Chapter 1

Introduction

A Monte Carlo method is a compuational method that uses random numbers to compute (estimate) some quantity of interest. Very often the quantity we want to compute is the mean of some random variable. Or we might want to compute some function of several means of random variables, e.g., the ratio of two means. Or we might want to compute the distribution of our random variable. Although Monte Carlo is a probabilistic method, it is often applied to problems that do not have any randomness in them. A classic example of this is using Monte Carlo to compute high dimensional integrals.

Monte Carlo methods may be divided into two types. In the first type we generate independent samples of the random variable. This is usually called direct, simple or crude Monte Carlo. The latter two terms are rather misleading. We will refer to these types of methods as direct Monte Carlo. In this type of Monte carlo the samples X_i that we generate are an i.i.d. sequence. So the strong law of large numbers tells us that the average of the X_i, i.e., the sample mean $\frac{1}{n}\sum_{i=1}^{n} X_i$, will converge to the mean of X as $n \to \infty$. Furthermore the central limit theorem tells us a lot about the error in our computation.

The second type of methods are Markov Chain Monte Carlo (MCMC) methods. These methods construct a Markov Chain whose stationary distribution is the probability measure we want to simulate. We then generate samples of the distribution by running the Markov Chain. As the chain runs we compute the value X_n of our random variable at teach time step n. The samples X_n are not independent, but there are theorems that tell us the sample mean still converges to the mean of our random variable.

We give some examples.

Example - Integration: Suppose we want to compute a definite integral over an interval like

$$I = \int_a^b f(x)\,dx \tag{1.1}$$

Let X be a random variable that is uniformly distributed on $[a, b]$. Then the expected value of $f(X)$ is

$$E[X] = \frac{1}{|b-a|}\int_a^b f(x)\,dx \tag{1.2}$$

So $I = (b-a)E[X]$. If we generate n independent sample of X, call them $X_1, X_2, \cdots, X_n$, then the law of large numbers says that

$$\frac{1}{n}\sum_{i=1}^{n} f(X_n) \tag{1.3}$$

conveges to $E[X]$. The error in this method goes to zero with n as $1/\sqrt{n}$. If f is smooth, simple numerical integration methods like Simpson's rule do much better - $1/n^4$ for Simpson. However, the rate of convergence of such methods is worse in higher dimensions. In higher dimension, Monte Carlo with its slow rate of convergence may actually be faster. And if the integrand is not smooth Monte Carlo may be the best method, especially if simplicity is important.

Example - more integration: Suppose we want to evaluate the integral

$$\int_0^{10} e^{-x^2} \tag{1.4}$$

We could follow the example above. However, e^{-x^2} is essentially zero on a large part of the interval $[0, 10]$. So for most of our samples X_i, $f(X_i)$ is essentially zero. This looks inefficent. A large part of our computation time is spent just adding zero to our total. We can improve the efficiency by a technique that is called importance sampling. We rewrite the integral we want to compute as

$$\int_0^{10} \frac{e^{-x^2}}{ce^{-x}} ce^{-x}\,dx \tag{1.5}$$

where the constant c is chosen so that $\int_0^{10} ce^{-x}\,dx = 1$. Let $g(x) = e^{-x^2}/ce^{-x}$. If X is a random variable with density ce^{-x}, then the expected value $E[g(X)]$ is the integral we want to compute. As we will learn, it is quite easy to use uniform random numbers on $[0, 1]$ to generate samples of X with the desired density. (It is not so easy to generate samples with density proportional to e^{-x^2} on $[0, 10]$.)

Example - yet more integration: Another example of the need for importance sampling is the following. Suppose we want to compute a d-dimensional integral over some region D. If D

is a parallelepiped, it is easy to generate a random vector uniformly distributed over D. But if D is a more complicated shape this can be quite difficult. One approach is to find a parallelepiped R that contains D. We then generate random vectors that are uniformly distributed on R, but we reject them when the vector is outside of D. How efficient this is depends on the ratio of the volume of D to that of R. Even if it is very difficult to generate uniform samples from D, it may be possible to find a domain D' which contains D and whose volume is not too much bigger than that of D. Then generating a uniform sample from D' and rejecting it if it is not in D can be a much more effecient method that generating a uniform sample from a parallelepiped that contains D.

Example - shortest path in a network: By a network we mean a collection of nodes and a collection of edges that connect two nodes. (Given two nodes there need not be an edge between them). For each edge there is a random variable that give the time it takes to traverse that edge. There random variables are taken to be independent. We fix a starting node and an ending node in the graph. The random variable we want to study is the minimum total time it takes to get from the starting node to the ending node. By minimum we mean the minimum over all possible paths in the network from the starting node to the ending node. For very small networks you work out the expected value of this random variable. But this analytic approach becomes intractable for every modest size networks. The Monte Carlo approach is to generate a bunch of samples of the network and for each sample network compute the minimum transit time. (This is easier said than done.) The average of these minima over the samples is then our estimate for the expected value we want.

Example - network connectivity or reliability We now consider a network but now it is random in a different way. We fix the nodes in the network. For each possible edge e, there is a parameter $p_e \in [0, 1]$. We include the edge in the graph with probability p_e. The edges are independent. Now we are interested in the probability that there is some path that connects the starting and ending nodes. Probabilties can be thought of as expected values and the strong law still provides the theoretical basis for a direct MC simulation. We can approximately compute the probability by generating a bunch of samples of the network and for each sample checking if the starting and ending nodes are connected. Our estimate for the probability of a connection is the number of samples that have a connection divided by the total number of samples.

Example - network connectivity with dependence In the previous example, if we let E_e be the event that the graph contains edge e, then these events are independent. In this example we again we fix the nodes in the network and make the edge configuration random. However, now the edges are not independent. There are many models that do this. Here is a relatively simple one. Let s_e be 0 if the edge e is not present, 1 if it is. So the graph is specified by the set of random variables s_e. Let s denote the collection $\{s_e\}$. We take the

probability density to be

$$P(s) = \frac{1}{Z} \exp(-\sum_e c_e s_e + \sum_{e,f} c_{e,f} s_e s_f) \tag{1.6}$$

where c_e and $c_{e,f}$ are parameters. The sum of pairs of edges e, f is only over $e \neq f$. The constant Z is determined by the requirement that this must be a probability measure. Even though the probability measure is quite explicit, there is no practical way to directly sample from it. In particular, computing Z is not feasible unless the number of nodes is small. But we can generate dependent samples of this distribution using a Markov Chain. Possible states are the possible edge configurations. If we are in state s the chain can transition to a new state s' which differs from s in only one edge. In other words, the transitions consist of either deleting an edge that is there or adding an edge that is not present. If we choose the transition probabilities for these transitions appropriately, the stationary distribution of the Markov chain will be P. So we can compute the probability that the starting and ending nodes are connected by running the chain.

Example - self-avoiding walks To be concrete we describe this in two dimensions on the square lattice. An N-step self-avoiding walk (SAW) is a nearest neighbor walk on the square lattice that does not visit a site more than once. We take all the N-step SAW's that start at the origin and put the uniform probability measure on this finite set. This is a really simple (to define) probability measure, but the number of such walks grows exponentially with N. So for large values of N there is no practical way to generate samples. We have to use a MCMC method.

Overview of the topics to be covered

Chapter 2 will introduce the basic idea of direct Monte Carlo and how one estimates the error involved. All Monte Carlo simulations require a source of randomness. Usually the starting point for this randomness is a pseudo-random number generator that produces numbers in $[0, 1]$ that look like they are uniformly distributed on the interval and independent. In chapter 3 we will look at the general structure of such generators and how one tests them. In chapter 4 we will study how you can use random numbers that are uniformly distributed on $[0, 1]$ to generate samples of a random variable with either a continuous or discrete distribution.

For all Monte Carlo methods the error in our estimate of the quantity we want to compute is of order $\sigma/\sqrt{n}$. In direct Monte Carlo the constant σ is just the standard deviation of the random variable whose mean we are computing. For MCMC σ is more involved. Obviously we can improve the estimate by increasing n, i.e., generating a larger number of samples. One can also attempt to reduce σ. This is known as variance reduction. We study some techniques for variance reduction in chapter 6. An important such technique that we touched on on some of the examples is importance sampling. All of chapter 7 is devoted to this topic.

In chapter 8 we will give a crash course on Markov chains, including the central idea of

MCMC. Chapter 9 will study some particular MCMC methods - the Gibbs sampler and the Metropolis-Hastings algorithm. In Chapter 10 we will look in more detail at the statistical analysis of what comes out of our Monte Carlo simulation. Further topics may include stochastic optimization, rare event simulation, perfect sampling, simulating annealing and who knows what.