1 Introduction

Monte Carlo methods are techniques that use random numbers as part of a scientific computation. Because Monte Carlo methods usually lead to large statistical errors, they are a method of last resort, to be used only when other methods are impractical. For example, we rarely would use Monte Carlo if the alternative was working a three dimensional integral by quadrature or solving a partial differential equation in two or three dimensions.

Standard grid based methods become impractical in high dimensions, a phenomenon called the "curse of dimensionality". Consider, for example, an integral over the d dimensional unit cube:

$$I = \int_{x_1=0}^{x_1=1} \cdots \int_{x_d=0}^{x_d=1} V(x_1, x_2, \ldots, x_d) dx_1 dx_2 \cdots dx_d \; . \tag{1}$$

If we use a mesh with step size $\Delta x = 1/n$ in each direction, the total number of mesh points is $N = n^d$. For $d = 20$ and $n = 10$, not a very fine mesh, the number of mesh points, and the number of operations needed, is more than any contemplated computer can do in a year. On the other hand, a Monte Carlo approximation with only $N = 10^6$ randomly chosen points could reasonably be expected to give three decimal digits accuracy.

Most Monte Carlo methods are based on the *law of large numbers* in probability. Suppose Y is a random variable with expected value $A = E[Y]$. The Monte Carlo algorithm generates many "samples" of Y (random variables with the same probability density), called $Y^{(1)}$, $Y^{(2)}$, ..., $Y^{(N)}$. The *sample mean* is used to estimate A:

$$A \approx \widehat{A}^{(N)} = \frac{1}{N} \sum_{k=1}^{N} Y^{(k)} \; . \tag{2}$$

The law of large numbers is a mathematical theorem that states that if the samples $Y^{(k)}$ are independent, and if $E[|Y|] < \infty$ so that the expected value is properly defined, then

$$\widehat{A}^{(N)} \to A \quad \text{as } n \to \infty.$$

More advanced Monte Carlo methods, including *Markov chain Monte Carlo*, produce samples that are not independent and have the probability density of Y only in the limit $N \to \infty$. These methods are beyond the scope of this course.

In the case of the integral (1), we could define a d dimensional uniform random variable $X = (X_1, \ldots, X_d)$ with probability density equal to one if $0 \leq x_j \leq 1$ for $k = 1, \ldots, x_d$ and zero otherwise. We can generate such an X by taking the components, X_k, to be independent scalar random variables uniformly distributed in the interval $[0, 1]$ (see below). We then define the scalar random variable $Y = V(X)$, so that $E[Y]$ is given by the integral (1). We can generate N independent samples $Y^{(k)}$ by generating N independent points $X^{(k)}$. The estimate of I is then the sample mean of the $Y^{(k)}$ as in (2).

I make a distinction between *Monte Carlo* and *simulation.* In Monte Carlo, the ultimate answer, A, is not itself a random number. For example, the integral

(1) is not random. We gladly would dispense with random numbers if d were small enough to make that practical. Even when A involves a random process (a hitting probability, expected time to failure, etc.), A itself is not random.

Simulations, in contrast, seek random outcomes. For example, computers use random numbers in generating artificial images of clouds or trees. Computer network designers simulate their design networks with random inputs to see what the answers look like. At some point, the designer may spot a curious phenomenon and wish to study it quantitatively. As soon as she or he formulates a specific goal, such as determining the expected number of packed collisions per minute, she or he is doing Monte Carlo.

The point of the distinction is that Monte Carlo has much more freedom than simulation. In simulation we only want to create a faithful computer representation of our model. In Monte Carlo we are free to seek other definitions of the answer A that might make the computation easier or more accurate. For example, in (1) it might be possible to integrate by parts and express I in terms of a different integrand, $W(x)$. Maybe W is easier to evaluate or has less variation. In simulation, variance is neither good nor bad, it is just a reflection of the model being simulated. In Monte Carlo, variance in the estimate of A is bad because it makes the estimate less accurate.

Reformulating a problem to reduce the Monte Carlo variance is called *variance reduction*. The major variance reduction techniques are *control variates*, *antithetic variates*, and *importance sampling*. Since Monte Carlo computations are often very slow, practitioners are amply motivated to explore variance reduction.

Because the errors in Monte Carlo are mainly statistical, the methods of statistics primarily are used to estimate them. If A is the answer and $\widehat{A}$ is the Monte Carlo estimate of A, the *bias* is $E[\widehat{A}] - A$. The *statistical error* is $\Delta\widehat{A} = \widehat{A} - E[\widehat{A}]$. In a majority of Monte Carlo computations, the bias is either zero or much smaller than the statistical error. For example, the estimate (2) will have zero bias if $E[Y^{(k)} = E[Y]$, which will happen if the probability density for all of the $Y^{(k)}$ is identical to that for Y, as in the Monte Carlo estimate of the integral (1). For this reason, Monte Carlo practitioners tend to give careful estimates of statistical error while trusting the bias to be less significant.

Statistical error estimates are represented through *error bars*, which are related to statistical *confidence intervals*. A confidence interval is a computed interval $[\widehat{A}_-, \widehat{A}_+]$ with the property that $E[\widehat{A}] \in [\widehat{A}_-, \widehat{A}_+]$ with probability p_c (the *confidence probability*). Typically statisticians use $p_c = 99\%$ or $p_c = 99\%$. Of course, $E[\widehat{A}]$ is not random, but $\widehat{A}_-$ and $\widehat{A}_+$ are computed during the course of the Monte Carlo computation and are random. For $p_c = 95\%$, there is a 5% chance that $\widehat{A}_- > E[\widehat{A}]$ or $\widehat{A}_+ < E[\widehat{A}]$. If I repeat the Monte Carlo computation a hundred times (with a different seed), $E[\widehat{A}]$ remains the same unknown value and in all but about five of the runs, it is inside the computed confidence interval. The error bars used by Monte Carlo practitioners tend to be $p_c \approx 33\%$ confidence intervals because they prefer to give a realistic idea how large the statistical error is likely to be than to be right a vast majority of the time.

This quite simple in all but the most sophisticated Monte Carlo. We compute an estimate, $\widehat{\sigma}$, of the standard deviation of $\widehat{A}$. The confidence interval bounds are then $\widehat{A}_\pm = \widehat{A} \pm 2\widehat{\sigma}$ or $\widehat{A}_\pm = \widehat{A} \pm 3\widehat{\sigma}$ for statisticians' $p_c = 95\%$ or $p_c = 99\%$. Monte Carlo practitioners often prefer $\widehat{A}_\pm = \widehat{A} \pm \widehat{\sigma}$. These are "one standard deviation" error bars, in contrast to two or three standard deviation error bars used by cautious statisticians. An error bar is plotted as a bar, running from $\widehat{A}_-$ to $\widehat{A}_+$, usually with a dot in the center to represent the actual $\widehat{A}$.

Error bars should be part of every Monte Carlo computation. The statistical errors in Monte Carlo are generally larger than errors (roundoff, truncation error, premature termination error) in other parts of scientific computing. It would be unprofessional to present the results of a numerical integration without having done some convergence study to verify the correctness. Error bars play that role in Monte Carlo. Error bars are part of most scientific presentations that involve Monte Carlo data. Presentations for nonscientific people may leave out the error bars, provided that the practitioner has looked at them and decided they are small enough.

We will follow to some extent the common convention that random variables are denoted by capital letters with generic numbers given by the corresponding lower case letter. We might, for example, have a random variable, X, with a probability density $f(x)$. We will use the informal differential notation of scientists rather than the rigorous notation of mathematicians. In particular, the probability density function is defined by the slightly informal expression $f(x)dx = Pr(x \leq X \leq x + dx)$.

2 Random number generators

Random number generators are computer programs that produce numbers may be used as random numbers in Monte Carlo computations. In principle if `float rand()` is a random number generator, then

```
for ( k=0; k<N; k++) U[k] = rand();
```

should produce random variables, U_k, that are independent and uniformly distributed in the unit interval. The random variable U is uniformly distributed in the interval $[a, b]$ if the probability density is $f(u) = 1/(b - a)$ for $u \in [a, b]$ and $f(u) = 0$ otherwise. The *standard uniform* random variable has $a = 0$ and $b = 1$. The random variables U_k are independent if their joint probability density is a product of the one variable densities. The joint uniform density was used in the Monte Carlo method for estimating (1) above.

Like computer arithmetic, computer random number generators are not perfect. The numbers are not actually random. Even the best random number generators have some correlation between the U_k. These correlations may be too subtle for an amateur to find, but the experts are not fooled. Nevertheless, most of our discussion of Monte Carlo methods assumes that we have access to as many truly uniform and independent random variables as we need. Much ingenious Monte Carlo technique goes into the *sampling* problem: using lots of independent standard uniform random variables to generate other more complicated random variables.

Practical random number generators are based on two functions, Φ, and Ψ. The *seed*, S, is a small collection of integers in a certain range, such as the range of 32 bit integer arithmetic. The operation $S' = \Phi(S)$ *updates* the seed. The operation $U = \Psi(S)$ produces a real number in the interval $[0,1]$ from the seed S. A call `U = rand()` has the effect of returning $U = \Psi(S)$ and replacing S with S'. A good random number generator will allow the user to access the seed through something like `S = getSeed()`, and to set the seed, as in `setSeed(S)`. If you write a code that uses `rand()` without `setSeed(S)`, the computer probably sets the seed from it's internal clock.

Programmers access and set the seed for several reasons. If you are debugging a code and want it to do the same thing every time, you can explicitly set the seed at the beginning. This will insure that the "random" numbers, U_k, are the same each run. It is common that Monte Carlo runs are so long that they need to be *checkpointed*. Checkpointing means stopping a long computation in the middle and writing all the data to disk so that the run can be continued later. If the computer crashes, you lose only the time since the most recent checkpoint. For Monte Carlo, the codes

```
for ( k = 0; k < 20; k++) U[k] = rand();
```

and

```
for ( k = 0; k < 10; k++) U[k] = rand();
S = getSeed();
y = rand();
  ( . . . other stuff involving rand() but not S . . . )
setSeed(S);
for ( k = 10; k < 20; k++) U[k] = rand();
```

do the same thing. Restoring the seed after step 10 continues the U_k sequence as if it had not been interrupted. A disastrous bug that beginners have been known to make (but never more than once) is

```
for ( k = 0; k < 20; k++) {
   U[k] = rand();
   setSeed(S);
 }
```

This gives each of the U_k the same value.

The details structure of random number generators (as determined by Φ and Ψ) are beyond the scope here. The simplest are congruental, which take S to be a single integer and $S' = a * S + b \bmod c$. If S is a single 32 bit integer, then the number of U possible is not more than $2^{32} \approx 8 \times 10^9$. On a modern computer, a program could use all these in a minute. Continuing the Monte Carlo computation beyond this point would necessarily mean reusing old numbers rather than generating new ones. Thus, the effective maximum N would be severely limited. Modern random number generators do not have this problem: the cycle time is too long.

3 Direct sampling methods

Most Monte Carlo computations require us to generate random variables with a given distribution using standard uniform random variables. Some of the tricks for doing this are stunningly brilliant and simple.

3.1 Bernoulli random variables

A Bernoulli random variable, X has two possible values, $X = 0$ or $X = 1$. It is characterized by the probability of getting 1: $p = Pr(X = 1)$. You can make a Bernoulli from a standard uniform by testing whether the uniform is less than p

```
if ( rand() < p ) X = 1;
else              X = 0;
```

If we run this code segment N times, we get N independent Bernoulli random variables with the same p. Recall that for any interval $[a, b] \subseteq [0, 1]$, $Pr(U \in [a, b]) = b - a$. In particular, $Pr(\xi < p) = p$.

More generally, suppose we want to choose between $d \geq 2$ outcomes with outcome j having probability p_j. The following code does it:

```
float U, q;
q  = 0;
U = rand();
for ( j=0; j<d; j++ ) {
   q += p[j];
   if ( U < q ) break;
 }
```

This code returns j if U falls into an interval of length p_j. It assumes that all d elements of p are defined even though in principle we can determine p_d from the others using the relation $p_1 + \cdots + p_d = 1$. The code is robust in the presence of roundoff. If the final q is slightly different from one or U is slightly larger than one, we will get $j = d$ (actually $j = d - 1$ because C starts with $j = 0$ while math starts with $j = 1$), which is a reasonable outcome. The probability of this happening is extremely small, but if it would happen even once, it could crash the code if it were written differently. Simulating discrete random systems, such as finite state space Markov chains, is usually done this way.

3.2 Exponential random variables

The exponential random variable with rate constant $\lambda > 0$ has probability density $f(t) = \lambda e^{-\lambda t}$ for $t \geq 0$ and $f(t) = 0$ otherwise. It is used to model random failure times. Suppose T is an exponential random variable with rate λ. We can define λ by $Pr(0 \leq T \leq dt) = \lambda dt$. The model is that if the thing has not failed by time t, it is good as new at time t, no matter how large t is.

More technically, $Pr(t \leq T \leq t + dt \mid T \geq t) = \lambda dt$. The exponential form of the probability density is a consequence of this axiom.

You can get an intuition for exponential random variables by thinking of randomly shaking a bowl with a marble in it. If you shake vigorously, the marble will fly out of the bowl almost immediately. If you shake less vigorously, the marble will move randomly around the bowl. It might suddenly fly out, but otherwise it stays near the bottom of the bowl, good as new.

Discrete event simulations in continuous time, continuous time discrete Markov chains, all involve exponentials. The random times between transitions are all exponential random variables.

Sampling an exponential random variable is simpler than saying what it is. We start with the "standard" exponential random variable, the one with rate constant $\lambda = 1$. If S is a standard exponential random variable, then $T = S/\lambda$ is an exponential with rate λ. (verify this). If ξ is a standard uniform random variable, then

$$S = -\ln(U) \tag{3}$$

is a standard exponential. Note that the logarithm is well defined because $U > 0$ and negative because $U < 1$. Changing the sign in (3) yields a positive S. Most random number generators never produce $U = 0$, but be careful if yours does.

We can calculate the probability density produced by the formula (3) and see that it is standard exponential. This density, $f(s)$, is defined by $f(s)ds = Pr(s \leq S \leq s + ds)$. Substituting (3) gives

$$\begin{aligned} f(s)ds &= Pr(s \leq -\ln(U) \leq s + ds) \\ &= Pr(-s - ds \leq \ln(U) \leq -s) \\ &= Pr(e^{-s-ds} \leq U \leq e^{-s}) \ . \end{aligned}$$

We can write $e^{-s-ds} = e^{-s}e^{-ds}$ and expand the second exponential: $e^{-ds} = 1 - ds$ to get

$$f(s)ds = Pr(e^{-s}(1 - ds) \leq U \leq e^{-s}) \ .$$

If $[a, b] \subseteq [0, 1]$, and U is a standard uniform random variable, then $Pr(U \in [a, b]) = b - a$. Here, the interval runs from $a = e^{-s}(1 - ds)$ to abe^{-s}, for a total length of $e^{-s}ds$. Therefore, we have

$$f(s)ds = e^{-s}ds \ ,$$

which shows that S is indeed a standard exponential.

3.3 Using the distribution function

If X is a random variable with density $f(x)$, then the distribution function (or CDF, for cumulative distribution function) is

$$F(x) = Pr(X \leq x) = \int_{-\infty}^{x} f(x')dx' \ . \tag{4}$$

We assume that $f(x) > 0$ for all x, which makes the discussion simpler because $F(x)$ is a strictly increasing function of x in that case. We show that the random variable

$$U = F(X) \tag{5}$$

is a standard uniform random variable. Conversely, if U is standard uniform and we define X using (5), then X will have $F(x)$ as its distribution function and $f(x)$ as its density.

Indeed, U is standard uniform if $U \in [0,1]$ and $Pr(U \leq u) = a$ for $u \in [0,1]$. Now, choose any such u and define x by $F(x) = u$. The probability that $U \leq u$ is the same as the probability that $F(X) \leq u = F(x)$, which is the same as the probability that $X \leq x$, which is $F(x) = u$. With the association $f = F(x)$, saying $Pr(U \leq u) = u$ for all $u \in [0,1]$ is the same as saying $Pr(X \leq x)\ F(x)$ for all x, which is what we wanted to show.

We give this argument slightly differently using calculus. For example, suppose we define U by (5) where $f(x)$ is the density for X, and seek $g(u)$, the density for U. Under the mapping $u = F(x)$, the interval $[x, x+dx]$ is mapped to the interval $[u, u+du]$, where $du = F'(x)dx = f(x)dx$. The probabilities of these intervals are the same, since $X \in [x, x+dx]$ is equivalent to $U \in [u, u+du]$. Expressing the probabilities in terms of densities and equating them gives $f(x)dx = g(u)du$. Since $du = f(x)dx$, we find $g(u) = 1$. Of course, this only holds for u in the allowed range $0 \leq u \leq 1$. Other values of u are impossible, so $g(u) = 0$ otherwise.

We apply this general principle to the standard exponential random variable with density $f(t) = e^{-t}$ for $t > 0$ and $f(t) = 0$ for $t < 0$. We will see that it works even though the hypothesis that $f(t) > 0$ for all t is not satisfied. Doing the integral in (4) gives $F(t) = 1 - e^{-t}$ for $t > 0$ and $F(t) = 0$ for $t < 0$. The identification $F(T) = U$ can be solved for T in terms of U to give $T = -\ln(1-U)$. Of course, if U is standard uniform, then $1 - U$ is also. Therefore, we get the same distribution if we take $T = -\ln(U)$ as before.

The CDF for the standard normal distribution is often called $N(x)$. It is given by

$$N(x) = \frac{1}{2\pi} \int_{-\infty}^{x} e^{-y^2/2} dy \ .$$

This is related to but different from the error function $erf(x)$. To generate a standard normal in this way, we need software to solve the equation $N(X) = U$ for X, given $U \in [0,1]$. Reliable and accurate C code for computing $N(x)$ and $N^{-1}(u)$ is available on the net.

The main obstacle to using the general recipe (5) is the difficulty of computing $F(x)$ and inverting it (solving for X given U). This makes other sampling methods, including rejection, interesting.

3.4 Standard normals using Box Muller

3.5 Multivariate normals and Cholesky factorization

From a standard normal, Z, we can generate a normal, X, with mean μ and variance σ^2 simply by multiplying by σ and adding μ: $X = \sigma Z + \mu$.

3.6 Rejection

Rejection sampling is also very general, but it has the potential to be very inefficient if not done carefully. We wish to generate a random variable, X with density function $f(x)$. Rejection sampling uses a *trial* or *proposal* density, $g(x)$, and an *acceptance probability*, $p(x)$. We assume that we have a way to sample the g density, and we want to use these samples to sample f.

4 Multivariate sampling

In many of the

5 Error bars and statistical analysis

Analysis of statistical error is a necessary part of any Monte Carlo computation. The analysis appropriate for most elementary Monte Carlo is error bars derived from the central limit theorem. This analysis is simple and should take much less computer time than the rest of the Monte Carlo computation.

In its most basic form, the central limit theorem concerns the sum of N independent random variables with the same probability density. Statisticians call this the i.i.d. case, for Independent and Identically Distributed. If $Y^{(1)}$, $Y^{(2)}$, ..., are i.i.d. random variables and $\widehat{A}^{(N)}$ is the average (2), then $\widehat{A}^{(N)}$ is approximately normal. The probability density of a normal random variable is completely determined by its mean and variance. A simple calculation shows that $E[\widehat{A}^{(N)}] = A = E[Y]$ and $var[\widehat{A}^{(N)}] = \frac{1}{N} var[Y]$. Since the $Y^{(k)}$ are identically distributed, we let Y represent any one of these. Statisticians sometimes say that the $Y^{(k)}$ are "independent samples of Y".

The central limit theorem describes the probability distribution of $\widehat{A}^{(N)}$. To understand the technical statement below, recall that for every gaussian random variable, the probability of being n standard deviations away from the mean is the same, and the same as for the standard normal. That is, if X is normal with mean μ and variance σ^2, and Z is a standard normal with mean zero and variance one, then:

$$P_n = Pr(|X = \mu| \geq n\sigma) = Pr(|Z| \geq n) = \frac{1}{\sqrt{2\pi}} \int_{-n}^{n} e^{-z^2/2} dz \; .$$

These probabilities are well known to statisticians: $P_1 = 33\%$, $P_2 = 5\%$, and $P_3 < 1\%$. n need not be an integer. If we apply this to $\widehat{A}^{(N)}$ with it's known mean and variance, we get

$$Pr\left(\left|\widehat{A}^{(N)} - A\right| > \frac{n\sigma_Y}{\sqrt{N}}\right) \approx P_n \; . \tag{6}$$

The central limit theorem formula (6) is the basis of error analysis in basic Monte Carlo. The n standard deviation *error bar* is the interval $[\widehat{A}^{(N)} - n\sigma, \widehat{A}^{(N)} + n\sigma]$, where $\sigma = \sigma_Y/\sqrt{N}$ is the standard deviation of $\widehat{A}^{(N)}$. The reader can verify that A is inside the error bar exactly if $\left|\widehat{A}^{(N)} - A\right| < n\sigma$, so the probability that the exact but unknown answer A is inside the error bar is $1 - P_n$. Statisticians call the error bar a *confidence interval* because there is $1 - P_n$ confidence that A lies within it. If you compute the n standard deviation error bar, you have probability $1 - P_n$ of including A in the error bar.

Monte Carlo practitioners usually use one standard deviation error bars. The purpose is to give a graphical representation of the actual size of the error rather than to give a much larger interval that is nearly certain to contain the answer. In tables of Monte Carlo results, you might see something like $A = 12.87 \pm .034$. Unless otherwise specified, the author probably means that $\widehat{A}^{(N)} = 12.87$ and $\sigma = .034$. An error bar is usually plotted as a bar with a visible dot in the center representing $\widehat{A}^{(N)}$ and a bar on either side of length $n\sigma$ representing the range of uncertainty of $\widehat{A}^{(N)}$.

In practice we have also to estimate σ_Y. It is natural to use the Monte Carlo estimate

$$\widehat{\sigma}_Y^2 = \frac{1}{N}\sum_{k=1}^{N}\left(Y^{(k)} - \widehat{A}^{(N)}\right)^2 \; . \tag{7}$$

Statistics books often recommend the factor $\frac{1}{n-1}$ instead of our $\frac{1}{N}$. The answer is that if this makes an appreciable difference, you do not have enough data. You should not be doing Monte Carlo estimation, or using the central limit theorem, with fewer than, say, $N = 10$ points. In favorable cases, you will have $N >\approx 10^6$.

5.1 Summary of the basic Monte Carlo procedure

You want a number A. You find a random variable, Y so that $E[Y] = A$. You generate N independent samples of Y, called $Y^{(k)}$, for $k = 1, 2, \ldots, N$. You form the sample mean (2) and the sample standard deviation (7) then you report

$$A \approx \widehat{A}^{(N)} \pm \frac{1}{\sqrt{N}}\widehat{\sigma}_Y \; .$$