

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

2019-04-02

Contents

Welcome	3
Statistical Computing	3
1 Methods for Generating Random Variables	5
1.1 Introduction	5
1.2 Pseudo-random Numbers	5
1.3 The Inverse Transform Method	7
1.4 The Acceptance-Rejection Method	12
1.5 Transformation Methods	18
1.6 Sums and Mixtures	18
1.7 Multivariate Normal Random Vector	19
1.8 Stochastic Processes	19
2 Monte Carlo Integration and Variance Reduction	20

Welcome

Statistical computing mainly treats useful simulation methods.

Statistical Computing

We first look at *random generation* methods. Lots of simulation methods are built based on this random numbers.

Sampling from a finite population

Generating random numbers is like sampling. From finite population, we can sample data with or without replacement. For example of sampling with replacement, we toss coins 10 times.

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

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

Sampling without replacement: Choose some lottery numbers which consist of 1 to 100.

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

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

Random generators of common probability distributions

R provides some functions which generate random numbers following famous distributions. Although we will learn some skills generating these numbers in basis levels, these functions do the same thing more elegantly.

```
gg_curve(dbeta, from = 0, to = 1, args = list(shape1 = 3, shape2 = 2)) +  
  geom_histogram(  
    data = tibble(  
      rand = rbeta(1000, 3, 2),  
      idx = seq(0, 1, length.out = 1000)  
    ),  
    aes(x = rand, y = ..density..),  
    position = "identity",  
    bins = 30,  
    alpha = .45,  
    fill = gg_hcl(1)  
  )
```

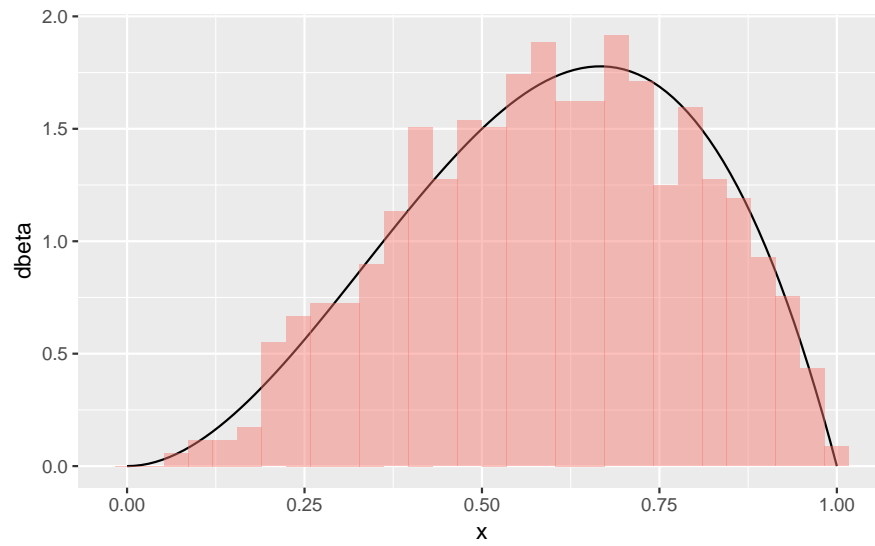


Figure 1: Beta(3,2) random numbers

Figure 1 shows that `rbeta()` function generate random numbers very well. Histogram is of the random number, and the curve is the true beta distribution.

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.

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

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

1.2.1 Linear congruential generator

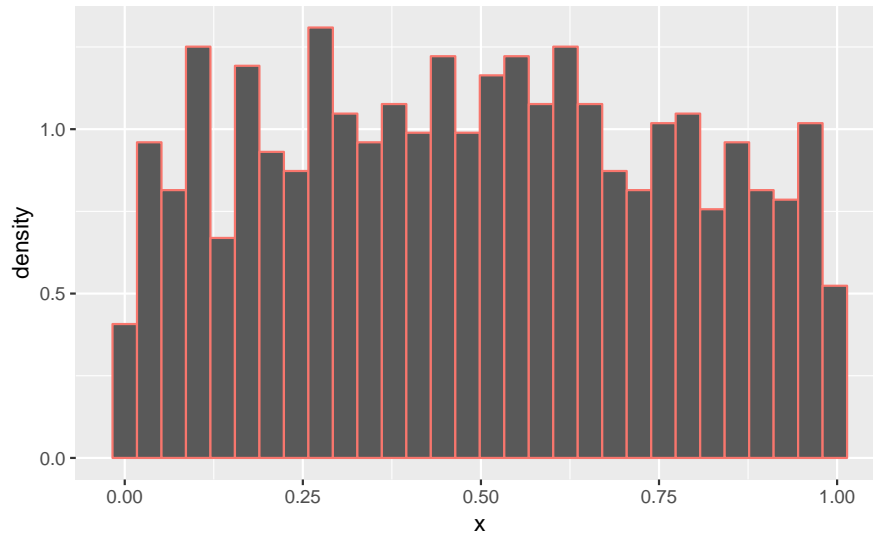
Let $x_0, x_1, \dots \in \mathbb{Z}_+$.

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

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

```
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

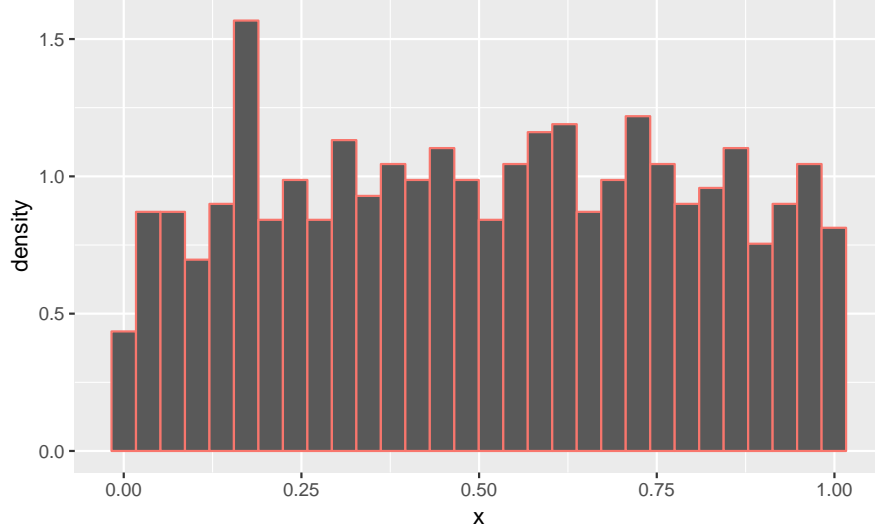
As we can expect from its name, this is congruential generator with $c = 0$.

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

Then $u_1, u_2, \dots \sim \text{unif}(0, 1)$

We just set $b = 0$ in our `lcg()` function. The **seed must not be zero**.

```
tibble(
  x = lcg(1000, 5, 1664525, 0, 2^32)
) %>%
  ggplot(aes(x = x)) +
  geom_histogram(aes(y = ..density..), bins = 30, col = gg_hcl(1))
```

1.2.3 Cycle

Generate LCG $n = 32$ with $a = 1$, $c = 1$, and $m = 16$ from the seed $x_0 = 0$.

```
lcg(32, 0, 1, 1, 16)
```

```
[1] 0.0625 0.1250 0.1875 0.2500 0.3125 0.3750 0.4375 0.5000 0.5625 0.6250
[11] 0.6875 0.7500 0.8125 0.8750 0.9375 0.0000 0.0625 0.1250 0.1875 0.2500
[21] 0.3125 0.3750 0.4375 0.5000 0.5625 0.6250 0.6875 0.7500 0.8125 0.8750
[31] 0.9375 0.0000
```

Observe that we have the cycle after m -th number. Against this problem, we give different seed from every $(im + 1)$ th random number.

1.3 The Inverse Transform Method

Definition 1.2 (Inverse of CDF). Since some cdf F_X is not strictly increasing, we define $F_X^{-1}(y)$ for $0 < y < 1$ by

$$F_X^{-1}(y) := \inf\{x : F_X(x) \geq y\}$$

Using this definition, we can get the following theorem.

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

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

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

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

□

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

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

Collect $x_1, x_2, \dots, x_n \stackrel{iid}{\sim} F_X$.

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

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

Note that

$$U \sim \text{unif}(0, 1) \Leftrightarrow 1 - U \sim \text{unif}(0, 1)$$

Then we just can use U instead of $1 - U$.

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

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

```
gg_curve(dexp, from = 0, to = 10) +
  geom_histogram(
    data = tibble(x = inv_exp(500, lambda = 1)),
    aes(x = x, y = ..density..),
    bins = 30,
    fill = gg_hcl(1),
    alpha = .5
  )
```

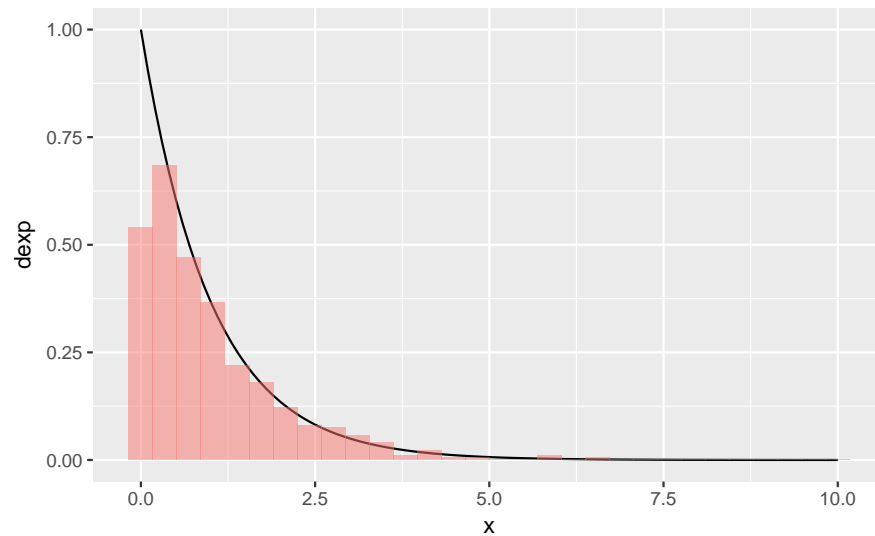


Figure 1.1: Inverse Transformation: Exp(1)

1.3.2 Discrete case

1. For each $i = 1, 2, \dots, n$:
 - a. Generate $u_i \sim \text{unif}(0, 1)$
 - b. Take x_i s.t. $F_X(x_{i-1}) < U \leq F_X(x_i)$

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

```
pmf <-
  tibble(
    x = 0:4,
    p = c(.1, .2, .2, .2, .3)
  )
```

Table 1.1: Example of a Discrete Random Variable

x	0.0	1.0	2.0	3.0	4.0
p	0.1	0.2	0.2	0.2	0.3

Example 1.2 (Discrete Random Variable). Consider a discrete random variable X with a mass function as in Table 1.1.

i.e.

```
pmf %>%
  ggplot() +
  geom_segment(aes(x = x, y = 0, xend = x, yend = p)) +
  ylab(expression(p(x)))
```

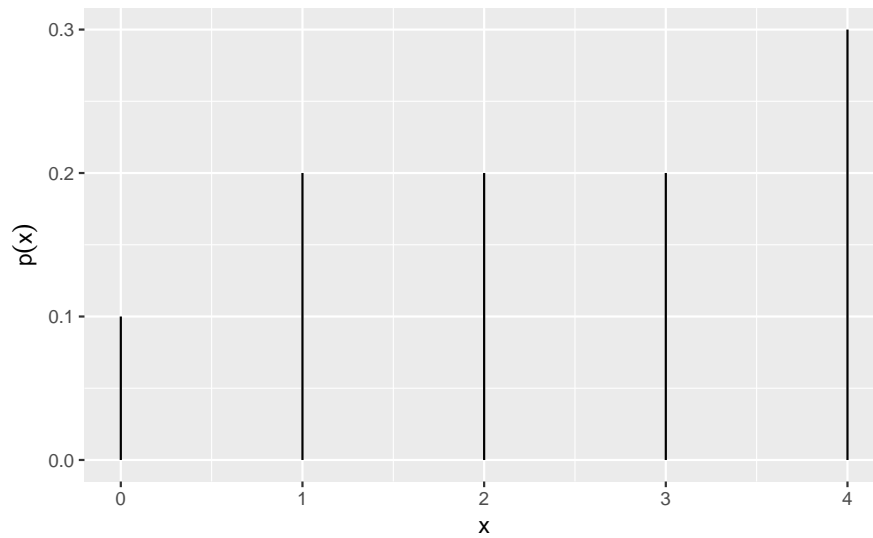


Figure 1.2: Probability Mass Function

Then we have the cdf

```
pmf %>%
  mutate(
    fx = cumsum(p),
    x_end = lead(x, default = 5),
    u = fx,
    u = ifelse(u == .5, .6, u),
    fx1 = lead(fx, default = 1),
    rand = u > fx & u <= fx1
  ) %>%
  ggplot() +
  geom_segment(aes(x = x, y = fx, xend = x_end, yend = fx)) +
  ylab(expression(F(x))) +
  geom_segment(
    aes(x = 0, y = u, xend = x_end, yend = u, colour = rand),
    linetype = "dashed",
    arrow = arrow(length = unit(.5, "cm")),
    show.legend = FALSE
  ) +
  geom_segment(
    aes(x = x_end, y = u, xend = x_end, yend = 0, colour = rand),
    linetype = "dashed",
    arrow = arrow(length = unit(.5, "cm")),
    show.legend = FALSE
  ) +
  scale_colour_manual(
    values = c("TRUE" = gg_hcl(1), "FALSE" = "#00000000")
  )
```

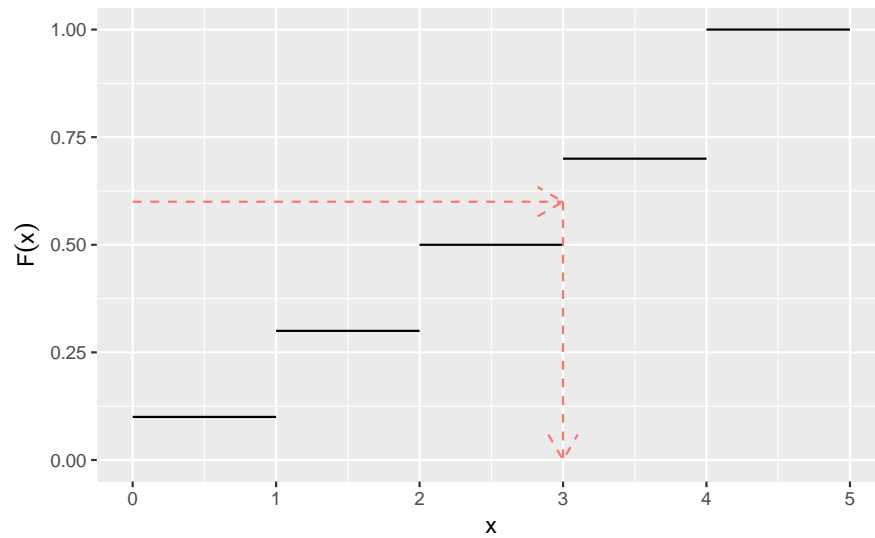


Figure 1.3: CDF of the Discrete Random Variable: Illustration for discrete case

Remembering the algorithm, we can implement `dplyr::case_when()` here.

```
rcustom <- function(n) {
  tibble(u = runif(n)) %>%
    mutate(
      x = case_when(
        u > 0 & u <= .1 ~ 0,
        u > .1 & u <= .3 ~ 1,
        u > .3 & u <= .5 ~ 2,
        u > .5 & u <= .7 ~ 3,
        TRUE ~ 4
      )
    ) %>%
    select(x) %>%
    pull()
}
```

```
tibble(
  x = rcustom(100)
) %>%
  ggplot(aes(x = x)) +
  geom_histogram(aes(y = ..density..), binwidth = .1)
```

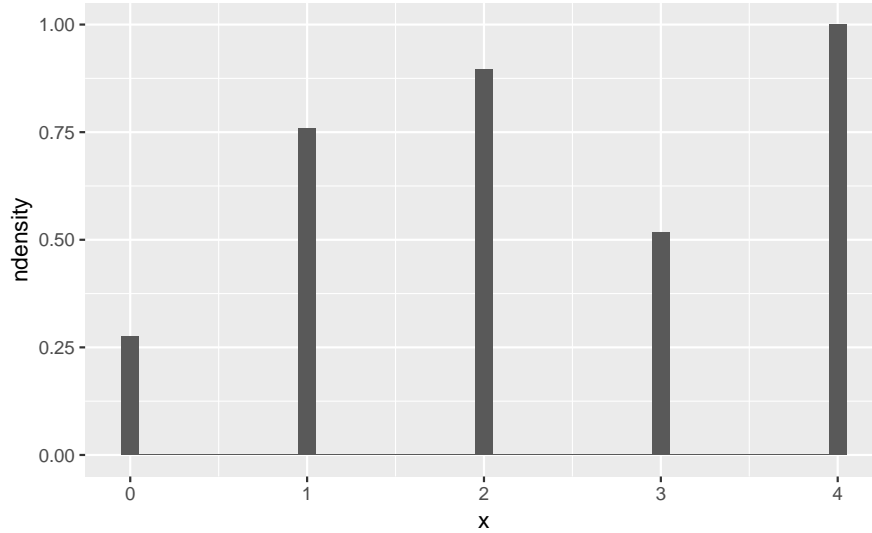


Figure 1.4: Generated discrete random numbers

See Figure 1.2 and 1.4. Comparing the two, the result can be said okay.

1.3.3 Problems with inverse transformation

Examples 1.1 and 1.2. We could generate these random numbers because we aware of

1. analytical F_X
2. F^{-1}

In practice, however, not all distribution have analytical F . Numerical computing might be possible, but it is not efficient. There are other approaches.

1.4 The Acceptance-Rejection Method

Acceptance-rejection method does not require analytical form of cdf. What we need is our *target* density (or mass) function and *proposal* density (or mass) function. Target function is what we want to generate. Proposal function is of any random variable that is *easy to generate random numbers*. From this approach, we can generate any distribution while computation is not efficient.

pdf or pmf	target or proposal
f	target
g	proposal - easy to generate random numbers

First of all, g should satisfy that

$$\text{spt} f \subseteq \text{spt} g$$

Next, for some (pre-specified) $c > 0$

$$\forall x \in \text{spt} f : \frac{f(x)}{g(x)} \leq c$$

1.4.1 A-R algorithm

For $i = 1, \dots, n$

1. $Y \sim g(Y)$
2. $U \sim \text{unif}(0, 1) \perp\!\!\!\perp Y$
3. Accept-Reject step
 - a. Accept: $U \leq \frac{f(Y)}{cg(Y)} \Rightarrow x_i = Y$
 - b. Reject: otherwise, go to step 1

Collect $x_1, x_2, \dots, x_n \stackrel{iid}{\sim} f(x)$.

1.4.2 Efficiency

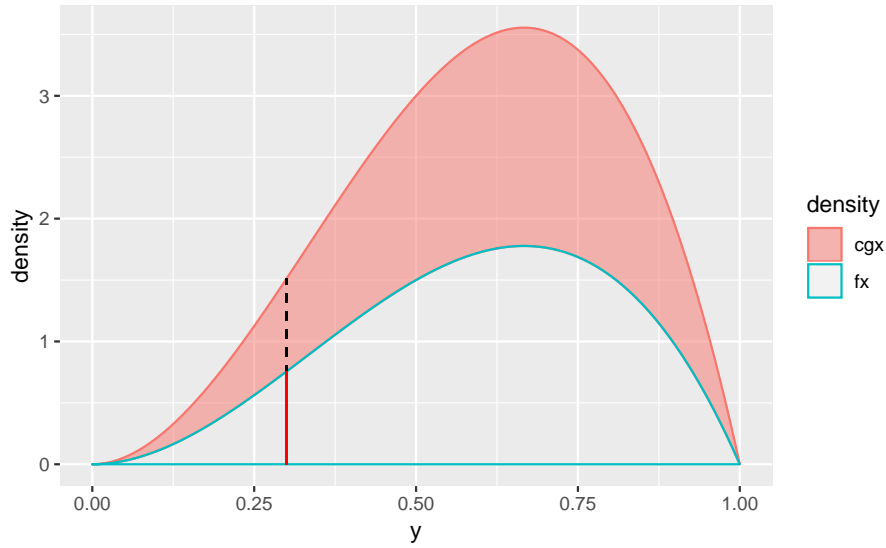


Figure 1.5: Property of AR method

See Figure 1.5. This illustrates the motivation of A-R method. Lower one is $f(x)$ and the upper one is $cg(x)$ which covers f . We can see that

$$0 < \frac{f(x)}{cg(x)} \leq 1$$

The algorithm takes random number from $Y \sim g$ in each recursive step i , which is represented as a line in the figure. At this value, the algorithm accept Y as random number of f if

$$U \leq \frac{f(Y)}{cg(Y)}$$

Suppose that we choose a point at random on a line drawn in the figure 1.5. If we get the red line, we accept. Otherwise, we reject. In other words, the *colored area is where we reject the given value*. The smaller the area is, the more efficient the algorithm will be.

Proposition 1.1 (Properties of A-R Method). (1) $\frac{f(Y)}{cg(Y)} \perp\!\!\!\perp U$

(2) $0 < \frac{f(x)}{cg(x)} \leq 1$

(3) Let N be the number of iterations needed to get an acceptance. Then

$$N \sim \text{Geo}(p) \quad \text{where } p \equiv P\left(U \leq \frac{f(Y)}{cg(Y)}\right)$$

and so

$$\begin{cases} P(N = n) = p(1-p)^{n-1} I_{\{1,2,\dots\}}(n) \\ E(N) = \text{average number of iterations} = \frac{1}{p} \end{cases}$$

$$(4) X \sim Y \mid U \leq \frac{f(Y)}{cg(Y)}, \text{ i.e.}$$

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

Remark (Efficiency). Efficiency of the A-R method depends on $p = P\left(U \leq \frac{f(Y)}{cg(Y)}\right)$. In fact,

$$E(N) = \frac{1}{p} = c$$

The algorithm becomes efficient for small c .

Proof. Note that

$$P\left(U \leq \frac{f(y)}{cg(y)}, Y = y\right) = P\left(Y \leq \frac{g(y)}{cg(y)} \mid Y = y\right) P(Y = y)$$

$$\text{Since } U \sim \text{unif}(0, 1), P\left(Y \leq \frac{g(y)}{cg(y)} \mid Y = y\right) = \frac{f(y)}{cg(y)}.$$

By construction, $P(Y = y) = g(y)$.

It follows that

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

Hence,

$$E(N) = \frac{1}{p} = c$$

We can say that the method is efficient when the acceptance rate p is large, i.e. c small. □

Corollary 1.1 (Efficiency of A-R Method). *A-R method is efficient when*

$g(\cdot)$ is close to $f(\cdot)$ and

have small c .

Corollary 1.2 (Choosing c). *To enhance the algorithm, we might choose c which satisfy*

$$c = \max \left\{ \frac{f(x)}{g(x)} : x \in \text{spt} f \right\}$$

1.4.3 Examples

Example 1.3 (Beta(a,b)). Let $X \sim \text{Beta}(a, b)$. Then the pdf of X is given by

$$f(x) = \frac{1}{B(a, b)} x^{a-1} (1-x)^{b-1} I_{(0,1)}(x)$$

Solution (Generating Beta(a,b) with A-R method). Consider proposal density $g(x) = I_{(0,1)}(x)$, i.e. $\text{unif}(0, 1)$.

To determine the optimal c s.t.

$$c = \max \left\{ \frac{f(x)}{g(x)} : x \in (0, 1) \right\}$$

find the maximum of

$$\frac{f(x)}{g(x)} = \frac{1}{B(a, b)} x^{a-1} (1-x)^{b-1}$$

Solve

$$\begin{aligned} \frac{d}{dx} \left(\frac{f(x)}{g(x)} \right) &= \frac{1}{B(a, b)} \left((a-1)x^{a-2}(1-x)^{b-1} - (b-1)x^{a-1}(1-x)^{b-2} \right) \\ &= \frac{x^{a-2}(1-x)^{b-2}}{B(a, b)} \left((a-1)(1-x) - (b-1)x \right) \\ &= \frac{x^{a-2}(1-x)^{b-2}}{B(a, b)} (a-1 - (a+b-2)x) = 0 \end{aligned}$$

It follows that

$$\frac{f(x)}{g(x)} \leq \frac{f\left(\frac{a-1}{a+b-2}\right)}{g\left(\frac{a-1}{a+b-2}\right)} = c$$

if $\frac{a-1}{a+b-2} \neq 0, 1$

```
ar_beta <- function(n, a, b) {
  opt_x <- (a - 1) / (a + b - 2)
  opt_c <- dbeta(opt_x, shape1 = a, shape2 = b) / dunif(opt_x)
  X <- NULL
  N <- 0
  while (N <= n) {
    Y <- runif(n)
    U <- runif(n)
    X <- c(X, Y[U <= dbeta(Y, shape1 = a, shape2 = b) / opt_c])
    N <- length(X)
    if ( N > n ) X <- X[1:n]
  }
}
```

```

}
x
}

```

Now we try to compare this A-R function to R `rbeta` function.

```

gen_beta <-
  tibble(
    ar_rand = ar_beta(1000, 3, 2),
    sam = rbeta(1000, 3, 2)
  ) %>%
  gather(key = "den", value = "value")

gg_curve(dbeta, from = 0, to = 1, args = list(shape1 = 3, shape2 = 2)) +
  geom_histogram(
    data = gen_beta,
    aes(x = value, y = ..density.., fill = den),
    position = "identity",
    bins = 30,
    alpha = .45
  ) +
  scale_fill_discrete(
    name = "random number",
    labels = c("AR", "rbeta")
  )

```

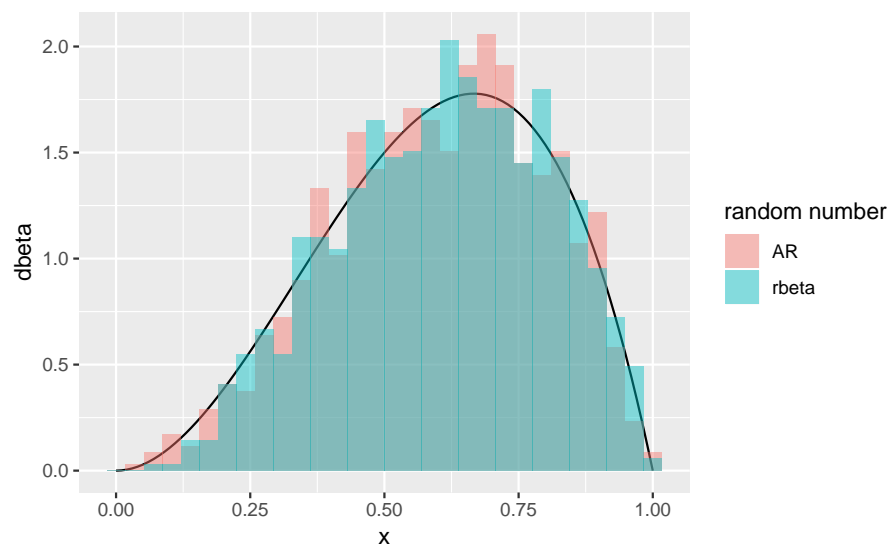


Figure 1.6: Beta(3,2) Random numbers from each function

In the Figure 1.6, the both histograms are very close to the true density curve. To see more statistically, we can draw a Q-Q plot.

```

gen_beta %>%
  ggplot(aes(sample = value)) +
  stat_qq_line(
    distribution = stats::qbeta,
    dparams = list(shape1 = 3, shape2 = 2),
    col = I("grey70"),

```

```

size = 3.5
) +
stat_qq(
  aes(colour = den),
  distribution = stats::qbeta,
  dparams = list(shape1 = 3, shape2 = 2)
) +
scale_colour_discrete(
  name = "random number",
  labels = c("AR", "rbeta")
)

```

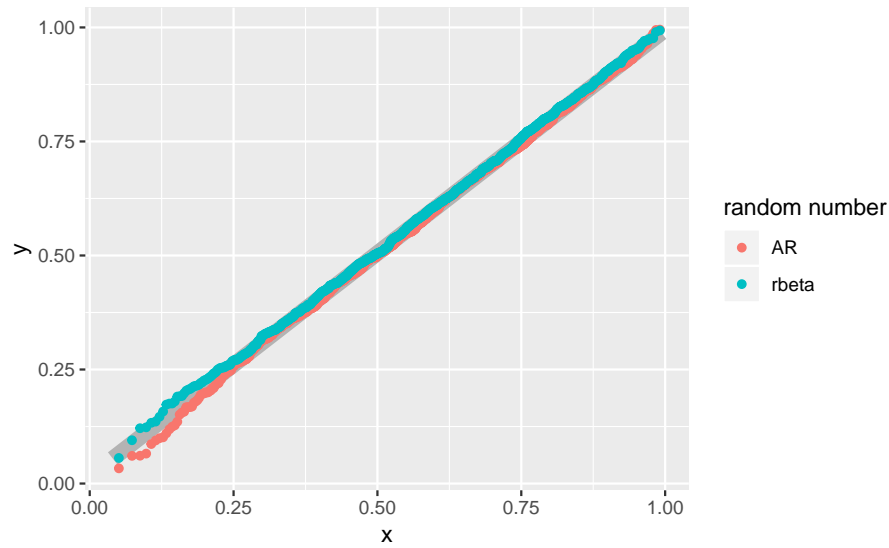


Figure 1.7: Q-Q plot for Beta(3,2) random numbers

See Figure 1.7. We have got series of numbers that are stuck to the beta distribution line.

Example 1.4 (A-R Method for Discrete case). A-R method can be also implemented to discrete case such as Example 1.2.

Table 1.3: Example of a Discrete Random Variable

x	0.0	1.0	2.0	3.0	4.0
p	0.1	0.2	0.2	0.2	0.3

Solution (Generating discrete random numbers using A-R methods). Consider proposal $g(x) \sim \text{Discrete unif}(0, 1, 2, 3, 4)$, i.e.

$$g(0) = g(1) = \dots = g(4) = 0.2$$

Then we set

$$c = \max \left\{ \frac{p(x)}{g(x)} : x = 0, \dots, 4 \right\} = \max \{ 0.5, 1, 1.5 \} = 1.5$$

1.5 Transformation Methods

1.6 Sums and Mixtures

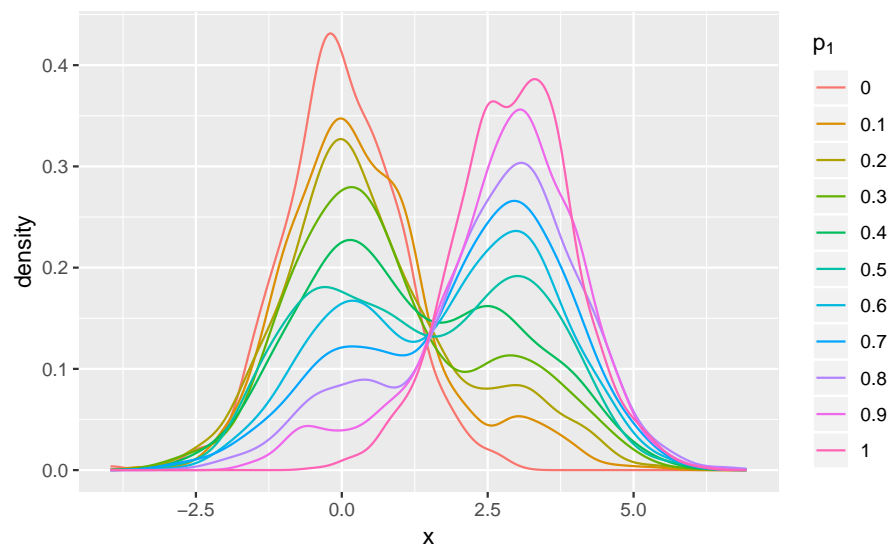
1.6.1 Sums

```
library(foreach)

mix_norm <- function(n, p1, mean1, sd1, mean2, sd2) {
  x1 <- rnorm(n, mean = mean1, sd = sd1)
  x2 <- rnorm(n, mean = mean2, sd = sd2)
  k <- as.integer(runif(n) > p1)
  k * x1 + (1 - k) * x2
}

mixture <-
  foreach(p1 = 0:10 / 10, .combine = bind_rows) %do% {
    tibble(
      value = mix_norm(n = 1000, p1 = p1, mean1 = 0, sd1 = 1, mean2 = 3, sd2 = 1),
      key = rep(p1, 1000)
    )
  }

mixture %>%
  ggplot(aes(x = value, colour = factor(key))) +
  stat_density(geom = "line", position = "identity") +
  scale_colour_discrete(
    name = expression(p[1]),
    labels = 0:10 / 10
  ) +
  xlab("x")
```



1.7 Multivariate Normal Random Vector

1.8 Stochastic Processes

1.8.1 Homogeneous poisson process

1.8.2 Nonhomogeneous poisson process

1.8.3 Symmetric random walk

Chapter 2

Monte Carlo Integration and Variance Reduction