http://www.math.nyu.edu/faculty/goodman/teaching/MonteCarlo12/

Class notes: Monte Carlo methods
Week 1, Direct sampling
Jonathan Goodman
February 5, 2013

1 Introduction to Monte Carlo

Monte Carlo means using random numbers to compute something that itself is not random.[1] For example, suppose X is a random variable with some distribution, $V(x)$ is some function, and we want to know $A = E[V(X)]$. There may be another random variable Y with another distribution, and another function $W(y)$, so that $A = E[W(Y)]$. *Simulation* is the process of generating random variables with the X distribution for their own sakes. Much of the effort in Monte Carlo goes into looking for alternative ways to estimate the same number. The alternatives may have lower variance or be easier to evaluate. This is a difference between Monte Carlo and simulation.

There are many scientific problems for which Monte Carlo is among the best known solution methods. Most of these arise through the *curse of dimensionality*. In its simplest form, this curse is that it is impractical to create a mesh for numerical integration or for PDE solving in high dimensions. A mesh with n points on a side in d dimensions has n^d mesh points in total. This is impractical, for example, if $n = 10$ and $d = 50$.

A more general version of the curse is that it is impossible to represent a generic function $f(x)$ in high dimensions. Consider a general polynomial in d variables. A polynomial is a linear combination of monomials. The number of monomials $x_1^{k_1} \cdots x_d^{k_d}$ of degree $\leq n$ is $\binom{n+d}{n}$. This is the number of coefficients you need to represent a general polynomial of degree n. For example, you need about $10,000$ coefficients for degree 4 in 20 variables, and about thirty million coefficients for a degree 10 polynomial in 20 variables. For example, dynamic programming is an algorithm that requires you to represent the "value function" (for economists) or the "cost to go function" (for engineers). Dynamic programming is impractical except for low dimensional problems.

On the other hand, if $f(x)$ is a probability density, it may be possible to represent f using a large number of samples. A *sample* of f is a random variable X that has f as its probability density. If X_k for $k = 1, ..., N$ is a collection of samples, then

$$E[V(X)] = \int V(x) f(x)\, dx \approx \frac{1}{N} \sum_{k=1}^{n} V(X_k) \, .$$

A collection of samples of f contains the information you need to estimate an integral. Sampling is one of the core technical issues in Monte Carlo.

[1] I learned this definition to Malvin Kalos, who introduced me to the subject.

This course covers technical issues of Monte Carlo. The first is sampling. If you have a description of the density f, how do you find samples from it? This depends on how f is defined and how much you know about it. The curse of dimensionality is a work here too. There are no direct sampling methods for generic densities in more than a few dimensions. *Markov chain Monte Carlo*, or *MCMC*, is the alternative. You construct a Markov chain that has f as its invariant density. Creating sample paths from the chain leads to samples of f. There are many surprising tricks for creating such chains and for analyzing their performance. This is an active and exciting area today. New MCMC sampling methods open up new possibilities in statistics, engineering, and science.

2 Direct sampling methods

It is a sad fact that math classes start with the most gritty technical part of their subject. A real analysis class starts with set theory and sigma algebras. A numerical computing class starts with the IEEE standards for floating point standard. In that spirit, this Monte Carlo course starts with the direct sampling methods that are at the deepest level of most Monte Carlo codes.

Suppose $f(x)$ is the probability density for an n component random variable X. A *direct sampler* is an algorithm or a piece of software that produces independent random variables with the density f. Consider the code fragment

```
X1 = fSamp();
X2 = fSamp();
```

If `fSamp()` is a direct sampler of f, then $X_1 \sim f$, and $X_2 \sim f$, and X_1 is independent of X_2.

Most probability distributions you meet in practice do not have practical direct samplers. Sampling them requires MCMC. But direct samplers are important parts of most MCMC methods.

3 Pseudo random number generators

A *pseudo random number generator* is the basic ingredient of any sampler. A perfect random number generator (we usually drop the "pseudo") would be a procedure `uSamp()` so that `U[i] = uSamp();` (in a loop over `i`) would fill the array `U` with independent random variables uniformly distributed in the interval $[0, 1]$. The word "pseudo" tells us that the numbers are not actually random, but are produced by a deterministic algorithm. More sophisticated random number generators produce "random" numbers that pass sophisticated tests for uniformity (easy) and independence (hard).

All random number generators in common use have the same structure. There is a *seed*, s, that is some small amount of data. *Congruential* random number generators have a seed that is an integer $s \in \{0, 1, \ldots, p-1\}$, where p is a large prime number. More sophisticated generators have a seed that may

consist of several integers or other discrete data. A random number generator is defined by two functions, a seed update function Φ, and an output function Ψ. The seed update function produces a new seed from the current one. If the current seed is s_n, the next seed is $s_{n+1} = \Phi(s_n)$. The output function produces a number in the interval $[0, 1]$ from the seed If s_n is the current seed, then $U_n = \Phi(s_n)$ is the corresponding "random" number. A call `U = uSamp()` has two effects. It returns the number $U = \Psi(s)$, and it updates the seed: $s \leftarrow \Phi(s)$. If you start with an s_0 and call `uSamp()` many times in succession, you will get the sequence $U_n = \Psi(s_n)$, where $s_{n+1} = \Phi(s_n)$. If you start again with the same s_0, you will get the same sequence U_n.

Congruential random number generators have update function $s_{n+1} \equiv as_n + b \pmod{p}$, with $s_{n+1} \in \{0, 1, \ldots, p-1\}$. Here a and b are integers smaller than p. A good one has a large p and a and b. The output function is just $U = s/p$. One talks about the number of bits in a congruential random number generator. A computer integer has 32 bits (regular integer) or 64 bits (long integer). A long unsigned integer is in the range $\left\{0, 1, \ldots, 2^{64} - 1\right\}$. If p is a prime number close to 2^{64}, then the generator has close to 64 bits. The number of integers between 0 and $2^{64} - 1$ is

$$2^{64} = 16 \cdot 2^{60} = 16 \cdot \left(2^{10}\right)^6 \approx 16 \cdot \left(10^3\right)^6 = 16 \cdot 10^{18} \ .$$

The most powerful computer being discussed today is an *exa-scale* computer, which would do 10^{18} operations per second and in principle could use all $16 \cdot 10^{18}$ numbers in one calculation. A good 128 bit random number generator simulates 128 bit arithmetic using two or more ordinary computer integers. The data structure of the seed for such a generator would be two or four ordinary computer integers.

A good random number generator has a 1 to 1 update function. If $s \neq s'$, then $\Phi(s) \neq \Phi(s')$. Such an update function will produce distinct seeds until is has used every seed available to it. The number of available seeds is the *cycle length* of the random number generator. A good random number generator has a cycle length significantly longer than any realistic computation. For example, an exa-scale computer would not exhaust a 128 bit congruential generator. If the output is a 64 bit double precision floating point number and you use a 128 bit seed, then many seeds map to the same output. Getting the same U does not force the random number generator to cycle.

A random number generator should allow the programmer to read and set the seed. A procedure such as `s = getSeed();` will return the current seed. Another procedure `setSeed(s);` will set the seed equal to the given s. A Monte Carlo computation should set the seed exactly once before using any random numbers. Reading the seed at the end of a Monte Carlo calculation would let you continue the computation where it left off. If you set the seed more than once in a Monte Carlo computation, you probably will ruin the computation. The random number generators that come with major languages (C/C++, Python, Matlab, ...) will set the seed in some arbitrary way if the programmer doesn't do it. You can run the program twice to see whether the seed is set the same way

each time. The random number generator `rand();` that comes with C/C++ and UNIX used to be a 16 bit generator and not suitable for scientific computing.

4 Mapping methods for direct sampling

I.i.d uniformly distributed random variables are used to generate all the other random variables in Monte Carlo. We usually assume that the uniform random number generator is perfect, that it produces a sequence U_n that is exactly i.i.d. and uniformly distributed in $[0, 1]$. A *sampler* is an algorithm that uses one or more such uniforms to generate samples from other densities.

A *direct sampler* of density f uses one or more uniforms to generate an independent sample $X \sim f$. Suppose it takes k uniforms to generate X. Mathematically, this means that there is a function $x = G(u_1, \ldots, u_k)$ so that if the U_j are i.i.d. uniforms, and $X = G(U_1, \ldots, U_k)$, then $X \sim f$. This section gives some examples of direct samplers where k is known in advance. These are *mapping methods*, with G being the mapping. The next section discusses rejection methods, where k is not known in advance.

4.1 Exponential random variables

The *exponential* distribution with rate parameter λ has probability density $f(t) = \lambda e^{-\lambda t}$, if $t \geq 0$, and $f(t) = 0$ if $t < 0$. You can think of an exponential random variable, T, as the time you have to wait until "a bell rings". The restriction $T \geq 0$ is natural if $t = 0$ represents the present. The probability that the bell rings right away is $P(0 \leq T \leq dt) = f(0)\, dt = \lambda\, dt$. We call λ the *rate constant* because it is the "rate" of bell ringing at the beginning. What is special about the exponential random variable is that it has no memory. If the bell has not rung by time s, it is as though time starts over then. More precisely: $P(T \in [s, s + dt] \mid T \geq s) = \lambda\, dt$. The conditional probability density is $f(t \mid T \geq s) = e^{-\lambda(t-s)}$. (The reader should do the easy verification that the two conditional statements are equivalent.)

Exponentials (i.i.d. exponential random variables with arbitrary λ) have many uses in Monte Carlo. The occupation times of a continuous time Markov chain are exponential. The Green's function commonly used in *Green's function Monte Carlo*, or *GFMC*, is sampled using an exponential, see Exercise 4. GFMC is a technique in quantum Monte Carlo, or *QMC*, which just means using Monte Carlo methods to solve problems in quantum mechanics.

To make an exponential, just set

$$T = \frac{-1}{\lambda} \log(U) \;, \tag{1}$$

where U is uniform $[0, 1]$. We verify this by computing the probability density of the random variable T defined by (1). This density is defied by

$$f(t)\, dt = P(T \in [t, t + dt]) \;.$$

But we can calculate that this event says about U:

$$\begin{aligned} t \le T \le t + dt &\iff t \le \frac{-1}{\lambda} \log(U) \le t + dt \\ &\iff -\lambda t - \lambda dt \le \log(U) \le -\lambda t \\ &\iff e^{-\lambda t} e^{-\lambda dt} \le U \le e^{-\lambda t} \\ &\iff e^{-\lambda t}(1 - \lambda dt) \le U \le e^{-\lambda t} \ . \end{aligned}$$

This is an interval in $[0, 1]$ of length $e^{-\lambda t}\lambda dt$. Since U is uniformly distributed in $[0, 1]$, the probability of this is $e^{-\lambda t}\lambda dt$. This implies that $f(t) = \lambda e^{-\lambda t}$. A careful reader may worry that we implicitly assumed that $t > 0$. The formula (1) does not produce negative T if $U \in [0, 1]$.

Here are two checks of (1). Since $U \le 1$, we have $T > 0$. That is why we need the minus sign. If the random number generator gives $U = 0$, then T is not defined. Unfortunately, some random number generators do that from time to time, so you might need to check that $U \neq 0$ in the code before applying (1). The other check involves the λ factor. If λ is large, the rate is fast and T happens sooner. Our formula does that. To say this in a deeper way, the units of λ are 1/Time because λ is a rate. Therefore $1/\lambda$ has the same units as T.

4.2 Normals, Box Muller

The Box Muller algorithm turns two independent uniforms into two independent standard normals. A standard normal is a normal with mean zero and variance 1. The algorithm is both elegant and easy to program. Alas, it might not be the very best way to make large numbers of i.i.d. standard normals.

It is based on the trick for computing the Gaussian integral

$$I = \int_{-\infty}^{\infty} e^{-x^2/2}\, dx \ .$$

The trick is to write

$$\begin{aligned} I^2 &= I \cdot I \\ &= \int_{-\infty}^{\infty} e^{-x^2/2}\, dx \cdot \int_{-\infty}^{\infty} e^{-y^2/2}\, dy \\ &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} e^{-x^2/2} e^{-y^2/2}\, dx dy \\ &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} e^{-(x^2+y^2)/2}\, dx dy \ . \end{aligned}$$

Then switch to polar coordinates $x = r\cos(\theta)$, $y = r\sin(\theta)$, $dxdy = d\theta\, r dr$, and

$x^2 + y^2 = r^2$. Therefore

$$I^2 = \int_{r=0}^{\infty} \int_{\theta=0}^{2\pi} e^{-r^2/2}\, d\theta\, r dr$$
$$= 2\pi \int_{r=0}^{\infty} e^{-r^2/2}\, r dr$$

To work the last integral, use $s = r^2/2$, $ds = rdr$, which gives

$$I^2 = 2\pi \int_0^{\infty} e^{-s}\, ds = 2\pi \ .$$

The point of the trick is that there is no explicit formula for the one dimensional indefinite integral $\int e^{-x^2/2} dx$. But in two dimensions, the indefinite integral is $\int e^{-r^2/2}\, rdr$, which we know from $\frac{d}{dr} e^{-r^2/2} = -re^{-r_2/2}$.

Here is the Monte Carlo version of this trick. We seek to make a pair, (X, Y), of independent standard normals. The probability density is $Ce^{-(x^2+y^2)/2}$, which is isotropic. See Exercise 1 for a related consequence of the fact that the joint density of independent standard normals is isotropic. The polar coordinate representation of $(X, Y) = (R\cos(\Theta), R\sin(\Theta))$ involves a random distance, R, and a random angle Θ. Clearly, Θ is uniformly distributed in $[0, 2\pi]$, so we can generate it using

$$\Theta = 2\pi U_1 \ , \tag{2}$$

where U_1 is uniform $[0, 1]$. Clearly R and Θ are independent. The density of R is $f(r) = re^{-r^2}$. This is because

$$f(r)\, dr = P(r \le R \le r+dr) = \frac{1}{2\pi} \int \int_{r \le R \le r+dr} e^{-(x^2+y^2)/2}\, dxdy = e^{-r^2/2}\, rdr \ .$$

Exercise 5 uses the method of Subsection 4.4 to show that a sampler for this distribution is

$$R = \sqrt{-2\log(U_2)} \ . \tag{3}$$

It is possible to find an explicit sampler because there is an explicit formula for the indefinite integral of f. Using the Box Muller algorithm you can fill an array `Z` with $2n$ i.i.d. standard normals

```
for ( i = 0; i < n; i++) {
  U1 = uSamp();
  U2 = uSamp();
  Th = 2*pi*U1;
  R  = sqrt( -2*log( U2) );
  X  = R*cos(Th);
  Y  = R*sin(Th);
  Z[2*i]   = X;
  Z[2*i+1] = Y;
 }
```

4.3 Order statistics

This is just a small trick to illustrate some possibilities. I don't necessarily recommend that you ever do this. Suppose U_1 and U_2 are independent standard uniforms. Then $X = \max(U_1, U_2)$ has density $f(x) = 2x$, if $0 \leq x \leq 1$, and $f(x) = 0$ otherwise. You can calculate this by caluclus. Going further, the result of sorting $(U_1, \ldots, U_n)$ is written $U_{(1)} \leq \cdots \leq U_{(n)}$. The k^{th} smallest of $(U_1, \ldots, U_n)$ is the k^{th} *order statistic*, $U_{(k)}$. If $n = 2$, then $X = U_{(2)} = \max(U_1, U_2)$ as above. If $n = 3$, the density of $X = U_{(2)}$ is $6x(1-x)$. The density goes to zero as $x \to 0$ or $x \to 1$ because it is hard for the middle of three to be very close to zero or 1. The density of $U_{(k)}$ has a factor of x for each $j < k$ and a factor of $(1-x)$ for each $j > k$. These densities do not arise so often by themselves, but they can be useful as proposals for rejection sampling discussed in Section 6.

4.4 Inverting the CDF

If X is a one dimensional random variable, the *cumulative distribution function*, or *CDF*, is $F(x) = P(X \leq x)$. For example, a standard uniform CDF is $F(u) = u$ for $u \in [0, 1]$. An exponential with rate constant λ has CDF $F(t) = 1 - e^{-\lambda t}$, for $t \geq 0$. In general, the CDF of X is an increasing function of x, and it is strictly increasing for any x with $f(x) > 0$. Also, $F(x) \to 0$ as $x \to -\infty$, and $F(x) \to 1$ as $x \to \infty$.

The CDF is useful for sampling because if $X \sim f$ with CDF F, then the random variable $U = F(X)$ is uniform in $[0, 1]$. Conversely, if U is uniform $[0, 1]$ and we find X with $F(X) = U$, then $X \sim f(x) = F'(x)$. This is "obvious". If $u \in [0, 1]$, and x is some number so that $u = F(x)$, then the events $U \leq u$ and $X \leq x$ are the same, so they have the same probability, which therefore is $F(x)$. You have to be careful about degenerate cases where $f(x)$ vanishes (e.g., the exponential for $t < 0$) and partly discrete random variables where $F(x)$ can be discontinuous. Otherwise, there is a unique x for each $u \in [0, 1]$ and a unique u for each x, and the inverse function $x = F^{-1}(u)$ is well defined.

$$X = F^{-1}(U) \tag{4}$$

generates a sample $X \sim f$.

For example, you can generate an exponential with rate constant λ by solving $F(T) = U$, which given $1 - e^{-\lambda T} = U$ and then

$$T = \frac{-1}{\lambda} \log(1 - U) \; .$$

This is the same as (1) because $1 - U$ has the same uniform distribution as U. Another example, with $f(r) = Cr^n$, for $0 \leq r \leq 1$ and $f = 0$ otherwise, is in Exercise 3.

It is not quite so simple for normals. The CDF, written $N(x)$, is not an elementary function. Nevertheless, there is fast and accurate software that computes $N(x)$ and $N^{-1}(u)$ quickly and almost to machine precision. Generating

normals this way is probably a little faster than using Box Muller. But it is not as fun.

5 Multivariate normal sampling via Cholesky factorization

It is nearly impossible, in general, to make efficient direct samplers from high dimensional distributions. The multivariate normal is an exception. A multivariate normal with mean μ and covariance C has probability density

$$f(x) = \frac{1}{(2\pi)^{n/2}\det(C)^{1/2}} e^{-(x-\mu)^t H(x-\mu)/2} \,, \tag{5}$$

where $H = C^{-1}$. If X has this density, we write $X \sim \mathcal{N}(\mu, C)$. In one dimension the covariance matrix is just the variance σ^2, where σ is the standard deviation. If Y is another multivariate normal, we write μ_X and μ_Y, C_X and C_Y for the parameters of the distributions.

Suppose $X \sim \mathcal{N}(\mu_X, C_X)$. Let $Y = AX + b$, where A is an $n \times k$ matrix with rank k. This requires $k \leq n$. Then Y is multivariate normal with parameters $\mu_Y = A\mu_X + b$ and $C_Y = AC_XA^t$. You can derive the covariance formula (if $\mu_X = 0$, $b = 0$, and $\mu_Y = 0$) using $C_Y = E[YY^t] = E[(AX)(AX)^t] = E[A(XX^t)A^t] = AE[XX^t]A^t = AC_XA^t$.

It is easy to generate a multivariate normal $Z \sim \mathcal{N}(0, I)$, just take the components of Z to be independent standard normals. If we take $X = AZ + b$, we get $X \sim \mathcal{N}(b, AA^t)$. This gives $X \sim \mathcal{N}(\mu_X, C_X)$ if $b = \mu_X$ and $C_X = AA^t$. If C_X is known, we can find a suitable matrix A using, for example, the Cholesky factorization. If C is a symmetric positive definite matrix (as any covariance matrix must be), there is a *Cholesky factor* L that is lower triangular and has $C = LL^t$. There is good software in most programming languages to compute Cholesky factors. In C, C++, or Fortran, you can use LAPACK. In Matlab or Python you can use built in functions. The drawback is that Cholesky factors are somewhat expensive to compute in high dimensions. But $n = 1000$ is still practical.

It often happens that you have $H = C^{-1}$ rather than C, an example is in Subsection 6.2. It may be that H is tridiagonal or for another reason has a simple Cholesky decomposition. If $H = MM^t$ is the Cholesky factorization of H, then $H^{-1} = (M^t)^{-1} M^{-1}$. Recall that $(M^t)^{-1} = \left(M^{-1}\right)^t$ and both are written M^{-t}. If we take $X = M^{-t}Z$, then $C_X = M^{-t}(M^{-t})^t = M^{-t}M^{-1} = H^{-1}$, as desired. The equation $X = M^{-t}Z$ is equivalent to $M^tX = Z$. Since M is lower triangular, M^t is upper triangular. The process of finding X from Z and M to satisfy $M^tX = Z$ is called *back substitution*. Any software system that computes Cholesky decompositions (LAPACK, Matlab, Python, ...) also has a procedure to do back substitution. If H is $n \times n$ and not sparse, it takes $O(n^3)$ work to compute the Cholesky factor M and $O(n^2)$ work to do a back substitution.

6 Rejection sampling

6.1 Basic rejection

Rejection sampling converts samples of a *proposal* density, $g(x)$, into samples of a *target* distribution, $f(x)$. If you can sample g, rejection allows you to sample f. It uses an *acceptance probabiltiy* function $A(x)$, which is a probability: $A(x) \in [0,1]$ for each x. A step of rejection sampling uses a sample $X \sim g$. This is the *proposal*. The proposal is *accepted* with probability $A(X)$. If X is accepted, it is the sample of f. If X is rejected (not accepted), you generate a new independent $X \sim g$ and continue. All proposals and acceptance/rejection decisions are independent of each other.

We compute the probability density of an accepted sample. This is defined by $f(x)\,dx = P(X \in [x, x+dx])$, where X is a typical accepted sample. Use Bayes' rule for the distribution of X conditional on acceptance. We let Z be the probability of acceptance in a given try, which is given by

$$\begin{aligned} Z &= P(\text{accept}) \\ &= \int P(\text{accept } X \mid \text{propose } X \in [x, x+dx]) \\ Z &= \int A(x)g(x)\,dx\ . \end{aligned} \tag{6}$$

With this,

$$\begin{aligned} f(x)\,dx &= P(\text{accepted } X \text{ is in } [x, x+dx]) \\ &= P(\text{proposed } X \text{ is in } [x, x+dx] \mid \text{accepted the proposal}) \\ &= \frac{P(\text{proposed } X \text{ is in } [x, x+dx] \text{ and accepted this } X)}{P(\text{accepted})} \qquad \text{(Bayes' rule)} \\ &= \frac{g(x)\,dx \cdot A(x)}{Z}\ . \end{aligned}$$

This leads to the important rejection sampling formula

$$f(x) = \frac{1}{Z} A(x)g(x)\ . \tag{7}$$

You can think of the rejection sampling formula (7) as a thinning process. You get a picture of f starting from g by reducing $g(x)$ by a factor $A(x)$. This changes the shape of the distribution because the reduction factor, $A(x)$, is different in different places. A drawback is that this thinning formula only removes probability, it never adds probability. For example, if $g(x) = 0$ for some x, then $f(x) = 0$. Going further, the *tails* of g (the values of g for large x) must be large enough to create the tails of f. For example, if $g(x) = 2e^{-2x}$ and $f(x) = e^{-x}$ (and $f = g = 0$ when $x < 0$), then the tails of g are too small relative to the tails of f. The formula (7) gives $A(x) = \frac{Z}{2}e^{x}$. This is impossible if $A(x) \leq 1$ for all x.

It is common to denote normalization constants in probability densities by Z, which is a letter physicists use for a partition function. If $h(x) \geq 0$ for all x, then there is a Z so that $f(x) = \frac{1}{Z} h(x)$ is a probability density. The normalization constant is determined by

$$\int f(x)\, dx = 1 \ ,$$

which gives

$$\frac{1}{Z} \int h(x)\, dx = 1 \ ,$$

and therefore

$$Z = \int h(x)\, dx \ .$$

You get the normalization formula (6) using this reasoning on (7).

The *efficiency* of a rejection sampler depends on the expected number of proposals to get an acceptance. Let N be the number of proposals to get the first success. This is a geometric random variable because proposals are independent. The first trial may be an acceptance or rejection. If it is a rejection, the number of subsequent proposals needed is the same as it was before. The probability of rejection is $1 - Z$. Therefore

$$E[N] = 1 \cdot P(\text{accept}) + E[1 + N] \cdot P(\text{reject}) = Z + (1 + E[N])(1 - Z) \ .$$

Solving for $E[Z]$ gives

$$E[N] = \frac{1}{Z} \ .$$

The smaller the acceptance probability, the more proposals you need, on average, to get an acceptance.

The rejection algorithm asks you to make a decision, or *toss a coin*, so that the probability of acceptance is $A(x)$. In general, a *Bernoulli* random variable, a "coin toss", is a random variable B so that $B = 1$ with probability p, and $B = 0$ with probability $1 - p$. You generate B using a uniform U. You accept (set $B = 1$) if $U \leq p$ and reject (set $B = 0$) otherwise.

The rejection algorithm is

```
do  {
   X = gSamp();
  }
until ( uSamp() <= A(X) );  // evaluate uSamp() and A(X)
return X;
```

We assume that every call to `gSamp()` produces an independent sample of g, just as every call to `uSamp()` produces an independent uniform.

Suppose we have decided to sample f by rejection from g. We want to maximize the efficiency of the sampler by making the acceptance probability as

large as possible. The constraint is that $A(x) \leq 1$ for every x, because $A(x)$ is a probability. From (7) we have

$$A(x) = Z \, \frac{f(x)}{g(x)} \, . \tag{8}$$

This implies that we should choose Z as large as possible, consistent with the constraint, so

$$1 = Z \sup \frac{f(x)}{g(x)} \, .$$

This gives

$$Z = \inf \frac{g(x)}{f(x)} \, . \tag{9}$$

For those who are not theoretical mathematicians, we distinguish between inf, for *infemum*, and min, for *minimum*. The minimum is the smallest value. But there may be no smallest value. For example, the function $h(x) = x^{-2}$ does not have a minimum, but the infemum is zero. In probability situations, it is common that the infemum happens as $x \to \infty$. This is related to the tails of f and g.

This formula tells you how to design an efficient rejection sampler. You have a bad sampler, one with small Z, if there is an x that is much more likely in the target distribution than the proposal distribution. It is unfortunate that (9) calls for a worst case analysis rather than an average case analysis. It might be that $\frac{g}{f}$ is reasonable for most x values and yet Z is very small. Exercise 7 is an example where $\frac{g(x)}{f(x)} \to 0$ as $x \to \infty$. This leads to acceptance probability $Z = 0$.

As an example, suppose we want to sample the density $f(x) \propto \sin(\pi x)$ in the interval $[0, 1]$. We find the normalization constant from

$$\int_0^1 \sin(\pi x) \, dx = \frac{-1}{\pi} \cos(\pi x) \Big|_0^1 = \frac{2}{\pi} \, .$$

The target density is $f(x) = \frac{\pi}{2} \sin(\pi x)$. A proposal density whose graph is similar, and that we know how to sample (Subsection 4.3), is $g(x) = 6x(1 - x)$. The efficiency is given by

$$Z = \inf \frac{6x(1-x)}{\frac{\pi}{2} \sin(\pi x)} \, .$$

The inf presumably is achieved in the middle of the interval, $x = \frac{1}{2}$, which gives the value

$$\frac{6 \cdot \frac{1}{4}}{\frac{\pi}{2}} = \frac{3}{\pi} = .955 \, .$$

This rejection sampler is a little better than 95% efficient. During your lifetime practicing Monte Carlo and inventing rejection samplers, it is not likely that you will be able to make one nearly this good for a problem that you care about.

Note that much of the work in evaluating Z goes into calculating and manipulating normalization constants.

6.2 A multivariate example

Many multivariate distributions are approximately normal. Some of them are Gibbs Boltzmann probability densities of the form

$$f(x) = \frac{1}{Z} e^{-\phi(x)/k_B T} \ , \tag{10}$$

where $\phi(x)$ is the energy in a system with configuration x. The parameters are T, the temperature, in degrees above absolute zero, and *Boltzmann's constant* k_B, which is a conversion factor that converts from units of temperature to units of energy. The distribution is often written

$$f(x) = \frac{1}{Z} e^{-\beta\phi(x)} \ ,$$

where $\beta = 1/k_B T$ is the *inverse temperature*. The *partition function*, Z, may be thought of as a function of T, or of β. This discussion assumes that ϕ is a smooth function of x.

Suppose that the energy minimizing configuration, or the *equilibrium* position, is x_0. This means that $\phi(x) > \phi(x_0$ if $x \neq x_0$. A *non-degenerate* equilibrium has Hessian matrix $H = \phi''(x_0)$ that is positive definite. In the limit $T \to 0$ ($\beta \to \infty$), the probability density (10) becomes a point mass at $x = x_0$. Differences between x and x_0, which are allowed only when $T > 0$, are called *thermal fluctuations*. If thermal fluctuations are small, we can approximate ϕ by its second derivative Taylor series about the equilibrium point:

$$\phi(x) \approx \phi(x_0) + \frac{1}{2} (x - x_0)^2 \beta H (x - x_0) \ .$$

This gives rise to the Gaussian approximation to the Gibbs distribution, called *semiclassical*,

$$f(x) \approx g(x) = \frac{1}{Z} e^{-(x-x_0)^t (\beta H)(x-x_0)/2} \ . \tag{11}$$

The equilibrium energy $\phi(x_0)$ has been absorbed into the normalization constant Z. The physical Boltzmann constant has the measured value $k_B = 1.38 \times 10^{-16} \frac{\text{erg}}{\text{deg Kelvin}}$. This is a pretty small number, which is why you don't see things as big as insects perturbed by thermal noise.

It seems like a good idea to use the Gaussian semi-classical approximation (11) as a trial distribution for the exact distribution (10). But can be is hard to put into practice. There is no guarantee that the acceptance probability defined by (9) is different from zero. It is true that $f \approx g$ for points that are likely (according to either f or g). But this need not be true *globally*. One could create a hybrid sampling strategy, sampling points close to x_0 by rejection and sampling points far from x_0 in a more expensive but rarely used way.

The curse of dimensionality works against us here too. The Gaussian approximation to the Gibbs distribution may be a poor trial distribution if the dimension of x is large. Being quite vague (for now), if each dimension contributes an error of size ε, then n dimensions creates an error of size $n\varepsilon$. If the overall error is not small, then f is not close to g in the sense that matters for rejection sampling.

7 Weighted sampling, importance sampling

A *weighted sample* of a density f is a pair of random variables (X, W) so that X, *weighted* by W, has the density f. An informal way to say this is

$$E[W\delta(X-x)] = f(x) \tag{12}$$

for every x. More formally, if $V(x)$ is a bounded continuous function, then

$$\int V(x)f(x)\,dx = E[WV(X)]\ . \tag{13}$$

A weighted sampler does not have to produce $X \sim f$. The weight W compensates for the discrepancy in the X distribution. We write $(X, W) \sim f$ if (12) or (13) are satisfied.

One form of weighted sampling allows you to use a rejection sampler without rejection. You can take $X \sim g$ in (7) and $W = \frac{1}{Z}A(X)$. You can check the property (12) by

$$E[W\delta(X-x)] = E_g[\frac{1}{Z}A(X)\delta(X-x)] = \frac{1}{Z}A(x)E_g[\delta(X-x)] = \frac{1}{Z}A(x)g(x) = f(x)\ .$$

This is because

$$E_g[\frac{1}{Z}A(X)\delta(X-x)] = \frac{1}{Z}\int A(x')\delta(x'-x)g(x')\,dx' = \frac{1}{Z}A(x)g(x)\ .$$

Equivalently, you can check the property (13) using (7):

$$\begin{aligned} E[WV(X)] &= \frac{1}{Z}E_g[A(X)V(X)] \\ &= \frac{1}{Z}\int A(x)V(x)g(x)\,dx \\ &= \int V(x)f(x)\,dx \\ &= E_f[V(X)]\ . \end{aligned}$$

There is a simpler way to do this that does not ask you to know Z. If f and g are two probability densities, their *likelihood ratio* is

$$L(x) = \frac{f(x)}{g(x)}\ . \tag{14}$$

You take $X \sim g$ and $W = L(X)$. The checks we just did verify that this is weight sample of f. In fact, the more complicated method of the previous paragraph has a factor $\frac{1}{Z}$ that compensates the factor Z in A, see (8). This avoids much of the pain in the example of 6.2, which arose from the difficulty of finding Z.

Weighted sampling using the likelihood ratio is called *importance sampling*. It has many uses beyond avoiding the rejection step. One application is *rare*

event simulation. You want to find the probability of something that is very unlikely.

Weighted sampling is more general than ordinary sampling in that it can represent more general functions. f does not have to be a probability density for the formula (12) to makes sense. $f(x)$ does not have to be non-negative. It does not even have to be real valued. For example, we could make weighed samples of a complex function using complex weights.

But many applications do not rejection sampling to be replaced with importance sampling. For example, weighted sampling is hard to use in the single variable heat bath algorithm (Week 3) or in Metropolis proposal distributions. The reason is that if you introduce a weight for each of a long series of sampling steps, then the weights can grow so large that the variance is impossibly large.

8 Monte Carlo estimation and error bars

Error estimation and correctness checking are essential parts of all scientific computing. This is particularly true in Monte Carlo, where the "exact" answer always comes with some noise. Small errors in samplers can be hard to spot unless you do high precision error checking. High precision in Monte Carlo usually entails significant computing time.

Error estimation in Monte Carlo may be thought of as a problem in statistics, and many statistical ideas apply. This is true both for producing error bars in production Monte Carlo runs and for checking correctness of components of Monte Carlo codes.

8.1 Error bars, the central limit theorem

Suppose you are trying to estimate a number A, which is not random. The Monte Carlo estimate is $\widehat{A}$, which is random. An *error bar* is an estimate of the size of the difference between $\widehat{A}$ and A. One more precise version of this idea is related to what statisticians call a *confidence interval*. The interval $[\widehat{A}-\varepsilon, \widehat{A}+\varepsilon]$ is a confidence interval with *confidence* level α if

$$P(A \in [\widehat{A} - \varepsilon, \widehat{A} + \varepsilon]) \geq \alpha \; . \tag{15}$$

In this definition A is not random. The random quantities are $\widehat{A}$ and ϵ. Suppose, for instance, that our code has 95% confidence. Then there is at least a 95% chance that the code will produce numbers $\widehat{A}$ and ε that have the property that $\widehat{A} - \varepsilon \leq A$ and $\widehat{A} + \varepsilon \geq A$. The interval $[\widehat{A} - \varepsilon, \widehat{A} + \varepsilon]$ is the *error bar*. It is often represented as a bar in plots with some symbol in the center of the bar representing $\widehat{A}$. In writing, you can report the error bar as $A = \widehat{A} \pm \varepsilon$. For example, a 95% confidence interval $[4.1, 4.5]$ might be written $A = 4.3 \pm .2$.

The central limit theorem, or *CLT*, gives simple reasonably accurate error bars for most computations involving direct samplers. Suppose you want $A =$

$E_f[V(X)]$ and the estimator is

$$\widehat{A} = \frac{1}{L}\sum_{k=1}^{L} V(X_k)\ ,$$

where the X_k are i.i.d. samples of f. The CLT applies because the numbers $V(X_k)$ are i.i.d. random variables with expected value A. The number of samples, L, is likely to be large enough for the CLT to be valid if we are trying to make an accurate estimate of A. Therefore, $\widehat{A}$ is approximately normal with mean A and variance σ^2/L, where σ^2 is the variance of $V(X)$ with $X \sim f$. The *one standard deviation error bar* is a confidence interval with ε equal to the standard deviation of $\widehat{A}$, which is $\varepsilon = \sigma/\sqrt{L}$. According to the CLT, the confidence of this error bar is $\alpha = 68\%$. The custom in scientific Monte Carlo is to report such one standard deviation error bars. Others may prefer to give two standard deviation error bars $\varepsilon = 2\sigma/\sqrt{L}$. This gives 95% confidence error bars.

Usually σ^2 is unknown and must be estimated from Monte Carlo data. The standard estimator of σ^2 is

$$\widehat{\sigma^2} = \frac{1}{L}\sum_{k=1}^{L}\left(V(X_k) - \widehat{A}\right)^2\ . \tag{16}$$

It is common to use $1/(L-1)$ rather than $1/L$ here. But if that makes a difference you probably don't have enough data to estimate A accurately, or to use error bars based on the CLT. If you use (16) instead of σ^2 in the error bar formulas, the α values will only be approximate. But even if you used the exact σ^2, the CLT is only a large L approximation. A wise and practical Monte Carlo expert says: "Don't put error bars on error bars." The purpose of an error bar is to know about how accurate $\widehat{A}$ is. Suppose $\widehat{A} = 2958$ and the 68% error bar is $\varepsilon = 2.543$. There doesn't seem to be much harm in reporting $A = 2958 \pm 2.3$, even though the error bar is off by 10%.

It is absolutely unprofessional to do a Monte Carlo computation without quantitative reasonably accurate error bars. You don't have to report error bars to non-technical people who would not appreciate them. But you do have to know how big they are, and to report them to any consumer of your results who has the technical training to know what they mean. For every computational assignment in this course, reasonable error bars are part of the assignment.

8.2 Histograms, verifying a sampler

All programming is error prone, and particularly scientific computing. And within scientific computing, particularly Monte Carlo. You need to verify each Monte Carlo procedure carefully. Whenever you write a sampler, you need to verify it before you put it into a larger Monte Carlo code. As with error bars, verifications are a part of every computing assignment in this class. Not

just verifications of final results, but separate verifications of the component subroutines.

The histogram is a practical way to verify most direct samplers of one component random variables. A histogram divides the real axis into *bins*, $B_j = [x_j - \frac{\Delta x}{2}, x_j - \frac{\Delta x}{2})$. Here Δx is the bin size, and $x_j = j\Delta x$ is the bin center. The bin as written contains its left endpoint but not its right endpoint. Mathematically, this is irrelevant as long as the probability density is continuous. But computational floating point numbers are discrete and may sometimes land on endpoints. If the samples are $X_1, \ldots, X_L$, the *bin counts* are $N_j = \#\{X_k \in B_j\}$. N_j is the number of samples in bin B_j. A *histogram* is a graph of the bin counts. It is traditional to plot bin counts using a bar graph, but there is no scientific reason do do that.

If the X_k are samples of a probability density f, then the expected bin counts are

$$n_j = E[N_j] = Lp_j = L \int_{B_j} f(x)\, dx \; .$$

Here, p_j is the probability that a particular sample lands in B_j. In practice, it often suffices to approximate the integral by $p_j \approx \Delta x f(x_j)$, but there is no scientific reason to do this. The cost of doing the integrals more accurately is trivial compared to the cost of generating the samples.

Since $n_j \neq N_j$, you have to have an idea how much difference to expect. You need error bars for bin counts. Bin counts are binomial random variables because N_j is the sum of L independent Bernoulli random variables with the same p_j. The variance of N_j, therefore, is $\sigma^2_{N_j} = Lp_j(1-p_j)$. You can estimate p_j from the empirical bin count

$$p_j \approx \widehat{p}_j = \frac{N_j}{L} \; .$$

This is accurate if N_j is more than just a few, which it will be for lots of bins if Δx is not too small and L is large.

The histogram verification procedure would be:

1. Generate a large number of samples $X_1, \ldots, X_L$.
2. Calculate the bin counts N_j for a range of x_j containing most of the probability.
3. Calculate the error bars for N_j using $\varepsilon_j = \widehat{p}_j(1-\widehat{p}_j)\sqrt{L}$.
4. Calculate the expected bin counts n_j.
5. Graph $N_j \pm \varepsilon_j$ and n_j in the same figure. Roughly a third of the n_j should be outside the error bars.

It is a good idea to take Δx somewhat small and L very large so that you get a picture of f with error bars as small as possible.

9 Examples and exercises

1. Suppose you want $X \in \mathbb{R}^n$ uniformly distributed on the unit $n-1$ dimensional sphere. This is the same as asking for a unit vector $\|X\|_{l^2} = 1$ whose probability distribution is isotropic. You can do this by starting with any isotropic probability distribution and normalizing. Let $Z = (Z_1, \ldots, Z_n)^t$ where the Z_k are independent one dimensional standard normals (made, for example, by Box Muller). Then Z is an n dimensional standard normal with isotropic probability density $f(z) = Ce^{-\|z\|^2/2}$. The normalized random variable $X = \frac{1}{\|Z\|} Z$ is both normalized and isotropic, as desired.

2. Suppose you want X uniformly distributed in the unit ball. One approach would be to take X uniformly distributed in the cube that contains the ball. That would be $X_k = 2U_k - 1$, where the U_k are i.i.d. standard uniforms. You can then accept X if it is inside the unit ball, $\|X\| \leq 1$ and reject otherwise. Eventually you will get an acceptance. The excepted X is uniformly distributed in the unit ball. Each proposal is simple and cheap. The efficiency of the overall algorithm depends on its acceptance probability, Z. Show that Z is exponentially small in n. The conclusion is that generating a uniform in the ball by rejection from a uniform in the cube is an exponentially bad idea. *One approach*: there is a formula for the volume of the unit ball in n dimensions. The cube has side 2 and volume 2^n. The ratio of these volumes is the acceptance probability. You will need to use an asymptotic approximation of the Gamma function, such as $\Gamma(n) = (n-1)! \approx (n-1)^{n-1} e^{-n-1}$. *Another approach*: If X_k is uniform in $[-1, 1]$ then $E\left[X^2\right] = \frac{1}{3}$. Therefore, in n dimensions, $E\left[\|X\|^2\right] = \frac{n}{3}$. Cramer's theorem from large deviation theory implies that $P\left(\|X\|^2 \leq 1\right)$ is exponentially small.

3. Another way to generate X uniform in the unit ball is to write $X = RY$, where $R = \|X\| \in [0, 1]$, and Y is uniform on the sphere. Think of this as working in spherical coordinates. Exercise 1 lets you generate Y. R is a scalar whose CDF is $F(r) = Cr^n$ ($P(R \leq r)$ is proportional to the volume of the ball of radius r.). The constant is found from $1 = F(1) = C$. The CDF inversion method gives R with the desired CDF by solving $F(R) = U$. In this case, that is just $R = U^{1/n}$.

4. Let $K(x)$ be the Green's function for the Debye Hückel operator. This satisfies $\triangle K(x) - mK(x) = \delta(x)$. Since K is negative and decays exponentially, $-K$ can be normalized to be a probability density. The normalization constant may be found by integrating both sides over $\mathbb{R}^n$:

$$\int \triangle K(x)\, dx - m \int K(x)\, dx = \int \delta(x)\, dx$$

$$m \int (-K(x))\, dx = 1 \, .$$

To sample the probability density $f(x) = \frac{-1}{m}K(x)$, you choose an exponential T with rate constant $\lambda = m$, then you take $X \sim \mathcal{N}(0, mI)$.

5. Suppose $R > 0$ has probability density $f(r) = re^{-r^2/2}$. Show that the CDF is $F(r) = 1 - e^{-r^2/2}$. Show that if you solve the equation $F(R) = U$, you get a sampler for f that is equivalent to (3).

6. (*easy*) Show that the formula (9) gives $Z \leq 1$ for any pair of probability densities f and g. Note that there are x values with $\frac{g(x)}{f(x)} > 1$. The problem is to show that $\frac{g(x)}{f(x)} \leq 1$ for some x provided that f and g are probability densities.

7. Show that it is not possible to generate an exponential random variable by rejection from a gaussian.

8. Suppose we have a direct weighted sampler of a probability density g. This means that there is a procedure so that `[X,W] = gSampW()` produces an independent weighted sample of g in the sense of (12) or (13).

   (a) Does the rejection method of Subsection 6.1 turn (X, W) into a weighted sample of f?

   (b) Suppose $L(x) = \frac{f}{g}$ is the likelihood ratio. Is $(X, WL(X))$ a weighted sample of f?

9. Describe the mechanics of using the CLT to estimate error bars when you are using a weighted direct sampler of f. Describe how to create "weighted" histograms to verify that (X, W) is a weighted sample of f.