13 Generating Random Variables for Simulation

At this point, we have discussed many discrete and continuous distributions. This chapter shows how we can generate instances of these distributions and others. This is helpful when performing simulations of computer systems, as in Chapter 14. For example, we might have a computer system where the interarrival times of jobs are well modeled by an Exponential distribution and the job sizes (service requirements) are well modeled by a Pareto distribution. To simulate the system, we need to be able to generate instances of Exponential and Pareto random variables. This chapter presents the two basic methods used for generating instances of random variables. Both of these methods assume that we already have a generator of Uniform$(0, 1)$ random variables,[1] as is provided by most operating systems.[2]

13.1 Inverse Transform Method

To generate instances of a random variable (r.v.), X, this method assumes that:

1. We know the cumulative distribution function (c.d.f.) of X, that is, we know $F_X(x) = \mathbf{P}\{X \le x\}$.
2. We can easily invert $F_X(x)$, that is, we can get x from $F_X(x)$.

The high-level plan is that we will generate a random instance of Uniform$(0, 1)$ (this is already available from our operating system), and then find a way to translate that to an instance of X.

[1] Actually, most operating systems provide a random integer between 1 and $N = 2^{32} - 1$. This is easy to convert into a Uniform$(0, 1)$ by just dividing by N.

[2] One cannot always trust the random number generator provided by one's operating system. It is worth reading the literature on what guarantees different random number generators provide and on how to "seed" the random number generator [10].

13.1.1 The Continuous Case

We assume without loss of generality that X ranges from 0 to ∞. The method works just as well when X has some finite upper bound.

Idea: Let u be our random instance from $U(0,1)$. We want to map u to x, where x is an instance of the r.v. X. The key point is that the x that we output needs to be consistent with the distribution of X.

Let's suppose there is some mapping which takes each u and assigns it a unique x. Such a mapping is illustrated by $g^{-1}(\cdot)$ in Figure 13.1. Here, the y-axis shows u, between 0 and 1, being mapped to an x on the x-axis between 0 and ∞.

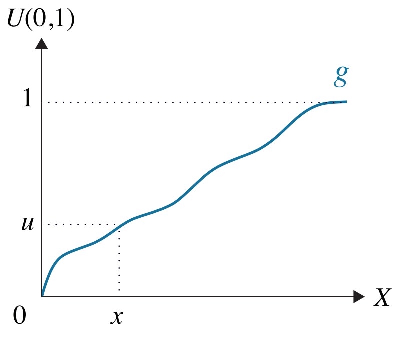

Figure 13.1 *Illustration of mapping $g(\cdot)$.*

Question: Can you figure out what the mapping, $g^{-1}(\cdot)$, should be?

Hint: Think about what property we want for our output. What should be the probability of outputting a value between 0 and x?

Answer: A value in $(0, x)$ should be output with probability $F_X(x)$.

Question: What is the actual probability that $g^{-1}(\cdot)$ outputs a value in $(0, x)$?

Answer: Because $g^{-1}(\cdot)$ only maps values in $(0, u)$ to values in $(0, x)$, the probability of outputting a value in $(0, x)$ is the probability that the uniform instance is in $(0, u)$.

Question: And what is the probability that the uniform instance is in $(0, u)$?

Answer: u.

So we want that

$$u = \mathbf{P}\{0 < U < u\} = \mathbf{P}\{0 < X < x\} = F_X(x).$$

That is, we want

$$u = F_X(x) \quad \text{or equivalently} \quad x = F_X^{-1}(u). \tag{13.1}$$

Question: So what was the $g(\cdot)$ function in Figure 13.1?

Answer: $g(\cdot) = F_X(\cdot)$, the c.d.f. of X.

Algorithm 13.1 (Inverse Transform method to generate continuous r.v. X)

1. *Generate $u \in U(0, 1)$.*
2. *Return $x = F_X^{-1}(u)$. That is, return x such that $F_X(x) = u$.*

Example 13.2 *Generate $X \sim Exp(\lambda)$:*

For the Exp(λ) distribution,

$$F(x) = 1 - e^{-\lambda x}.$$

So, by (13.1) we want,

$$\begin{aligned} x &= F^{-1}(u) \\ F(x) &= u \\ 1 - e^{-\lambda x} &= u \\ -\lambda x &= \ln(1-u) \\ x &= -\frac{1}{\lambda}\ln(1-u). \end{aligned} \tag{13.2}$$

Given $u \in U(0, 1)$, setting $x = -\frac{1}{\lambda}\ln(1-u)$ produces an instance of $X \sim$ Exp(λ).

13.1.2 The Discrete Case

The discrete case follows the same basic idea as the continuous case (see Figure 13.2). This time, we want to generate a discrete r.v., X, with the following probability mass function (p.m.f.):

$$X = \begin{cases} x_0 & \text{w/prob } p_0 \\ x_1 & \text{w/prob } p_1 \\ \dots & \\ x_k & \text{w/prob } p_k \end{cases}.$$

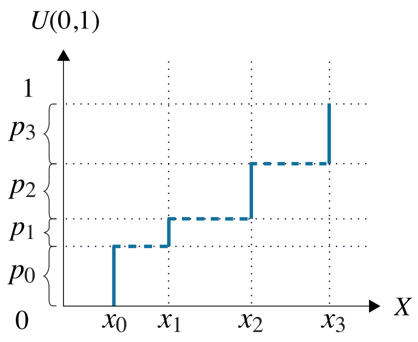

Figure 13.2 *Generating a discrete r.v. with four values.*

Algorithm 13.3 (Inverse Transform method to generate discrete r.v. X)

```
1. Arrange $x_0, \ldots, x_k$, the possible values of $X$, s.t. $x_0 < x_1 < \ldots < x_k$.
2. Generate $u \in U(0,1)$.
3. If $0 < u \leq p_0$, then output $x_0$.
   If $p_0 < u \leq p_0 + p_1$, then output $x_1$.
   If $p_0 + p_1 < u \leq p_0 + p_1 + p_2$, then output $x_2$.
   If $\sum_{i=0}^{\ell-1} p_i < u \leq \sum_{i=0}^{\ell} p_i$, then output $x_\ell$, where $0 \leq \ell \leq k$.
```

Notice that again our $g(\cdot)$ function, shown in blue in Figure 13.2, is $F_X(\cdot)$, the c.d.f.

This sounds easy enough, but it is not always practical. If X can take on many values, then we have to compute many partial sums: $\sum_{i=0}^{\ell} p_i$ for all $0 \leq \ell \leq k$. For this method to be practical, we therefore need closed-form expressions for $\sum_{i=0}^{\ell} p_i$ for all ℓ. Equivalently, we need a closed form for $F_X(x) = \mathbf{P}\{X \leq x\}$ for any x. Then we could do the same thing as in the continuous case, as in (13.1): generate $u \in U(0,1)$, and set $x = F_X^{-1}(u)$, where a ceiling may be necessary since x is discrete. Thus, as in the continuous case, we need to *both* have a closed-form expression for the c.d.f. and also know how to invert this function.

13.2 Accept–Reject Method

The Inverse Transform method required both knowing the c.d.f., $F_X(x)$, of the r.v. X that we're trying to generate, and also being able to invert $F_X(x)$. However, there are many cases where we aren't able to satisfy both of these requirements.

The Accept–Reject method has easier requirements:

1. We need the p.d.f., $f_X(t)$ (or p.m.f.) of the r.v. X that we're trying to generate.
2. We need to know how to generate some other r.v. Y, where Y and X take on the same set of values, that is,

$$f_X(t) > 0 \iff f_Y(t) > 0.$$

The Accept–Reject method is very simple. We generate an instance of Y. Then with some probability we return that value as our instance of X, and otherwise we reject that value and try again.

13.2.1 Discrete Case

Here's the algorithm for a discrete r.v. X, with p.m.f. $p_X(i) = \mathbf{P}\{X = i\}$.

Algorithm 13.4 (Accept-Reject algorithm to generate discrete r.v. X)

1. *Find a discrete r.v. Y, which we already know how to generate, where*

$$p_Y(i) > 0 \iff p_X(i) > 0.$$

2. *Let $c > 1$ be the smallest constant such that*

$$\frac{p_X(i)}{p_Y(i)} \le c, \quad \forall i \text{ s.t. } p_X(i) > 0.$$

3. *Generate an instance of Y. Call this instance i.*
4. *With probability Accept-Ratio$(i) = \frac{p_X(i)}{c p_Y(i)}$, accept i and return $X = i$. Else, reject i and return to step 3.*

Question: In Step 4 of the Accept–Reject algorithm, how do we implement accepting i with probability Accept-Ratio(i)?

Answer: We generate a Uniform$(0, 1)$ r.v. and accept if the generated Uniform is smaller than Accept-Ratio(i).

Question: What's the intuition behind Accept-Ratio(i) in Step 4?

Answer: We can think of

$$\text{Accept-Ratio}(i) = \frac{p_X(i)}{c p_Y(i)}$$

as representing the *relative likelihood* of i being an instance of X versus Y. If this likelihood is high, then we are more likely to trust i as a reasonable instance of X. If the likelihood is low, then even if i is a common instance for Y, it is not a common instance for X and hence we are more likely to reject i as an instance of X.

Question: What role does c play in the accept ratio?

Answer: c is just a normalizing constant which is needed to ensure that the accept ratio is a probability (< 1).

Formally, we have the following argument:

$$\begin{aligned}
&\text{Fraction of time } i \text{ is generated and accepted}\\
&\quad = \mathbf{P}\{i \text{ is generated}\} \cdot \mathbf{P}\{i \text{ is accepted given } i \text{ is generated}\}\\
&\quad = p_Y(i) \cdot \frac{p_X(i)}{c p_Y(i)}\\
&\quad = \frac{p_X(i)}{c}. \qquad (13.3)
\end{aligned}$$

$$\begin{aligned}
&\text{Fraction of time any value is accepted}\\
&\quad = \sum_i \text{Fraction of time } i \text{ is generated and is accepted}\\
&\quad = \sum_i \frac{p_X(i)}{c}\\
&\quad = \frac{1}{c}. \qquad (13.4)
\end{aligned}$$

Combining (13.3) and (13.4), we have:

$$\mathbf{P}\{X \text{ is set to } i\} = \frac{\text{Frac. time } i \text{ is generated and accepted}}{\text{Frac. time any value is accepted}} = \frac{\frac{p_X(i)}{c}}{\frac{1}{c}} = p_X(i),$$

as desired.

Question: On average, how many values of Y are generated before one is accepted?

Answer: c. Because the fraction of time any value is accepted is $\frac{1}{c}$.

13.2.2 Continuous Case

The Accept–Reject method works the same way for continuous random variables, except that we now use the p.d.f. instead of the p.m.f.

Algorithm 13.5 (Accept-Reject algorithm to generate continuous r.v. X)

1. *Find a continuous r.v. Y, which we already know how to generate, where*
$$f_Y(t) > 0 \iff f_X(t) > 0.$$
2. *Let $c > 1$ be the smallest constant such that*
$$\frac{f_X(t)}{f_Y(t)} \le c, \quad \forall t \text{ s.t. } f_X(t) > 0.$$
3. *Generate an instance of Y. Call this instance t.*
4. *With probability Accept-Ratio$(t) = \frac{f_X(t)}{c \cdot f_Y(t)}$, accept t and return $X = t$. Else, reject t and return to step 3.*

Similarly to the Accept–Reject algorithm for the discrete case, we can show that:

$$\begin{aligned}\text{Density of returning } t \text{ on an iteration} &= \text{Density of generating } t \cdot \mathbf{P}\{\text{accept } t\}\\ &= f_Y(t) \cdot \frac{f_X(t)}{c \cdot f_Y(t)}\\ &= f_X(t) \cdot \frac{1}{c}.\end{aligned}$$

Hence,

$$\mathbf{P}\{\text{Return some value on a given iteration}\} = \int_t f_X(t) \cdot \frac{1}{c} dt = \frac{1}{c},$$

so the expected number of iterations needed to get an instance of X is c.

Example 13.6 *Generate r.v. X with p.d.f. $f_X(t) = 20t(1-t)^3, \quad 0 < t < 1$.*

If you plot $f_X(t)$, it looks like Figure 13.3. Observe that X has positive p.d.f. only in the interval $(0, 1)$. Thus we want to choose a Y that is easy to generate and also has positive p.d.f. only in $(0, 1)$.

Question: Any ideas for what $f_Y(t)$ should be?

Answer: Consider simply $f_Y(t) = 1$, where $0 < t < 1$.

Question: Suppose we now apply the Accept–Reject method. What will c be?

Answer: Based on the plot, c should not be too bad – just over 2. To determine c precisely, we want to determine

$$\max_t \left\{ \frac{f_X(t)}{f_Y(t)} \right\} = \max_t \left\{ 20t(1-t)^3 \right\}.$$

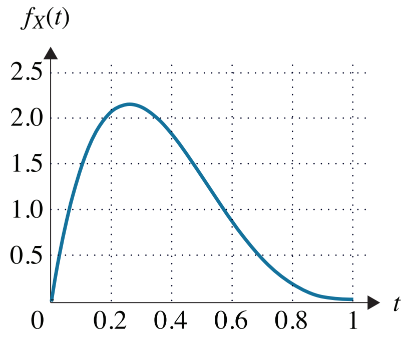

Figure 13.3 *Plot of $f_X(t)$.*

Taking the derivative with respect to t, and setting it equal to zero, we have

$$\frac{d}{dt}(20t(1-t)^3) = 0 \Longleftrightarrow t = \frac{1}{4}.$$

So the maximum value is obtained when $t = \frac{1}{4}$:

$$c = \frac{f_X\left(\frac{1}{4}\right)}{f_Y\left(\frac{1}{4}\right)} = 20\left(\frac{1}{4}\right)\left(\frac{3}{4}\right)^3 = \frac{135}{64}. \qquad (13.5)$$

Observe how easy it was to make a good guess for $f_Y(t)$ just by looking at the plot of $f_X(t)$.

Question: Could we have used the Inverse Transform method to generate X?

Answer: No. While it is easy to get $F_X(x)$, unfortunately $F_X(x)$ is not easy to invert. Thus we won't be able to solve $u = F_X(x)$ for x.

Example 13.7 (Generating a Normal r.v.)

We now turn to generating the Normal distribution. By the Linear Transformation Property (Theorem 9.5), it suffices to generate a standard Normal. Here it's clearly impossible to use the Inverse Transform method since we don't know the c.d.f.

Goal: Generate $N \sim \text{Normal}(0, 1)$.

Idea: It suffices to generate $X = |N|$ and then multiply X by -1 with probability 0.5.

So how do we generate such an X? A plot of X is shown in Figure 13.4.

$$f_X(t) = \frac{2}{\sqrt{2\pi}} e^{-\frac{t^2}{2}}, \quad 0 < t < \infty.$$

Question: What is a good choice for a r.v. Y that we know how to generate, such that $f_Y(t)$ fits $f_X(t)$ reasonably well?

Answer: Let $Y \sim \text{Exp}(1)$.

$$f_Y(t) = e^{-t}, \quad 0 < t < \infty.$$

Observe that $f_X(t)$ is not too much higher than $f_Y(t)$, according to Figure 13.4.

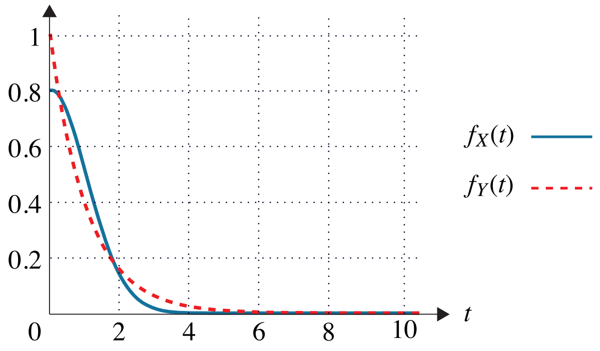

Figure 13.4 *Solid line shows $f_X(t)$. Dashed line shows proposed $f_Y(t)$.*

Question: How many iterations are needed on average?

Answer: We need to determine c.

$$\frac{f_X(t)}{f_Y(t)} = \frac{2}{\sqrt{2\pi}} e^{-\frac{t^2}{2}+t} = \sqrt{\frac{2}{\pi}} e^{t-\frac{t^2}{2}}.$$

So, the maximum value occurs when $t - \frac{t^2}{2}$ is maximized.

$$0 = \frac{d}{dt}\left(t - \frac{t^2}{2}\right) = 1 - t \quad \Longrightarrow \quad t = 1.$$

So,

$$c = \frac{f_X(1)}{f_Y(1)} = \sqrt{\frac{2e}{\pi}} \approx 1.3.$$

Thus we only need 1.3 iterations on average!

13.2.3 A Harder Problem

Consider $X \sim \text{Poisson}(\lambda)$, with $p_X(i) = \frac{e^{-\lambda}\lambda^i}{i!}$.

Question: Can we use the Inverse Transform method to generate an instance of a Poisson r.v.?

Answer: There is no closed form for $F(x) = \mathbf{P}\{X \le x\}$ so the Inverse Transform method will not work.

Question: Can we use the Accept–Reject method?

Answer: It looks like we should be able to apply the Accept–Reject method, but it is hard to find the right Y distribution to match up to (see [48, p. 503]).

We will show a different way to generate $X \sim \text{Poisson}(\lambda)$ in Exercise 13.5 by relating X to a Poisson process with rate λ.

13.3 Readings

A lot more is known about generating random variables than we have described in this chapter. Some particularly well-written texts are [63] and [48].

13.4 Exercises

13.1 **Generating random variables for simulation** – (from [63])
Give an algorithm for generating X with p.d.f. $f_X(x) = 30(x^2 - 2x^3 + x^4)$ where $0 < x < 1$.

13.2 **Inverse Transform method**
Provide a simple algorithm for generating values from a continuous distribution with p.d.f. $f(t) = \frac{5}{4}t^{-2}$, where $1 < t < 5$.

13.3 **Generating a Geometric distribution**
Give a simple and efficient algorithm for generating values from a Geometric(p) distribution. Now use your algorithm to generate 50 instances of Geometric(0.2). Determine the sample mean (the average of the 50 generated instances). Compare the sample mean with the desired answer.

13.4 **Simulation of heavy-tailed distributions**
Write a short program to generate 100 instances of

$$X \sim \text{BoundedPareto}(k = 332.067, p = 10^{10}, \alpha = 1.1),$$

as defined in Definition 10.5. Take their average. Record your answer. Now generate 1000 instances of this distribution, and again take their average and record your answer. Keep going. This time, generate 10,000 instances of this distribution and take their average and record it. Next, generate 100,000 instances. Keep going until you run out of patience. Create a plot where your x-axis shows the number of instances and your y-axis shows the sample average. You should find that your sample averages are well below the true average (compute this!). Explain why this is. What trend do you see in your sample averages?

13.5 **Generating a Poisson r.v.**
Describe an efficient algorithm for generating instances of a Poisson r.v. with mean 1. It will be helpful to start by recalling what you learned in Chapter 12 about the Poisson process and where the Poisson distribution comes up in the context of a Poisson process.

13.6 **Simulating jointly distributed random variables**
Let X and Y be non-negative, continuous random variables whose joint density is given by:

$$f_{X,Y}(x, y) = \lambda e^{-\lambda x} x e^{-xy}, \quad x \geq 0,\ y \geq 0.$$

Provide a simple algorithm, using the Inverse Transform method, that generates a point (x, y) drawn from the above joint p.d.f. Explain your reasoning.