Monte Carlo Methods

Honours Statistical Computing

Theo Stewart, Birgit Erni

2024-03-05

Introduction

We frequently need to establish the distribution, or at least certain parameters of the distribution, of random variables which are themselves complicated functions of random quantities. For example, statistical inference requires tail probabilities for test statistics; the actuary requires properties (at least the mean and variance) of total claims arising at random times from different sources. It is rare for these distributions, or even the required parameter values, to be obtainable in closed form from analytical arguments. The Behrens-Fisher problem is a good example of this: the distribution of the two-sample t-test statistic with unequal variances remains unknown analytically, even for the simple case of independent random samples from normal distributions.

Monte Carlo methods refer to the use of random number simulations to evaluate mathematical expressions (Gentle 2003). In a slightly stricter sense Monte Carlo methods refer to the methods used to generate these random numbers from probability distributions, including importance sampling, rejection sampling, the Metropolis method, and Gibbs sampling (MacKay 2003). The third, strictest, definition is as follows:

Monte Carlo methods broadly refer to: the use of generating random numbers / statistical sampling to solve problems that are difficult to solve analytically. During the 1940's nuclear physicists at the Nuclear Weapons Lab in Los Alamos wanted to understand the path and history of neutrons. At the time they used differential equations to describe these systems, but this was very difficult to solve analytically. Stan Ulan had the idea that one could, instead of solving differential equations, simulate the paths and behaviours of a large number of neutrons and their interactions with other neutrons a large number of times and then use the results to understand what the typical behaviour of these neutrons would be. John von Neumann knew how one could implement these simulations on the new computers developed at the time (Eckardt 1987, Metropolis 1987).

The term 'Monte Carlo' was used as a code name, because the nuclear work had to be kept secret, with 'Monte Carlo' alluding to the stochastic nature of gambling / games of chance in casinos.

The problems solved by Monte Carlo methods can often be phrased as the evaluation of an integral:

$$E_f[h(X)] = \int_X h(x)f(x)dx. \tag{1}$$

If one had a sample from $f(x)$, $(X_1, \ldots, X_m)$, one can approximate equation 1 by the empirical average

$$\bar{h}_m = \frac{1}{m}\sum_{j=1}^{m} h(x_j) \tag{2}$$

since $\bar{h}_m$ converges almost surely to $E_f[h(X)]$ by the Strong Law of Large Numbers (Robert and Casella 2004).

The approach in equation 2 is the *Monte Carlo method* in its strictest and original sense (Metropolis and Ulam 1949). Hence these methods are often referred to as Monte Carlo Integration.

MacKay summarises the problems to be solved using Monte Carlo methods as either or both of

Problem 1: to generate samples $x_r, r = 1, \ldots, R$ from a given probability distribution $P(x)$

Problem 2: to estimate expectations of functions under this distribution, e.g.

$$\Phi = \int \phi(x)P(x)d^N x$$

where N refers to the dimension of the distribution.

The second of these problems is solved by equation 2 once we have generated values from $P(x)$.

Example

Consider the following hypothetical example: $X_1, X_2, \ldots$ is a sequence of random variables, such that X_i is exponentially distributed with mean X_{i-1}. We assume that the mean of X_1 is a known constant, say x_0. Suppose that we are interested in the distribution of $S = \sum_{i=1}^{N} X_i$, where N is itself a random variable, say negative binomial with parameters p and r. The mean of S can be found analytically (it is $x_0 E[N]$), but what about $\mathrm{Var}[S]$, or $\Pr[S > c]$ for some $c > 0$? Although you (probably!) will not be able to obtain these values analytically, consider the following pseudo-code.

```
Set SSQ=0, SUM=0
FOR i=1 to BIG
        "Generate" a realization of N from the negative
                binomial distribution
        Set MEAN=X0 (given), S=0
        FOR j=1 to N
                "Generate" an X from the exponential
                        distribution with mean MEAN
                Set S=S+X
                Set MEAN=X
        NEXT j
        Set SSQ=SSQ+(S-E[S])*(S-E[S]) {Where E[S] is presumed known}
        IF S>c Set SUM=SUM+1
NEXT i
V_EST=SSQ/BIG
P_EST=SUM/BIG
```

Execution of such code would generate estimates of $\mathrm{Var}[S]$ and of $\Pr[S > c]$. With a little more effort we could get estimates of the standard errors in these estimates. For example, the standard error in the estimate of $\Pr[S > c]$ could be estimated by $\sqrt{\texttt{P\_}EST * (1 - P\_EST)/BIG}$. Since the entire procedure is under our control, we can in principle make this standard error as small as needed by choosing `BIG` as large as needed.

The above defines a *Monte Carlo* procedure for estimating properties or parameters of a distribution. There are two important

practical questions to be answered, and these form the theme of the remainder of this chapter. These questions are:

1. How do we get the computer to generate realizations of a random variable from a desired probability distribution?
2. Are there ways to improve the accuracy of Monte Carlo methods, other than simply increasing the numbers of iterations of the procedure to astronomical numbers?

Generating random numbers from arbitrary probability distributions

Most computer systems and programming languages provide some means of generating random numbers (e.g. the `RAND()` function in Excel, `runif()` in R). Unless otherwise stated, such numbers are generated from the *uniform distribution* on (0,1), although sometimes other options are provided.

What is not always appreciated is that such 'random' numbers are in fact only *pseudo-random*, as they are generated by a deterministic algorithm.

For this chapter, the important question is: How can we use the built-in uniform random number generator to produce simulated samples from any desired distribution, say the random variable X with (cumulative) distribution function given by $F(x)$.

Probability integral transform

One way to sample from a distribution $f(x)$ is inverse transform sampling. Inverse transform sampling makes use to the probability integral transform, which makes use of a well-known and elementary theorem.

Consider a random variable X with cumulative distribution function F_X. Then the random variable $Y = F_X(X)$ has the standard uniform distribution.

Proof:

$$\begin{aligned} F_Y(t) &= P(Y \leq y) \\ &= P(F_X(X) \leq y) \\ &= P(X \leq F_X^{-1}(y)) \\ &= F_X(F_X^{-1}(y)) \\ &= y \end{aligned}$$

This is true for the uniform distribution: $\Pr[U \leq a] = a$ for any $0 \leq a \leq 1$. Therefore $Y \sim U(0,1)$.

$$\Pr[X \leq x] \;=\; \Pr[U \leq F(x)] \;=\; F(x)$$

This suggests the following algorithm for sampling from $f(x)$.

Step 1: Generate a realization (say u) from the uniform distribution on [0,1].

Step 2: Solve for x in the (generally non-linear) equation $F(x) = u$; use this as a realization of X. Formally, we often write this in the form $x = F^{-1}(u)$, where $F^{-1}(u)$ is defined as the "inverse function".

For *continuous* random variables, we have to solve the equation $F(x) = u$, i.e. find $F_X^{-1}(u)$. In some cases this is simple, as the following examples illustrate.

Example: Exponential distribution. Suppose X has probability density function given by:

$$f(x) \;=\; \frac{1}{\mu} e^{-x/\mu}$$

The distribution function is then:

$$F(x) \;=\; 1 - e^{-x/\mu}$$

and the solution to $F(x) = u$ is thus $x = -\mu \ln(1-u)$. In fact, there is a slight simplification possible by realizing that if U is uniformly distributed, then so is $1-U$. We can therefore replace $1-U$ by U, in which case the algorithm requires only:

- Generate $U = u$ from the uniform distribution.
- Return $X = -\mu \ln u$

Inverse transform sampling for discrete random variables: Suppose that the discrete random variable X takes on non-negative integer values only. Let $\Pr[X = k] = p_k$ for any non-negative integer k. Then $F(0) = 0$, and for $k > 0$, $F(k) = F(k-1) + p_k$. For any non-integer x such that $k < x < k+1$, $F(x) = F(k)$. The algorithm to generate X is thus implemented as follows:

Step 1: Generate a realization (say u) from the uniform distribution on [0,1]. Set $k = 0$ and $F = p_0 = 0$.

Step 2:
- **i**f $u \leq F$, return $X = k$
- else set $k = k + 1$, $F = F + p_k$ and repeat step 2.

The Box-Muller method for the normal distribution

No closed form expression exists for the cumulative distribution function of the normal distribution, so that the algorithm does not apply directly (although see below for an approximation). Consider, however, a *pair* of independent standard normal deviates, say X and Y. The joint p.d.f. is given by:

$$f(x, y) \;=\; \frac{1}{2\pi} \exp[-\frac{1}{2}(x^2 + y^2)]$$

Now transform to two new random variables V and Φ, defined such that $X = \sqrt{2V}\cos(\Phi)$ and $Y = \sqrt{2V}\sin(\Phi)$. It can be shown that the joint p.d.f. of V and Φ is given by:

$$f(v, \phi) \;=\; \frac{1}{2\pi} e^{-v}$$

over the region defined by $0 < v < \infty$ and $0 < \phi < 2\pi$. One can further show that V and Φ are independent random variables, V being exponential with mean 1, and Φ being uniform on the interval $[0, 2\pi]$.

To simulate random values X and Y from $N(0, 1)$ we can thus simulate realizations of V and Φ from a uniform and exponential

distribution, and then transform them into the desired X and Y.

In many cases, one or both of the following complications may arise:

- The function $F(x)$ may not be expressible in closed form; for example the distribution function for the gamma distribution can only be calculated numerically, or looked up in tables.
- Even if an expression for $F(x)$ is available, it may not be possible to solve for x from $F(x) = u$ in closed form.

There do exist many good approximation formulae for both $F(x)$ and, more importantly in our context, for the inverse function $F^{-1}(u)$. The following are two examples of such approximations taken from Abramowitz and Stegun's (1964) *Handbook of Mathematical Functions*.

Normal Distribution: For the standard normal distribution, define $Q(z) = \Pr[Z > z]$. Let z_p be such that $Q(z_p) = p$, i.e. $z_p = Q^{-1}(p)$. For $0 < p \leq 0.5$, the following approximation has a maximum absolute error of 0.003:

$$z_p \approx t - \frac{a_0 + a_1 t}{1 + b_1 t + b_2 t^2}$$

where:

$$z_p \approx t - \frac{a_0 + a_1 t}{1 + b_1 t + b_2 t^2}$$

where:

$$t = \sqrt{-2 \ln p}$$

and

$$\begin{array}{ll} a_0 = 2.30753 & b_1 = 0.99229 \\ a_1 = 0.27061 & b_2 = 0.04481 \end{array}$$

For $0.5 < p < 1$, use $(1 - p)$ in place of p in the formula, and reverse the sign of the result.

Beta Distribution: For the Beta distribution with parameters a and b, the cumulative distribution function is:

$$F(x; a, b) = \frac{1}{B(a, b)} \int_0^x s^{a-1}(1-s)^{b-1} ds.$$

Let x_p be the solution to $F(x; a, b) = p$, and let $z_p = Q^{-1}(p)$ as defined for the normal distribution above. Then:

$$x_p \approx \frac{a}{a + be^{2w}}$$

where:

$$w = \frac{z_p\sqrt{h+\lambda}}{h} - \left(\frac{1}{2b-1} - \frac{1}{2a-1}\right)\left(\lambda + \frac{5}{6} - \frac{2}{3h}\right)$$

$$h = 2\left(\frac{1}{2a-1} + \frac{1}{2b-1}\right)^{-1}$$

$$\lambda = \frac{z_p^2 - 3}{6}$$

Exercise: Use the `qbeta` function and the idea of the probability integral transform to generate 1000 values from a Beta(5,1) distribution.

Often, however, we need to generate random numbers from distributions for which no closed form inverse solutions (exact or approximate) are available. Some approaches which can be applied in such situations are reviewed in the next two sections.

Accept-Reject methods

Suppose that we wish to generate a random number X from a distribution defined by the p.d.f. $f(x)$, but that no inverse transformation procedure is available. Suppose, however, that there exists another p.d.f., say $h(x)$, from which we can easily generate random numbers, and which is such that $h(x) = 0$ only when $f(x) = 0$. The idea is that the density $h(x)$ should be *similar* in some sense to $f(x)$, and that we will generate from the distribution defined by $h(x)$, but discard (reject) a certain

proportion of the values generated so as to compensate for the differences.

In order to describe the algorithm, let us first define two other quantities:

$$C \;=\; \max_{\{x|h(x)>0\}} \frac{f(x)}{h(x)}$$

and

$$g(x) \;=\; \frac{f(x)}{C\,h(x)}$$

which is also only defined when $h(x) > 0$. Note that since both $f(x)$ and $h(x)$ have unit integrals, one cannot lie entirely above the other for all x; it thus follows that $C \geq 1$. By definition of C, and non-negativity of the density functions, we also have that $0 \leq g(x) \leq 1$.

The algorithm then proceeds as follows:

- Generate a random variable Y from the distribution having p.d.f. $h(y)$; and an independent random variable U drawn from the standard uniform distribution (i.e. $0 < U < 1$). Let y and u be the values generated.
- If $u \leq g(y)$ then ACCEPT the value, i.e. set $X = y$; otherwise REJECT both values and repeat the previous step.

When $g(y) = 1$, i.e. at the point where $h(y)$ is at its *lowest* relative to $f(y)$, we always accept. We accept increasingly less frequently as we move into regions in which $h(y)$ becomes larger relative to $f(y)$, always rejecting in any region in which $f(y) = 0$ while $h(y) > 0$, i.e. when $g(y) = 0$. We never generate values in any region for which $h(y) = 0$.

This algorithm generates random variables Y which have the *conditional* distribution for Y given the event $\{U \leq g(Y)\}$. The conditional density $h(y|U \leq g(Y))$ is obtained from Bayes' theorem as follows:

$$h(y|U \leq g(Y)) \;=\; \frac{h(y)\,\Pr[U \leq g(Y)|Y = y]}{\Pr[U \leq g(Y)]}$$

Trivially, $\Pr[U \leq g(Y)|Y = y] = \Pr[U \leq g(y)] = g(y)$, and thus also:

$$\begin{aligned}\Pr[U \leq g(Y)] &= \int_{\mathcal{Y}} \Pr[U \leq g(Y)|Y = y]\, h(y)dy \\ &= \int_{\mathcal{Y}} g(y)h(y)dy \\ &= \frac{1}{C}\int_{\mathcal{Y}} f(y)dy \;=\; \frac{1}{C}\end{aligned}$$

where $\mathcal{Y}$ is range of values for which $h(y) > 0$.

Putting this all together gives:

$$h(y|U \leq g(Y)) \;=\; Ch(y)g(y) \;=\; f(y)$$

which is the desired density. As a bonus, we have also established that the probability of an acceptance is $1/C$. Since each iteration of the procedure is independent, the number of iterations required is the number of rejections before the first acceptance. This number has the geometric distribution with parameter $1/C$, and the expected number is thus C. Recall that $C > 1$, but as the functions $f(.)$ and $h(.)$ approach each other we will have $C \to 1$. But recall that C is determined by a maximum ratio; two densities which are superficially similar may differ in their tails (where both densities tend to zero) yielding surprisingly large ratios. It is thus important to ensure that the density $h(x)$ is at least as heavy-tailed as $f(x)$.

Example: Suppose that we wish to generate random numbers from the gamma distribution, with p.d.f. as follows:

$$f(x) \;=\; \frac{x^{\alpha-1}e^{-x}}{\Gamma(\alpha)}$$

For integer values of α, x can be generated as the sum of α exponentially distributed random variables; for half-integer values, x can be generated from the sum of the squares of 2α normally distributed random variables (since this gives the χ^2 distribution which is a special case of the gamma). Neither of these approaches help if α is not an integer or half-integer, and in any case, both methods become computationally expensive as α gets to even

moderate size. Let us, therefore, attempt an acceptance-rejection approach.

For the density $h(x)$, we could use the exponential distribution with the same mean as $f(x)$, α. Thus:

$$h(x) \;=\; \frac{1}{\alpha} e^{-x/\alpha}$$

and:

$$\frac{f(x)}{h(x)} \;=\; \frac{\alpha x^{\alpha-1} \exp[-(1-\frac{1}{\alpha})x]}{\Gamma(\alpha)}$$

The maximum occurs at $x = \alpha$, giving:

$$C \;=\; \frac{\alpha^{\alpha} e^{1-\alpha}}{\Gamma(\alpha)}$$

and

$$g(x) \;=\; \left(\frac{x}{\alpha}\right)^{\alpha-1} \exp[-\left(\frac{\alpha-1}{\alpha}\right) x + \alpha - 1]$$

It is interesting to record the size of C, and the numbers of logarithms and generations of uniform random deviates expected per single generation of x, for various values of α:

α	C	No. of logs ($= 3C$)	No. of uniforms ($= 2C$)
5	2.39	7.15	4.77
10	3.40	10.20	6.80
15	4.18	12.53	8.35
20	4.83	14.49	9.66

Unnormalized densities

The above explanation did assume that the probability densities given by $f(x)$ and $h(x)$ were fully defined, i.e. that they were normalized to integrate to 1. Suppose now, however, that only the functional forms are known, but not the normalizing constants. In other words, suppose that $f(x) = \kappa\phi(x)$ and $h(x) = \lambda\psi(x)$, where the functions $\phi(x)$ and $\psi(x)$ are non-negative, but the constants κ and λ are still undetermined.

Situations such as this occur particularly when using Monte Carlo methods to generate *Bayesian* posterior distributions. Typically, we might wish to generate values from the *posterior distribution} on a parameter θ (e.g. to evaluate expected costs of different actions). Using the nomenclature from the third year notes on decision theory, the posterior distribution for θ given a set of data can be expressed as (see notes for explanation):

$$\pi(\theta|\mathbf{data}) \propto \pi(\theta)\Lambda(\theta;\mathbf{data})$$

where $\pi(\theta)$ and $\pi(\theta|\mathbf{data})$ are the prior and posterior densities respectively, and $\Lambda(\theta;\mathbf{data})$ is the likelihood function. When non-conjugate priors are used, the posterior is typically not in a recognizable form, and the appropriate normalizing constant cannot be determined analytically.

In principle, the normalization constant can be determined by numerical integration, but especially for multivariate problems this can be computationally burdensome and subject to substantial imprecisions. In fact, often the best way to do the integration might be to use Monte Carlo methods! It is thus very useful to be able to generate random numbers from the desired distribution without necessarily determining κ or λ. This is no problem when using acceptance-rejection methods.

As defined before:

$$C \;=\; \max_{\{x|h(x)>0\}} \frac{f(x)}{h(x)} = \rho \max_{\{x|\psi(x)>0\}} \frac{\phi(x)}{\psi(x)} = \rho\xi \quad \text{(say)}$$

where $\rho = \kappa/\lambda$. Then, we can define:

$$g(x) \;=\; \frac{f(x)}{C\,h(x)} = \frac{\kappa\phi(x)}{\rho\xi\lambda\psi(x)} = \frac{\phi(x)}{\xi\psi(x)}$$

which can be calculated without knowing κ or λ, and the method can still be applied. The only problem is that we do not know the value of C, and thus cannot estimate the probability of acceptance up front.

Bayesian example: Suppose that we observe a single sample value X from a Gamma distribution with parameters ½ and θ, i.e. with a p.d.f. given by:

$$\theta^{1/2}e^{-\theta x}.$$

A "conjugate prior" would be a Gamma distribution for θ. But suppose that we prefer to model prior uncertainty by the standard half normal distribution, i.e. the standard normal distribution truncated to positive values with prior p.d.f. proportional to:

$$e^{-\theta^2/2} \quad \text{for } \theta > 0.$$

Suppose now that we observe the value $X = 0.25$. The posterior density is then proportional to:

$$\theta^{1/2} e^{-0.25\theta - 0.5\theta^2} = \phi(\theta), \text{ say,}$$

which is not of any recognizable form. It is nevertheless simple to find the mode of the posterior density as a crude location estimator, by maximizing the logarithm of the p.d.f.

$$\text{constant } + 0.5 \ln \theta - 0.25\theta - 0.5\theta^2.$$

It is left as an exercise to show that the mode occurs at $\theta = 0.593$. (Remember that negative values are not permissible!)

In generating values from the distribution with density proportional to $\phi(\theta)$, we could try an exponential distribution as the basis distribution in an acceptance-rejection method, possibly with a mean of 0.593. In other words, $\psi(\theta) = e^{-1.686\theta}$. (The constant is 1.686 in this case, but we do not need it.) The logarithm of the ratio $\phi(\theta)/\psi(\theta)$ is given by:

$$0.5 \ln \theta + 1.436\theta - 0.5\theta^2$$

for which the maximum (across positive values of θ) occurs at $\theta = 1.726$. The corresponding value for ξ is 3.532. The algorithm thus consists of the following steps:

- Generate a value of θ from the exponential distribution with a mean of 0.593
- Calculate:

$$g(\theta) = \frac{\theta^{1/2} e^{1.436\theta - 0.5\theta^2}}{3.532}$$

- Generate a u from the uniform distribution, and accept this value of θ if $u \leq g(\theta)$.

The above algorithm was coded and implemented so as to generate 10000 simulated values from the posterior distribution. This turned out to require a total of 25833 values to be generated from the basis exponential distribution, i.e., the probability of acceptance was approximately 0.39. The resulting posterior distribution, compared with the prior distribution, is illustrated by the histograms in Figure 1. There's not much of a difference, but what can you expect from a sample of size 1?!

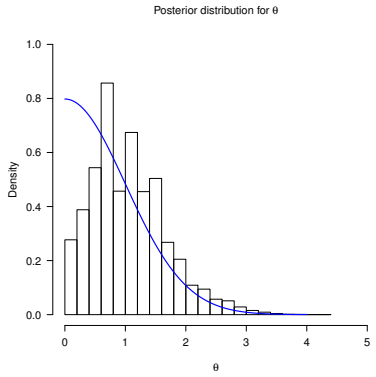


Figure 1: Monte Carlo generated histogram representing the posterior distribution in the example, compared to the prior distribution (solid line)

A particularly convenient way of applying the above ideas for generation of values from the Bayesian posterior is to use the prior distribution for the initial generation. In our earlier notation, we would then have:

$$\phi(\theta) = \pi(\theta)\Lambda(\theta; \mathbf{data})$$

and

$$\psi(\theta) = \pi(\theta)$$

so that the ratio to be maximized in order to define $g(\theta)$ will be:

$$\frac{\phi(\theta)}{\psi(\theta)} = \Lambda(\theta; \mathbf{data})$$

the likelihood function.

Exercise: Generate 10000 values from the above posterior distribution. Find the posterior mean and an interval estimate for θ.

Importance Sampling

The aim of Monte Carlo methods is (often) to evaluate an integral or expectation of the form

$$E[h(x)] = \int h(x)f(x)dx.$$

Simple Monte Carlo simulations would simulate from $p(x)$ and use this Monte Carlo sample to evaluate the integral. This can

be very wasteful and inefficient (large variance) when $f(x)$ is zero over most of the domain of $p(x)$ and only non-zero where $p(x)$ is very small, e.g. for tail probabilities.

Importance sampling helps with this problem by generating values from a distribution different than $p(x)$, let's call this $h(x)$, where $h(x)$ is much more likely to generate values in the area of interest (where we want to evaluate $f(x)$), than $p(x)$.

The other situation where we can use importance sampling is when we can't (or don't know how to) sample from $p(x)$, but can find a function $h(x)$ with same support as $p(x)$, which we CAN easily sample from.

The trick is to rewrite the above integral as

$$\theta = E[h(x)] = \int h(x)f(x)dx = \int h(x)\frac{f(x)}{g(x)}g(x)dx.$$

The $\frac{f(x)}{g(x)}$ are called *importance weights*. An estimate of the integral can be obtained as

$$\hat{\theta} = \frac{1}{N}\sum_{i=1}^{N} h(x_i)\frac{f(x_i)}{g(x_i)},$$

with x_i sampled from $g()$.

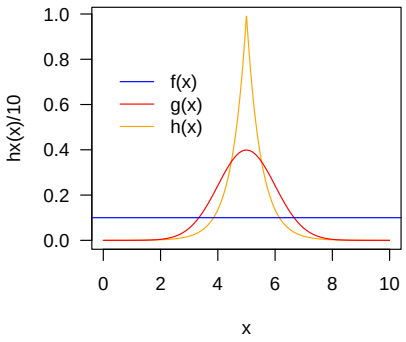
Example

Suppose we want to evaluate the integral

$$\int_0^{10} \exp(-2|x-5|)dx.$$

This can be written as $\int_0^{10} 10\exp(-2|x-5|)\frac{1}{10}dx$ with $h(x) = 10\exp(-2|x-5|)$ and $f(x) : X \sim U(0,10)$. The true value of this integral is close to 1.

We could sample from the uniform distribution:



```r
hx <- function(x) {
  hx <- 10 * exp(-2 * abs(x-5))
  return(hx)
}

N = 10000
x <- runif(N, 0, 10)

est <- mean(hx(x))
mcse <- sd(hx(x)) / sqrt(N)
```

Estimate	MC.Error
1.0192	0.0203

The above code does not make use of importance sampling, but is just simple Monte Carlo integration (see section on Monte Carlo Integration).

With importance sampling we can make this more efficient. Let's choose a normal distribution centered at 5 for $h(x)$. This distribution puts much more weight on the area of interest.

```r
xvals <- rnorm(N, 5, 1)
iw <- dunif(xvals, 0, 10) / dnorm(xvals, 5, 1)
hxx <- hx(xvals)
est.is <- sum(hxx * iw) / N
mcse.is <- sd(hxx * iw) / sqrt(N)
```

```r
knitr::kable(data.frame(IS.est = est.is,
                        "IS error" = mcse.is), digits = 4)
```

IS.est	IS.error
0.9969	0.006

The Monte Carlo error of the estimate from importance sampling is smaller: it is roughly 30% compared to that of the previous estimate, for the same sample size.

Markov Chain Monte Carlo (MCMC) methods, Gibbs Sampling

In this section, we briefly outline a numerical method which has become popular in recent years, for generating random variables from multivariate distributions (although we shall restrict ourselves to the bivariate case only). Suppose that we need to generate bivariate pairs (X, Y) for which the joint

References

1. Taboga M. 2021. Importance sampling: Lectures on probability theory and mathematical statistics. Kindle Direct Publishing. Online appendix. https://www.statlect.com/asymptotic-theory/importance-sampling
2. Owen AB. Importance Sampling. In: Monte Carlo theory, methods and examples. https://artowen.su.domains/mc/Ch-var-is.pdf
3. Kennedy T. Importance Sampling. https://www.math.arizona.edu/~tgk/mc/book_chap6.pdf
4. Tokdar ST, Kass RE. 2010. Importance sampling: a review. WIREs Computational Statistics 2(1):54–60. https://doi.org/10.1002/wics.56

distribution function is not available in closed analytical form. Suppose, however, that we are able to obtain or to estimate both conditional probability distributions, i.e. the distribution of $X|Y$ and of $Y|X$. For example, we might have determined that the conditional probability densities are of the following forms:

$$f_{X|Y}(x|y) = \frac{ye^{-yx}}{1-e^{-y}}$$

$$f_{Y|X}(y|x) = \frac{xe^{-xy}}{1-e^{-x}}$$

for $0 < x < 1$ and $0 < y < 1$.

In general it may be very difficult to derive the joint or marginal distributions from the conditionals, let alone invert the cumulative distributions to generate the desired random numbers. The following scheme may, however, be relatively simple to apply, requiring only that we are able to generate random numbers from the two conditional distributions:

Step 0: Choose an arbitrary value for X having non-zero probability density. Call this x_0. Set $k = 1$.

Step 1: Generate a realization of Y from the conditional distribution for Y given $X = x_{k-1}$. Call this y_k.

Step 2: Generate a realization of X from the conditional distribution for X given $Y = y_k$. Call this x_k. Set $k = k + 1$ and repeat from step 1.

This generates sequences of x_k and y_k values according to a Markov process. It turns out that under mild assumptions, the process leads to a stationary asymptotic distribution which is precisely the marginal bivariate distribution of (X, Y). Clearly, consecutive values will not be independent of each other, so we can't use the sequence directly as a random sample from the bivariate distribution, even after allowing the process to reach approximate stationarity. In practice, we could choose some sufficiently large number, say K, and use every K-th pair of values generated.

Two practical problems relate to how long the process must run before stationarity is achieved, and to choice of the K to ensure independence. These are non-trivial problems, but perhaps beyond the scope of the current course.

MCMC approaches are particularly relevant to problems of Bayesian inference.

Monte Carlo Integration

Monte Carlo integration can be justified by the *mean value theorem for integrals*:

The average value of a function $f(x)$ over the interval $[a, b]$ is given by

$$f_{avg} = \frac{1}{b-a}\int_a^b f(x)dx.$$

$$\frac{1}{b-a}\int_a^b f(x)dx \approx \frac{1}{b-a}\sum f(x_i)\Delta x = \sum f(x_i)\frac{\Delta x}{b-a} = \frac{\sum f(x_i)}{n}$$

where $\Delta x = \frac{a-b}{n}$.

$$\underbrace{\frac{1}{b-a}\int_a^b f(x)dx}_{\text{average f(x)}} \times (b-a) = area$$

In practice this means that we can estimate the mean value of $f(x)$ (expectation) by simulating a large number of values from a uniform distribution (on the interval over which we want to integrate), multiply by the width of the integral and obtain the area under the curve (integral).

Why simulate from a uniform?

$$E(f(x)) = \int_a^b f(x)p(x)dx = \int_a^b f(x)\frac{1}{a-b}dx = \frac{1}{a-b}\int_a^b f(x)dx$$

where $X \sim U(a,b) \rightarrow p(X=x) = \frac{1}{a-b}$.

Variance reduction techniques

We can always view a Monte Carlo simulation as providing an estimate of the expectation of a random variable, say θ=E[X], based on a sample of size n, say. For example, even if we interested in tail probabilities of the form $\theta = \Pr[Y > c]$ for some random variable Y, we can always define X to be the indicator variable of the event $\{Y > c\}$, i.e. $X = 1$ if $Y > c$ and $X = 0$ otherwise. Clearly then E[X]=θ.

If we simply observe n *independent} realizations of X, and use $\bar{X}$ as the estimate $\hat{\theta}$, then the standard deviation of the estimate is $\sigma_X/\sqrt{n}$. This can require very large numbers of iterations to obtain the sorts of accuracy often desirable in numerical work, which can be problematical both practically (time required) and theoretically (if n is larger than the cycle length of the random number generator). There are however a number of procedures which can be used to reduce the variance of the estimator substantially. We discuss three of these. Note that they are not mutually exclusive – it is often possible to combine the use of all three.

Antithetic sampling

Suppose we generate the X's in negatively correlated pairs. For example, suppose that the generated value of X_{2j-1} is obtained from the uniform random number U_j, and that of X_{2j} from $1 - U_j$, for $j = 1, \ldots, \frac{n}{2}$. Then:

$$\begin{aligned} Var[X_{2j-1} + X_{2j}] &= Var[X_{2j-1}] + Var[X_{2j}] + 2Covar[X_{2j-1}, X_{2j}] \\ &\leq Var[X_{2j-1}] + Var[X_{2j}] \end{aligned}$$

and thus:

$$Var[\sum X_i] \ \leq \ \sum Var[X_i]$$

In other words, we find two identically distributed estimators, say $\hat{\mu}_1$ and $\hat{\mu}_2$, that are negatively correlated. Averaging these estimators will be superior to using either estimator alone with double the sample size, since the estimator

$$\hat{\mu}_{AS} = (\hat{\mu}_1 + \hat{\mu}_2)/2$$

has variance equal to

$$Var[\hat{\mu}_{AS}] = \frac{1}{4}\left(Var[\hat{\mu}_1] + Var[\hat{\mu}_2]\right) + \frac{1}{2}Cov[\hat{\mu}_1, \hat{\mu}_2] = \frac{(1+\rho)\sigma^2}{2n},$$

where ρ is the correlation between the two estimators and σ^2/n is the variance of either estimator using a sample size of n (Givens and Hoeting, 2005).

The following simple example illustrates this approach. The examples used here are trivial (in the sense that simulation is not needed to obtain the answers), but serve to demonstrate what can be achieved.

Example: Suppose that Y is exponentially distributed with a mean of $\frac{1}{5}$, and that we wish to obtain (or estimate) the value of $\theta = E[e^{2Y}]$. If we set $X = e^{2Y}$ then $\theta = E[X]$, and the variance of $\bar{X}$ for independent sampling (or "brute force" Monte Carlo) is $2.222/n$. We can however generate the exponentially distributed Y's in pairs as $Y_{2j-1} =$

$-\frac{1}{5}\ln U_j$ and $Y_{2j} = -\frac{1}{5}\ln(1-U_j)$. The variance based on n realizations of Y is now $1.855/n$; if we use the full set of n uniform random variables as before, we generate $2n$ realizations of Y which reduces the variance of the estimator to $0.928/n$. Thus we can either achieve a nearly 20% improvement in variance while halving the number of pseudo-random numbers needing to be generated, or a nearly 60% reduction using the same number of pseudo-random numbers.

```
N <- 10000
uu <- runif(N)
y1 <- 1/5 * log(uu)
y2 <- 1/5 * log(1 - uu)

x1 <- exp(2 * y1)
x2 <- exp(2 * y2)

# antithetic: average of two correlated values
x12 <- (x1 + x2) / 2

est.ant <- mean(x12)
se.ant <- sd(x12) / sqrt(N)

# ignoring correlation
y <- c(y1, y2)
x <- exp(2*y)

est.sim <- mean(x)
se.sim <- sd(x) / sqrt(2*N)
```

	estimate	MC error
classical	0.71406	0.00151
antithetic	0.71406	0.00049

Control variates

Suppose that we can find a random variable W, for which the expectation E[W] is known, and which is positively correlated with X. Then define $Z = X - W + E[W]$. Clearly E[Z]=E[X]=θ, and thus the average from a random sample of Z will also be an estimator of θ. But the variance of Z is Var[X]+Var[W]-2Covar[X, W], which is less than Var[X]

provided that Var[W]<2*Covar*[X,W$]. If X and W have approximately equal variances, this condition requires that the coefficient of correlation be greater than 0.5.

Example (continued): Continuing with the example of the previous section, we note that for the exponential distribution, the standard deviation equals the expectation. Thus $Y = 0.8$ is three standard deviations above the mean of Y (and in fact $\Pr[Y > 0.8] < 0.02$). Let us approximate X by a linear function over this range. At $Y = 0$, $X = 1$; while at $Y = 0.8$, $X = 4.95$. A crude approximation may be $X \approx 1 + 5Y$. Since constant terms do not affect correlations, let us define $W = 5Y$, which should then be well correlated with X. Clearly, E[W]=1, and thus we have $Z = X - 5Y + 1$. The sample average of Z becomes our estimate for θ.

It can be shown (see Appendix) that Var[Z]=1, and thus the variance of the estimator is Var[$\bar{Z}$]=$1/n$.

Exploitation of conditional expectations

Once again we seek an estimate of $\theta = E[X]$, where by assumption this expectation cannot be obtained in an analytical form. Let us suppose, however, that there exists another random variable, say V, such that the *conditional expectation} E[$X|V$] is easily calculated (i.e. relative to calculating E[X] itself). A well-known result is:

$$\begin{aligned} Var[X] &= Var[E(X|V)] + E[Var(X|V)] \\ &\geq Var[E(X|V)] \end{aligned}$$

Define the random variable $Z = E[X|V]$, i.e. the random variable taking on the value E[$X|V = v$] whenever $V = v$. By a standard result, E[Z]=E[E($X|Y$)]=E[X], as required, while the above demonstrates that Var[Z]$\leq$Var[X]. By clever choice of V, the difference in variances can be made quite substantial. The Monte Carlo procedure can thus generate values for the random variable V, transform each V into its corresponding Z, and use the sample average of the Z's as the required estimate for θ.

Example: W and V are independent random variables. V is the standard truncated normal, i.e.:

$$f_V(v) \;=\; \sqrt{\frac{2}{\pi}} e^{-\frac{1}{2}v^2}$$

for $0 < v < \infty$, while W is exponential with mean of 1. We are required to estimate $\theta = \Pr[W > V]$. The direct ("brute force") approach is to obtain a sample average of X, where $X = 1$ if $W > V$, and $X = 0$ otherwise. In fact, $\sum_{i=1}^{n} X_i$ is binomially distributed with parameters θ and n, from which it follows that $\mathrm{Var}[\bar{X}]=\theta(1-\theta)/n$. In this case, the true value for θ is 0.5233, and thus the variance of $\bar{X}$ will be $0.2495/n$.

But $\mathrm{E}[X|V = v]=\Pr[W > V|V = v] = \Pr[W > v] = e^{-v}$. We can thus define $Z = e^{-V}$, and use the sample average of the Z's by generating only the V sequence (and not the W's at all). It turns out that $\mathrm{Var}[\bar{Z}]$ is $0.0624/n$, which is obtained with half the number of random number generations.

Generation of pseudo-random numbers from the uniform distribution

This Appendix gives an outline of the manner in which uniformly distributed pseudo-random numbers may be generated. A sequence of numbers is "pseudo-random'' if it arises from the execution of a deterministic algorithm, but has statistical properties which are indistinguishable (on the basis of any known statistical methods) from a truly random sequence of uniformly distributed numbers.

Most pseudo-random number generators are of the *congruential* type, i.e. the numbers are derived from an algorithm of the following form:

step 0: Choose an integer seed z_0.

step 1: Set $z_1 = (a\, z_0 + b)$ (*modulo* c) for some given integer values a, b and c (where "modulo" c means remainder after dividing by c, and thus $0 \le z_1 \le c - 1$).

The usual implementation is to set the seed equal to the integer z_1 generated in the previous call to the random number generator (see step 2 below). Thus within any program, the seed is usually set externally only once (or at most a small number of times). In some packages, the user can set the seed directly, but in others it is chosen in the algorithm, for example by reading the system clock. User-control over the initial seed is useful for debugging, in that this ensures that the same sequence of random numbers is used in every run, which helps in investigating bugs which only occur occasionally.

step 2: Return $r = z_1/c$ as the "random number", set $z_0 = z_1$ as the seed for the next random number.

One point to note immediately is that eventually the seed must return to a value used before, since there are only c integers between 0 and $c-1$. Once the seed repeats, the generator will simply cycle through the same sequence of random numbers over and over again (where the cycle length is at most c). It is necessary thus to make c as large as possible, which usually means the maximum integer which can be represented on the machine and in the language being used (quite often $2^{31}-1$, but also frequently only $2^{15}-1 = 32767$).

For any c, choice of a and b is quite critical. As a very simple example, suppose $c = 10$ (which of course means that z_1 can only take on the values $0, 1, \ldots, 9$, with corresponding pseudo-random numbers $0.0, 0.1, \ldots, 0.9$). After at most 10 generations, the same sequence will start repeating. Now suppose that we choose $a = 5$ and $b = 3$; this results in a cycle of length 2, alternating between 0.3 and 0.8, which is hardly random! Changing a to 7 results in cycles of length 4 for most seeds, which is a slight improvement, but the numbers then lie either all in the top half, or all in the bottom half of the range!

As a slightly less trivial example, let $c = 100$, and use $a = 337$ and $b = 42$. Starting with $z_0 = 17$ gives a return to this seed after 20 iterations. But with $b = 43$, the seeds cycle through only four values, *viz.* 17, 72, 7 and 2. A rather better sequence for $c = 100$ is obtained with $a = 291$ and $b = 37$. The cycle length here is 50 (in comparison with the theoretical maximum of 100). The first 50 values generated with a seed of 17 are displayed in Figure 2. By eye these appear quite random, and the first order auto-correlation is only -0.01. The second order correlation is however 0.12, which may be a little bothersome. Sometimes auto-correlation can be a problem with random number generators, and one should test for this before using the generator. There are, however, schemes for shuffling the order in which the generated values are used to create greater independence.

Most random number generators in commercial packages today use $b = 0$ and $c =$ the maximum integer which can be repre-

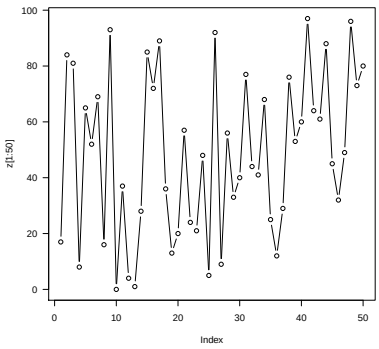


Figure 2: Sequence of pseudorandom numbers ($a = 291$, $b = 37$, $c = 100$)

sented. Effort is then made to fine-tune a so that maximum cycle length and minimum autocorrelation is achieved. How well this is achieved is hard to say without extensive testing.

Technical Calculations for the Variance Reduction

Example for antithetic sampling

Since $X = e^{2Y}$, the expectation of X^r is:

$$E[X^r] \;=\; 5\int_0^\infty e^{-(5-2r)y}dy \;=\; \frac{5}{5-2r}$$

Thus $\theta = E[X] = 5/3$, while the variance of X is $E[X^2] - \theta^2 = 5 - (5/3)^2 = 2.222$.
For each pair of observations generated from the same underlying uniform random deviate we would have that:

$$E[X_{2j-1}X_{2j}] = E[e^{-\frac{2}{5}\ln(U)}.e^{-\frac{2}{5}\ln(1-U)}] = E[U^{-\frac{2}{5}}(1-U)^{-\frac{2}{5}}]$$

where U has the standard uniform distribution. It is easily seen that this is the Beta function with parameters $\frac{3}{5}$ and $\frac{3}{5}$, i.e.:

$$B(\frac{3}{5},\frac{3}{5}) = \frac{\left[\Gamma(\frac{3}{5})\right]^2}{\Gamma(\frac{6}{5})} = 2.4109$$

which is obtained from tables of the gamma function. Thus the covariance of X_{2j-1} and X_{2j} is $2.4109 - (1.6667)^2 = -0.3669$ (since both X_{2j-1} and X_{2j} have mean of θ); as both variables also have the same variance, the variance of $X_{2j-1} + X_{2j}$ is $2 \times (2.222) + 2 \times (-0.3669) = 3.711$. Since each pair will still be independent, the variance of $\sum_{i=1}^n X_i$ will be $\frac{n}{2} \times 3.711 = 1.855n$, and the variance of the mean then $1.855/n$.

Example for control variates

We have defined $Z = X - 5Y + 1$, and know that E[Z]=θ. The variance of Z is Var[X]+25.Var[Y]-10.Covar[X, Y]. We already know that Var[X]=2.222, and that Var[Y]= $\{E[Y]\}^2$= 0.04 (from standard properties of the exponential distribution). In order to find the covariance, we first calculate:

$$\begin{aligned} E[XY] &= \int_0^\infty 5ye^{2y}e^{-5y}dy \\ &= 5\int_0^\infty ye^{-3y}dy \;=\; \frac{5}{9} \;=\; 0.5556 \end{aligned}$$

Thus the Covariance of X and Y is 0.5556-(1.6667)(0.2)=0.2222, and the variance of Z is thus precisely 1.

Example exploiting conditional expectations

Starting immediately with $Z = \Pr[W > V|V] = e^{-V}$, we can in fact calculate the moments of Z directly (without Monte Carlo methods - this was only an illustration!).

$$\begin{aligned} E[Z^k] = E[e^{-kV}] &= \int_0^\infty \sqrt{\frac{2}{\pi}}e^{-v^2/2-kv}dv \\ &= \int_0^\infty \sqrt{\frac{2}{\pi}}e^{-(v+k)^2/2}e^{k^2/2}dv \\ &= e^{k^2/2}\int_k^\infty \sqrt{\frac{2}{\pi}}e^{-y^2/2}dy \\ &= e^{k^2/2}\Pr[V > k] \end{aligned}$$

where the third line follows by transformation of the variable of integration from v to $y = v + k$, and the last line by recognition of the p.d.f. of the truncated normal distribution. For $k > 0$, $\Pr[V > k] = 2(1-\Phi(k))$, where $\Phi(x)$ is the distribution function of the standard normal distribution. Thus from tables of the normal distribution, we obtain $\theta = E[Z] = 2e^{\frac{1}{2}}(1 - \Phi(1)) = 0.5233$ and $E[Z^2] = 2e^2(1 - \Phi(2)) = 0.3362$. The variance of Z is then easily found to be 0.0624.

References

- Andrieu C, De Freitas N, Doucet A, Jordan MI. 2003. An Introduction to MCMC for Machine Learning. Machine Learning, 50, 5–43.
- Eckhardt R. 1987. Stan Ulam, John von Neumann, and the Monte Carlo Method. Los Alamos Science, Special Issue.
- Geyer CJ. 2011. Introduction to Markov chain Monte Carlo. Handbook of Markov Chain Monte Carlo, 20116022, p.45.
- Givens GH, Hoeting JA. 2013. Computational Statistics. Wiley.
- MacKay DJC, 2003. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press.
- MacKay DJC, *Introduction to Monte Carlo Methods*.
- Metropolis N and Ulam S, 1949. The Monte Carlo method. *J. American Statist. Assoc.* 44: 335-341
- Metropolis N. 1987. The beginning of the Monte Carlo Method. Los Alamos Science, Special Issue.
- Rizzo ML. 2007. Statistical computing with R. Chapman and Hall/CRC.
- Robert CP and Casella G. 2004. *Monte Carlo Statistical Methods*. 2nd edition. Springer.
- Tanner MA. 1996. *Tools for Statistical Inference: Methods for the Exploration of Posterior Distributions and Likelihood Functions*. Springer.