Monte Carlo Methods

* It is quite common in statistics that we need to evaluate complex integrals.

* Such integrals are often quite difficult (or indeed impossible) to evaluate in closed form.

* Monte Carlo methods are a way of approximating the value of an integral using large samples of random variables.

* These samples of random variables are typically computer generated.

Monte Carlo Integration

* Suppose that we need to evaluate

$$I = \int_{\mathcal{A}} h(x)\,dx$$

* Let f be a probability density function with support $\mathcal{A}$.

* Then we can write

$$I = \int_{\mathcal{A}} \frac{h(x)}{f(x)} f(x)\,dx = \int_{\mathcal{A}} g(x) f(x)\,dx$$

* Now if Y is a random variable with pdf f then we have

$$I = \mathsf{E}[g(Y)].$$

Monte Carlo Integration

* Hence if we have $Y_1, \ldots, Y_N \overset{iid}{\sim} f$, the Weak Law of Large Numbers tells us that

$$\widehat{I} = \frac{1}{N} \sum_{i=1}^{N} g(Y_i) \xrightarrow{p} I.$$

* We can therefore use $\widehat{I}$ to approximate I very well for large enough N.

* Furthermore we can use estimate the variability in $\widehat{I}$ using the sample variance of the random sample $g(Y_1), \ldots, g(Y_N)$ divided by N.

* Since N is totally in our control we can choose N to be large enough to make the variability as low as we desire.

Generating Uniform Random Variates

* Computers are unable to generate random numbers.

* The can, however, be used to generate pseudo-random numbers.

* These are sequences of numbers which are generated from a deterministic algorithm but which behave like a sequence of *iid* random variates.

Generating Uniform Random Variates

* Generally a computer uses an algorithm to generate a sequence of integers $x_0, x_1, x_2, \ldots$ such that

$$x_t \in \{0, 1, \ldots, M-1\}$$

* The generated values are then rescaled to be approximately Uniform$(0, 1)$

$$U_t = \frac{x_t}{M}$$

* Many algorithms have been proposed. Most are of the form

$$x_t = g(x_{t-1})$$

* The function g is chosen so that the resulting sequence resembles a random sequence.

Common Random Number Generators

Definition 8.1

A linear congruential generator on $\{0, 1, \ldots, M-1\}$ *is a sequence of integers defined by*

$$x_{t+1} = (ax_t + b) \mod M$$

Definition 8.2

For a given $k \times k$ *matrix* T *whose entries are all either* 0 *or* 1, *the associated shift register generator is*

$$\boldsymbol{x}_{t+1} = T\boldsymbol{x}_t \mod 2$$

where $\boldsymbol{x}_t$ *and* $\boldsymbol{x}_{t+1}$ *are binary representations of the corresponding numbers.*

Combination Generators

* Most modern generators use two or more parallel generators and return a sum or product of the results modulo the largest integer representable.

* One such generator is George Marsaglia's KISS (Keep It Simple, Stupid) generator which uses one linear congruential generator and two shift register generators, returning their sum modulo 2^{32}.

* The current state of the art generator is generally believed to be the Mersenne-Twister Generator and this is the default in R.

* No test of uniformity or randomness has yet been found that the sequence from this generator does not pass!

Reproducibility of Generated Random Numbers

* An important concept is that of a Random Seed.

* The seed specifies initial conditions for the algorithm which generates the random variates.

* Since the algorithm is actually deterministic, setting the same random seed will result in the same sequence of random variates.

* When using simulation to compare two or more methods, we will often want to use to same random numbers for each method to remove the simulation variability between methods and so give a better comparison.

Generating Non-Uniform Random Numbers

* Uniform random variates are rarely what we need for simulation or Monte Carlo inference.

* They are, however, the building blocks for generating random variates from any other distributions.

* Much of this is based on the following theorem

Theorem 8.1 (Probability Integral Transform)
Suppose that $U \sim$ *Uniform*$(0, 1)$ *and that* F *is a continuous cdf with unique inverse* F^{-1}*. Then the random variable*

$$Y = F^{-1}(U)$$

has a distribution with cdf F*.*

Generating Discrete Random Variates

* The method as described above requires that we have a continuous cdf.

* A similar technique can also be used to generate discrete random variables.

* Suppose that $p(y)$ is the probability mass function and the support of the random variable is $\mathcal{Y} = \{y : p(y) > 0\}$. Then we can define the inverse cdf as

$$F^{-1}(u) = \min\{y \in \mathcal{Y} : F(y) \geqslant u\}$$

* Then if $U \sim \text{uniform}(0, 1)$, the random variable $Y = F^{-1}(u)$ will be distributed with probability mass function $p(y)$.

Special Methods

* In many cases, the inverse cdf is not available in closed form and so this method cannot be used. We can often, however, use algorithms based on transformations for such situations.

* Suppose that U_1 and U_2 are two independent Uniform$(0,1)$ random variables then it is easy to show that

  $$Y_1 = \sqrt{-2\log U_1}\sin(2\pi U_2) \quad \text{and} \quad Y_2 = \sqrt{-2\log U_1}\cos(2\pi U_2)$$

  are independent standard normal random variables.

* This is known as the Box-Muller Algorithm.

Accept/Reject Algorithm

* A more general technique which is useful when the inverse cdf method cannot be applied is called the Accept/Reject Algorithm.

* This method relies on generating a different random variable V which has the same support as the required variable Y.

* We also require that the ratio of densities is bounded by a known constant

$$M = \sup_y \frac{f_Y(y)}{f_V(y)} < \infty$$

Accept/Reject Algorithm

1. Calculate $M = \sup_y f_Y(y)/f_V(y)$.

2. Generate $V \sim f_V$ and independently $U \sim \text{Uniform}(0,1)$.

3. If
$$U < \frac{f_Y(V)}{M f_V(V)}$$
then set $Y = V$. Otherwise discard U and V and return to step 2.

Markov Chain Monte Carlo Methods

* Many of the methods described so far are not very useful for generating multivariate random variates.

* Markov Chain Monte Carlo methods are now widely used in these settings.

* The methods work on the idea of constructing a Markov chain which has a stationary distribution equal to the distribution of interest.

* Under certain conditions, the distribution of the elements in such a chain will converge to this stationary distribution.

Markov Chain Monte Carlo Methods

* These algorithms start with some initial value for the random variable of interest.

* They then run a carefully constructed Markov chain starting from that initial value for a sufficiently long time.

* It is not always easy to know how long the chains should be run but various diagnostics have been proposed.

* Any observations in the chain after this burn-in period may be considered as (at least approximately) distributed with the stationary distribution.

Metropolis–Hastings Algorithm

* First introduced in statistical physics in 1954 by Metropolis et al. Statistical properties shown by Hastings in 1970.

* It is basically a Markov chain version of the accept/reject algorithm.

* Random variates are generated from some candidate distribution conditional on the current state of the chain and then either the new state is accepted or rejected in which case the chain stays where it is.

Metropolis–Hastings Algorithm

Suppose we wish to sample $Y \sim f_Y$.

First initialize the chain with some value $Y^{(0)}$.

Then for $t = 1, 2, \ldots$ we generate $Y^{(t)}$ by

1. Generate $V^{(t)} \sim f_{V|Y}(v \mid Y^{(t-1)})$.
2. Calculate the acceptance probability

$$\rho_t = \min\left\{ \frac{f_Y\left(V^{(t)}\right)}{f_Y\left(Y^{(t-1)}\right)} \times \frac{f_{V|Y}\left(Y^{(t-1)} \mid V^{(t)}\right)}{f_{V|Y}\left(V^{(t)} \mid Y^{(t-1)}\right)}, 1 \right\}$$

3. Generate $U_t \sim \text{Uniform}(0, 1)$ and set

$$Y^{(t)} = \begin{cases} V^{(t)} & \text{if } U_t \leqslant \rho_t \\ Y^{(t-1)} & \text{if } U_t > \rho_t \end{cases}$$

Independence Metropolis–Hastings Algorithm

* It is often convenient to generate $V^{(t)}$ from the same distribution at every iteration.

* In this case we have $f_{V|Y}(v \mid Y^{(t-1)}) = f_V(v)$ and so the acceptance probability becomes

$$\begin{aligned}\rho_t &= \min\left\{\frac{f_Y\left(V^{(t)}\right)}{f_Y\left(Y^{(t-1)}\right)} \times \frac{f_V\left(Y^{(t-1)}\right)}{f_V\left(V^{(t)}\right)},\ 1\right\} \\ &= \min\left\{\frac{f_Y\left(V^{(t)}\right)}{f_V\left(V^{(t)}\right)} \times \frac{f_V\left(Y^{(t-1)}\right)}{f_Y\left(Y^{(t-1)}\right)},\ 1\right\}\end{aligned}$$

Random Walk Metropolis–Hastings Algorithm

* Another special case is where $f_{V|Y}(v \mid y) = f_Z(v - y)$ where f_Z is a distribution symmetric about 0.

* We generate $Z^{(t)} \sim f_Z$ and set $V^{(t)} = Y^{(t-1)} + Z^{(t)}$.

* In stochastic processes this is called a random walk.

* The acceptance probability for the Metropolis–Hastings algorithm then becomes

$$\rho_t = \min\left\{\frac{f_Y\left(V^{(t)}\right)}{f_Y\left(Y^{(t-1)}\right)}, 1\right\}$$

Gibbs Sampler

* The Gibbs Sampler (Geman & Geman, 1984) is designed to generate observations from a complex multivariate distribution.

* The Markov chain is constructed by considering the univariate conditional distributions.

* Suppose that the random vector of interest is $\boldsymbol{Y} = (Y_1, \ldots, Y_d)$ and that we can generate observations from the full conditional distributions

$$f_j\left(y \mid Y_{-j} = y_{-j}\right) \;=\; f_{y_j|y_{-j}}\left(y \mid Y_{-j} = y_{-j}\right) \qquad j = 1, \ldots, d$$

where $Y_{-j} \;=\; \left(Y_1, \ldots, Y_{j-1}, Y_{j+1}, \ldots, Y_d\right)$.

Gibbs Sampler

Initialise the chain to some value $\boldsymbol{Y}^{(0)} = \left(Y_1^{(0)}, \ldots, Y_d^{(0)}\right)$.

For $t = 1, 2, \ldots$

1 Generate $Y_1^{(t)}$ from $f_1\left(y_1 \mid Y_2^{(t-1)}, \ldots, Y_d^{(t-1)}\right)$.

2 Generate $Y_2^{(t)}$ from $f_2\left(y_2 \mid Y_1^{(t)}, Y_3^{(t-1)}, \ldots, Y_d^{(t-1)}\right)$.

$\vdots$

j Generate $Y_j^{(t)}$ from $f_j\left(y_j \mid Y_1^{(t)}, \ldots, Y_{j-1}^{(t)}, Y_{j+1}^{(t-1)}, \ldots, Y_d^{(t-1)}\right)$.

$\vdots$

d Generate $Y_d^{(t)}$ from $f_d\left(y_d \mid Y_1^{(t)}, \ldots, Y_{d-1}^{(t)}\right)$.

Then we set $\boldsymbol{Y}^{(t)} = \left(Y_1^{(t)}, \ldots, Y_d^{(t)}\right)$.