Chapter 14

Introduction to Simulation Using Python

E. Sink, A. Rakhshan, and H. Pishro-Nik

14.1 Analysis versus Computer Simulation

A computer simulation is a computer program which attempts to represent the real world based on a model. The accuracy of the simulation depends on the precision of the model. Suppose that the probability of heads in a coin toss experiment is unknown. We can perform the experiment of tossing the coin n times repetitively to approximate the probability of heads.

$$P(H) = \frac{\text{Number of times heads observed}}{\text{Number of times the experiment executed}}$$

However, for many practical problems it is not possible to determine the probabilities by executing experiments a large number of times. With today's computers processing capabilities, we only need a high-level language, such as Python, which can generate random numbers, to deal with these problems.

In this chapter, we present basic methods of generating random variables and simulating probabilistic systems. The provided algorithms are general and can be implemented in any computer language. However, to have concrete examples, we provide the actual code in Python. If you are unfamiliar with Python, you should still be able to understand the algorithms.

14.2 Introduction: What is Python?

Python is a high-level language that assists developers and researchers in solving problems with fewer lines of code than traditional programming languages, such as C/C++ or Java, by leveraging its comprehensive standard library. Python can be utilized for a multitude of applications, ranging from web development and data analysis to artificial intelligence and scientific computations. Lists and dictionaries are fundamental data structures in Python. Thus, a foundational understanding of basic programming concepts is advantageous to harness Python's full capabilities. While this text presumes you are acquainted with basic Python commands, we will explore not only how to generate probability distributions using modules like Numpy, but also how to derive these distributions from the uniform distribution.

In this chapter, we will use the Numpy and Scipy libraries for mathematical functions and Matplotlib's Pyplot interface for plotting, though there are many good choices available. Many Python distributions will include these modules by default. If yours does not, they can be installed from your operating system command line (Command Prompt on Windows, Terminal on MacOS, etc.) using pip. For example, to install Numpy, use

```
python -m pip install numpy
```

To import these modules, use the following Python code:

```
import numpy as np
import scipy as sp
import matplotlib.pyplot as plt
```

We will also need an instance of numpy's Generator class:

```
rng = np.random.default_rng()
```

Make sure to include these commands at the beginning of your code.

14.3 Discrete and Continuous Random Number Generators

Most programming languages can deliver samples from the uniform distribution to us. (In reality, the given values are pseudorandom instead of being completely random.) The rest of this section shows how to convert uniform random variables to any other desired random variable. The Python code for generating uniform random variables is:

```
rng.random()
```

which returns a pseudorandom value drawn from the standard uniform distribution on the half-open interval $[0, 1)$. Also,

```
rng.random(n)
```

returns a list of n independent pseudorandom values drawn from the standard uniform distribution on the half-open interval $[0, 1)$, while

```
rng.random([m,n])
```

returns an $m \times n$ matrix.

14.3.1 Generating Discrete Probability Distributions from Uniform Distribution

Let's see a few examples of generating certain simple distributions:

Example 1. (Bernoulli) Simulate tossing a coin with probability of heads p.

Solution: Let U be a $Uniform(0, 1)$ random variable. We can write a Bernoulli random variable X as:

$$X = \begin{cases} 1 & U < p \\ 0 & U \geq p \end{cases}$$

Thus,

$$\begin{aligned} P(H) &= P(X = 1) \\ &= P(U < p) \\ &= p \end{aligned}$$

Therefore, X has $Bernoulli(p)$ distribution. The Python code for $Bernoulli(p)$ is:

```
def bernoulli(p):
        U = rng.random()
        return (U < p)
```

Since the "random" command returns a number between 0 and 1, we divided the interval $[0, 1]$ into two parts, p and $1 - p$ in length. Then, the value of X is determined based on where the number generated from the uniform distribution fell.

Example 2. (Coin Toss Simulation) Write code to simulate tossing a fair coin to see how the law of large numbers works.

Solution: You can write:

```
n = 1000
U = rng.random(n)
toss = (U < 0.5)
avg = [sum(toss[:i])/i for i in range(1,n+1)]
plt.xlabel("Coin Toss Number")
plt.ylabel("Proportion of Heads")
plt.axis([0,n,0,1])
plt.plot(range(1,n+1), avg)
plt.show()
```

If you run the above code to compute the proportion of 1's in the variable "toss," the result will look like Figure 14.1. You can also assume the coin is biased with probability of heads equal to 0.6 by replacing the third line of the previous code with:

```
toss = (U < 0.6)
```

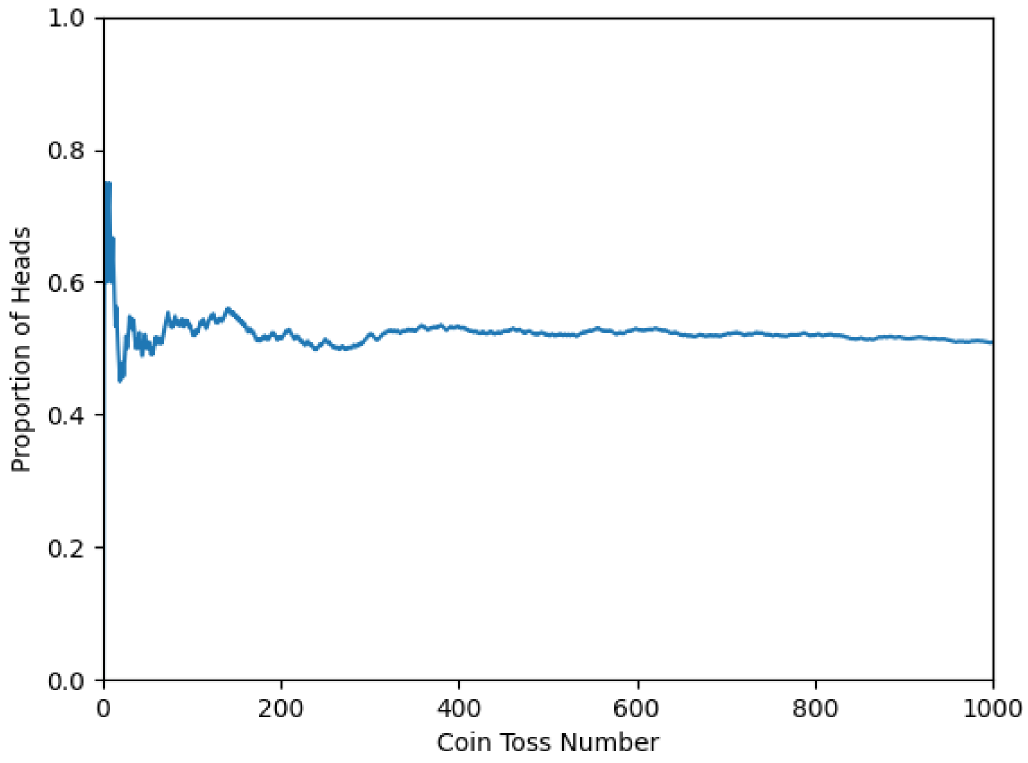


Figure 14.1: Python coin toss simualtion

Example 3. (Binomial) Generate a $Binomial(50, 0.2)$ random variable.

Solution: To solve this problem, we can use the following lemma:

Lemma 1. If $X_1, X_2, ..., X_n$ are independent $Bernoulli(p)$ random variables, then the random variable X defined by $X = X_1 + X_2 + ... + X_n$ has a $Binomial(n, p)$ distribution.

To generate a random variable $X \sim Binomial(n, p)$, we can toss a coin n times and count the number of heads. Counting the number of heads is exactly the same as finding $X_1+X_2+...+X_n$, where each X_i is equal to 1 if the corresponding coin toss results in heads and 0 otherwise.

Since we know how to generate Bernoulli random variables, we can generate a $Binomial(n, p)$ by adding n independent $Bernoulli(p)$ random variables.

```
def binomial(n,p):
        U = rng.random(n)
        return sum(U < p)
```

Generating Arbitrary Discrete Distributions

In general, we can generate any discrete random variables similar to the above examples using the following algorithm. Suppose we would like to simulate the discrete random variable X with range $R_X = \{x_1, x_2, ..., x_n\}$ and $P(X = x_j) = p_j$, so $\sum_j p_j = 1$.

To achieve this, first we generate a random number U (i.e., $U \sim Uniform(0, 1)$). Next, we divide the interval $[0, 1]$ into subintervals such that the jth subinterval has length p_j (Figure 14.2). Assume

$$X = \begin{cases} x_0 & \text{if} \quad (U < p_0) \\ x_1 & \text{if} \quad (p_0 \leq U < p_0 + p_1) \\ \vdots & \\ x_j & \text{if} \quad \left(\sum_{k=0}^{j-1} p_k \leq U < \sum_{k=0}^{j} p_k\right) \\ \vdots & \end{cases}$$

In other words

$$X = x_j \quad \text{if} \quad F(x_{j-1}) \leq U < F(x_j),$$

where $F(x)$ is the desired CDF. We have

$$\begin{aligned} P\left(X = x_j\right) &= P\left(\sum_{k=0}^{j-1} p_k \leq U < \sum_{k=0}^{j} p_k\right) \\ &= p_j \end{aligned}$$

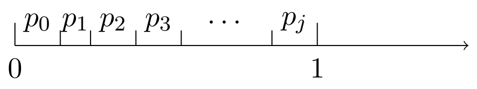


Figure 14.2: Generating discrete random variables

Example 4. Give an algorithm to simulate the value of a random variable X such that

$$\begin{aligned} P(X = 1) &= 0.35 \\ P(X = 2) &= 0.15 \\ P(X = 3) &= 0.4 \\ P(X = 4) &= 0.1 \end{aligned}$$

Solution: We divide the interval $[0, 1]$ into subintervals as follows:

$$\begin{aligned} A_0 &= [0, 0.35) \\ A_1 &= [0.35, 0.5) \\ A_2 &= [0.5, 0.9) \\ A_3 &= [0.9, 1) \end{aligned}$$

Subinterval A_i has length p_i. We obtain a uniform number U. If U belongs to A_i, then $X = x_i$.

$$\begin{aligned} P(X = x_i) &= P(U \in A_i) \\ &= p_i \end{aligned}$$

```
def X():
        A = [0.35,0.5,0.9,1]
        x = [1,2,3,4]
        j = 0
        U = rng.random()
        while U > A[j]:
                j = j + 1
        return x[j]
```

The same result can be achevied using the Numpy choice function:

```
rng.choice([1,2,3,4],p=[0.35,0.15,0.4,0.1])
```

14.3.2 Generating Continuous Probability Distributions from the Uniform Distribution: Inverse Transformation Method

At least in principle, there is a way to convert a uniform distribution to any other distribution. Let's see how we can do this. Let $U \sim Uniform(0, 1)$ and F be a CDF. Also, assume F is continuous and strictly increasing as a function.

Theorem 1. Let $U \sim Uniform(0, 1)$ and F be a CDF which is strictly increasing. Also, consider a random variable X defined as

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

Then,

$$X \sim F \quad \text{(The CDF of } X \text{ is } F\text{)}.$$

Proof:

$$\begin{aligned} P(X \leq x) &= P(F^{-1}(U) \leq x) \\ &= P(U \leq F(x)) \quad \text{(increasing function)} \\ &= F(x) \end{aligned}$$

Now, let's see some examples. Note that to generate any continuous random variable X with the continuous CDF F, $F^{-1}(U)$ has to be computed.

Example 5. (Exponential) Generate an *Exponential*(1) random variable.

Solution: To generate an Exponential random variable with parameter $\lambda = 1$, we proceed as follows:

$$\begin{aligned} F(x) &= 1 - e^{-x} \qquad x > 0 \\ U &\sim Uniform(0,1) \\ X &= F^{-1}(U) \\ &= -\ln(1-U) \\ X &\sim F. \end{aligned}$$

This formula can be simplified since

$$(1-U) \sim Uniform(0,1)$$

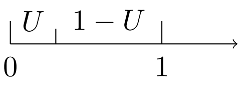


Figure 14.3: Symmetry of uniform distribution

Hence we can simulate X using

$$X = -\ln(U).$$

```
def exponential():
        U = rng.random()
        return -np.log(U)
```

Example 6. (Gamma) Generate a Gamma(20,1) random variable.

Solution: For this example, F^{-1} is even more complicated than the complicated gamma cdf F itself. Instead of inverting the CDF, we generate a gamma random variable as a sum of n independent exponential variables.

Theorem 2. Let $X_1, X_2, \cdots, X_n$ be independent random variables with $X_i \sim$ *Exponential*(λ). Define

$$Y = X_1 + X_2 + \cdots + X_n$$

By the moment generating function method, you can show that Y has a gamma distribution with parameters n and λ, i.e., $Y \sim Gamma(n, \lambda)$.

Having this theorem in mind, we can write:

```
def gamma(n,lam):
        return (-1/lam)*sum(np.log(rng.random(n)))
gamma(20,1)
```

Example 7. (Poisson) Generate a Poisson random variable. Hint: In this example, use the fact that the number of events in the interval $[0, t]$ has Poisson distribution when the elapsed times between the events are Exponential.

Solution: We want to employ the definition of Poisson processes. Assume N represents the number of events (arrivals) in [0,t]. If the interarrival times are distributed exponentially (with parameter λ) and independently, then the number of arrivals occurred in [0,t], N, has Poisson distribution with parameter λt (Figure 14.4). Therefore, to solve this problem, we can repeat generating $Exponential(\lambda)$ random variables while their sum is not larger than 1 (choosing $t = 1$). More specifically, we generate $Exponential(\lambda)$ random variables

$$T_i = \frac{-1}{\lambda} \ln(U_i)$$

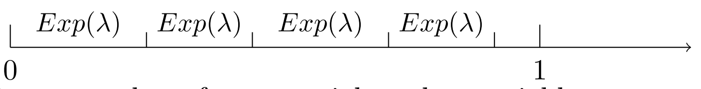
by first generating uniform random variables U_i's. Then we define

$$X = \max\{j : T_1 + \cdots + T_j \leq 1\}$$

The algorithm can be simplified:

$$X = \max\left\{j : \frac{-1}{\lambda} \ln(U_1 \cdots U_j) \leq 1\right\}$$

```
def poisson(lam):
        i = 0
        U = rng.random()
        Y = -(1/lam)*np.log(U)
        sum = Y
        while sum <= 1:
                U = rng.random()
                Y = -(1/lam)*np.log(U)
                sum = sum + Y
                i = i+1
        return i
```



X= Maximum number of exponential random variables

Figure 14.4: Poisson Random Variable

To finish this section, let's see how to convert uniform numbers to normal random variables. The normal distribution is extremely important in science because it is very commonly occuring.

Theorem 3. (Box-Muller transformation) We can generate a pair of independent normal variables (Z_1, Z_2) by transforming a pair of independent $Uniform(0, 1)$ random variables (U_1, U_2) [1].

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

Example 8. (Box-Muller) Generate 5000 pairs of normal random variables and plot both histograms.

Solution:

```
U = rng.random([2,5000])
Z1 = np.sqrt(-np.log(U[0]))*np.cos(2*np.pi*U[1])
Z2 = np.sqrt(-np.log(U[0]))*np.sin(2*np.pi*U[1])
plt.hist((Z1,Z2))
plt.show()
```

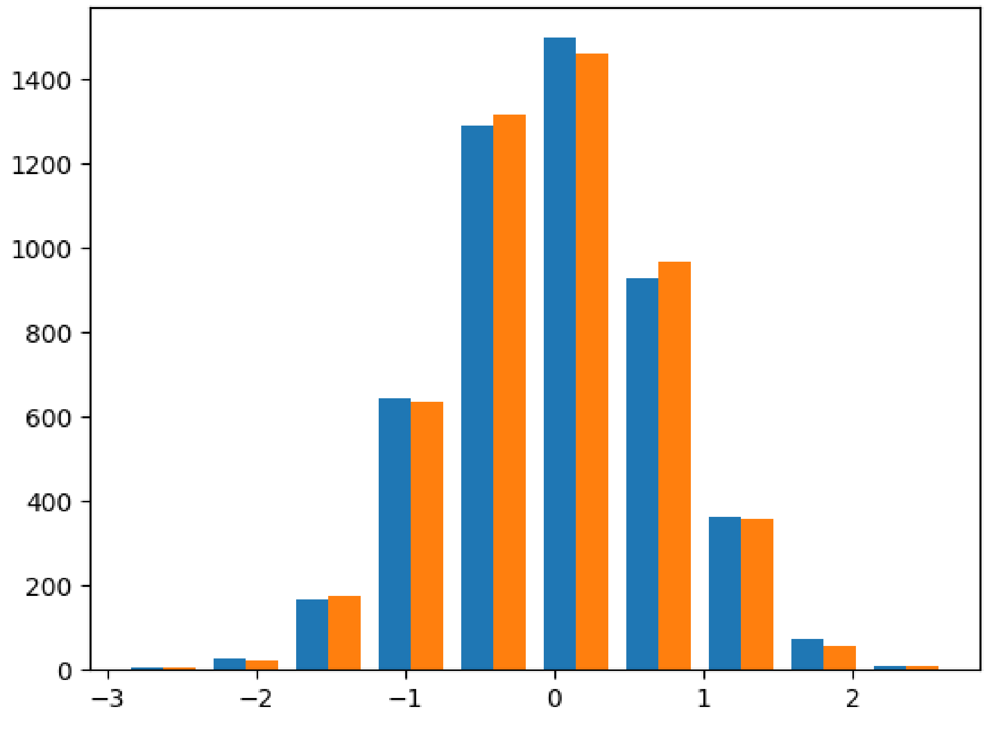


Figure 14.5: Histogram of a pair of normal random variables generated by Box-Muller transformation

14.4 Python Commands for Special Distributions

In the previous section, we saw how we can sample arbitrary distributions using a uniform random number generator. However, for many standard distributions, libraries like Numpy have done this work for us. In practice, it is preferable to use the efficient, extensively documented, and well-tested library functions when they are available.

The following tables catalog some of the distributions available in Numpy. The full list can be found in the Numpy documentation [2]. They are implemented as methods of the Generator class. In addition to the listed arguments, they all take an optional size parameter, which can be an integer or a list of integers. For example, to generate a $a \times b$ matrix drawn from a $Binomial(n,p)$ distribution, we use the following code:

```
rng.binomial(n,p,[a,b])
```

Even more functionality is available in Scipy's stats module. In addition to random sampling of various distributions, Scipy can also compute of the corresponding PMFs/PDFs, CDFs, and more. The above Numpy command is equivalent to the Scipy command

```
sp.stats.binom.rvs(n,p,size=[a,b])
```

The value of the corresponding PMF at k is

```
sp.stats.binom.pmf(k,n,p)
```

and similarly for the CDF. See the Scipy documentation for details [3].

Name	Numpy (rng.*)	Scipy (sp.stats.*)
$Binomial(n,p)$	binomial(n,p)	binom
$Poisson(\lambda)$	poisson(lam)	poisson
$Geometric(p)$	geometric(p)	geom

Table 14.1: Discrete distributions. Scipy name can be followed by .rvs(...) for a random sample, .pmf(k,...) for the PMF at k, or .cdf(k,...) for the CDF at k.

Name	Numpy (rng.*)	Scipy (sp.stats.*)
$Normal(\mu,\sigma)$	normal(mu,sigma)	norm
$Exponential(\lambda)$	exponential(1/lam)	expon

Table 14.2: Continuous distributions. PMFs are replaced by PDFs, given by .pdf(x,...). Note that in both libraries, the paramater to the exponential distribution is the scale $1/\lambda$, *not* the rate λ itself.

14.5 Exercises

1. Write Python programs to generate Geometric(p) and Negative Binomial(i,p) random variables from a uniform random sample (i.e., without using the library functions from Section 14.4).

   Solution: To generate a Geometric random variable, we run a loop of Bernoulli trials until the first success occurs. K counts the number of failures plus one success, which is equal to the total number of trials.

```
def geometric(p):
        K = 1
        while (rng.random() > p):
                K = K + 1
        return K
```

   Now, we can generate Geometric random variable i times to obtain a Negative Binomial(i, p) variable as a sum of i independent Geometric (p) random variables.

```
def negativebinomial(i,p):
        return sum([geometric(p) for j in range(i)])
```

2. (Poisson) Use the algorithm for generating discrete random variables to obtain a Poisson random variable with parameter $\lambda = 2$.

   Solution: We know a Poisson random variable takes all nonnegative integer values with probabilities

$$p_i = P(X = x_i) = e^{-\lambda}\frac{\lambda^i}{i!} \quad \text{for} \quad i = 0, 1, 2, \cdots$$

   To generate a $Poisson(\lambda)$, first we generate a random number U. Next, we divide the interval $[0, 1]$ into subintervals such that the jth subinterval has length p_j (Figure 14.2). Assume

$$X = \begin{cases} x_0 & \text{if} \quad (U < p_0) \\ x_1 & \text{if} \quad (p_0 \le U < p_0 + p_1) \\ \vdots & \\ x_j & \text{if} \quad \left(\sum_{k=0}^{j-1} p_k \le U < \sum_{k=0}^{j} p_k\right) \\ \vdots & \end{cases}$$

   Here $x_i = i - 1$, so

$$\begin{aligned} X = i \quad \text{if} \quad & p_0 + \cdots + p_{i-1} \le U < p_0 + \cdots + p_{i-1} + p_i \\ & F(i-1) \le U < F(i) \quad \text{F is CDF} \end{aligned}$$

```
from math import factorial
def poisson(lam):
        def pdf(i):
                return np.exp(-lam)*lam**i/factorial(i)
        def cdf(i):
                return sum([pdf(j) for j in range(i+1)])
        U = rng.random()
        i = 0
        while U > cdf(i):
                i = i+1
        return i
```

3. Explain how to generate a random variable with the density

$$f(x) = 2.5x\sqrt{x} \quad \text{for} \quad 0 < x < 1$$

if your random number generator produces a Standard Uniform random variable U. Hint: use the inverse transformation method.

Solution:

$$\begin{aligned} F_X(X) &= X^{\frac{5}{2}} = U \quad (0 < x < 1) \\ X &= U^{\frac{2}{5}} \end{aligned}$$

```
def X():
        U = rng.random()
        return U**(2/5)
```

We have the desired distribution.

4. Use the inverse transformation method to generate a random variable having distribution function

$$F(x) = \frac{x^2 + x}{2}, \quad 0 \le x \le 1$$

Solution:

$$\begin{aligned} \frac{X^2 + X}{2} &= U \\ (X + \frac{1}{2})^2 - \frac{1}{4} &= 2U \\ X + \frac{1}{2} &= \sqrt{2U + \frac{1}{4}} \\ X &= \sqrt{2U + \frac{1}{4}} - \frac{1}{2} \quad (X, U \in [0,1]) \end{aligned}$$

By generating a random number, U, we have the desired distribution.

```
def X():
        U = rng.random()
        return np.sqrt(2*U + 1/4) - 1/2
```

5. Let X have a standard Cauchy distribution.

$$F_X(x) = \frac{1}{\pi}\arctan(x) + \frac{1}{2}$$

Assuming you have $U \sim Uniform(0,1)$, explain how to generate X. Then, use this result to produce 1000 samples of X and compute the sample mean. Repeat the experiment 100 times. What do you observe and why?

Solution: Using Inverse Transformation Method:

$$\begin{aligned} U - \frac{1}{2} &= \frac{1}{\pi}\arctan(X) \\ \pi\left(U - \frac{1}{2}\right) &= \arctan(X) \\ X &= \tan\left(\pi(U - \frac{1}{2})\right) \end{aligned}$$

Next, here is the Python code:

```
U = rng.random([100,1000])
X = np.tan(np.pi*(U-0.5))
means = np.mean(X,axis=1)
plt.xlabel("Experiment")
plt.ylabel("Sample mean")
plt.plot(means)
plt.show()
```

Cauchy distribution has no mean (Figure 14.6), or higher moments defined.

6. (The Rejection Method) When we use the Inverse Transformation Method, we need a simple form of the cdf $F(x)$ that allows direct computation of $X = F^{-1}(U)$. When $F(x)$ doesn't have a simple form but the pdf $f(x)$ is available, random variables with density $f(x)$ can be generated by the **rejection method**. Suppose you have a method for generating a random variable having density function $g(x)$. Now, assume you want to generate a random variable having density function $f(x)$. Let c be a constant such that

$$\frac{f(y)}{g(y)} \le c \quad \text{(for all y)}$$

Show that the following method generates a random variable with density function $f(x)$.

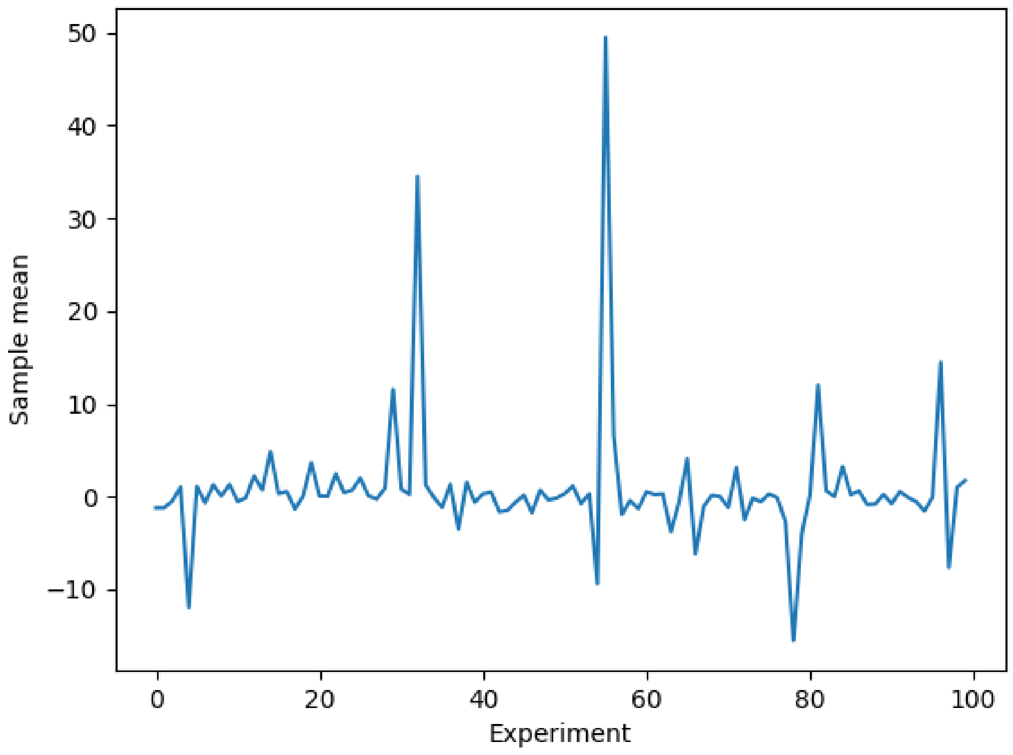


Figure 14.6: Simulation of a Cauchy distribution shows that the mean is not defined.

- Generate Y having density g.

- Generate a random number U from $Uniform(0,1)$.

- If $U \leq \frac{f(Y)}{cg(Y)}$, set $X = Y$. Otherwise, return to step 1.

Solution: The number of times N that the first two steps of the algorithm need to be called is itself a random variable and has a geometric distribution with "success" probability

$$p = P\left(U \leq \frac{f(Y)}{cg(Y)}\right)$$

Thus, $E(N) = \frac{1}{p}$. Also, we can compute p:

$$\begin{aligned}
P\left(U \leq \frac{f(Y)}{cg(Y)}|Y = y\right) &= \frac{f(y)}{cg(y)} \\
p &= \int_{-\infty}^{\infty} \frac{f(y)}{cg(y)} g(y) dy \\
&= \frac{1}{c} \int_{-\infty}^{\infty} f(y) dy \\
&= \frac{1}{c} \\
\text{Therefore,} \quad E(N) &= c
\end{aligned}$$

Let F be the desired CDF (CDF of X). Now, we must show that the conditional distribution of Y given that $U \leq \frac{f(Y)}{cg(Y)}$ is indeed F, i.e. $P(Y \leq y|U \leq \frac{f(Y)}{cg(Y)}) = F(y)$. Assume $M = \{U \leq \frac{f(Y)}{cg(Y)}\}$, $K = \{Y \leq y\}$. We know $P(M) = p = \frac{1}{c}$. Also, we can compute

$$\begin{aligned}
P(U \leq \frac{f(Y)}{cg(Y)}|Y \leq y) &= \frac{P(U \leq \frac{f(Y)}{cg(Y)}, Y \leq y)}{G(y)} \\
&= \int_{-\infty}^{y} \frac{P(U \leq \frac{f(y)}{cg(y)}|Y = v \leq y)}{G(y)} g(v) dv \\
&= \frac{1}{G(y)} \int_{-\infty}^{y} \frac{f(v)}{cg(v)} g(v) dv \\
&= \frac{1}{cG(y)} \int_{-\infty}^{y} f(v) dv \\
&= \frac{F(y)}{cG(y)}
\end{aligned}$$

Thus,

$$\begin{aligned}
P(K|M) &= P(M|K)P(K)/P(M) \\
&= P(U \leq \frac{f(Y)}{cg(Y)}|Y \leq y) \times \frac{G(y)}{\frac{1}{c}} \\
&= \frac{F(y)}{cG(y)} \times \frac{G(y)}{\frac{1}{c}} \\
&= F(y)
\end{aligned}$$

7. Use the rejection method to generate a random variable having density function $Beta(2, 4)$. Hint: Assume $g(x) = 1$ for $0 < x < 1$.

   Solution:

$$\begin{aligned}
f(x) &= 20x(1-x)^3 \quad 0 < x < 1 \\
g(x) &= 1 \quad 0 < x < 1 \\
\frac{f(x)}{g(x)} &= 20x(1-x)^3
\end{aligned}$$

We need to find the smallest constant c such that $f(x)/g(x) \leq c$. Differentiation of this quantity yields

$$\frac{d\left(\frac{f(x)}{g(x)}\right)}{dx} = 0$$

$$\text{Thus,} \quad x = \frac{1}{4}$$

$$\text{Therefore,} \quad \frac{f(x)}{g(x)} \leq \frac{135}{64}$$

$$\text{Hence,} \quad \frac{f(x)}{cg(x)} = \frac{256}{27}x(1-x)^3$$

```
def X():
        Y = rng.random()
        U = rng.random()
        while U > 256/27*Y*(1-Y)**3:
                Y = rng.random()
                U = rng.random()
        return Y
```

8. Use the rejection method to generate a random variable having the $Gamma(\frac{5}{2}, 1)$ density function. Hint: Assume $g(x)$ is the pdf of the $Gamma\left(\alpha = \frac{5}{2}, \lambda = 1\right)$.

   Solution:

$$f(x) = \frac{4}{3\sqrt{\pi}}x^{\frac{3}{2}}e^{-x}, x > 0$$

$$g(x) = \frac{2}{5}e^{-\frac{2x}{5}} \quad x > 0$$

$$\frac{f(x)}{g(x)} = \frac{10}{3\sqrt{\pi}}x^{\frac{3}{2}}e^{-\frac{3x}{5}}$$

$$\frac{d\left(\frac{f(x)}{g(x)}\right)}{dx} = 0$$

$$\text{Hence,} \quad x = \frac{5}{2}$$

$$c = \frac{10}{3\sqrt{\pi}}\left(\frac{5}{2}\right)^{\frac{3}{2}} e^{\frac{-3}{2}}$$

$$\frac{f(x)}{cg(x)} = \frac{x^{\frac{3}{2}}e^{\frac{-3x}{5}}}{\left(\frac{5}{2}\right)^{\frac{3}{2}}e^{\frac{-3}{2}}}$$

We know how to generate an Exponential random variable.

- Generate a random number U_1 and set $Y = -\frac{5}{2}\log U_1$.
- Generate a random number U_2.

- If $U_2 < \frac{Y^{\frac{3}{2}}e^{\frac{-3Y}{5}}}{\left(\frac{5}{2}\right)^{\frac{3}{2}}e^{\frac{-3}{2}}}$, set $X = Y$. Otherwise, execute the step 1.

9. Use the rejection method to generate a standard normal random variable. Hint: Assume $g(x)$ is the pdf of the exponential distribution with $\lambda = 1$.

   Solution:

$$
\begin{aligned}
f(x) &= \frac{2}{\sqrt{2\pi}}e^{-\frac{x^2}{2}} \quad 0 < x < \infty \\
g(x) &= e^{-x} \quad 0 < x < \infty \quad \text{(Exponential density function with mean 1)} \\
\text{Thus,} \quad \frac{f(x)}{g(x)} &= \sqrt{\frac{2}{\pi}}e^{x-\frac{x^2}{2}} \\
\text{Thus,} \quad x = 1 \quad &\text{maximizes} \quad \frac{f(x)}{g(x)} \\
\text{Thus,} \quad c &= \sqrt{\frac{2e}{\pi}} \\
\frac{f(x)}{cg(x)} &= e^{-\frac{(x-1)^2}{2}}
\end{aligned}
$$

- Generate Y, an exponential random variable with mean 1.
- Generate a random number U.
- If $U \le e^{\frac{-(Y-1)^2}{2}}$ set $X = Y$. Otherwise, return to step 1.

10. Use the rejection method to generate a *Gamma*$(2, 1)$ random variable conditional on its value being greater than 5, that is

$$
\begin{aligned}
f(x) &= \frac{xe^{-x}}{\int_5^\infty xe^{-x}dx} \\
&= \frac{xe^{-x}}{6e^{-5}} \quad (x \ge 5)
\end{aligned}
$$

Hint: Assume $g(x)$ be the density function of exponential distribution.

Solution: Since a *Gamma*$(2, 1)$ random variable has expected value 2, we use an exponential distribution with mean 2 that is conditioned to be greater than 5.

$$
\begin{aligned}
f(x) &= \frac{xe^{(-x)}}{\int_5^\infty xe^{(-x)}dx} \\
&= \frac{xe^{(-x)}}{6e^{(-5)}} \quad x \ge 5 \\
g(x) &= \frac{\frac{1}{2}e^{(-\frac{x}{2})}}{e^{\frac{-5}{2}}} \quad x \ge 5 \\
\frac{f(x)}{g(x)} &= \frac{x}{3}e^{-(\frac{x-5}{2})}
\end{aligned}
$$

We obtain the maximum in $x = 5$ since $\frac{f(x)}{g(x)}$ is decreasing. Therefore,

$$c = \frac{f(5)}{g(5)} = \frac{5}{3}$$

- Generate a random number V.
- $Y = 5 - 2\log(V)$.
- Generate a random number U.
- If $U < \frac{Y}{5} e^{-(\frac{Y-5}{2})}$, set $X = Y$; otherwise return to step 1.

Bibliography

[1] `http://projecteuclid.org/download/pdf_1/euclid.aoms/1177706645`

[2] `https://numpy.org/doc/stable/reference/random/generator.html#distributions`

[3] `https://docs.scipy.org/doc/scipy/reference/stats.html#probability-distributions`

[4] Michael Baron, *Probability and Statistics for Computer Scientists.* CRC Press, 2006

[5] Sheldon M. Ross, *Simulation.* Academic Press, 2012