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

Class notes: Monte Carlo methods
Week 2, Simulation
Jonathan Goodman
February 5, 2013

1 Introduction

Simulation, in this Monte Carlo course, will mean *direct simulation*. That means producing random objects according to some description. This was supposed to be almost the opposite of Monte Carlo, where the main activity is finding ways to "cheat" – to find different random variables with the same expectation value. This chapter is here for three reasons. One is that the moderately complex random objects here are great examples of things to do "Monte Carlo" tricks on – variance reduction and so on. Another is that there are interesting tricks already here that people doing computational probability (or *computational stochastics* use all the time.

The final reason is possibly perverse pedagogical principle, the primacy of programming in the present applied mathematics paradigm.[1] In other words, you don't do applied mathematics in the 21-st century without computing. A class in applied mathematics has to have algorithms and programs. Simulation is a setting with mostly simple math where it is easy to do some more serious programming.

2 Basic operations

Much of this week's material is about simulating random processes. Two fundamental operations for that are "ringing a bell" and choosing from a finite list.

By *ringing a bell*, we mean sampling an exponential random variable $T \sim \lambda e^{-\lambda t}$. Last week we saw that this can by done by setting $T = \frac{-1}{\lambda}\log(U)$, where U is uniformly distributed in $[0,1]$.

Choosing from a list means choosing one of the numbers $\{1,\ldots,n\}$ so that $P(N=j) = p_j$. We assume the numbers $p_1,\ldots,p_n$ are known, $p_j \geq 0$ for all j, and $p_1 + \cdots + p_n = 1$. A general and reasonably efficient way to do this is a discrete version of the CDF sampling method from last week. The discrete CDF is

$$F_j = P(N \leq j) = \sum_{i=1}^{j} p_i \ .$$

This makes $F_1 = p_1$, $F_2 = p_1 + p_2$, etc. It also makes $0 = F_0 \leq F_1 \leq F_2 \leq \cdots \leq F_{n-1} \leq F_n = 1$. The numbers F_j partition the unit interval into n disjoint

[1] I plead your pardon for this pointless pile of "p" words.

pieces whose length is $F_j - F_{j-1} = p_j$. If $U \in [0,1]$ is uniformly distributed, then $p_j = P(F_{j-1} \le U < F_j)$. The algorithm is: generate U, then N is the unique number with $F_{N-1} \le U < F_N$.

There are different ways to code this depending on n and how the sampler will be used. The simple and slow way is[2]

```
//  Choose N = j with probability p[j], for j = 0, ..., n-1
//  Assume p[0] + ... + p[n-1] = 1.

   double p[n];    // The given probabilities.
   double U;       // The uniform [0,1] random variable
   int    N;       // The desired random number
   double F;       // The discrete CDF

   U = uSamp();    // a uniform random number generator
   N = 0;
   F = 0.d0;       // double precision zero is different from integer zero

   while ( F < U ) // Continue until you find the appropriate interval
      F += p[N++]; // First add p[N], then increment N.

   N--;            // Undo the last N++ to get N in (0,1,...,n-1)
```

This algorithm uses $O(n)$ additions and comparisons to generate the random number N. There probably is not a better generic method if you want just one such N. But Monte Carlo and simulation applications often call for a large number of samples from a given distribution. *Binary search* (also called *bisection search*) find an element in a sorted list using $\log_2(n)$ probes. If you want many samples N_k from the same discrete distribution, you can compute and store the numbers F_j, which takes n operations. Then you can find N_k from U_k using binary search, which is an exra $\log_2(n)$ operations per sample.

A good way to program binary search in C++ is to use the *STL* (*standard template library*). This is a collection of elementary data structures and algorithms. The STL approach to binary search is described in

`http://www.codecogs.com/reference/computing/stl/algorithms/binarysearch/binary_search.php`

Here is another potential improvement, but there are not many situations where this one helps much. Suppose some of the probabilities p_j are much larger than others. In particular, suppose there are a few large p_j and many small ones. You could take advantage of that by sorting the p_j so that $p_1 \ge p_2 \ge \cdots$. Then the loop `while ( F < U ) F += p[N++];` would be likely to terminate earlier. But this idea is that it is not likely to beat the $O(\log(n))$ binary search strategy.

[2]Students are discouraged from programming in the style of the statement `F += p[N++];`. But it is so common that most C/C++ would recognize it immediately.

The initial sort, which you also would use the STL for, takes $O(n \log(n))$ steps. The extra $\log(n)$ is not a big price to pay for a big speedup, but it is unnecessary if you only want one N, and probably uncompetitive if you want many N samples. Exercise 1 gives a combination of these ideas that I conjecture is the best possible for some theoretical reason (I have not even formulated the conjecture precisely).

3 Discrete Markov chains

A stationary Markov chain with states $\{1, \ldots, n\}$ is described by its *transition matrix* R. At each time $t = 0, 1, 2 \ldots$, the state of the chain is a random variable X_t which is one of the n states. The entry R_{ij} is the $i \to j$ transition probality

$$R_{ij} = P(X_{t+1} = j \mid X_t = i) \ .$$

The definition of a stationary (or time homogenous) Markov chain is that you get all probabilities from these by multiplication. If $X_0 = x_0$ is known and not random, the probability of the sequence $(X_1, \ldots, X_T)$ is

$$P(X_1, X_2, \ldots, X_T) = \prod_{t=0}^{T-1} R_{X_{t-1}, X_t} \ .$$

This is equivalent to the *Markov property*

$$\begin{aligned} &P(X_{t+1} = j \mid X_t = i) \\ &\quad = P(X_{t+1} = j \mid X_t = i \text{ , and } X_{t-1} = i_1 \text{ , } \cdots \text{ , } X_1 = i_{t-1}) \ . \end{aligned}$$

You can simulate a discrete time finite state space Markov chain using the discrete selection algorithm from Section 2. If you know $X_t = i$, let $N = j$ with probability P_{ij} and take $X_{t+1} = N$. It is unlikely that you will ever meet a Markov chain so that this is the best simulation method. Either n is too large for this to be practical, or the chain has special structure that makes the generic sampling method unnecessary.

4 Continuous time processes

We say X_t is a *continuous time process* if X_t is defined for every real number t (or every t in some range). You cannot simulate a continuous time process exactly, if only because there are infinitely many t values in any interval. Instead, we usually make a *discrete time approximation*, which is a discrete time process X^h with values X^h_k. The *time step* is h or Δt. The discrete times are $t_k = k\Delta t$. The discrete time path is $X^h_1, \ldots, X^h_N$. This is supposed to approximate the values of the continuous time path at the same discrete times, which are $X_{t_1}, \ldots, X_{t_N}$. The simulation is not likely to be perfect, which means that the distribution of X^h_N is not exactly the same as the distribution of X_{t_N}.

Suppose we want to simulate X_t for $t \in [0, T]$. If the time step is Δt, then the number of steps is $n = T/\Delta t$. We hope the distribution of X_n^h converges to the distribution of X_T as $\Delta t \to 0$. Unfortunately, the number of time steps goes to infinity as well.

An *exact* simulation strategy is one that produces X_n^h (or some related values, see below) whose distribution is exactly the same as the corresponding X_{t_n}. The discrete time path still is not exactly the same as the continuous time path, because the discrete time path has been "simulated" only at a finite number of times. Still, it is usually good to do exact simulation if it is practical.

4.1 Continuous time Markov chains

A discrete state space continuous time Markov chain is described by its *transition rate matrix*, R. If $i \neq j$, then

$$P(X_{t+dt} = j \mid X_t = i) = R_{ij}\, dt \ .$$

You can say this more formally as $P(X_{t+\Delta t} = j \mid X_t = i) = R_{ij}\, \Delta t + O(\Delta t^2)$.

A direct approach to simulation with a finite Δt is to take

$$P(X_{k+1}^h = j \mid X_k^h = i) = R_{ij}\, \Delta t \ ,$$

for $i \neq j$. This makes X_k^h a discrete time Markov chain whose transition matrix is $R_{ij}^h = R_{ij} \Delta t$ if $i \neq j$. Of course, this makes no sense unless Δt is small enough. The diagonal entries of the discrete time approximation transition matrix are

$$R_{ii}^h = 1 - \left(\sum_{j \neq i} R_{ij} \right) \Delta t \ .$$

This goes to 1 as $\Delta t \to 0$, so the probability of a transition in a given step is small. See Exercise 2 for a more efficient way to simulate the approximate chain. An example of this approximation and its error is in Exercise 3.

There is an exact method for simulating a continuous time Markov chain. You simulate the sequence of states using an appropriate transition probability matrix. This is called the *embedded* discrete time Markov chain. The transitions times are exponentials, with rates that some from the rate matrix. The whole method is sometimes called *Gillespie's stochastic simulation algorithm*, or *SSA*. This is a bit of a misnomer because the algorithm was widely used already when Gillespie proposed it. The method also is called *event driven simulation*. That too is a bit off, as event driven simulation is a more sophisticated idea.

The "right" view of the exponential random variable makes the method is almost obvious. Suppose S_1, S_2, ..., are independent exponential random variables with rate constant λ. Define the *arrival times* T_k as $T_0 = 0$, $T_1 = S_1$, and in general $T_k = T_{k-1} + S_k$. The S_k are the *inter-arrival times* for the T_k. The collection of times T_k form a *Poisson processes* with rate constant λ. The Poisson process has a simple description – there is an arrival in interval $(t, t+dt)$

with probability λdt, and disjoint intervals are independent. You can make a discrete time approximation to the Poisson process by choosing a small Δt and dividing time into intervals $(t_j, t_{j+1}) = (t_j, t_j + \Delta t)$. Let Y_j be a $0, 1$ random variable, with $Y_j = 0$ meaning that there was no arrival in (t_j, t_{j+1}), and $Y_j = 1$ meaning that there was an arrival. These are the only choices, we never have more than one arrival per small interval in the approximate Poisson process. All arrivals are independent. The probability that $Y_j = 1$ is $\lambda \Delta t$. Let $j_1 < j_2 < \cdots$ denote the intervals with arrivals $Y = 1$. The approximate arrival times are $T_k^h = j_k \Delta t$.

A Poisson process is something like light rain falling on dry ground. The each small piece of ground has a small probability of getting a drop. Disjoint pieces of ground are independent.

Now suppose $Y_j^{(1)}$ and $Y_j^{(2)}$ are the binary random variables corresponding to independent Poisson processes with rate parameters λ_1 and λ_2 respectively. Set $Y_j = 1$ if either $Y_j^{(1)} = 1$ or $Y_j^{(2)} = 1$, or both. This is almost the same as $Y_j = Y_j^{(1)} + Y_j^{(2)}$, because it is very unlikely that one small interval gets visits from both processes. (The probability for a given interval is $O(\Delta t^2)$. The number if intervals is $O(1/\Delta t)$, so the probability that there is an interval with visits from both processes is $O(\Delta t)$.) The probability that $Y_j = 1$ is (approximately) $\lambda_1 \Delta t + \lambda_2 \Delta t = (\lambda_1 + \lambda_2)\Delta t$. Of course, the variables Y_j for distinct j are independent. Therefore, Y_j is an approximation to the Poisson process with rate parameter $\lambda = \lambda_1 + \lambda_2$.

This is supposed to make obvious the fact that the superposition of independent Poisson processes is a Poisson process. The rate constant of the superposition is the sum of the individual rate constants. In the rain picture: the sum of rain with intensity λ_1 and rain with intensity λ_2 is rain with intensity $\lambda_1 + \lambda_2$.

Now suppose $T_1^{(1)} = S_1^{(1)}$ and $T_1^{(t)} = S_1^{(2)}$ are the first arrivals for each process. The first arrival for the superposition is $S_1 = T_1 = \min(S_1^{(1)}, S_1^{(2)})$. This "proves" a basic fact about exponential random variables. If $S^{(1)}$ and $S^{(2)}$ are independent exponential random variables with rate constants λ_1 and λ_2 respectively, then $S = \min(S^{(1)}, S^{(2)})$ is an exponential with rate parameter $\lambda = \lambda_1 + \lambda_2$. It is easy to see this by direct computation, see Exercise 4. It is also easy to calculate that the probability that the first arrival is T_1 is $\lambda_1/(\lambda_1 + \lambda_2)$. To see this informally, note that it is (nearly) the same as

$$P(Y_j^{(1)} = 1 \mid Y_j^{(1)} = 1 \text{ or } Y_j^{(2)} = 1) = \frac{\lambda_1 \Delta t}{\lambda_1 \Delta t + \lambda_2 \Delta t}$$

Suppose a continuous time Markov chain has $X_t = i$, and you want to know where and when the next transition will be. The transition rates are $\text{rate}(i \to j) = R_{ij}$. If $X_t = i$, then the total rate of transitions out of i is

$$\sum_{j \neq i} R_{ij} = -R_{ii} \; . \tag{1}$$

The probability that the transition is to j is

$$\widetilde{R}_{ij} = \frac{R_{ij}}{\sum_{k \neq i} R_{ik}} \ . \tag{2}$$

You can check that

$$\sum_{j \neq i} \widetilde{R}_{ij} = 1 \ ,$$

which means that there is zero probability of not moving in the embedded discrete time Markov chain. The embedded discrete time chain records only jumps, not non-jumps.

Here is a description of the exact simulation algorithm. The continuous time state at time t is X_t. The embedded discrete time Markov chain after k transitions is $\widetilde{X}_k$. The transitions are at times T_k. The inter-transition times are S_k.

- Start with $k = 0$, and $T_0 = 0$, and $X_0 = \widetilde{X}_0 = x_0$
- While $T_k < T_{\text{final}}$
  - Set $i = \widetilde{X}_k$.
  - Choose S_k exponential with rate parameter $\lambda_k = -R_{ii}$ in (1).
  - Set $T_{k+1} = T_k + S_k$
  - $X_t = \widetilde{X}_k$ when $T_k \leq t < T_{k+1}$
  - Choose j with probability $p_j = \widetilde{R}_{ij}$.
  - Set $\widetilde{X}_{k+1} = j$
  - $k \leftarrow k + 1$

4.2 Brownian motion

Diffusion processes are continuous time random processes that have *continuous sample paths* (i.e., X_t is a continuous function of t). Brownian motion is the most basic diffusion process. To describe it, we use the notation that if $0 \leq a \leq b$ are two times, then $X_{[a,b]}$ is the part of the path for t between a and b. In particular, $X_{[0,a]}$ is the path from the initial point up to time a. The *standard* Brownian motion has the following properties:

- $X_0 = 0$
- X_t is a Markov process. This means that the distribution of $X_{[a,T]}$ conditional on X_a is the same as the distribution of $X_{[a,T]}$ conditional on $X_{[0,a]}$ (of course $0 \leq a \leq T$).
- The *increment* is Gaussian. This means that if $t > a$, then $X_t - X_a$ is normal with mean zero and variance $t - a$. This is the conditional distribution given X_a.

- X_t is a continuous function of t.

The *independent increments* property is as follows. Suppose $0 \leq t_1 < t_2 < \cdots < t_n$. The corresponding increments are $Y_j = X_{t_{j+1}} - X_{t_j}$. Let $Y = (Y_1, \ldots, Y_{n-1})$. Then Y is a multivariate normal such that Y_j is independent of Y_k if $j \neq k$ (the independent increments property), and $Y_j \sim \mathcal{N}(0, t_{j+1} - t_j)$.

The specifications of the distributions of the increments is self-consistent in the following sense. Suppose an increment from time t_1 to time t_3 is broken up into two increments with time t_2 in between. The increment for the full interval is $Y_1 = X_{t_3} - X_{t_1}$. The two smaller increments are $Y_2 = X_{t_2} - X_{t_1}$ and $Y_3 = X_{t_3} - X_{t_2}$. The specifications say that Y_2 and Y_3 are independent, and that $Y_2 + Y_3 = Y_1$. This is possible only if $\text{var}(Y_2) + \text{var}(Y_3) = \text{var}(Y_1)$. But the respective variances are equal to the lengths of their corresponding time intervals: $\text{var}(Y_2) = t_2 - t_1$, $\text{var}(Y_3) = t_3 - t_2$, which do add up.

Brownian motion is a model of a function that is the sum of infinitely many i.i.d. increments that are each infinitely small. X_t is the sum of all the increments up to time t. This makes the variance proportional to the number of terms, which is proportional to t. Standard Brownian motion is normalized to that the variance is equal to t. An approximation to Brownian motion would have a large number of small ili.d. increments. The CLT suggests that the macroscopic increments Y_j should be Gaussian. The macroscopic increments Y_j are independent because the microscopic increments they are made of are independent.

Brownian motion is a Markov process. Let $X_{[0,s]}$ be the path from time 0 to time s. The Markov property is that the conditional distribution of X_t for $t \geq s$ is the same whether you condition on X_s or on $X_{[0,s]}$. This is because of the independent increments property. All the increments that arrive after time s are independent of anything that happened up to time s.

You cannot make a Brownian motion sample path in the computer because it is a continuous time process. But if you choose a fixed time step Δt, you can make a vector that has the exact distribution of $\vec{X}_n = (X_{t_1}, \ldots, X_{t_n})$. You just take $X_{t_{k+1}} = X_{t_k} + \sqrt{\Delta t} Z_k$, where $Z_k \sim \mathcal{N}(0, 1)$. Since the variances of the increments are consistent, any increment $X_{t_k} - X_{t_j}$ will have the correct distribution, which is normal mean zero, variance $t_k - t_j$.

The *Brownian bridge construction* is a way to sample $\vec{X}_n$ that is useful for variance reduction. It is a *multiscale* algorithm in that it generates increments separately for different time scales. Suppose we want a path up time time $T = 1$ with $n = 2^L$ observations. This means $\Delta t = 2^{-L}$. On *level* k, we will finalize increments for time scale $\Delta t_k = 2^{-k}$. We start on the *coarsest* scale, which corresponds to $k = 0$ and $\Delta t_0 = 1$. Then we work from scale $k - 1$ to scale k, refining each time interval on level $k - 1$ by a factor of two to get two level k time intervals. We are done when we are done with level L.

The increment for level zero is just the value $X_1 = X_{t_n}$, which is $X_1 \sim \mathcal{N}(0, 1)$. At the next level, $k = 1$, we determine $X_{\frac{1}{2}} = X_{t_{\frac{n}{2}}}$. We need to sample $X_{\frac{1}{2}}$ from its conditional distribution knowing X_1 (already sampled) and $X_0 = 0$.

We do this from the joint density, which is

$$f(x_{\frac{1}{2}}, x_1) = \frac{1}{Z} \exp\left[\frac{-1}{2}\left(\frac{x_{\frac{1}{2}}^2}{1/2} + \frac{(x_1 - x_{\frac{1}{2}})^2}{1/2}\right)\right] . \tag{3}$$

This is the product of two factors. One is the probability density of $X_{\frac{1}{2}}$, which is normal mean zero variance $\frac{1}{2}$, which is

$$f(x_{\frac{1}{2}}) = \frac{1}{Z} \exp\left[\frac{-1}{2} \cdot \frac{x_{\frac{1}{2}}^2}{1/2}\right] .$$

The other factor is the conditional density of X_1 given $X_{\frac{1}{2}}$. The increment, $X_1 - X_{\frac{1}{2}}$ is normal mean zero variance $\frac{1}{2}$, so

$$f(x_1 \mid x_{\frac{1}{2}}) = \frac{1}{Z} \exp\left[\frac{-1}{2} \cdot \frac{(x_1 - x_{\frac{1}{2}})^2}{1/2}\right] .$$

Note two conventions: (1) f denotes any probability density, (2) Z is any normalization constant, not the same in different places. The conditional density $f(x_{\frac{1}{2}} \mid x_1)$ is also given by the right side of (3).

We sample this conditional distribution by figuring out what Gaussian it is. The exponent is quadratic in $x_{\frac{1}{2}}$, which we would like to write as

$$\frac{(x_{\frac{1}{2}} - \mu)^2}{\sigma^2} .$$

You can find σ^2 by asking what happens as $x_{\frac{1}{2}} \to \infty$. By inspection of (3), this is

$$\frac{x_{\frac{1}{2}}^2}{1/2} + \frac{x_{\frac{1}{2}}^2}{1/2} = \frac{x_{\frac{1}{2}}^2}{1/4} .$$

This shows that the conditional variance of $x_{\frac{1}{2}}$ is $\frac{1}{4}$. We find μ by minimizing the exponent in (3) over $x_{\frac{1}{2}}$.

$$\min_{x_{\frac{1}{2}}}\left[x_{\frac{1}{2}} + (x_1 - x_{\frac{1}{2}})^2\right] \overset{\partial_{x_{\frac{1}{2}}}}{\Longrightarrow} x_{\frac{1}{2}} - (x_1 - x_{\frac{1}{2}}) = 0 \implies x_{\frac{1}{2}} = \frac{1}{2} x_1 .$$

Therefore the conditional mean of $x_{\frac{1}{2}}$ is $\frac{1}{2}x_1$ and the variance is $\frac{1}{4}$. This lets us sample $X_{\frac{1}{2}}$.

The conditional distribution of $X_{\frac{1}{2}}$ is that its mean is the average of the values at the endpoints of its interval, X_0 and X_1, and its variance is $\frac{1}{2}\Delta t_1$. The conditional sampling on finer levels is the same. Suppose we have sampled on level $k-1$ and want to sample at level k. The level k time step is $\Delta t_k = 2^{-k}$. From level $k-1$ sampling, we know $X_{j\Delta t_k}$ if j for even values of j. The task at level k is to sample $X_{j\Delta t_k}$, conditional on $X_{(j+1)\Delta t_k}$ and $X_{(j-1)\Delta t_k}$. We can

do the algebra as above. The conditional expectation of $X_{j\Delta t_k}$ is the average of the endpoint values: $\frac{1}{2}\left(X_{(j+1)\Delta t_k} + X_{(j-1)\Delta t_k}\right)$. The conditional variance is $\frac{1}{2}\Delta t_k$.

Here is a slightly different description of this Brownian bridge construction. The *hat function* is

$$h(t) = \begin{cases} t & \text{if } 0 \le t \le 1 \\ 1-t & \text{if } 1 \le t \le 2 \\ 0 & \text{otherwise.} \end{cases}$$

The level $k = 0$, approximate Brownian motion path is

$$X_t^{(0)} = Z_{10}\, h(t) \ .$$

In this paragraph, all Z variables are standard normals. For $t \in [0,1]$, this is the straight line path from $X_0 = 0$ to $X_1 =$ *standard normal*. The approximate path at level $k = 1$ is

$$X^{(1)}(t) = X^{(0)}(t) + \sigma_1 Z_{11} h(2t) \ ,$$

with $\sigma_1^2 = \frac{1}{2}\Delta t_1 = \frac{1}{4}$. The values of $X_t^{(1)}$ at $t = \frac{1}{2}$ are the same as the recipies for $X_{\frac{1}{2}}$ and X_1 in the earlier description of the Brownian bridge. This is because $h(2t) = 1$ when $t = \frac{1}{2}$, and $h(2t) = 0$ when $t = 1$. In general, suppose we have $X_t^{(k-1)}$, then

$$X_t^{(k)} = X_t^{(k-1)} + \sum_{j=1, j\,odd}^{2^k-1} \sigma_k Z_{jk} h(2^k(t - j\Delta t)) \ .$$

4.3 Multi-dimensional Brownian motion

Multi-dimensional Brownian motion is a model of a particle diffusing in a liquid or gas. This is different from *advection*, which is motion of the particle caused by fluid flow. Brownian motion was observed by looking at tiny particles in in a liquid with a light microscope. Einstein's explanation is that the motion is due to the particle being bombarded by individual liquid molecules. The wikepedia page has a fuller account. For us, what matters is that

1. Brownian motion is homogeneous and isotropic. The statistical properties of the random motion are the same wherever the particle is and however the medium is rotated and however much time has passed.

2. Particle displacements at different times may be treated as independent.

3. The Brownian motion path X_t is a continuous function of t.

The second property is not exactly in the microscopic motion of the particle. It is an idealization that describes reasonably well particle motions you can see, which are the large scale motions.

The properties of Brownian motion above imply (take a different class for technical details) that the increments of Brownian motion are independent isotropic normals with mean zero (the only isotropic mean is zero). To be consistent, the variances must be proportional to t. This may be expressed in terms of a *diffusion coefficient*, D. Each of the components of X_t satisfy

$$E[(X_{k,t} - X_{k,0})^2] = Dt \ . \tag{4}$$

Since the Brownian motion is isotropic, the covariance at time offset t is

$$DI = E[(X_{s+t} - X_s)(X_{t+s} - X_s)^t] \ .$$

The variance and covariance properties of Brownian motion imply that the components $X_{k,t}$ are independent one dimensional Brownian motions. Therefore, we can simulate a Brownian motion path component by component.

Suppose the diffusing Brownian motion particle sticks to the boundary of a domain the first time it touches. To describe this, suppose Γ is some (closed) set, and the corresponding *hitting time* is

$$\tau = \min \{t \mid X_t \in \Gamma\} \ .$$

(Mathematicians often use Ω for a domain where a PDE will be satisfied, and Γ for the boundary of this domain.) The hitting point is a random point in Γ. There is a probability distribution describing this random hitting point

$$\mu(A) = P(X_\tau \in A) \ .$$

If A is a piece of Γ, the probability that the particle hits within A is The measure μ is called *harmonic measure*. The term "harmonic" is because of the relation of this measure to functions that satisfy $\triangle u = 0$, which are *harmonic* functions. The harmonic measure is particularly interesting if Γ has irregularities such as ridges or points. The probability density of harmonic measure becomes infinite at a point or ridge that protrudes into the diffusion domain. That is because diffusion is particularly likely to hit these features. The density goes to zero at ridges that point out of the diffusion domain. This is because a diffusing particle is more likely to hit a wall near this ridge than to hit the ridge itself. These intuitions, which are backed by mathematical theorems, become stronger as you learn more about Brownian motion.

To be more specific, suppose $X_0 = 0$, and $D = 1$ (standard multi-variate Brownian motion). The simulation is less subtle if the origin is completely enclosed by Γ so the expected hitting time is finite. Direct simulation can study the harmonic measure – just generate many Brownian motion paths and record where they first touch Γ. In practice, you would choose a time step Δt and keep simulating $X_{t_n+\Delta t} = X_{t_n} + \sqrt{\Delta t} Z$ (each Z being an independent multivariate standard normal) until X_n is within some ε distance of Γ. We then record the hitting point as being the nearest point in Γ to X_n.

There are several sources of bias in this approach. The actual hitting time will not be one of the discrete times $t_n = n\Delta t$. There is a difference between

being within ε of Γ and being on Γ. In fact, if ε is too small relative to $\sqrt{\Delta t}$, the computed approximate harmonic measure will be inconsistent with the true harmonic measure (details omitted).

There is a method related to what is called *kinetic Monte Carlo* that estimates the harmonic measure with less work. It is based on the the following observation: Suppose we observe a Brownian motion to be located at x_0 at time t_0. Suppose a ball of radius r_0 about x_0 is completely contained in the diffusion domain. Let T_1 be the hitting time for the boundary of this ball:

$$T_1 = \min \{t > t_0 \text{ such that } |X_t - X_0| = r_0\} \ .$$

Then X_{T_1} is uniformly distributed on the surface of this ball. We can simulate X_{T_1} simply by sampling uniformly from this surface. An exercise from Week 1 explains how to do this. For harmonic measure, T_1 is not important, only X_{T_1}. This is a much more efficient way to simulate X_{T_1} than by generating a path. And it is unbiased; it gets the distribution of X_{T_1} exactly.

The algorithm for simulating a hitting location is simple: start with $X_0 =$. If you have simulated X_{T_k}, choose r_k as small as possible so that the ball of radius r_k just touches Γ. This is the distance from X_{T_k} and Γ. Simulate a point $X_{T_{k+1}}$ uniformly on the surface of this ball. Repeat until $r_k \leq \varepsilon$. Not only is this method faster. It has fewer sources of bias because it has fewer parameters. The simulated harmonic measure converges to the actual harmonic measure as $\varepsilon \to 0$.

There is a faster but more complicated version of this method that does not require the ball at step k to have X_{T_k} in the center. This relies on an explicit formula, the *Poisson kernel* for the harmonic measure of the surface of a ball for a particle starting at any given point inside the ball It is not hard to sample the hitting times T_k, which are needed for other applications of this general idea.

5 Examples and exercises

1. This is a fancier method for sampling from a finite list. The *Shannon entropy* (also called *information*) in a finite probability distribution is

$$H(p) = -\sum_{j=1}^{n} p_j \log_2(p_j) \ .$$

   (a) Describe a binary search strategy that uses bisection in the U (or F) space rather than in j. The method should take $O(n)$ operations to set up, and a smaller expected number of operations per random sample N_k.

   (b) Show that this strategy use expected number of probes per sample that is equal to the Shannon entropy.

   (c) Show that for fixed n, $H(p) \leq \log_2(n)$. Use this to show that your algorithm is not more expensive per sample than the bisection strategy in j space described in the text.

2. A *geometric* random variable is a discrete analogue of an exponential. If N is a geometric random variable with parameter p, then it "goes off" with probability p at each step. The probability it goes off right away is $p_0 = P(N = 0) = p$. The probability it goes off after one step is the probability of not going off then going off, which is $p_1 = P(N = 1) = (1-p)p$. In general, $p_k = (1-p)^k p$, because it must not go off the first k times (starting at zero) and then go off.

   (a) Show that N is a Markov process in the same sense the exponential random variable is. That is, $P(N = j + k \mid N \geq j) = P(N > k) = (1-p)^k p$.

   (b) Show that the p_k add up: $p_0 + \cdots = 1$.

   (c) Suppose T is an exponential random variable with rate parameter λ. Let $N = \lfloor T \rfloor$. Show that N is a geometric random variable. Find the formula for λ in terms of p to that N has parameter p. If x is a real number, then the *floor* of x is $n = \lfloor x \rfloor$, which is the largest integer that is not larger than x. For example $\lfloor \pi \rfloor = 3$, and $\lfloor 4 \rfloor = 4$, and $\lfloor -5.9 \rfloor = -6$. In C/C++, the *cast* operation `(int)` performs the floor operation. For example,

```
double x;
int    n;

x = -3.1415926535d0;
n = (int) x;
```

   Produces $n = -4$.

   (d) Suppose R is a transition matrix for a Markov chain, and for each i,

$$\sum_{j \neq i} R_{ij} \leq \varepsilon \ .$$

   Show that the work to simulate this chain directly is $O(\frac{1}{\varepsilon})$ per transition (a *transition* is a t with $X_{t+1} \neq X_t$). Suggest a method that uses $O(1)$ samples per transition using a geometric random variable.

3. Consider a two state continuous time Markov process that starts with $X_0 = 1$. The $1 \to 2$ transition rate is r and there are no $2 \to 1$ transitions. Then $P(X_t = 1) = e^{-rt}$. The $1 \to 2$ transition time is an exponential random variable with rate constant r. The direct discrete time approximation does a $1 \to 2$ transition with probability $r\Delta t$ in each time step.

   (a) Show that the discrete transition time is a geometric random variable as in Exercise 2. Give the formula for p in terms of r and Δt (an easy question).

   (b) Suppose t is fixed, and $n \to \infty$ as $\Delta t \to 0$ so that $t_n = n\Delta t = t$. Write the formula for $P(X_n^h = 1)$ in terms of n, t, and r.

(c) Show that $P(X_n^h = 1) = P(X_t = 1) - \frac{r^2 t\,\Delta t}{2} + O(\Delta t^2)$ as $\Delta t \to 0$. *Comment:* This discrete time approximation is *first order* accurate because the leading error term has Δt to the power 1.

4. Show that T is an exponential random variable with rate parameter λ if and only if $G(t) = P(T > t) = e^{-\lambda t}$. Suppose T_1, ..., T_n are independent exponentials with rate constants λ_1, ..., λ_n. Let $T = \min(T_1, \ldots, T_n)$. Show that $G(t) = G_1(t) \cdot \cdots \cdot G_n(t)$, where $G_k(t) = P(T_k > t)$. Use this to show that T is an exponential random variable with rate constant $\lambda_1 + \cdots + \lambda_n$. Show that

$$P(T = T_k) = \frac{\lambda_k}{\lambda_1 + \cdots + \lambda_n} \,. \tag{5}$$

Hint: show that

$$P(T = T_k) = \int_{t_k=0}^{\infty} P(T_k \in [t_k, t_k + dt_k])\, P(T_j > T_k \text{ for } j \neq k) \,.$$