YOUNG-GEUN statistics

Statistical Computing

R Lab

O RLY?

Young-geun Kim

R Lab for Statistical Computing

Young-geun Kim
Department of Statistics, SKKU
dudrms33@g.skku.edu

2019-03-31

Contents

Statistical Computing

Statistical computing mainly treats *random generation* methods. Additionaly, it treats useful simulation methods.

Chapter 1

Methods for Generating Random Variables

1.1 Introduction

Most of the methods so-called *computational statistics* requires generation of random variables from specified probability distribution. In hand, we can spin wheels, roll a dice, or shuffle cards. The results are chosen randomly. However, we want the same things with computer. Here, `r`. As we know, computer cannot generate complete uniform random numbers. Instead, we generate **pseudo-random** numbers.

1.2 Pseudo-random Numbers

Definition 1.1 (Pseudo-random numbers)**.** Sequence of values generated deterministically which have all the appearances of being independent $unif(0,1)$ random variables, i.e.

$$x_1, x_2, \ldots, x_n \stackrel{iid}{\sim} unif(0,1)$$

- behave *as if* following $unif(0,1)$
- typically generated from an *initial seed*

1.2.1 Linear congruential generator

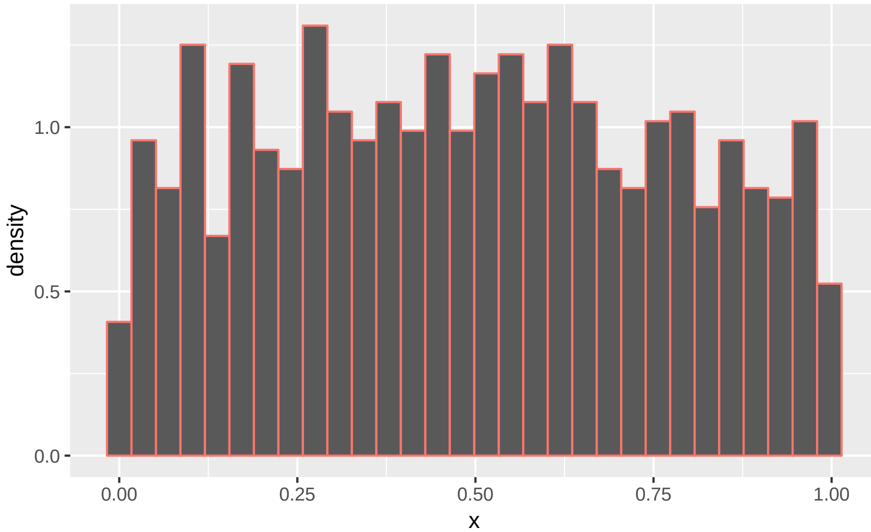
Let $x_0, x_1, \ldots \in \mathbb{Z}_+$.

1. Set x_0 as initial seed.
2. Generate $x_n, n = 1, 2, \ldots$ recursively:
   a. $x_n = (ax_{n-1} + c) \mod m$
   b. where $a, c \in \mathbb{Z}_+, m$: modulus
3. Compute $u_n = \frac{x_n}{m} \in (0,1)$

Then $u_1, u_2, \ldots \sim unif(0,1)$

```r
lcg <- function(n, seed, a, b, m) {
  x <- rep(seed, n + 1)
  for (i in 1:n) {
    x[i + 1] <- (a * x[i] + b) %% m
  }
  x[-1] / m
}
```

```
tibble(
  x = lcg(1000, 0, 1664525, 1013904223, 2^32)
) %>%
  ggplot(aes(x = x)) +
  geom_histogram(aes(y = ..density..), bins = 30, col = gg_hcl(1))
```



1.2.2 Multiplicative congruential generator

1.2.3 Sampling from a finite population

From finite population, we can sample data with or without replacement.

```
sample(0:1, size = 10, replace = TRUE)
```

```
 [1] 1 0 0 1 0 1 1 0 1 1
```

```
sample(1:100, size = 6, replace = FALSE)
```

```
[1] 61 83 50 74 34 35
```

1.3 The Inverse Transform Method

Theorem 1.1 (Probability Integral Transformation)**.** *If X is a continuous random variable with cdf $F_(x)$, then*

$$U \equiv F_X(X) \sim unif(0,1)$$

Probability Integral Transformation. Let $U \sim unif(0,1)$. Then

$$\begin{aligned}
P(F_X^{-1}(U) \le x) &= P(\inf\{t : F_X(t) = U\} \le x) \\
&= P(U \le F_X(x)) \\
&= F_U(F_X(x)) \\
&= F_X(x)
\end{aligned}$$

□

Thus, to generate n random variables $\sim F_X$,

1. form of $F_X^{-1}(u)$
2. For each $i = 1, 2, \ldots, n$:
    a. Generate $u_i \sim unif(0, 1)$
    b. $x_i = F_X^{-1}(u_i)$

Collect $x_1, x_2, \ldots, x_n \sim F_X$.

1.3.1 Continuous case

Denote that the *probability integral transformation* holds for a continuous variable. When generating continuous random variable, applying above algorithm might work.

Example 1.1 (Exponential distribution)**.** If $X \sim Exp(\lambda)$, then $F_X(x) = 1 - e^{-\lambda x}$. We can derive the inverse function of cdf

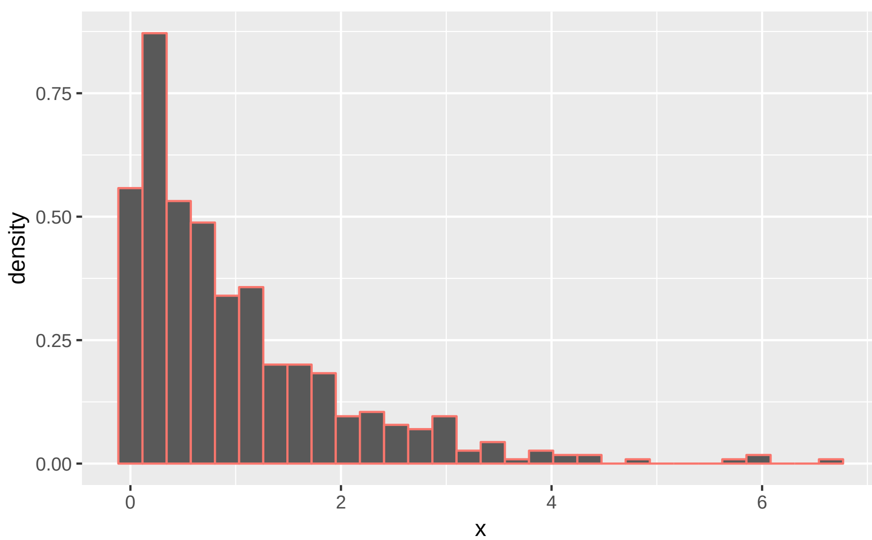
$$F_X^{-1}(u) = \frac{1}{\lambda} \ln(1 - u)$$

From above example 1.1, we just type the inverse cdf in the function to use the method.

```
inv_exp <- function(n, lambda) {
  -log(runif(n)) / lambda
}
```

If we generate $x_1, \ldots, x_{500} \sim Exp(\lambda = 1)$,

```
tibble(x = inv_exp(500, lambda = 1)) %>%
  ggplot(aes(x = x)) +
  geom_histogram(aes(y = ..density..), bins = 30, col = gg_hcl(1))
```



1.3.2 Discrete case

Chapter 2

Monte Carlo Integration and Variance Reduction