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

12 Apr, 2019

Contents

Welcome

Statistical computing mainly treats useful simulation methods.

```r
library(tidyverse)
```

`tidyverse` package family will be used in every chapter. Loading step is in `_common.R`, so it is not included in the text. Sometimes `data.table` library will be called for efficiency.

Statistical Computing

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

Sampling from a fininte 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.

```r
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.

```r
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.

```r
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)
  )
```

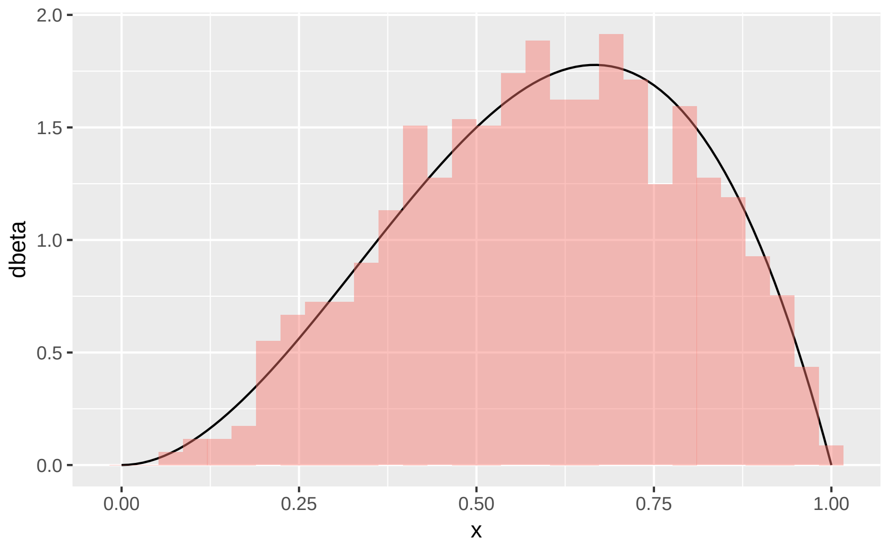


Figure 1: Beta(3,2) random numbers

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.

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

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

1.2.1 Linear congruential generator

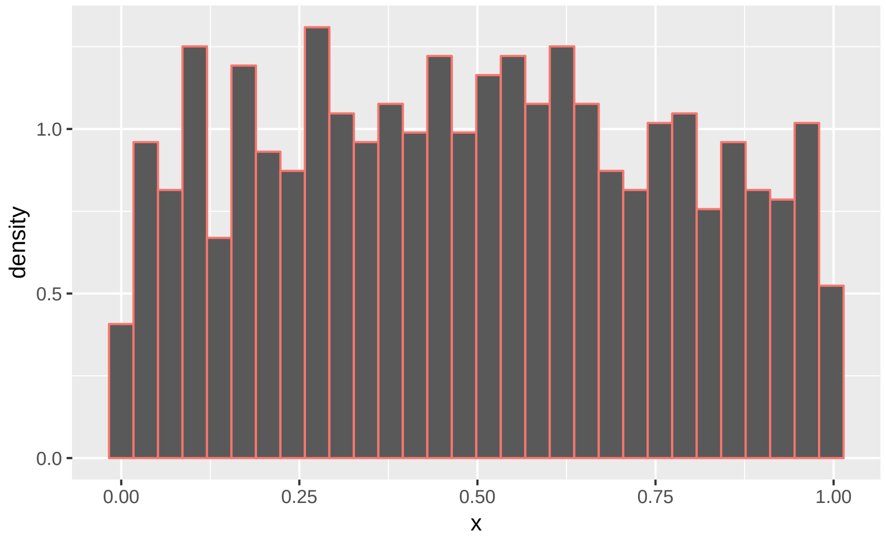
Then $u_1, u_2, \ldots, u_n \sim unif(0,1)$

Algorithm 1: Linear congruential generator

```
  input  : $a, c \in \mathbb{Z}_+$ and modulus $m$
1 Initialize $x_0$;
2 for $i \leftarrow 1$ to $n$ do
3     $x_i = (ax_{i-1} + c) \mod m$;
4 end
5 $u_i = \frac{x_i}{m} \in (0,1)$;
  output: $u_1, u_2, \ldots, u_n \sim unif(0,1)$
```

```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

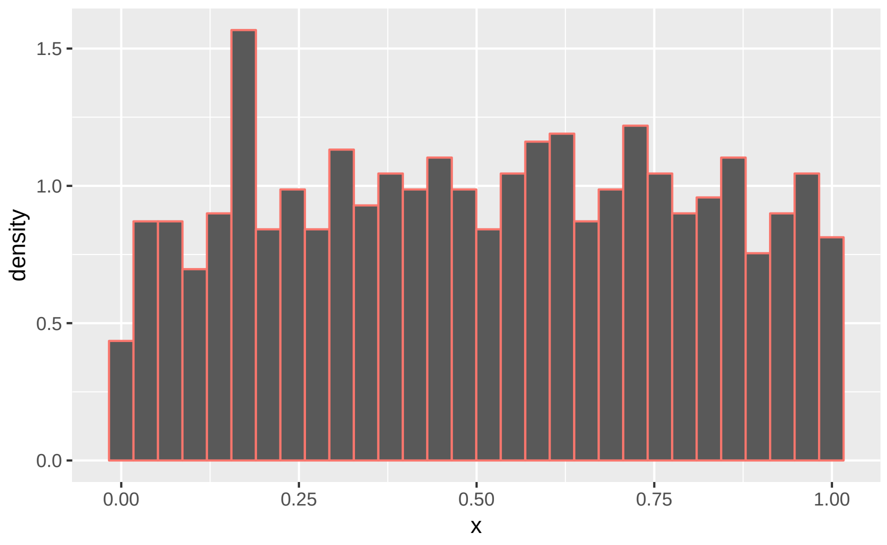
As we can expect from its name, this is congruential generator with $c = 0$.

Algorithm 2: Multiplicative congruential generator

```
   input : $a, \in \mathbb{Z}_+$ and modulus $m$
1  Initialize $x_0$;
2  for $i \leftarrow 1$ to $n$ do
3      $x_i = a x_{i-1} \mod m$;
4  end
5  $u_i = \frac{x_i}{m} \in (0,1)$;
   output: $u_1, u_2, \ldots, u_n \sim unif(0,1)$
```

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 $x_0 = 0$.

```r
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 difine $F_X^{-1}(y)$ 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*

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

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

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

$\square$

Thus, to generate n random variables $\sim F_X$, we can use *uniform random numbers*.

Algorithm 3: Inverse transformation method

```
  input  : analytical form of $F_X^{-1}$
1 for $i \leftarrow 1$ to $n$ do
2     $u_i \overset{iid}{\sim} unif(0, 1)$;
3     $x_i = F_X^{-1}(u_i)$;
4 end
  output: $x_1, x_2, \ldots, x_n \overset{iid}{\sim} F_X$
```

Note that in `R`, vectorized operation would be better, i.e. generate `runif(n)` and plug it into given inverse cdf.

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

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

Note that

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

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

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

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

```r
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
  )
```

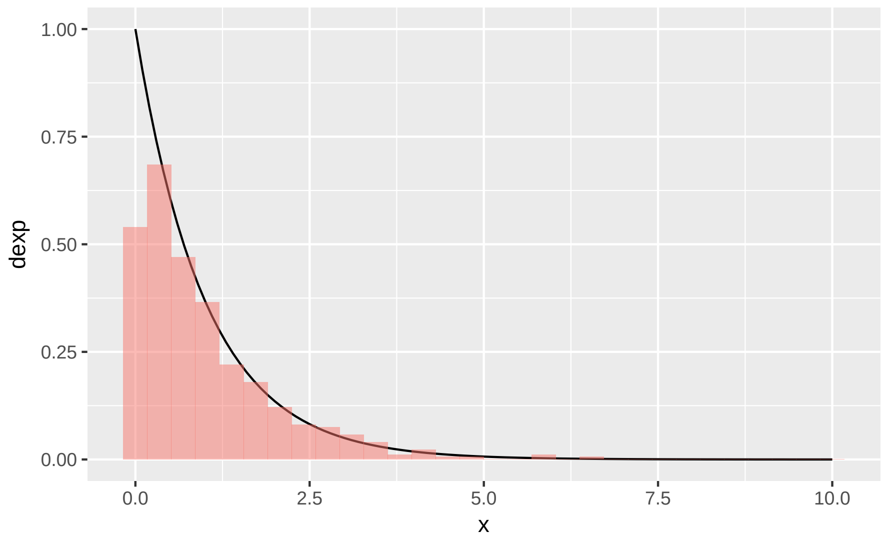


Figure 1.1: Inverse Transformation: Exp(1)

1.3.2 Discrete case

Algorithm 4: Inverse transformation method in discrete case

```
  input  : analytical form of $F_X$
1 for $i \leftarrow 1$ to $n$ do
2     $u_i \overset{iid}{\sim} unif(0,1)$;
3     Take $x_i$ s.t. $F_X(x_{i-1}) < U \leq F_X(x_i)$;
4 end
  output: $x_1, x_2, \ldots, x_n \overset{iid}{\sim} F_X$
```

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.

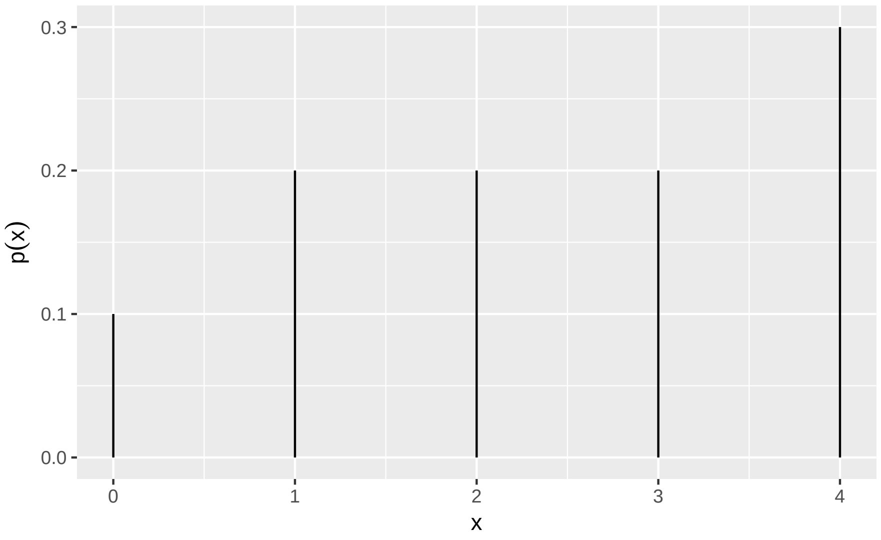


Figure 1.2: Probability Mass Function

Then we have the cdf

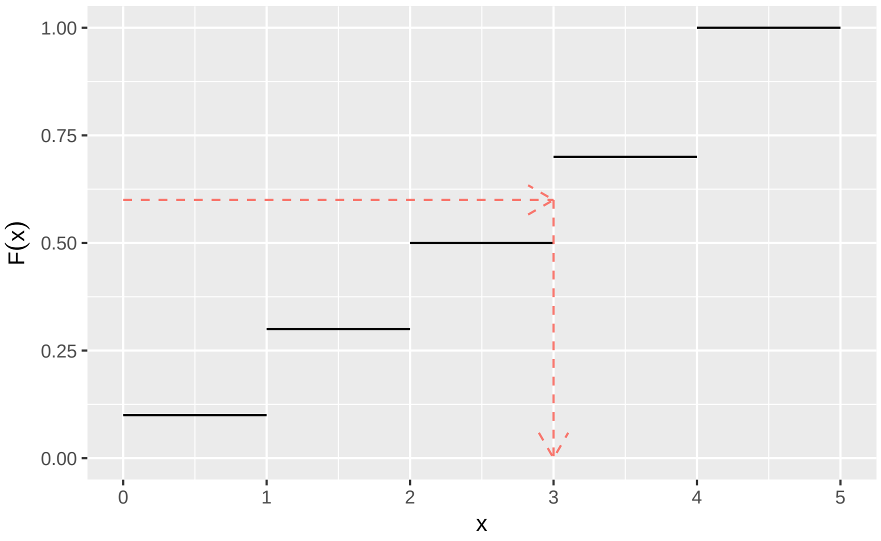


Figure 1.3: CDF of the Discrete Random Variable: Illustration for discrete case

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

```r
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 = ..ndensity..), binwidth = .1)
```

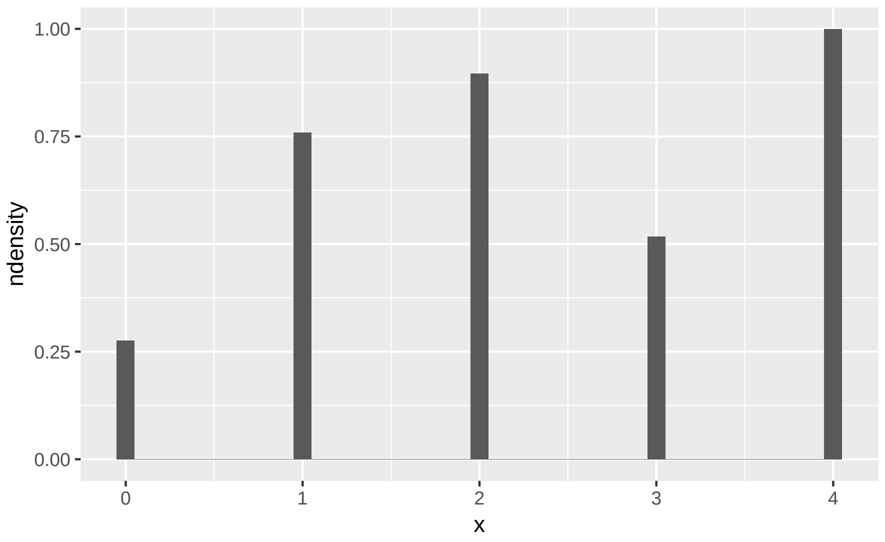


Figure 1.4: Generated discrete random numbers

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. Propsal 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

$$sptf \subseteq sptg$$

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

$$\forall x \in sptf : \frac{f(x)}{g(x)} \le c$$

Algorithm 5: Acceptance-rejection algorithm

```
    input  : target $f$, proposal $g$, and $c$
1   for $i \leftarrow 1$ to $n$ do
2       $Y \sim g(y)$;
3       $U \sim unif(0,1) \perp\!\!\!\perp Y$;
4       if $U \le \frac{f(Y)}{cg(Y)}$ then
5           Accept $x_i = Y$;
6       else
7           go to Line 2;
8       end
9   end
    output: $x_1, x_2, \ldots, x_n \overset{iid}{\sim} f(x)$
```

1.4.1 Efficiency

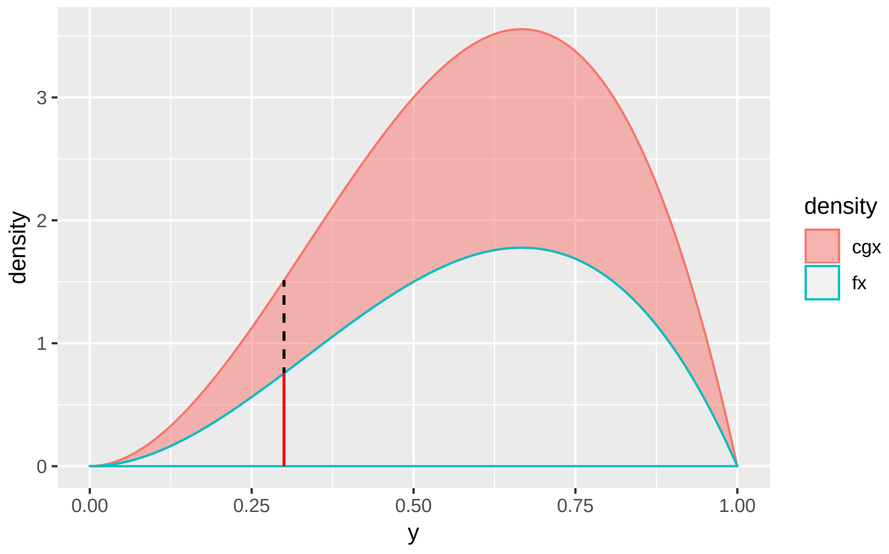


Figure 1.5: Property of AR method

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

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

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)**.** *See Figure 1.5.*

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*

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

and so

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

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

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

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

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

It follows that

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

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*

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

1.4.2 Examples

Example 1.3 (Beta(a,b))**.** Let $X \sim Beta(a, b)$. 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. $unif(0, 1)$. 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

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

It follows that

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

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

```r
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.

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

```r
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")
  )
```

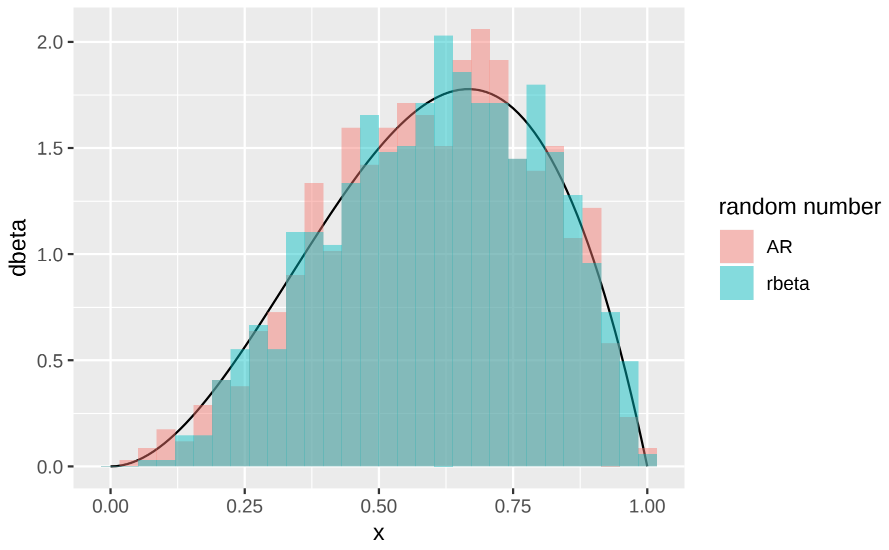


Figure 1.6: Beta(3,2) Random numbers from each function

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")
  )
```

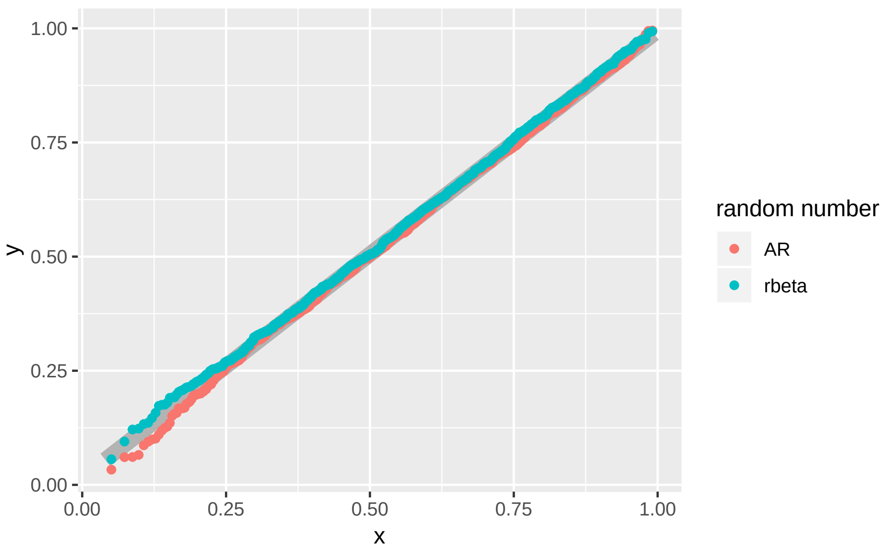


Figure 1.7: Q-Q plot for Beta(3,2) random numbers

See Figure 1.7. We have got series of numbers that are sticked 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 $g(x) \sim$ Discrete unif$(0, 1, 2, 3, 4)$, i.e.

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

Then we set

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

1.5 Transfomation Methods

1.5.1 Continuous

Proposition 1.2 (Transformation between continuous random variables)**.** *Relation between random variables enables generating target numbers from the others.*

1. $Z_1, \ldots, Z_n \overset{iid}{\sim} N(0,1) \Rightarrow \sum Z_i^2 \sim \chi^2(n)$
2. $Y_1 \sim \chi^2(m) \perp\!\!\!\perp Y_2 \sim \chi^2(n) \Rightarrow \frac{Y_1/m}{Y_2/n} \sim F(m,n)$
3. $Z \sim N(0,1) \perp\!\!\!\perp Y \sim \chi^2(n) \Rightarrow \frac{Z}{\sqrt{Y/n}} \sim t(n)$

4. $Y_1, \ldots, Y_n \overset{iid}{\sim} Exp(\lambda) \Rightarrow \sum Y_i^2 Gamma(n, \lambda)$
5. $U \sim unif(0,1) \Rightarrow (b-a)U + a \sim unif(a,b)$
6. $U \sim Gamma(r,\lambda) \perp\!\!\!\perp V \sim Gamma(s,\lambda) \Rightarrow \frac{U}{U+V} \sim Beta(r,s)$
7. $Z \sim N(0,1) \Rightarrow \mu + \sigma Z \sim N(\mu, \sigma^2)$
8. $Y \sim N(\mu, \sigma^2) \Rightarrow e^Y \sim LogNormal(\mu, \sigma^2)$

Example 1.5 (Generating Beta(a, b) using rgamma)**.** From Proposition 1.2, we can generate $Beta(a,b)$ random numbers using $Gamma(a,1)$ and $Gamma(b,1)$.

```r
trans_beta <- function(n, shape1, shape2) {
  u <- rgamma(n, shape = shape1, rate = 1)
  v <- rgamma(n, shape = shape2, rate = 1)
  u / (u + v)
}
```

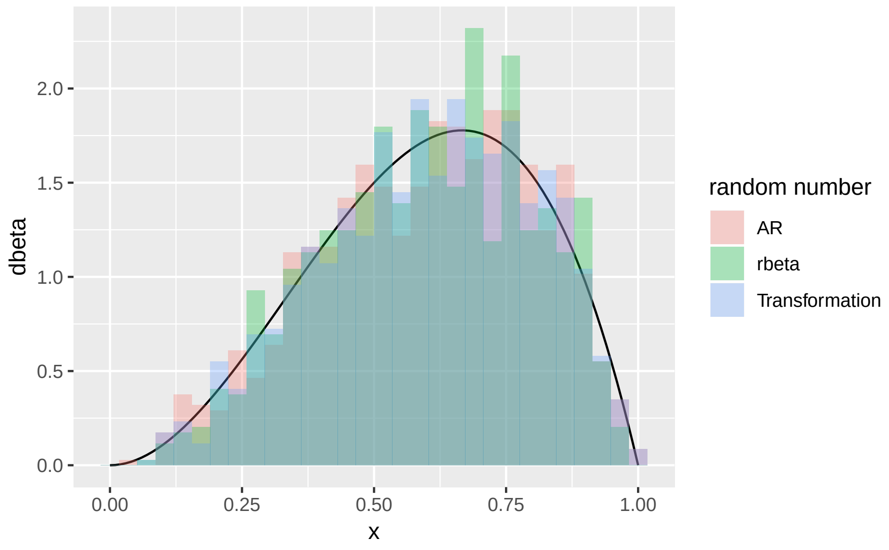


Figure 1.8: Beta(3,2) Random numbers from each function, including transformation method

1.5.2 Box-Muller transformation

Denote that Gaussian cdf has no closed form of F_X^{-1}. Using polar coordiantes, we can generate Normal random numers.

Theorem 1.2 (Box-Muller transformation)**.** *Let* $U_1, U_2 \overset{iid}{\sim} unif(0,1)$. *Then*

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

Proof. Write

$$(Z_1, Z_2)^T \sim N\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right)$$

Then the joint pdf is given by

$$f_{Z_1,Z_2}(x_1,x_2) = \frac{1}{2\pi}\exp\bigg(-\frac{x_1^2+x_2^2}{2}\bigg)$$

Consider polar coordiate transformation (R,θ): $x_1 = R\cos\theta$ and $x_2 = R\sin\theta$.

Since it is also random vector,

$$\begin{aligned}
f_{R,\theta}(r,\theta) &= f_{Z_1,Z_2}(x_1,x_2)|J| \\
&= \frac{1}{2\pi}\exp\bigg(-\frac{x_1^2+x_2^2}{2}\bigg)\begin{vmatrix} \frac{\partial x_1}{\partial r} & \frac{\partial x_1}{\partial \theta} \\ \frac{\partial x_2}{\partial r} & \frac{\partial x_2}{\partial \theta} \end{vmatrix} \\
&= \frac{1}{2\pi}\exp\bigg(-\frac{r^2}{2}\bigg)\begin{vmatrix} \frac{\partial x_1}{\partial r} & \frac{\partial x_1}{\partial \theta} \\ \frac{\partial x_2}{\partial r} & \frac{\partial x_2}{\partial \theta} \end{vmatrix} \\
&= \frac{r}{2\pi}\exp\bigg(-\frac{r^2}{2}\bigg)
\end{aligned}$$

Then each marginal density function can be computed as

$$\begin{aligned}
f_\theta(\theta) &= \int_0^\infty \frac{r}{2\pi}\exp\bigg(-\frac{r^2}{2}\bigg)dr \\
&= \frac{1}{2\pi}I_{(0,2\pi)}(\theta) \\
&\overset{d}{=} unif(0,2\pi)
\end{aligned}$$

$$\begin{aligned}
f_R(r) &= \int_0^\theta \frac{r}{2\pi}\exp\bigg(-\frac{r^2}{2}\bigg)d\theta \\
&= r\exp\bigg(-\frac{r^2}{2}\bigg)I_{(0,\infty)}(r)
\end{aligned}$$

Thus,

$$f_{R,\theta} = f_\theta f_R \Rightarrow R \perp\!\!\!\perp \theta$$

It follows from inverse transformation theorem that

$$Z_1 = R\cos\theta = \sqrt{-2\ln U_2}\cos(2\pi U_1)$$

and that

$$Z_2 = R\sin\theta = \sqrt{-2\ln U_2}\sin(2\pi U_1)$$

where $U_1, U_2 \overset{iid}{\sim} unif(0,1)$ $\square$

Algorithm 6: Box-Muller transformation

```
1 for $i \leftarrow 1$ to $n$ do
2     $U_1, U_2 \overset{iid}{\sim} unif(0, 1)$;
3     $z_{2i-1} = \sqrt{-2 \ln U_2} \cos(2\pi U_1)$;
4     $z_{2i} = \sqrt{-2 \ln U_2} \sin(2\pi U_1)$;
5 end
  output: $z_1, \ldots, z_n \overset{iid}{\sim} N(0, 1)$
```

```r
bmnorm <- function(n, mean = 0, sd = 1) {
  n_bm <- ceiling(n / 2)
  tibble(
    theta = runif(n = n_bm, max = 2 * pi),
    R = sqrt(-2 * log(runif(n_bm)))
  ) %>%
    mutate(
      x1 = R * cos(theta),
      x2 = R * sin(theta)
    ) %>%
    gather(x1, x2, key = "key", value = "value") %>%
    mutate(value = mean + sd * value) %>%
    select(value) %>%
    pull()
}
```

```r
gg_curve(dnorm, from = 0, to = 6, args = list(mean = 3, sd = 1)) +
  geom_histogram(
    data = tibble(x = bmnorm(1000, mean = 3, sd = 1)),
    aes(x = x, y = ..density..),
    bins = 30,
    fill = gg_hcl(1),
    alpha = .5
  )
```

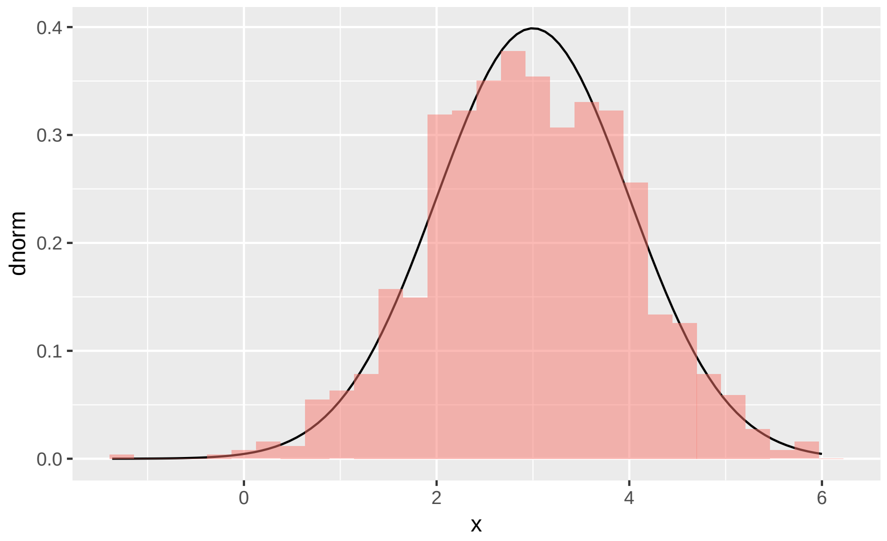


Figure 1.9: Normal random numbers by Box-Muller transformation

1.5.3 Discrete

Proposition 1.3 (Transformation between discrete random variables)**.** *Relation between random variables enables generating target numbers from the others.*

1. $Y_1, \ldots, Y_n \overset{iid}{\sim} Bernoulli(p) \Rightarrow \sum Y_i^2 \sim B(n,p)$
2. $U \sim unif(0,1) \Rightarrow X_i = \lfloor mU \rfloor + 1$
3. $X =$ *the number of events occurring in 1 unit of time* $\sim Poisson(\lambda)$

Proposition 1.4 (Bernoulli process)**.** *Let* $X_1, X_2, \ldots \overset{iid}{\sim} Bernoulli(p)$*.*

1. $N =$ *the number of trials until we see a success, i.e.*$X_N = 1 \Rightarrow N \sim Geo(p)$
2. $Y_1, \ldots, Y_r \overset{iid}{\sim} Geo(p) \Rightarrow \sum_{i=1}^r Y_i =$ *the number of trials until we see* r *successes* $\sim NegBin(r,p)$

Proposition 1.5 (Count process)**.** *Let* $Y_1, Y_2, \ldots \overset{iid}{\sim} Exp(\lambda)$ *be interarrival times. Then*

$$X = \max\{n : \sum Y_i \le 1\} = \text{the number of events occurring in 1 unit of time} \sim Poisson(\lambda)$$

1.6 Sums and Mixtures

1.6.1 Convolutions

Definition 1.3 (Convolution)**.** Let $X_1, \ldots, X_n$ be independent and identically distributed and let $S = X_1 + \cdots X_n$. Then the distribution of S is called the n-fold convolution of X and denoted by $F_X^{*(n)}$.

In the last chapter, we have already seen a bunch of random variables that can be generated by summing the other.

Example 1.6 (Chisquare)**.** Let $Z_1, \ldots, Z_n \overset{iid}{\sim} N(0,1)$. We know from Proposition 1.2 that

$$V = \sum_{i=1}^n Z_i \sim \chi^2(n)$$

Building a `n` $\times$ `df` matrix can be a good strategy here. After that, `rowSums` or `colSums` ends the generation work.

```r
conv_chisq <- function(n, df) {
  X <-
    matrix(rnorm(n * df), nrow = n, ncol = df)^2
  rowSums(X)
}
```

```r
gg_curve(dchisq, from = 0, to = 15, args = list(df = 5)) +
  geom_histogram(
    data = tibble(x = conv_chisq(1000, df = 5)),
    aes(x = x, y = ..density..),
    bins = 30,
    fill = gg_hcl(1),
    alpha = .5
  )
```

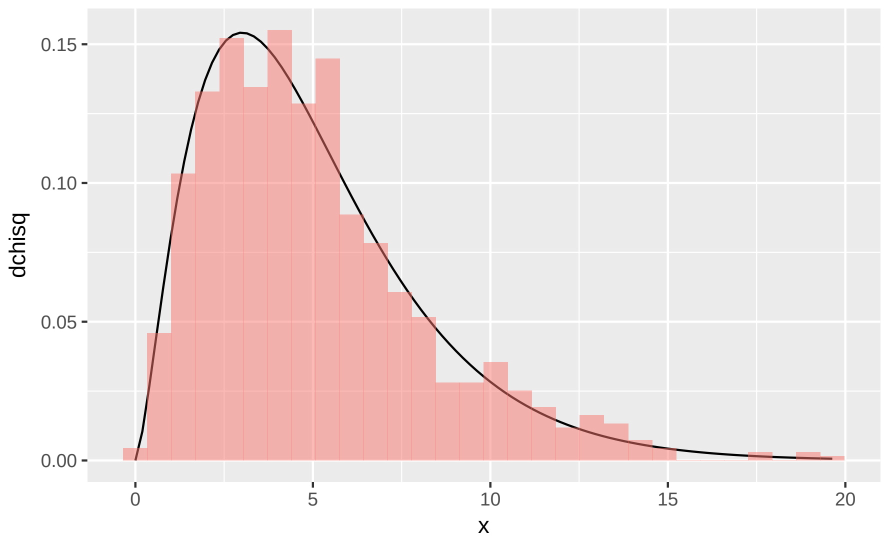


Figure 1.10: χ^2 random numbers from Normal sums

1.6.2 Mixtures

Definition 1.4 (Discrete mixture)**.** A random variable X is a discrete mixture if the distribution of X is a weighted sum

$$F_X(x) = \sum \theta_i F_{X_i}(x)$$

where constants θ_i are called the mixing weights or mixing probabilities.

Definition 1.5 (Continuous mixture)**.** A random variable X is a continuous mixture if the distribution of X is a weighted sum

$$F_X(x) = \int_{\infty}^{\infty} F_{X|Y=y}(x) f_Y(y) dy$$

Example 1.7 (Mixture of several Normal distributions)**.** Generate a random sample of size 1000 from a normal location mixture with components of the mixture $N(0,1)$ and $N(3,1)$, i.e.

$$F_X = p_1 F_{X_1} + (1 - p_1) F_{X_2}$$

For easy combining samples, we use `foreach` library.

```
library(foreach)
```

As in A-R method, Bernoullin splitting would be used.

$$\begin{cases} F_{X_1} & U > p_1 \\ F_{X_2} & \text{otherwise} \end{cases}$$

```
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)
```

```r
  k * x1 + (1 - k) * x2
}
```

Try various p_1, from 0.1 to 1

```r
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)
    )
  }
```

```r
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")
```

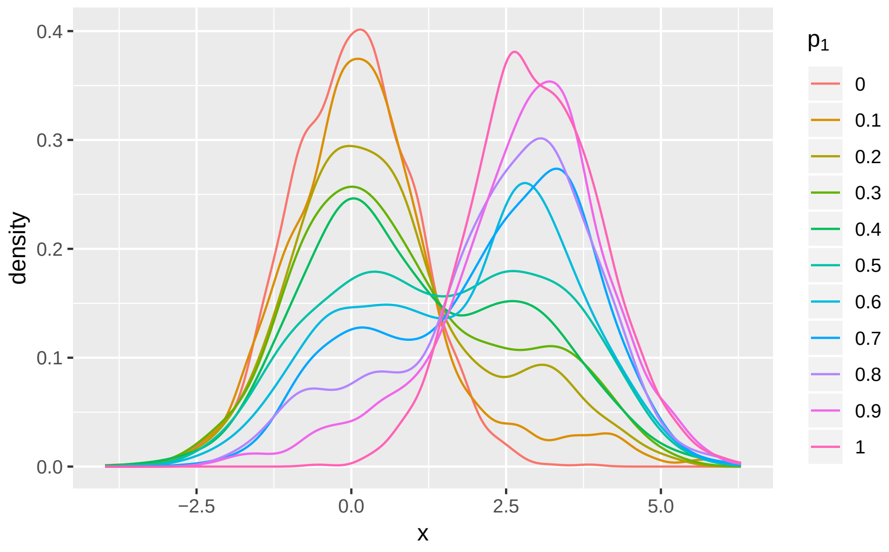


Figure 1.11: Mixture normal random number for each mixing probability

1.7 Multivariate Normal Random Vector

Definition 1.6 (Multivariate normal random vector)**.** A random vector $\mathbf{X} = (X_1, \ldots, X_p)^T$ follows multivariate normal distribution if

$$f(\mathbf{x}) = \frac{1}{(2\pi)^{\frac{p}{2}|\Sigma|}} \exp\left[-\frac{1}{2}(\mathbf{x}\boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x}\boldsymbol{\mu}) \right]$$

Remark. Let $\mathbf{Z} \sim MVN(\mathbf{0}, I)$. Then

$$\Sigma^{\frac{1}{2}}\mathbf{Z} + \boldsymbol{\mu} \sim MVN(\boldsymbol{\mu}, \Sigma) \tag{1.1}$$

From this remark, we get to generate *standard normal random vector*.

1.7.1 Spectral decomposition method

Note that covariance matrix is symmetric.

Theorem 1.3 (Spectral decomposition)**.** *Suppose that* Σ *is symmetric. Then*

$$\Sigma = P\Lambda P^T$$

where $(\mathbf{v}_j, \lambda_j)$ *corresponding eigenvector-eigenvalue*

$$\begin{cases} P = \begin{bmatrix} \mathbf{v}_1 & \cdots & \mathbf{v}_p \end{bmatrix} \in \mathbb{R}^{p \times p} \textit{ orthogonal} \\ \Lambda = diag(\lambda_1, \ldots, \lambda_p) \end{cases}$$

Corollary 1.3. *Suppose that* Σ *is symmetric. Then*

$$\Sigma^{\frac{1}{2}} = P\Lambda^{\frac{1}{2}}P^T$$

where $\Lambda^{\frac{1}{2}} = diag(\sqrt{\lambda_1}, \ldots, \sqrt{\lambda_p})$

`eigen()` performs spectral decomposition. `$values` has eigenvalues and `$vectors` has eigenvectors. We first generate matrix that consists of standard normal random vector:

$$\begin{bmatrix} Z_{11} & Z_{12} & \cdots & Z_{1p} \\ Z_{21} & Z_{22} & \cdots & Z_{2p} \\ \vdots & \vdots & \vdots & \vdots \\ Z_{n1} & Z_{n2} & \cdots & Z_{np} \end{bmatrix}$$

Denote that each observation is row. To use Equation (1.1), we should multiply $\Sigma^{\frac{1}{2}}$ behind this matrix, not in front of. $\boldsymbol{\mu}$ matrix should be also made to matrix, in form of

$$\begin{bmatrix} \mu_{11} & \mu_{12} & \cdots & \mu_{1p} \\ \mu_{11} & \mu_{22} & \cdots & \mu_{1p} \\ \vdots & \vdots & \vdots & \vdots \\ \mu_{11} & Z_{n2} & \cdots & \mu_{1p} \end{bmatrix} \in \mathbb{R}^{n \times p}$$

```r
rmvn_eigen <- function(n, mu, sig) {
  d <- length(mu)
  ev <- eigen(sig, symmetric = TRUE)
  lambda <- ev$values
  P <- ev$vectors
  sig2 <- P %*% diag(sqrt(lambda)) %*% t(P)
  Z <- matrix(rnorm(n * d), nrow = n, ncol = d)
  X <- Z %*% sig2 + matrix(mu, nrow = n, ncol = d, byrow = TRUE)
  colnames(X) <- paste0("x", 1:d)
  X %>% tbl_df()
}
```

```r
# mean vector ------------------------------
mu <- c(0, 1, 2)
# symmetric matrix -------------------------
sig <- matrix(numeric(9), nrow = 3, ncol = 3)
diag(sig) <- rep(1, 3)
sig[lower.tri(sig)] <- c(-.5, .5, -.5) * 2
sig <- (sig + t(sig)) / 2
```

Generate

$$\mathbf{X}_i \sim MVN\left((0,1,2), \begin{bmatrix} 1 & -0.5 & 0.5 \\ -0.5 & 1 & -0.5 \\ 0.5 & -0.5 & 1 \end{bmatrix} \right)$$

```r
(mvn3 <- rmvn_eigen(1000, mu = mu, sig = sig))
#> # A tibble: 1,000 x 3
#>         x1       x2    x3
#>      <dbl>    <dbl> <dbl>
#>  1 -0.168   1.41    1.80
#>  2  1.39   -0.00942 2.40
#>  3 -0.710   1.30    1.37
#>  4  0.0314  2.04    1.80
#>  5  0.177   0.568   1.71
#>  6 -0.960   1.23    1.61
#>  7 -1.01    1.28    0.106
#>  8  0.272   0.0842  2.12
#>  9  0.148   1.63    2.53
#> 10 -1.24    1.53    1.28
#> # ... with 990 more rows
```

```r
mvn3 %>%
  GGally::ggpairs(
    lower = list(continuous = GGally::wrap(gg_scatter, size = 1))
  )
```

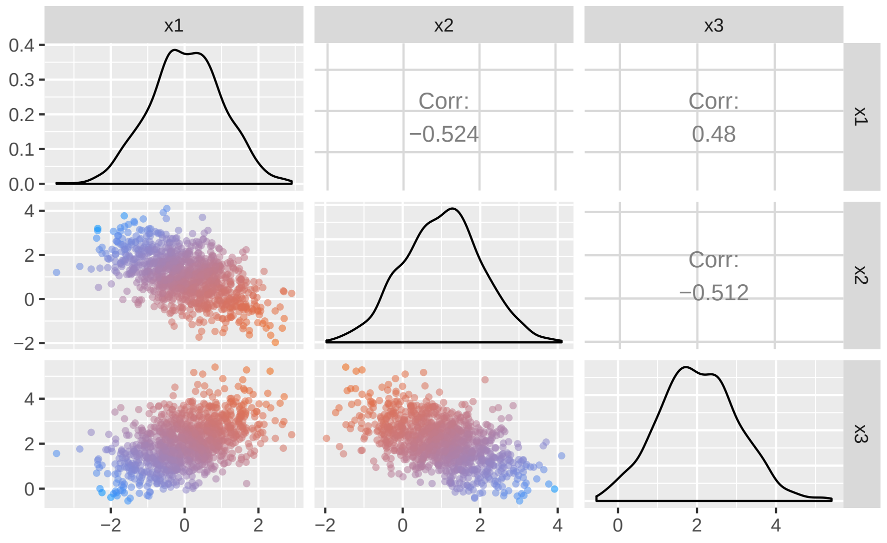


Figure 1.12: Multivariate normal random vector - spectral decomposition method

1.7.2 Singular value decomposition

SVD can be said to be a kind of generalization of spectral decomposition. This method can be used for any matrix, i.e. non-symmetric matrix. For Σ, SVD and spectral decomposition is equivalent. However, SVD does not account for symmetric property, so this method is less efficient compared to spectral decomposition.

```r
rmvn_svd <- function(n, mu, sig) {
  d <- length(mu)
  S <- svd(sig)
  sig2 <- S$u %*% diag(sqrt(S$d)) %*% t(S$v)
  Z <- matrix(rnorm(n * d), nrow = n, ncol = d)
  X <- Z %*% sig2 + matrix(mu, nrow = n, ncol = d, byrow = TRUE)
  colnames(X) <- paste0("x", 1:d)
  X %>% tbl_df()
}
```

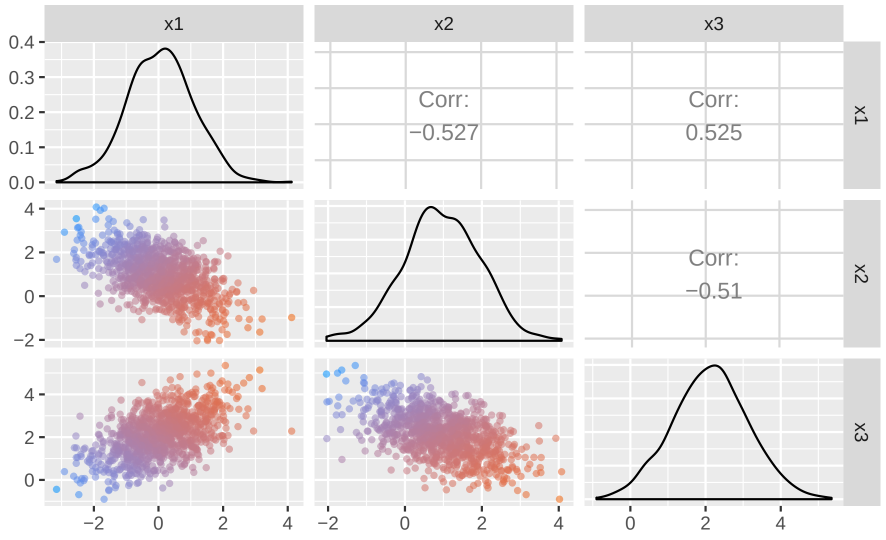


Figure 1.13: Multivariate normal random vector - svd

1.7.3 Choleski decomposition

Theorem 1.4 (Cholesky decomposition). *Suppose that Σ is symmetric and positive definite. Then*

$$\Sigma = Q^T Q$$

where Q is an upper triangular matrix.

Corollary 1.4. *Suppose that Σ is symmetric and positive definite. For cholesky decomposition 1.4, define*

$$\Sigma^{\frac{1}{2}} = Q$$

`chol()` computes cholesky decomposition. In `R`, it gives upper triangular Q. Since some statements cholesky decomposition by $\Sigma = LL^T$ with lower triangular matrix, try not to confuse.

```r
rmvn_chol <- function(n, mu, sig) {
  d <- length(mu)
  sig2 <- chol(sig)
  Z <- matrix(rnorm(n * d), nrow = n, ncol = d)
  X <- Z %*% sig2 + matrix(mu, nrow = n, ncol = d, byrow = TRUE)
  colnames(X) <- paste0("x", 1:d)
  X %>% tbl_df()
}
```

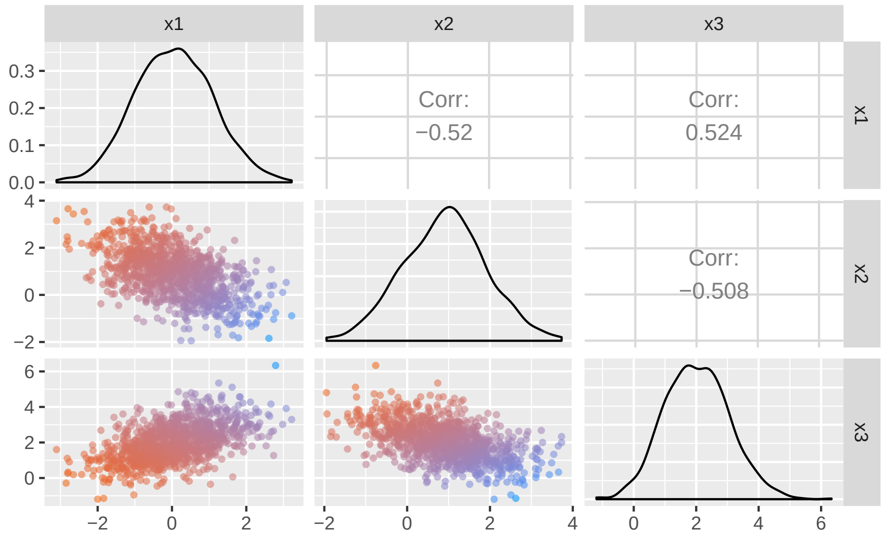


Figure 1.14: Multivariate normal random vector - cholesky decomposition

1.8 Stochastic Processes

Definition 1.7 (Stochastic process)**.** A stochastic process is a collection $\{X(t) : t \in T\}$ of random variables indexed by the set T. The index set T could be discrete or continous.

A State space is called te set of possible values that $X(t)$ can take.

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

2.1 Monte Carlo Integration

Consider integration problem of a integrable function $g(x)$. We want to compute

$$\theta \equiv \int_a^b g(x)dx$$

For instance, standard normal cdf.

Example 2.1 (Standard normal cdf)**.** Compute values for

$$\Phi(x) = \int_{-\infty}^x \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{t^2}{2} \right) dt$$

It might be impossible to compute this integral with hand. So we implement *simulation* concept here, based on the following theorems.

Theorem 2.1 (Weak Law of Large Numbers)**.** *Suppose that* $X_1, \ldots, X_n \overset{iid}{\sim} (\mu, \sigma^2 < \infty)$*. Then*

$$\frac{1}{n}\sum_{i=1}^n X_i \xrightarrow{p} \mu$$

Let g *be a measurable function. Then*

$$\frac{1}{n}\sum_{i=1}^n g(X_i) \xrightarrow{p} g(\mu)$$

Theorem 2.2 (Strong Law of Large Numbers)**.** *Suppose that* $X_1, \ldots, X_n \overset{iid}{\sim} (\mu, \sigma^2 < \infty)$*. Then*

$$\frac{1}{n}\sum_{i=1}^n X_i \xrightarrow{a.s.} \mu$$

Let g *be a measurable function. Then*

$$\frac{1}{n}\sum_{i=1}^{n} g(X_i) \overset{a.s.}{\to} g(\mu)$$

2.1.1 Simple Monte Carlo estimator

Theorem 2.3 (Monte Carlo Integration)**.** *Consider integration* (2.1)*. This can be approximated via appropriate pdf $f(x)$ by*

$$\hat{\theta}_M = \frac{1}{N}\sum_{i=1}^{N} g(X_i)$$

Suppose that we have a distribution $f(x) = I_{sptg}(x)$, i.e. *uniform distribution*. Let $sptg = (a, b)$.

$$\begin{aligned}
\theta &\equiv \int_{sptg} g(x)dx \\
&= \int_a^b g(x)dx \\
&= \int_0^1 g(a + (b-a)t)(b-a)dt \\
&\equiv \int_0^1 h(t)dt \\
&= \int_0^1 h(t)I_{(a,b)}(t)dt \\
&= E[h(U)] \qquad U \sim unif(0,1)
\end{aligned} \tag{2.1}$$

By *the Strong law of large numbers* 2.2,

$$\frac{1}{n}\sum_{i=1}^{n} h(U_i) \overset{a.s.}{\to} E\Big[h(U)\Big] = \theta$$

where $U \sim unif(0, 1)$. Thus, what we have to do here are two things.

1. representing g as h.
2. generating lots of U_i

Go back to Example 2.1.

Solution. Case 1: $x > 0$

Since $\Phi(x)$ is symmetry,

$$\Phi(0) = \frac{1}{2}$$

Fix $x > 0$. Let $y = \frac{t}{x}$ be a change of variable.

$$\begin{aligned}
\int_0^x \exp\Big(-\frac{t^2}{2}\Big)dt &= \int_0^x x\exp\Big(-\frac{t^2}{2}\Big)\frac{I_{(0,x)}(t)}{x}dt \\
&\approx \frac{1}{N}\sum_{i=1}^{N} x\exp\Big(-\frac{U_i^2}{2}\Big)
\end{aligned}$$

Table 2.1: Simple MC estimates of Normal cdf for each x

x	pnorm	mc
0.100	0.540	0.540
0.367	0.643	0.643
0.633	0.737	0.737
0.900	0.816	0.816
1.167	0.878	0.878
1.433	0.924	0.923
1.700	0.955	0.958
1.967	0.975	0.976
2.233	0.987	0.987
2.500	0.994	0.990

with $U_1, \ldots, U_N \overset{iid}{\sim} unif(0, x)$.

Case 2: $x \le 0$

Recall that $\Phi(x)$ is symmetry.

Hence,

$$\hat{\Phi}(x) = \begin{cases} \frac{1}{\sqrt{2\pi}} \frac{1}{N} \sum_{i=1}^N x \exp\left(-\frac{U_i^2}{2} \right) + \frac{1}{2} \equiv \hat{\theta}(x) & x \ge 0 \\ 1 - \hat{\theta}(-x) & x < 0 \end{cases}$$

```r
phihat <- function(x, y) {
  yi <- abs(y)
  theta <- mean(yi * exp(-x^2 / 2)) / sqrt(2 * pi) + .5
  ifelse(y >= 0, theta, 1 - theta)
}
```

Then compute $\hat{\Phi}(x)$ for various x values.

```r
phi_simul <- foreach(y = seq(.1, 2.5, length.out = 10), .combine = bind_rows) %do% {
  tibble(
    x = y,
    phi = pnorm(y),
    Phihat =
      tibble(x = runif(10000, max = y)) %>%
      summarise(cdf = phihat(x, y = y)) %>%
      pull()
  )
}
```

2.1.2 Hit-or-Miss Monte Carlo

Hit-or-Miss approach is another way to evaluate integrals.

Example 2.2 (Estimation of π)**.** Consider a circle in $\mathbb{R}$ coordinate.

$$x^2 + y^2 = 1$$

Since $y = \sqrt{1 - x^2}$,

$$\int_0^1 \sqrt{1-t^2}dt = \frac{\pi}{4} \tag{2.2}$$

By estimating Equation (2.2), we can estimate π, i.e.

$$\pi = 4\int_0^1 \sqrt{1-t^2}dt$$

Simple MC integration can also be used.

$$\begin{aligned}\int_0^1 \sqrt{1-t^2}dt &= \int_0^1 \sqrt{1-t^2}I_{(0,1)}(t)dt \\ &\approx \frac{1}{N}\sum_{i=1}^N \sqrt{1-U_i^2}\end{aligned}$$

```r
circ <- function(x) {
  4 * sqrt(1 - x^2)
}
```

```r
tibble(x = runif(10000)) %>%
  summarise(mc_pi = mean(circ(x)))
#> # A tibble: 1 x 1
#>   mc_pi
#>   <dbl>
#> 1  3.14
```

On the other way, hit-or-miss MC method applies geometric probability.

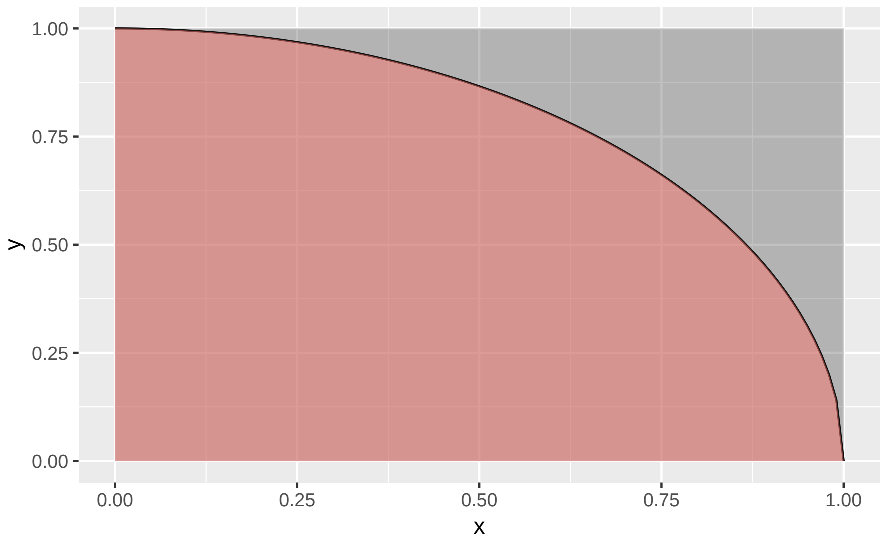


Figure 2.1: Hit-or-Miss

See Figure 2.1. From each coordinate, generate

- $X_i \overset{iid}{\sim} unif(0,1)$

- $Y_i \overset{iid}{\sim} unif(0,1)$

Then the proportion of $Y_i \le \sqrt{1 - X_i^2}$ estimates $\frac{\pi}{4}$.

```
tibble(x = runif(10000), y = runif(10000)) %>%
  summarise(hitormiss = mean(y <= sqrt(1 - x^2)) * 4)
#> # A tibble: 1 x 1
#>   hitormiss
#>       <dbl>
#> 1      3.15
```

2.2 Variance and Efficiency

We have seen two apporoaches doing the same task. Now we want to *evaluate them.* Denote that simple Monte Carlo integration 2.3 is estimating the *expected value of some random variable.* Proportion, which approximates probability is expected value of identity function.

The common statistic that can evaluate estimators expected value might be their variances.

2.2.1 Variance

Note that $Var(\overline{g(X)}) = \frac{Var(g(X))}{N}$. This property is one of estimating variance of $\hat{\theta}$. However, this *variance of sample mean* is used in situation when we are in sample limitation situation. Now, we can generate samples as many as we want to. So we try another approach: *parametric bootstrap*.

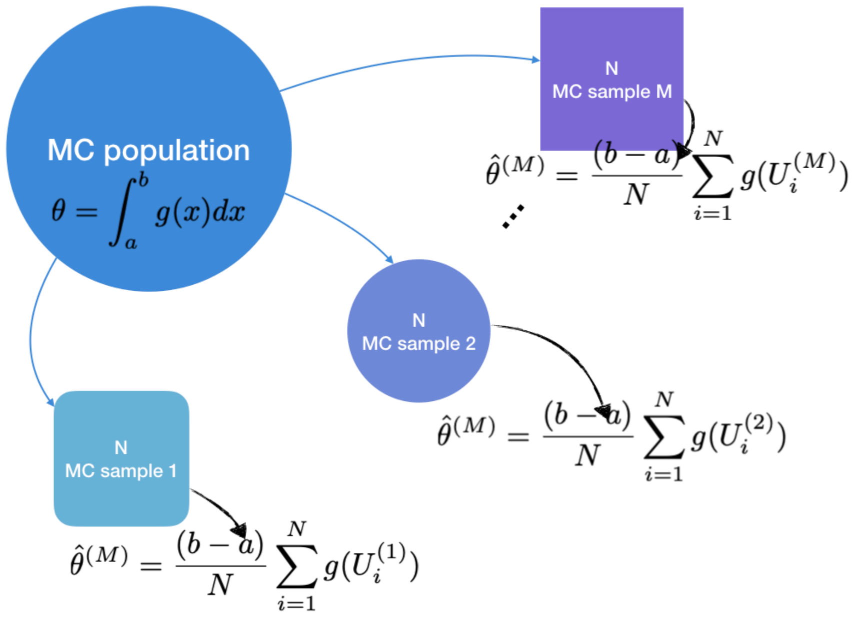

MC population
$\theta = \int_a^b g(x)dx$
N
MC sample M
$\hat{\theta}^{(M)} = \frac{(b-a)}{N}\sum_{i=1}^N g(U_i^{(M)})$
N
MC sample 2
$\hat{\theta}^{(M)} = \frac{(b-a)}{N}\sum_{i=1}^N g(U_i^{(2)})$
N
MC sample 1
$\hat{\theta}^{(M)} = \frac{(b-a)}{N}\sum_{i=1}^N g(U_i^{(1)})$

Figure 2.2: Empircal distribution of $\hat{\theta}$

See Figure 2.2. If we estimate $E\Big[g(U \sim unif(a,b))\Big]$, we can get θ. Generate M samples $\{U_1^{(j)}, \ldots, U_N^{(j)}\}, j = 1, \ldots M$ from this $U \sim unif(a,b)$. In each sample, calculate MC estimates $\hat{\theta}^{(j)}$. Now we have M MC estimates

$\hat{\theta}$. This gives empirical distribution of $\hat{\theta}$. By *drawing a histogram*, we can see the outline.

Algorithm 7: Variance of $\hat{\theta}$

```
  input  : $\theta = \int_a^b g(x)dx$
1 for $m \leftarrow 1$ to $M$ do
2     Generate $U_1^{(m)}, \ldots, U_N^{(m)} \overset{iid}{\sim} unif(a,b)$ ;
3     Compute $\hat{\theta}^{(j)} = \frac{(b-a)}{N} \sum g(U_i^{(j)})$;
4 end
5 $\bar{\hat{\theta}} = \frac{1}{M} \sum \hat{\theta}^{(j)}$;
6 $\widehat{Var}(\hat{\theta}) = \frac{1}{M-1} \sum (\hat{\theta}^{(j)} - \bar{\hat{\theta}})^2$;
  output: $\widehat{Var}(\hat{\theta})$
```

Since we have to generate large size of data, `data.table` package will be used.

```r
library(data.table)
```

Group operation can be used. Additional column (`sam`) would indicate group, and for each group MC operation would be processed. The following is the function generating `data.table` before group operation.

```r
mc_data <- function(rand, N = 10000, M = 1000, char = "s", ...) {
  data.table(
    u = rand(n = N * M, ...),
    sam = gl(M, N, labels = paste0("s", 1:M))
  )
}
```

```r
pi_mc <-
  mc_data(runif)[,
                 .(mc_pi = mean(circ(u))),
                 keyby = sam]
```

```r
pi_mc %>%
  ggplot(aes(x = mc_pi)) +
  geom_histogram(bins = 30, col = gg_hcl(1), alpha = .7) +
  xlab(expression(pi)) +
  geom_vline(xintercept = pi, col = gg_hcl(2)[2])
```

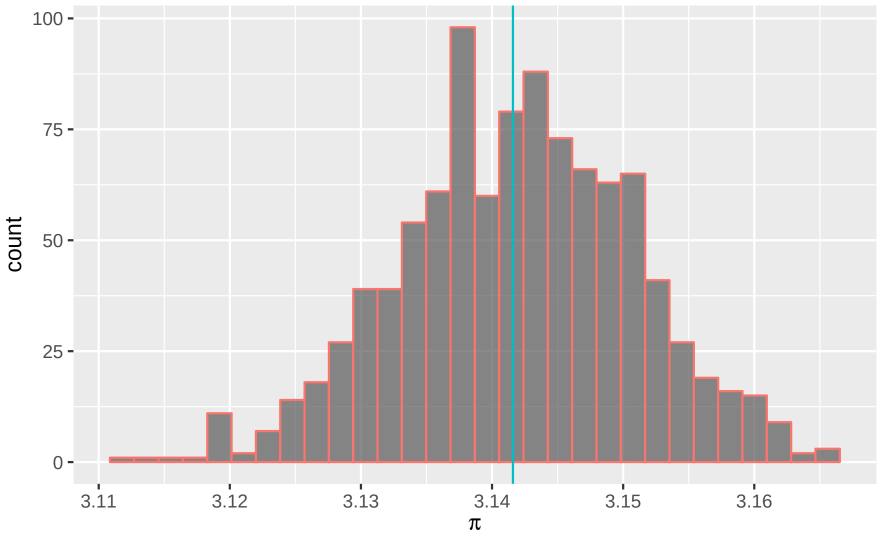


Figure 2.3: Empirical distribution of $\hat{\pi}$ by simple MC

As in Algorighm 7, we can compute the variance as below.

```
(mc_var <-
  pi_mc[,
        .(mc_variance = var(mc_pi))])
#>    mc_variance
#> 1:    8.07e-05
```

On the other hand, we need to generate two sets of random numbers for hit-or-miss MC.

```
pi_hit <-
  mc_data(runif)[
    , u2 := runif(10000 * 1000)
  ][,
    .(hitormiss = mean(u2 <= sqrt(1 - u^2)) * 4),
    keyby = sam]
```

```
pi_mc[pi_hit] %>%
  melt(id.vars = "sam", variable.name = "hat") %>%
  ggplot(aes(x = value, fill = hat)) +
  geom_histogram(bins = 30, alpha = .5, position = "identity") +
  xlab(expression(pi)) +
  geom_vline(xintercept = pi, col = I("red")) +
  scale_fill_discrete(
    name = "MC",
    labels = c("Simple", "Hit-or-Miss")
  )
```

Table 2.2: Simple MC versus Hit-or-Miss

SimpleMC	Hit-or-Miss	SimpleMCefficiency
0	0	TRUE

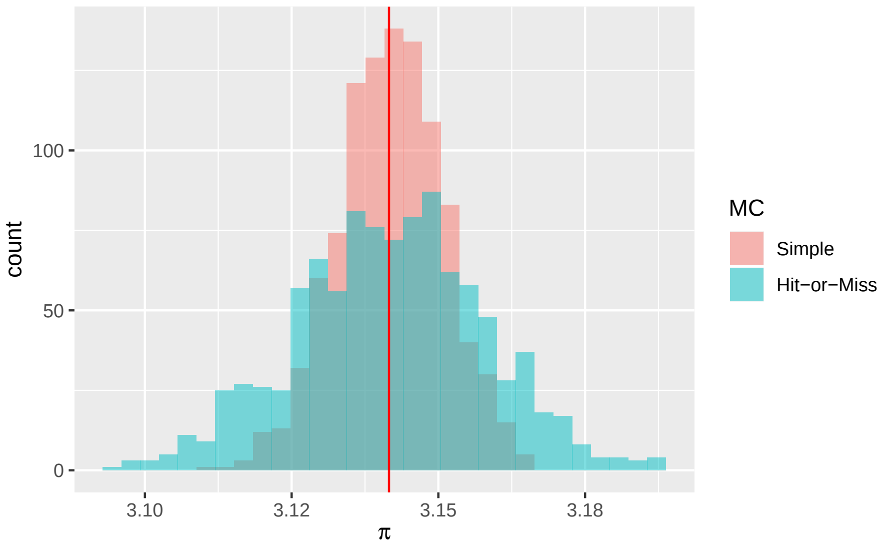


Figure 2.4: Simple MC and Hit-or-Miss MC

```
(hit_var <-
  pi_hit[,
         .(hit_variance = var(hitormiss))])
#>    hit_variance
#> 1:     0.000257
```

2.2.2 Efficiency

See Figure 2.4. It is obvious that Hit-or-Miss estimate produces larger variance than simple MC.

Definition 2.1 (Efficiency)**.** Let $\hat{\theta}_1$ and $\hat{\theta}_2$ be two estimators for θ. Then $\hat{\theta}_1$ is more efficient than $\hat{\theta}_2$ if

$$\frac{Var(\hat{\theta}_1)}{Var(\hat{\theta}_2)} < 1$$

In other words, if $\hat{\theta}_1$ has smaller variance than $\hat{\theta}_2$, then $\hat{\theta}_1$ is said to be efficient, which is preferable.

2.3 Variance Reduction

2.4 Antithetic Variables

2.5 Control Variates

2.6 Importance Sampling