Generating Random Variates

* Much of this course will deal with the idea of simulation.

* This requires the ability to generate observations from a specific distribution.

* Sometimes we can generate random numbers by conducting a random experiment (e.g. rolling a die, tossing a coin etc.)

* This is usually quite time-consuming and available only for a very small number of distributions.

* Some physical phenomena are inherently random so observing them can produce random variates. See, for example http://www.random.org which uses random atmospheric noise.

* An issue with these methods, however, is that the sequence is generally not reproducible.

Computer Generation of Random Variates

* Our aim is to use a computer to produce a sequence of independent random numbers.

* Such sequences should be of arbitrary length and be reproducible.

* We must also be able to specify the exact distribution from which the random variates are generated.

* In fact we only need to be able to generate random observations from a Uniform$(0, 1)$ distribution.

* Clever use of these random uniforms can produce sequences from any arbitrary distribution.

Pseudo-random Numbers

* Computers are deterministic machines so can never produce random numbers!

* Instead we will try to generate pseudo-random numbers.

* Pseudo-random numbers are generated from an initial state and a deterministic function.

* Even though they come from a deterministic sequence, the resulting observations should behave like a sample of random Unif$(0, 1)$ observations.

* There are a number of tests that can be applied to the output of a random generator to ensure this is the case.

Uniform Pseudo-random Generators

* Most uniform random generators actually generate integers in the range $0, ..., M-1$ where M is a very large number (e.g. 2^{32}).

* Dividing these numbers by M gives values in $[0, 1)$.

* Eventually the state of the random number generator will return to the initial (or some other) state at which point the sequence of numbers will repeat.

Definition 1

The period of a sequence of numbers is defined to be the smallest integer T such that $x_{i+T} = x_i$ for every $i \geqslant T_0 \geqslant 1$. The value T_0 is called the initial sequence length.

Linear Congruential Generators

Definition 2

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

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

* Any such generator can have period of at most M although it could be smaller.

* Choosing a and b appropriately can get the period very close to M.

* All such generators always generate pairs (x_t, x_{t+1}) which lie on parallel lines although good choices of a and b can ensure that the number of lines is very large.

Shift Generators

* In a computer all integers are stored in a binary representation. That is by the ordered k-tuple $(e_0, e_1, \ldots, e_{k-1})$ where

$$x = \sum_{i=0}^{k-1} e_i 2^i.$$

* Theoretically, for a random number on $0, ..., 2^k - 1$, the components of this k-tuple are independent.
* This is the basis behind shift generators.

Definition 3

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 linear combination of the results modulo the largest integer representable.

* By combining generators in this way the period of the final sequence can be close to the product of the sequences of the individual generators.

* 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 resulting sequence has period of about 2^{95}.

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

The Mersenne-Twister Generator.

* The current state of the art generator is generally believed to be the Mersenne-Twister Generator.
* Matsumoto, M. and Nishimura, T. (1998) Mersenne Twister: A 623-dimensionally equidistributed uniform pseudo-random number generator, *ACM Transactions on Modeling and Computer Simulation*, **8**, 3–30.
* The state of this generator is a vector of 626 integers, the first two of which refer to positions in the remaining vector of length 624.
* The algorithm is rather involved but it results in a generator with period of $2^{19937} - 1$!
* This is the default generator in R.
* R does, however, have a number of other generators and also allows for a user defined generator to be supplied.

Non-uniform Random Number Generation

* In general we want observations from a distribution other than the uniform $(0, 1)$ distribution.

* All current pseudo-random number generators, however, generate "uniform" random variates.

* In what follows I will assume that we are using a good generator so we can treat the observations from that generator as if they are actually distributed as Uniform$(0, 1)$ random variates.

* We will look at various methods to transform these observations into ones that have a specified distribution.

* The function `runif` in R will generate uniform random numbers.

The Probability Integral Transform

Definition 4

Suppose that X is a random variable with cumulative distribution function F. Define the inverse of F to be

$$F^{-1}(u) \;=\; \inf\{x : F(x) \geqslant u\}$$

Theorem 1

Suppose that X is a random variable with cumulative distribution function F whose inverse F^{-1} is defined as above. If $U \sim$ uniform$(0,1)$ then

$$F^{-1}(U) \stackrel{d}{=} X.$$

The Inverse Method to Generate Observations

* The probability integral transform gives us a general way of generating observations from any distribution for which we have a cdf.

* If the cdf F is strictly continuous then the inverse cdf exists and is a function.

* In that case we simply need to find the inverse cdf F^{-1}.

* We then generate $U_1, \ldots, U_n \overset{iid}{\sim}$ uniform$(0, 1)$ and define

$$X_i = F^{-1}(U_i) \qquad i = 1, \ldots, n$$

* The resulting sample $x_1, \ldots, x_n$ will be a sample from the required distribution.

Generating Discrete Random Variables

* Suppose we wish to generate observations from the discrete distribution with possible values $x_1 < x_2 < \cdots < x_k$ with probability mass function

$$\mathsf{P}(X = x_i) \;=\; p_i \qquad i = 1, \ldots, k$$

* The cdf is defined by

$$F(x_j) \;=\; \mathsf{P}(X \leqslant x_j) \;=\; \sum_{i=1}^{j} p_i.$$

* This is not a continuous function but from the Definition 4 we see that $F^{-1}(u) \;=\; x_i \iff F(x_{i-1}) \;<\; u \;\leqslant\; F(x_i)$

* Hence we generate $U \sim$ uniform$(0, 1)$ and set the observation equal to x_i which satisfies

$$F(x_{i-1}) \;<\; U \;\leqslant\; F(x_i)$$

Generating Discrete Random Variables

* This algorithm requires very little computation but does require a number of comparisons.

* Generally one compares U with the sequence of cdf values $F(x_1), F(x_2), \ldots, F(x_k)$ until you find a value x_i such that $F(x_i) \geqslant U$.

* This can be quite time consuming.

* A better method is to first find a central value of the distribution, such as the median.

* Then if $U > 0.5$ start at the median and work upwards until you find a value x_i such that $F(x_i) \geqslant U$.

* If $U < 0.5$ then start at the median and work downwards until you find a value x_{i-1} such that $F(x_{i-1}) < U$ and return x_i.

Other Methods

* For many continuous distributions, the cdf does not exist in closed form. Hence it is often not possible to apply the inverse method.
* For many common distributions special methods have been proposed based on transformations of random variables.

Theorem 2 (Box–Muller Algorithm)
Suppose that U_1 *and* U_2 *are two independent Uniform*$(0,1)$ *random variables and define*

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

Then Y_1 *and* Y_2 *are independent standard normal random variables.*

Chi-Squared Random variables

* If we can generate standard normal random variables then we can generate chi-squared random variables using the following theorems.

Theorem 3

Suppose that $Z \sim$ *Normal*$(0, 1)$ *then* $Z^2 \sim \chi^2_1$.

Theorem 4

If $X_1 \sim \chi^2_p$ *and* $X_2 \sim \chi^2_q$ *and* X_1 *and* X_2 *are independent then* $X_1 + X_2 \sim \chi^2_{p+q}$

Student's t and Snedecor's F Distributions

Theorem 5

Suppose that Z is a standard normal random variable and $X \sim \chi^2_p$ and that Z and X are independent then

$$T = \frac{Z}{\sqrt{X/p}} \sim t_p$$

Theorem 6

Suppose that $X \sim \chi^2_p$, $Y \sim \chi^2_q$ and X and Y are independent then

$$F = \frac{X/p}{Y/q} \sim F_{p,q}$$

Random Variate Generation in R

* In R there are inbuilt functions to generate observations from most "named" distribution

* The random seed is stored in a special variable called `.Random.seed` whose exact form depends on the uniform random number generator being used.

* Generally you should not access this variable directly but use the function `set.seed` to set the seed. This function takes a single integer argument.

* If a random number generating function is called and `.Random.seed` does not exist it is created using the current system time.

* Each time a random observation is created the value of `.Random.seed` is updated.

Random Number Generating Functions in R

Distribution	Function	Parameter(s)
Beta	`rbeta`	`shape1` (α), `shape2` (β)
Binomial	`rbinom`	`size` (n), `prob` (θ)
Cauchy	`rcauchy`	`location`, `scale`
Chi-squared	`rchisq`	`df` (degrees of freedom)
Exponential	`rexp`	`rate` (λ)
F	`rf`	`df1` (numerator df), `df2` (denominator df)
Gamma	`rgamma`	`shape` (α), `rate` (λ)
Geometric	`rgeom`	`prob` (θ)
Normal	`rnorm`	`mean` (μ), `sd` (σ)
Poisson	`rpois`	`lambda` (λ)
Student's t	`rt`	`df` (degrees of freedom)
Uniform	`runif`	`min`, `max`

Finite Mixture Distributions

Definition 5

Suppose that f_1, f_2, ..., f_k are probability distributions and that $p_1, \ldots, p_k$ are positive numbers such that $\sum p_j = 1$. Then

$$f(x) = \sum_{j=1}^{k} p_k f_k(x)$$

is a valid probability distribution and is called a finite mixture distribution

* Suppose we can generate from observations from all of the f_1, f_2, ..., f_k.
* To generate from the mixture distribution we first generate the latent variable z with discrete pmf given by

$$\mathrm{P}(Z = j) = p_j \qquad j = 1, \ldots, k.$$

Having generated z we then generate $X \sim f_z(x)$.

Accept-Reject Methods

* General technique which can be used for most continuous distributions.

* Let f be the pdf of interest (called the target density) and suppose that f is difficult to sample from.

* Instead of sampling from f, we shall sample from another density g (called the candidate density) which is easier to sample from.

* We then decide whether or not to accept the observation sampled from g as coming from f or to reject it and start over.

* How we make this decision will ensure that the resulting set of accepted observations do form a random sample from f.

Motivating Example

* Suppose that f is the uniform distribution on the unit circle

$$f(x,y) = \begin{cases} \frac{1}{2\pi} & 0 < x^2 + y^2 < 1 \\ 0 & \text{otherwise} \end{cases}$$

* It is not trivial to sample from f but it is easy to sample from

$$g(x,y) = \begin{cases} \frac{1}{4} & -1 < x < 1, \ -1 < y < 1 \\ 0 & \text{otherwise} \end{cases}$$

* If we generate pairs $(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$ from g and accept only those points with $x_i^2 + y_i^2 \leqslant 1$ we will generate a sample from f.

* This is a special case of the accept-reject algorithm.

The Candidate Density

* The candidate density g must be easy to sample from!

* The supports of f must be the same or a subset of the support of g ($f(x) > 0 \;\Rightarrow\; g(x) > 0$).

* The ratio f/g must be bounded. That is there must exist a constant $M < \infty$ such that

$$\frac{f(x)}{g(x)} \leqslant M \qquad \text{for every } x \text{ with } g(x) > 0.$$

The Accept-Reject Algorithm

1. Select a candidate density g satisfying the previous conditions and calculate the bounding constant M.

2. Generate a candidate random variate $Y \sim g$.

3. Independently of Y generate $U \sim \text{uniform}(0, 1)$.

4. If

$$U \leqslant \frac{1}{M} \frac{f(Y)}{g(Y)}$$

then stop and return $X = Y$, otherwise discard Y and U and repeat from step 2.

Theorem 7

The observation returned from the accept-reject algorithm described above has probability density function f.

Normalizing Constants

* In many situations we do not know the correct normalizing constant for the target distribution f but only know that

$$f(x) \;=\; K f_1(x) \qquad \text{for every } x.$$

* In that case we cannot calculate M so it seems that the algorithm will fail.

* Suppose, however, that we can calculate M_1 such that

$$\frac{f_1(x)}{g(x)} \;\leqslant\; M_1 \qquad \text{for every } x \text{ with } g(x) > 0.$$

* Then the algorithm will still work if we use the acceptance rule

$$U \;\leqslant\; \frac{1}{M_1}\,\frac{f_1(Y)}{g(Y)}$$

Some Properties of the Accept-Reject Method

* The bound M need not be a tight bound, we can use any constant M such that

$$M \geqslant \sup_{x:g(x)>0} \frac{f(x)}{g(x)}.$$

* The probability of acceptance, however, is $1/M$ so using a larger M results in a drop in efficiency.

* Different candidate densities g result in different values of M and so we may need to balance ease of simulation from g with efficiency of the algorithm.

* In the case when the target is known only up to a normalizing constant K, the acceptance probability is equal to $1/(KM_1)$ and so is not known in advance.