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

20 Jun, 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] 0 1 0 1 0 0 0 0 1 0
```

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

```r
sample(1:100, size = 6, replace = FALSE)
#> [1] 26 31 69 12 48 21
```

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)
  )
```

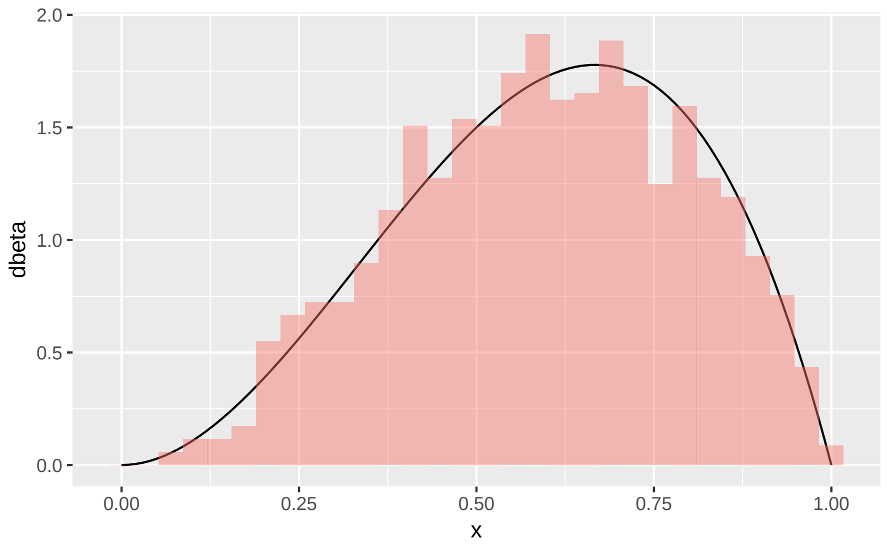


Figure 1: Beta(3,2) random numbers

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

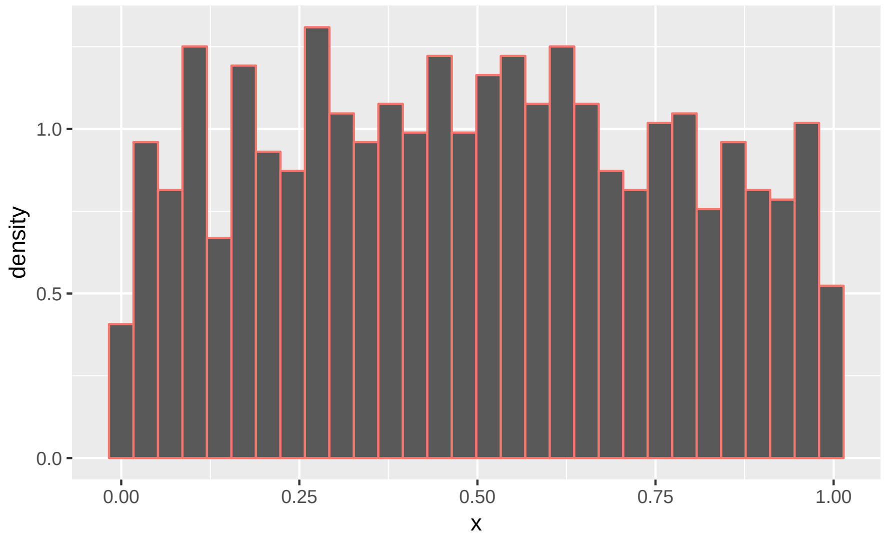
Then $u_1, u_2, \ldots, u_n \sim unif(0,1)$

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     $x_i = (a x_{i-1} + c) \mod m$;
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))
```



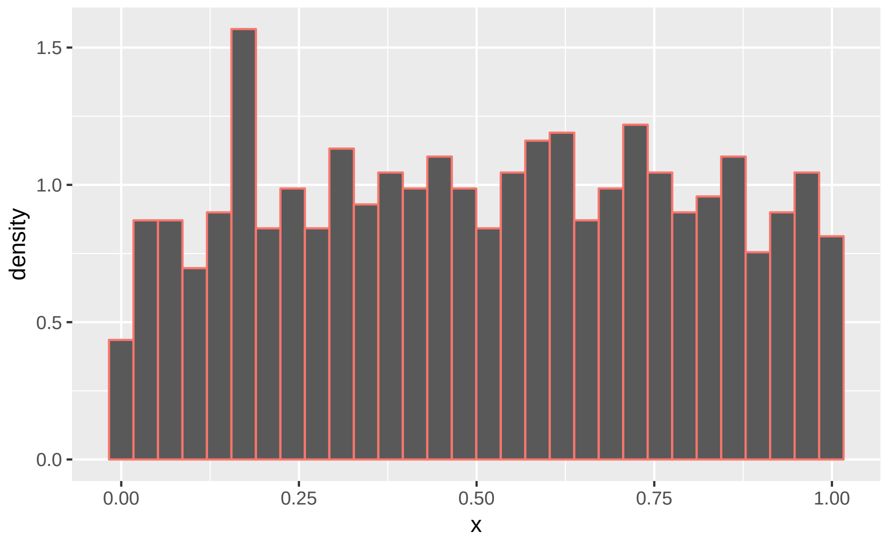
1.2.2 Multiplicative congruential generator

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 $x_0 = 0$.

```
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

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

□

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
  )
```

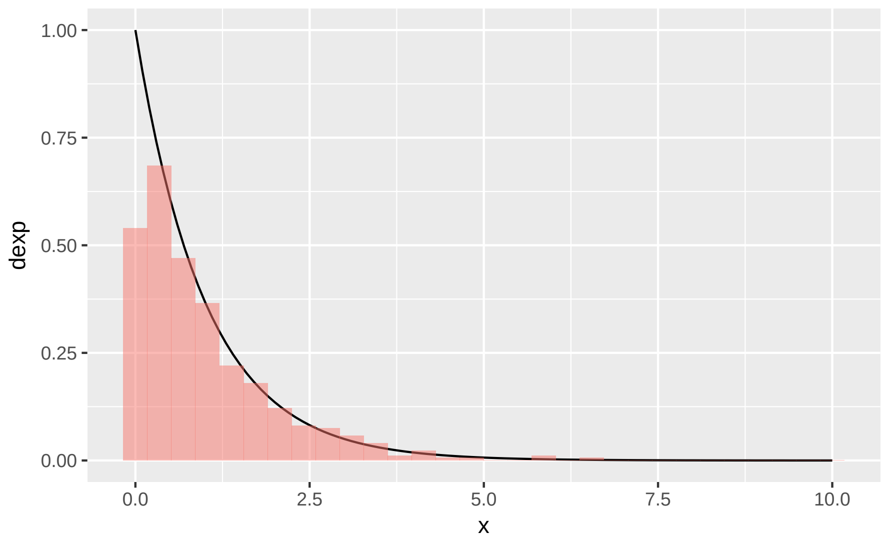


Figure 1.1: Inverse Transformation: Exp(1)

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.

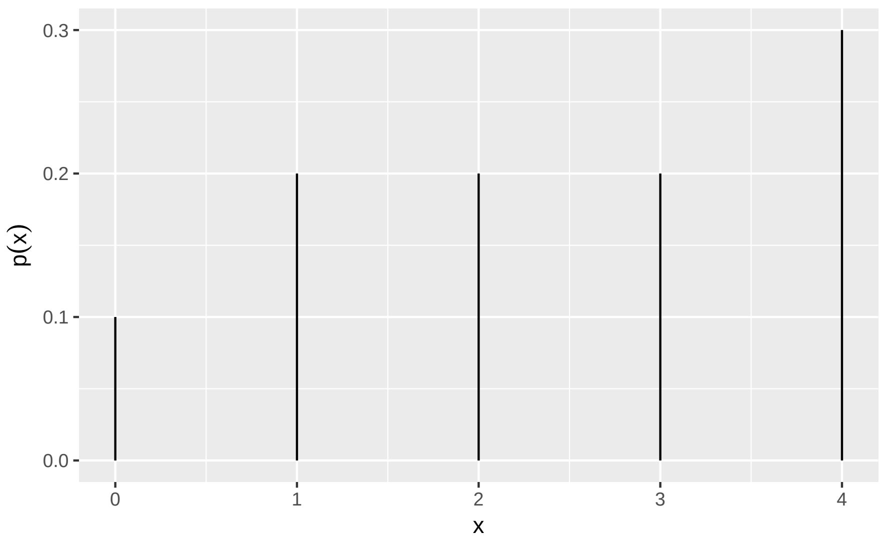


Figure 1.2: Probability Mass Function

Then we have the cdf

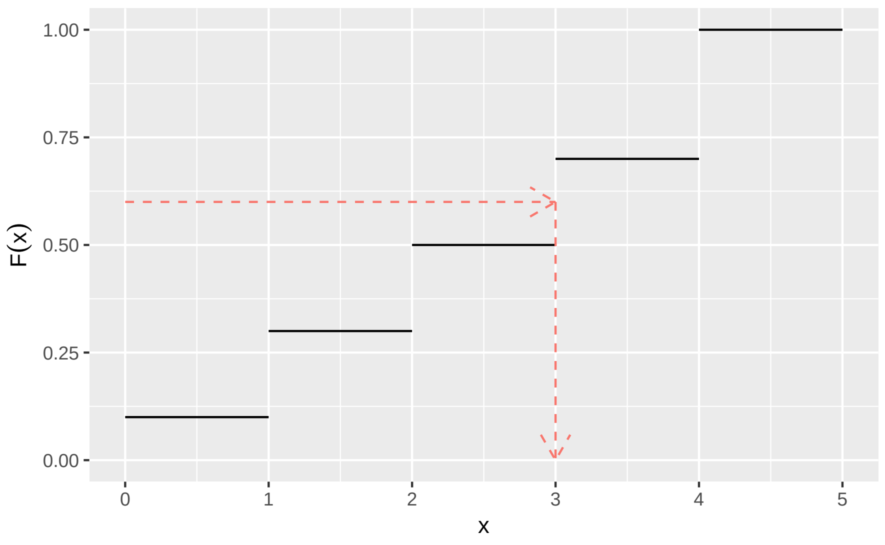


Figure 1.3: CDF of the Discrete Random Variable: Illustration for discrete case

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()
}
```

```r
tibble(x = rcustom(100)) %>%
  count(x) %>%
  mutate(n = n / sum(n)) %>%
  bind_cols(px = pmf %>% select(p)) %>% # pmf table
  gather(-x, key = "key", value = "value") %>%
  ggplot(aes(x = x, fill = key)) +
  geom_bar(aes(y = value), stat = "identity", position = "dodge", width = .2) +
  scale_fill_discrete(
    name = "Compare",
    labels = c("InvTrans", expression(p(x)))
  ) +
  ylab("prob")
```

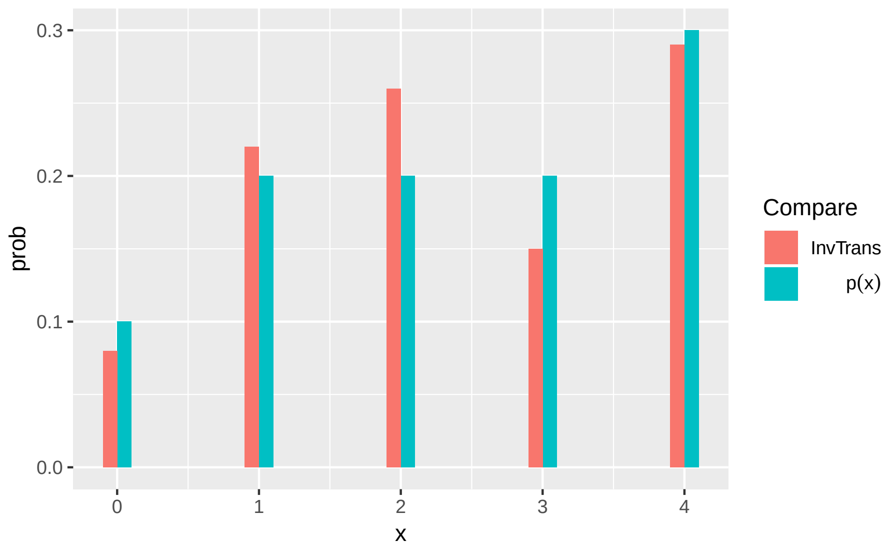


Figure 1.4: Generated discrete random numbers

See Figure 1.4. Comparing random numbers to true pmf, 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

$$spt f \subseteq spt g$$

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

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

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

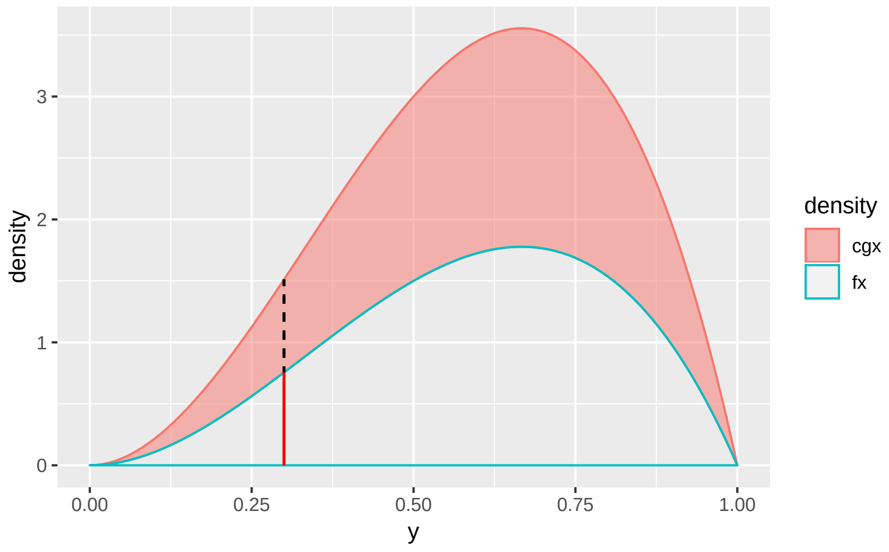


Figure 1.5: Property of AR method

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

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

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)**.** *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

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

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

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

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*

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

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

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

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")
  )
```

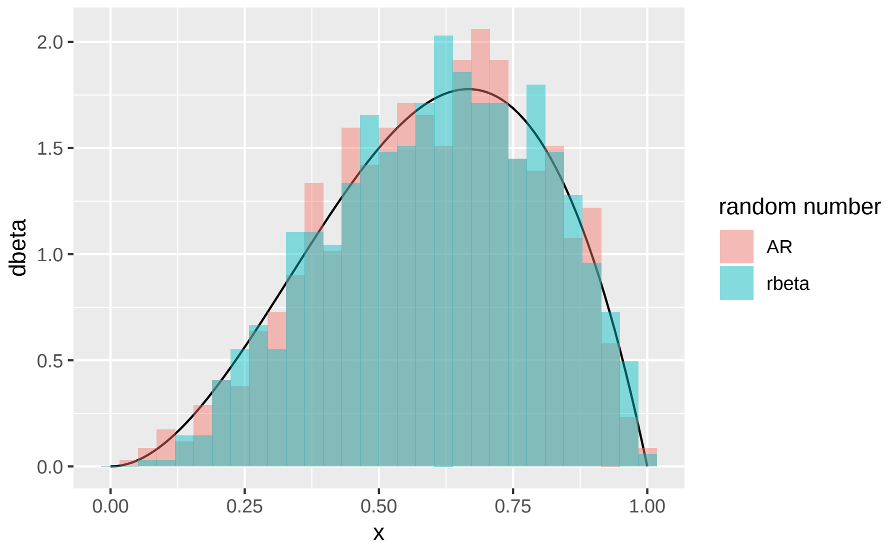


Figure 1.6: Beta(3,2) Random numbers from each function

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.

```r
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")
  )
```

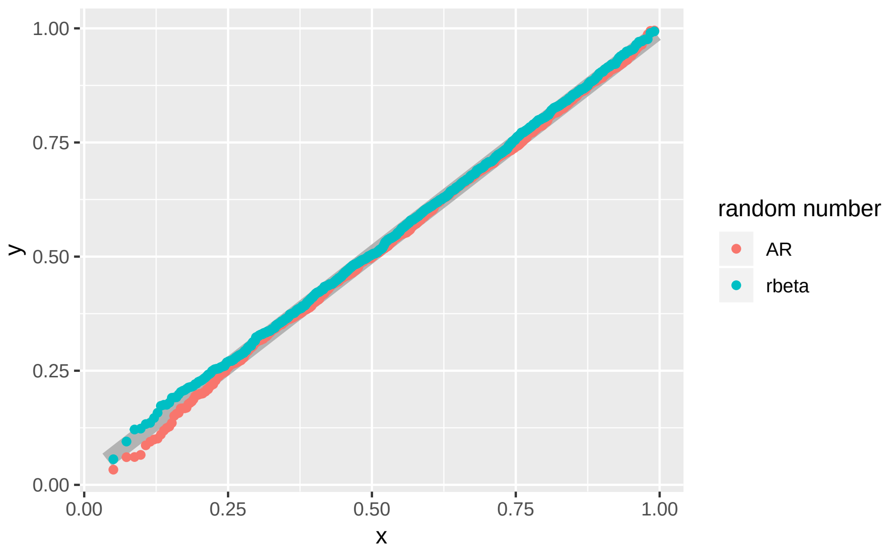


Figure 1.7: Q-Q plot for Beta(3,2) random numbers

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)
}
```

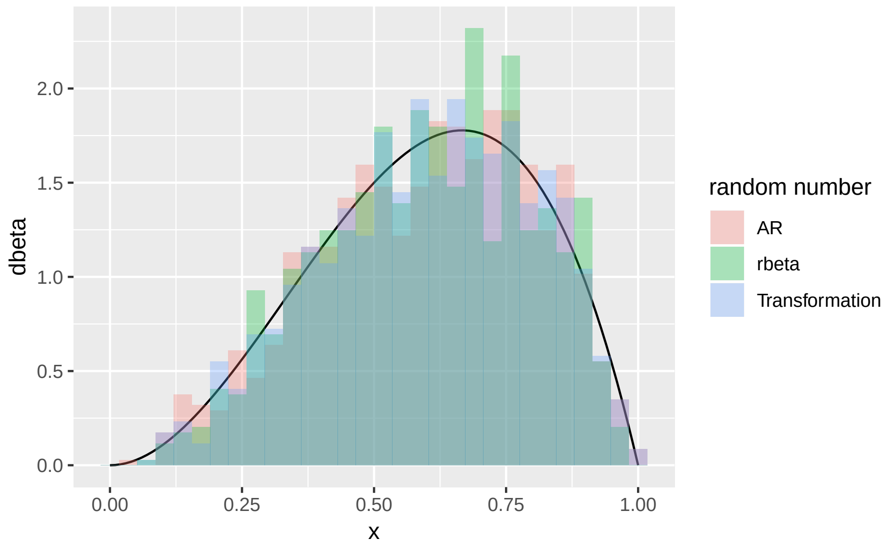


Figure 1.8: Beta(3,2) Random numbers from each function, including transformation method

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

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
  )
```

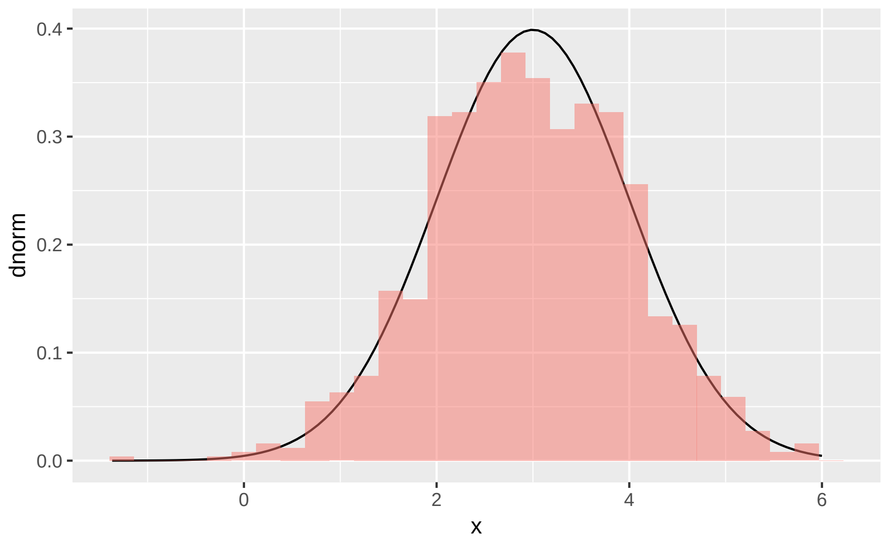


Figure 1.9: Normal random numbers by Box-Muller transformation

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*

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

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
  )
```

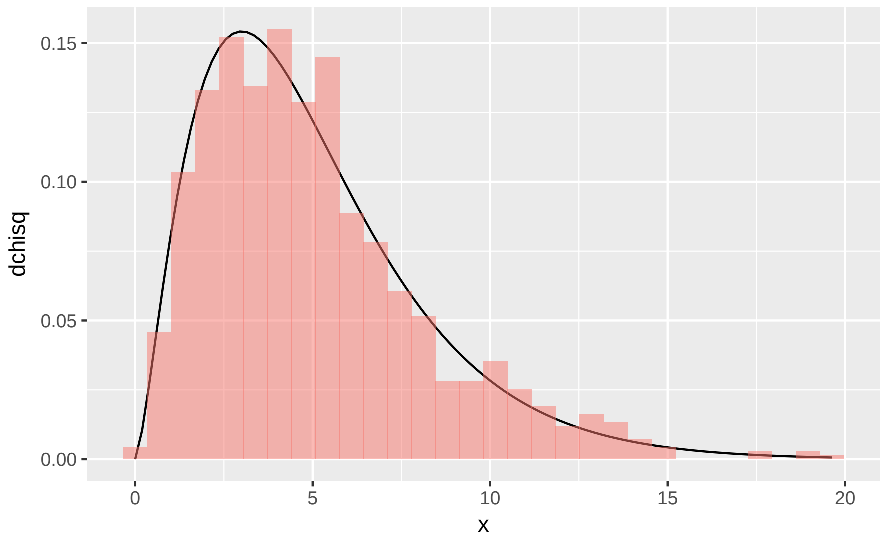


Figure 1.10: χ^2 random numbers from Normal sums

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

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

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.

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

To combine samples easily, 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)
  k * x1 + (1 - k) * x2
}
```

Try various p_1, from 0.1 to 1. We would loop and combine by `dplyr::bind_rows()`. Reason for binding is to plot.

```
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)
    )
  }
```

Output is long data format. So we can easily draw a line for each group (`key`).

```
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")
```

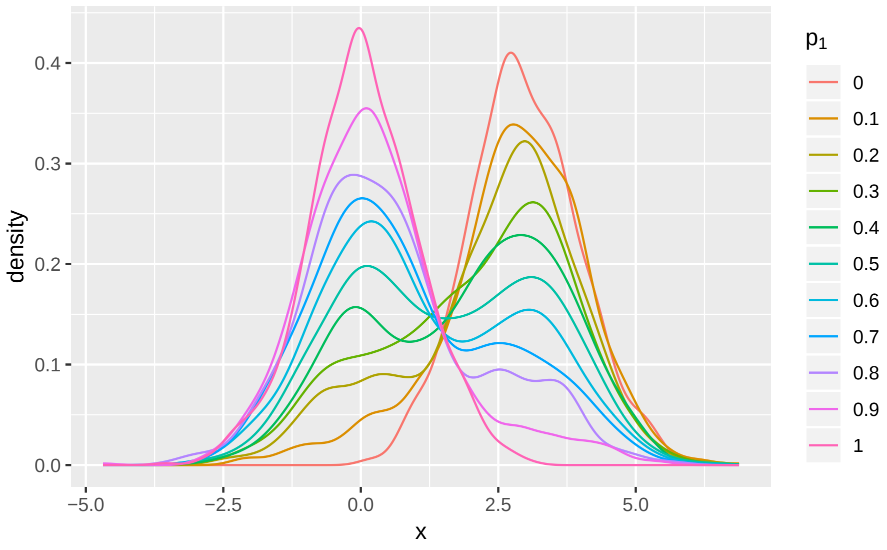


Figure 1.11: Mixture normal random number for each mixing probability

As p_1 becomes larger to 1, the distribution becomes $N(0,1)$. On the contrary, $p_1 = 0$ results in $N(3,1)$.

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()
}
```

```
# 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

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

```
(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
```

```
mvn3 %>%
  GGally::ggpairs(
    lower = list(continuous = GGally::wrap(gg_scatter, size = 1))
  )
#> Registered S3 method overwritten by 'GGally':
#>   method from
#>   +.gg   ggplot2
```

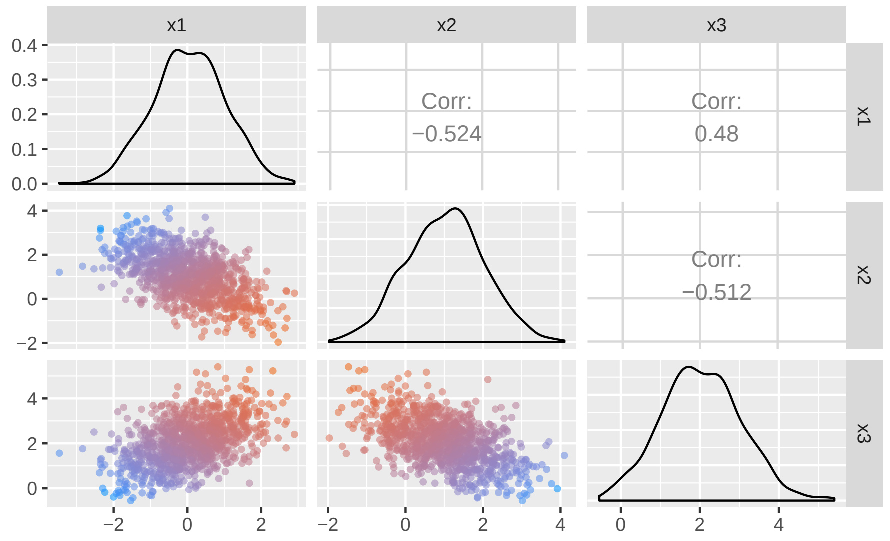


Figure 1.12: Multivariate normal random vector - spectral decomposition method

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.

```
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()
}
```

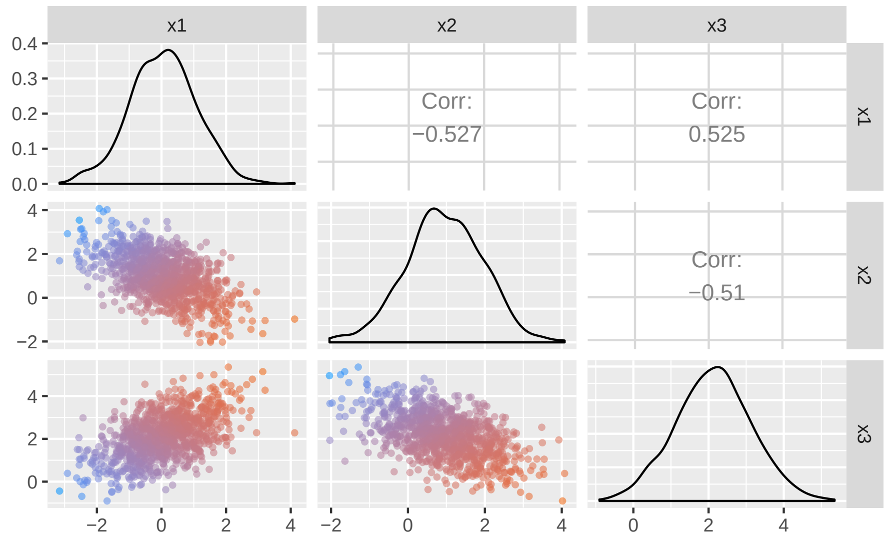


Figure 1.13: Multivariate normal random vector - svd

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()
}
```

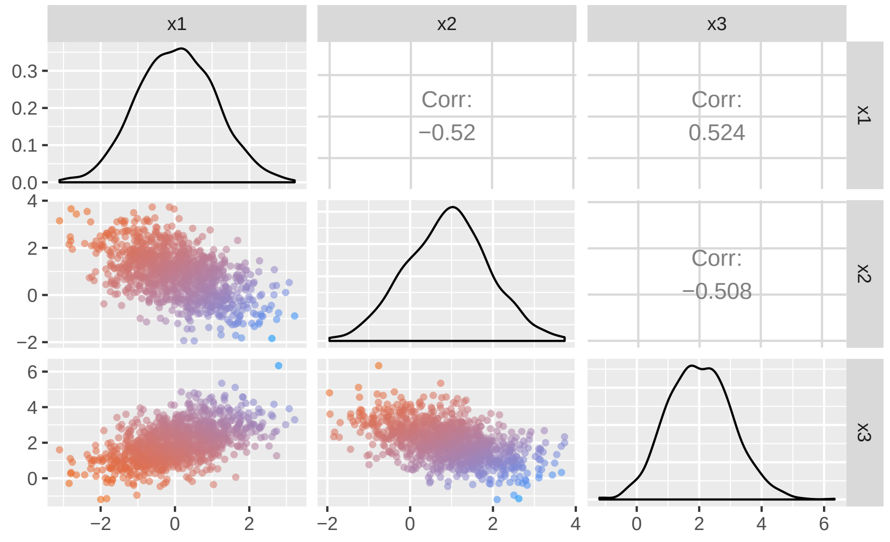


Figure 1.14: Multivariate normal random vector - cholesky decomposition

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.

Definition 1.8 (Discrete Time Markov Chain)**.** $\{X_n : n = 0, 1, 2, \ldots\}$ is a Discrete time markov chain on S if and only if

1. S is at most countable

2. Markov property $P(X_{n+1} = j \mid X_n = i, X_{n-1} = i_{n-1}, \ldots, X_0 = i_0) = P(X_{n+1} = j \mid X_n = i) = P_{ij}$

If P_{ij} is fixed, then $\{X_n\}$ is called time homogeneous. Otherwise, it is called nonhomogeneous.

Definition 1.9 (Random walk model)**.** Let $\{Y_n : n \in \mathbb{N}\}$ be an IID process on S s.t.

$$P(Y_n = k) = p_k$$

Define

$$S_n := \begin{cases} 0 & n = 0 \\ S_0 + Y_1 + \cdots + Y_n & n \in \mathbb{N} \end{cases}$$

1.8.1 Gambler's ruin model

Definition 1.10 (Gambler's ruin model)**.** Let $\{Y_n : n \in \mathbb{N}\}$ be a process on $\{-1, 1\}$ s.t.

$$P(Y_n = 1) = p, \quad P(Y_n = -1) = 1 - p$$

Define

$$X_n := \begin{cases} a & n = 0 \\ a + Y_1 + \cdots + Y_n & n \in \mathbb{N} \end{cases}$$

Example 1.8 (Gambling with coin)**.** Suppose that A and B each start with a stake of \$10, and bet \$1 on consecutive coin flips. The game ends when either one of the players has all the money. Let S_n be the fortune of player A at time n Then $\{S_n, n \geq 0\}$ is a symmetric random walk with absorbing barriers at 0 and 20. Simulate a realization of the process $\{S_n, n \geq 0\}$ and plot S_n vs the time index from time 0 until a barrier is reached.

Here we have

$$P(Y_n = 1) = P(Y_n = -1) = \frac{1}{2}$$

$$S_n := \begin{cases} 10 & n = 0 \\ 10 + Y_1 + \cdots + Y_n & n \in \mathbb{N} \end{cases}$$

```r
gambling <- function(begin = 10, betting = 1, prob = .5) {
  N <- begin * 2
  sa <- begin
  record <- tibble(a = begin, b = begin)
  while(all(record > 0)) {
    sa <- ifelse(runif(1) <= prob, sa + betting, sa - betting)
    record <-
      record %>%
      bind_rows(c(a = sa, b = N - sa))
    if (sa == N) break()
  }
  record %>%
    mutate(idx = 1:n()) %>%
    select(idx, a, b)
}
```

```r
gambling(begin = 10, betting = 1, prob = .5) %>%
  gather(-idx, key = "player", value = "fortune") %>%
  ggplot(aes(x = idx, y = fortune, colour = player)) +
  geom_path() +
  geom_point(alpha = .5, size = 1) +
  geom_hline(yintercept = c(0, 20), col = I("grey")) +
  labs(
    x = "Betting",
    y = "Fortune"
  )
```

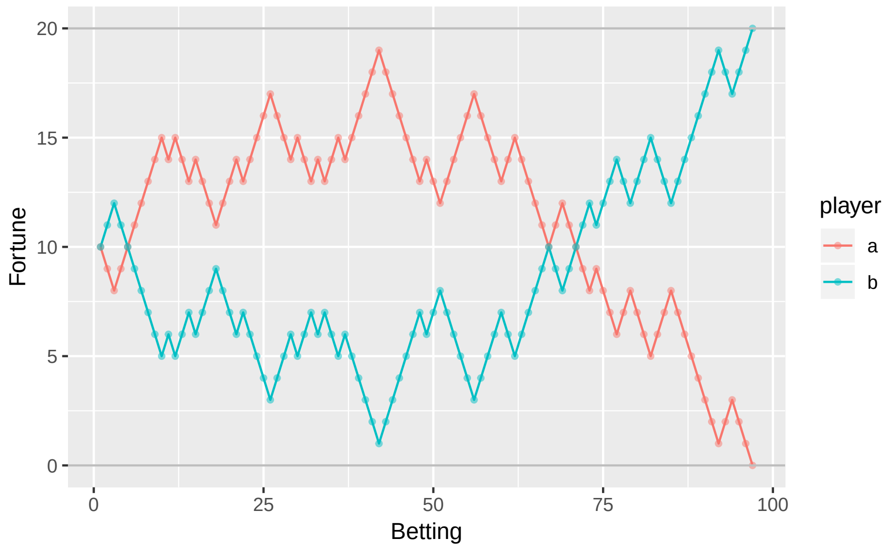


Figure 1.15: Sample path of Gambler's ruin model

In Figure 1.15, we can see the result of process with $p = \frac{1}{2}$. In fact, this process with probability 0.5 is also called *symmetric random walk*.

1.8.2 Homogeneous poisson process

Definition 1.11 (Count process)**.** A stochastic process $\{N(t) : t \geq 0\}$ where $N(t)$ is total number of events that occur by time t is called counting process.

1. $N(t) \geq 0$
2. $N(t) \in \mathbb{Z}$
3. $s \leq t \Rightarrow N(s) \leq N(t)$
4. For $s < t$, $N(t) - N(s) =$ the number of events that occur in $(s, t]$

Poisson process is one of this counting process.

Definition 1.12 (Poisson process)**.** The counting process $\{N(t), t \geq 0\}$ is said to be a Poisson process with rate $\lambda > 0$

1. $N(0) = 0$
2. $N(t) \perp\!\!\!\perp N(t + s) - N(t)$
3. Distribution of $N(t + s) - N(t)$ is the same for all values of t

4. $\lim\limits_{h\to 0} \frac{P(N(h)=1)}{h} = \lambda$

5. $\lim\limits_{h\to 0} \frac{P(N(h)\geq 2)}{h} = 0$

Remark.

$$\{N(t), t \geq 0\} \sim PP(\lambda) \Rightarrow N(t) \sim Poisson(\lambda t)$$

We can generate Poisson process using this relationship. However, it is slow. Thus, we find another way.

Theorem 1.5 (Campbell's Theorem)**.** *Let $\{N(t), t \geq 0\} \sim PP(\lambda)$, let X_t be the interarrival time, and let S_t be the waiting time until t-th event, i.e. $S_t := \sum_{i=1}^t X_i$. Then*

$$S_1, S_2, \ldots, S_n \mid N(t) = n \overset{d}{=} (U_{(1)}, U_{(2)}, \ldots, U_{(n)})$$

where $U_i \sim unif(0,t)$.

This Campbell's theorem gives solution to the PP generation.

Algorithm 7: Fast algorithm for Poisson Process

```
  input : end time $T$
1 Generate $N \sim Poisson(\lambda T)$;
2 For $N$, generate $U_1, \ldots, U_N \overset{iid}{\sim} unif(0,T)$;
3 Sort $U_1, \ldots, U_N$ in ascending order, i.e. $\{U_{(1)}, \ldots, U_{(N)}\}$;
4 Set $S_1 = U_{(1)}, \ldots, S_N = U_{(N)}$;
  output: $\begin{bmatrix} 1 & S_1 \\ 2 & S_2 \\ \cdots & \cdots \\ N & S_n \end{bmatrix}$
          $\Rightarrow \{N(S_n)\} \sim PP(\lambda)$ on $[0,T]$
```

```r
rpp <- function(lambda, t0) {
  N <- rpois(1, lambda = lambda * t0)
  tibble(sn = runif(N) * t0) %>%
    arrange(sn) %>% # arrival time
    mutate(pp = 1:n()) # N(sn) ~ PP
}
```

```r
rpp(lambda = 1, t0 = 50) %>%
  mutate(
    true_mean = sn, # sn * lambda
    true_sd = sqrt(sn) # sn * lambda
  ) %>%
  ggplot(aes(x = sn)) +
  geom_ribbon(
    aes(ymin = true_mean - true_sd, ymax = true_mean + true_sd),
    fill = "grey70",
    alpha = .5
  ) +
  geom_line(aes(y = true_mean), col = I("white"), size = 2) +
  geom_path(aes(y = pp)) +
  labs(
    x = "t",
    y = expression(N(t))
  )
```

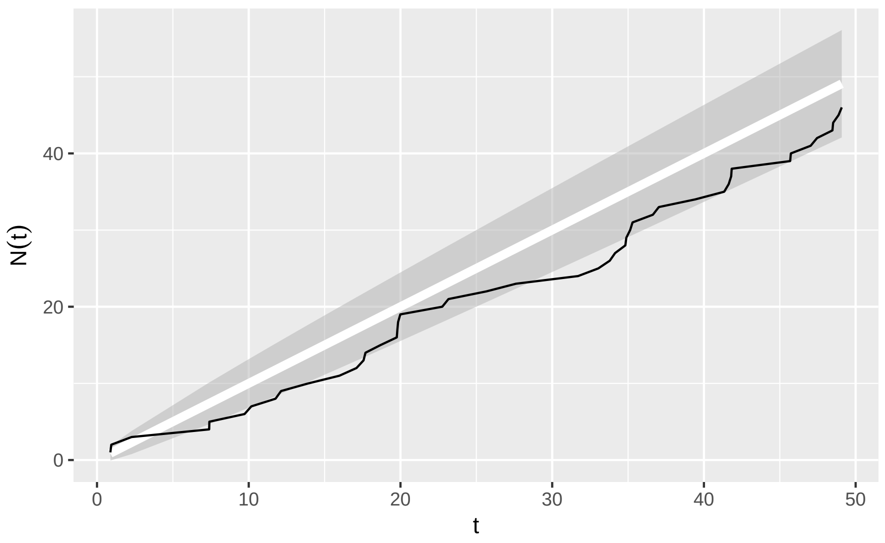


Figure 1.16: Sample path of Poisson process

1.8.3 Nonhomogeneous poisson process

Last section, what we have seen was homogeneous PP whose distribution does not depend on t. On the other hand, this condition can be broken.

Definition 1.13 (Nonhomogeneous Poisson Process)**.** The counting process $\{N(t), t \geq 0\}$ is said to be a Nonhomogeneous Poisson process with rate $\lambda(t) > 0$ if the third condition does not hold.

1. $N(0) = 0$
2. $N(t) \perp\!\!\!\perp N(t+s) - N(t)$
3. $\lim\limits_{h \to 0} \frac{P(N(h)=1)}{h} = \lambda$
4. $\lim\limits_{h \to 0} \frac{P(N(h)\geq 2)}{h} = 0$

$\lambda(t)$ is called intensity at time t.

$m(t) := \int_0^t \lambda(s)ds, t \geq 0$ is called mean-value function.

We can generate this NPP by Bernoulli splitting, which is called *thining approach.*

Lemma 1.1 (Thinning approach)**.** *Choosing λ s.t. $\forall t \leq T \lambda(t) \leq \lambda$. If an event of $PP(\lambda)$ counted with*

$$p(t) = \frac{\lambda(t)}{\lambda}$$

then the process follows $NPP(\lambda(t))$ on $[0, T]$

Algorithm 8: Thinning algorithm

```
1 Set $t = 0, N = 0$;
2 repeat
3     Generate $Y \sim Exp(\lambda)$;
4     Set $t \leftarrow t + Y$;
5     Generate $U \sim unif(0, 1)$;
6     if $U \leq \frac{\lambda(t)}{\lambda}$ then
7         Set $N \leftarrow N + 1$;
8         Set $S(N) \leftarrow t$;
9 until $t > T$;
```

```r
npp <- function(lambda, t0, intensity) {
  t <- 0
  N <- 0
  S <- tibble(t = t, N = N)
  while (t <= t0) {
    t <- t + rexp(1, lambda)
    U <- runif(1)
    if ( U <= intensity(t) / lambda) {
      N <- N + 1
      S <-
        S %>%
        bind_rows(tibble(t = t, N = N))
    }
  }
  S %>% slice(-1)
}
```

```r
intensity <- function(x) {
  tt <- x %% 480
  rate <-
    case_when(
      tt >= 0 && tt <= 120 ~ .5,
      tt > 120 && tt <= 240 ~ 1,
      tt > 240 && tt <= 360 ~ 2,
      tt > 360 && tt <= 480 ~ 1.5
    )
  rate
}
#--------------------------
npp(2, 480, intensity = intensity) %>%
  ggplot(aes(x = t, y = N)) +
  geom_path()
```

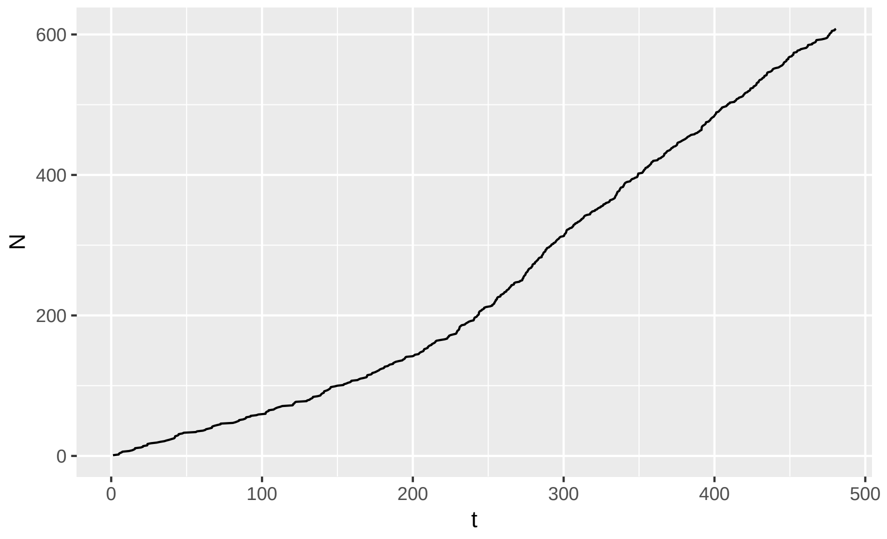


Figure 1.17: Sample path of nonhomogeneous poisson process

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

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

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*

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

Let g *be a measurable function. Then*

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

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

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

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 $x > 0$.

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

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

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()
  )
}
```

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

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.

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

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

```
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.

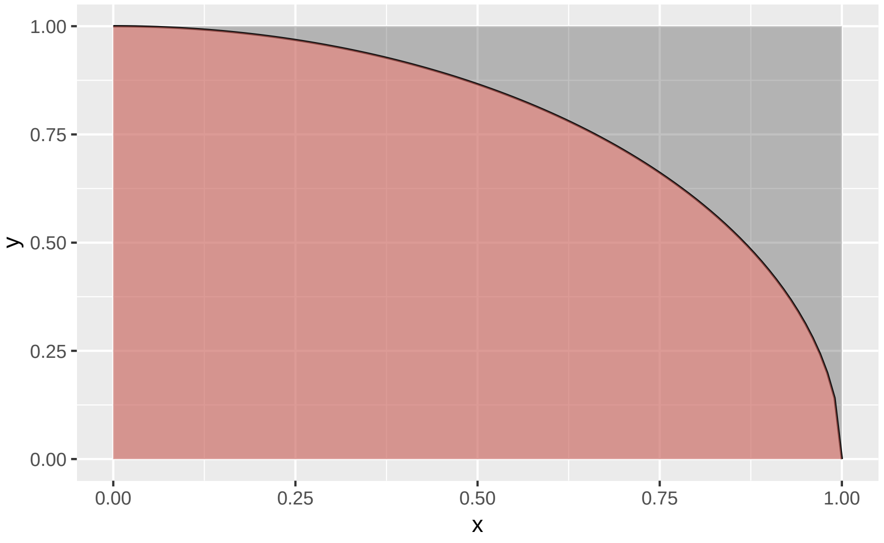


Figure 2.1: Hit-or-Miss

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 $Y_i \leq \sqrt{1 - X_i^2}$ estimates $\frac{\pi}{4}$.

```r
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 variance of sample mean is $Var(\overline{g(X)}) = \frac{Var(g(X))}{N}$. This property is one of estimating variance of $\hat{\theta}$.

$$\widehat{Var}(\hat{\theta}) = \frac{1}{N}\left(\frac{1}{N}\sum_{i=1}^{N}(g(X_i) - \overline{g(X_i)})\right) = \frac{1}{N^2}\sum_{i=1}^{N}(g(X_i) - \overline{g(X_i)}) \tag{2.3}$$

For example,

```r
tibble(x = runif(10000)) %>%
  summarise(mc_pi = var(circ(x)) / 10000)
#> # A tibble: 1 x 1
#>       mc_pi
#>       <dbl>
#> 1 0.0000795
```

However, this *variance of sample mean* is used in situation when we are in sample limitation situation. We do not have to stick to this. Now, Generating samples as many as we want is possible. So we try another approach: *parametric bootstrap*.

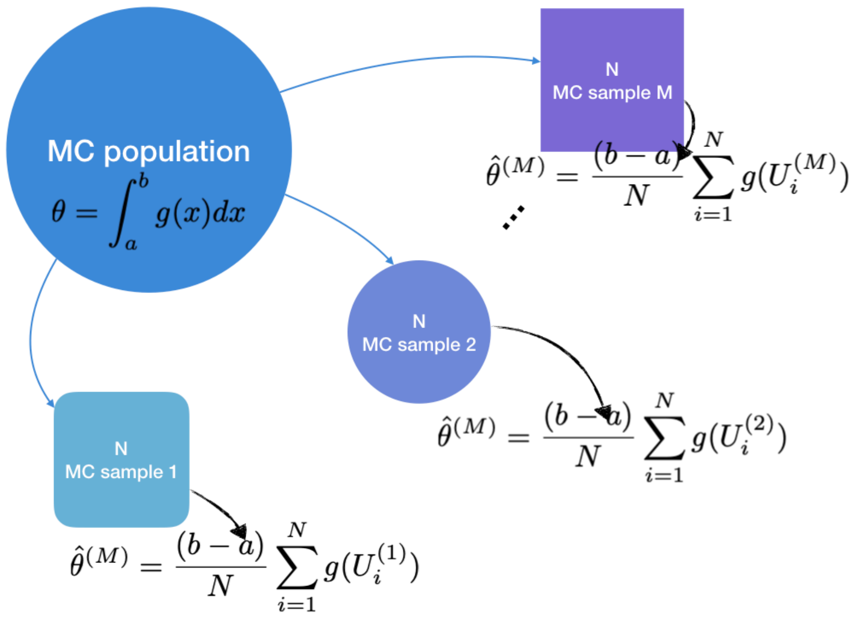

MC population
$\theta = \int_a^b g(x)dx$
N
MC sample M
$\hat{\theta}^{(M)} = \frac{(b-a)}{N} \sum_{i=1}^N g(U_i^{(M)})$
N
MC sample 2
$\hat{\theta}^{(M)} = \frac{(b-a)}{N} \sum_{i=1}^N g(U_i^{(2)})$
N
MC sample 1
$\hat{\theta}^{(M)} = \frac{(b-a)}{N} \sum_{i=1}^N g(U_i^{(1)})$

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 9: 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))
  )
}
```

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

```
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])
```

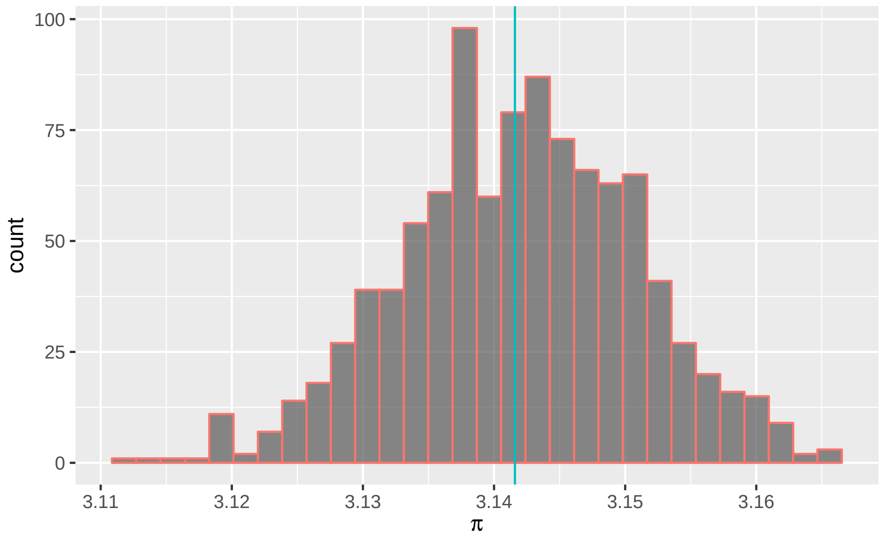


Figure 2.3: Empirical distribution of $\hat{\pi}$ by simple MC

As in Algorighm 9, we can compute the variance as below.

```
(mc_var <-
  pi_mc[,
        .(mc_variance = var(mc_pi))])
#>    mc_variance
#> 1:    8.09e-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")
  )
```

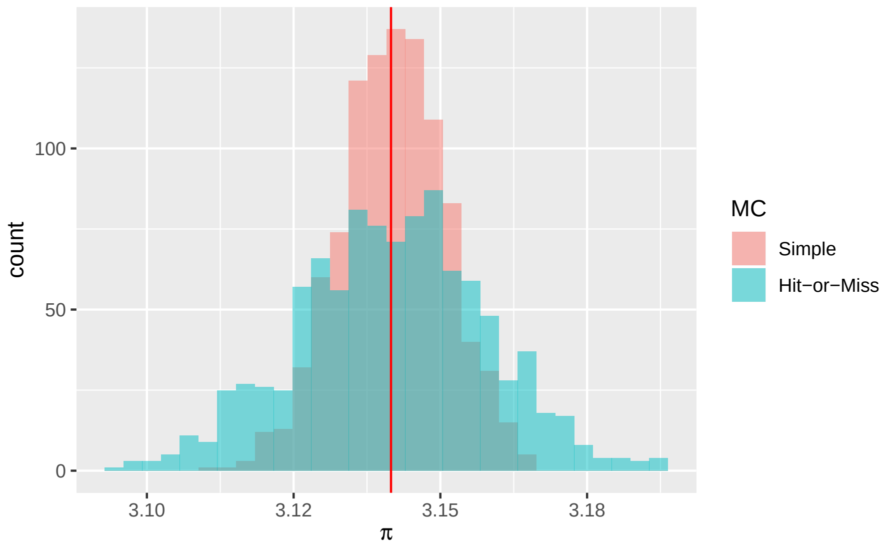


Figure 2.4: Simple MC and Hit-or-Miss MC

```
(hit_var <-
  pi_hit[,
         .(hit_variance = var(hitormiss))])
#>    hit_variance
#> 1:      0.000258
```

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.

Table 2.2: Simple MC versus Hit-or-Miss

SimpleMC	Hit-or-Miss	SimpleMCefficiency
0	0	TRUE

2.3 Variance Reduction

Consider Equation (2.3) based on $Var(\hat{\theta}) = \frac{\sigma^2}{N}$. This variance can always reduced by adding N. But we want to reduce variance less computationally.

2.3.1 Antithetic Variables

Consider correlated random variables U_1 and U_2. Then we have

$$Var\bigg(\frac{U_1 + U_2}{2}\bigg) = \frac{1}{4}\Big(Var(U_1) + Var(U_2) + 2Cov(U_1, U_2)\Big)$$

See the last term $Cov(U_1, U_2)$. If we generate U_{i1} and U_{i2} negatively correlated, we can get reduced variance than previous i.i.d. sample

$$Var\bigg(\frac{U_1 + U_2}{2}\bigg) = \frac{1}{4}\Big(Var(U_1) + Var(U_2)\Big)$$

Lemma 2.1. *U and $1 - U$ are identically distributed, but* negatively correlated.

1. *$U \sim unif(0, 1) \Leftrightarrow 1 - U \sim unif(0, 1)$*
2. *$Corr(U, 1 - U) = -1$*

This is well-known property of uniform distribution. Instead of generating N uniform numbers, try $\frac{N}{2}$ U_i and make corresponding $\frac{N}{2}$ $1 - U_i$. This sequence becomes negatively correlated, so we can reduce the variance as mentioned.

When can we replace previous numbers with these *antithetic variables*? We usually plug-in the numbers in some function h to get Monte carlo integration. The thing is, our target is h, not U. $h(U)$ and $h(1 - U)$ should *still be negatively correlated*. Hence, h should be *monotonic function*.

Corollary 2.1. *If $g = g(X_1, \ldots, X_n)$ is monotone, then*

$$Y = g(F_X^{-1}(U_1), \ldots, F_X^{-1}(U_n))$$

and

$$Y' = g(F_X^{-1}(1 - U_1), \ldots, F_X^{-1}(1 - U_n))$$

are negatively correlated.

Algorithm 10: Variance of $\hat{\theta}$ using antithetic variables

```
   input  : $h$ : monotonic
1  for $m \leftarrow 1$ to $M$ do
2      Generate $U_{1,1}^{(m)}, \ldots, U_{\frac{N}{2},1}^{(m)} \overset{iid}{\sim} unif(0,1)$;
3      Set $U_{i,2}^{(m)} := 1 - U_{i,1}^{(m)} \overset{iid}{\sim} unif(0,1)$;
4      $\{U_i^{(m)}\}_1^N = \{U_{1,1}^{(m)}, \ldots, U_{\frac{N}{2},2}^{(m)}\}$;
5      $\hat{\theta}^{(j)} = \frac{1}{N}\sum h(U_i^{(j)})$;
6  end
7  $\bar{\hat{\theta}} = \frac{1}{M}\sum \hat{\theta}^{(j)}$;
8  $\widehat{Var}(\hat{\theta}) = \frac{1}{M-1}\sum(\hat{\theta}^{(j)} - \bar{\hat{\theta}})^2$;
   output: $\widehat{Var}(\hat{\theta})$
```

Check again Example 2.1. We have try to calculate

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

using simple monte carlo. To make the support $(0, 1)$, let $y = \frac{t}{x}$ be a change of variable. Then

$$\begin{aligned}\int_0^x \exp\left(-\frac{t^2}{2}\right) dt &= \int_0^1 x \exp\left(-\frac{(xy)^2}{2}\right) dy \\ &\approx \frac{1}{N}\sum_{i=1}^{N} x \exp\left(-\frac{(xU_i)^2}{2}\right)\end{aligned}$$

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

Consider $\Phi(2)$.

```
phi2 <-
  mc_data(runif)[,
                 .(p2 = phiunif(u, y = 2)),
                 keyby = sam]
```

Now apply antithetic variables.

```
phi2_anti <-
  mc_data(runif, N = 10000 / 2)[,
                                u2 := 1 - u] %>%
  melt(id.vars = "sam", value.name = "U") %>%
  .[,
    .(anti_p2 = phiunif(U, y = 2)),
    keyby = sam]
```

```
phi2[phi2_anti] %>%
  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 = pnorm(2), col = I("red")) +
  scale_fill_discrete(
    name = "MC",
    labels = c("Simple", "Antithetic")
  )
```

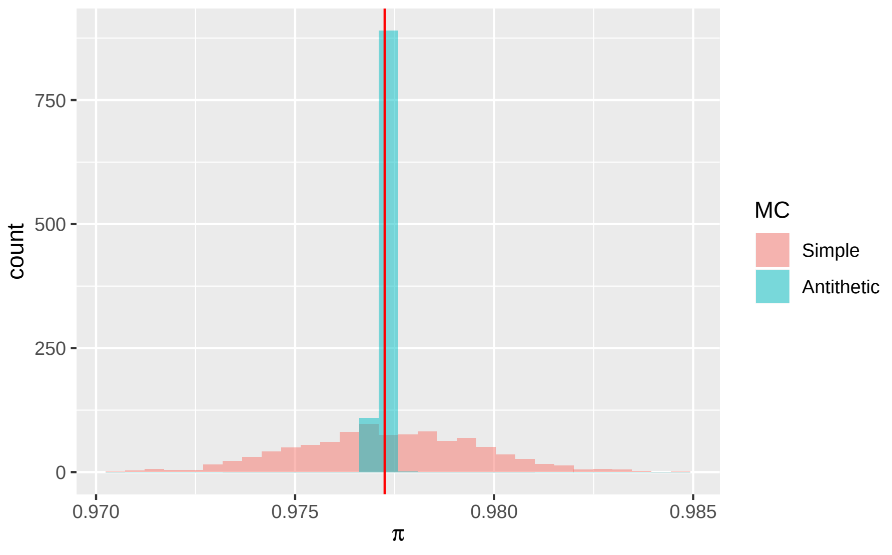


Figure 2.5: Use of antithetic variables

Obviously, variance has been reduced.

```
phi2[phi2_anti] %>%
  melt(id.vars = "sam", variable.name = "hat") %>%
  .[,
    .(variance = var(value)),
    by = hat]
#>        hat variance
#> 1:      p2  5.2e-06
#> 2: anti_p2  1.5e-08
```

2.3.2 Control Variates

Recall that we are trying to estimate $\theta = Eg(X)$ here in MC integration. Consider other output random variable. Suppose that $\mu_f \equiv Ef(Y)$ is known. It is obvious that

$$\hat{\theta}_c = g(X) + c\Big(f(Y) - \mu_f\Big) \tag{2.4}$$

is an unbiased estimator for θ for any $c \in \mathbb{R}$. Then we have

$$Var\hat{\theta}_c = Varg(X) + c^2 Varf(X) + 2cCov(g(X), f(X)) \tag{2.5}$$

Recall that our goal is to minimize this $Var\hat{\theta}_c$. What value of c is to be determined? Note that Equation (2.5) is quadratic function of c.

$$\begin{aligned} Var\hat{\theta}_c &= Varf(X)c^2 + 2cCov(g(X), f(X)) + Varg(X) \\ &= Varf(X)\left(c + \frac{Cov(g(X), f(X))}{Varf(X)}\right)^2 + Varg(X) - \frac{Cov(g(X), f(X))^2}{Varf(X)} \end{aligned} \tag{2.6}$$

From Equation (2.6), the variance is minimized at

$$c^* = -\frac{Cov(g(X), f(X))}{Var f(X)} \tag{2.7}$$

with minimum variance

$$Var\hat{\theta}_{c^*} = Var g(X) - \frac{Cov(g(X), f(X))^2}{Var f(X)}$$

By this, we can reduce the variance of estimation as much as possible (using $f(X)$). Here, $f(X)$ is called a *control variate* for $g(X)$.

Algorithm 11: Variance of $\hat{\theta}$ using control variables

```
  input  : $g$, control variate $f$ with mean $\mu_f$
1 for $m \leftarrow 1$ to $M$ do
2     Generate $U_1^{(m)}, \ldots, U_N^{(m)} \overset{iid}{\sim} unif$;
3     Set $g = g(U_i)$ and $f = f(U_i)$;
4     Compute $\hat{c}^{*(m)} = -\frac{\widehat{Cov}(g,f)}{\widehat{Var}(f)}$;
5     $\hat{\theta}_{c^*}^{(j)} = g + c^{*(m)}(f - \mu_f)$;
6 end
7 $\bar{\hat{\theta}} = \frac{1}{M}\sum \hat{\theta}_{c^*}^{(j)}$;
8 $\widehat{Var}(\hat{\theta}) = \frac{1}{M-1}\sum(\hat{\theta}_{c^*}^{(j)} - \bar{\hat{\theta}})^2$;
  output: $\widehat{Var}(\hat{\theta})$
```

Example 2.3 (Variance reduction by control variate)**.** Apply each simple MC, antithtic variate, and control variate to

$$\int_0^1 e^x dx$$

```r
N <- 100
M <- 1000
```

Denote that the true value is

$$\int_0^1 e^x dx = e - 1 = 1.718$$

We might compare each estimate to this.

Solution (Simple MC). We only need $U \sim unif(0,1)$.

$$\begin{aligned}\theta &= \int_0^1 e^x dx \\ &= \int_0^1 e^x I_{(0,1)}(x) dx \\ &\approx \frac{1}{N}\sum_{i=1}^N e^{u_i}, \qquad u_i \overset{iid}{\sim} unif(0,1)\end{aligned} \tag{2.8}$$

```r
theta_sim <-
  mc_data(runif, N = N, M = M)[,
                               .(mc = mean(exp(u))),
                               keyby = sam]
```

```
#>         sam   mc
#>    1:    s1 1.70
#>    2:    s2 1.71
#>    3:    s3 1.76
#>    4:    s4 1.71
#>    5:    s5 1.73
#>   ---
#>  996:  s996 1.79
#>  997:  s997 1.82
#>  998:  s998 1.68
#>  999:  s999 1.73
#> 1000: s1000 1.76
```

Solution (Antithetic variate). For $N' = \frac{N}{2}$,

Consider $u_1, \ldots, u_{N'} \overset{iid}{\sim} unif(0,1)$ and $1 - u_1, \ldots, 1 - u_{N'} \overset{iid}{\sim} unif(0,1)$.

See Equation (2.8). Then we can compute antithetic estimator by

$$\begin{aligned}
\hat{\theta}_A &= \frac{1}{N} \sum_{i=1}^{N/2} \left(e^{u_i} + e^{1-u_i} \right) \\
&= \frac{1}{N/2} \sum_{i=1}^{N/2} \left(\frac{e^{u_i} + e^{1-u_i}}{2} \right) \\
&= \text{sample mean}
\end{aligned}$$

```r
theta_anti <-
  mc_data(runif, N = N / 2, M = M)[,
                                   u2 := 1 - u][,
                                                .(anti = mean((exp(u) + exp(u2)) / 2)),
                                                keyby = sam]
```

Now look at the results of the two.

```r
theta_sim[theta_anti] %>%
  melt(id.vars = "sam", variable.name = "simul", value.name = "integral") %>%
  ggplot(aes(x = integral, fill = simul)) +
  geom_histogram(bins = 30, position = "identity", alpha = .5) +
  xlab(expression(theta)) +
  geom_vline(xintercept = exp(1) - 1, col = I("red")) +
  scale_fill_discrete(
    name = "MC",
    labels = c("Simple", "Antithetic")
  )
```

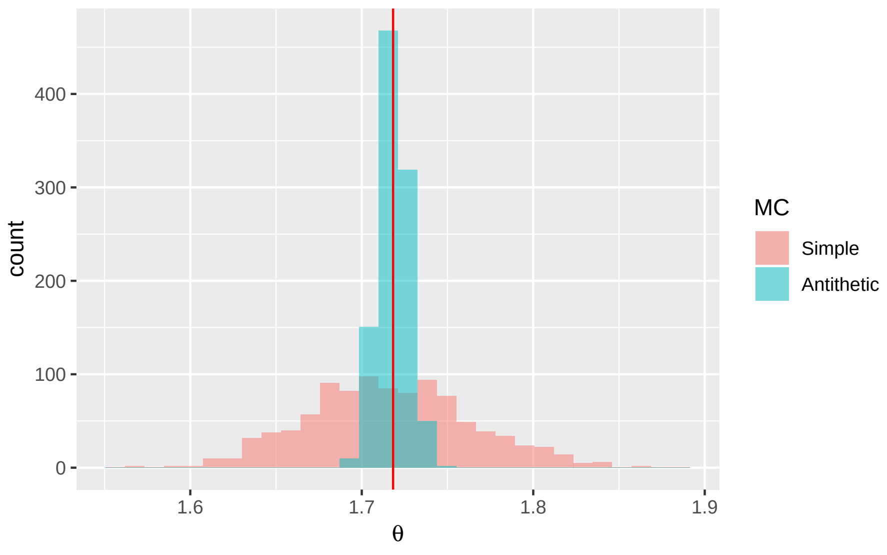


Figure 2.6: Antithetic variate estimator

It is clear that antithetic variate have reduced variance.

Solution (Control variate). Consider

$$g(U) = e^U$$

and

$$f(U) = U$$

with $U \sim unif(0,1)$.

Note that

$$E(U) = \frac{1}{2}$$

Then

$$\hat{\theta}_C = e^U + c\left(U - \frac{1}{2}\right)$$

is an unbiased estimator of $\theta = \int_0^1 e^x dx$.

To reduce variance, we need to set c to be

$$c^* = -\frac{Cov(e^U, U)}{Var(U)}$$

Since we do not know the exact number, we estimate this from each Monte Carlo sample.

```
theta_con <-
  mc_data(runif, N = N, M = M)[,
                               chat := - cov(exp(u), u) / var(u),
                               by = sam][,
                                         .(con = mean(exp(u) + chat * (u - 1 / 2))),
                                         keyby = sam]
```

```
thetahat <- theta_sim[theta_anti][theta_con]
```

```
thetahat %>%
  melt(id.vars = "sam", variable.name = "simul", value.name = "integral") %>%
  ggplot(aes(x = integral, fill = simul)) +
  geom_histogram(bins = 30, position = "identity", alpha = .5) +
  xlab(expression(theta)) +
  geom_vline(xintercept = exp(1) - 1, col = I("red")) +
  scale_fill_discrete(
    name = "MC",
    labels = c("Simple", "Antithetic", "Control")
  )
```

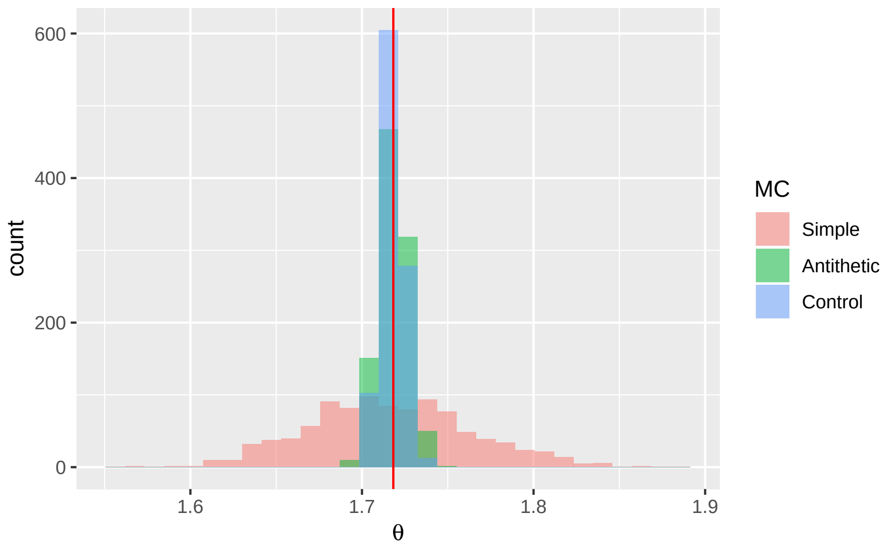


Figure 2.7: Use of Control variable

It looks like control variate have less variance, but what is more important is that both methods successfully have reduced it.

```
thetahat[,
         lapply(.SD, sd),
         ,.SDcols = -"sam"]
#>        mc    anti     con
#> 1: 0.049 0.00872 0.00643
```

2.3.3 Antithetic variate as control variate

Both antithetic variate and control variate reduce variance using covariance between two random variables. Actually, *antithetic variate is a special case of control variate.* See Equation (2.4).

Lemma 2.2. *Control variate estimator is a linear combination of unbiased estimators of* θ*.*

Consider any two unbiased estimator $\hat{\theta}_1$ and $\hat{\theta}_2$ for $\theta = Eg(X)$. Build control variate as following.

$$\hat{\theta}_c = c\hat{\theta}_1 + (1-c)\hat{\theta}_2$$

It is obvious that $\hat{\theta}_c$ is also unbiased of θ for every $c \in \mathbb{R}$.

$$Var(\hat{\theta}_c) = Var(\hat{\theta}_2) + c^2 Var(\hat{\theta}_1 - \hat{\theta}_2) 2c Cov(\hat{\theta}_2, \hat{\theta}_1 - \hat{\theta}_2) \tag{2.9}$$

Let $\hat{\theta}_1$ and $\hat{\theta}_2$ be antithetic variate choice. Recall that *antithetic variate* give that for $\hat{\theta}_1$ and $\hat{\theta}_2$,

$$\hat{\theta}_1, \hat{\theta}_2 \sim IID, \quad Corr(\hat{\theta}_1, \hat{\theta}_2) = -1$$

It follows that

$$Cov(\hat{\theta}_1, \hat{\theta}_2) = -Var(\hat{\theta}_1)$$

and that

$$Var(\hat{\theta}_c) = (4c^2 - 4c + 1) Var(\hat{\theta}_1)$$

Hence, it leads to choosing optimal

$$\hat{\theta}_{c^*} = \frac{\hat{\theta}_1 + \hat{\theta}_2}{2}$$

which we have been used in antithetic variate.

2.3.4 Several control variates

To summarize, control variate try to reduce variance by combining unbiased estimatros of the target parameter. We have used one variate $f(X)$. It might be possible to extend to multiple variates, so to speak, $f_1(X), \ldots, f_k(X)$. Thanks to the linearity of expectation,

$$\hat{\theta}_c = g(X) + \sum_{i=1}^{k} c_i \Big(f_i(X) - \mu_i \Big)$$

is also unbiased estimator, where $\mu_i = Ef_i(X)$. How to get each c_i^*? Rather than using variance and covariance, we can *fitting linear regression.*

2.3.5 Control variates and regression

See Equation (2.4) and Equation (2.7). It can be found that we were estimating linear regression coefficient as LSE.

Lemma 2.3 (Least squares estimator)**.** *Consider $Y_i = \beta_0 + \beta_1 X_i + \epsilon_i$. Then*

$$\hat{\beta}_1 = \frac{\sum (X_i - \overline{X})(Y_i - \overline{Y})}{\sum (Y_i - \overline{Y})} = \frac{\widehat{Cov}(X, Y)}{\widehat{Var}(Y)}$$

Control variate estimator $\hat{\theta}_c = g(X) + c\Big(f(Y) - \mu_f\Big)$ can be expressed in regression model as

$$Eg(X) = \beta_0 + \beta_1 Ef(X)$$

Then

$$\hat{\beta}_1 = \text{LSE of } g(X) \text{ on } f(X) = \frac{\widehat{Cov}(g(X), f(X))}{\widehat{Var}(f(X))} = -\hat{c}^* \tag{2.10}$$

Note that

$$\hat{\beta}_0 = \overline{g(X)} + \hat{c}^* \overline{f(X)}$$

This matches to $\hat{\theta}_{c^*}$ in previous section.

$$\hat{\beta}_0 + \hat{\beta}_1 \mu_f = \overline{g(X)} + \hat{c}^* (\overline{f(X)} - \mu_f) = \hat{\theta}_{c^*} \tag{2.11}$$

Also, we can get the error variance estimate

$$\hat{\sigma}^2 = \widehat{Var}(X + \hat{c}^* Y) = MSE$$

and

$$\widehat{Var} \hat{\theta}_c^* = \frac{\hat{\sigma}^2}{N}$$

From Example 2.3, we can change the code computing c^* - `cov(exp(u), u) / var(u)` to `lm(exp(u) ~ u)$coef[2]`.

```r
mc_data(runif, N = N, M = M) %>%
  .[,
    chat := lm(exp(u) ~ u)$coef[2],
    by = sam] %>%
  .[,
    .(con = mean(exp(u) + chat * (u - 1 / 2))),
    by = sam]
#>         sam  con
#>    1:    s1 1.82
#>    2:    s2 1.93
#>    3:    s3 1.62
#>    4:    s4 1.62
#>    5:    s5 1.76
#>   ---
#>  996:  s996 1.46
#>  997:  s997 1.72
#>  998:  s998 1.61
#>  999:  s999 1.59
#> 1000: s1000 1.71
```

In fact, we can use Equation (2.11) directly: `predict(lm, newdata = data.frame(u = mean(u)))`.

Algorithm 12: Control variables and regression

```
  input  : $g$, control variate $f$ with mean $\mu_f$
1 for $m \leftarrow 1$ to $M$ do
2     Generate $U_1^{(m)}, \ldots, U_N^{(m)} \stackrel{iid}{\sim} unif$;
3     Set $g = g(U_i)$ and $f = f(U_i)$;
4     Regression $g \sim f$;
5     Predict the regression at $\overline{U}^{(m)}$. It is $\hat\theta_{c^*}^{(j)}$;
6 end
7 $\bar{\hat\theta} = \frac{1}{M}\sum \hat\theta_{c^*}^{(j)}$;
8 $\widehat{Var}(\hat\theta) = \frac{1}{M-1}\sum(\hat\theta_{c^*}^{(j)} - \bar{\hat\theta})^2$;
  output: $\widehat{Var}(\hat\theta)$
```

```r
mc_data(runif, N = N, M = M)[,
                                  .(con = predict(lm(exp(u) ~ u, data = .SD),
                                                  newdata = data.table(u = 1 / 2))),
                                  by = sam]
#>          sam  con
#>    1:     s1 1.73
#>    2:     s2 1.71
#>    3:     s3 1.71
#>    4:     s4 1.72
#>    5:     s5 1.72
#>   ---
#>  996:   s996 1.72
#>  997:   s997 1.72
#>  998:   s998 1.72
#>  999:   s999 1.72
#> 1000:  s1000 1.71
```

Now, how to deal with multiple control variates?

$$X = \beta_0 + \sum_{i=1}^{k} \beta_i Y_i + \epsilon$$

Using *multiple linear regression* model, we can choose optimal c^* and estimate control variate estimate.

2.4 Importance Sampling

Simple MC computes

$$\int_A g(x)f(x)dx = Eg(X) = \theta$$

for some density function f. This method uses random number from f itself so that

$$\int_A g(x)f(x)dx \approx \frac{1}{N}\sum_{i=1}^{N} g(X_i)$$

where $X_1, \ldots, X_N \overset{iid}{\sim} f$. This is why MC integration is called *direct sampling*. Sometimes, however, we face unkown distribution. In this case, generating from f directly is not easy. Even we can, it can be inefficient. The solution is *indirect method*: draw a sample from another pdf h. This is called **importance sampling**.

2.4.1 Importance sampling

Consdier MC integration as before.

$$\int_A g(x)f(x)dx = Eg(X) = \theta$$

How about uniform random number set with simple MC as before? However, uniform random numbers does not apply to unbounded intervals. When the target function is not that uniform, especially, generating numbers uniformly can be inefficient.

$$\begin{aligned}
E_f g(X) &= \int_A g(x)f(x)dx \\
&= \int_A g(x)\frac{f(x)}{\phi(x)}\phi(x)dx, \qquad \phi : \text{density on } A \\
&= E_\phi \frac{g(X)f(X)}{\phi(X)} \\
&\approx \frac{1}{N}\sum_{i=1}^N \frac{g(X_i)f(X_i)}{\phi(X_i)}, \qquad X_i \overset{iid}{\sim} \phi
\end{aligned} \tag{2.12}$$

Here, ϕ is called the *envelope* or the *importance sampling function*. This is just simple arithmetic, so it is possible to choose any density ϕ. However, we should take good one. Typically, one should select ϕ so that

$$\phi(x) \approx |g(x)|f(x) \quad \text{on } A \tag{2.13}$$

with finite variance.

Example 2.4 (Choice of importance function)**.** Obtain MC estimate of

$$\int_0^1 \frac{e^{-x}}{1+x^2}dx$$

by importance sampling.

```r
g_target <- function(x) {
  exp(-x - log(1 + x^2)) * (x > 0) * (x < 1)
}
```

Consider candiate envelopes

$$\begin{cases}
\phi_0(x) = 1, & 0 < x < 1 \\
\phi_1(x) = e^{-x}, & 0 < x < \infty \\
\phi_2(x) = \frac{1}{\pi(1+x^2)}, & x \in \mathbb{R} \\
\phi_3(x) = \frac{e^{-x}}{1-e^{-1}}, & 0 < x < 1 \\
\phi_4(x) = \frac{4}{\pi(1+x^2)} & 0 < x < 1
\end{cases}$$

```r
f0 <- function(x) {
  (x > 0) * (x < 1)
}
#------------------
f1 <- function(x) {
  exp(-x) * (x > 0)
}
#------------------
f2 <- function(x) {
  1 / (pi * (1 + x^2))
}
#------------------
f3 <- function(x) {
  exp(-x) / (1 - exp(-1)) * (x > 0) * (x < 1)
}
#------------------
f4 <- function(x) {
  4 / (pi * (1 + x^2)) * (x > 0) * (x < 1)
}
```

```r
tibble(x = seq(.01, .99, by = .01)) %>%
  mutate_all(.funs = list(~g_target(.), ~f0(.), ~f1(.), ~f2(.), ~f3(.), ~f4(.))) %>%
  gather(-x, key = "funs", value = "value") %>%
  ggplot(aes(x = x, y = value, colour = funs)) +
  geom_path()
```

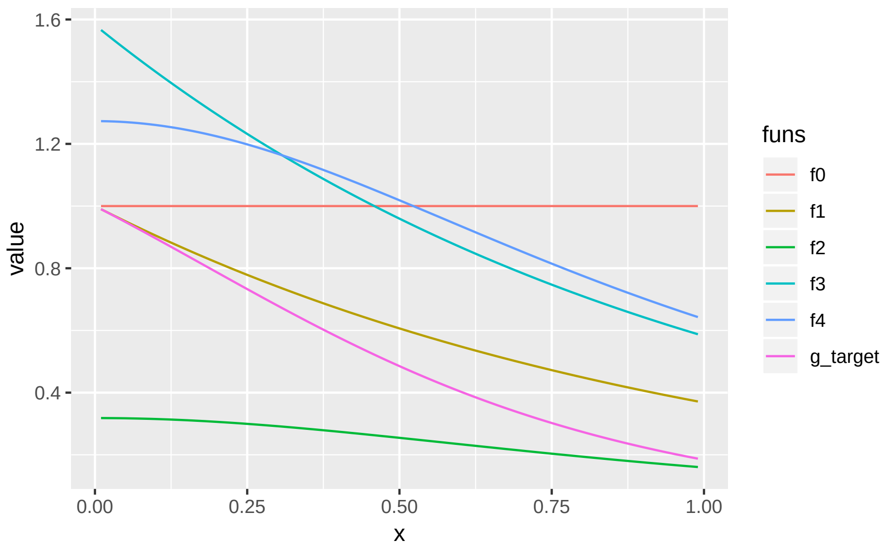


Figure 2.8: Importance funtions $\phi_0, \ldots, \phi_4$

Each importance function is drawn in Figure 2.8. f_1 shows similar patterns to g.

2.4.2 Variance in importance sampling

From Equation (2.12),

$$\theta = \int_A g(x)dx = \int_A \frac{g(x)}{\phi(x)}\phi(x)dx = E\left[\frac{g(X)}{\phi(X)}\right] \approx \frac{1}{N}\sum \frac{g(X_i)}{\phi(X_i)}$$

where $X_1, \ldots, X_N \overset{iid}{\sim} \phi$. Then

$$\begin{aligned} Var\hat\theta &= E\hat\theta^2 - (E\hat\theta)^2 \\ &= \int_A \left(\frac{g(x)}{\phi(x)} \right)^2 \phi(x) dx - \theta^2 \\ &= \int_A \frac{g(x)^2}{\phi(x)} dx - \theta^2 \end{aligned}$$

Hence, the mimimum variance

$$\left(\int_A \lvert g(x) \rvert dx \right)^2 - \theta^2$$

is obtained when

$$\phi(x) = \frac{\lvert g(x) \rvert}{\int_A \lvert g(x) \rvert dx}$$

But we do not know the value of denominator. It might be hart to get the exact function giving the minimum variance, but choosing ϕ close to the shape of $|g|$ would produce good result. To check our criterion (2.13) more clearly, compute $\frac{g}{\phi_i}$.

```r
tibble(x = seq(.01, .99, by = .01)) %>%
  mutate_all(.funs = list(~g_target(.), ~f0(.), ~f1(.), ~f2(.), ~f3(.), ~f4(.))) %>%
  gather(-x, -g_target, key = "funs", value = "value") %>%
  mutate(value = g_target / value) %>%
  ggplot(aes(x = x, y = value, colour = funs)) +
  geom_path()
```

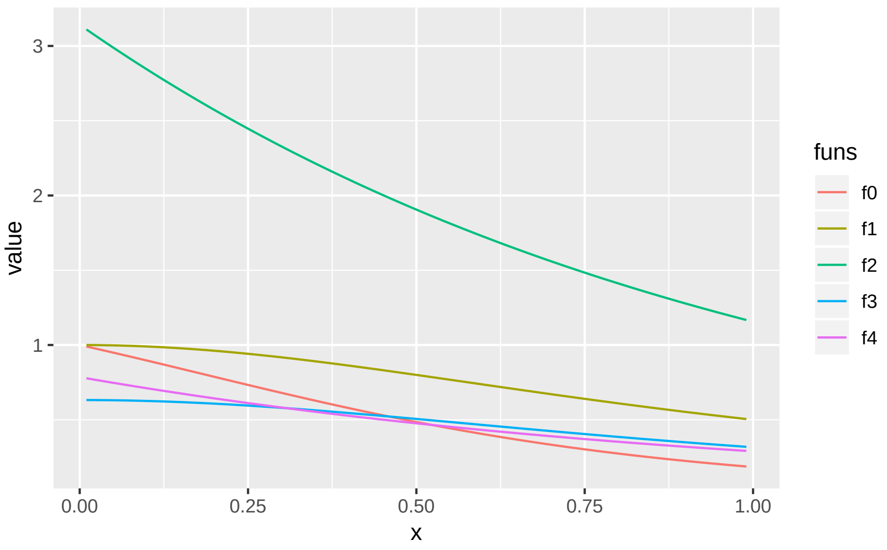


Figure 2.9: Ratio $\frac{g}{\phi_i}$

What is the closest to 1? f_1, of course. Would this function produce the best result, i.e. variance?

```r
theta_imp0 <-
  mc_data(runif, N = 100)[,
                          .(phi0 = mean(g_target(u) / f0(u))),
                          keyby = sam]
```

```r
theta_imp1 <-
  mc_data(rexp, N = 100, rate = 1)[,
                                   .(phi1 = mean(g_target(u) / f1(u))),
                                   keyby = sam]
```

```r
rf2 <- function(n) {
  x <- rcauchy(n)
  x[(x > 1) | (x < 0)] <- 2 # catch overflow errors in g
  x
}
theta_imp2 <-
  mc_data(rf2, N = 100)[,
                        .(phi2 = mean(g_target(u) / f2(u))),
                        keyby = sam]
```

```r
rf3 <- function(n) {
  u <- runif(n)
  x <- -log(1 - u * (1 - exp(-1))) # inverse transformation method
  x
}
#---------------------------
theta_imp3 <-
  mc_data(rf3, N = 100)[,
                        .(phi3 = mean(g_target(u) / f3(u))),
                        keyby = sam]
```

```r
rf4 <- function(n) {
  u <- runif(n)
  tan(pi * u / 4) # inverse transformation method
}
#--------------------
theta_imp4 <-
  mc_data(rf4, N = 100)[,
                        .(phi4 = mean(g_target(u) / f4(u))),
                        keyby = sam]
```

```r
theta_imp <- theta_imp0[theta_imp1][theta_imp2][theta_imp3][theta_imp4]
```

```r
theta_imp %>%
  melt(id.vars = "sam", variable.name = "imp_fun", value.name = "integral") %>%
  ggplot(aes(x = integral, fill = imp_fun)) +
  geom_histogram(bins = 30, position = "identity", alpha = .5) +
  xlab(expression(theta))
```

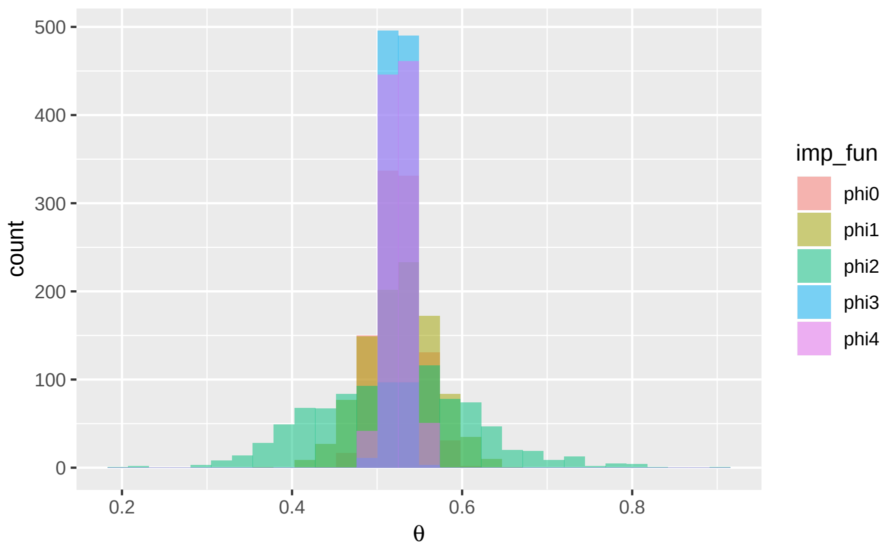


Figure 2.10: Empirical distribution of each importance sampling

```
theta_imp[,
          lapply(.SD, sd),
          , .SDcols = -"sam"]
#>      phi0  phi1   phi2    phi3   phi4
#> 1: 0.0246 0.042 0.0941 0.00977 0.0144
```

f_3 and possibly f_4 yields the lowest variance. What happened to f_1? Its support is $(0, \infty)$, so many values would be generated outside of $(0, 1)$. This results in many zeros in the sum of $\frac{g}{f}$.

Chapter 3

Monte Carlo Methods in Inference

3.1 Parametric Bootstrap

In this setting, we know distribution of X. We can freely generate from this distribution.

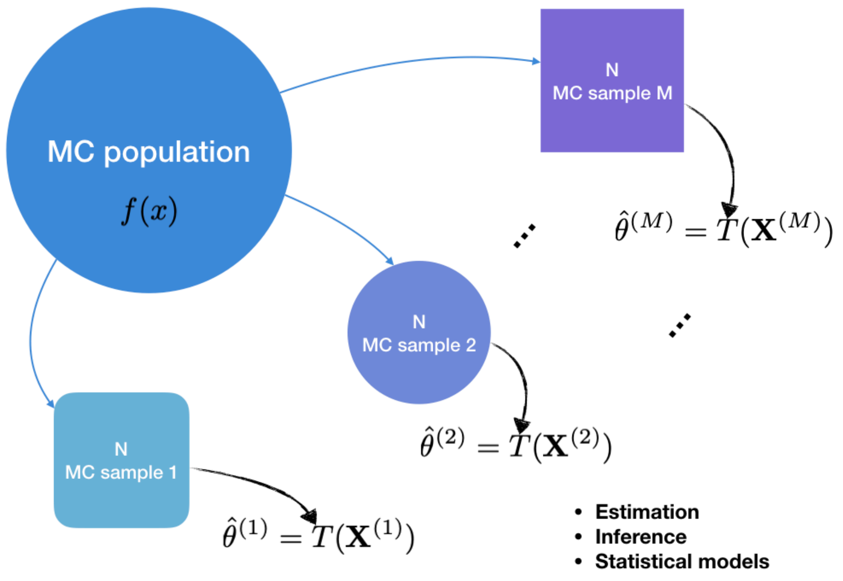

MC population
$f(x)$
N
MC sample M
N
MC sample 2
N
MC sample 1
$\hat{\theta}^{(M)} = T(\mathbf{X}^{(M)})$
$\hat{\theta}^{(2)} = T(\mathbf{X}^{(2)})$
$\hat{\theta}^{(1)} = T(\mathbf{X}^{(1)})$
Estimation
Inference
Statistical models

Figure 3.1: Parametric bootstrap

See Figure 3.1. From the "true" distribution, we can generate multiple samples. From each sample estimator can be computed. Then we can check these multiple estimates. Multiple estimates are close to motivation of estimator, so it helps exploring statistical inference with simple steps.

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

3.2 Monte Carlo Methods for Estimation

Example 3.1 (Any quantity of interest)**.** Suppose that $X_1, X_2 \overset{iid}{\sim} N(0,1)$. We want to estimate

$$\theta = E\lvert X_1 - X_2 \rvert$$

3.2.1 Empirical distribution

Algorithm 13: Empirical distribution of $\hat{\theta}$

```
   input  : distribution $f$
1  for $m \leftarrow 1$ to $M$ do
2      Generate $(X_1^{(m)}, X_2^{(m)}) \overset{iid}{\sim} N(0,1)$;
3      Compute $\hat{\theta}^{(m)} = \lvert X_1^{(m)} - X_2^{(m)} \rvert$;
4  end
5  Draw a histogram;
   output: $\bar{\hat{\theta}} = \frac{1}{M} \sum_{m=1}^{M} \hat{\theta}_m^{(m)}, \{\hat{\theta}^{(1)}, \ldots, \hat{\theta}^{(M)}\}$
```

```
basicmc <-
  mc_data(rnorm, N = 2)[,
                        xname := gl(2, 1, length = 2000, labels = c("x1", "x2"))] %>%
  dcast(sam ~ xname, value.var = "x") %>%
  .[,
    .(that = mean(abs(x1 - x2))),
    by = sam]
```

```
basicmc[,
        .(est = mean(that))]
#>    est
#> 1: 1.1
```

```
basicmc %>%
  ggplot(aes(x = that)) +
  geom_histogram(bins = 30, col = gg_hcl(1), alpha = .7) +
  xlab(expression(theta))
```

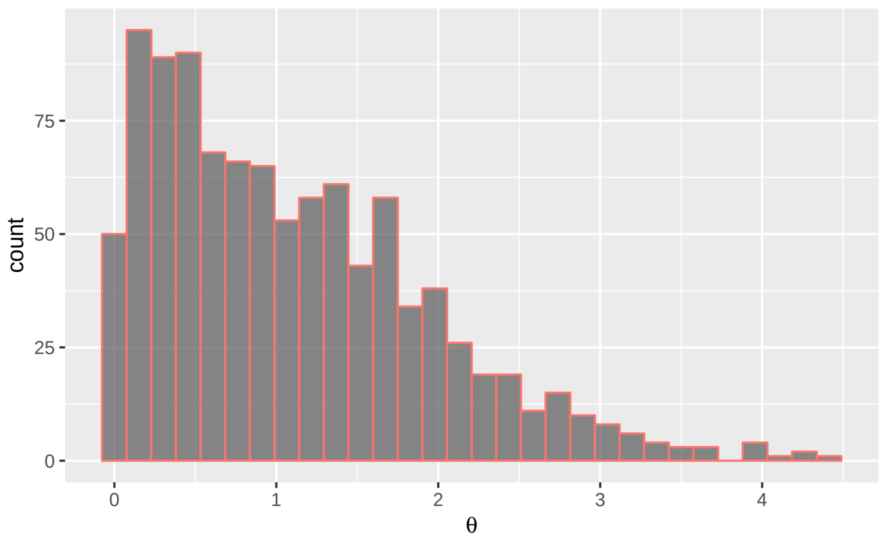


Figure 3.2: Empirical distribution of $\hat\theta$ for $|X_1 - X_2|$

3.2.2 Standard error

In Algorithm 13, we can get standard error by just calculating standard deviation of

$$\{\hat\theta^{(1)}, \ldots, \hat\theta^{(M)}\}$$

Algorithm 14: Standard error of $\hat\theta$

```
   input  : distribution $f$
 1 for $m \leftarrow 1$ to $M$ do
 2     Generate $(X_1^{(m)}, X_2^{(m)}) \overset{iid}{\sim} N(0,1)$;
 3     Compute $\hat\theta^{(m)} = |X_1^{(m)} - X_2^{(m)}|$;
 4 end
 5 $\bar{\hat\theta} = \frac{1}{M} \sum_{m=1}^{M} \hat\theta_m^{(m)}$;
 6 $\widehat{SE}(\hat\theta) = \sqrt{\frac{1}{M-1} \sum_{m=1}^{M} (\hat\theta^{(m)} - \bar{\hat\theta})}$;
   output: $\widehat{SE}(\hat\theta)$
```

```
basicmc[,
        .(se = sd(that))]
#>        se
#> 1: 0.844
```

3.2.3 Mean squared error

MSE is used when comparing several estimators.

Definition 3.1 (Mean squared error)**.**

$$MSE(\hat\theta) := E(\hat\theta - \theta)^2$$

To know *MSE*, however, we should compute expectation. Some of them might be complicated even though we know true distribution. As the last chapter, we can apply Monte carlo method.

Example 3.2 (MSE of a trimmed mean)**.** Suppose that $X_1, \ldots, X_n \overset{iid}{\sim} N(2, 1)$. Consider three estimators for $\mu = 2$.

1. mean $\overline{X}$
2. median $\tilde{X}$
3. kth trimmed mean $\overline{X}_{[-k]}$

```
Algorithm 15: MSE of mean, median, and kth trimmed mean
   input  : distribution $f$
 1 for $m \leftarrow 1$ to $M$ do
 2     Generate $(X_1^{(m)}, \ldots, X_N^{(m)}) \overset{iid}{\sim} N(2, 1)$;
 3     Sort $(X_1^{(m)}, \ldots, X_N^{(m)})$ in increasing order, i.e. $(X_{(1)}^{(m)}, \ldots, X_{(N)}^{(m)})$;
 4     Mean $\overline{X}^{(m)} = \frac{1}{N} \sum_{i=1}^N X_i^{(m)}$;
 5     Median $\tilde{X}^{(m)} = \begin{cases} X_{\frac{N}{2}+1}^{(m)} & N \text{ odd} \\ \frac{X_{\frac{N}{2}}^{(m)} + X_{\frac{N}{2}+1}^{(m)}}{2} & N \text{ even} \end{cases}$;
 6     kth trimmed mean $\overline{X}_{[-k]}^{(m)} = \frac{1}{N-2k} \sum_{i=k+1}^{n-k} X_{(i)}^{(m)}$
 7 end
 8 $\widehat{MSE}(\overline{X}) = \frac{1}{M} \sum_{m=1}^M (\overline{X}^{(m)} - 2)^2$;
 9 $\widehat{MSE}(\tilde{X}) = \frac{1}{M} \sum_{m=1}^M (\tilde{X}^{(m)} - 2)^2$;
10 $\widehat{MSE}(\overline{X}_{[-k]}) = \frac{1}{M} \sum_{m=1}^M (\overline{X}_{[-k]}^{(m)} - 2)^2$;
   output: $\widehat{MSE}(\overline{X}), \widehat{MSE}(\tilde{X})$, and $\widehat{MSE}(\overline{X}_{[-k]})$
```

```r
trim <- function(x, k = 1) {
  n <- length(x)
  x <- sort(x)
  sum(x[(k + 1):(n - k)]) / (n - 2 * k)
}
#--------------------------------------
mu_list <- function(x, k) {
  list(mean = mean(x), median = median(x), trim = trim(x, k))
}
```

Try $k = 1$.

```r
(trim_mc <-
  mc_data(rnorm, mean = 2, sd = 1)[,
                                   unlist(lapply(.SD, mu_list, k = 1)) %>% as.list,
                                   by = sam])
#>          sam x.mean x.median x.trim
#>    1:     s1   2.02     2.02   2.02
#>    2:     s2   2.00     2.00   2.00
#>    3:     s3   2.00     2.01   2.00
#>    4:     s4   1.99     1.98   1.99
#>    5:     s5   2.00     1.99   2.00
#>   ---
#>  996:   s996   2.02     2.02   2.02
#>  997:  s997   2.00     1.99   2.00
#>  998:  s998   1.99     1.99   1.99
#>  999:  s999   1.99     1.99   1.99
#> 1000: s1000   2.00     2.01   2.00
```

```
trim_mc %>%
  melt(id.vars = "sam", variable.name = "hat") %>%
  ggplot(aes(x = value, fill = hat)) +
  geom_histogram(bins = 30, alpha = .3, position = "identity") +
  xlab(expression(mu)) +
  geom_vline(xintercept = 2, col = I("red")) +
  scale_fill_discrete(
    name = "Estimates",
    labels = c("Mean", "Median", "Trimmed")
  )
```

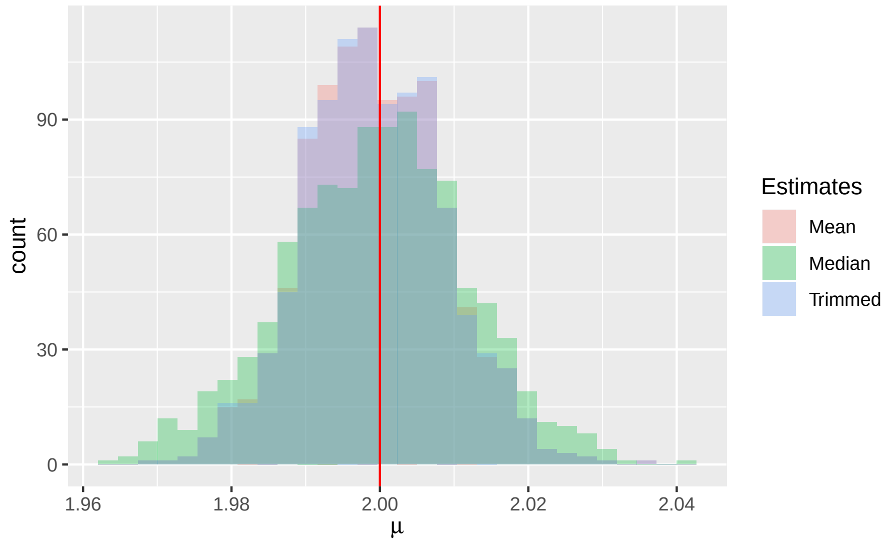


Figure 3.3: Empirical distribution of each estimator for $\mu = 2$

Here, median shows the largest standard error.

```
trim_mc[,
        lapply(.SD, sd),
        .SDcols = -"sam"]
#>     x.mean x.median  x.trim
#> 1: 0.00942   0.0123 0.00942
```

Now try various k for trimmed mean.

```
mse_list <- function(x, k) {
  list(mse = mean((x - 2)^2), se = sd(x))
}
#------------------------------------------
trim_mse <-
  mc_data(rnorm, mean = 2, sd = 1)[,
                                   lapply(.SD, function(x) {
                                     sapply(0:9, function(k) {
                                       trim(x = x, k = k)
```

```
                                          })
                                        }) %>%
                                          unlist() %>%
                                          as.list(),
                                        by = sam][,
                                                  lapply(.SD, mse_list) %>%
                                                    unlist() %>%
                                                    as.list(),
                                                  .SDcols = -"sam"]
```

```
trim_mse %>%
  transpose() %>%
  .[,
    `:=`(
      k = rep(0:9, each = 2),
      hat = gl(2, k = 1, length = 2 * 10, labels = c("mse", "se"))
    )] %>%
  dcast(k ~ hat, value.var = "V1")
#>     k      mse      se
#>  1: 0 9.83e-05 0.00992
#>  2: 1 9.83e-05 0.00992
#>  3: 2 9.83e-05 0.00992
#>  4: 3 9.83e-05 0.00992
#>  5: 4 9.82e-05 0.00992
#>  6: 5 9.83e-05 0.00992
#>  7: 6 9.83e-05 0.00992
#>  8: 7 9.83e-05 0.00992
#>  9: 8 9.83e-05 0.00992
#> 10: 9 9.83e-05 0.00992
```

3.3 Confidence interval

Remember the meaning of 95% confidence interval. *If we have 100 samples and construct confidence interval in each sample, 95 intervals would include true parameter.* In this Monte Carlo setting, we know true population distribution, so we can generate multiple samples. Thus, we can reproduce this confidence interval situation.

3.3.1 Empirical confidence interval

See one of histograms of Figure 3.3. Estimates are sorted. Calculating the upper and lower quantiles would give values close to confidence interval. See Figure 3.2. While the former show symmetric distribution, this is not. 0.25 and 0.975 quantile might be inappropriate. In this case, we should pick the *shortest interval* *with 95%.* Best critical region leads to the shortest length of CI given α, so we are finding this one.

Algorithm 16: Empirical confidence interval by Monte Carlo method

```
   input  : distribution $f$
 1 for $m \leftarrow 1$ to $M$ do
 2     Generate $X_1^{(m)}, \ldots, X_n^{(m)} \overset{iid}{\sim} f$;
 3     Compute $\hat{\theta}^{(m)} = \hat{\theta}(\mathbf{X^{(m)}})$;
 4 end
 5 if Distribution of $\{\hat{\theta}^{(m)}\}_1^M$ symmetric then
 6     Sort $\{\hat{\theta}^{(1)}, \ldots, \hat{\theta}^{(M)}\}$ in decreasing order, i.e. $\{\hat{\theta}_{(1)}^{(1)}, \ldots, \hat{\theta}_{(M)}^{(M)}\}$;
 7     Compute $LB = \frac{\alpha}{2}$ sample quantile and $UB = 1 - \frac{\alpha}{2}$ sample quantile;
 8 else
 9     foreach $lb < 0.05$ with $ub - lb = 1 - \alpha$ do
10         Candidate interval $(lb, ub)$;
11         calculate length $l_i = ub - lb$;
12     end
13     $(LB, UB)$: pick up the interval with the smallest length $l_i$;
14 end
   output: $(LB, UB)$
```

3.3.2 Empirical confidence level

On the contrary, we can estiamte confidence level given confidence interval.

Example 3.3 (Confidence interval for variance)**.** If $X_1, \ldots, X_n \overset{iid}{\sim} N(\mu, \sigma^2)$, then

$$T = \frac{(n-1)S^2}{\sigma^2} \sim \chi^2(n-1)$$

Thus, $100(1-\alpha)\%$ confidence interval is given by

$$(0, \frac{(n-1)S^2}{\chi^2_\alpha(n-1)})$$

For each MC sample, compute confidence interval. Just check if *known true parameter* is in the interval. Its proportion becomes the confidence level. It is simpler that estimate confidence interval itself.

Algorithm 17: Empirical confidence level by Monte Carlo method

```
   input  : distribution $f$ with parameter $\theta$
 1 for $m \leftarrow 1$ to $M$ do
 2     Generate $X_1^{(m)}, \ldots, X_n^{(m)} \overset{iid}{\sim} f$;
 3     Compute the confidence interval $C_m$;
 4     Compute $Y_j = I(\theta \in C_m)$, i.e. whether $\theta$ is in the CI;
 5 end
 6 Empirical confidence level $\overline{Y} = \sum_{m=1}^{M} Y_m$;
   output: $\overline{Y}$
```

Let $\mu = 0$, $\sigma = 2$, $N = 20$, and let $M = 1000$.

```r
ci_var <- function(x, variance, alpha) {
  n <- length(x)
  s2 <- var(x)
  (n - 1) * s2 / qchisq(alpha, df = n - 1) > variance
}
#--------------------------
ci_lev <-
  mc_data(rnorm, N = 20, M = 1000, mean = 0, sd = 2)[,
                                                      .(hat = mean(ci_var(x, variance = 4, alpha = .05))
                                                      by = sam]
```

```
ci_lev[,
       .N,
       by = hat][,
                 proportion := N / sum(N)] %>%
  ggplot(aes(x = hat, y = proportion, fill = factor(hat))) +
  geom_bar(stat = "identity") +
  scale_fill_discrete(
    name = "CI",
    labels = c("out", "in")
  ) +
  xlab(expression(y))
```

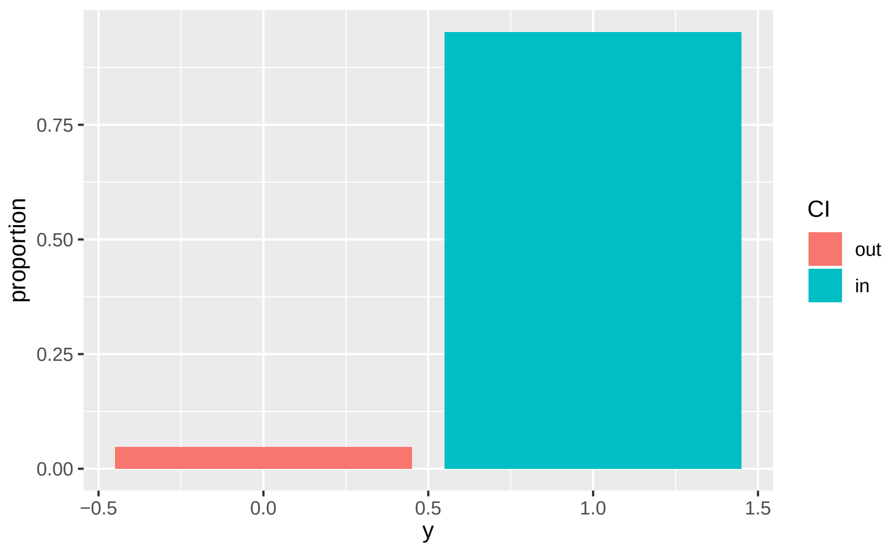


Figure 3.4: Proportion of σ^2 in confidence intervals

This leads to empirical confidence level, i.e. *sample proportion*. Just follow the last step 6 of Algorithm 17.

```
(ci_lev <-
  ci_lev[,
         .(level = mean(hat))])
#>    level
#> 1: 0.952
```

It is very close to 0.95. One of advantages of simulation study is we can assume various situation. For example, *violation of Gausiannity*.

Example 3.4 (Violation of Normal distribution assumption)**.** Refer to Example 3.3. This has assumed that $X_i \overset{iid}{\sim} N(\mu = 2, \sigma^2 = 4)$. What if not? For instance,

$$X_1, \ldots, X_n \overset{iid}{\sim} \chi^2(df = 2)$$

Just change random numbers.

```r
ci_lev2 <-
  mc_data(rchisq, N = 20, M = 1000, df = 2)[,
                                            .(hat = mean(ci_var(x, variance = 4, alpha = .05))),
                                            by = sam][,
                                                      .(non_normal = mean(hat))]
```

Table 3.1: Empirical confidence level for each population

Normal	Chisq
0.952	0.763

From Table 3.1, we found that *non-normality lowers confidence level* from 0.952 to 0.763.

3.4 Hypothesis tests

Using MC method, we have done point estimation and interval estimation. Now consider *hypothesis testing*.

$$H_0 : \theta \in \Theta_0 \qquad \text{vs} \qquad H_1 : \theta \in \Theta_1$$

where $\{\Theta_0, \Theta_1\}$ is a partition of the parameter space Θ. First of all, we have *test statistic*

$$T(\mathbf{X}) \overset{H_0}{\sim} f$$

and f is called *null distribution*. Given observed data, we compute this test statistic T_0. Where T_0 is located in the null distribution f decides whether we reject or accept H_0. If T_0 is very far from the middle, we can say that the realized data set is very rare event under H_0. In this case, we reject H_0. Otherwise, accept it. This is why we compute the tail probability, p-value.

3.4.1 Empirical p-value

Example 3.5. Suppose that $X_1, \ldots, X_{10} \overset{iid}{\sim} Exp(\lambda = 1)$, which are observed as follows

$$`rxexp`$$

Let $\theta = E(X) = \frac{1}{\lambda}$.

$$H_0 : \theta = 0.5 \qquad \text{vs} \qquad H_1 : \theta > 0.5$$

Test using $T = \frac{\overline{X} - \theta_0}{S/\sqrt{n}}$ statistic.

Before looking at p-value, briefly look at *empirical null distribution* of test statistic.

```r
mc_data(rexp, rate = 2)[,
                        .(tstat = t.test(x, mu = .5)$statistic),
                        by = sam] %>%
  ggplot(aes(x = tstat)) +
  geom_histogram(bins = 30, col = gg_hcl(1), alpha = .7) +
  geom_vline(xintercept = t.test(xexp, mu = .5)$statistic, col = I("red")) + # xexp: observed data
  geom_vline(xintercept = -t.test(xexp, mu = .5)$statistic, col = I("red")) +
  xlab("T")
```

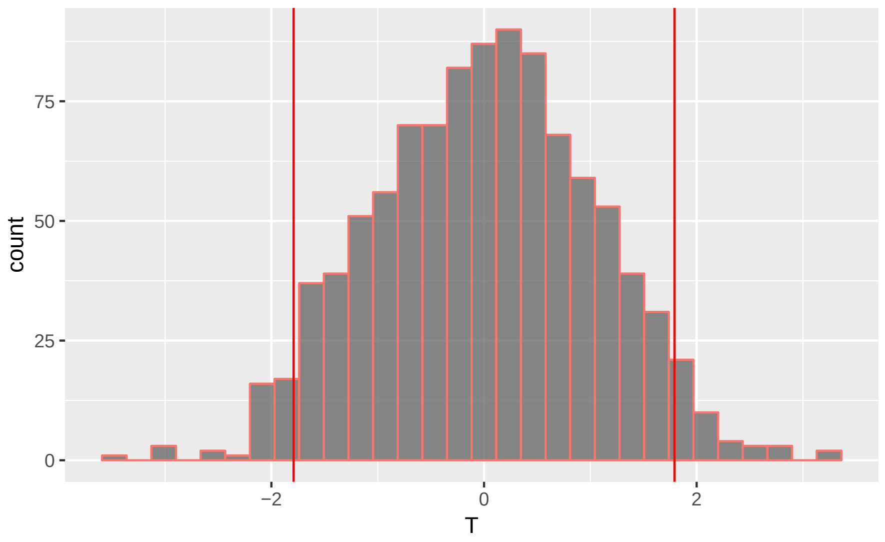


Figure 3.5: Emprirical Null Distribution

By proceeding the similar way, we can get empirical distribution of test statistics. Some are out of observed T_0, some are not. Motivation is that we just count these. Proportion of these would estimate p-value. Recap what p-value is.

Definition 3.2 (p-value)**.** Let T be test statistic and let T_0 be observed test statistic given data. Then p-value is

$$p - value := \begin{cases} P(|T| \geq T_0 \mid H_0) & \text{both sided} \\ P(T \geq T_0 \mid H_0) & \text{one sided} \\ P(T \leq T_0 \mid H_0) & \text{one sided} \end{cases}$$

Denote that p-value is probability. So in MC setting, we can estimate this by computing *sample mean of identity function.*

Lemma 3.1 (Empirical p-value)**.** *Let T_0 be observed test statistic and let $\{T_1, \ldots, T_M\}$ be test statistic computed in each MC sample.*

$$Empirical\ p\text{-}value = \begin{cases} \frac{\left|\{T_j : (T_j > |T_0|)\ or\ (T_j < -|T_0|)\}\right|}{M} & \textit{both-sided} \\ \frac{\left|\{T_j : (T_j > T_0)\}\right|}{M}\ or\ \frac{\left|\{T_j : (T_j < T_0)\}\right|}{M} & \textit{one-sided} \end{cases}$$

Algorithm 18: Empirical p-value by Monte Carlo method

```
   input  : Given observed data, compute $T_0$
1  for $m \leftarrow 1$ to $M$ do
2      Generate $X_1^{(m)}, \ldots, X_n^{(m)} \overset{H_0}{\sim} f$;
3      Compute $T_m(\mathbf{X}^{(m)})$;
4  end
5  Empirical p-value $\hat{p} = \begin{cases} \frac{\left|\{T_j : (T_j > |T_0|) \text{ or } (T_j < -|T_0|)\}\right|}{M} & \text{both-sided} \\ \frac{\left|\{T_j : (T_j > T_0)\}\right|}{M} \text{ or } \frac{\left|\{T_j : (T_j < T_0)\}\right|}{M} & \text{one-sided} \end{cases}$;
   output: $\hat{p}$
```

Go back to Example 3.5. Only left is computing 5 of Algorighm 18. (Denote that `xexp` in the code is vector object of observed data).

```r
(tt_exp <-
  mc_data(rexp, rate = 2)[,
                          .(tstat = t.test(x, mu = .5)$statistic),
                          by = sam][,
                                    .(pval = mean(tstat > abs(t.test(xexp, mu = .5)$statistic)))])
#>      pval
#> 1: 0.046
```

It is smaller that `0.05`, so we reject H_0.

3.4.2 Comparing several tests

MC method would be used in comparing tests rather than conducting test itself. By generating random number, we can evaluate tests.

$$H_0 : \theta \in \Theta_0 \qquad \text{vs} \qquad H_1 : \theta \in \Theta_1$$

As mentioned earlier, $\{\Theta_0, \Theta_1\}$ is a partition of the parameter space Θ. For this test, we can perform several tests. Test method 1, test method 2, et cetera. All these methods produce error, but these errors might be different. So we try to compare this.

what is true	accept H_0	reject H_0
H_0	correct decision	*Type I error*
H_1	*Type II Error*	correct decision

In most tests, we aims to reject H_0. By rejecting it, we can evidently say that H_0 is not true. In this sense, we treat type I error more importantly that type II error in general. Test strategy becomes to control type I error probability first and then lower type II error probabilty.

Definition 3.3 (Power function)**.** Let $\theta \in \Theta$ be a parameter of a test.

$$\beta(\theta) := P(\text{reject } H_0 \mid \theta)$$

With this power function, each type I error and type II error probability is given.

Lemma 3.2 (typeerr)**.** *1. $P(\text{Type I error}) = \beta(\theta_0), \quad \theta_0 \in \Theta_0$*

2. Power $\beta(\theta_1) = 1 - P(\text{Type II error}), \quad \theta_1 \in \Theta_1$

Following our test strategy, fixing $P(\text{Type I error})$ and maximizing $\beta(\theta_1)$, we construct following test.

Definition 3.4 (Size α Test)**.** A test with $\beta(\theta)$ is called size α test if and only if

$$\alpha := \sup_{\theta \in \Theta_0} \beta(\theta), \quad 0 \leq \alpha \leq 1$$

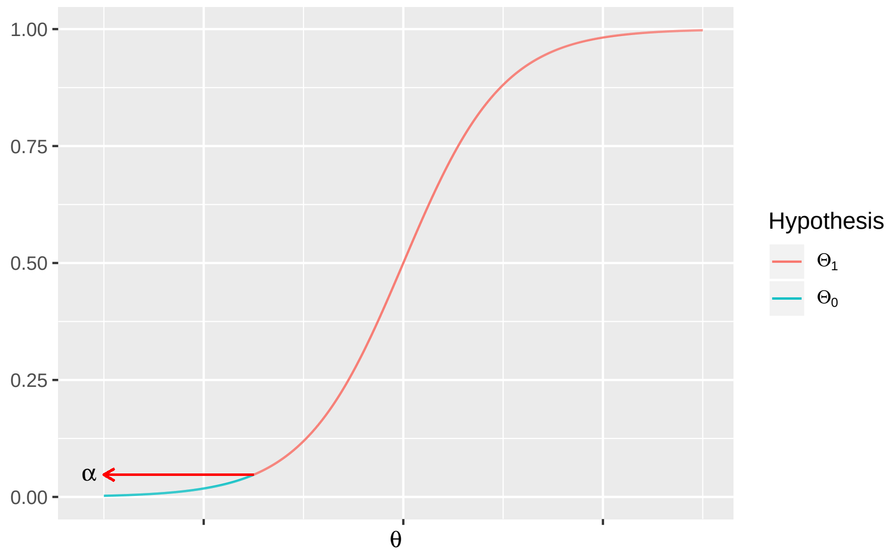


Figure 3.6: Size α Test

Then how to compare tests? Look at the following example. Three columns of the middle part are type I error rate.

test methods	$\alpha = 0.01$	$\alpha = 0.05$	$\alpha = 0.01$	Power
Test 1	0.09	0.04	0.01	0.7
Test 2	0.11	0.06	0.01	0.65
Test 3	0.15	0.07	0.02	0.9

Here, we will choose **Test 1**.

1. Type I error rate $\approx \alpha$
   - before looking at power, this should be satisfied.
   - So Test 3 is excluded
2. Larger power
   - Thus, we select Test 1.

3.4.3 Empirical type-I error rate

Recall Lemma 3.2. As in p-value, we just compute sample proportion for each type I error rate and power under null and alternative distribution.

Lemma 3.3. *Consider* $H_0 : \theta \in \Theta_0 \qquad vs \qquad H_1 : \theta \in \Theta_1$.

Define $I(\mathbf{X})$ *by*

$$I(\mathbf{X}) = \begin{cases} 1 & H_0 \textit{ is rejected} \mid H_0 \\ 0 & \textit{otherwise} \end{cases}$$

For each MC sample, compute this statistic $I_m = I(\mathbf{X}^m)$. Then empirical type I error rate can be computed as

$$\frac{1}{M}\sum_{m=1}^{M} I_m$$

Algorithm 19: Empirical type I error rate by Monte Carlo method

```
   input  : $H_0 : \theta \in \Theta_0$   vs   $H_1 : \theta \in \Theta_1$
 1 for $m \leftarrow 1$ to $M$ do
 2     Generate $X_1^{(m)}, \ldots, X_n^{(m)} \overset{H_0}{\sim} f$;
 3     Compute $T_m(\mathbf{X}^{(m)})$;
 4     Compute $I_m = \begin{cases} 1 & H_0 \text{ is rejected} \mid H_0 \\ 0 & \text{otherwise} \end{cases}$;
 5 end
 6 Empirical Type I error rate $\hat{\alpha} = \frac{1}{M}\sum_{m=1}^{M} I_m$;
   output: compare $\hat{\alpha}$ with $\alpha$
```

Example 3.6 (Testing normal mean)**.** Suppose that $X_1, \ldots, X_{20} \overset{iid}{\sim} N(\mu, \sigma^2 = 100)$. Test

$$H_0 : \mu = 500 \qquad \text{vs} \qquad H_1 : \mu > 500$$

1. Z-test: $Z = \frac{\overline{X} - 500}{\sigma^2/\sqrt{20}} \overset{H_0}{\sim} N(0,1)$
2. t-test: $T = \frac{\overline{X} - 500}{S/\sqrt{20}} \overset{H_0}{\sim} t(20-1)$

```
test_list <- function(x, mu, sig, a = .05) {
  n <- length(x)
  xbar <- mean(x) - mu
  list(
    z = xbar / (sig / sqrt(n)) > qnorm(a, lower.tail = FALSE),
    t = xbar / (sd(x) / sqrt(n)) > qt(a, df = n - 1, lower.tail = FALSE)
  )
}
#--------------------
err_mc <-
  mc_data(rnorm, N = 20, mean = 500, sd = 10)[,
                                              lapply(.SD, test_list, mu = 500, sig = 10) %>%
                                                unlist() %>%
                                                as.list(),
                                              by = sam][,
                                                        lapply(.SD, mean),
                                                        .SDcols = -"sam"]
```

Both test have Type I error close to α, but Z-test seems bit better.

Table 3.4: Empirical Type I error for Z and T

Z-test	T-test
0.048	0.051

3.4.4 Empirical power

Next step is power. See Figure 3.6. Power is different in that this is computed in *alternative distribution, not null distribution.*

$$\beta(\theta_1) = P(\text{reject } H_0 \mid \theta_1 \in \Theta_1)$$

Lemma 3.4. *Consider* $H_0 : \theta \in \Theta_0 \qquad vs \qquad H_1 : \theta \in \Theta_1$.

Define $I(\mathbf{X})$ *by*

$$I(\mathbf{X}) = \begin{cases} 1 & H_0 \textit{ is rejected} \mid H_1 \\ 0 & \textit{otherwise} \end{cases}$$

For each MC sample, compute this statistic $I_m = I(\mathbf{X}^m)$. *Then empirical power can be computed as*

$$\frac{1}{M}\sum_{m=1}^{M} I_m$$

Process will be same but we test under H_1. However, this makes a lot difference due to structure of each hypothesis. In many cases, H_0 is simple, i.e. $\mu = 500$. In 2 of Algorithm 19, we can consider only $N(\mu = 500, 100)$. Since Θ_0 and Θ_1 form partition, alternative hypothesis usually is not simple. In this example, $\mu > 500$. We cannot specify one distribution for alternatrive. How to deal with this?

Trying many points for $\mu_1 \in \Theta_1 = \{\mu : \mu > 500\}$ might be possible. Our goal is finding larger power. So *find test with larger power for all points in* Θ_1.

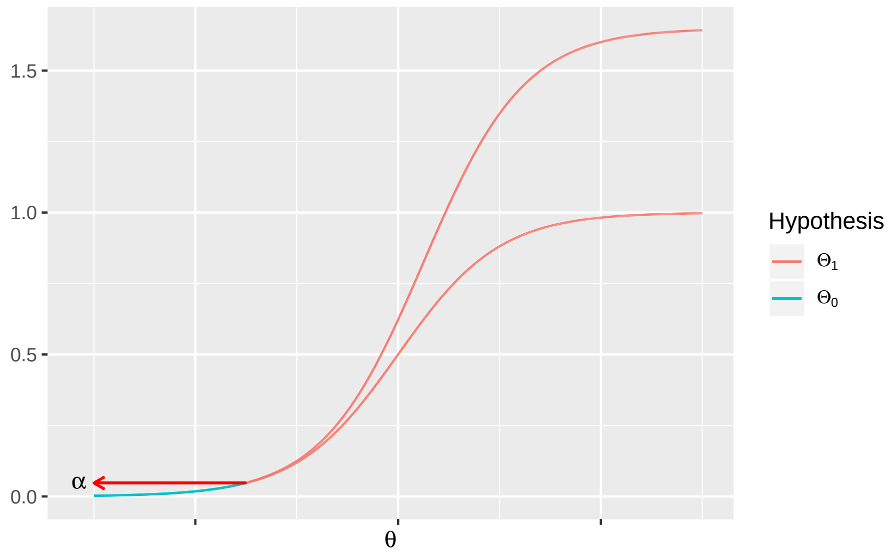


Figure 3.7: Comparing power between two test methods

See Figure 3.7. One test method has higher $\beta(\theta)$ function curve in Θ_1. This test is *powerful than the other*. We would choose this test in this step. So what we have to do is choose some points $\theta_1 \in \Theta_1$, and draw the power curve.

Algorithm 20: Empirical power by Monte Carlo method

```
   input : $H_0 : \theta \in \Theta_0$  vs  $H_1 : \theta \in \Theta_1$
 1 foreach $\theta_1 \in \Theta_1$ do
 2     for $m \leftarrow 1$ to $M$ do
 3         Generate $X_1^{(m)}, \ldots, X_n^{(m)} \overset{H_0}{\sim} f$;
 4         Compute $T_m(\mathbf{X}^{(m)})$;
 5         Compute $I_m = \begin{cases} 1 & H_0 \text{ is rejected} \mid H_1 \\ 0 & \text{otherwise} \end{cases}$;
 6     end
 7     Empirical power $\hat{\beta} = \frac{1}{M} \sum_{m=1}^{M} I_m$;
 8 end
 9 Draw a power curve $\hat{\beta}$ against $\theta_1$ output: curve and $\{\hat{\beta}\}$
```

In fact, we can try every $\theta \in \Theta$ and *draw entire power curve*. Refer to Example 3.6.

```r
pw_mc <-
  lapply(seq(450, 650, by = 10), function(mu) {
    mc_data(rnorm, N = 20, mean = mu, sd = 10)[,
                                               h1 := mu]
  })
pw_mc <- rbindlist(pw_mc)
```

One column is added from previous process. This is group for H_1. So we should specify `by = .(h1, sam)`.

```r
pw_mc <-
  pw_mc[,
        lapply(.SD, test_list, mu = 500, sig = 10) %>%
          unlist() %>%
          as.list(),
        by = .(h1, sam)][,
                         lapply(.SD, mean),
                         by = h1, .SDcols = -"sam"]
```

```r
pw_mc %>%
  melt(id.vars = "h1", variable.name = "test") %>%
  ggplot(aes(x = h1, y = value, colour = test)) +
  geom_path() +
  geom_point() +
  scale_colour_discrete(
    name = "Test",
    labels = c("Z", "T")
  ) +
  labs(
    x = expression(mu),
    y = expression(beta)
  )
```

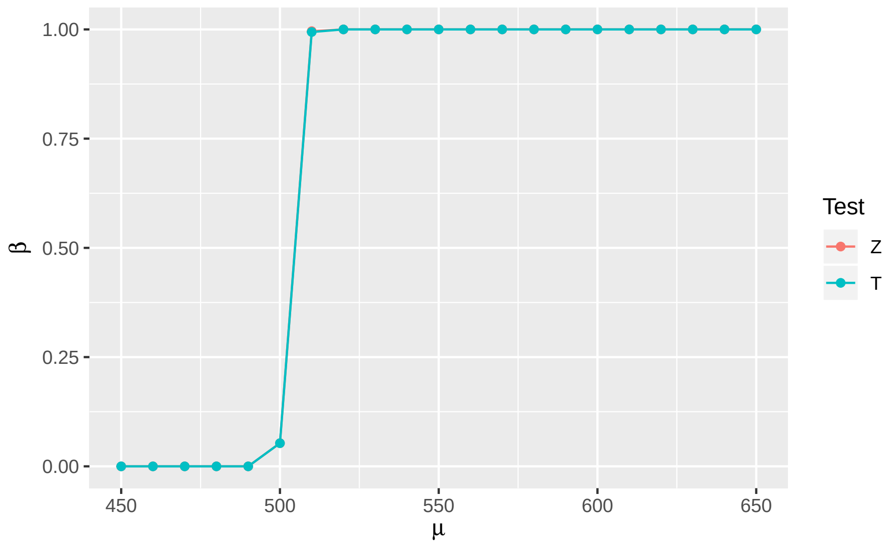


Figure 3.8: Empirical power curve of each z-test and t-test

Recall that we are estimating power. Instead of `mean()`, we can use `sd()`. This would give us *standard error* of our estimator for power. Since it is sample proportion,

$$\widehat{SE}(\hat{p}) = \sqrt{\frac{\hat{p}(1-\hat{p})}{M}}$$

Consider T-test.

```r
pw_mc2 <- lapply(seq(450, 650, by = 10), function(mu) {
  mc_data(rnorm, N = 20, mean = mu, sd = 10)[,
                                              h1 := mu]
})
pw_mc2 <- rbindlist(pw_mc2)
#-------------------
pw_mc2 <-
  pw_mc2[,
         .(te = t.test(x, alternative = "greater", mu = 500)$p.value <= .05),
         by = .(h1, sam)][,
                          .(te = mean(te)),
                          by = h1][,
                                   se := sqrt(te * (1 - te) / 1000)]
```

```r
pw_mc2 %>%
  ggplot(aes(x = h1, y = te)) +
  geom_ribbon(aes(ymin = te - se, ymax = te + se), col = gg_hcl(1)) +
  geom_path(alpha = .7) +
  geom_point() +
  labs(
    x = expression(mu),
    y = expression(beta)
  )
```

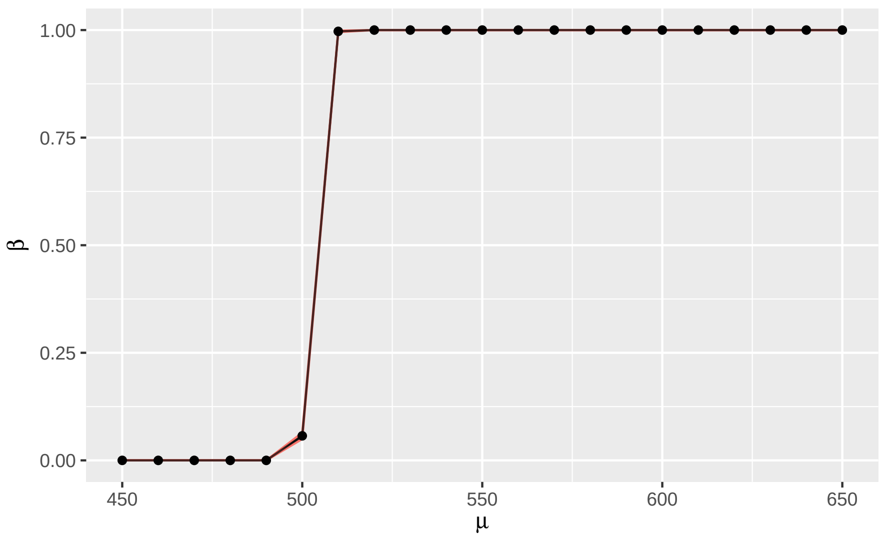


Figure 3.9: Empirical power curve $\hat{p} \pm \widehat{SE}(\hat{p})$ for t-test

3.4.5 Count Five test for equal variance

Commonly, F-test is used for equality of two population variances. McGrath and Yeh (2005) suggests nonparametric testing without Normal assumption, so called *Count Five*. Instead, this method requires some conditions.

1. same mean
2. same sample size

Algorithm 21: Count Five test

```
input : $X_1, \ldots, X_{n_x} \perp\!\!\!\perp Y_1, \ldots, Y_{n_y}$
        $H_0 : \sigma_X^2 = \sigma_Y^2$
1 Compute $C_X = \left| \{ i : |X_i - \overline{X}| > \max_j |Y_j - \overline{Y}| \} \right|$;
2 if $C_X \ge 5$ then
3     return reject $H_0$;
4 else
5     return accept $H_0$;
6 end
```

```r
gauss <-
  tibble(
    x1 = rnorm(20, mean = 0, sd = 1),
    x2 = rnorm(20, mean = 0, sd = 1.5)
  )
```

```r
gauss %>%
  gather(key = "variable", value = "value") %>%
  ggplot(aes(x = variable, y = value, fill = variable)) +
  geom_boxplot() +
  geom_point(alpha = .5)
```

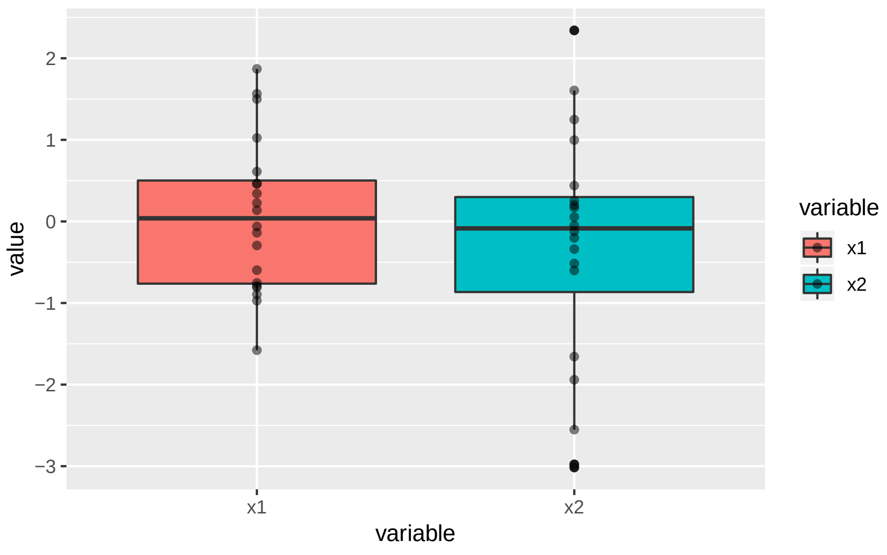


Figure 3.10: Side-by-side boxplot

We would perform *Count Five* test for multiple simulated data sets such as in Figure 3.10.

$$X_1^{(m)}, \ldots, X_{20}^{(m)} \sim N(0,1) \perp\!\!\!\perp Y_1^{(m)}, \ldots, Y_{20}^{(m)} \sim N(0,1.5)$$

```
count5test <- function(x, y) {
  X <- x - mean(x)
  Y <- x - mean(y)
  outx <- sum(X > max(Y)) + sum(X < min(Y))
  outy <- sum(Y > max(X)) + sum(Y < min(X))
  max(c(outx, outy)) > 5
}
```

Apply MC method to get *empirical type I error*.

```
mc_data(rnorm, N = 20, M = 1000)[,
                                  x2 := rnorm(20 * 1000)][,
                                                          .(chat = count5test(x = x, y = x2)),
                                                          by = sam][,
                                                                    .(chat = mean(chat))]
#>     chat
#> 1: 0.026
```

3.5 Statistical Methods

3.6 Bootstrap

3.6.1 Resampling

Bootstrap is a class of nonparametric Monte Carlo methods that estimate the distribution of a population by *resampling*. Different with previous MC method, we do not know population distribution. Instead, resampling methods treat an observed sample as a finte population. This is called *pseudo-population* in that this is regarded as having the same characteristics as the true population.

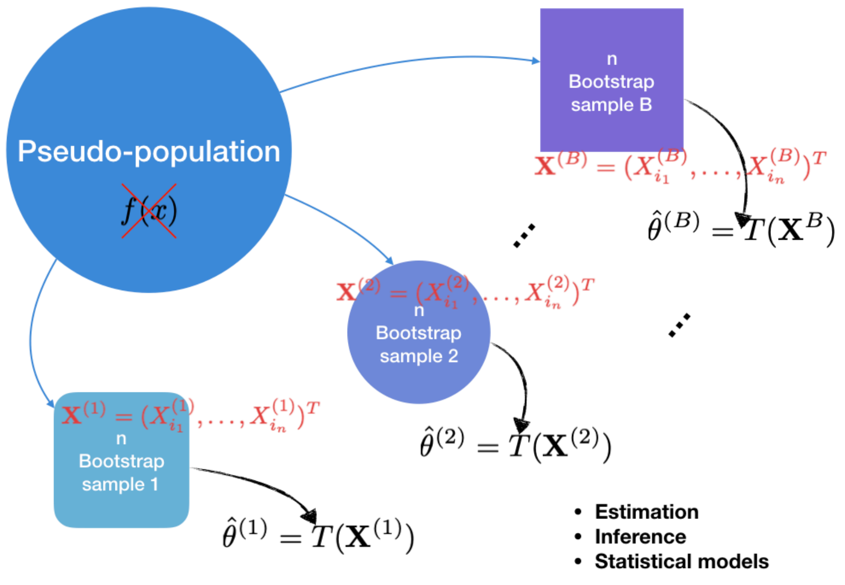

Pseudo-population
$f(x)$
n Bootstrap sample B
$\mathbf{X}^{(B)} = (X_{i_1}^{(B)}, \dots, X_{i_n}^{(B)})^T$
$\hat{\theta}^{(B)} = T(\mathbf{X}^B)$
$\mathbf{X}^{(2)} = (X_{i_1}^{(2)}, \dots, X_{i_n}^{(2)})^T$
n Bootstrap sample 2
$\hat{\theta}^{(2)} = T(\mathbf{X}^{(2)})$
$\mathbf{X}^{(1)} = (X_{i_1}^{(1)}, \dots, X_{i_n}^{(1)})^T$
n Bootstrap sample 1
$\hat{\theta}^{(1)} = T(\mathbf{X}^{(1)})$
Estimation
Inference
Statistical models

Figure 3.11: Resampling

See Figure 3.11. From the observed sample, which is pseudo-population, resampling generates multiple bootstrap samples by *sampling with replacement*. Surprisingly, this simple sampling procedure approximate the true population distribution quite successful.

3.6.2 Approximations in bootstrap

How does bootstrap work? Efron and Gong (1983) provides simple example $T = \overline{x}$, i.e. sample average.

Example 3.7 (Estimation of sample mean)**.** Having observed $X_1 = x_1, \dots, X_n = x_n$, compute

$$\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

Using bootstrap, we try to see the empirical distribution of this.

Note that

$$X_1, \dots, X_n \overset{iid}{\sim} F$$

where F is true unknown distribution. Having observed $X_1 = x_1, \dots, X_n = x_n$, we get *empirical distribution function* by computing the sample average.

$$\hat{F}(x) = \frac{1}{n}\sum_{i=1}^{n} I(X_i \le x_i) \tag{3.1}$$

This works for estimator of F. $\hat{F}$ endows mass $\frac{1}{n}$ on each observed point x_i, $i = 1, \dots, n$. In other words, pseudo-distribution becomes *discrete uniform*.

$$\hat{F} \overset{d}{=} unif(x_1, \dots, x_n) \tag{3.2}$$

We have set the population distribution which is bogus. Now we can apply previous MC sampling with $\frac{1}{n}$. Write $\{X_1^*, \ldots, X_n^*\}$ as *bootstrap sample* by resampling. Then

$$P(X^* = x_i) = \frac{1}{n}$$

i.e.

$$X^* \overset{iid}{\sim} unif(x_1, \ldots, x_n) \tag{3.3}$$

This gives *bootstrap cdf*, cdf of bootstrap samples, by

$$F^*(x) = \text{cdf of } unif(x_1, \ldots, x_n) = \hat{F} \tag{3.4}$$

One proceeds in a similar way for estimating cdf that

$$\hat{F}^* = \frac{1}{n}\sum_{i=1}^{n} I(X^* \le x_i) \tag{3.5}$$

This is called *ecdf of bootstrap replicates.*

Remark. For any data points $X_1, \ldots, X_n$,

1. ***Empirical Distribution Function*** $\hat{F}(x) = \frac{1}{n}\sum_{i=1}^{n} I(X_i \le x_i)$
2. ***Bootstrap cdf*** $F^*(x) = \hat{F}$
3. ***ECDF of bootstrap replicates*** $\hat{F}^* = \frac{1}{n}\sum_{i=1}^{n} I(X^* \le x_i)$

This remark can explain Figure 3.11 in a distribution way.

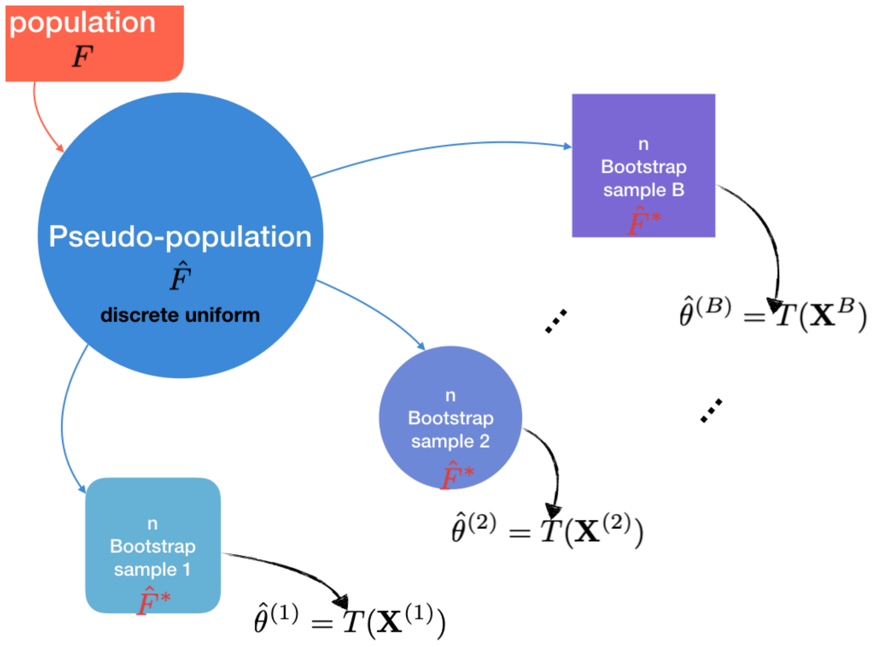


Figure 3.12: Empirical distribution of bootstrap

Figure 3.12 shows how each sample is distributed, approximately. We get the data set from true F. From this finite population, we first estimate F by $\hat{F}$. Resampling multiple bootstrap samples, each sample estimates $\hat{F}$ by $\hat{F}^*$.

Theorem 3.1 (Two approximations in bootstrap)**.** *There are two approximations in bootstrap. For large B, bootstrap samples approximate bootstrap replicates. For large n, bootstrap replicates approximate true population.*

$$\begin{tikzcd}[column sep=tiny]
\mathbf{X}^{(b)} \arrow[r] & \hat{f} \arrow[r] & f \\
\textit{large } B \arrow[ur, phantom, "\uparrow"] & & \\
& \textit{large } n \arrow[uur, phantom, "\uparrow"] &
\end{tikzcd}$$

Proof. Since

$$E\hat{F} = \frac{1}{n}\sum_{i=1}^{n} P(X_i \le x_i)$$

$$\hat{F} \overset{a.s.}{\to} F$$

as $n \to \infty$ by the strong law of large numbers. Let

$$\overline{\hat{F}^*} := \frac{1}{B}\sum_{b} \hat{F}_b^*$$

where $\hat{F}_b^*$ is ecdf of b-th bootstrap replicate. Since

$$E\hat{F}_b^* = \frac{1}{n}\sum_{i=1}^{n} P(X^{(b)} \le x_i)$$

$$\overline{\hat{F}^*} \overset{a.s.}{\to} F^* = \hat{F}$$

as $B \to \infty$ by S.L.L.N. □

Denote that Theorem 3.1 can be also expressed as

$$\begin{tikzcd}[column sep=tiny]
\overline{\hat{F}^*} \arrow[r] & \hat{F} \arrow[r] & F \\
\text{large B} \arrow[ur, phantom, "\uparrow"] & & \\
& \text{large n} \arrow[uur, phantom, "\uparrow"] &
\end{tikzcd} \tag{3.6}$$

This approximation not only justifies the bootstrap procedure but also shows the problem of it. We can always increase B if we want. Then we earn $\overline{\hat{F}^*} \approx \hat{F}$, i.e. bootstrap samples approximate pseudo-population. However, n is fixed. For $\hat{F}$ to approximate F, large n is required. It is not under control. If small n is given, $\hat{F}$ will not be close to F. Then the bootstrap samples will not be close to F finally.

Corollary 3.1. *Resampling the large number of replicates, i.e. large B produces a good estimates of $\hat{F}$ but it does not guarantee a good estimate of F.*

Example 3.8 (Poisson population)**.** Suppose that $\{2, 2, 1, 1, 5, 4, 4, 3, 2, 1\} \sim Poisson(\lambda = 2)$. Resampling from this pseudo-population, can we appropriately explain the population?

Table 3.5: Empirical distribution

1	2	3	4	5
0.3	0.3	0.1	0.2	0.1

Table 3.5 is just a result of averaging, emprical distribution $\hat{F}$. General bootstrap will resample by distribution. We can see the problem at once.

$$P(X = 0) = e^{-2} = 0.135$$

However, we did not observe 0, so the bootstrap sample will never include zero, i.e. not a full domain. In sum,

$$\overline{\hat{F}^*} \rightarrow \hat{F} \not\rightarrow F$$

3.6.3 Bootstrap standard error

Look at the Figure 3.12 again. For each sample, we calculate esimator of our interest. For instance, Example 3.7 - sample average $\overline{x}$. After that, we get empirical distribution of $\overline{x}$ such as *standard error.*

Before looking at the empirical distribution, let's review sample estimation.

Theorem 3.2 (Standard error of sample mean)**.** *Standard error of* $\overline{X} = \overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$*, i.e. the root mean squred error is estimated by*

$$\hat{\sigma} = \left[\frac{1}{n(n-1)} \sum_{i=1}^{n} (x_i - \overline{x})^2 \right]^{\frac{1}{2}}$$

Here, $n - 1$ was divided for *unbiasedness*. Recall that

$$Var(\overline{X}) = \frac{\sigma^2}{n}$$

It follows that

$$\hat{\sigma}^2 = \frac{s^2}{n}$$

and we know that s^2 should be divided by $n - 1$ to be unbiased. Bootstrap generalizes point estimation process 3.2 a bit differently (Efron and Gong, 1983). From Equation (3.1) to (3.4), we resample bootstrap sample with empricial probability distribution $\hat{F}$ so that

$$X_1^*, \ldots, X_n^* \overset{iid}{\sim} \hat{F} \tag{3.7}$$

where $\hat{F}$ is discrete uniform in each observed data point. In this sample, compute the estimate of interest, e.g. average

$$\overline{X}^* = \frac{1}{n} \sum_{i=1}^{n} X_i^*$$

From Theorem 3.2, this $\overline{X}^*$ has *estimated variance* of

$$Var\overline{X}^* = \frac{1}{n(n-1)}\sum_{i=1}^{n}(X_i - \overline{X})^2 \tag{3.8}$$

In fact, this is a variance *under sampling scheme* (3.7), i.e. indicates one-time-sampling from $\hat{F}$. Using this, we can construct **bootstrap estimate of standard error for sample mean**, which come by sampling B times.

$$\hat{\sigma}_B = \left[\frac{1}{B-1}\sum_{b=1}^{B}(\overline{X}_b^* - \overline{X}_.^*)^2\right]^{\frac{1}{2}} \tag{3.9}$$

where $\overline{X}_b^*$ is a sample mean of b-th bootstrap replicate and $\overline{X}_.^*$ is average of every $\overline{X}_b^*$.

Theorem 3.3 (Bootstrap standard error)**.** *Bootstrap estimate of standard error for any estimator* $\hat{\theta}(X_1, \ldots, X_n)$ *is*

$$\hat{\sigma}_B = \left[\frac{1}{B-1}\sum_{b=1}^{B}(\hat{\theta}_b^* - \overline{\hat{\theta}^*})^2\right]^{\frac{1}{2}}$$

where $\hat{\theta}_b^*$ *is independent bootstrap replications and*

$$\overline{\hat{\theta}^*} = \frac{1}{B}\sum_{b=1}^{B}\hat{\theta}_b^*$$

Now we format Figure 3.12 to practical algorithm.

Algorithm 22: Bootstrap algorithm

```
  Data: $n$ observations $x_1, \ldots, x_n$
  input : statistic of interest $\hat{\theta}$, the number of bootstrap replicates $B$
1 for $b \leftarrow 1$ to $B$ do
2     Sampling with replacement $X_1^{(b)}, \ldots, X_n^{(b)}$ from the observed sample;
3     Compute estimate
          $\hat{\theta}(X_1^{(b)}, \ldots, X_n^{(b)}) \equiv \hat{\theta}_b^*$
      ;
4 end
5 $\overline{\hat{\theta}^*} = \frac{1}{B}\sum_{b=1}^{B}\hat{\theta}_b^*$;
6 Bootsrap standard error
      $\hat{\sigma}_B = \left[\frac{1}{B-1}\sum_{b=1}^{B}(\hat{\theta}_b^* - \overline{\hat{\theta}^*})^2\right]^{\frac{1}{2}}$
  ;
  output: $\hat{\sigma}_B$
```

Efron and Tibshirani (1994) suggests that $B = 40$ is usually enough to estimate standard error well. It rarely require $B > 200$. On the other hand, much larger B is needed in interval estimation.

First we try perform bootstrap without package doing bootstrap such as `boot` or `bootstrap`. The following is observed sample $n = 50$, i.e. finite population in bootstrap literature.

```
X
#> # A tibble: 50 x 1
#>        x
#>    <dbl>
#>  1 35.2
#>  2  3.24
#>  3 11.5
#>  4  6.00
#>  5 48.9
#>  6  2.71
#>  7 38.7
#>  8 34.0
#>  9 19.8
#> 10 40.8
#> # ... with 40 more rows
```

Sample mean of this sample is

```
X %>%
  summarise(x = mean(x))
#> # A tibble: 1 x 1
#>       x
#>   <dbl>
#> 1  22.2
```

```
MC_CORES <- parallel::detectCores() - 1 # parallelization
```

```
resample <- function(data, statistic = mean) {
  # sampling with replacement
  xb <-
    data %>%
    data.table() %>%
    .[sample(1:.N, size = .N, replace = TRUE)]
  # estimator
  xb[,
     lapply(.SD, statistic)] %>%
    as.numeric()
}
```

To fasten the process, we implement parallization `parallel::mclapply`. This function is based on fork mechanism of Unix OS. So this is not available in Windows OS.

```
B <- 40
#-------------------
Xse <-
  parallel::mclapply(
    1:B,
    function(x) {resample(data = X, statistic = mean)},
    mc.cores = MC_CORES
  ) %>%
    unlist()
```

```
tibble(se = Xse) %>%
  ggplot(aes(x = se)) +
  geom_histogram(bins = 30) +
  xlab(expression(hat(sigma)[B]))
```

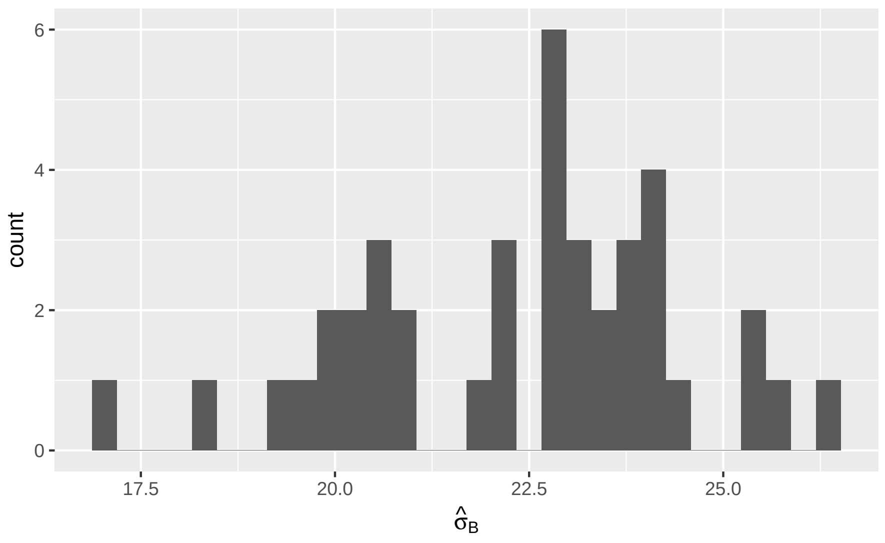


Figure 3.13: Bootstrap replicates of sample mean done by mclapply

Bootstrap standard error is given by

```
sd(Xse)
#> [1] 2.09
```

Another way and possible also in Windows OS is `foreach::foreach`. Using `%dopar%` with pre-specifed workers instead of `%do%`, we can parallize the jobs. `.inorder` argument enable the task done in order different with the other functions. Default is `FALSE` and it is more stable.

When we do `foreach` parallization, we should distribute the jobs to workers manually.

```
cl <- parallel::makeCluster(MC_CORES)
doParallel::registerDoParallel(cl, cores = MC_CORES)
parallel::clusterExport(cl, c("X", "resample"))
parallel::clusterEvalQ(cl, c(library(dplyr), library(data.table)))
#> [[1]]
#>  [1] "dplyr"      "stats"      "graphics"   "grDevices"  "utils"
#>  [6] "datasets"   "methods"    "base"       "data.table" "dplyr"
#> [11] "stats"      "graphics"   "grDevices"  "utils"      "datasets"
#> [16] "methods"    "base"
#>
#> [[2]]
#>  [1] "dplyr"      "stats"      "graphics"   "grDevices"  "utils"
#>  [6] "datasets"   "methods"    "base"       "data.table" "dplyr"
#> [11] "stats"      "graphics"   "grDevices"  "utils"      "datasets"
#> [16] "methods"    "base"
#>
#> [[3]]
#>  [1] "dplyr"      "stats"      "graphics"   "grDevices"  "utils"
#>  [6] "datasets"   "methods"    "base"       "data.table" "dplyr"
#> [11] "stats"      "graphics"   "grDevices"  "utils"      "datasets"
#> [16] "methods"    "base"
```

```
Xse_foreach <-
  foreach(b = 1:B, .combine = c, .inorder = FALSE) %dopar% {
    resample(data = X, statistic = mean)
  }
```

To end this process, make sure `stopCluster()`.

```
parallel::stopCluster(cl)
```

```
tibble(se = Xse_foreach) %>%
  ggplot(aes(x = se)) +
  geom_histogram(bins = 30) +
  xlab(expression(hat(sigma)[B]))
```

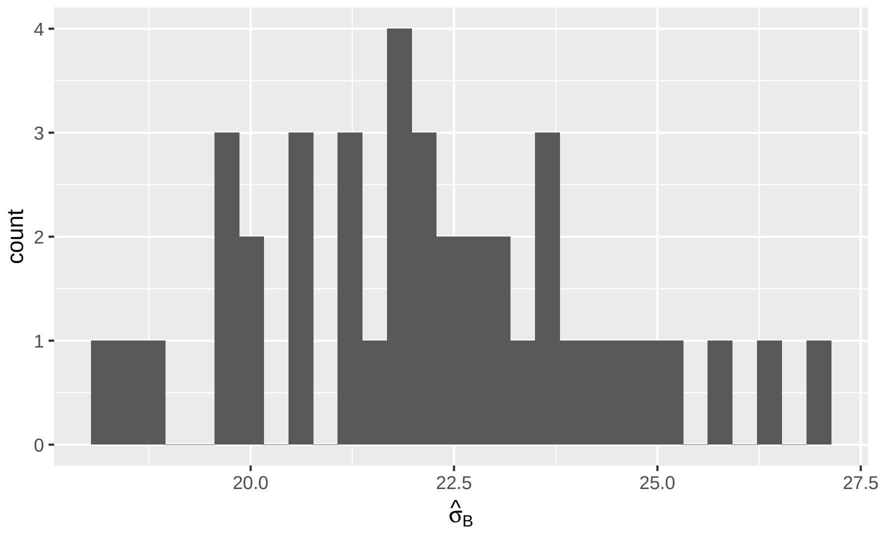


Figure 3.14: Bootstrap replicates for sample mean done by foreach

It gives boostrap standard error as

```
sd(Xse_foreach)
#> [1] 2.08
```

Are these kinds of parallization useful?

```
microbenchmark::microbenchmark(
  "MCLAPPLY2" = {
    parallel::mclapply(
      1:B,
      function(x) {resample(data = X, statistic = mean)},
      mc.cores = 2
    ) %>%
      unlist()
  },
  "MCLAPPLY3" = {
    parallel::mclapply(
      1:B,
      function(x) {resample(data = X, statistic = mean)},
      mc.cores = 3
    ) %>%
      unlist()
  },
  "MCLAPPLY4" = {
    parallel::mclapply(
      1:B,
      function(x) {resample(data = X, statistic = mean)},
      mc.cores = 4
    ) %>%
      unlist()
  },
  "FORLOOP" = {
    for (b in 1:B) {
      resample(data = X, statistic = mean)
    }
  },
  times = 5,
  unit = "s"
) %>%
  autoplot()
```

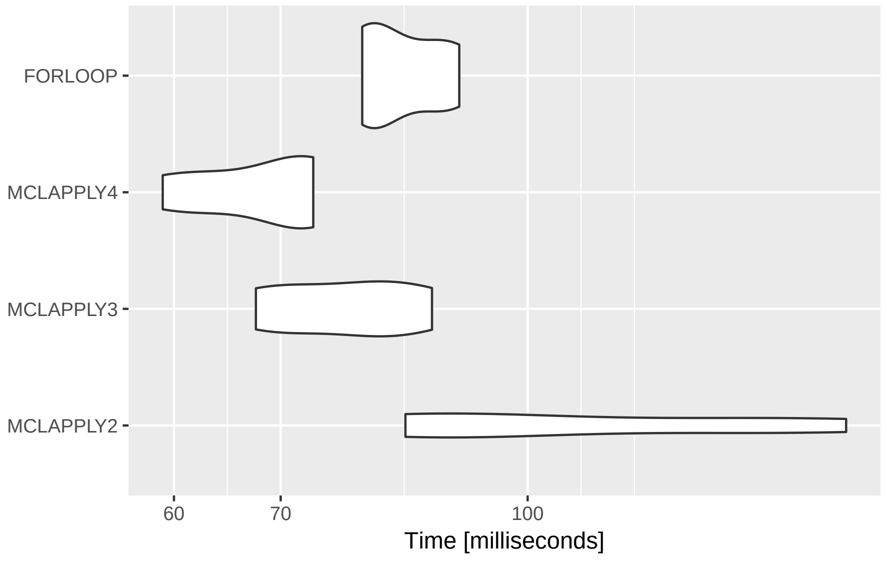


Figure 3.15: Benchmark between mclapply and for loop

Figure 3.15 is comparing `for` loop with each `mc.cores`. It is clear that parallization is faster than ordinary loop. In fact, all these procedures can be done by `boot` package.

```
library(boot)
```

Before performing bootstrap, we should define a `statistic` function. This function must take at least 2 arguments, data and index (`i`). About the second argument, `stype = c("i", "f", "w")` in `boot` is specifying in detail. Each `f` and `w` represents frequency and weight.

```
mean_boot <- function(x, i) {
  mean(x[i])
}
#-----------------------------
boot(data = X %>% pull(), statistic = mean_boot, R = B)
#>
#> ORDINARY NONPARAMETRIC BOOTSTRAP
#>
```

```
#>
#> Call:
#> boot(data = X %>% pull(), statistic = mean_boot, R = B)
#>
#>
#> Bootstrap Statistics :
#>     original  bias    std. error
#> t1*     22.2  -0.117        2.14
```

It gives sample mean `original`, bootstrap bias `bias`, and bootstrap se `std. error`. We will cover bias later. Due to the programming fact, this is much more faster than the previous one. Also, we can parallize with this function. `parallel = c("no", "multicore", "snow")`. If we choose `"multicore"` option, we should specify `ncpus` as in `mclapply`. If `"snow"`, cluster should be supplied in `cl` argument like in `foreach`.

```
boot(
  data = X %>% pull(),
  statistic = mean_boot,
  R = B,
  parallel = "multicore",
  ncpus = MC_CORES
)
#>
#> ORDINARY NONPARAMETRIC BOOTSTRAP
#>
#>
#> Call:
#> boot(data = X %>% pull(), statistic = mean_boot, R = B, parallel = "multicore",
#>     ncpus = MC_CORES)
#>
#>
#> Bootstrap Statistics :
#>     original  bias    std. error
#> t1*     22.2   0.304        2.31
```

```
microbenchmark::microbenchmark(
  "MCLAPPLY" = {
    parallel::mclapply(
      1:B,
      function(x) {resample(data = X, statistic = mean)},
      mc.cores = MC_CORES
    ) %>%
      unlist()
  },
  "BOOT" = {
    boot(
      data = X %>% pull(),
      statistic = mean_boot,
      R = B,
      parallel = "multicore",
      ncpus = MC_CORES
    )
  },
  times = 5,
  unit = "s"
) %>%
  autoplot()
```

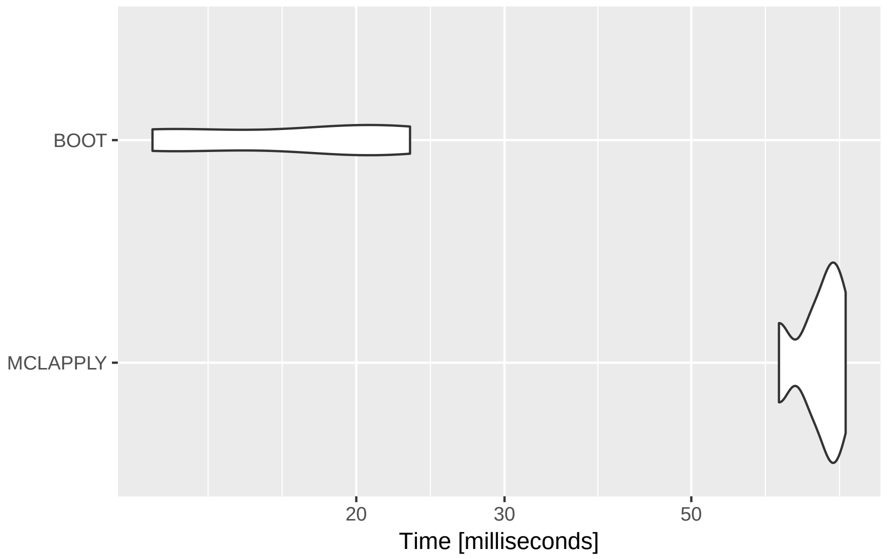


Figure 3.16: Benchmark between mclapply and boot

In Figure 3.16, we can see the difference of the speed.

3.6.4 Estimation of correlation coefficient

Consider traditional bootstrap example (Efron and Gong, 1983). The dataset is GPA scores of various entering classes at American law schools in 1973.

```
law %>%
  ggplot(aes(x = LSAT, y = GPA)) +
  geom_point()
```

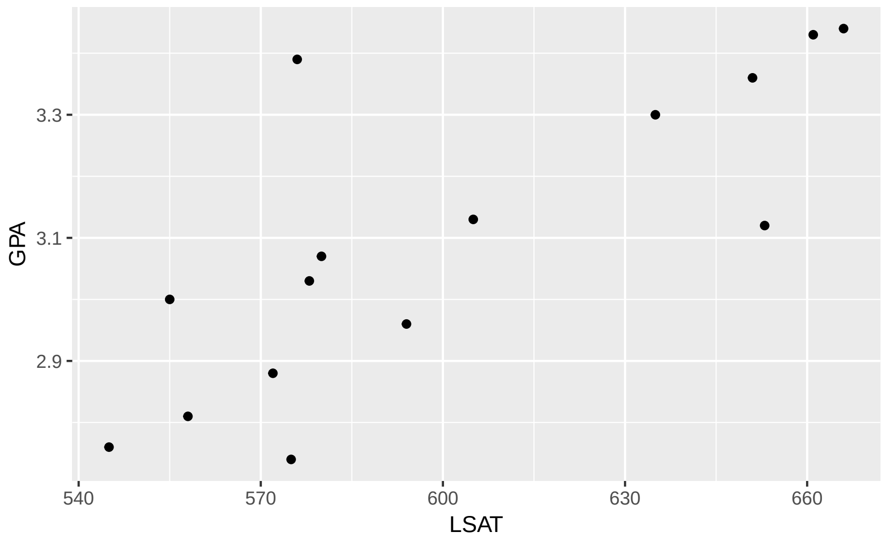


Figure 3.17: The law school data (Efron and Gong (1983))

Each point represents one law school.

- `LSAT`: average LSAT (national test) score of entering students
- `GPA`: average GPA score of entering students

Example 3.9 (Estimation of correlation coefficient)**.** In this $(Y_i, Z_i) = (\text{LSAT}, \text{GPA})$ data set, we are interested in the correlation ρ of the two variables. Especially, we want to explore the distribution of $\hat{\rho}$.

```r
law %>%
  summarise(rho = cor(LSAT, GPA))
#> # A tibble: 1 x 1
#>     rho
#>   <dbl>
#> 1 0.776
```

Let $\mathbf{X}_i^T = (Y_i, Z_i)$ be each observation. Dependency should be kept, so we should sample (Y_i, Z_i) pairs, not individuals.

Algorithm 23: Estimation of correlation coefficient - standard error

```
  input  : $n$ observations $\mathbf{X}_i^T = (Y_i, Z_i)$, $i = 1, \ldots, n$
1 for $b \leftarrow 1$ to $B$ do
2     Sampling with replacement $\mathbf{X}_1^{(b)}, \ldots, \mathbf{X}_n^{(b)}$ from the observed sample;
3     Sample correlation coefficient
          $$\hat{\rho}_b^* = \frac{\sum (Y_i^{(b)} - \overline{Y}^{(b)})(Z_i^{(b)} - \overline{Z}^{(b)})}{\sqrt{\sum (Y_i^{(b)} - \overline{Y}^{(b)})^2} \sqrt{\sum (Z_i^{(b)} - \overline{Z}^{(b)})^2}}$$
      ;
4 end
5 $\overline{\hat{\rho}^*} = \frac{1}{B} \sum_{b=1}^{B} \hat{\rho}_b^*$;
6 Bootsrap standard error
          $$\hat{\sigma}_B(\hat{\rho}) = \left[ \frac{1}{B-1} \sum_{b=1}^{B} (\hat{\rho}_b^* - \overline{\hat{\rho}^*})^2 \right]^{\frac{1}{2}}$$
  ;
  output: $\hat{\sigma}_B(\hat{\rho})$
```

Following Efron and Gong (1983), try $B = 1000$.

```r
boot_cor <- function(x, i) {
  cor(x[i, 1], x[i, 2])
}
#---------------------------
B <- 1000
(rho <- boot(law, statistic = boot_cor, R = B,
            parallel = "multicore", ncpus = MC_CORES))
#>
#> ORDINARY NONPARAMETRIC BOOTSTRAP
#>
#>
#> Call:
#> boot(data = law, statistic = boot_cor, R = B, parallel = "multicore",
#>     ncpus = MC_CORES)
#>
```

```
#>
#> Bootstrap Statistics :
#>     original    bias    std. error
#> t1*    0.776 -0.00531       0.137
```

Lemma 3.5 (Gaussian standard error of $\hat{\rho}$)**.** *When the data follow Normal distribution, the standard error of $\hat{\rho}$ can be estimated by*

$$\hat{\sigma}_{NORM} = \frac{1 - \hat{\rho}^2}{\sqrt{n-3}}$$

- `t0` is statistic computed from the sample, i.e. correlation coefficient of the data
- `t` is each bootstrap replicate, `matrix` object

Using `t`, we might draw empirical distribution and get standard error. In the real world, many data for scores follow normal. To check bootstrap works well, we compare the empirical distribution of `rho$t` and one with Lemma 3.5.

$$\hat{\sigma}_{NORM} = \frac{1 - \hat{\rho}^2}{\sqrt{n-3}} = 0.115$$

Construct $\hat{F}$ by

$$\hat{F}_{NORM} \sim MVN\Big(\overline{\mathbf{x}}, \frac{n-1}{n}S\Big)$$

To see how normal population work, we *draw bootstrap sample from the parametric maximum likeihood distribution.*

$$X_1^*, \ldots, X_n^* \sim \hat{F}_{NORM}$$

`boot` has arguments `sim`, `ran.gen` and `mle`.

- `sim`: type of simulation method. By default, `"ordinary"`. In this case, change this to `"parametric"`.
- `ran.gen`: if `sim = "parametric"`, we should specify `ran.gen` generating random values. Function should have two arguments of `data` and `mle`.
- `mle`: Second argument of `ran.gen`. MLE of parameters.

```
# mle --------------------------
lawmu <-
  law %>%
  summarise_all(mean) %>%
  as.numeric()
lawcov <- cov(law) * nrow(law) / (nrow(law) - 1)
# ran.gen ----------------------
gen_mvn <- function(data, mle) {
  mvtnorm::rmvnorm(nrow(data), mean = mle[[1]], sigma = mle[[2]])
}
# boot -------------------------
(rho_norm <-
  boot(
    law,
    statistic = boot_cor,
    R = B,
    sim = "parametric",
    ran.gen = gen_mvn,
    mle = list(lawmu, lawcov),
    parallel = "multicore",
    ncpus = MC_CORES
  ))
#>
#> PARAMETRIC BOOTSTRAP
#>
#>
#> Call:
#> boot(data = law, statistic = boot_cor, R = B, sim = "parametric",
#>     ran.gen = gen_mvn, mle = list(lawmu, lawcov), parallel = "multicore",
#>     ncpus = MC_CORES)
#>
#>
#> Bootstrap Statistics :
#>     original  bias    std. error
#> t1*    0.776 -0.0116       0.119
```

As $B \to \infty$, $\hat{\sigma}_B$ approximates $\hat{\sigma}_{NORM} = 0.115$. See the similarity of the two values.

```
tibble(
  rho1 = rho$t[,1],
  rho2 = rho_norm$t[,1]
) %>%
  ggplot() +
  geom_histogram(bins = 30, aes(x = rho1, y = ..density..)) +
  stat_density(aes(x = rho2, y = ..density..), geom = "line", col = gg_hcl(1)) +
  xlab(expression(hat(rho)^"*"))
```

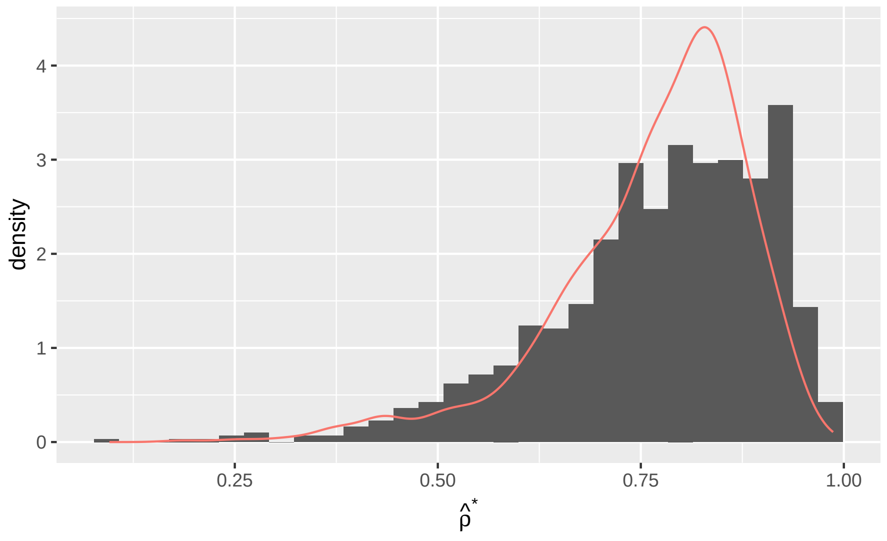


Figure 3.18: Bootstrap replicates for correlation in law school data - histogram of ordinary, line of parametric

See Figure 3.18. Normal density has a similar shape to ordinary bootstrap, except that normal bootstrap falls off more quickly at the right tail.

3.6.5 Boostrap bias

Definition 3.5 (Bias)**.** Bias of a estimator θ is

$$\beta := E(\hat{\theta}) - \theta$$

Note that β is a kind of function of the unkown probability ditribution F.

$$\beta = \beta(F)$$

In this sense, bootstrap estimate of β can be given as

$$\hat{\beta}_B = \beta(\hat{F}) = E_{F^*}\left[\theta(\hat{F}^*) - \theta(\hat{F})\right] \tag{3.10}$$

Here, expectation E_{F^*} can be approximated by Monte Carlo methods. Sampling

$$X_1^*, \ldots, X_n^* \overset{iid}{\sim} F^*$$

compute

$$\hat{\beta}_B \approx \overline{\hat{\theta}^*} - \hat{\theta} = \frac{1}{B}\sum_{b=1}^{B}(\hat{\theta}_b^* - \hat{\theta}) \tag{3.11}$$

In Algorithm 23, we only need to replace Step 6 with above Equation (3.11).

Theorem 3.4 (Bootstrap Bias)**.** *Bootstrap estimate of bias for any estimator $\hat{\theta}(X_1, \ldots, X_n)$ of θ is*

$$\hat{\beta}_B = \overline{\hat{\theta}^*} - \hat{\theta}$$

where $\overline{\hat{\theta}^*} = \frac{1}{B}\sum\limits_{b=1}^{B} \hat{\theta}_b^*$ and $\hat{\theta} = \hat{\theta}(X_1, \ldots, X_n)$, i.e. one estimated by observed sample

Is it reasonable to compare $\overline{\hat{\theta}^*}$ with $\hat{\theta}$ even though we do not know the true one? This is natural from bootstrap construction. Bootstrap takes its population as observed sample of which distribution is $\hat{F}$. So $\hat{\theta}$

of pseudo-population, $\theta(\hat{F})$ of Equation (3.10), is able to represent the true value.

Algorithm 24: Estimation of correlation coefficient - bias

```
    input  : $n$ observations $\mathbf{X}_i^T = (Y_i, Z_i),\ i = 1, \ldots, n$
  1 for $b \leftarrow 1$ to $B$ do
  2     Sampling with replacement $\mathbf{X}_1^{(b)}, \ldots, \mathbf{X}_n^{(b)}$ from the observed sample;
  3     Sample correlation coefficient
            $$\hat{\rho}_b^* = \frac{\sum(Y_i^{(b)} - \overline{Y}^{(b)})(Z_i^{(b)} - \overline{Z}^{(b)})}{\sqrt{\sum(Y_i^{(b)} - \overline{Y}^{(b)})^2}\sqrt{\sum(Z_i^{(b)} - \overline{Z}^{(b)})^2}}$$
        ;
  4 end
  5 $\overline{\hat{\rho}^*} = \frac{1}{B}\sum_{b=1}^{B} \hat{\rho}_b^*$;
  6 Bootsrap bias
            $$\hat{\beta}_B = \overline{\hat{\rho}^*} - \hat{\rho}$$
    ;
    output: $\hat{\beta}_B$
```

If

$$\frac{\hat{\beta}_B}{\hat{\sigma}_B} < \frac{1}{4}$$

then it might be okay to ignore the bias, i.e. not necessary to adjust for it(Efron and Tibshirani, 1994).

Refer to Example 3.9. `boot::boot()` have given following output.

```
rho
#> 
#> ORDINARY NONPARAMETRIC BOOTSTRAP
#> 
#> 
#> Call:
#> boot(data = law, statistic = boot_cor, R = B, parallel = "multicore",
#>     ncpus = MC_CORES)
#> 
#> 
#> Bootstrap Statistics :
#>     original    bias    std. error
#> t1*    0.776 -0.00531       0.137
```

$\hat{\beta}_B$ is `bias`.

Example 3.10 (Bootstrap estimate of a ratio estimate)**.** Consider medical patch data from Efron and Tibshirani (1994). It contains measurement for hormone into the blood stream of 8 subjects after wearing a medical patch. There are three different patches.

- placebo patch
- old patch, manufactured at an older plant
- new patch, manufactured at a new plant

Define a parameter bioequivalence by

$$\theta = \frac{E(new) - E(old)}{E(old) - E(placebo)} \leq 0.2$$

This is the parameter of our interest.

```
as_tibble(bootstrap::patch)
#> # A tibble: 8 x 6
#>   subject placebo oldpatch newpatch     z     y
#>     <int>   <dbl>    <dbl>    <dbl> <dbl> <dbl>
#> 1       1    9243    17649    16449  8406 -1200
#> 2       2    9671    12013    14614  2342  2601
#> 3       3   11792    19979    17274  8187 -2705
#> 4       4   13357    21816    23798  8459  1982
#> 5       5    9055    13850    12560  4795 -1290
#> 6       6    6290     9806    10157  3516   351
#> 7       7   12412    17208    16570  4796  -638
#> 8       8   18806    29044    26325 10238 -2719
```

In `patch`,

- `z`: `oldpatch` - `placebo`
- `y`: `newpatch` - `oldpatch`

We only need these two columns. `z` goes to denominator, `y` to numerator.

```
bioequiv <- function(x, i) {
  # select(z, y)
  mean(x[i, 2]) / mean(x[i, 1])
}
#---------------------------
B <- 2000
(ratio <-
  bootstrap::patch %>%
  select(z, y) %>%
  boot(
    statistic = bioequiv,
    R = B,
    parallel = "multicore",
    ncpus = MC_CORES
  ))
#> 
#> ORDINARY NONPARAMETRIC BOOTSTRAP
#> 
#> 
#> Call:
#> boot(data = ., statistic = bioequiv, R = B, parallel = "multicore",
#>     ncpus = MC_CORES)
#> 
#> 
#> Bootstrap Statistics :
#>     original  bias    std. error
#> t1*  -0.0713 0.00708      0.0994
```

`broom::tidy()` gives tidies a `boot` object so that we can deal with above three statistic more easily.

```
(ratio_tidy <-
  broom::tidy(ratio) %>%
    mutate(
      bias_se = bias / std.error,
      ignore_bias = ifelse(bias_se < .25, "ignore bias", "adjust for bias")
    ))
#> # A tibble: 1 x 5
#>   statistic    bias std.error bias_se ignore_bias
#>       <dbl>   <dbl>     <dbl>   <dbl> <chr>
#> 1   -0.0713 0.00708    0.0994  0.0712 ignore bias
```

Since $\frac{\hat{\beta}_B}{\hat{\sigma}_B} = 0.071$, we ignore bias.

3.7 Jackknife

Jackknife is another resampling method, which was developed earlier than bootstrap. Instead of bootstrap sample $\mathbf{X}^{(b)}$, *jackknife sample* $\mathbf{X}_{(i)}$ will be used. It leaves out the i-th observation.

$$\mathbf{X}^{(i)} = (X_1^{(i)}, \ldots, X_{i-1}^{(i)}, X_{i+1}^{(i)}, \ldots, X_n^{(i)})^T \tag{3.12}$$

For each $\mathbf{X}^{(i)}$, *jackknife replicate* which is a form of our interest is computed.

$$\hat{\theta}^{(i)} = \hat{\theta}(\mathbf{X}^{(i)}) \tag{3.13}$$

so that we can get its empirical distribution.

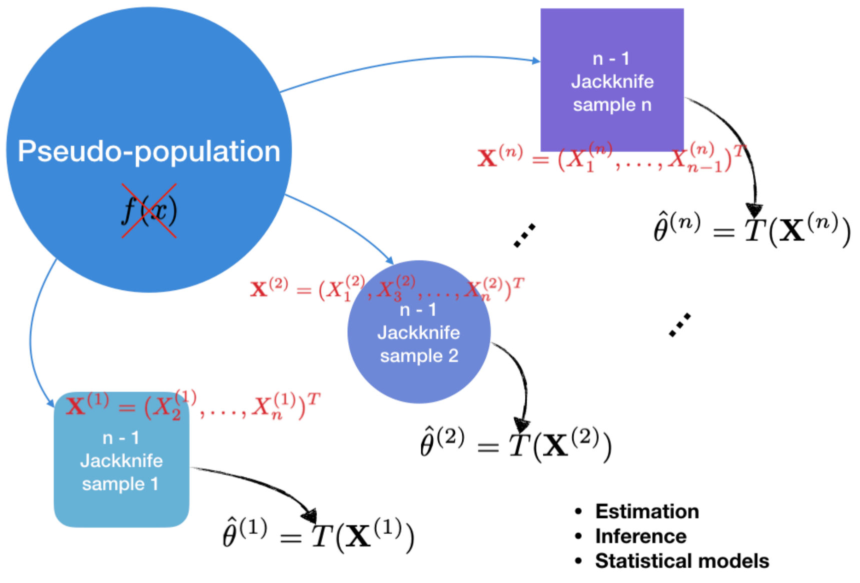

Pseudo-population
$f(x)$
n - 1 Jackknife sample n
$\mathbf{X}^{(n)} = (X_1^{(n)}, \ldots, X_{n-1}^{(n)})^T$
$\hat{\theta}^{(n)} = T(\mathbf{X}^{(n)})$
$\mathbf{X}^{(2)} = (X_1^{(2)}, X_3^{(2)}, \ldots, X_n^{(2)})^T$
n - 1 Jackknife sample 2
$\hat{\theta}^{(2)} = T(\mathbf{X}^{(2)})$
$\mathbf{X}^{(1)} = (X_2^{(1)}, \ldots, X_n^{(1)})^T$
n - 1 Jackknife sample 1
$\hat{\theta}^{(1)} = T(\mathbf{X}^{(1)})$
Estimation
Inference
Statistical models

Figure 3.19: Leave-one-out sampling

3.7.1 Jackknife standard error

As in Section 3.6.2, we start with simple example, estimating sample mean. Refer to Example 3.7. We have seen estimated standard error $\hat{\sigma}$ and boostrap estimate of standard error $\hat{\sigma}_B$ in Theorems 3.2 and 3.3.

Theorem 3.5 (Jackknife standard error for sample mean)**.** *Let*

$$\overline{X}_{(i)} := \frac{n\overline{X} - X_i}{n-1} = \frac{1}{n-1}\sum_{j \neq i} X_j$$

be the sample average of the set deleting i-th point. Write average of the deleted averages by $\overline{X}_{(.)} = \frac{1}{n}\sum\limits_{i=1}^{n}\overline{X}_{(i)}$. *By construction,*

$$\overline{X}_{(.)} = \overline{X}$$

Jackknife estimate of standard error for sample mean $\overline{X}$ *is*

$$\hat{\sigma}_J = \left[\frac{n-1}{n}\sum_{i=1}^{n}\left(\overline{X}_{(i)} - \overline{X}_{(.)}\right)^2\right]^{\frac{1}{2}}$$

Remark.

$$\hat{\sigma} = \hat{\sigma}_J$$

Proof. Note that

$$\overline{X}_{(i)} = \frac{n\overline{X} - X_i}{n-1}$$

By construction,

$$\begin{aligned}
\overline{X}_{(.)} &= \frac{1}{n}\sum_{i=1}^{n}\left[\frac{n\overline{X} - X_i}{n-1}\right] \\
&= \frac{n\overline{X}}{n-1} - \frac{\overline{X}}{n-1} \\
&= \overline{X}
\end{aligned}$$

It follows that

$$\begin{aligned}
\hat{\sigma}_J^2 = \frac{n-1}{n}\sum_{i=1}^{n}\left(\overline{X}_{(i)} - \overline{X}_{(.)}\right)^2 &= \frac{n-1}{n}\sum_{i=1}^{n}\left(\frac{n\overline{X} - X_i}{n-1} - \overline{X}\right)^2 \\
&= \frac{n-1}{n}\sum_{i=1}^{n}\left(\frac{\overline{X} - X_i}{n-1}\right)^2 \\
&= \frac{1}{n(n-1)}\sum_{i=1}^{n}\left(X_i - \overline{X}\right)^2 \\
&= \hat{\sigma}^2
\end{aligned}$$

□

Using a set of jackknife replicates, get empirical distribution of this estimator such as standard error and bias.

Algorithm 25: Jackknife algorithm

```
   Data: $n$ observations $x_1, \dots, x_n$
   input : statistic of interest $\hat{\theta}$
1  for $i \leftarrow 1$ to $n$ do
2      Subset of $\mathbf{X}$ that leaves out the $i$-th observation $X_1, \dots, X_{i-1}, X_{i+1}, \dots, X_n$;
3      Compute estimate
           $$\hat{\theta}(X_1, \dots, X_{i-1}, X_{i+1}, \dots, X_n) \equiv \hat{\theta}_{(i)}$$
       ;
4  end
5  $\hat{\theta}_{(.)} = \frac{1}{n} \sum_{i=1}^{n} \hat{\theta}_{(i)}$;
6  Jackknife standard error
           $$\hat{\sigma}_J = \left[ \frac{n-1}{n} \sum_{i=1}^{n} (\hat{\theta}_{(i)} - \hat{\theta}_{(.)})^2 \right]^{\frac{1}{2}}$$
   ;
   output: $\hat{\sigma}_J$
```

Now extend to any statistic such as correlation 3.9. Let $\hat{\theta} \equiv \hat{\theta}(X_1, \dots, X_n)$ be the statistic of interest. Suppose that

$$X_1, \dots, X_n \overset{iid}{\sim} F$$

Let $\mathbf{P}$ be a probability mass vector for each X_i and let $\mathbf{P}^*$ be a vector for bootstrap sample. Recall that each point has a mass of

$$\frac{1}{n}$$

Then we have

$$\mathbf{P}^* \sim Multi(n, \mathbf{P}_0)$$

with $\mathbf{P}_0 = \frac{1}{n}\mathbf{1} \in \mathbb{R}^n$. We already have seend that the bootstrap standard error is

$$\hat{\sigma}_B = \left[Var\hat{\theta}(\mathbf{P}^*) \right]^{\frac{1}{2}}$$

In comparison, the jackknife resamples leaving out one observation so that

$$\mathbf{P}_{(i)} = \frac{1}{n-1}(1, \dots, 1, \overset{i\text{-th}}{0}, 1, \dots, 1)^T, \quad i = 1, \dots, n$$

Efron and Gong (1983) draws a picture that indicates this difference between resampling procedure of bootstrap and jackknife. Among $\{X_1, X_2, X_3\}$, bootstrap sample with replacement with probability $\frac{1}{3}$. Jackknife leaves out one in each sample.

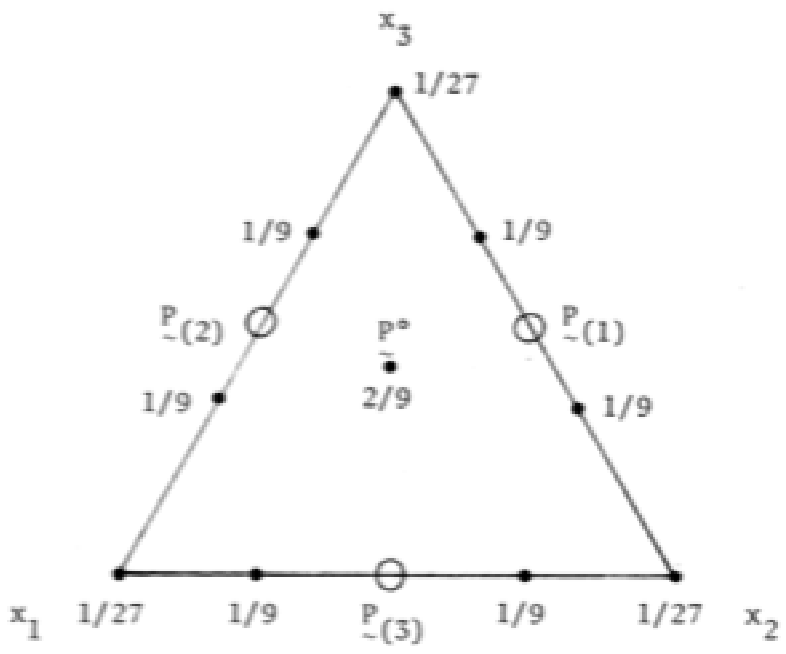


Figure 3.20: Bootstrap and Jackknife sampling points - · is bootstrap and ∘ is jackknife

Lemma 3.6. *Jackknife estimation approximates $\hat{\theta}(\mathbf{P})$ by a linear function of $\mathbf{P}$, say $\hat{\theta}_L(\mathbf{P})$.*

$$\hat{\theta}_L(\mathbf{P}) = \hat{\theta}_{(.)} + (\mathbf{P} - \mathbf{P}^0)^T U$$

where $\hat{\theta}_{(.)} = \frac{1}{n}\sum_i \hat{\theta}_{(i)} = \frac{1}{n}\sum_i \hat{\theta}(\mathbf{P}_{(i)})$, $U_i = (n-1)(\hat{\theta}_{(.)} - \hat{\theta}_{(.)})$, and $U = \begin{bmatrix} U_1 & \cdots & U_n \end{bmatrix}$.

Theorem 3.6 (Jackknife standard error)**.** *Let $\hat{\theta}_L$ of Lemma 3.6 be the statistic of interest. Then the Jackknife estimate of standard error for $\hat{\theta}_L$ is*

$$\hat{\sigma}_J = \left[\frac{n}{n-1} Var\hat{\theta}_L(\mathbf{P}^*)\right]^{\frac{1}{2}}$$

where $\hat{\theta}_L(\mathbf{P}^*)$ is the bootstrap estimate.

```r
cor_jack <- function(x) {
  cor(x[,1], x[,2])
}
#----------------------
jack_cor <- function(x, data) {
  # sampling with replacement
  xb <-
    data %>%
    data.table() %>%
    .[-x]
  # estimator
  cor(xb[,1], xb[,2]) %>% as.numeric()
}
#---------------------
law_jack <-
```

```
  parallel::mclapply(
    1:nrow(law),
    jack_cor,
    data = law,
    mc.cores = MC_CORES
  ) %>%
  unlist()
#--------------------
sqrt((nrow(law) - 1) * mean((law_jack - mean(law_jack))^2))
#> [1] 0.143
```

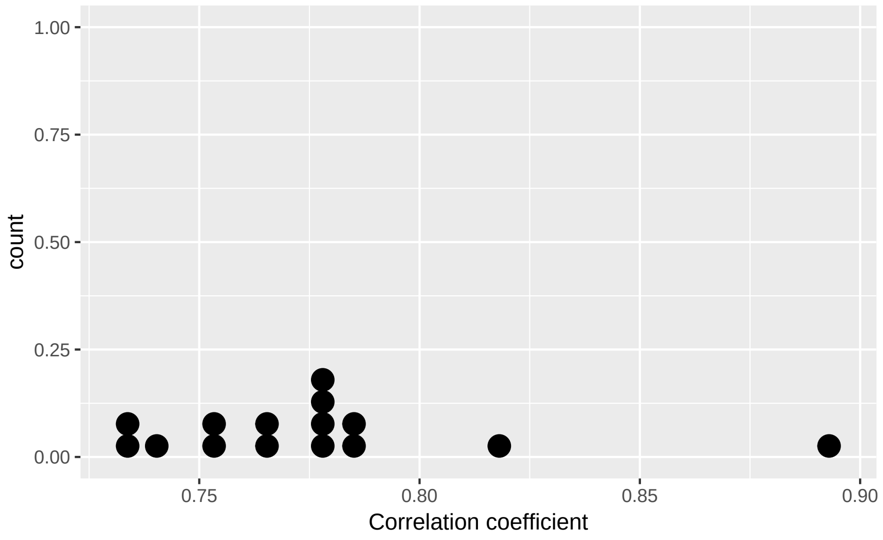


Figure 3.21: Jackknife estimate of correlation

Visually, the distribution is quite similar to of bootstrap 3.18. Also, standard error is not that different.

`bootstrap` library provides `jackknife()` function. Since this kind of *leave-one-out* procedure does not have randomness, the result is exactly same.

```
cor_law <- function(x, xdata) {
  cor(xdata[x, 1], xdata[x, 2])
}
#------------------------------
bootstrap::jackknife(1:nrow(law), cor_law, xdata = law)
#> $jack.se
#> [1] 0.143
#>
#> $jack.bias
#>           GPA
#> LSAT -0.00647
#>
#> $jack.values
#>  [1] 0.893 0.764 0.755 0.776 0.731 0.780 0.785 0.736 0.752 0.776 0.818
#> [12] 0.786 0.740 0.767 0.780
#>
#> $call
#> bootstrap::jackknife(x = 1:nrow(law), theta = cor_law, xdata = law)
```

3.7.2 Jackknife bias

Consider bias β of $\hat{\theta} = \hat{\theta}(\hat{F})$ 3.5.

Lemma 3.7 (Quenouille's estimate for bias)**.** *In the notation of Lemma 3.6, Quenouille's estimate for bias is*

$$\hat{\beta}_J = (n-1)(\hat{\theta}_{(.)} - \hat{\theta})$$

As in the previous section, there is a relationship between a bias estimate 3.7 and Bootstrap bias 3.4.

Lemma 3.8. *Jackknife estimation approximates $\hat{\theta}(\mathbf{P})$ by a quadratic function of $\mathbf{P}$, say $\hat{\theta}_Q(\mathbf{P})$.*

$$\hat{\theta}_Q(\mathbf{P}_0) = a + (\mathbf{P} - \mathbf{P}_0)^T \mathbf{b} + \frac{1}{2}(\mathbf{P} - \mathbf{P}_0)^T \mathbf{c} (\mathbf{P} - \mathbf{P}_0)$$

From this Lemma, Jackknife bias $\hat{\beta}_J$ can be derived as follows.

Theorem 3.7 (Jackknife bias)**.** *Let $\hat{\theta}_Q(\mathbf{P})$ be any quadratic satisfying*

$$\hat{\theta}_Q(\mathbf{P}_0) = \hat{\theta}(\mathbf{P}_0) = \hat{\theta} \quad \textit{and} \quad \hat{\theta}_Q(\mathbf{P}_{(i)}) = \hat{\theta}(\mathbf{P}_{(i)})$$

Then the jackknife estimate of bias is

$$\beta_J = \frac{n}{n-1}\Big[E(\hat{\theta}_Q(\mathbf{P}^*) - \hat{\theta})\Big]$$

i.e. $\frac{n}{n-1}$ times the bootstrap bias for $\hat{\theta}_Q$.

3.8 Bootstrap Confidence Intervals

Bootstrap gives empirical distribution of a estimator. Naturally, we want to get confidence interval of this estimator by using standard error or by just arranging replicates. See Figure 3.13 or 3.18. These are empirical distribution.

3.8.1 Standard normal bootstrap confidence interval

Standard normal bootstrap confidence interval is the simplest approach, but not necessarily the best (Rizzo, 2007). If the estimator of interest $\hat{\theta}$ follows Normal distribution, the only thing we have to do is computing standard error. When is this case? If $\hat{\theta}$ is a sample mean,

$$Z = \frac{\hat{\theta} - E\hat{\theta}}{\hat{\sigma}_B} \xrightarrow{d} N(0,1) \quad \text{as } n \to \infty \tag{3.14}$$

by the *Central limit theorem.* Thus, $100(1-\alpha)\%$ confidence for θ can be computed in this frame

$$\hat{\theta} \pm z_{\frac{\alpha}{2}} \hat{\sigma}_B \tag{3.15}$$

Remark (Assumptions of standard normal bootstrap CI)*.* To compute this CI, some entries should be assumed about $\hat{\theta}$.

1. $\hat{\theta}$ is Normal, or is sample mean with large sample size (CLT-based)

2. $\hat{\theta}$ is unbiased.

In case of biasedness, it should be corrected. Estimating bias by bootstrap by $\hat{\beta}_B$,

$$\hat{\theta} - \hat{\beta}_B$$

Hence, we replace $\hat{\theta}$ in CI (3.15) with above corrected one.

Theorem 3.8 (Standard normal bootstrap CI)**.** *For the estimator of interest $\hat{\theta}$, standard normal bootstrap CI is*

$$[\hat{\theta} - \hat{\beta}_B] \pm z_{\frac{\alpha}{2}} \hat{\sigma}_B$$

If $\hat{\theta}$ is unbiased, correction is omitted.

$$\hat{\theta} \pm z_{\frac{\alpha}{2}} \hat{\sigma}_B$$

Algorithm 26: Bootstrap algorithm for standard normal bootstrap CI

```
   Data: $n$ observations $x_1, \ldots, x_n$
   input : statistic of interest $\hat{\theta}$, the number of bootstrap replicates $B$
 1 for $b \leftarrow 1$ to $B$ do
 2     Sampling with replacement $X_1^{(b)}, \ldots, X_n^{(b)}$ from the observed sample;
 3     Compute estimate $\hat{\theta}(X_1^{(b)}, \ldots, X_n^{(b)}) \equiv \hat{\theta}_b^*$;
 4 end
 5 $\overline{\hat{\theta}^*} = \frac{1}{B} \sum_{b=1}^{B} \hat{\theta}_b^*$;
 6 Bootsrap standard error $\hat{\sigma}_B = \left[ \frac{1}{B-1} \sum_{b=1}^{B} (\hat{\theta}_b^* - \overline{\hat{\theta}^*})^2 \right]^{\frac{1}{2}}$;
 7 Bootsrap bias $\hat{\beta}_B = \overline{\hat{\theta}^*} - \hat{\theta}(X_1, \ldots, X_n)$;
 8 if $\frac{\hat{\beta}_B}{\hat{\sigma}_B} < \frac{1}{4}$ then
 9     Standard Normal Bootstrap CI
           $$\hat{\theta} \pm z_{\frac{\alpha}{2}} \hat{\sigma}_B$$
       ;
10 else
11     Standard Normal Bootstrap CI
           $$[\hat{\theta} - \hat{\beta}_B] \pm z_{\frac{\alpha}{2}} \hat{\sigma}_B$$
       ;
12 end
   output: Standard normal bootstrap CI
```

Bias correction, however, is not just constant subtraction. $\hat{\beta}_B$ is also an random variable, so the transformed random variable might not have Normal distribution.

3.8.2 Basic bootstrap confidence interval

Instead of using bootstrap standard error for observed statistic, i.e. $\hat{\theta}$, the basic bootstrap CI transforms the distribution of $\hat{\theta}_b^*$ by subtracting $\hat{\theta}$. Consider $\frac{\alpha}{2}$ and $1 - \frac{\alpha}{2}$ quantile values $\hat{\theta}^*_{\frac{\alpha}{2}}$ and $\hat{\theta}^*_{1-\frac{\alpha}{2}}$. Consider random variable $\hat{\theta} - \theta$.

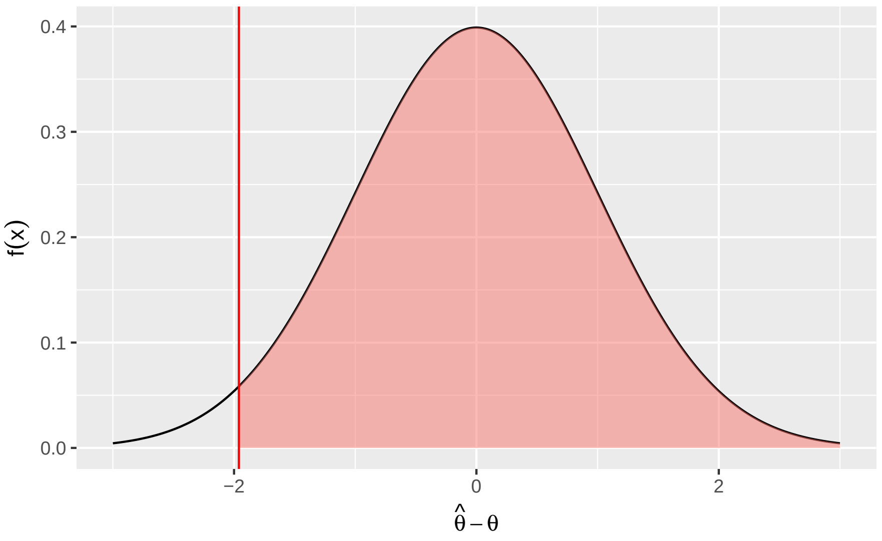


Figure 3.22: $P(\hat{\theta} - \theta > \alpha_{\frac{\alpha}{2}}) = 1 - \alpha$

Let $\alpha_{\frac{\alpha}{2}}$ be the $\frac{\alpha}{2}$ quantile of $\hat{\theta} - \theta$. Then

$$P(\hat{\theta} - \theta > \alpha_{\frac{\alpha}{2}}) = 1 - \alpha$$

See Figure 3.22 for this. It follows that a $100(1 - \alpha)\%$ CI with symmetricity

$$(\hat{\theta} - \alpha_{1-\frac{\alpha}{2}}, \hat{\theta} - \alpha_{\frac{\alpha}{2}})$$

However, α is unknown. So use bootstrap replicates $\hat{\theta}_b^* - \hat{\theta}$. Since $\hat{\theta}$ is not changed for given sample, we can estimate $\frac{\alpha}{2}$ quantile by

$$\hat{\theta}_{\frac{\alpha}{2}}^* - \hat{\theta}$$

$\alpha_{\frac{\alpha}{2}}$ is replaced by this, so upper limit would be

$$\hat{\theta} - (\hat{\theta}_{\frac{\alpha}{2}}^* - \hat{\theta}) = 2\hat{\theta} - \hat{\theta}_{\frac{\alpha}{2}}^*$$

One proceeds in a similar way for $1 - \frac{\alpha}{2}$ quantile that

$$\hat{\theta} - (\hat{\theta}_{1-\frac{\alpha}{2}}^* - \hat{\theta}) = 2\hat{\theta} - \hat{\theta}_{1-\frac{\alpha}{2}}^*$$

Theorem 3.9 (Basic bootstrap CI)**.** *Let $\hat{\theta}_{\frac{\alpha}{2}}^*$ and $\hat{\theta}_{1-\frac{\alpha}{2}}^*$ be $\frac{\alpha}{2}$ and $1 - \frac{\alpha}{2}$ quantile values, respectively, obtained from the bootstraped empirical distribution. Then the basic bootstrap confidence interval is*

$$(2\hat{\theta} - \hat{\theta}_{1-\frac{\alpha}{2}}^*, 2\hat{\theta} - \hat{\theta}_{\frac{\alpha}{2}}^*)$$

In other words, what we need is quantiles, not standard error.

Algorithm 27: Bootstrap algorithm for basic bootstrap CI

```
   Data: $n$ observations $x_1, \ldots, x_n$
   input : statistic of interest $\hat{\theta}$, the number of bootstrap replicates $B$
 1 for $b \leftarrow 1$ to $B$ do
 2     Sampling with replacement $X_1^{(b)}, \ldots, X_n^{(b)}$ from the observed sample;
 3     Compute estimate $\hat{\theta}(X_1^{(b)}, \ldots, X_n^{(b)}) \equiv \hat{\theta}_b^*$;
 4 end
 5 $1 - \frac{\alpha}{2}$ and $\frac{\alpha}{2}$ quantile values $\hat{\theta}^*_{1-\frac{\alpha}{2}}$ and $\hat{\theta}^*_{\frac{\alpha}{2}}$;
 6 Basic bootstrap CI
       $$(2\hat{\theta} - \hat{\theta}^*_{1-\frac{\alpha}{2}}, 2\hat{\theta} - \hat{\theta}^*_{\frac{\alpha}{2}})$$
     ;
   output: Basic bootstrap CI
```

3.8.3 Percentile bootstrap confidence interval

Percentile bootstrap confidence interval is quite intuitive. It just finds upper and lower quantile of empirical distribution.

Theorem 3.10 (Percentile bootstrap CI)**.** *Let $\hat{\theta}^*_{\frac{\alpha}{2}}$ and $\hat{\theta}^*_{1-\frac{\alpha}{2}}$ be $\frac{\alpha}{2}$ and $1 - \frac{\alpha}{2}$ quantile values, respectively, obtained from the bootstraped empirical distribution. Then the basic bootstrap confidence interval is*

$$(\hat{\theta}^*_{\frac{\alpha}{2}}, \hat{\theta}^*_{1-\frac{\alpha}{2}})$$

Process of getting the CI is similar to Algorithm 27 but 6.

Algorithm 28: Bootstrap algorithm for percentile bootstrap CI

```
   Data: $n$ observations $x_1, \ldots, x_n$
   input : statistic of interest $\hat{\theta}$, the number of bootstrap replicates $B$
 1 for $b \leftarrow 1$ to $B$ do
 2     Sampling with replacement $X_1^{(b)}, \ldots, X_n^{(b)}$ from the observed sample;
 3     Compute estimate $\hat{\theta}(X_1^{(b)}, \ldots, X_n^{(b)}) \equiv \hat{\theta}_b^*$;
 4 end
 5 $1 - \frac{\alpha}{2}$ and $\frac{\alpha}{2}$ quantile values $\hat{\theta}^*_{1-\frac{\alpha}{2}}$ and $\hat{\theta}^*_{\frac{\alpha}{2}}$;
 6 Percentile bootstrap CI
       $$(\hat{\theta}^*_{\frac{\alpha}{2}}, \hat{\theta}^*_{1-\frac{\alpha}{2}})$$
     ;
   output: Basic bootstrap CI
```

3.8.4 Bootstrap CI in `R`

`boot` package has a function `boot.ci()`. This is used for `boot` object. Refer to Example 3.10. By specifying `type` argument, we can get various confidence intervals.

- `"norm"`: Standard normal bootstrap confidence interval
- `"basic"`: Basic bootstrap confidence interval
- `"stud"`: Bootstrap t interval
- `"perc"`: Percentile bootstrap confidence interval
- `"bca"`: Better bootstrap confidence interval (BCa)

Confidence level is set to be `conf = .95` by default.

```
boot.ci(ratio, type = c("norm", "basic", "perc"))
#> BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
#> Based on 2000 bootstrap replicates
#>
#> CALL :
#> boot.ci(boot.out = ratio, type = c("norm", "basic", "perc"))
#>
#> Intervals :
#> Level      Normal              Basic              Percentile
#> 95%   (-0.2732,  0.1164 )   (-0.2998,  0.0797 )   (-0.2223,  0.1572 )
#> Calculations and Intervals on Original Scale
```

Chapter 4

Numerical Methods

4.1 Introduction

4.1.1 Computer representation of real numbers

Any positive decimal number x is represented by the ordered coefficents $\{d_j : j = n, n-1, \ldots\} \subseteq \{0, 1, \ldots, 9\}$

$$x = d_n 10^n + d_{n-1} 10^{n-1} + \cdots + d_1 10 + d_0 + d_{-1} 10^{-1} + \cdots \tag{4.1}$$

For same number x, other base 2 can also be used with binary digits $\{a_j\} \subseteq \{0, 1\}$

$$x = a_k 2^k + a_{k-1} 2^{k-1} + \cdots + a_1 2 + a_0 + a_{-1} 2^{-1} + \cdots \tag{4.2}$$

Point between a_0 and a_{-1} is called the radix point here.

```
sfsmisc::digitsBase(320, base = 10)
#> Class 'basedInt'(base = 10) [1:1]
#>      [,1]
#> [1,]    3
#> [2,]    2
#> [3,]    0
sfsmisc::digitsBase(320, base = 2)
#> Class 'basedInt'(base = 2) [1:1]
#>       [,1]
#>  [1,]    1
#>  [2,]    0
#>  [3,]    1
#>  [4,]    0
#>  [5,]    0
#>  [6,]    0
#>  [7,]    0
#>  [8,]    0
#>  [9,]    0
```

See Equations (4.1) and (4.2). Numbers are expressed with series.

Example 4.1 (Identical and nearly equal)**.** 0.3 − 0.1 is equal to 0.2. Can we check this?

```
(.3 - .1) == .2
#> [1] FALSE
```

It is obviously same, but R says it is different. Why?

```
.Machine$double.eps
#> [1] 2.22e-16
```

The above number is the smallest positive floating number that the machine can recognize. `all.equal()` function can solve this kind of near-equality problem.

```
all.equal(.2, .3 - .1)
#> [1] TRUE
```

4.2 Root-finding in One Dimension

In statistics, it is one of issues to find solutions of

$$f(x) = 0$$

There are various algorithms.

4.2.1 Bisection method

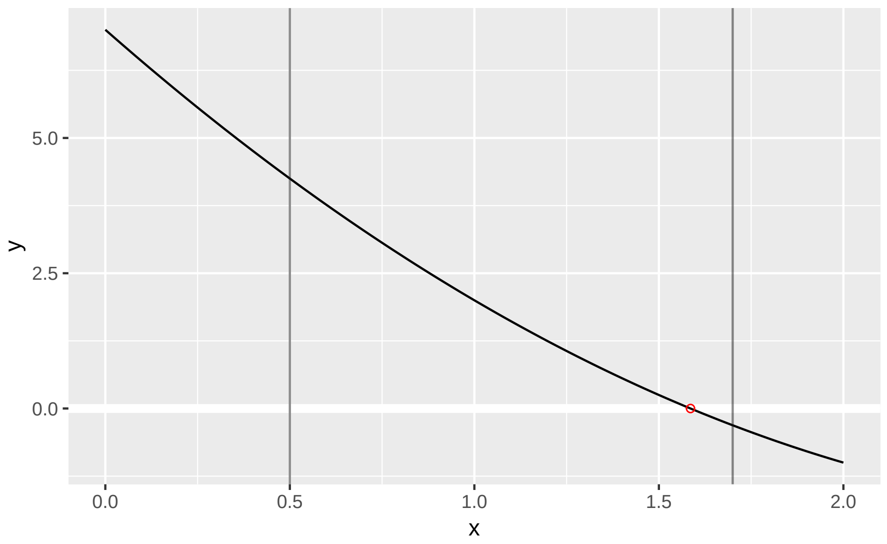


Figure 4.1: Illustration of bisection method

Figure 4.1 presents the motivation of bisection method. On both sides of the root, one side of function value is positive and the other side is negative. Thus, if we find any set like this, then we only narrow the two points until finding the solution.

Algorithm 29: Bisection algorithm

```
   input  : Equation system $f(x) = 0$, error bound $\epsilon$
 1 Initialize two points $x_0$ and $x_1$ such that
           $$f(x_0)f(x_1) \le 0$$
    ;
 2 if $f(x_0)f(x_1) < 0$ then
 3 |   Change initial values;
 4 end
 5 Set error $e = |x_1 - x_0|$;
 6 while $e > \epsilon$ do
 7 |   Half
   |       $$x_2 = \frac{x_0 + x_1}{2}$$
   |    ;
 8 |   Length of the interval becomes half $e = \frac{e}{2}$;
 9 |   if $f(x_0)f(x_2) < 0$ then
10 |   |   Update $x_1 = x_2$;
11 |   else
12 |   |   Update $x_0 = x_2$;
13 |   end
14 end
   output: $x = x_2$
```

In Line 6, we can use condition

$$|f(x_2)| > \epsilon$$

instead, which means that we did not find the root yet.

Example 4.2. Solve

$$a^2 + y^2 + \frac{2ay}{n-1} = n - 2$$

where a is a specified constant and $n > 2$ is an integer.

Solution. It can be shown that the analytical solution is

$$y = -\frac{a}{n-1} \pm \sqrt{n - 2 + a^2 + \left(\frac{a}{n-1}\right)^2}$$

```r
f_bisec <- function(x, a = .5, n = 20) {
  a^2 + x^2 + 2 * a * x / (n - 1) - (n - 2)
}
#------------------------------
bisection <- function(x0, x1, fun, eps = .Machine$double.eps^.25, rep_max = 1000, ...) {
  iter <- 0 # stop too many iteration
  if (fun(x0, ...) * fun(x1, ...) > 0) {
    stop(gettextf("both %s and %s should be satisfy the condition", expression(x0), expression(x1)))
  }
  init <- seq(x0, x1, length.out = 3) # x0 x2 x1
  y <- f_bisec(init)
  while (iter < 1000 && abs(y[2]) > eps) {
    iter <- iter + 1
    if (y[1] * y[2] < 0) {
      init[3] <- init[2]
      y[3] <- y[2]
    } else {
      init[1] <- init[2]
      y[1] <- y[2]
    }
    init[2] <- (init[1] + init[3]) / 2
    y[2] <- fun(init[2], ...)
  }
  c(init[2], y[2])
}
```

Using initioal values $x_0 = 0$ and $X_1 = 100$,

```r
(bi_exm <- bisection(0, 100, fun = f_bisec, a = .5, n = 20))
#> [1] 4.19e+00 2.98e-05
```

$x = 4.187$ has been computed. The following figure shows that this answer is reasonable.

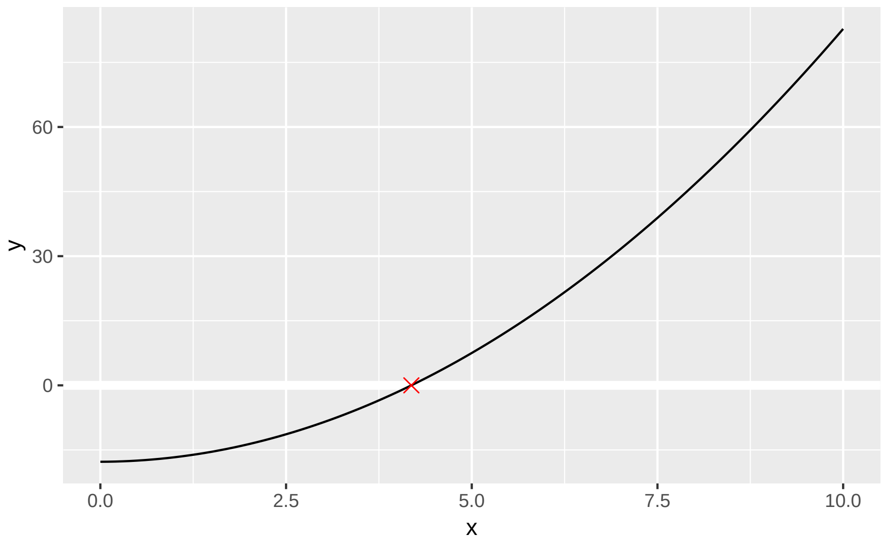


Figure 4.2: Example curve

4.2.2 Brent's method

Brent's method combines the root bracketing and bisection with inverse quadratic interpolation. `uniroot()` uses this method. Refer to Example 4.2.

```r
(brent <-
  uniroot(
    f = f_bisec,
    interval = c(0, 100),
    a = .5,
    n = 20
  ))
#> $root
#> [1] 4.19
#>
#> $f.root
#> [1] 0.000238
#>
#> $iter
#> [1] 14
#>
#> $init.it
#> [1] NA
#>
#> $estim.prec
#> [1] 6.1e-05
```

This method assures convergence of the bisection method. Morover, it is generally faster than bisection.

4.3 Numerical Integration

Try to compute

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

4.3.1 Trapezoidal rule

From definition of Riemann integration, we can compute integration I by partitioning intervals. Areas of rectangles can be considered or trapezoids can also be considered. If we use trapezoids, it will be more closed to the target curve, but the formula might be quite complex. For the length of subintervals $h = \frac{b-a}{n}$,

$$\frac{h}{2}f(a) + h\sum_{i=1}^{n-1} f(x_i) + \frac{h}{2}f(b) \tag{4.3}$$

For fun, we use `Rcpp` for trapezoid method. `Rcpp` integrate `R` and `C++`. This accelerate execution speed like loop.

```
library(Rcpp)
```

The following code should be written in `cpp` file separately, or in `cppFunction()` as character.

```
#include <Rcpp.h>
using namespace Rcpp;

// [[Rcpp::export]]
NumericVector trapezoid(Function target, double a, double b, int n) {
  double h = (b - a) / n;
  NumericVector fa = target(a);
  NumericVector fb = target(b);
  NumericVector integral = (fa + fb) / 2;
  double x = a;
  NumericVector fx = target(x);

  for(int i = 0; i < n; i++) {
    x += h;
    NumericVector fx = target(x);
    integral += fx;
  }

  integral = integral * h;

  return(integral);
}
```

Consider standard normal densitiy. Compare

$$P(-1.96 \leq Z \leq 1.96)$$

```
phi <- function(x) {
  1 / sqrt(2 * pi) * exp(- x^2 / 2)
}
#-----------------------------------
tibble(x1 = -1.96, x2 = 1.96) %>%
  summarise(
    trapezoid = trapezoid(
      phi,
      a = x1,
      b = x2,
      n = 100
    ),
    pnorm = pnorm(x2) - pnorm(x1)
  )
#> # A tibble: 1 x 2
#>   trapezoid pnorm
#>       <dbl> <dbl>
#> 1     0.952 0.950
```

4.3.2 Adaptive quadrature method

`R` provides a function `integrate()`. This implements a method called an *adaptive quadrature method*. Get

$$\int_0^\infty \frac{1}{(\cosh y - \rho r)^{n-1}} dy$$

with $\rho \in (-1, 1)$, $r \in (-1, 1)$, and $n \geq 2$ integer.

```
integrate_exm <- function(y, n, r, rho) {
  (cosh(y) - rho * r)^(1 - n)
}
```

Denote that ρ, r, and n should be pre-specified. Consider $(0.2, 0.5, 10)$.

```
integrate(
  f = integrate_exm,
  lower = 0,
  upper = Inf,
  n = 10,
  r = .5,
  rho = .2
)
#> 1.05 with absolute error < 2.3e-05
```

4.4 Maximum Likelihood Problems

Maximum likelihood estimator (MLE) is a estimator such that maximizes likelihood function. Given random sample $x_1, \ldots, x_n \overset{iid}{\sim} f(x_i; \theta)$, likelihood function can be given by

$$L(\theta) = \prod_{i=1}^{n} f(x_i)$$

Then MLE $\hat{\theta}$ is

$$\hat{\theta} = \operatorname*{argmax}_{\theta \in \Theta} L(\theta) \tag{4.4}$$

Denote that it is equivalent to maximizing log-likilihood $l(\theta) := \ln L(\theta)$.

$$\hat{\theta} = \operatorname*{argmax}_{\theta \in \Theta} l(\theta) \tag{4.5}$$

Either for L or l, we can find the critical point by differentiating in a mathematical point of view.

$$\begin{cases} \frac{d}{d\theta} l(\theta) = 0 \\ \frac{d^2}{d\theta^2} l(\theta) > 0 \end{cases}$$

Ignoring the second line, try to find root of first dervative. Finding MLE becomes *root-finding of first derivative function* problem. What we need are

1. Likelihood function or log-likelihood function
2. Its derivative

Example 4.3 (Exponential distribution)**.** Let $Y_1, Y_2 \overset{iid}{\sim} Exp(\theta)$, i.e.

$$f(y) = \theta e^{-\theta y}, \quad y > 0, \theta > 0$$

Then the likelihood function is

$$L(\theta) = \theta^2 e^{-\theta(y_1 + y_2)}$$

and log-likelihood

$$l(\theta) = 2 \ln \theta - \theta(y_1 + y_2)$$

Find its MLE $\hat{\theta}$.

Solution. Note that for $\theta > 0$,

$$\frac{d}{d\theta} l(\theta) = \frac{2}{\theta} - (y_1 + y_2)$$

Hence, we know that the analytical solution is

$$\hat{\theta} = \frac{2}{y_1 + y_2}$$

```r
y <- c(.043, .502)
```

Give input as l and $(y_1, y_2) = (0.043, 0.502)$. Here we will use `D()` which enables to output analytical derivative function for `expression`. For example,

```r
D(expression(2 * log(theta) - theta * (y1 + y2)), name = "theta")
#> 2 * (1/theta) - (y1 + y2)
```

Then we can make the following function.

```r
find_mle <- function(l, args, name, interval = c(1, 5), ...) {
  differ <- D(substitute(l), name = name)
  args[[name]] <- 0
  differ_fun <- function(x) {
    args[[name]] <- x
    eval(differ, envir = args, enclos = parent.frame())
  }
  uniroot(
    f = differ_fun,
    interval = interval,
    ...
  )$root
}
#-------------------------------------
find_mle(
  2 * log(theta) - theta * (y1 + y2),
  args = list(y1 = y[1], y2 = y[2]),
  name = "theta",
  interval = c(1, 5)
)
#> [1] 3.67
```

In `stats4` library, there is a function called `mle()`. We can also use this one.

```r
exp_logLik <- function(theta = 1) {
  - length(y) * log(theta) + theta * sum(y) # -l(theta)
}
#------------------------------------
stats4::mle(exp_logLik) %>%
  stats4::summary()
#> Maximum likelihood estimation
#>
#> Call:
#> stats4::mle(minuslogl = exp_logLik)
#>
#> Coefficients:
#>       Estimate Std. Error
#> theta     3.67       2.59
#>
#> -2 log L: -1.2
```

4.5 One-Dimensional Optimization

In the last section, our custom function finding MLE have tried to find root. On the contrary, `stats4::mle()` optimizes given negative log-likelihood, i.e. find its minimum.

Example 4.4 (Find maximum of univariate function)**.** Maximize the function

$$f(x) = \frac{\ln(1 + \ln x)}{\ln(1 + x)}$$

with respect to x.

```r
log_frac <- function(x) {
  log(1 + log(x)) / log(1 + x)
}
#---------------------------
gg_curve(
  log_frac, from = 2, to = 14,
  ylab = expression(log(1 + log(x)) / log(1 + x))
)
```

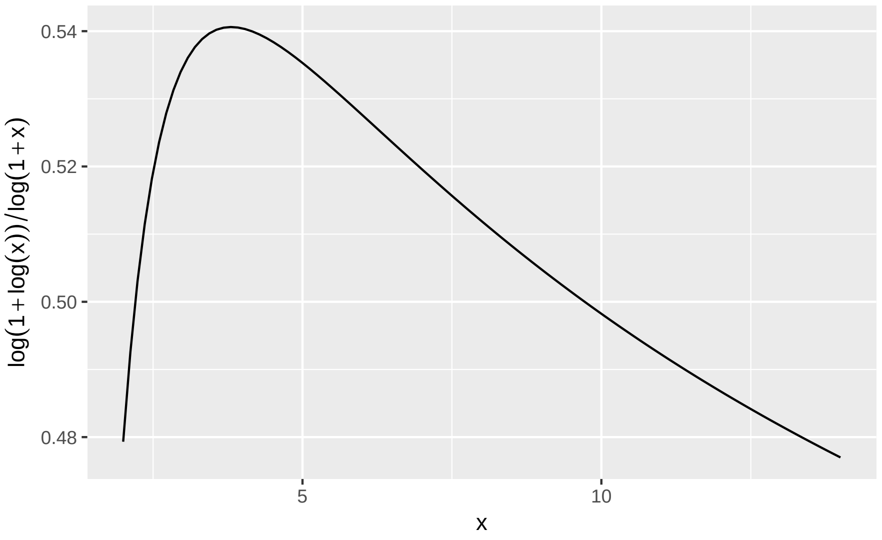


Figure 4.3: Function f in Example

`nlm()` finds minimization with a Newton-type algorithm. `optimize()` performs optimization based on various `method`. To find maximum, we should specify `maximum = TRUE`. It is set to be `FALSE` by default and find the minimum.

```r
optimize(
  log_frac,
  lower = 2,
  upper = 8,
  maximum = TRUE
)
#> $maximum
#> [1] 3.8
#>
#> $objective
#> [1] 0.541
```

`maximum` is a point where maximum is occurred and `objective` is a maximum value of the function.

4.6 Two-Dimensional Optimization

Example 4.5. Let $X_1, \ldots, X_3 \overset{indep}{\sim} f(x \mid \lambda_j) \equiv Gamma(r = \frac{1}{2}, \lambda_j)$, i.e.

$$f(x \mid \lambda_j) = \frac{\lambda^{\frac{1}{2}}}{\Gamma(\frac{1}{2})} x^{-\frac{1}{2}} e^{-\lambda x}$$

Then set a mixture $Y \sim f$ s.t. with mixing probaiblity p_1, p_2, p_3 $(p_1 + p_2 + p_3 = 1)$.

$$f(y) = \sum_{j=1}^{3} p_j f_j(y \mid \lambda_j)$$

provided that $\lambda_1 + \lambda_2 + \lambda_3 = 1$.

`optim()` can be used for multi-parameter optimization. It finds minimum of given function in `fn`. Since we are interested in maximum, we have to *add minus sign in front of the final result to get minimum.*

$$\max f = -\min(-f)$$

For simplicity, set `param = c(p1, p2, lambda1, lambda2)`. First compute log-likelihood for mixture.

```r
mix_ll <- function(param, y) {
  # mixing probability
  prob <- param[1:2]
  prob <- c(prob, 1 - sum(prob))
  # rate of dgamma
  rate <- param[3:4]
  rate <- c(rate, 1 - sum(rate)) # constraint
  dens <-
    sapply(rate, function(b) {
      dgamma(x = y, shape = 1 / 2, rate = 1 / (2 * b))
    }) %*%
    diag(prob) %>% # p_j * f_j
    rowSums()
  -sum(log(dens)) # negative log-likelihood
}
```

```r
thet_lam <- c(.6, .25, .15)
```

Let 0.6, 0.25, 0.15 be the true $(\lambda_1, \lambda_2, \lambda_3)^T$. Also, $p_1 = p_2 = p_3 = \frac{1}{3}$.

1. Generate random number from this parameter, which play a role of observed sample
2. Set an initial value for each parameter
3. Each step: Get log-likelihood value from this sample using pre-defined `mix_ll()`
4. Find minimum

Following `y` is used for `mix_ll(y = y)`, i.e. sample for likelihood.

```r
init_lam <-
  sample( # p1 = p2 = p3 = 1/3
    thet_lam, # true parameter
    size = 2000,
    replace = TRUE
  )
y <- rgamma(2000, shape = 1 / 2, rate =  1 / (2 * init_lam)) # mixture gamma with 1/3
```

In `optim()`, initial value should be supplied to `par`.

```
(opt <- optim(par = c(.3, .3, .5, .3), fn = mix_ll, y = y))
#> $par
#> [1] 0.326 0.296 0.631 0.181
#>
#> $value
#> [1] -852
#>
#> $counts
#> function gradient
#>      101       NA
#>
#> $convergence
#> [1] 0
#>
#> $message
#> NULL
```

`$value` is the minimum value of `fn`. `$par` is our interest, parameter vector where the function is minimized. `broom::tidy()` also has a method for `optim` result. This changes it to `tibble` with two columns - `parameter` (parameter name) and `value` (its value).

```
opt_df <-
  broom::tidy(opt) %>%
  spread(parameter, value)
#-------------------------
colnames(opt_df) <- c("prob1", "prob2", "lambda1", "lambda2")
opt_df %>%
  mutate(
    prob3 = 1 - prob1 - prob2,
    lambda3 = 1 - lambda1 - lambda2
  ) %>%
  select(order(colnames(.)))
#> # A tibble: 1 x 6
#>   lambda1 lambda2 lambda3 prob1 prob2 prob3
#>     <dbl>   <dbl>   <dbl> <dbl> <dbl> <dbl>
#> 1   0.631   0.181   0.188 0.326 0.296 0.378
```

Compare this MLE to the true λ_j 0.6, 0.25, 0.15.

4.7 EM Algorithm

EM algorithm is often applied to find MLE, especially when data are incomplete. This section is mainly following explanation of Bilmes (1998). Let $\mathcal{X}$ be the observed data set. Assume that this is incomplete. Then let $\mathcal{Y}$ be the missing part. Assume that the complete data set exists $\mathcal{Z} = (\mathcal{X}, \mathcal{Y})$. Then the joint density of $\mathcal{Z}$ is

$$p(\mathbf{z} \mid \boldsymbol{\theta}) = p(\mathbf{x}, \mathbf{y} \mid \boldsymbol{\theta}) = p(\mathbf{y} \mid \mathbf{x}, \boldsymbol{\theta}) p(\mathbf{x} \mid \boldsymbol{\theta}) \tag{4.6}$$

MLE problem tries to find the maximum of

$$L(\boldsymbol{\theta} \mid \mathcal{X}, \mathcal{Y}) = p(\mathcal{X}, \mathcal{Y} \mid \boldsymbol{\theta}) \tag{4.7}$$

which is called *complete-data likelihood*. For this kind of likelihood, we use EM algorithm. **E** stands for expectation and **M** for maximization. EM algorithm consists of these two step.

4.7.1 Expectation step (E-step)

At first, expectation step finds conditional expectation of complete-data likelihood (4.7) or its log given observed sample and current parameter estimates. Denote that random $\mathcal{Y}$ has not been observed. So we should remove it. Conditinal expectation does this job.

Let $\boldsymbol{\theta}^{(i-1)}$ be the current parameter estimates. EM algorithm keeps updating the parameters. At each step, we would have each updated value. What we want is conditional expectation under $\mathcal{Y} \mid \mathcal{X}, \boldsymbol{\theta}$. Then we should know its conditional density, i.e.

$$f(\mathbf{y} \mid \mathcal{X}, \boldsymbol{\theta}) \tag{4.8}$$

Then E-step calculate the following. Writing the support of $\mathcal{Y}$ by $\mathbb{Y}$,

$$Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(i-1)}) := E\left[\ln p(\mathcal{X}, \mathcal{Y} \mid \boldsymbol{\theta}) \,\middle|\, \mathcal{X}, \boldsymbol{\theta}^{(i-1)}\right] = \int_{\mathbb{Y}} \ln p(\mathcal{X}, \mathbf{y} \mid \boldsymbol{\theta}) f(\mathbf{y} \mid \mathcal{X}, \boldsymbol{\theta}^{(i-1)}) d\mathbf{y} \tag{4.9}$$

4.7.2 Maximization step (M-step)

Maximization step maximizes the conditional expectation Q (4.9) with respect to $\boldsymbol{\theta}$ given $\boldsymbol{\theta}^{(i-1)}$ (Rizzo, 2007).

$$\boldsymbol{\theta}^{(i)} = \operatorname*{argmax}_{\boldsymbol{\theta}} Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(i-1)}) \tag{4.10}$$

E-step (4.9) and M-step (4.10) are repeated. It is guaranteed that the log-likelihood is always increasing and the algorithm converges to local maximum.

4.7.3 EM algorithm for a mixture model

EM algorithm is widely used for mixture model. Recall that

$$p(\mathbf{x} \mid \boldsymbol{\beta}) = \sum_{m=1}^{M} \alpha_m p_m(\mathbf{x} \mid \theta_m)$$

with parameter vector $\boldsymbol{\beta} = (\alpha_1, \ldots, \alpha_M, \theta_1, \ldots, \theta_M)$. α_m is an mixing probability satisfying that

$$\sum_{m=1}^{M} \alpha_m = 1$$

Each θ_m parameterizes individual density p_m. Consider complete sample $\mathcal{Z} = (\mathcal{X}, \mathcal{Y}) = \{(\mathbf{x}_i, \mathbf{y}_i)\}_1^N$ with unobserved $\mathcal{Y}$. Assume that for each $i = 1, \ldots, N$,

$$\mathbf{x}_i \sim p_k(\mathbf{x} \mid \theta_k) \Rightarrow y_i = k$$

Given $\mathcal{Y}$ values, the log of complete-data likelihood is

$$\begin{aligned} l(\boldsymbol{\beta} \mid \mathcal{X}, \mathcal{Y}) &= \ln p(\mathcal{X}, \mathcal{Y} \mid \boldsymbol{\beta}) \\ &= \sum_{i=1}^{N} \ln p(\mathbf{x}_i \mid \mathbf{y}_i) p(\mathbf{y}_i) \\ &= \sum_{i=1}^{N} \ln \alpha_{\mathbf{y}_i} p_{\mathbf{y}_i}(\mathbf{x}_i \mid \theta_{\mathbf{y}_i}) \end{aligned} \tag{4.11}$$

Recall that $\mathcal{Y}$ has not been observed. So let $\mathcal{Y}$ be random. In each step we update (guess) the parameter for above likelihood (4.11). Write it as

$$\boldsymbol{\beta}^g = (\alpha_1^g, \ldots, \alpha_M^g, \theta_1^g, \ldots, \theta_M^g)$$

Given $\boldsymbol{\beta}^g$, compute $p_j(\mathbf{x}_i \mid \theta_j^g)$ for each i and j. Bayes rule implies that

$$\begin{aligned} p(\mathbf{y}_i \mid \mathbf{x}_i, \boldsymbol{\beta}^g) &= \frac{\alpha_{\mathbf{y}_i}^g p_{\mathbf{y}_i}(\mathbf{x}_i \mid \theta_{\mathbf{y}_i}^g)}{p_j(\mathbf{x}_i \mid \theta_j^g)} \\ &= \frac{\alpha_{\mathbf{y}_i}^g p_{\mathbf{y}_i}(\mathbf{x}_i \mid \theta_{\mathbf{y}_i}^g)}{\sum\limits_{m=1}^{M} \alpha_m^g p_m(\mathbf{x}_i \mid \theta_m^g)} \end{aligned} \tag{4.12}$$

Denote that $\alpha_j = P(\text{component } j)$ are kind of prior probabilities. Let $\mathbf{y} = (\mathbf{y}_1, \ldots, \mathbf{y}_N)$. Then E-step (4.9) becomes

$$\begin{aligned} Q(\boldsymbol{\beta}, \boldsymbol{\beta}^g) &= \sum_{\mathbf{y} \in \mathbb{Y}} l(\boldsymbol{\beta} \mid \mathcal{X}, \mathbf{y}) p(\mathbf{y} \mid \mathcal{X}, \boldsymbol{\beta}^g) \\ &= \sum_{\mathbf{y} \in \mathbb{Y}} \sum_{i=1}^{N} \ln \alpha_{\mathbf{y}_i} p_{\mathbf{y}_i}(\mathbf{x}_i \mid \theta_{\mathbf{y}_i}) \prod_{j=1}^{N} p(\mathbf{y}_j \mid \mathbf{x}_j, \boldsymbol{\beta}^g) \\ &= \underbrace{\sum_{y_1=1}^{M} \sum_{y_2=1}^{M} \cdots \sum_{y_N=1}^{M}}_{\mathbb{Y}} \sum_{i=1}^{N} \sum_{l=1}^{M} \delta_{l,\mathbf{y}_i} \ln \alpha_l p_l(\mathbf{x}_i \mid \theta_l) \prod_{j=1}^{N} p(\mathbf{y}_j \mid \mathbf{x}_j, \boldsymbol{\beta}^g) \\ &= \sum_{l=1}^{M} \sum_{i=1}^{N} \ln \alpha_l p_l(\mathbf{x}_i \mid \theta_l) \underbrace{\sum_{y_1=1}^{M} \sum_{y_2=1}^{M} \cdots \sum_{y_N=1}^{M} \delta_{l,\mathbf{y}_i} \prod_{j=1}^{N} p(\mathbf{y}_j \mid \mathbf{x}_j, \boldsymbol{\beta}^g)}_{(*)} \end{aligned} \tag{4.13}$$

To simplify Equation (4.13), see $(*)$ part.

$$\begin{aligned}
(*) &= \left(\sum_{y_1=1}^{M} \cdots \sum_{y_{i-1}=1}^{M} \sum_{y_{i+1}=1}^{M} \cdots \sum_{y_N=1}^{M} \prod_{j \neq i}^{N} p(\mathbf{y}_j \mid \mathbf{x}_j, \boldsymbol\beta^g) \right) \\
&= \prod_{j \neq i}^{N} \Bigg(\underbrace{\sum_{y_j=1}^{M} p(\mathbf{y}_j \mid \mathbf{x}_j, \boldsymbol\beta^g)}_{=1} \Bigg) p(l \mid \mathbf{x}_i, \boldsymbol\beta^g) \\
&= p(l \mid \mathbf{x}_i, \boldsymbol\beta^g) \quad \because \sum_{i=1}^{N} p(i \mid \mathbf{x}_j, \boldsymbol\beta^g) = 1
\end{aligned} \tag{4.14}$$

From Equations (4.13) and (4.14), we can conclude E-step.

$$\begin{aligned}
Q(\boldsymbol\beta, \boldsymbol\beta^g) &= \sum_{l=1}^{M} \sum_{i=1}^{N} \ln(\alpha_l p_l(\mathbf{x}_i \mid \theta_l)) p(l \mid \mathbf{x}_i, \boldsymbol\beta^g) \\
&= \sum_{l=1}^{M} \sum_{i=1}^{N} (\ln \alpha_l) p(l \mid \mathbf{x}_i, \boldsymbol\beta^g) + \sum_{l=1}^{M} \sum_{i=1}^{N} (\ln p_l(\mathbf{x}_i \mid \theta_l)) p(l \mid \mathbf{x}_i, \boldsymbol\beta^g)
\end{aligned} \tag{4.15}$$

Now M-step: maximize Q.

$$\begin{cases}
\alpha_l^{new} = \frac{1}{N} \sum_{i=1}^{N} p(l \mid \mathbf{x}_i, \boldsymbol\beta^g) \\
\boldsymbol\mu_l^{new} = \frac{\sum \mathbf{x}_i p(l \mid \mathbf{x}_i, \boldsymbol\beta^g)}{\sum p(l \mid \mathbf{x}_i, \boldsymbol\beta^g)} \in \mathbb{R}^p \\
\Sigma_l^{new} = \frac{\sum p(l \mid \mathbf{x}_i, \boldsymbol\beta^g)(\mathbf{x}_l - \boldsymbol\mu_l^{new})(\mathbf{x}_l - \boldsymbol\mu_l^{new})^T}{\sum p(l \mid \mathbf{x}_i, \boldsymbol\beta^g)} \in \mathbb{R}^{p \times p}
\end{cases} \tag{4.16}$$

Refer to Example 4.5. Use the same generated data.

```r
mix_em <- function(fn = dgamma, x, N = 10000, par, par_name = "rate", tol = .Machine$double.eps^.5, ...)
  ll <- list(x = x, ...)
  dens <-
    tibble(
      key = paste0("f", 1:3),
      mu = double(3L),
      lam = par
    )
  lam_old <- par + 1
  for (i in seq_len(N)) {
    dens <-
      lapply(
        dens$lam,
        function(y) {
          ll[[par_name]] <- 1 / (2 * y)
          do.call(fn, ll)
        }
      ) %>%
      bind_cols() %>%
      rename_all(
        .funs = list(
          ~str_replace_all(., pattern = "V", replacement = "f")
        )
      ) %>%
      mutate(total = apply(., 1, sum)) %>%
      gather(-total, key = "key", value = "value") %>%
      mutate(post = value / total) %>% # posterior prob y from each f1, f2, f3 - E step
      group_by(key) %>%
      summarise(mu = sum(y * post) / sum(post)) %>%  # update means - M step
      mutate(lam = mu / sum(mu))
    if (
      dens %>%
      summarise(
        sum(abs(lam - lam_old) / lam_old)
      ) %>%
      pull() < tol
    ) break()
    lam_old <- dens %>% select(lam) %>% pull()
  }
  list(lambda = dens %>% select(lam), iter = i)
}
```

The result for same initial λ is:

```
mix_em(x = y, par = c(.5, .4, .1), shape = 1 / 2)
#> $lambda
#> # A tibble: 3 x 1
#>     lam
#>   <dbl>
#> 1 0.631
#> 2 0.185
#> 3 0.185
#>
#> $iter
#> [1] 878
```

Chapter 5

Markov Chain Monte Carlo Methods

Previously, we keep trying to compute

$$Eh(X)$$

by generating random numbers. It is based on the law of large numbers that says

$$Eh(X) \approx \sum_{i=1}^{N} \frac{h(X_i)}{N}$$

The question is, when this convergence happens. Some random numbers might require expremely large N, while others needs affordable size. It is known that if this $\{X_1, \ldots, X_N\}$ is *generated from Markov chain, the series converges quite fast.*

5.1 Limiting Distribution of Markov Chain

Definition 1.8 presents the definition of markov chain and *markov property.*

$$P(X_{n+1} = j \mid X_n = i, X_{n-1} = i_{n-1}, \ldots, X_0 = i_0) = P(X_{n+1} = j \mid X_n = i) = P_{ij}$$

Consider discrete state space S.

Definition 5.1 (Transition kernel)**.** One-step transition matrix for discrete time markov chain on S is

$$P = \left[P_{ij}\right]$$

n-stem transition matrix is written as

$$P^{(n)} = \left[P_{ij}^{(n)} = P(X_{n+k} = j \mid X_k = j)\right]$$

Theorem 5.1 (Chapmen-Kolmogorov Equation)**.** *For every $n, m \in \mathbb{Z}$,*

$$P^{(n+m)} = P^{(n)} P^{(m)}$$

Corollary 5.1. *By the Chapmen-Kolmogorov equation,*

$$\forall n \in \{0, 1, 2, \ldots\} : \; P^{(n)} = P^n$$

Does Markov chain converge to same state after time has passed much enough?

$$P(\text{starts at } i \text{ and ends at } j \text{ state}) = \lim_{n\to\infty} P(X_n = j \mid X_0 = i) = \lim_{n\to\infty} P_{ij}^n$$

This holds when the process satisfies some conditions.

5.1.1 Ergodic theorem

Let S be the state of MC.

Definition 5.2 (Aperiodicity)**.** Let $i \in S$ be a state.

- Period $d(i) := \gcd\{n : P_{ii}^(n) > 0, n \in \mathbb{N}\}$
- A state i is said to be ***periodic***
$$:\Leftrightarrow d(i) > 1$$
- A state i is said to be ***aperiodic***
$$:\Leftrightarrow d(i) = 1$$

It is obvious that if a chain has a period, it won't be convergent.

Definition 5.3 (Irreducibility)**.** Markov chain is ***irreducible*** iff it is possible to go from any state to any other state. Otherwise, it is called ***reducible***.

Intuitively, the states must be a *single closed communicating* class for convergence.

Definition 5.4 (Positive recurrence)**.** Markov chain is ***recurrent*** iff

$$\forall i \in S : \text{ chain starts at } i \text{ and it will eventually return to } i \text{ with probabbility } 1$$

When these properties - aperiodicity, irreducibility, and positive recurrent - MC can be guaranteed to be convergent provided finite moment.

Theorem 5.2 (Ergodic theorem)**.** *Suppose that $\{X_i\} \sim MC$ is aperiodic, irreducible and positive recurrent with $E|h(X_j)| < \infty$. Then*

$$\frac{1}{N}\sum_{i=1}^{N} h(X_i) \overset{a.s}{\to} \int_\Omega h(X_i)\pi(X_i)dP$$

as $N \to \infty$

This ergodic theorem 5.2 is an MC analog to the strong law of large numbers.

5.1.2 Stationary limiting distribution

Using transition kernel, we might get the limiting distribution. For example,

$$\begin{aligned}\boldsymbol{\pi}^{(1)} &= \boldsymbol{\pi}^{(0)} P \\ &= \begin{bmatrix}\pi_1 & \pi_2 & \pi_3\end{bmatrix}\begin{bmatrix}\pi_{11} & \pi_{12} & \pi_{13} \\ \pi_{21} & \pi_{22} & \pi_{23} \\ \pi_{31} & \pi_{32} & \pi_{33}\end{bmatrix}\end{aligned} \tag{5.1}$$

Recursively,

$$\begin{aligned}\boldsymbol{\pi}^{(t)} &= \boldsymbol{\pi}^{(t-1)} P \\ &= \boldsymbol{\pi}^{(0)} P^t\end{aligned} \tag{5.2}$$

Theorem 5.3 (Stationary probabilities)**.** *Suppose that $\{X_i\} \sim MC$ is aperiodic, irreducible and positive recurrent with $E|h(X_j)| < \infty$. Then there exists an invariant distribution $\boldsymbol{\pi}$ uniquely s.t.*

$$\begin{cases}\boldsymbol{\pi} = \boldsymbol{\pi} P \\ \boldsymbol{\pi}\mathbf{1}^T = 1\end{cases}$$

Denote that every vector is a row vector here.

5.1.3 Burn-in period

This kind of convergence is usually gauranted for any starting distribution, but the time varies according to its starting point. Thus, we *throw out a certain number of the first draws* so that stationarity less dependends on the starting point. It is called burn-in period.

5.1.4 Thinning

Denote that MC has a dependency structure. So we jump the chain, i.e. break the dependence. However, this process is unnecessary with Ergodic theorem and increases the variance of MC estiamtes.

5.2 Metropolis-Hastings Algorithm

Markov Chain Monte Carlo (MCMC) Methods includes in metropolis-hastings algorithm and gibbs sampler. In fact, gibbs sampler is a special form of the former. Here we follow the notation of Chib and Greenberg (1995).

Definition 5.5 (Density)**.** In Metropolis-hastings (M-H) algorithm, we take care about the following two densities. Denote that terms and process are similar to A-R process.

1. Target density $\pi(\cdot)$ density that we try to generate sample from

2. Candidate-generating density $q(\cdot \mid \cdot)$ density that we will actually generate random sample from

5.2.1 Metropolis-hastings sampler

Algorithm 30: Metropolis-Hastings algorithm with burn-in period

```
   input  : Starting point $x_0$, burn-in period $b$
1  for $i \leftarrow 1$ to $N$ do
2      Draw a candidate distribution $Y \sim q(\cdot \mid x^{(j)})$;
3      $U \sim unif(0,1) \perp\!\!\!\perp Y$;
4      Acceptance rate
           $$\alpha(x^{(j)}, y) := \min\left(\frac{\pi(y)q(x^{(j)} \mid y)}{\pi(x^{(j)}), q(y \mid x^{(j)})}, 1\right)$$
       ;
5      if $U \le \alpha(x^{(j)}, y)$ then
6          Accept so that $x^{(j+1)} = y$;
7      else
8          Reject so that $x^{(j+1)} = x^{(j)}$;
9      end
10 end
11 Draw out the first $b$ $x^{(j)}$ (Burn-in);
   output: $x^{(b+1)}, \ldots, x^{(N)}$
```

Example 5.1 (Rayleigh density)**.** Generate a sample from a Rayleigh density

$$f(x) = \frac{x}{\sigma^2} e^{-\frac{x^2}{2\sigma^2}}$$

```r
dray <- function(x, sd) {
  if (sd <= 0 ) stop(gettextf("%s should be positive", expression(sd)))
  ifelse(
    x >= 0,
    x / sd^2 * exp(- x^2 / (2 * sd^2)),
    0
  )
}
```

Consider $\chi^2(x^{(j)})$ as candidate. The following function calcuates acceptance rate.

```r
acc_mc <- function(x, y, sd = 4) {
  ( (dray(y, sd) * dchisq(x, df = y)) / (dray(x, sd) * dchisq(y, df = x)) ) %>%
    min(1)
}
```

To enhance the speed, we register parallel backends.

```r
MC_CORES <- future::availableCores() - 1
cl <- parallel::makeCluster(MC_CORES)
doParallel::registerDoParallel(cl, cores = MC_CORES)
parallel::clusterExport(cl, c("acc_mc", "dray"))
parallel::clusterEvalQ(cl, c(library(dplyr), library(data.table)))
#> [[1]]
#>  [1] "dplyr"      "stats"      "graphics"   "grDevices"  "utils"
#>  [6] "datasets"   "methods"    "base"       "data.table" "dplyr"
#> [11] "stats"      "graphics"   "grDevices"  "utils"      "datasets"
#> [16] "methods"    "base"
#>
#> [[2]]
#>  [1] "dplyr"      "stats"      "graphics"   "grDevices"  "utils"
#>  [6] "datasets"   "methods"    "base"       "data.table" "dplyr"
#> [11] "stats"      "graphics"   "grDevices"  "utils"      "datasets"
#> [16] "methods"    "base"
#>
#> [[3]]
#>  [1] "dplyr"      "stats"      "graphics"   "grDevices"  "utils"
#>  [6] "datasets"   "methods"    "base"       "data.table" "dplyr"
#> [11] "stats"      "graphics"   "grDevices"  "utils"      "datasets"
#> [16] "methods"    "base"
```

```
mc_ray <- function(N = 10000, x0, sd = 4, burn = 1000) {
  x <- x0
  y <- numeric(1L)
  acc <- numeric(1L)
  foreach(i = seq_len(N), .combine = rbind, .inorder = TRUE) %dopar% {
    y <- rchisq(1, df = x)
    acc <- runif(1) <= acc_mc(x, y, sd)
    x <- ifelse(acc, y, x)
    data.table(
      draw = i,
      acc = acc,
      x = x
    )
  } %>%
    .[(burn + 1):(.N)]
}
```

For a better result, try *burn-in period* 2000.

```
ray <- mc_ray(N = 10000, x0 = 1, sd = 4, burn = 2000)
#----------------------------------------------------
parallel::stopCluster(cl)
```

Among 8000 chain, 4740 candidiate points are rejected.

```
ray[,
    .N,
    by = acc]
#>      acc    N
#> 1: FALSE 3260
#> 2:  TRUE 4740
```

Recall that A-R method have tried to elevate the acceptance rate for efficiency.

```
ray %>%
  ggplot(aes(x = draw, y = x)) +
  geom_path(aes(colour = acc, group = 1)) +
  labs(
    x = "Draw",
    colour = "Acceptance"
  ) +
  theme(legend.position = "bottom")
```

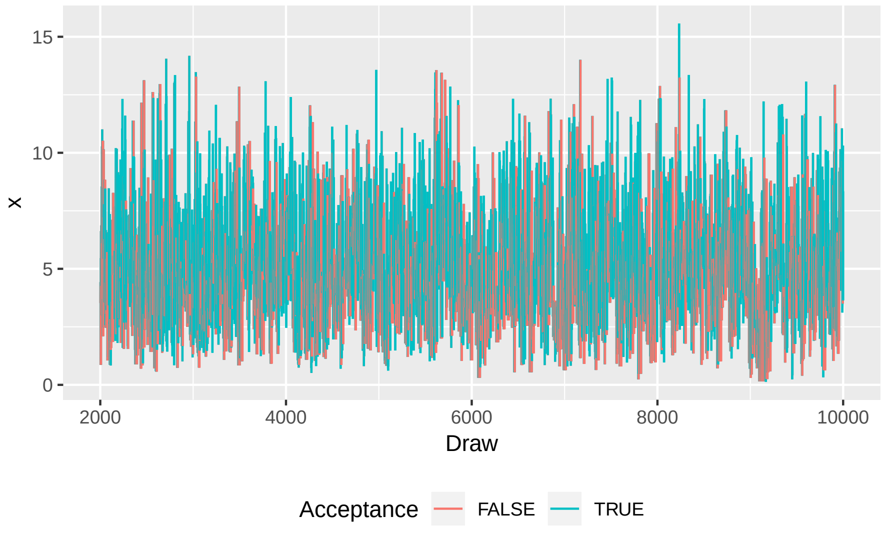


Figure 5.1: M-H sampling from Chisq to target Rayleigh

In Figure 5.1, the short horizontal paths might be represented as rejection points.

```r
ray[3000:3500] %>%
  ggplot(aes(x = draw, y = x)) +
  geom_path(aes(colour = acc, group = 1)) +
  labs(
    x = "Draw",
    colour = "Acceptance"
  ) +
  theme(legend.position = "bottom")
```

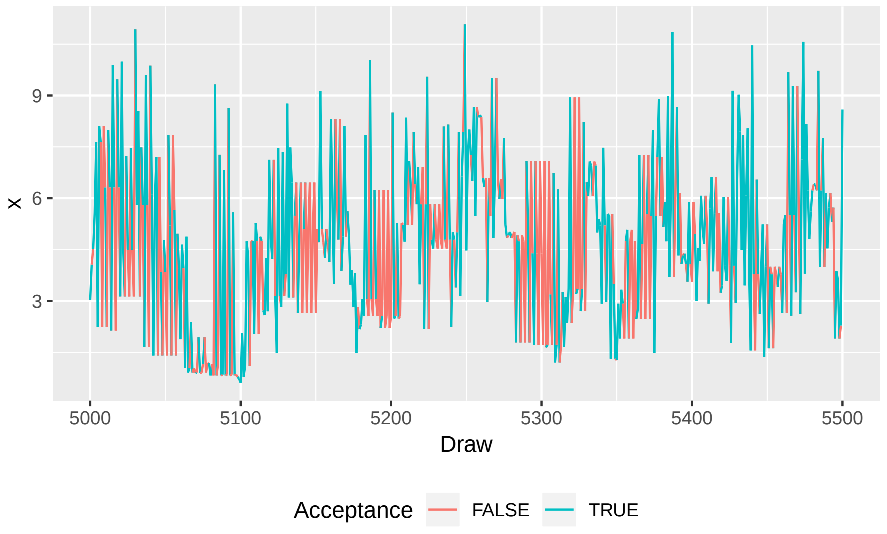


Figure 5.2: Part of a chain from M-H sampling

5.2.2 Mixing

Looking at Figure 5.1, it seems that the chain converges well.

```r
ray %>%
  ggplot(aes(x = draw, y = x)) +
  geom_jitter(aes(colour = x, alpha = abs(x)), show.legend = FALSE) +
  scale_colour_gradient(low = "#0091ff", high = "#f0650e") +
  xlab("Draw")
```

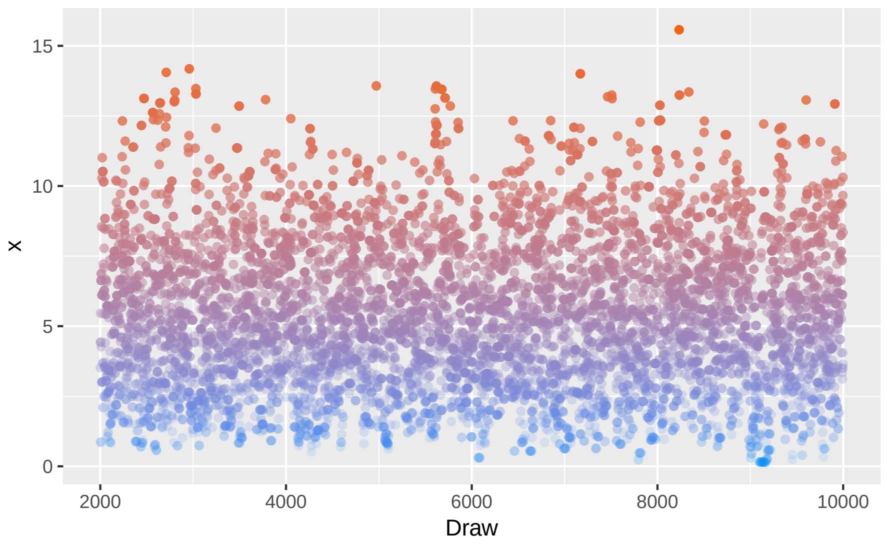


Figure 5.3: Metropolis-Hastings sampling mixing

See Figure 5.3. We can see that the random numbers are mixed well.

5.2.3 Random walk metropolis

5.2.4 Independence sampler

5.3 Gibbs Sampler

5.3.1 Concept of gibbs sampler

We are given the joint density. For this joint density, the following theorem can be proven.

Theorem 5.4 (Hammersley-Clifford Theorem)**.** *Suppose that* $(X, Y)^T \sim f(x, y)$*. Then*

$$f(x, y) = \frac{f(y \mid x)}{\int_{\mathbb{R}} \frac{f(y|x)}{f(x|y)} dy}$$

By definition, $f(x, y) \propto f(y \mid x)$. However, the above theorem gives that this joint density is proportional to both conditional densities, i.e. also to $f(x \mid y)$.

Corollary 5.2. *Theorem 5.4 implies the second*

- $f(x, y) \propto f(y \mid x)$
- $f(x, y) \propto f(x \mid y)$

This can be extended to cases more than two blocks.

Definition 5.6 (Full conditional distribution)**.** Let $\mathbf{X} = (X_1, \ldots, X_p)^T \in \mathbb{R}^p$ be a p-dimensional random vector. Then the ***full conditional distribution*** of X_j is

$$f(X_j \mid \mathbf{X}_{(-j)})$$

where $\mathbf{X}_{(-j)} = (X_1, \ldots, X_{j-1}, X_{j+1}, \ldots, X_p)^T$.

Gibbs sampler iterate to generate a number from each full conditional distribution so that we finally get the joint density, i.e.

$$X_j \sim f(X_j \mid \mathbf{X}_{(-j)})$$

For instance, for $p = 3$,

$$\begin{cases} X^{(1)} \sim f(x \mid y^{(0)}, z^{(0)}) \\ Y^{(1)} \sim f(y \mid x^{(1)}, z^{(0)}) \\ Z^{(1)} \sim f(z \mid x^{(1)}, y^{(1)}) \end{cases}$$

and so $(X^{(1)}, Y^{(1)}, Z^{(1)})^T \sim f(x, y, z)$. Next,

$$\begin{cases} X^{(2)} \sim f(x \mid y^{(1)}, z^{(1)}) \\ Y^{(2)} \sim f(y \mid x^{(2)}, z^{(1)}) \\ Z^{(2)} \sim f(z \mid x^{(2)}, y^{(2)}) \end{cases}$$

so that $(X^{(2)}, Y^{(2)}, Z^{(2)})^T \sim f(x, y, z)$, and so on.

5.3.2 Full conditional distributions

Suppose that we only have

Here, of course, we should know $f(\cdot \mid \cdot)$. In some cases, the closed form can be given. Otherwise, there are some calculation methods.

1. normalized posterior
2. drop the irrelevant terms
3. closed form
4. Repeat 2 and 3 for all parameter blocks

Example 5.2 (Bivariate normal distribution)**.** Generate

$$(X_1, X_2) \mid \mu_1, \mu_2, \sigma_1^2, \sigma_2^2, \rho \sim N_2\bigg((\mu_1, \mu_2)^T, \begin{bmatrix} \sigma_1^2 & \rho \\ \rho & \sigma_2^2 \end{bmatrix}\bigg)$$

In this problem, its closed can easily calculated.

$$\begin{cases} X_1 \mid X_2, \mu_1, \mu_2, \sigma_1^2, \sigma_2^2, \rho \sim N\Big(\mu_1 + \rho\frac{\sigma_1}{\sigma_2}(X_2 - \mu_2), (1-\rho^2)\sigma_1^2\Big) \\ X_2 \mid X_1, \mu_1, \mu_2, \sigma_1^2, \sigma_2^2, \rho \sim N\Big(\mu_2 + \rho\frac{\sigma_2}{\sigma_1}(X_1 - \mu_1), (1-\rho^2)\sigma_2^2\Big) \end{cases}$$

Hence, we just iterate the above set of process until gaining N draws.

5.3.3 Gibbs sampler step

Algorithm 31: Gibbs-sampler steps

```
   Data: Full conditional distribution $f$
   input  : Starting values $(x_1^{(0)}, x_2^{(0)})$, burn-in period $b$
1  for $i \leftarrow 1$ to $N$ do
2      Set $x_2^* = x_2^{(i-1)}$;
3      for $j \leftarrow 1$ to 2 do
4          Generate $x_j^{(i)} \sim f(x_j \mid x_{(-j)} = x_{(-j)}^*)$;
5          Set or update $x_j^* = x_j^{(i)}$;
6      end
7  end
8  Draw out the first $b$ $x^{(j)}$ (Burn-in);
   output: $x^{(b+1)}, \ldots, x^{(N)}$
```

Sometimes Gibbs sampler algorithm 31 requires nested loop, whose efficiency becomes quite awful. In `R`, `C++` implementation can be a solution (Wickham, 2019). The following code is executed in `Rcpp` environment in `rmd` document. In practice, this should be placed in `cpp` file. Or `cppFunction()` can also be used.

```cpp
#include <Rcpp.h>
using namespace Rcpp;

// [[Rcpp::export]]
NumericMatrix gibbs_bvn(int N, double x, double y, int burn,
                        double mu1, double mu2, double sig1, double sig2, double rho) {
  NumericMatrix mat(N - burn, 2);

  for(int i = 0; i < burn; i++) {
    x = rnorm(1, mu1 + rho * sig1 / sig2 * (y - mu2), (1 - pow(rho, 2)) * pow(sig1, 2))[0];
    y = rnorm(1, mu2 + rho * sig2 / sig1 * (x - mu1), (1 - pow(rho, 2)) * pow(sig2, 2))[0];
  }

  for(int i = burn; i < N; i++) {
    x = rnorm(1, mu1 + rho * sig1 / sig2 * (y - mu2), (1 - pow(rho, 2)) * pow(sig1, 2))[0];
    y = rnorm(1, mu2 + rho * sig2 / sig1 * (x - mu1), (1 - pow(rho, 2)) * pow(sig2, 2))[0];
    mat(i - burn, 0) = x;
    mat(i - burn, 1) = y;
  }

  return(mat);
}
```

By executing above code, `gibbs_bvn(N, x, y, burn, mu1, mu2, sig1, sig2, rho)` function is created.

```r
bvn <-
  gibbs_bvn(5000, 0, 0, 1000, 0, 2, 1, .5, -.75) %>%
  data.table()
setnames(bvn, c("x", "y"))
```

We have generated bivariate normal random numbers. See Figure 5.4. Compare with our $\boldsymbol{\mu}$ and Σ.

```r
gg_scatter(bvn, aes(x, y))
```

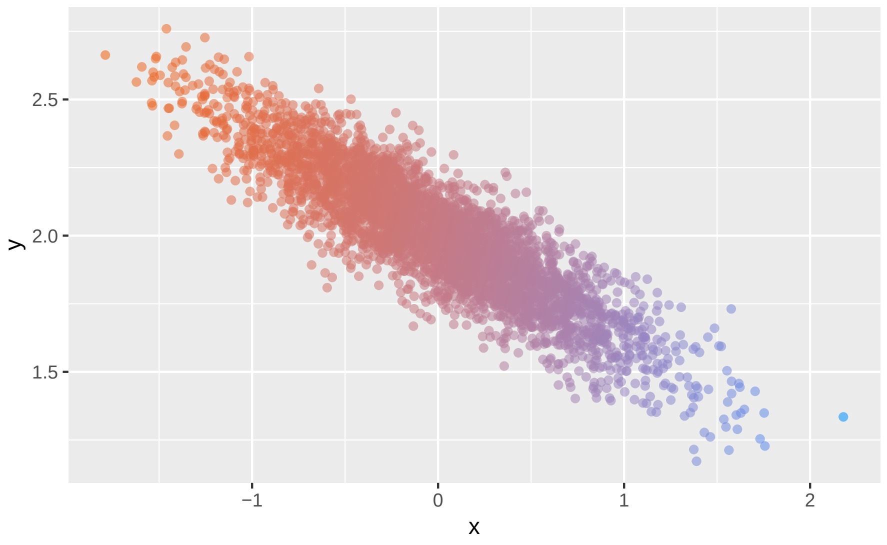


Figure 5.4: Bivariate normal chain by the gibbs sampler

5.4 Monitoring Convergence

Bibliography

Bilmes, J. A. (1998). A gentle tutorial of the EM algorithm and its application to parameter estimation for Gaussian mixture and hidden Markov models. *International Computer Science Institute*, (4):126.

Chib, S. and Greenberg, E. (1995). Understanding the metropolis-hastings algorithm. *The American Statistician*, 49(4):327–335.

Efron, B. and Gong, G. (1983). A Leisurely Look at the Bootstrap, the Jackknife, and Cross-Validation. *The American Statistician*, 37(1):36–48.

Efron, B. and Tibshirani, R. (1994). *An Introduction to the Bootstrap*. CRC Press.

McGrath, R. N. and Yeh, A. B. (2005). A Quick, Compact, Two-Sample Dispersion Test. *The American Statistician*, 59(1):47–53.

Rizzo, M. L. (2007). *Statistical Computing with R*. Chapman and Hall/CRC.

Wickham, H. (2019). *Advanced R, Second Edition*. CRC Press.