^{\infty} h(x,y) dy \; .$$

Note that $f(x)dx$ is the probability of the event $A = \{(x,y) \text{ such that } x \leq X \leq x + dx\}$, which leads to the formula for f. Of course we can think of joint densities for more than two random variables.

Two events, A, and B, are independent if $Pr(A \mid B) = Pr(A)$, or, equivalently, if $Pr(A \text{ and } B) = Pr(A)Pr(B)$. This may be interpreted as saying that knowing whether B happened gives no information about whether A happened or not. Two continuous random variables X and Y are independent if their joint density is a product of the marginal densities of the individual random variables: $h(x, y) = f(x)g(y)$. This is the same as saying that any event defined in terms of X only is independent of any event defined in terms only of Y. We say that n random variables X_1, …, X_n, are independent of their joint PDF is a product of the individual PDF's:

$$h(x_1, \ldots, x_n) = f_1(x_1) \cdot \; \cdots \; \cdot f_n(x_n) \; .$$

Here $f_k(x_k)$ is the (marginal) PDF for X_k. It is easy to check, for example, that if X_1 and X_2 are independent random variables and $Y_1 = u_1(X_1)$ and $Y_2 = u_2(X_2)$, then Y_1 and Y_2 are independent also.

When we have two or more random variables with some joint density, we may refer to the situation abstractly by calling the random vector $(X_1, \ldots, X_n) \in R^n$ as a random variable, or possibly a "vector valued" random variable. The joint density will just be called the PDF, but integrals defining probabilities and expectation values will be n dimensional integrals. We talk about vector valued random variables the same way we talk about scalar valued random variables. For example, if X and X' are two n dimensional random variables, we can ask about their joint density, which would be a function of $2n$ variables. We say that X and X' are independent if the joint density is a product of the X density (a function of n variables) and the X' density.

3 Sampling and the theorems of simple Monte Carlo

We often describe sampling in statisticians' language. Suppose X is a (possibly vector valued) random variable. If X' has the same PDF as X, we say that X' "is a sample of X". If $f(x)$ is the PDF for X, we then write $X' \sim f$ or sometimes $X' \sim X$. Given a PDF, $f(x)$, the "sampling problem" is to find an algorithm that constructs a potentially unlimited number of independent samples of f. We call these samples $X^{(1)}$, …, $X^{(k)}$, … . Statisticians would call this sequence "iid", for "independent and identically distributed".

The typical situation in Monte Carlo is that there is some n dimensional random variable, X, and a one dimensional random variable Y derived from X is some way: $Y = u(X)$. Though we know the PDF for X, we are unable to do the integrals to find the one variable PDF[9] for Y, though we could find it numerically by Monte Carlo. We produce "samples" of Y by sampling X and applying u. Of course, if the $X^{(k)}$ are iid samples of X, then the $Y_k = u(X^{(k)})$ are iid samples of $Y = u(X)$.

[9]To understand the challenge, try to find the PDF for $Y = X_1^2 + X_2^3$ where X_1 and X_2 are iid $\mathcal{N}(0, 1)$.

Questions about probabilities of events may be formulated in this framework. If B is an event depending on an n dimensional random variable, we define $Y = 1$ if $X \in B$ and $Y = 0$ if $X \notin B$. Then Y is a discrete random variable and $E(Y) = Pr(Y = 1) = Pr(X \in B)$. Such a $0-1$ function defined by a set, B, is called the "characteristic function"[10] or "indicator function" of B, and written $Y = \chi_B(X)$ or $Y = \mathbf{1}_B(X)$. We might use this notation a bit informally, for example, as $Pr(X_1^2 + X_2^3 < 2) = E(\mathbf{1}_{X_1^2+X_2^3<2})$

Two theorems of probability are important for our Monte Carlo applications The first is the "law of large numbers". Suppose that Y_1, Y_2, $\ldots$, is a sequence of iid samples of Y, and that $E(|Y|) < \infty$. Then the sample mean converges to the expected value as the sample size goes to infinity. That is, if

$$A = E(Y) \ ,$$

and the "estimators" of A are the sample means given by

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

Then[11]

$$\lim_{n\to\infty} \widehat{A}_n = A \ .$$

This is sometimes worded as "the sample mean converges to the population mean as the sample size goes to infinity".

This is what we will call "simple" Monte Carlo: the estimator is the mean of a collection of independent samples. For some applications, simple methods are not known or are impractical. In these cases, either the samples, Y_k will not be independent or the estimator is got from the data using more than simple averaging. Non independen samples arise in "Markov chain Monte Carlo" (MCMC). The Robins Munro algorithm for "stochastic approximation" (finding the minimum of a function whose values are calculated by simple Monte Carlo).

For simple Monte Carlo, the central limit theorem provides a simple way to estimate the size of the statistical error $A - \widehat{A}_n$. If we suppose that[12] $E(Y^4) < \infty$ then the statistical error is approximately a normal random variable. If $\sigma^2 = var(Y) = E[(Y-A)^2]$, then the distribution of the random variable

$$R_n = \frac{\widehat{A}_n - A}{\sqrt{n}\sigma}$$

is approximately $\mathcal{N}(0,1)$.

[10]Probabilists often use the term characteristic function for the Fourier transform of the PDF. We will not use those Fourier transforms here.

[11]This is the Kolmogorov "strong law" of large numbers.

[12]There is a branch of statistics devoted to problems in which the hypotheses of the central limit theorem are not satisfied because of "fat tails". In this case statistical estimation and error analysis are very different. We will not encounter those cases here, but you might in practice.

This theorem may be used to construct "error bars" from the Monte Carlo data. First, we estimate σ from the data using the formula[13]

$$\widehat{\sigma}^2 = \frac{1}{n} \sum_{k=1}^{n} \left(Y_k - \widehat{A}_n \right)^2 \quad . \tag{3}$$

This differs from the "simple" estimate of σ^2 in that we have used $\widehat{A}_n$ in place of the unknown A. The "one standard deviation error bar" is the interval

$$[\widehat{A}_n - \frac{1}{\sqrt{n}}\widehat{\sigma}_n, \widehat{A}_n + \frac{1}{\sqrt{n}}\widehat{\sigma}_n] \ . \tag{4}$$

Using the central limit theorem, a little algebra, and $\widehat{\sigma}_n$ for σ, we see that A is inside the error bar when $R_n \in [-1, 1]$. The probability of this is $\frac{1}{\sqrt{2\pi}} \int_{-1}^{1} e^{-x^2/2} dx \approx 2/3$. What we say about error bars is exactly what statisticians say about confidence intervals. The answer is not random but the estimate, $\widehat{A}_n$, is, and so is the error bar interval (4). If several people independently calculate $\widehat{A}_n$ and (4), they each should get slightly different intervals. About 2/3 of them will get error bars that contain A.

The error bar (4) is called "one sigma error bars" because it extends one standard deviation (of $\widehat{A}_n$) on either side of $\widehat{A}_n$. We could also make 2σ or even 3σ error bars by inserting a factor of 2 or 3 in front of $\widehat{\sigma}_n$. The central limit theorem tells us that the chance of not including A goes down to less than 5% for 2σ, and less than 1% for 3σ error bars. By convention, most reported Monte Carlo error bars are one σ. For one thing, this is truly an estimate of the size of the error: 2σ or 3σ is most likely too big. For another thing, the central limit theorem is less accurate about very high confidence error bars. For normal n (say, $n = 10^6$), the central limit theorem correctly predicts the confidence of 1σ error bars but might not get the conficence of 3σ error bars as accurately.

Use error bars. Reporting or using Monte Carlo results without error bars is dangerous. Monte Carlo is so slow that people often take unrealistically small samples. Without error bars, there is little protection from taking this too far. Besides, computing error bars is easy and cheap. The formulas (3) and (4) are simple. Generating the samples and computing the Y_k will almost always be much more expensive.

4 Random number generator

The random number generator (more properly, psuedo random number generator) is the ultimate source of random numbers in a Monte Carlo computation. The numbers you get are not truly random (whatever that may mean). In fact, they are produced by a simple deterministic algorithm, generally just a few lines

[13] Statisticians sometimes prefer to write $1/(n-1)$ instead of $1/n$. If the sample size, n, is large enough for Monte Carlo purposes, the distinction is irrelevent. In real statistical applications with less than 10 data points, the distinction can be important.

of code. A good random number generator will produce a sequence of numbers that passes many statistical tests for “randomness”, the more the better.

In it’s simplest form, a random number generator is a procedure, such as `float rand()`, that produces a random number when called. The random number is supposed to be a standard uniform random number and successive calls are supposed to produce independent samples. That is, the code fragment

```
for (i = 0; i < n; i++)
   t[i] = rand();
```

is should produce n independent random variables, each uniformly distributed in the interval $[0, 1]$. Practical codes approximate this ideal to varying degrees. All of them (almost) produce numbers uniformly distributed in the unit interval. Bad generators produce numbers with correlations. There are well documented cases of Monte Carlo estimations being wrong because of correlations produced by the random number generator. Bad generators abound (see, e.g. Numerical Recipies).

Random number generators in common use are defined by ψ and phi. The generator has a “seed”, s, which is an integer or small collection of integers. The function ψ computes a new seed from the old one by some simple algorithm. For example, congruential random number generators use a single integer as seed and $\psi(s) = (as + b)_{\text{mod } l}$. The function ϕ produces a real number in the interval $[0, 1]$ for every seed s. In a congruential random number generator, we have $\phi(s) = s/l$. A call to `rand()` has the effect of performing $s_{n+1} = \psi(s_n)$ and returning $\phi(s_{n+1})$. For this reason if you want to reproduce a sequence of “random numbers”, you just need to reproduce the calling seed and run your program again. Random number generators that do not take a seed from the user usually get it from some “random” source such as the system clock.

Even though the random number generator is not perfect, the best ones are so good that it is very unlikely that you would be able to tell that your numbers were not ideal independent standard uniforms. From now on, we will assume this.

5 Basic sampling methods

All random variables used in Monte Carlo are created from standard uniform random variables in some way. Monte Carlo “sampling” is the problem of producing random variables of a certain kind starting with independent standard uniform random variables.

5.1 Coin tossing: Bernoulli random variables

The very simplest sampling problem is “coin tossing”: producing a random variable so that $X + 1$ with probability p and $X = 0$ with probability $1 - p$. This is sometimes called the Bernoulli random variable. A simple code to do this is

```
   if ( p > rand() ) . . . yes (X=1)
   else  . . . . . . . . . no  (X=0)
```

This works because if T is a standard uniform random variable, then $Pr(T < p) = p$. Of course, we produce independent Bernoulli random variables if we use independent uniforms. In this way, we can simulate (terminology as in the introduction) any random process given by discrete choices with defined probabilities.

To make discrete sampling more concrete, we outline a code that would take n steps of a d state discrete time Markov chain. If you don't know what that is, skip to the next section. The transition matrix is a $d \times d$ matrix, R, with entries r_{ij} being the probability of going from state i to state j in one time step. We denonte the state at time t by $X(t)$. For each $t = 0, 1, 2, ...$, $X(t)$ will be one of the integers $1, 2, ..., d$, with $X(t) = j$ meaning that the system was in state j at time t. The technical definition of the transition probabilities is $r_{ij} = Pr(i \to j) = Pr(X(t+1) = j \mid X(t) = i)$. For use in the code we define "cumulative probabilities" to be the probabilities of going from state i to a state numbered less than j:

$$rc_{ij} = Pr(X(t+1) < j \mid X(t) = i) = \sum k < j r_{ik} \ .$$

We define $rc_{i1} = 0$ if the lowest numbered state is state 1. We know that $\sum_{k=1}^{d} r_{ik} = 1$ for any i, corresponding to the fact that $X(t+1)$ must be one of the d states. This means that $rc_{id} + r_{id} = 1$.

These definitions give us a simple way to take a random step of a Markov chain. For any given current state, i, the cumulative probabilities rc_{ij} divide the unit interval into subintervals of length r_{ij}. That is, $0 = rc_{i1} \leq rc_{i2} \leq \cdots \leq rc_{id} \leq 1$, and $rc_{i,j+1} - rc_{ij} = r_{ij}$. For this reason, the probability that a standard uniform random variable falls in the interval $[rc_{i,j},]rc_{i,j+1}]$ is r_{ij}. Also, for any given i, these events are disjoint. Thus, to take a random step from state i, we sample the standard uniform density to get T, then determine which of the intervals $[rc_{i,j},]rc_{i,j+1}]$ T falls into. That value of j is the next state. Here is a sketch of a C procedure that does this.

```
#define SinglePrecisionEPSMACH 1.e-6 /* not the exact number, but OK here. */

int runChain(int X[], int d, float *r, int n, int X0 ){

/* X  is an array of n ints so that X[t] will hold that state at time t.
   d  is the number of states and the dimensions of the matrices r and rc.
   p  is a d X d matrix of transition probabilities
   n  is the number of states to be created, counting the initial state.
   X0 is the initial state.   */

   float *rc;  // Will be the array of cumulative probabilities.
   int i, j, t;
   rc = new float[d*d]; // You need a d X d matrix.
        if ( rc == 0 ) {    // Paranoid error checking.
           cout << "In runChain, could not allocate " << d << " by << d
                << " array.  Returning without simulating." << endl;
           return 1; }

        // Compute the entries of the cumulative probability matrix.

    float pSum;  // The partial sum of probabilities in a row of r.
    for ( i = 0; i < d; i++) {
       pSum = 0.;
       for ( j = 0; j < d; j++ ) {
          rc[ n*i + j ]  = pSum;
          psum           += r[ n*i + j ];
        }
         // Just for fun, you can chech the the row sum of r was about 1.
       if ( abs( psum - 1. ) > d * SinglePrecisionEPSMACH 1.e-6 {
          cout << "In runChain, it looks like the sum of the transition "
               << "probabilities from state " << i << " add up to << psum
               << ", which doesn't seem close to 1." << endl;
          cout << "Returning without simulating." << endl;
          return 2;}

   //   Run the chain.  This is the main part.

   if ( X0 < 0 || X0 >= d ) { print an error message and bail }
   X[0] = X0;
   float U; // The standard uniform random variable.
   for ( t = 0; t < n; /* the increment is in the loop*/ ) {
      i = X[t];  // The state you are stepping from.
      U = rand();

              // This is the inner loop where most of the work is.

      for ( j = 0; k < d; k++ )   // Figure out which interval U falls in.
         if ( rc[ n*i + j + 1 ] > U ) break;
      X[++t] = j;
     }
   return 0;  // As far as I know, it worked.
 }
```

The reader who knows something about computer algorithms might complain that the inner loop consists of a brute force linear search. A binary search would be much faster for even moderate values of d. I hope to return to this point later.

5.2 Exponential random variables

Another simple sampling problem is for exponential random variables. It is easy and fun to show that if T is a standard uniform and $X = -\ln(T)$, then X is a standard exponential. It is easy to show that $Pr(x \leq X \leq x + dx)$ when $X = -\ln(T)$ as we did in section 2. We can then get an exponential with rate λ by dividing by λ. In this way we can simulate any continuous time discrete state space Markov chain.

5.3 Standard normal random variables: Box Muller

Standard normal random variables can be made using the amazing Box Muller method. We start with independent uniforms, T_1 and T_2. Then we compute $\Theta = 2\pi T_1$ and $R = \sqrt{-2\ln(T_2)}$. Finally, the numbers $X_1 = R\cos(\Theta)$ and $X_2 = R\sin(\Theta)$ are two independent standard normals. If I want 1000 independent standard normals, I start with 1000 standard uniforms, then 500 times create a pair of standard normals from a pair of new standard uniforms. `Explain why this works!`

5.4 Mapping methods in one D

In general, it is possible to sample a one dimensional random variable if you are able to compute and invert its cumulative distribution function (CDF), $F(x) = Pr(X \leq x)$. To get a sample of X, you let T be a standard uniform, then solve the equation $F(X) = T$ for X. If you apply this idea to the exponential random variable, you get the formula $X = -\ln(1 - T)$. Of course, if T is standard uniform, then so is $1 - T$. The CDF for the standard normal is often called $N(x)$. This is related to, but not the same as the error function $erf(x)$. Even though there is no algebraic formula for $N(x)$ or its inverse, finding standard normals by evaluation a numerical approximation to the inverse of $N(x)$ is as efficient as the Box Muller method, but not nearly as cool.

As an example of this, suppose $X = (X_1, \ldots, X_n)$ is uniformly distributed in the unit ball in n dimensions. This means that the PDF for X is

$$f(x) = \frac{1}{vol(\text{ball})} \times \begin{cases} 1 & \text{if } \|x\|_{l^2} < 1 \;, \\ 0 & \text{otherwise.} \end{cases}$$

We want to sample the random variable $Y \;\; \|X\|$. We can find the CDF for Y by integration in polar coordinates in n dimensions:

$$\begin{aligned} Pr(Y \leq y) &= \int_{\|x\|\leq y} 1dx \Big/ \int_{\|x\|\leq y} 1dx \\ &= \int_{0\leq r\leq y} r^{n-1}dr \Big/ \int_{0\leq r\leq 1} r^{n-1}dr \\ &= y^n \;, \end{aligned}$$

as long as $0 \leq y \leq 1$. Thus, to sample Y, we need to solve $Y^n = T$, where T is a standard uniform random variable. This is obviously given by $Y = T^{1/n}$. Note that this Y is in the interval $[0, 1]$ as desired.

6 Rejection sampling

The direct sampling methods discussed above do not suffice for all practical applications. Many computations call for finding a single sample, X_n, from each of the probability densities $f_n(x)$. This may happen when the density function has some parameters whose values are adjusted during the computation. Other applications call for multidimensional random variables with a given multivariable PDF. In many physical applications we have a PDF of the form

$$f(x) = \frac{1}{Z} g(x) \; , \tag{5}$$

where g is known but the normalization constant $Z = \int_{-\infty}^{\infty} g(x)dx$ is not. This is the case, for example, when g is a Gibbs-Boltzmann distribution and Z is the partition function.

Rejection sampling is a general method for sampling from a density of the form (5) even though we do not know Z. What we do need is a source of independent random variables from the "trial" distribution $f_0(x)$, and an acceptance probability function, $p(x)$. If we suppose that successive calls to $fZero()$ produce independent samples from f_0, the rejection algorithm is (in not very transpearent code)

```
while( rand() > p( x = fZero() ) );
```

This algorithm draws a sample, X from the f_0 population. Then it evaluates the acceptance probability that depends on the sample drawn, $p(X)$. Finally, it "accepts" X with that probability. If X is accepted, that X is the result of the rejection algorithm. If X is rejected, the algorithm starts over, drawing a new independent X from the $f_)$ population. The efficiency of the rejection algorithm depends on the work required to sample f_0 and evaluate p, and on the number of samples required before getting an acceptance. In developing practical Monte Carlo codes based on rejection, we often spend considerable time fine tuning in order to improve the acceptence probability.

Let us determine the probability density for the random variable returned by the rejection algorithm. We find this using the definition (2) of $f(x)$ together with the rules for conditional probability. Suppose one "experiment" consists of sampling from f_0, evaluating p, and accepting with probability p. Let A be the event that the X was accepted. Denote the overall acceptence probability by $\zeta = Pr(A)$. The formula for ζ is

$$\zeta = \int_{-\infty}^{\infty} p(x) f_0(x) dx \; .$$

Here we sum (integrate) over all possible x values, multiplying the conditional probability of acceptence given $x \leq X \leq d+dx$, which is $p(x)$, by the probability that $x \leq X \leq x + dx$, which is $f_0(x)dx$. Then we have

$$f(x)dx \quad = \quad Pr(x \leq X \leq x + dx \mid A)$$

$$\begin{aligned} &= \frac{Pr(x \leq X \leq x + dx \quad \text{and} \quad A)}{Pr(A)} \\ &= \frac{1}{\zeta} f_0(x)dx \cdot p(x) \; . \end{aligned}$$

To get the last line, we used the fact that to get $x \leq X \leq x + dx$ and A, we first must generate X in that interval, which happens with probability $f_0(x)dx$, then we must accept that x value, which happens with probability $p(x)$. Altogether we have

$$f(x) = \frac{1}{\zeta} f_0(x) p(x) \; . \tag{6}$$

The value of ζ determines the overall efficiency of the rejection algorithm. Since the probability of success on any given trial is ζ, and the trials are independent, the expected number of trials before the first success in $1/\zeta$. This is an elementary probability calculation left to the reader.

Example. Here is a rejection algorithm for generating a standard normal random variable by rejection from a standard exponential density. First we will sample from the positive half of the standard normal density then at the end multiply by -1 with probability $1/2$ to get the whole density. This works because the standard normal density is symmetric. The density for the positive half is

$$f(x) = \begin{cases} \frac{2}{\sqrt{2\pi}} e^{-x^2/2} & \text{if } x > 0 \\ 0 & \text{otherwise.} \end{cases}$$

The factor of 2 on the top line comes from the fact that the top half of the normal density has exactly half the total probability. If we reject from the standard exponential and use (6) to define $p(x)$, we get

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

This is for $x \geq 0$, we never get $x < 0$. For efficiency, we want to make ζ as large as possible subject to the constraint that $p(x) \leq 1$ for all x. A little calculus shows that the maximum of $p(x)$ is

$$p_{\text{max}} = \zeta \frac{2e^{1/2}}{\sqrt{2\pi}} \; .$$

The largest possible *zeta* is the one that gives $p_{\text{max}} = 1$, which is

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

Thus, the overall acceptence probability is more than $3/4$, which implies that this rejection algorithm might have overall performance not much worse than Mox Muller. The final formula for the acceptance probability is

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

Example. This example shows that the curse of dimensionality can strike Monte Carlo methods as well. We give a rejection algorithm for finding a "random" point inside the unit ball or on the unit sphere in n dimensions. By random, we mean uniformly distributed. The algorithm "works" for any n in the sense that it is correct, producing 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 bak idea. However, for large n the algorithm is very inefficient. In fact, ζ will be an exponentially decreasing function of n. The amount of time needed to generate a point uniformly distributed inside the unit ball in $n = 100$ dimensions would be more than a century on the fastest computer.

We first discuss a random point inside the unit ball. The trial density will be uniform density inside the smallest cube that contains the unit ball:

$$f_0(x_1, \ldots, x_n) = \begin{cases} 2^{-n} & \text{if } |x_k| \leq 1 \text{ for all } k = 1, \ldots, n \\ 0 & \text{otherwise.} \end{cases}$$

It is easy to sample this density because it is a product of the one dimensional uniform densities. In other words, we can sample f_0 by choosing n independent standard uniforms:

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

It seems clear that we get a random point inside the unit ball if we simply reject all the samples outside the ball:

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

   for( k = 0; k < n; k++) x[k] = 2*rand() - 1; // Generate a trial vector
                                                // of independent uniforms
                                                // in [-1,1].
   ssq = 0;  // ssq stands for "sum of squares",
             // a statisticians' abbreviation.
   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

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

which is pretty healthy. In 3D it is

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

The table 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

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
6	$2\pi^{n/2}/(n\Gamma(n/2))$	2^n	.081	.42
10	$2\pi^{n/2}/(n\Gamma(n/2))$	2^n	.0025	.60
15	$2\pi^{n/2}/(n\Gamma(n/2))$	2^n	1.2×10^{-5}	.76
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

$1/.0025 = 400$, which is slow but not entirely impractical if nobody has a better idea. For a ten dimensional problem this is a possible approch. For dimension $n = 40$, the expected number of trials has grown to about 3×10^{20}, which makes this approach impractical. Monte Carlo simulations in 40 dimensions and far more 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.

There are several other sampling methods that work fine in high dimensions.