Algorithms for random variates generation

Enrico Fortuna, 1872458

Statistics, Cybersecurity Master Course
Sapienza University of Rome
December 2023

Contents

Chapter 1

Introduction

In statistics and probability, a **random variate** represents a specific outcome or manifestation of a random variable. The random variates play a crucial role in various fields and applications due to their ability to mimic and model uncertainty, variability, and randomness found in the real world.
Usually a random variate is generated from a particular *probability distribution*: this process is known as **random variate generation** or *non-uniform pseudo-random variate generation.*

There are several method or **algorithms** to generate a random variate, in which the main goal is to use the *standard uniform distribution* $\mathcal{U}(0, 1)$ to generate variates from other distributions, e.g., *discrete distributions* like Bernoulli, Binomial and Poisson, or *continuous distributions* like the exponential or the normal.

Chapter 2

Definitions

To better understand and describe the random variate algorithms, we need first to define the definition of *random variable* (X), *random variate* (x) and *standard uniform distribution* ($\mathcal{U}(0,1)$).

2.1 Random variable

A **random variable** (*RV*), usually denoted with X, is a measurable function from Ω, the set of the possible outcomes of a probability triple $(\Omega, \mathcal{F}, P)$, to another measurable space R.
In the standard case X is a real value, i.e. $R = \mathbb{R}$. When X is countable, the *RV* is called **discrete random variable**; if the image is uncountable (like an interval), X is called **continuous random variable**. There is a further case in which the *RV* is **absolutely continuous** and its distribution can be described with a **probability density function** (*pdf*). The latter describes how the probability of the proportion of observations change over the range of the distribution.
It is important to notice that any random variable can be characterized by its **cumulative distribution function** (*cdf*), which calculates the probability of an observation equal or less than a value (i.e. $(F(x) = P(X \leq x)$ for some x.

2.2 Random variate

A random variate generation algorithm was defined by **Luc Devroye**[2] as follows:

Suppose that

1. Computers can manipulate real numbers;

2. Computers have access to a source of random variates that are uniformly distributed on the closed interval [0,1];

Then a random **variate generation algorithm** is defined as any program that stops almost surely and provides in output a real number x. This x is called **random variate**.

The above two assumption are violated in almost all the real computers, because computers lack the capability to directly manipulate real numbers and often using floating-point representations instead. In addition to this, many of the computers lacking a source of *true randomness*, while rather using *pseudo-random number* sequence.

2.3 Standard uniform distribution

The **standard uniform distribution** is a particular case of the **continuous uniform distribution**. The latter is a two-parameter family of symmetric curves which has a *constant probability density function* between its two bounding parameters. Usually the distribution is abbreviated with $\mathcal{U}(a, b)$, where $\mathcal{U}$ stands for the *uniform distribution*, while a and b define the bounds, which are the minimum and the maximum values respectively. The interval can be closed, i.e. $[a, b]$, or open, i.e. (a, b).

The **standard uniform distribution** is a uniform distribution where $a = 0$ and $b = 1$, which is very common in statistics, especially for random number generation. Its expected value is $\frac{1}{2}$ and variance is $\frac{1}{12}$[5].

The notation $X \sim \mathcal{U}(0, 1)$ indicates that the random variable X follows *standard uniform distribution* over the interval $[0, 1]$[7]. A standard uniform random variable X has the following *pdf*:

$$f(x) = 1 \qquad 0 < x < 1 \tag{2.1}$$

Chapter 3

Algorithm for random variate generation

Before to show the algorithms, we must assume that a source of *uniform random number* $\mathcal{U}(0,1)$ exists (i.e. the **standard uniform distribution**, where every point in the continuous range $[0.0, 1.0]$ has the same opportunity of appearing). There are several methods for generating random variates: some are easier to implements than others, some are more efficient than others.

3.1 Inverse Transform Method

The **inverse transform method** can be used in principle for any distribution, but it is useful when we want to sample from some continuous distribution with *cdf* $F(x) = P(X \leq x)$, which has an **inverse** $F^{-}1(x)$ that is easy to calculate (3.1). Therefore let X be a continuous RV with *cdf* $F(x)$, then:

$$F(X) \sim \mathcal{U}(0,1) \tag{3.1}$$

Is it possible to define the inverse *cdf* as follows:

$$F^{-1}(u) = inf\,[x : F(x) \geq u] \quad u \in [0,1] \tag{3.2}$$

So let $U \sim \mathcal{U}(0,1)$. Then the **inverse transform method** for generating X having *cdf* $F(x)$ is the following:

Algorithm 1 Inverse transform method

```
$U \xleftarrow{\$} \mathcal{U}(0,1)$                ▷ Sample $U$ from $\mathcal{U}(0,1)$
return $X \leftarrow F^{-1}(U)$
```

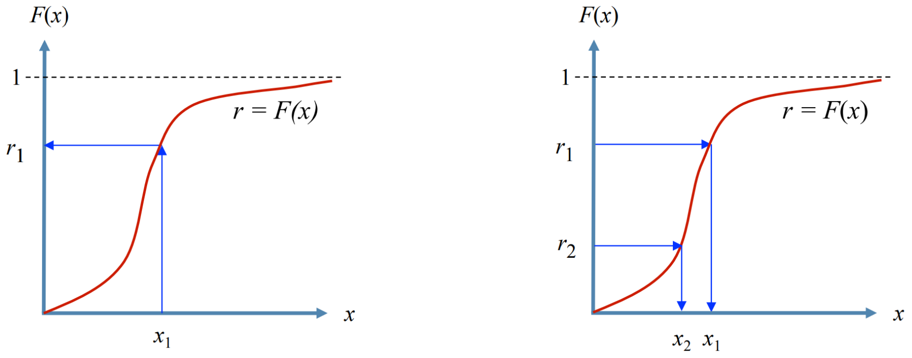

Figure 3.1: In this case the inverse of x_1 is r_1, left image, and the inverse of x_2 is r_2, right image. [3]

Example 3.1.1. The exponential distribution.

- $F(x) = 1 - e^{\lambda x}$, $x > 0$;
- Solving $F(X) = U$ for X,

$$X = -\tfrac{1}{\lambda}\ln(1-U) \quad \text{or} \quad X = -\tfrac{1}{\lambda}\ln(U). \qquad \square$$

The below snippet of code shows an example of implementation in Python:

```
    def rand_expon(lambda):
        U = rand()
        return -ln(1-U)/lambda

```

Listing 3.1: Example of Python implementation for the exponential distribution case.

Example 3.1.2. The Weibull distribution.

- $F(x) = 1 - e^{-(\lambda x)^{\beta}}$, $x > 0$;
- Solving $F(X) = U$ for X,

$$X = \tfrac{1}{\lambda}\left[-ln(1-U)\right]^{1/\beta} \quad \text{or} \quad X = \tfrac{1}{\lambda}\left[-\ln(U)\right]^{1/\beta}. \qquad \square$$

3.2 Acceptance-Rejection Method

Suppose this time we want to sample from some *continuous distribution* $f(x)$, but we can't easily/efficiently compute the inverse *cdf* $F^{-1}(x)$. The **Acceptance-rejection Method** *(A-R)* samples from a distribution which closely approximates the desired one, and then refines by "accepting" only a specific portion of those samples.

Generally, as shown in the figure 3.2, we can approximate $f(x)$ by using a distribution similar to it, denoted $g(x)$. We know the density of $g(x)$ and we can easily sample from it, so there is a constant c such that $f(x) \leq c * g(x)$, $\forall x$.

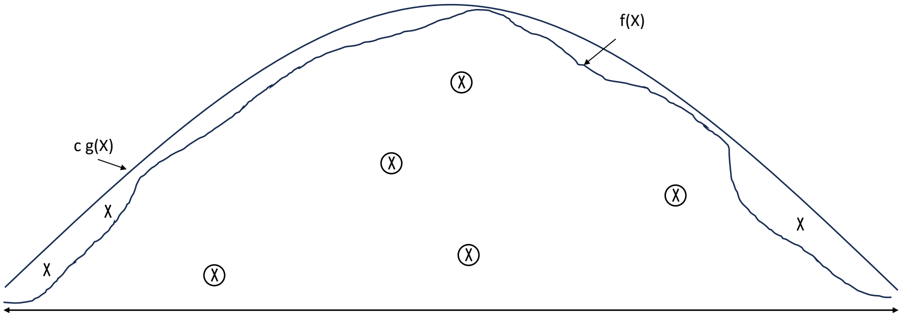

Figure 3.2: The function $g(x)$ majorizes $f(x)$ for a constant c, i.e. $f(x) \leq c * g(x)$, $\forall x$.

The generalized Acceptance-Rejection Algorithm is the following[4]:

Algorithm 2 Acceptance-Rejection Algorithm

```
$c \leftarrow max(f(x)/g(x)),\ \forall x$
repeat
    $X \xleftarrow{\$} g(x)$                ▷ draw a sample from $g(x)$
    $U \xleftarrow{\$} \mathcal{U}[0,1)$    ▷ draw a sample from $\mathcal{U}[0,1)$
until $U * c * g(X) > f(X)$
return X
```

Example 3.2.1. Suppose the distribution we want has the following density:

- $f(x) = Kx^{\frac{1}{2}}e^{-x}$, where $K = \frac{2}{\sqrt{\pi}},\ x > 0$;
- $f(x)$ is the *pdf* of a gamma distribution with parameters $(\frac{3}{2}, 1)$.

We choose $g(x)$ to be a distribution which looks similar to $f(x)$. Now determine the value of c:

- Find the maximum of $\frac{f(x)}{g(x)}$: take the derivative and set to 0, then solve for $x \to x = \frac{3}{2}$;
- $c = \frac{f(\frac{3}{2})}{g(\frac{3}{2})} = \frac{3^{\frac{3}{2}}}{2\pi e^{\frac{1}{2}}} \sim 1.257$.

After we compute c, we can define $\frac{f(x)}{(c*g(x))}$:

- $\frac{f(x)}{(c*g(x))} = (\frac{2e}{3})^{\frac{1}{2}}x^{\frac{1}{2}}e^{-\frac{x}{3}}$.

Now use the algorithm2:

```
repeat
    $X \leftarrow$ gen_exponential(2/3)          ▷ the function draw from $g(x)$
    $U \leftarrow_{\$} \mathcal{U}[0,1)$                    ▷ draw from uniform
until $(\frac{2e}{3})^{\frac{1}{2}} X^{\frac{1}{2}} e^{-\frac{X}{3}} > U$
return X
```

3.3 Cutpoint Method

Imagine we want to generate a variate from the following *discrete distribution*:

$$P(X = k) = p_k, \quad k = a, a+1, \ldots, b$$

with large $b - a$.

Let

$$q_k = P(X \leq k), \quad k = a, a+1, \ldots, b$$

For fixed m the cutpoint method of Fishman and Moore computes and store the cutpoints:

$$I_j = min\left[k : q_k > \frac{j-1}{m}\right], \quad j = 1, \ldots, m$$

The above cutpoints let us to scan trough the list of possible k-values much more quickly than regular inverse transform 3.1.

Here there is the algorithm to calculate cutpoints:

Algorithm 3 Cutpoint Set Method

```
$j \leftarrow 0$, $k \leftarrow a - 1$, $A \leftarrow 0$
while $j < m$ do
    while $A \leq j$ do
        $k \leftarrow k + 1$
        $A \leftarrow mq_k$
    end while
    $j \leftarrow j + 1$
    $I_j \leftarrow k$
end while
```

After the cutpoints I_j are computed we can use the Cutpoint Method:

Algorithm 4 Cutpoint Method

```
$U \xleftarrow{\$} \mathcal{U}(0,1)$                ▷ Generate $U$ from $\mathcal{U}(0,1)$
$L \leftarrow \lfloor mU \rfloor + 1$              ▷ Select an integer $X \leftarrow I_L$
$X \leftarrow I_L$                         ▷ Starts the search at value $I_L$
while $U > q_X$ do
    $X \leftarrow X + 1$
end while
```

It is correct because of:

$$P(I_L \leq X \leq I_{L+1}) = 1 \quad (I_{m+1} = b)$$

Now, to gain a better understanding, let's look at an example.

Example 3.3.1. Consider this distribution:

k	1	2	3	4	5	6	7	8
p_k	0.01	0.04	0.07	0.15	0.28	0.19	0.21	0.05
q_k	0.01	0.05	0.12	0.27	0.55	0.74	0.95	1

If $m = 8$ the cutpoints are the following (using 3):

$$\begin{aligned} &I_1 = min[i : q_i > 0] = 1 \\ &I_2 = min[i : q_i > 1/8] = 4 \\ &I_3 = min[i : q_i > 2/8] = 4 \\ &I_4 = min[i : q_i > 3/8] = 5 = I_5 \\ &I_6 = min[i : q_i > 5/8] = 6 \\ &I_7 = min[i : q_i > 6/8] = 7 = I_8 \\ &I_9 = 8 \end{aligned}$$

If we choose $U = 0.219$, using the algorithm 4, we have $L = \lfloor mU \rfloor + 1 = \lfloor 8(0.219) \rfloor + 1 = 2$, and then:

$$X = min[i : I_2 \leq i \leq I_3,\, q_i \geq 0.219] = 4. \quad \square$$

3.4 Convolution Method

In short **convolution** means adding things up, i.e., express the random variable itself as the sum of other random variables. Suppose desired *RV* Y has the same distribution as $X_1 + X_2 + \dots X_n$, where the X_i's are *iid* and n is finite and fixed; $Y \sim X_1 + X_2 + \dots X_n$ is called *n-fold convolution* of the distribution of X_i.
The obviously algorithm would be:

Algorithm 5 Convolution Method

```
Generate $X_1 + X_2 + \dots X_n$ independently from their distribution
return $Y \leftarrow X_1 + X_2 + \dots X_n$
```

This technique will be showed with an example.

Example 3.4.1. The Binomial distribution $Bin(n, p)$.

Let $X_1, \dots, X_n$ be *iid* random variable with a Bernoulli distribution $Bern(p)$. Then $Y = \sum_{i=1}^{n} X_i \sim Bin(n, p)$.

How to get Bernoulli RVs:

```
Require: iid $U_1, \dots U_n \leftarrow U(0, 1)$
for every $i = 1, \dots, n$ do
    if $U_i \leq p$ then
        $X_i \leftarrow 1$
    else
        $X_i \leftarrow 0$
    end if
end for
```

3.5 Composition Method

Suppose that a random variable comes from two RVs (e.g., your plane can leave the airport gate late for two reasons — air traffic delays and maintenance delays, which compose the overall delay time [1]). We want to generate a *RV* with the following *cdf*:

$$F(x) = \sum_{j=1}^{\infty} p_j F_j(x)$$

where $p_j > 0$ for all j, $\sum_j p_j = 1$, and all the $F_j(x)$ are easy *cdf's* to generate from.

Algorithm 6 Composition Method

```
generate a positive integer $J$ s.t. $\forall j$, $P(J = j) = p_j$
return $X$ from cdf $F_J(x)$
```

Proof. (X has *cdf* $F(x)$) By the *law of total probability*:

$$\begin{aligned}
P(\text{returned}\, X \leq x) \quad &= \sum_{j=1}^{\infty} P(X \leq x \,|\, J = j) P(J = j) \quad (\text{condition on}\, J = j) \\
&= \sum_{j=1}^{\infty} P(X \leq x \,|\, J = j) p_j \quad (\text{distribution of}\, J) \\
&= \sum_{j=1}^{\infty} F_j(x) p_j \quad (\text{given}\, J = j, X \sim F_j) \\
&= F(x) \quad (\text{decomposition of}\, F)
\end{aligned}$$

□

The strategy is to find F_j's from which generation is easy and fast.

Example 3.5.1. Laplace distribution.

$$f(x) = \begin{cases} \frac{1}{2} e^{x} & \text{if}\, x < 0 \\ \frac{1}{2} e^{-x} & \text{if}\, x > 0 \end{cases}$$

Using some algebra we have:

$$F(x) = \begin{cases} \frac{1}{2} e^{x} & \text{if}\, x < 0 \\ 1 - \frac{1}{2} e^{-x} & \text{if}\, x > 0 \end{cases}$$

Now decompose X into negative exponential and regular exponential distributions:

$$F_1(x) = \begin{cases} e^{x} & \text{if}\, x < 0 \\ 1 & \text{if}\, x > 0 \end{cases}$$

and

$$F_2(x) = \begin{cases} 0 & \text{if}\, x < 0 \\ 1 - e^{x} & \text{if}\, x > 0 \end{cases}$$

Then

$$F(x) = \frac{1}{2} F_1(x) + \frac{1}{2} F_2(x)$$

Therefore with inverse transform using $F_1(x)$ and $F_2(x)$, let:

$$X \leftarrow \begin{cases} \ln U & \text{w.p. } 1/2 \\ -\ln U & \text{w.p. } 1/2 \end{cases} \quad \square$$

3.6 Box-Muller Method

The **Box-Muller Method** is a simple way to generate standard normals.

Theorem 1. *Suppose U_1, U_2 are iid $\mathcal{U}(0,1)$, then:*

$$\begin{aligned} Z_1 &= \sqrt{-2ln(U_1)}cos(2\pi U_2) \\ Z_2 &= \sqrt{-2ln(U_1)}cos(2\pi U_2) \end{aligned}$$

are *iid* $Nor(0,1)$.

3.7 Polar Method

This method is a little faster than the previous one.

Algorithm 7 Polar Method

```
1: repeat
2:     $U_1 \xleftarrow{\$} \mathcal{U}(0,1)$, $U_2 \xleftarrow{\$} \mathcal{U}(0,1)$        ▷ Generate $U_1, U_2$ iid $\mathcal{U}(0,1)$
3:     $V_1 \leftarrow 2U_1 - 1$, $V_2 \leftarrow 2U_2 - 1$
4:     $W \leftarrow V_1^2 + V_2^2$
5: until $W > 1$
6: $Y \leftarrow \sqrt{-2(\ln(W))/W}$
7: $Z_1 \leftarrow V_1 Y$, $Z_2 \leftarrow V_2 Y$
8: return $Z_1$, $Z_2$        ▷ $Z_1, Z_2$ are iid $Nor(0,1)$
```

The following snippet of code[6] implements the Polar method in **Java** using the *mean* and the *standard deviation*:

```java
    private static double spare;
    private static boolean hasSpare = false;

    public static synchronized double generateGaussian(double mean, double stdDev) {
        if (hasSpare) {
            hasSpare = false;
            return spare * stdDev + mean;
        } else {
            double u, v, s;
            do {
                u = Math.random() * 2 - 1;
                v = Math.random() * 2 - 1;
                s = u * u + v * v;
            } while (s >= 1 || s == 0);
            s = Math.sqrt(-2.0 * Math.log(s) / s);
            spare = v * s;
            hasSpare = true;
            return mean + stdDev * u * s;
        }
    }
```

Listing 3.2: Example of Java implementation of Polar method with mean and standard deviation (stDev).

Chapter 4

Conclusion

In this concluding chapter we examine various applications of random variate generation algorithms and provide a summary of the topics covered.

4.1 Applications and Conclusion

Random variates generation algorithms find applications in various fields and scenarios thanks to their ability to simulate randomness and uncertainty. Examples include: **Monte Carlo Simulations**, where random variates are used to model complex systems and make predictions; **Cryptographic Applications**, to generate secure cryptographic keys and ensure the unpredictability of cryptographic algorithms; **Machine Learning** and **Artificial Intelligence**, to introduce randomness in training processes, contributing to model robustness; **Financial Modelling**, in which random variates can be used to model important financial variables; Biostatistics and Epidemiology, to model the variability in patient responses and simulate the spread of diseases. So random variates are used in many disciplines such cryptography, biology and statistics.

In this paper we first introduced the fundamental concepts of random variable, random variate and standard uniform distribution. This allowed us to better understand how the algorithms for generating random variates works. The pseudo-code of the most important algorithm has been shown, as well as some useful examples and code implementation (Java and Python). It is certainly possible to delve deeper into the topic, exploring other types of algorithms and illustrating how to generate stochastic processes using random variates.

Bibliography

[1] Christos Alexopoulos and Dave Goldsman. Random variate generation, 2010. https://www2.isye.gatech.edu/~sman/courses/Mexico2010/Module07-RandomVariateGeneration.pdf.

[2] Luc Devroye. Nonuniform random variate generation. *Handbooks in operations research and management science*, 13:83–121, 2006.

[3] Prof. Dr. Mesut Güneş. Random-variate generation. https://www.mi.fu-berlin.de/inf/groups/ag-tech/teaching/2012_SS/L_19540_Modeling_and_Performance_Analysis_with_Simulation/07.pdf.

[4] Michael Lipschultz. Random variate generation. https://people.cs.pitt.edu/~lipschultz/cs1538/06_random_variates.pdf.

[5] maths support centre UCD. Statistics: Uniform distribution (continuous). https://www.ucd.ie/msc/t4media/Uniform%20Distribution.pdf.

[6] Wikipedia. Marsaglia polar method — Wikipedia, the free encyclopedia, 2022. https://en.wikipedia.org/wiki/Marsaglia_polar_method#:~:text=The%20Marsaglia%20polar%20method%20is,of%20the%20Monte%20Carlo%20method.

[7] William and Mary. Standard uniform distribution. https://www.math.wm.edu/~leemis/chart/UDR/PDFs/Standarduniform.pdf.