Generation of Non-Uniform Random Numbers

Refs: L&P, Chs. 6 and 7, L&K, Ch. 7, and Devroye, Non-Uniform Random Variate Generation (watch for typos)

Problem: Given a uniform random variable U, generate a random variable X having a prescribed distribution function \( F_X(\cdot) \).

We previously discussed the inversion method. While inversion is a very general method, it may be computationally expensive. In particular, computing \( F_X^{-1}(\cdot) \) may have to be implemented via a numerical root-finding method in many cases. Therefore, we will now describe other methods for non-uniform random number generation.

1. Acceptance-Rejection Method

We now discuss a method that is well-suited to generating random variates with an easily-calculated density. This method is called acceptance-rejection.

**Goal:** Generate a random variate \( X \) having given probability density \( f_X(x) \), where \( f_X(x) \) is positive only on the interval \([a, b]\) (where \(-\infty < a < b < \infty\)).

Enclose the density in a rectangle having base \((b-a)\) and height \( m \), where \( m = \sup_{a \leq x \leq b} f_X(x) \).

Suppose we throw down points uniformly in the rectangle \( R \) (denote these points by the symbol \( x \)). Throw away (or reject) the points above the density \( f_X(x) \).

**Claim:** The x-coordinate of each accepted point (denoted by the symbol \( \otimes \)) has density \( f_X(x) \).

**Proof:** Let \((Z_1, Z_2)\) be the \((x, y)\) coordinates of a random point distributed uniformly in \( R \). Then, for \( a \leq x \leq b \),

\[
P(X \leq x) = P(Z_1 \leq x \mid \text{acceptance}) = \frac{P(Z_1 \leq x, Z_2 \leq f_X(Z_1))}{P(Z_2 \leq f_X(Z_1))}.
\]
But \( P(Z_1 \leq x, Z_2 \leq f_x(Z_1)) \) is just the probability that \((Z_1,Z_2)\) falls into the shaded region below:

![Graph showing the shaded region](image)

This probability is just

\[
\frac{\int_a^b f_X(t) \, dt}{\text{area of } R}.
\]

Putting \( x = b \), we conclude that

\[
P(Z_2 \leq f_x(Z_1)) = \left(\text{area of } R\right)^{-1}.
\]

So,

\[
P(X \leq x) = \int_a^x f_X(t) \, dt.
\]

as desired. Hence, acceptance-rejection works. A more precise specification of the algorithm is:

1. Generate \( U_1, U_2 \) as two independent \( U(0,1) \) random variates.
2. Set \( Z_1 = a + (b - a)U_1 \) and \( Z_2 = mU_2 \) [We are essentially using the inversion method here]
3. If \( Z_2 \leq f_X(Z_1) \), return \( X = Z_1 \). Else, return to 1.

Note that the number of pairs \( N \) of uniform random variates \((U_1,U_2)\) required to generate \( X \) is geometrically distributed with mass function

\[
P(N = k) = p(1 - p)^{k-1} \quad \text{for } k \geq 1,
\]

where \( p = \left(\text{area of } R\right)^{-1} \). The mean number of pairs that would be required is then

\[
E(N) = \frac{1}{p} = (b-a)m.
\]

Hence, a good measure of how fast such an acceptance-rejection algorithm will be is the closeness of the quantity \((b-a)m\) to 1. (So we want \( m \) to be as small as possible.)

2. Generalized Acceptance-Rejection

There are many random variates having densities that are positive over infinite intervals, or such that \((b-a)m\) is very large. In such cases, the following generalization of acceptance-rejection may be useful.

**Goal:** Generate a random variate \( X \) having given probability density \( f_X(\cdot) \).

Page 2 of 8
We assume that there exists a random variable $Y$, having known density $g(\cdot)$, such that:

1. We have a means of efficiently generating replicates of $Y$ (e.g., $Y$ is exponential).
2. There exists a constant $c > 0$ such that $f_X(x) \leq c g(x)$ for all $x$.

A random variable $Y$ having the above properties is called a majorizing random variable and the density $g(\cdot)$ is called a majorizing density.

Let

$$c = \sup_x \frac{f_X(x)}{g(x)}$$

be the smallest such constant.

Generalized acceptance-rejection works as follows:

1. Generate the random variable $Y$ having density $g(\cdot)$.
2. Generate a uniform(0,1) random variable $U$, independent of $Y$.
3. If $Ucg(Y) \leq f_X(Y)$, return $X = Y$. Else, return to 1.

An argument similar to that used in the case of acceptance-rejection shows that the above algorithm produces variates with the required density $f_X(\cdot)$. In this case, we have, using the law of total expectation and the independence of $U$ and $Y$,

$$P(Y \leq x, \text{acceptance}) = P(Y \leq x, U \leq f_X(Y)/cg(Y)) = \int_{-\infty}^{x} P(U \leq f_X(Y)/cg(Y) | Y = y) g(y) \, dy$$

$$= \int_{-\infty}^{x} P(U \leq f_X(y)/cg(y)) g(y) \, dy = \int_{-\infty}^{x} (f_X(y)/cg(y)) g(y) \, dy = (1/c) \int_{-\infty}^{x} f_X(y) \, dy.$$

Taking $x = \infty$ in the above calculation yields $P(\text{acceptance}) = 1/c$, so that

$$P(X \leq x | \text{acceptance}) = \frac{P(Y \leq x, \text{acceptance})}{P(\text{acceptance})} = \frac{(1/c) \int_{-\infty}^{x} f_X(y)/c \, dy}{1/c} = \int_{-\infty}^{x} f_X(y) \, dy.$$
The following figure provides some intuition; it was obtained by scaling the previous figure by a factor of 1/c. The majorizing density curve plays the role of the bounding rectangle in the simple A/R method.

Note that the number N of random variates Y required to generate a single replicate of X is geometric with mean c. Hence, c is a measure of the efficiency of the algorithm—the closer c is to 1, the better.

3. Convolution Method

To generate a random variable X that can be represented as

\[ X = Y_1 + \cdots + Y_m, \]

where the \( Y_i \)'s are independent random variables, an obvious strategy is to generate the \( Y_i \)'s independently, and sum them to obtain X. This is known as the convolution method because the distribution function of X can be computed analytically as the convolution of the distributions of the \( Y_i \)'s.

**Example**: Suppose X is binomially distributed with parameters n and p (0 ≤ p ≤ 1). Then, it is well known that

\[ X \stackrel{\text{D}}{=} Y_1 + \cdots + Y_m, \]

where the \( Y_i \)'s are i.i.d. Bernoulli (p) random variables having mass function

\[ P(Y_i = 0) = 1 - p \quad \text{and} \quad P(Y_i = 1) = p. \]

If n isn't too large, the convolution method can be a reasonably efficient way of generating X. Otherwise, one can turn to more sophisticated algorithms. See Devroye.

4. Composition Method

Suppose that either the distribution \( F_X(\cdot) \) or the probability density \( f_X(\cdot) \) can be represented either of the following two forms:

\[ F_X(x) = p_1 F_{Y_1}(x) + p_2 F_{Y_2}(x) + \cdots + p_m F_{Y_m}(x) \]

\[ f_X(x) = \sum_{i=1}^{m} p_i f_{Y_i}(x) \]
\begin{align*}
(2) & \quad f_X(x) = p_1 f_{Y_1}(x) + p_2 f_{Y_2}(x) + \ldots + p_m f_{Y_m}(x)
\end{align*}

where \( p_1, \ldots, p_m \) are non-negative and sum to one (so that they form a probability mass function). Then, assuming that the \( Y_i \)'s are relatively easily to generate, we can generate \( X \) as follows:

1. Generate a discrete random variable \( I \) on \([1, \ldots, m]\), where \( P(I = j) = p_j \) for \( 1 \leq j \leq m \).
2. Generate \( Y_i \) from \( F_{Y_i} \) (or \( f_{Y_i} \)).
3. Return \( X = Y_i \).

This approach can sometimes lead to very fast algorithms for non-uniform variate generation. To see why, suppose we approximate the density \( f_X(\cdot) \) (of the random variable \( X \) to be generated) from below by a piecewise-linear function.

\begin{center}
\begin{tikzpicture}
\draw[->, thick] (-0.5,0) -- (2.5,0) node[right] {}; \draw[->, thick] (0,-1) -- (0,1) node[above] {}; \draw (0,0) -- (0,1) node[above] {\( f_X(x) \)};
\draw (0,0) -- (1,1) node[above] {\( R_1 \)};
\draw (1,1) -- (1,2) node[above] {\( R_2 \)};
\draw (1,2) -- (2,1) node[above] {\( R_3 \)};
\draw (2,1) -- (2,0) node[below] {\( a \)};
\draw (1,1) -- (1,0) node[below] {\( b \)};
\draw (1,0) -- (1,-1) node[below] {\( c \)};
\end{tikzpicture}
\end{center}

The area underneath the piecewise-linear function consists of two regions \( R_1 \) and \( R_2 \); the area above the approximation and below \( f_X(\cdot) \) is \( R_3 \). If we let \( p_i \) denote the area of \( R_i \), then it is clear that \( p_1, p_2, \) and \( p_3 \) form a probability mass function. Furthermore,

\begin{align*}
f_X(x) &= p_1 f_{Y_1}(x) + p_2 f_{Y_2}(x) + p_3 f_{Y_3}(x)
\end{align*}

where

\begin{align*}
f_{Y_1}(x) &= \begin{cases} 
2(x - a) \\ (b - a)^2 
\end{cases}, \quad a \leq x \leq b \\
&0, \quad \text{otherwise} \\

f_{Y_2}(x) &= \begin{cases} 
2(c - x) \\ (c - b)^2 
\end{cases}, \quad b \leq x \leq c \\
&0, \quad \text{otherwise} \\

f_{Y_3}(x) &= \frac{1}{p_3} (f_X(x) - p_1 f_{Y_1}(x) - p_2 f_{Y_2}(x))
\end{align*}
To see this, set \( g_i(x) = p_i f_{y_i}(x) \) for \( i=1,2,3 \) and observe that \( g_i(x) = f_{y_i}(x) = 0 \) for \( x \in [a,b] \). Now pick a value \( z \in [a,b] \) and observe that

\[
P(X \leq z) = \int_{a}^{z} f_{y_i}(x) \, dx = \int_{a}^{z} \left[ g_i(x) + (f_{y_i}(x) - g_i(x)) \right] \, dx = \int_{a}^{z} \left( p_1 f_{y_i}(x) + p_2 f_{y_2}(x) \right) \, dx,
\]

so that the desired result follows after taking derivatives. A similar argument establishes the result when \( x \in [b,c] \).

To generate random variables from triangular densities is easy and very fast. Inversion can be used, or one can take advantage of the following:

**Exercise:** Let \( Z_1, Z_2 \) be two independent uniform random variables on \([a, b]\). Prove that the random variable \( \max(Z_1, Z_2) \) has density \( f_{y_1}(\cdot) \). (If inversion is used, a square root needs to be computed; this avoids the square root, at the cost of generating an additional random variable).

**Exercise:** Let \( Z_1, Z_2 \) be two independent uniform random variables on \([b, c]\). Prove that the random variable \( \min(Z_1, Z_2) \) has density \( f_{y_2}(\cdot) \).

Hence, so long as \( p_3 \) is small, this composition approach to generating the random variable \( X \) will be efficient, because whatever routine is used to generate \( Y_3 \) will only be used infrequently (and hence we don’t care much if the routine is relatively inefficient).

Very complex densities \( f_{x}(\cdot) \) may be approximated by combinations of "triangles" (i.e. triangular densities of the above form) and "rectangles" (i.e. uniform random variables). One then needs some (possibly inefficient) algorithm to generate variates from what is "left over" after the approximation. But, as in the above example, this will only be called infrequently. This approach, while time-consuming to code, can lead to very fast variate generation schemes.

### 5. Generating Discrete Random Variables: The Alias Method

We return now to the question of how to efficiently generate discrete random variables with a given probability mass function. Specifically, suppose our goal is to generate a random variable \( X \) for which

\[
P(X = x_i) = p_i \quad \text{for} \quad 1 \leq i \leq n.
\]

The alias method relies upon the idea that if the \( p_i \)'s are uniform (so that \( p_i = 1/n \) for \( 1 \leq i \leq n \)), then \( X \) can be rapidly generated. Conceptually, we can think of dropping points randomly in a rectangle that is \( 1/n \) units tall and \( n \) units wide. The rectangle is divided into \( n \) vertical regions, each 1 unit wide, numbered \( 1,2,\ldots, n \) from left to right. E.g., for \( n = 4 \) we have:
If the generated point lands in region \( i \), then we return the value \( x_i \). In this simple case we don’t need to actually generate the \( y \)-coordinate of our random point, and can implement our technique efficiently by generating a uniform(0,1) random variable \( U \) and setting

\[
X = \left\lceil nUx \right\rceil,
\]

where \( \left\lceil z \right\rceil \) denotes the smallest integer greater than or equal to \( z \). (We don’t generate an explicit \( x \) value, we merely determine in which region our \( x \) value falls.)

For non-uniform \( p_i \)'s, the idea now is to “shift” probability mass around so as to make a rectangular region---where the vertical regions may now be horizontally partitioned into subregions---and use essentially the above idea. In shifting mass to make the rectangle, it turns out that the \( i \)th vertical region never needs to be chopped up into more than two parts: one part is associated with the original \( x_i \) value, and the other part is associated with an alternate value \( x_j \) for some \( j \neq i \). The alternate value is called the alias value. (See figures.)

The \( x \)-coordinate of the random point is handled as before---i.e., we don’t generate explicit \( x \)-values but merely choose which region we fall in. Similarly, we don’t generate explicit \( y \)-values, but rather choose which of the two parts of the region that the point falls in. The point falls in the lower part of vertical region \( i \) with probability \( r_i = (\text{height of lower part of region } i)/(\text{height of region } i) \).

In general, to use the alias method, we first need to execute a set-up algorithm to generate the alias values and probability table. For \( 1 \leq i \leq n \), this yields a set of \( a_i \)'s (the alias values) and \( r_i \)'s (the aliasing probabilities).

We can then employ the following algorithm:

1. Generate two independent uniform \((0,1)\) random variables \( U_1 \) and \( U_2 \).
2. Set \( I = \left\lceil nU_1 \right\rceil \).
3. If \( U_2 \leq r_I \), then return \( x_I \). Otherwise, return \( a_I \).

To set up the alias table requires running an algorithm that executes in \( O(n) \) time. (In other words, the algorithm runs in time which is essentially linear in \( n \).) See Appendix 8B in L&K for details. When \( n \) is large, some care has to be taken when programming to avoid numerical round-off errors.
Example 1:

```
<table>
<thead>
<tr>
<th>i</th>
<th>x_i</th>
<th>p_i</th>
<th>a_i</th>
<th>r_i</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>x_1</td>
<td>1/6</td>
<td>x_2</td>
<td>0.5</td>
</tr>
<tr>
<td>2</td>
<td>x_2</td>
<td>1/2</td>
<td>x_2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>x_3</td>
<td>1/3</td>
<td>x_3</td>
<td>1</td>
</tr>
</tbody>
</table>
```

Example 2:

```
<table>
<thead>
<tr>
<th>i</th>
<th>x_i</th>
<th>p_i</th>
<th>a_i</th>
<th>r_i</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>x_1</td>
<td>3/7</td>
<td>x_1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>x_2</td>
<td>3/7</td>
<td>x_1</td>
<td>5/7</td>
</tr>
<tr>
<td>3</td>
<td>x_3</td>
<td>1/7</td>
<td>x_2</td>
<td>3/7</td>
</tr>
</tbody>
</table>
```