Generating Random Variables and Stochastic Processes

1 Generating U(0,1) Random Variables

The ability to generate $U(0,1)$ random variables is of fundamental importance since they are usually the building block for generating other random variables. We will look at:

- Properties that a random number generator should possess
- Linear Congruential Generators
- Using Matlab to generate $U(0,1)$ variates

Some History

- Earliest methods were manual
  - throwing dice, dealing cards, drawing balls from an urn
- Mechanized devices in early 1900's
- Later, methods based on electric circuits
- Many other schemes based on phone books, π etc.
- Advent of computing led to interest in numerical and arithmetic methods
  - such methods generate numbers sequentially and are easily programmed
  - the most commonly used methods today

Desirable Properties of a U(0,1) Generator

- Numbers should appear to be $\sim U(0,1)$ and independent
- Generator should be fast and not require too much storage
- Should be able to reproduce a given set of numbers
  - important for comparison purposes

1.1 Linear Congruential Generators

A linear congruential generator has the property that numbers are generated according to

$$Z_i = (aZ_{i-1} + c) \bmod m$$

where m, a, c and Z_0 are non-negative integers. We say that m is the *modulus* and that Z_0 is the *seed*. The sequence that we obtain clearly satisfies $0 \leq Z_i < m$. In order to generate *pseudo-random numbers*, $U_1, \ldots, U_n, \ldots$, we set $U_i = Z_i/m$. Note that $U_i \in (0,1)$ for each i.

We now **act** as though the U_i's constitute an independent sequence of $U(0,1)$ random variables.

Example 1 (Law and Kelton)

$Z_0 = 1, \quad Z_i = (11\ Z_{i-1}) \bmod 16$

Now iterate to determine the Z_i's

$$\begin{aligned} Z_0 &= 1 \\ Z_1 &= (11) \bmod 16 = 11 \\ Z_2 &= (121) \bmod 16 = 9 \\ Z_3 &= (99) \bmod 16 = 3 \\ Z_4 &= (33) \bmod 16 = 1 \\ Z_5 &= \cdots \\ Z_6 &= \cdots \end{aligned}$$

What is wrong with this? ∎

Possible Objections

1. The Z_i's are notrandom?
2. They can only take on a finite number of values
3. The **period** of the generator can be very poor: see previous example

Responses

1. As long as the Z_i's **appear** to be random, it's ok
2. Choose m very large
3. See theorem below

How to Guarantee a Full Period

Theorem 1 *The linear congruential generator has full period if and only if the following* 3 *conditions hold:*

1. *If* 4 *divides* m*, then* 4 *divides* $a - 1$
2. *The only positive integer that exactly divides both* m *and* c *is* 1*, i.e.,* m *and* c *are relatively prime*
3. *If* q *is a prime number that divides* m*, then it divides* $a - 1$

Example 2 (Law and Kelton)

$Z_0 = 1, \quad Z_i = (13Z_{i-1} + 13) \bmod 16$

Now iterate to determine the Z_i's

$$\begin{aligned} Z_0 &= 1 \\ Z_1 &= (26) \bmod 16 = 10 \\ Z_2 &= (143) \bmod 16 = 15 \\ Z_3 &= (248) \bmod 16 = 0 \\ Z_4 &= (13) \bmod 16 = 13 \\ Z_5 &= \cdots \end{aligned}$$

Check to see that this LCG has full period:

- Are the conditions of the theorem satisfied?
- Does it matter what integer we use for Z_0?

Testing Random Number Generators

- Some simple checks are to examine mean and variance
- Tests for $U(0,1)$ distribution
  - Kolmogorov Smirnov test
  - χ_2 test
- Tests for independence
  - serial tests
  - runs tests
  - autocorrelation tests

Constructing and testing good random number generators is very important! However, we will not study these issues in this course. Instead, we will assume that we have a good random number generator available to us and we will use it as a black box. See Law and Kelton, Chapter 7, for an excellent treatment and further details.

Generating $U(0,1)$ Variates in Matlab

```
> x = rand(10,1);
% generate a vector of 10 U(0,1) random variables

> m = mean(x);
% compute the mean of x

> var = std(x)^2
% compute the variance of x
```

2 Monte Carlo Integration

2.1 One-Dimensional Monte Carlo Integration

Suppose we want to compute

$$\theta = \int_0^1 g(x)\, dx.$$

If we cannot compute θ analytically, then we could use numerical methods. However, we can also use simulation and this can be especially useful for high-dimensional integrals. The key observation is to note that

$$\theta = \mathsf{E}[g(U)]$$

where $U \sim U(0,1)$. How do we use this observation?

1. Generate $U_1, U_2, \ldots U_n \sim U(0,1)$ and independent
2. Estimate θ with

$$\widehat{\theta}_n := \frac{g(U_1) + \ldots + g(U_n)}{n}$$

There are two reasons why $\widehat{\theta}_n$ is a good estimator

1. It is *unbiased*, i.e., $\mathsf{E}[\widehat{\theta}_n] = \theta$ and
2. It is *consistent*, i.e., $\widehat{\theta}_n \to \theta$ as $n \to \infty$ with probability 1

   - follows from the Strong Law of Large Numbers

Proof of Consistency

- $U_1, U_2, \ldots U_n$ are IID $U(0,1)$
- So $g(U_1), g(U_2), \ldots, g(U_n)$ are IID with mean θ
- Then by the Strong Law of Large Numbers

$$\lim_{n\to\infty} \frac{g(U_1) + \ldots + g(U_n)}{n} \to \theta \qquad \text{with probability } 1$$

Example 3 (Computing a 1-Dimensional Integral)

Suppose we wish to estimate $\int_0^1 x^3 \, dx$ using simulation:

- Know the exact answer is $1/4$
- Using simulation
  - generate n $U(0,1)$ independent variables
  - cube them
  - take the average

Sample Matlab Code

```
> n=100;
> x = rand(n,1);
> g = x.^3;
> estimate = mean(g)
% or more economically
> estimate = 2*mean(rand(100,1).^3)
```

Example 4

Suppose we wish to estimate $\theta = \int_1^3 (x^2 + x) \, dx$ using simulation:

- Know the exact answer is 12.67
- Using simulation
  - note that

$$\theta \;=\; 2\int_1^3 \frac{x^2 + x}{2} \, dx \;=\; 2\mathsf{E}[X^2 + X]$$

    where $X \sim U(1,3)$
  - so generate n $U(0,1)$ independent variables

– convert them to $U(1,3)$ variables; how?

– estimate θ

Sample Matlab Code

```
> n=100000;
> x=2*rand(n,1)+1;
> y=x.^2 + x;
> estimate = mean(y)
```

We could also have used a change of variables to convert the integral to an integral of the form $\int_0^1 \ldots dx$ and then estimated θ as in Example 3. ∎

2.2 Multi-Dimensional Monte Carlo Integration

Suppose now that we wish to approximate

$$\theta = \int_0^1 \int_0^1 g(x_1, x_2)\ dx_1 dx_2.$$

Then we can write

$$\theta = \mathsf{E}[g(U_1, U_2)]$$

where U_1, U_2 are IID $U(0,1)$ random variables.

Note that the joint PDF satisfies $f_{u_1,u_2}(u_1,u_2) = f_{u_1}(u_1) f_{u_2}(u_2) = 1$ on $(0,1)^2$.

As before we can estimate θ using simulation:

- Generate $2n$ independent $U(0,1)$ variables
- Compute $g(U_1^i, U_2^i)$ for $i = 1, \ldots, n$
- Estimate θ with

$$\widehat{\theta}_n = \frac{g(U_1^1, U_2^1) + \ \ldots\ + g(U_1^n, U_2^n)}{n}$$

As before, the Strong Law of Large Numbers justifies this approach.

Example 5 (Computing a Multi-Dimensional Integral)

We can estimate

$$\theta := \int_0^1 \int_0^1 (4x^2 y + y^2)\ dxdy.$$

using simulation though of course the true value of θ is easily calculated to be 1.

Sample Matlab Code

```
> x=rand(1,n);
> y=rand(1,n);
> g=4*(x.^2).*y + y.^2;
> ans=mean(g)
```

∎

We can also apply Monte Carlo integration to more general problems. For example, if we want to estimate

$$\theta = \int \int_A g(x,y) f(x,y)\ dx\ dy$$

where $f(x,y)$ is a density function on A, then we observe that

$$\theta = \mathsf{E}[g(X,Y)]$$

where X, Y have joint density $f(x,y)$. To estimate θ using simulation we

- Generate n random vectors (X,Y) with joint density $f(x,y)$
- Estimate θ with
$$\widehat{\theta}_n = \frac{g(X_1,Y_1) + \ \ldots \ + g(X_n,Y_n)}{n}$$

In the next part of this lecture, we will begin learning how to generate random variables that are not uniformly distributed.

---

3 Generating Univariate Random Variables

We will study a number of methods for generating univariate random variables. The three principal methods are the *inverse transform* method, the *composition* method and the *acceptance-rejection* method. All of these methods rely on having a $U(0,1)$ random number generator available, and for the duration of the course we will assume this to be the case.

3.1 The Inverse Transform Method for Discrete Random Variables

Suppose X is a discrete random variable with probability mass function (PMF)

$$X = \begin{cases} x_1, & \text{wp} \quad p_1 \\ x_2, & \text{wp} \quad p_2 \\ x_3, & \text{wp} \quad p_3 \end{cases}$$

where $p_1 + p_2 + p_3 = 1$. We would like to generate a value of X and we can do this by using our $U(0,1)$ generator:

1. Generate U
2. Set
$$X = \begin{cases} x_1, & \text{if} \quad 0 \le U \le p_1 \\ x_2, & \text{if} \quad p_1 < U \le p_1 + p_2 \\ x_3, & \text{if} \quad p_1 + p_2 < U \le 1 \end{cases}$$

We can check that this is correct as follows.

$$\mathbf{P}(X = x_1) = \mathbf{P}(0 \le U \le p_1) = p_1$$

since U is $U(0,1)$. The same is true for $\mathbf{P}(X = x_2)$ and $\mathbf{P}(X = x_3)$.

More generally, suppose X can take on n distinct values, $x_1 < x_2 < \ldots < x_n$, with

$$\mathbf{P}(X = x_i) = p_i \quad \text{for} \quad i = 1, \ldots, n.$$

Then to generate a sample value of X we

1. Generate U
2. Set $X = x_j$ if $\sum_{i=1}^{j-1} p_i < U \le \sum_{i=1}^{j} p_i$

   That is, set $X = x_j$ if $\mathsf{F}(x_{j-1}) < U \le \mathsf{F}(x_j)$

If n is large, then we might want to search for x_j more efficiently.

Example 6 (Generating a Geometric Random Variable)

Suppose X is geometric with parameter p so that $\mathbf{P}(X = n) = (1-p)^{n-1}p$. Then we can generate X as follows.

1. Generate U
2. Set $X = j$ if $\sum_{i=1}^{j-1}(1-p)^{i-1}p < U \leq \sum_{i=1}^{j}(1-p)^{i-1}p$

   That is, set $X = j$ if $1-(1-p)^{j-1} < U \leq 1-(1-p)^{j}$

   In particular, we set $X = \mathsf{Int}\left(\frac{\log(U)}{\log(1-p)}\right) + 1$ where $\mathsf{Int}(y)$ denotes the integer part of y.

You should convince yourself that this is correct! How does this compare to the coin-tossing method for generating X?

Example 7 (Generating a Poisson Random Variable)

Suppose that X is Poisson(λ) so that $\mathbf{P}(X = n) = \exp(-\lambda)\ \lambda^n/n!$. We can generate X as follows.

1. Generate U
2. Set
$$X = j \text{ if } \mathsf{F}(j-1) < U \leq \mathsf{F}(j)$$

Some questions arise:

- How do we find j? We could use the following algorithm.

```
set j = 0, p = e^{-λ}, F = p
while U > F
        set p = λp/(j+1), F = F + p, j = j + 1
set X = j
```

- How much work does this take
- What if λ is large?
  - can we find j more efficiently?
  - yes: check if j is close to λ first.
  - why might this be useful?
  - how much work does this take? (See Ross)

3.2 The Inverse Transform Method for Continuous Random Variables

Suppose now that X is a continuous random variable and we want to generate a value of X. Recall that when X was discrete, we could generate a value as follows:

- Generate U
- Then set $X = x_j$ if $\mathsf{F}(x_{j-1}) < U \leq \mathsf{F}(x_j)$

This suggests that when X is continuous, we might generate X as follows.

Inverse Transform Algorithm for Continuous Random Variables:

1. Generate U
2. Set $X = x$ if $\mathsf{F}_x(x) = U$, i.e., set $X = \mathsf{F}_x^{-1}(U)$

We need to prove that this actually works!

Proof: We have

$$\begin{aligned} \mathbf{P}(X \leq x) &= \mathbf{P}(\mathsf{F}_x^{-1}(U) \leq x) \\ &= \mathbf{P}(U \leq \mathsf{F}_x(x)) \\ &= \mathsf{F}_x(x) \end{aligned}$$

■

This assumes F_x^{-1} exists but even when F_x^{-1} does not exist we're still ok. All we have to do is

1. Generate U
2. Set
$$X = \min\{x : \mathsf{F}_x(x) \geq U\}$$

This works for discrete and continuous random variables or mixtures of the two.

Example 8 (Generating an Exponential Random Variable)

Problem: Generate $X \sim \mathsf{Exp}(\lambda)$

Solution: $\mathsf{F}_x(X) = 1 - e^{-\lambda x}$

Compute $F_x^{-1}(u)$: $u = 1 - e^{-\lambda x}$ so $x = -\frac{1}{\lambda}\log(1-u)$

Can use $x = -\log(u)/\lambda$. Why?

▮

Example 9 (Generating a Gamma(n,λ) Random Variable)

Problem: Generate $X \sim \mathsf{Gamma}(n, \lambda)$

Solution: Suppose n a positive integer

Let X_i be IID $\sim$ exp(λ) for $i = 1, \ldots, n$

So if $Y := X_1 + \ldots + X_n$ then $Y \sim$ Gamma(n, λ) !

So how can we generate a sample value of Y?

If n not an integer, need another method to generate Y

Example 10 (Order Statistics)

Order statistics are very important and have many applications.

- Suppose X has CDF F_x
- Let $X_1, \ldots, X_n$ be IID $\sim X$
- Let $X_{(1)}, \ldots, X_{(n)}$ be the *ordered* sample
  - so that $X_{(1)} \leq X_{(2)} \leq \ldots \leq X_{(n)}$
  - say $X_{(i)}$ is the i^{th} *ordered statistic*

Question: How do we generate a sample of $X_{(i)}$?

Method 1:

- Generate $U_1, \ldots, U_n$
- Compute $X_1 = \mathsf{F}_X^{-1}(U_1), \ldots, \mathsf{F}_X^{-1}(U_n)$
- Order the X_i's and take the i^{th} smallest
- How much work does this take?

Question: Can we do better?

Method 2:

- Sure, use the monotonicity of F!

Question: Can we do even better?

Method 3:

- Say $Z \sim$ beta(a, b) on $(0, 1)$ if

$$f(z) = cz^{a-1}(1-z)^{a-1} \quad \text{for } 0 \leq z \leq 1$$

  where c is a constant

- How can we use this?

Question: Can we do even better?

Advantages of Inverse Transform Method

- Monotonicity
  - we have already seen how this can be useful
- Variance reduction techniques
  - inducing correlation
  - '1 to 1', i.e. one $U(0,1)$ variable produces one X variable

Disadvantages of Inverse Transform Method

- F_x^{-1} may not always be computable
  - e.g. if $X \sim \mathsf{N}(0,1)$ then
$$\mathsf{F}_x(x) = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} \exp\left(\frac{-z^2}{2}\right)\, dz$$
  - here we cannot even express F_x in closed form
- Even if F_x available in closed form, may not be possible to find F_x^{-1} in closed form.
  - e.g. $F_x(x) = x^5(1+x)^3/8$ for $0 \le x \le 1$

One possible solution to these problems is to find F_x^{-1} numerically.

3.3 The Composition Approach

Another method for generating random variables is the composition approach. Suppose again that X has CDF F_x and we wish to simulate a value of X.

- Can often write
$$F_x(x) = \sum_{j=1}^{\infty} p_j \mathsf{F}_j(x)$$
where the F_j's are also CDFs, $p_j \ge 0$ for all j, and $\sum p_j = 1$
- Equivalently, if the densities exist then we can write
$$f_x(x) = \sum_{j=1}^{\infty} p_j \mathsf{f}_j(x)$$
- Such a representation often occurs very naturally
  - e.g. $X \sim$ Hyperexponential$(\lambda_1, \alpha_1, \ldots, \lambda_n, \alpha_n)$
$$f_x(x) = \sum_{j=1}^{n} \alpha_i \lambda_i e^{-\lambda_i x}$$
where $\lambda_i,\ \alpha_i \ge 0$, and $\sum_i^n \alpha_i = 1$. Here $\alpha_i = 0$ for $i > n$

Suppose now that it's difficult to simulate X directly using the inverse transform method. Then we could use the composition method instead.

Composition Algorithm:

1. Generate I that is distributed on the non-negative integers so that
$$\mathbf{P}(I = j) = p_j$$
How do we do this?
2. If $I = j$, then simulate Y_j from F_j
3. Set $X = Y_j$

We claim that X has the desired distribution!

Proof:

$$\begin{aligned}
\mathbf{P}(X \leq x) &= \sum_{j=1}^{\infty} \mathbf{P}(X \leq x | I = j)\, \mathbf{P}(I = j) \\
&= \sum_{j=1}^{\infty} \mathbf{P}(Y_j \leq x)\, \mathbf{P}(I = j) \\
&= \sum_{j=1}^{\infty} \mathsf{F}_j(x) p_j \\
&= \mathsf{F}_x(x)
\end{aligned}$$

■

The proof actually suggests that the composition approach might arise naturally from 'sequential' type experiments. Consider the following example.

Example 11 (A Sequential Experiment)

Suppose we roll a dice and let $Y \in \{1, 2, 3, 4, 5, 6\}$ be the outcome. If If $Y = i$ then we generate Z_i from the distribution F_i and set $X = Z_i$.

What is the distribution of X? How do we simulate a value of X? ▮

Example 12 (The Hyperexponential Distribution)

Let $X \sim$ Hyperexponential$(\lambda_1, \alpha_1, \lambda_2, \alpha_2)$ so that

$$f_x(x) = \alpha_1 \lambda_1 e^{-\lambda_1 x} + \alpha_2 \lambda_2 e^{-\lambda_2 x}.$$

In our earlier notation we have

$$\begin{aligned}
\alpha_1 &= p_1 \\
\alpha_2 &= p_2 \\
f_1(x) &= \lambda_1 e^{-\lambda_1 x} \\
f_2(x) &= \lambda_2 e^{-\lambda_2 x}
\end{aligned}$$

and the following algorithm will then generate a sample of X.

generate U_1
if $U_1 \leq p_1$ **then**
    set $i = 1$
else
    set $i = 2$
generate U_2
/* Now generate X from Exp(λ_i) */
set
$$X = -\frac{1}{\lambda_i}\log(U_2)$$

Question: How would you simulate a value of X if $\mathsf{F}_x(x) = (x + x^3 + x^5)/3$?

When the decomposition

$$\mathsf{F}_x = \sum_{j=1}^{\infty} p_j \mathsf{F}_j(x)$$

is not obvious, we can create an *artificial* decomposition by *splitting*.

Example 13 (Splitting)

Suppose

$$f_x(x) = \frac{1}{5}\ 1_{[-1,0]}(x) + \frac{6}{15}\ 1_{[0,2]}(x).$$

How do we simulate a value of X using vertical splitting?

How would horizontal splitting work?

3.4 The Acceptance-Rejection Algorithm

Let X be a random variable with density, $f(\cdot)$, and CDF, $F_x(\cdot)$. Suppose it's hard to simulate a value of X directly using either the inverse transform or composition algorithm. We might then wish to use the acceptance-rejection algorithm.

Let Y be another random variable with density $g(\cdot)$ and suppose that it is easy to simulate a value of Y. If there exists a constant a such that

$$\frac{f(x)}{g(x)} \leq a \text{ for all } x$$

then we can simulate a value of X as follows.

The Acceptance-Rejection Algorithm

generate Y with PDF $g(\cdot)$
generate U
while $U > \frac{f(Y)}{ag(Y)}$
    generate Y
    generate U
set $X = Y$

Question: Why must $a \geq 1$?

We must now check that this algorithm does indeed work. We define B to be the event that Y has been accepted in a loop, i.e., $U \leq f(Y)/ag(Y)$. We need to show that $\mathbf{P}(X \leq x) = \mathsf{F}_x(x)$

Proof: First observe

$$\begin{aligned} \mathbf{P}(X \leq x) &= \mathbf{P}(Y \leq x | B) \\ &= \frac{\mathbf{P}\left((Y \leq x) \cap B\right)}{\mathbf{P}(B)}. \end{aligned} \tag{1}$$

Then the denominator in (1) is given by

$$\begin{aligned} \mathbf{P}(B) &= \mathbf{P}\left(U \leq \frac{f(Y)}{ag(Y)}\right) \\ &= \frac{1}{a} \end{aligned}$$

while the numerator in (1) satisfies

$$\begin{aligned} \mathbf{P}\left((Y \leq x) \cap B\right) &= \int_{-\infty}^{\infty} \mathbf{P}\left((Y \leq x) \cap B \mid Y = y\right) g(y)\, dy \\ &= \int_{-\infty}^{\infty} \mathbf{P}\left((Y \leq x) \cap \left(U \leq \frac{f(Y)}{ag(Y)}\right) \;\middle|\; Y = y\right) g(y)\, dy \\ &= \int_{-\infty}^{x} \mathbf{P}\left(U \leq \frac{f(y)}{ag(y)}\right) g(y)\, dy \quad \text{(why?)} \\ &= \frac{F_x(x)}{a} \end{aligned}$$

Therefore $\mathbf{P}(X \leq x) = F_x(x)$, as required. ■

Example 14 (Generating a Beta(a, b) Random Variable)

Recall that X has a Beta(a, b) distribution if $f(x) = cx^{a-1}(1-x)^{b-1}$ for $0 \leq x \leq 1$.

Suppose now that we wish to simulate from the Beta$(4, 3)$ so that

$$f(x) = 60x^3(1-x)^2 \quad \text{for } 0 \leq x \leq 1.$$

We could, for example, integrate $f(\cdot)$ to find $\mathsf{F}(\cdot)$, and then try to use the inverse transform approach. However, it is hard to find $\mathsf{F}^{-1}(\cdot)$. Instead, let's use the acceptance-rejection algorithm:

1. First choose $g(y)$: let's take $g(y) = 1$ for $y \in [0, 1]$, i.e., $Y \sim U(0, 1)$
2. Then find a. Recall that we must have
$$\frac{f(x)}{g(x)} \leq a \text{ for all } x,$$
which implies
$$60x^3(1-x)^2 \leq a \text{ for all } x \in [0, 1].$$
So take $a = 3$. It is easy to check that this value works. We then have the following algorithm.

Algorithm

generate $Y \sim U(0,1)$
generate $U \sim U(0,1)$
while $U > 20Y^3(1-Y)^2$
    generate Y
    generate U
set $X = Y$

Efficiency of the Acceptance-Rejection Algorithm

Let N be the number of loops in the A-R algorithm until acceptance, and as before, let B be the event that Y has been accepted in a loop, i.e. $U \leq \frac{f(Y)}{ag(Y)}$. We saw earlier that $\mathbf{P}(B) = 1/a$.

Questions:

1: What is the distribution of N?

2: What is $\mathrm{E}[N]$?

How Do We Choose a ?

$\mathrm{E}[N] = a$, so clearly we would like a to be as small as possible. Usually, this is just a matter of calculus.

Example 15 (Generating a Beta(a,b) Random Variable continued)

Recall the Beta$(4,3)$ example with PDF $f(x) = 60x^3(1-x)^2 \quad$ for $x \in [0,1]$.

We chose $g(y) = 1$ for $y \in [0,1]$ so that $Y \sim U(0,1)$. The constant a had to satisfy

$$\frac{f(x)}{g(x)} \leq a \quad \text{for all } x \in [0,1]$$

and we chose $a = 3$.

We can do better by choosing

$$a = \max_{x \in [0,1]} \frac{f(x)}{g(x)} \approx 2.073.$$

How Do We Choose $g(\cdot)$?

- Would like to choose $g(\cdot)$ to minimize the computational load
  - can do this by taking $g(\cdot)$ 'close' to $f(\cdot)$
  - then a close to 1 and fewer iterations required
- But there is a tradeoff
  - if $g(\cdot)$ 'close' to $f(\cdot)$ then will probably also be hard to simulate from $g(\cdot)$
- So often need to find a balance between having a 'nice' $g(\cdot)$ and a small a

3.5 Acceptance-Rejection Algorithm for Discrete Random Variables

So far, we have expressed the A-R algorithm in terms of PDF's, thereby implicitly assuming that we are generating continuous random variables. However, the A-R algorithm also works for discrete random variables where we simply replace PDF's with PMF's.

So suppose we wish to simulate a discrete random variable, X, with PMF, $p_i = \mathbf{P}(X = x_i)$. If we do not wish to use the discrete inverse transform method for example, then we can use the following version of the A-R algorithm. We assume that we can generate Y with PMF, $q_i = \mathbf{P}(Y = y_i)$, and that a satisfies $p_i/q_i \leq a$ for all i.

The Acceptance-Rejection Algorithm for Discrete Random Variables

generate Y with PMF q_i
generate U
while $U > \frac{p_Y}{aq_Y}$
    generate Y
    generate U
set $X = Y$

Generally, we would use this A-R algorithm when we can simulate Y efficiently.

Exercise 1 *(Ross Q4.13)*

Suppose $Y \sim$ *Bin*(n, p) *and that we want to generate* X *where*

$$\mathbf{P}(X = r) = \mathbf{P}(Y = r | Y \geq k)$$

for some fixed $k \leq n$. *Assume* $\alpha = \mathbf{P}(Y \geq k)$ *has been computed.*

1. *Give the inverse transform method for generating* X
2. *Give another method for generating* X
3. *For what values of* α, *small or large, would the algorithm in (2) be inefficient?*

Example 16 (Generating from a Uniform Distribution over a 2-D Region)

Suppose (X, Y) is uniformly distributed over a 2-dimensional area, A. How would you simulate a sample of (X, Y)?

- Note first that if $X \sim U(-1, 1)$, $Y \sim U(-1, 1)$ and X and Y are independent then (X, Y) is uniformly distributed over
$$A := \{(x, y) : -1 \leq x \leq 1, -1 \leq y \leq 1\}$$
  - how would you show this?
- So we can simulate a sample of (X, Y) when A is a square. How?
- Suppose now A is a circle of radius 1 centered at the origin
  - then how do we simulate a sample of (X, Y)?

∎

4 Other Methods for Generating Univariate Random Variables

Besides the inverse transform, composition and acceptance-rejection algorithms, there are a number of other important methods for generating random variables. We begin with the convolution method.

4.1 The Convolution Method

Suppose $X \sim Y_1 + Y_2 + \ldots + Y_n$ where the Y_i's are IID with CDF $F_y(\cdot)$. Suppose also that it's easy to generate the Y_i's. Then it is straightforward to generate a value of X:

1. Generate $\widetilde{Y}_1, \ldots, \widetilde{Y}_n$ that have CDF F_y
2. Set $X = \widetilde{Y}_1 + \ldots + \widetilde{Y}_n$

We briefly mentioned this earlier in Example 9 when we described how to generate a Gamma(λ, n) random variable. The convolution method is not always the most efficient method. Why?

4.2 Methods Using Special Properties

Suppose we want to simulate a value of random variable X, and we know that

$$X \sim g(Y_1, \ldots, Y_n)$$

for some random variables Y_i and some function $g(\cdot)$. (Note the Y_i's need not necessarily be IID.) If we know how to generate each of the Y_i's then we can generate X as follows:

1. Generate $\widetilde{Y}_1, \ldots, \widetilde{Y}_n$
2. Set $X = g(\widetilde{Y}_1, \ldots, \widetilde{Y}_n)$

Example 17 (Convolution)

The convolution method is a special case where $g(Y_1, \ldots, Y_n) = Y_1 + \ldots + Y_n$. ∎

Example 18 (Generating Lognormal Random Variables)

Suppose $X \sim \mathsf{N}(\mu, \sigma^2)$. Then $Y := \exp(X)$ has a lognormal distribution, i.e., $Y \sim \mathsf{LN}(\mu, \sigma^2)$. (Note $\mathrm{E}[Y] \neq \mu$ and $\mathrm{Var}(Y) \neq \sigma^2$.) How do we generate a log-normal random variable? ∎

Example 19 (Generating χ^2 Random Variables)

Suppose $X \sim \mathsf{N}(0, 1)$. Then $Y := X^2$ has a a chi-square distribution with 1 degree of freedom, i.e., $Y \sim \chi^2_1$.

Question: How would you generate a χ^2_1 random variable?

Suppose now that $X_i \sim \chi^2_1$ for $i = 1, \ldots, n$. Then $Y := X_1 + \ldots + X_n$ has a chi-square distribution with n degree of freedom, i.e., $Y \sim \chi^2_n$.

Question: How would you generate a χ^2_n random variable? ∎

Example 20 (Generating t_n Random Variables)

Suppose $X \sim \mathsf{N}(0,1)$ and $Y \sim \chi^2_n$ with X and Y independent. Then

$$Z := \frac{X}{\sqrt{\frac{Y}{n}}}$$

has a a t distribution with n degrees of freedom , i.e., $Z \sim t_n$.

Question: How would you generate a t_n random variable? ∎

Example 21 (Generating $F_{m,n}$ Random Variables)

Suppose $X \sim \chi^2_m$ and $Y \sim \chi^2_n$ with X and Y independent. Then

$$Z := \frac{\left(\frac{X}{m}\right)}{\left(\frac{Y}{n}\right)}$$

has an F distribution with m and n degrees of freedom, i.e., $Z \sim F_{m,n}$.

Question: How would you generate a $F_{m,n}$ random variable? ∎

Note: The proof of the statements in these examples can be found in many probability or statistics textbooks

5 Generating Normal Random Variables

We have not yet seen how to generate normal random variables though they are of course very important in practice. Though important in their own right, we need to be able to generate normal random variables so that we can then generate lognormal random variables. These variables are very important for financial engineering applications.

- Note that if $Z \sim \mathsf{N}(0,1)$ then
$$X := \mu + \sigma Z \sim \mathsf{N}(\mu, \sigma^2)$$
  - so we need only worry about generating $\mathsf{N}(0,1)$ random variables
  - why?
- One possible generation method is the inverse transform method
  - but we would have to use numerical methods since we cannot find $F_z^{-1}(\cdot) = \Phi^{-1}(\cdot)$ in closed form
  - so not very efficient
- Will therefore look at the following methods for generating $\mathsf{N}(0,1)$ random variables
  1. Box-Muller method
  2. Polar method
  3. Rational approximations
- There are many other methods, e.g., A-R algorithm

5.1 The Box Muller Algorithm

The Box-Muller algorithm uses two IID $U(0,1)$ random variables to produce two IID N$(0,1)$ random variables. It works as follows:

The Box-Muller Algorithm for Generating Two IID N$(0,1)$ Random Variables

generate U_1 and U_2 IID $U(0,1)$
set

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

We now show that this algorithm does indeed produce two IID N$(0,1)$ random variables, X and Y.

Proof: We need to show that

$$f(x,y) = \frac{1}{\sqrt{2\pi}}\exp\left(-\frac{x^2}{2}\right)\ \frac{1}{\sqrt{2\pi}}\exp\left(-\frac{y^2}{2}\right)$$

First, make a change of variables:

$$\begin{aligned} R &:= \sqrt{X^2+Y^2} \\ \theta &:= \tan^{-1}\left(\frac{Y}{X}\right) \end{aligned}$$

so R and θ are polar coordinates of (X,Y).

To transform back, note $X = R\cos(\theta)$ and $Y = R\sin(\theta)$. Note also that

$$\begin{aligned} R &= \sqrt{-2\log(U_1)} \\ \theta &= 2\pi U_2 \end{aligned}$$

Since U_1 and U_2 are IID, R and θ are independent. Clearly $\theta \sim U(0,2\pi)$ so $f_\theta(\theta) = 1/2\pi$ for $0 \le \theta \le 2\pi$.

It is also easy to see that $f_R(r) = re^{-r^2/2}$, so

$$f_{R,\theta}(r,\theta) = \frac{1}{2\pi}re^{-r^2/2}.$$

This implies

$$\begin{aligned} \mathbf{P}(X \le x_1, Y \le y_1) &= \mathbf{P}(R\cos(\theta) \le x_1, R\sin(\theta) \le y_1) \\ &= \int\int_A \frac{1}{2\pi}re^{-r^2/2}\ dr\ d\theta \end{aligned}$$

where $A = \{(r,\theta) : r\cos(\theta) \le x,\ r\sin(\theta) \le y\}$.

Now transform back to (x,y) coordinates:

$$x = r\cos(\theta) \quad \text{and} \quad y = r\sin(\theta)$$

and note that $dxdy = rdrd\theta$, i.e., the Jacobian of the transformation is r.

We then have

$$\mathbf{P}(X \le x_1, Y \le y_1) = \frac{1}{2\pi}\int_{-\infty}^{x_1}\int_{-\infty}^{y_1} exp\left(-\frac{(x^2+y^2)}{2}\right)\ dxdy$$

$$= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x_1} \exp(-x^2/2)\, dx \; \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{y_1} \exp(-y^2/2)\, dy$$

as required.

■

5.2 The Polar Method

One disadvantage of the Box-Muller method is that computing sines and cosines is inefficient. We can get around this problem using the polar method.

The Polar Algorithm for Generating Two IID $\mathbf{N}(0,1)$ Random Variables

generate U_1 and U_2 IID $U(0,1)$
set $V_1 = 2U_1 - 1$, $V_2 = 2U_2 - 1$ and $S = V_1^2 + V_2^2$
while $S > 1$
    generate U_1 and U_2 IID $U(0,1)$
    set $V_1 = 2U_1 - 1$, $V_2 = 2U_2 - 1$ and $S = V_1^2 + V_2^2$
set

$$X = \sqrt{\frac{-2\log(S)}{S}} V_1 \quad \text{and} \quad Y = \sqrt{\frac{-2\log(S)}{S}} V_2$$

Can you see why this algorithm[1] works?

5.3 Rational Approximations

Let $X \sim \mathsf{N}(0,1)$ and recall that $\Phi(x) = \mathbf{P}(X \leq x)$ is the CDF of X. If $U \sim U(0,1)$, then the inverse transform method seeks x_u such that

$$\Phi(x_u) = U \quad \text{i.e.} \quad x_u = \Phi^{-1}(U).$$

Finding Φ^{-1} in closed form is not possible but instead, we can use *rational approximations*. These are very accurate and efficient methods for estimating x_u.

Example 22 (Rational Approximations)

For $0.5 \leq u \leq 1$

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

where a_0, a_1, b_1 and b_2 are constants, and $t = \sqrt{-2\log(1-u)}$. The error is bounded in this case by .003. Even more accurate approximations are available, and since they are very fast, many packages (including Matlab) use them for generating normal random variables. ▮

[1] See Ross for further details.

6 Simulating Poisson Processes

Recall that a Poisson process, $N(t)$, with intensity λ is such that

$$\mathsf{P}\left(N(t) = r\right) = \frac{(\lambda t)^r e^{-\lambda t}}{r!}.$$

- the numbers of arrivals in non-overlapping intervals are independent
- and the distribution of the number of arrivals in an interval only depends on the length of the interval

It is good for modelling many phenomena including the emission of particles from a radioactive source, market crashes, and the arrivals of customers to a queue.

The i^{th} inter-arrival time, X_i, is defined to be the interval between the $(i-1)^{th}$ and i^{th} arrivals of the Poisson process, and it is easy to see that the X_i's are IID $\sim$ Exp(λ).

In particular, this means we can simulate a Poisson process with intensity λ by simply generating the inter-arrival times, X_i, where $X_i \sim \mathsf{Exp}(\lambda)$. We have the following algorithm for simulating the first T time units of a Poisson process:

Simulating T Time Units of a Poisson Process

set $t = 0$, $I = 0$
generate U
set $t = t - \log(U)/\lambda$
while $t < T$
    set $I = I + 1$, $S(I) = t$
    generate U
    set $t = t - \log(U)/\lambda$

6.1 The Non-Homogeneous Poisson Process

A non-homogeneous Poisson process, $N(t)$, is obtained by relaxing the assumption that the intensity, λ, is constant. Instead we take it to be a deterministic function of time, $\lambda(t)$. The non-homogeneous Poisson process can often be very important in practice. Consider, for example:

- a person who is measuring particle emissions from a radioactive source while moving closer to the source
- the occurrence of a market crash or a company bankruptcy as economic variables change

More formally, if $\lambda(t) \geq 0$ is the intensity of the process at time t, then we say that $N(t)$ is a non-homogeneous Poisson process with intensity $\lambda(t)$. Define the function $m(t)$ by

$$m(t) := \int_0^t \lambda(s)\, ds.$$

Then it can be shown that $N(t+s) - N(t)$ is a Poisson random variable with parameter $m(t+s) - m(t)$, i.e.,

$$\mathbf{P}\left(N(t+s) - N(t) = r\right) = \frac{\exp\left(-m_{t,s}\right)\left(m_{t,s}\right)^r}{r!}$$

where $m_{t,s} := m(t+s) - m(t)$.

Simulating a Non-Homogeneous Poisson Process

Before we describe the *thinning* algorithm for simulating a non-homogeneous Poisson process, we first need the following[2] proposition.

Proposition 2 *Let $N(t)$ be a Poisson process with constant intensity λ. Suppose that an arrival that occurs at time t is counted with probability $p(t)$, independently of what has happened beforehand. Then the process of counted arrivals is a non-homogeneous Poisson process with intensity $\lambda(t) = \lambda p(t)$.*

Suppose now $N(t)$ is a non-homogeneous Poisson process with intensity $\lambda(t)$ and that there exists a λ such that $\lambda(t) \leq \lambda$ for all $t \leq T$. Then we can use the following algorithm, based on Proposition 2, to simulate $N(t)$.

The Thinning Algorithm for Simulating T Time Units of a NHPP

set $t = 0$, $I = 0$
generate U_1
set $t = t - \log(U_1)/\lambda$
while $t < T$
    generate U_2
    if $U_2 \leq \lambda(t)/\lambda$ **then**
        set $I = I + 1$, $S(I) = t$
    generate U_1
    set $t = t - \log(U_1)/\lambda$

Questions

1) Can you give a more efficient version of the algorithm when there exists $\lambda > 0$ such that $\min_{0 \leq t \leq T} \lambda(t) \geq \lambda$?

2) Can you think of another algorithm for simulating a non-homogeneous Poisson process that is not based on thinning?

6.2 Credit Derivatives Models

Many credit derivatives models use *Cox* processes to model company defaults. A Cox process, $C(t)$, is similar to a non-homogeneous Poisson process except that the intensity function, $\lambda(t)$, is itself a stochastic process. However, conditional upon knowing $\lambda(t)$ for all $t \in [0, T]$, $C(t)$ becomes a non-homogeneous Poisson process. In credit derivatives models, bankruptcy of a company is often modelled as occurring on the first arrival in the Cox process where the intensity at time t, $\lambda(t)$, generally depends on the level of other variables in the economy. Such variables might include, for example, interest rates, credit ratings and stock prices, all of which are themselves random.

An understanding of and ability to simulate non-homogeneous Poisson processes is clearly necessary for analyzing such credit derivatives models.

[2] A proof may be found in Ross, for example.

7 Simulating Geometric Brownian Motions

Definition 1 *We say that a stochastic process, $\{X_t : t \geq 0\}$, is a Brownian motion with parameters (μ, σ) if*

1. *For $0 < t_1 < t_2 < \ldots < t_{n-1} < t_n$*

$$(X_{t_2} - X_{t_1}),\ (X_{t_3} - X_{t_2}), \ldots,\ \ (X_{t_n} - X_{t_{n-1}})$$

are mutually independent. (This is the independent increments property.)

2. *For $s > 0$, $X_{t+s} - X_t \sim N(\mu s, \sigma^2 s)$ and*

3. *X_t is a continuous function of t.*

Notation

- We say that X is a $B(\mu, \sigma)$ Brownian motion
  - μ is called the drift
  - σ is called the volatility
- When $\mu = 0$ and $\sigma = 1$ we have a *standard* Brownian motion (SBM)
  - we will use B_t to denote an SBM
  - *always* assume $B_0 = 0$
- Note that if $X \sim B(\mu, \sigma)$ and $X_0 = x$ then we can write

$$X_t = x + \mu t + \sigma B_t$$

where B is an SBM. We will usually write a $B(\mu, \sigma^2)$ Brownian motion in this way.

Remark 1 *Bachelier (1900) and Einstein (1905) were the first to explore Brownian motion from a mathematical viewpoint whereas Wiener (1920's) was the first to show that it actually exists as a well-defined mathematical entity.*

Questions

1) What is $\mathrm{E}[B_{t+s}B_s]$?

2) What is $\mathrm{E}[X_{t+s}X_s]$ where $X \sim B(\mu, \sigma)$?

3) Let B be an SBM and let $Z_t := |B_t|$. What is the CDF of Z_t for t fixed?

7.1 Simulating a Standard Brownian Motion

It is not possible to simulate an entire sample path of Brownian motion between 0 and T as this would require an infinite number of random variables. This is not always a problem, however, since we often only wish to simulate the value of Brownian motion at certain fixed points in time. For example, we may wish to simulate B_{t_i} for $t_1 < t_2 < \ldots < t_n$, as opposed to simulating B_t for every $t \in [0, T]$.

Sometimes, however, the quantity of interest, θ, that we are trying to estimate does indeed depend on the entire sample path of B_t in $[0, T]$. In this case, we can still estimate θ by again simulating B_{t_i} for $t_1 < t_2 < \ldots < t_n$ but where we now choose n to be very large. We might, for example, choose n so that $|t_{i+1} - t_i| < \epsilon$ for all i, where $\epsilon > 0$ is very small. By choosing ϵ to be sufficiently small, we hope to minimize the *numerical error* (as

opposed to the *statistical error*), in estimating θ. We will return to this topic at the end of the course when we learn how to simulate stochastic differential equations.

In either case, we need to be able to simulate B_{t_i} for $t_1 < t_2 < \ldots < t_n$ and for a fixed n. We will now see how to do this. The first observation we make is that

$$(B_{t_2} - B_{t_1}),\ (B_{t_3} - B_{t_2}), \ldots,\quad (B_{t_n} - B_{t_{n-1}})$$

are mutually independent, and for $s > 0$, $B_{t+s} - B_t \sim \mathsf{N}(0, s)$.

The idea then is as follows: we begin with $t_0 = 0$ and $B_{t_0} = 0$. We then generate B_{t_1} which we can do since $B_{t_1} \sim \mathsf{N}(0, t_1)$. We now generate B_{t_2} by first observing that $B_{t_2} = B_{t_1} + (B_{t_2} - B_{t_1})$. Then since $(B_{t_2} - B_{t_1})$ is independent of B_{t_1}, we can generate B_{t_2} by generating an $\mathsf{N}(0, t_2 - t_1)$ random variable and simply adding it to B_{t_1}. More generally, if we have already generated B_{t_i} then we can generate $B_{t_{i+1}}$ by generating an $\mathsf{N}(0, t_{i+1} - t_i)$ random variable and adding it to B_{t_i}. We have the following algorithm.

Simulating a Standard Brownian Motion

set $t_0 = 0$, $B_{t_0} = 0$
for $i = 1$ to n
    generate $X \sim \mathsf{N}(0, t_i - t_{i-1}))$
    set $B_{t_i} = B_{t_{i-1}} + X$

Remark 2 *When we generate $(B_{t_1}, B_{t_2}, \ldots, B_{t_n})$ we are actually generating a* random vector *that does* not *consist of IID random variables. In fact the method that we use to simulate $(B_{t_1}, B_{t_2}, \ldots, B_{t_n})$ is one of the most common methods for generating correlated random variables and stochastic processes. We will return to this later.*

Remark 3 *It is very important that when you generate $B_{t_{i+1}}$, you do so* conditional *on the value of B_{t_i}. If you generate B_{t_i} and $B_{t_{i+1}}$ independently of one another then you are effectively simulating from* different *sample paths of the Brownian motion. This is not correct!*

Simulating a $\mathbf{B}(\mu, \sigma)$ Brownian Motion

Suppose now that we want to simulate a $B(\mu, \sigma)$ BM, X, at the times $t_1, t_2, \ldots, t_{n-1}, t_n$. Then all we have to do is simulate an SBM, $(B_{t_1}, B_{t_2}, \ldots, B_{t_n})$, and use our earlier observation that $X_t = x + \mu t + \sigma B_t$.

Brownian Motion as a Model for Stock Prices?

There are a number of reasons why Brownian motion is **not** a good model for stock prices:

- Limited liability
- People care about *returns*, not absolute prices
    - so independent increments property should not hold for stock prices

7.2 Geometric Brownian Motion

Definition 2 *We say that a stochastic process,* $\{X_t : t \geq 0\}$*, is a* (μ, σ) *geometric Brownian motion (GBM) if* $\log(X) \sim B(\mu - \sigma^2/2,\ \sigma)$*. We write* $X \sim GBM(\mu, \sigma)$*.*

The following properties of GBM follow immediately from the definition of BM:

1. Fix $t_1, t_2, \ldots, t_n$. Then $\frac{X_{t_2}}{X_{t_1}},\ \frac{X_{t_3}}{X_{t_3}},\ \frac{X_{t_n}}{X_{t_{n-1}}}$ are mutually independent.
2. For $s > 0$, $\log\left(\frac{X_{t+s}}{X_t}\right) \sim \mathsf{N}\left((\mu - \sigma^2/2)s,\ \sigma^2 s\right)$.
3. X_t is continuous.

Again, we call μ the drift and σ the volatility. If $X \sim GBM(\mu, \sigma)$, then note that X_t has a lognormal distribution. In particular, if $X \sim \mathsf{GBM}(\mu, \sigma)$, then $X_t \sim \mathsf{LN}\left((\mu - \sigma^2/2)t,\ \sigma^2 t\right)$.

Question: How would you simulate a sample path of $GBM(\mu, \sigma^2)$ at the fixed times $0 < t_1 < t_2 < \ldots < t_n$?

Answer: Simulate $\log(X_{t_i})$ first and then take exponentials! (See below for more details.)

7.3 Modelling Stock Prices as Geometric Brownian Motion

Note the following:

1. If $X_t > 0$, then X_{t+s} is always positive for any $s > 0$. Why?
   - so limited liability is not violated
2. The distribution of $\frac{X_{t+s}}{X_t}$ only depends on s
   - so the distribution of *returns* from one period to the next is constant

This suggests that GBM might be a reasonable[3] model for stock prices. In fact, we will usually model stock prices as GBM's in this course, and we will generally use the following notation:

- S_0 is the known stock price at $t = 0$
- S_t is the random stock price at time t and

$$S_t = S_0 e^{(\mu - \sigma^2/2)t + \sigma B_t}$$

  where B is a standard BM. The drift is μ, σ is the volatility and S is a therefore a $GBM(\mu, \sigma)$ process that begins at S_0

Questions

1) What is $\mathrm{E}[S_t]$?

2) What is $\mathrm{E}[S_t^2]$?

2) Show $S_{t+s} = S_t e^{(\mu - \sigma^2/2)s + \sigma(B_{t+s} - B_t)}$.

7.4 Simulating a Geometric Brownian Motion

Suppose we wish to simulate $S \sim GBM(\mu, \sigma)$. Then as before,

$$S_t = S_0 e^{(\mu - \sigma^2/2)t + \sigma B_t},$$

so it is clear that we can simulate S by simply simulating B.

[3] Of course many other models are used in practice for studying various markets.