RANDOM VARIABLE GENERATION

Elif Yılmaz

Boğaziçi University

Overview

1. Introduction
   - Uniform Simulation
   - Inverse Transform
2. General Transformation Methods
3. Accept-Reject Methods
   - Fundamental Theorem of Simulation
   - Accept-Reject Algorithm and Examples
4. Envelope Accept-Reject Methods
   - Squeeze Principle

Uniform Simulation

The problem associated with the generation of "random numbers" is producing a deterministic sequence of values in [0,1] which imitates a sequence of iid uniform random variables $\mathcal{U}_{[0,1]}$.

For our purposes, there are methods that use a fully deterministic process to produce a random sequence in the following sense:

- Having generated $(X_1, ..., X_n)$, knowledge of X_n, [or of$(X_1, ..., X_n)$] imparts no discernible knowledge of the value of X_{n+1}, if the transformation function is not available. By given the initial value X_0 and the transformation function, the sample $(X_1, ..., X_n)$ is always the same.

Uniform Simulation

The "pseudo-randomness" produced by these techniques is limited since two samples $(X_1, ..., X_n)$ *and* $(Y_1, ..., Y_n)$ produced by the algorithm will not be independent, nor identically distributed, nor comparable in any probabilistic sense.

Limitation: The validity of a random number generator is based on a single sample $X_1, ..., X_n$ when n tends to $+\infty$ and not on replications $(X_{11}, ..., X_{1n}), (X_{21}, ..., X_{2n}), ..., (X_{k1}, ..., X_{kn})$ where n is fixed and k tends to infinity. In fact, the distribution of these n-tuples depends only on the manner in which the initial values $X_{r1}, (1 \leq r \leq k)$ were generated.

Pseudo-random number generator

Definition

A uniform pseudo-random number generator is an algorithm which, starting from an initial value u_0 and a transformation D, produces sequence $(u_i) = (D^i(u_0))$ of values in [0,1]. For all n, the values $(u_1, ..., u_n)$ reproduce the behavior of an iid sample $(V_1, ..., V_n)$ of uniform random variables when compared through a usual set of tests.

We can use this definition for only testable aspects of the random variable generation, which are connected through the deterministic transformation $u_i = D(u_{i-1})$. So, the algorithm is valid for the verification that the sequence $U_1, U_2, ..., U_n$ leads to acceptance of the hypothesis $H_0 : U_1, U_2, ..., U_n$ are iid $\mathcal{U}_{[0,1]}$.

Definition is **functional**: An algorithm that generates uniform numbers is acceptable if it is not rejected by a set of tests.

Examples of Tests

- "Kolmogorov-Smirnov test" is one of the tests of uniformity.
- "Methods of Time Series" is to determine the degree of correlation between between U_i *and* $(U_{i-1}, ..., U_{i-k})$.
- "Die Hard" is to measure the quality of random number generator.

Inverse Transform

Definition

For a non-decreasing function F on $\mathbb{R}$, the generalized inverse of F, F^-, is the function defined by

$$F^-(u) = inf\{z : F(x) \geq u.\}$$

Lemma

If $U \sim \mathcal{U}_{[0,1]}$, then the random variable $F^-(U)$ has the distribution F.

Lemma, sometimes known as the *probability integral transform*, gives us a representation of any random variable as a transform of a uniform random variable.

Inverse Transform

Proof of Lemma

For all u $\in [0,1]$ and for all z $\in F^-([0,1])$, the generalized inverse satisfies

$$F(F^-(u)) \geq u \text{ and } F^-(F(x)) \leq x.$$

Then,

$$\{(u,x) : F^-(u) \leq x\} = \{(u,x) : F(x) \geq u\}$$

and

$$\mathbb{P}(F^-(U) \leq x) = \mathbb{P}(U \leq F(x)) = F(x).$$

Therefore, it suffices to generate U according to $\mathcal{U}_{[0,1]}$ and then make the transformation $x = F^-(u)$ to generate a random variable $X \sim F$

General Transformation Methods

When a distribution f is linked in a relatively simple way to another distribution that is easy to simulate, this relationship can often be exploited to construct an algorithm to simulate variables from f.

By using python, we can simulate random variables. Let's observe the examples (Normal variable generation, Exponential generation, Poisson generation, Beta generation, Gamma generation and Student's t generation) in the python. For these random variable generations, the starting point is from uniform distribution.

General Transformation Methods

Example

Normal random variable generation has a special algorithm, called Box-Muller Algorithm. It produces two normal random variables from two uniform random variables. Box-Muller Algorithm:

- Generate U_1, U_2 iid $\mathcal{U}_{[0,1]}$
- Define
  $x_1 = \sqrt{-2log(u_1)}cos(2\pi u_2)$
  $x_2 = \sqrt{-2log(u_1)}sin(2\pi u_2)$
- Take x_1 and x_2 as two independent draws from $N(0, 1)$.

Accept-Reject Methods

There are many distributions from which it is difficult, or even impossible, to directly simulate by an inverse transform. In such settings, it is impossible to exploit direct probabilistic properties to derive a simulation method. Thus, we turn to another class of methods that only requires us to know the functional form of the density f of interest.

The key to this method is to use a simpler (simulationwise) density g from which the simulation is actually done. For a given density g -**the instrumental density**-, there are many densities f -**the target densities**- which can be simulated this way.

The Accept-Reject algorithm is based on a simple connection with the uniform distribution.

Fundamental Theorem of Simulation

Theorem (Fundamental Theorem of Simulation)

Simulating $X \sim f(x)$ is equivalent to simulating

$$(X, U) \sim \mathcal{U}\{(x, u) : 0 < u < f(x)\}$$

Proof: One thing that is made clear by this theorem is that we can generate X in three ways:

- first generate $X \sim f(x)$, and then $U \mid X = x$ but this is useless because we already have X and we do not need U.
- first generate U, and then $X \mid U = u$, that is just accept-reject algorithm.
- generate (X,U) jointly, which will eventually turn out be the smart idea because it allows us to generate data on a larger set where simulation is easier and then to use the pair if the constraint is satisfied.

Fundamental Theorem of Simulation

For a simple case:
In a one-dimensional setting, suppose that

$$\int_a^b f(x)dx = 1$$

and that f is bounded by m.
Say X has support $[A, B]$ and $f(x) < M$ for all x. So, we can generate the random pair $(Y, U) \sim \mathcal{U}\{(x, u) : 0 < u < M\}$ by simulating $Y \sim \mathcal{U}(a, b)$ and $U \mid Y = y \sim \mathcal{U}(0, m)$ and take the pair iff $0 < u < f(y)$.

This works because

$$\mathbb{P}(X \leq x) = \mathbb{P}(Y \leq x \mid U < f(Y)) = \frac{\mathbb{P}(Y \leq x, U < f(Y))}{\mathbb{P}(U < f(Y))}$$

$$= \frac{\int_a^x \int_0^{f(y)} dudy}{\int_a^b \int_0^{f(y)} dudy} = \frac{\int_a^x f(y)dy}{\int_a^b f(y)dy} = \int_a^x f(y)dy$$

Accept-Reject Algorithm

Accept-Reject Algorithm

1. Generate $X \sim g, U \sim \mathcal{U}_{[0,1]}$
2. If $U \leq \frac{f(Y)}{Mg(Y)}$, then set $X = Y$ ("accept") ; otherwise go back to 1 ("reject").

Before we prove the algorithm and give examples, note that:

- The ratio is bounded between 0 and 1; $0 < \frac{f(Y)}{Mg(Y)} \leq 1$.
- The number of times N that steps 1 and 2 need to be called (e.g., the number of iterations needed to successfully generate X) is itself a rv and has a geometric distribution with "success" probability $p = \mathbb{P}(U \leq \frac{f(Y)}{Mg(Y)})$ and $\mathbb{P}(N = n) = (1-p)^{n-1}p, n \geq 1$. Thus, average number of iterations required is given by $\mathbb{E}[N] = \frac{1}{p}$.

Accept-Reject Algorithm

$\mathbb{P}(U \leq \frac{f(Y)}{Mg(Y)} \mid Y = y) = \frac{f(y)}{Mg(y)}$ and Y has density g(y); therefore, we get

$$p = \int_{-\infty}^{\infty} \frac{f(y)}{Mg(y)} g(y)dy = \frac{1}{M} \int_{-\infty}^{\infty} f(y)dy = \frac{1}{M}$$

since f is density function. Thus $\mathbb{E}[N] = M$, the bounding constant, and we can now indeed see that it is desirable to choose our alternative density g so as to minimize this constant $M = sup_x\{f(x)/g(x)\}$. Of course, the optimal function would be g(x) = f(x) which is not what we have in mind since *the whole point is to choose a different (easy to simulate) alternative from f.*

Accept-Reject Algorithm

Proof of the algorithm

We must show that $\mathbb{P}(Y \leq y \mid U \leq \frac{f(Y)}{Mg(Y)}) = F(y)$. Let $A = \{Y \leq y\}$ and $B = \{U \leq \frac{f(Y)}{Mg(Y)}\}$. Recall that $\mathbb{P}(B) = p = \frac{1}{M}$ and by using $\mathbb{P}(A \mid B) = \mathbb{P}(b \mid a)\frac{\mathbb{P}(A)}{\mathbb{P}(B)}$, we get $\mathbb{P}(Y \leq y \mid U \leq \frac{f(Y)}{Mg(Y)})\frac{G(y)}{1/M} = \frac{F(y)}{MG(y)}\frac{G(y)}{1/M} = F(y)$ where

$$\mathbb{P}(Y \leq y \mid U \leq \frac{f(Y)}{Mg(Y)}) = \frac{U \leq \frac{f(Y)}{Mg(Y)}, Y \leq y}{G(y)}$$

$$= \int_{-\infty}^{y} \frac{U \leq \frac{f(y)}{Mg(y)} \mid Y = w \leq y}{G(y)} g(w)dw = \frac{1}{G(y)} \int_{-\infty}^{y} \frac{f(w)}{Mg(w)} g(w)dw$$

$$= \frac{1}{MG(y)} \int_{-\infty}^{y} f(w)dw = \frac{F(y)}{MG(y)}$$

Envelope Accept-Reject Methods

Squeeze Principle Lemma

If there exist a density g_m, a function g_n, and a constant M such that

$$g_n(x) \leq f(x) \leq Mg_m(x)$$

Envelope Accept-Reject Algorithm

Then, the envelope accept-reject algorithm:

1. Generate $X \sim g_m(x), U \sim \mathcal{U}_{[0,1]}$
2. Accept X if $U \leq \frac{g_n(X)}{Mg_m(X)}$.
3. Otherwise, accept X if $U \leq \frac{f(X)}{Mg_m(X)}$

produces random variables that are distributed according to f.

References

- Robert, C., Casella, G. (2013). *Monte Carlo statistical methods* . Springer Science Business Media, 35-70.
- Brownlee, J. (2018). *Statistical methods for machine learning: Discover how to transform data into knowledge with Python*.
- Marsaglia, G. (1984). The exact-approximation method for generating random variables in a computer. *Journal of the American Statistical Association* , 79(385), 218-221.
- Howell, L. W., Rheinfurth, M. H. (1984). Generation of pseudo-random numbers. *The Journal of the Acoustical Society of America* , 75(2), 639-639.
- Mehta, H. K. (2015). *Mastering Python scientific computing* . Packt Publishing Ltd.
- Devroye, L. (1986, December). Sample-based non-uniform random variate generation. *In Proceedings of the 18th conference on Winter simulation*, 260-265.
- Knuth, D. E. (1997). *The art of computer programming (Vol. 3)*

The End