蒙特卡洛仿真

黄嘉平

2025-03-02

1 准备工作

为了给今后的学习做准备，也为了熟悉如何在 R 中安装和调用程序包（package），我们首先安装 `tidyverse` 包。

1. 第一步要选择合适的镜像服务器（mirror server）[1]。在 RStudio 的顶部菜单中选择 Tools > Global Options... ，然后在新出现的窗口左侧找到 Packages 项，在 Primary CRAN Repository 一栏点击 Change... ，并从列表中选择位于国内的任意选项（例如 China (Shenzhen) [https] - Southern University of Science and Technology (SUSTech)），点击 OK，最后点击 Apply。
2. 从顶部菜单中选择 Tools > Install Packages... ，输入 tidyverse，并勾选下面的 Install dependencies，最后点击 Install。
3. 等待安装程序结束后，在 Console 中输入下面的命令，如果运行结果和下面展示的一致（版本号有可能发生变化），就是成功调用了 tidyverse。

```
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.1     v tibble    3.2.1
## v lubridate 1.9.4     v tidyr     1.3.1
## v purrr     1.0.4
## -- Conflicts ---------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to beco
```

如需安装其他程序包，可参照第 2 条操作。今后我们默认调用 tidyverse，也就是说在每次启动 RStudio 时，都要首先运行 `library(tidyverse)`。

[1] R 的官方程序包发布网站称为 CRAN，在世界各地都有同步的服务器，称为镜像。选择地理位置接近的镜像可以避免因连接超时造成的安装失败。

2 伪随机数

蒙特卡洛仿真（Monte Carlo simulation）指利用计算机生成的伪随机数（pseudo-random number）进行数据生成的方法。首先需要明确的是，真正的随机现象是无法准确预测的。因此，利用计算机得出的任何数值都不是随机的。但是我们可以生成很难预测的近似随机的数值，称为伪随机数。

最早的伪随机数生成法是线性同余法（linear congruential generator），它是根据下面的递归方程

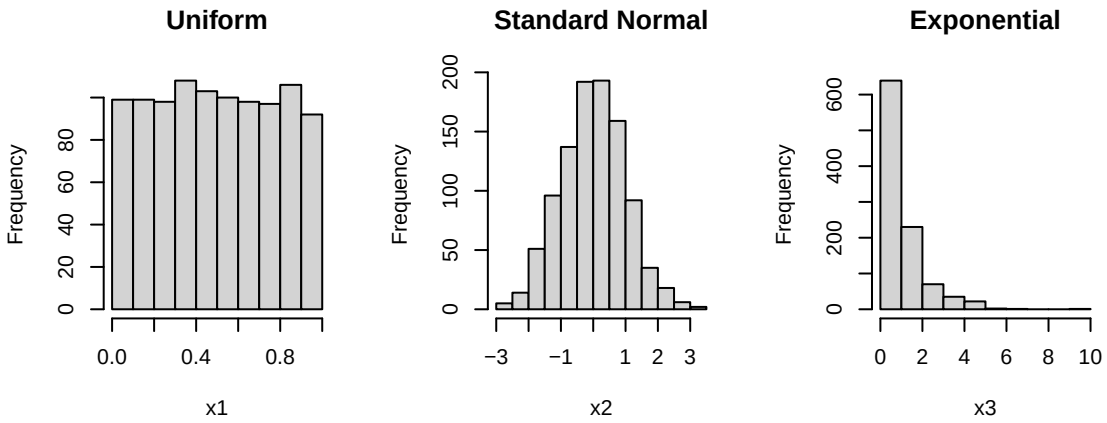
$$x_{n+1} = (ax_n + c) \mod m,$$

生成数列 $\{x_n\}$，其中 $0 < a < m, 0 < c < m, 0 < x_0 < m$。当 (a, c, m) 满足一定条件时，$\{x_n\}$ 的周期为 m 且不受 x_0 取值的影响。通常需要取很大的 m 以增加预测的难度，例如 Microsoft Visual C++ 中 $(a, c, m) = (214013, 2531011, 2^{31})$。

如今的伪随机数生成器原理更为复杂，各种现代编程语言大多采用 Mersenne-Twister 法，它生成的数列周期为 $2^{19937} - 1$，且计算速度快。如果想了解更多信息可以参考 Bilibili 上的视频 https://www.bilibili.com/video/BV1XW411s7mK/。

R 中生成伪随机数的函数是 `r???()`，其中 `???` 可以替换为具体的分布名称。例如，均匀分布对应 `runif()`，正态分布对应 `rnorm()`，t 分布对应 `rt()` 等。下面的程序中依次生成了服从均匀分布、正态分布和指数分布的伪随机数，并画出各自的直方图。

```
set.seed(123)
x1 <- runif(1000)
x2 <- rnorm(1000)
x3 <- rexp(1000, rate = 1)
par(mfrow = c(1, 3))
hist(x1, main = "Uniform")
hist(x2, main = "Standard Normal")
hist(x3, main = "Exponential")
```

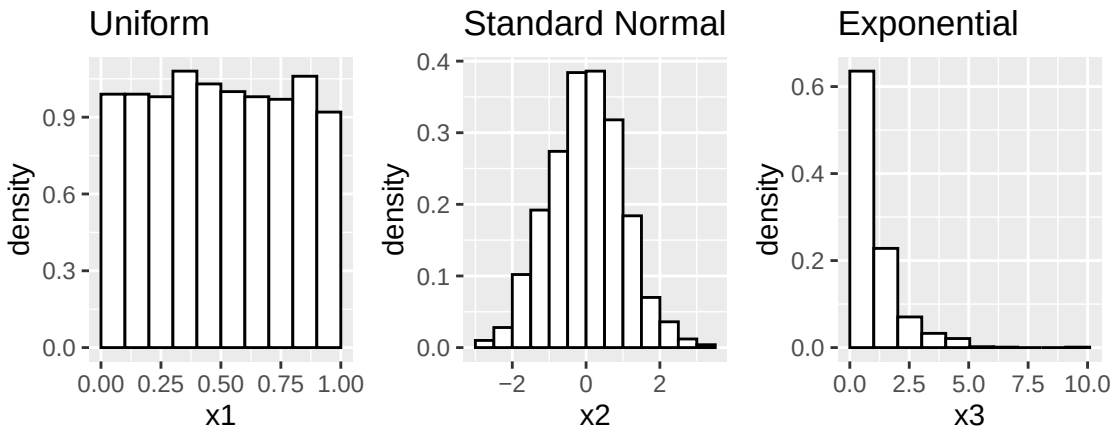


第一行的 `set.seed()` 函数指定了生成伪随机数所需的种子值。如果不指定种子值，则每次运行的结果都不相同，虽然能够体验随机效果，但无法进行重复验证。换句话说，如果知道伪随机数生成机制以及种子值，就可以完美预测所生成的数值。种子值可以是任意整数。

以上是利用 base R 进行数据生成和绘图的例子。下面我们用 tidyverse 中的 tibble 数据结构和 ggplot2 程序包提供的绘图功能进行相同的操作。Tibble 是对 base R 中的 data.frame 数据结构的

改良，是 tidyverse 中各程序包的通用数据保存形式。ggplot2 则提供了更加丰富美观的绘图功能。我们还用到了 patchwork 程序包（需安装后调用），它提供了将多个图表合并为一个的功能。

```
library(patchwork)
set.seed(123)
rn <- tibble(x1 = runif(1000), x2 = rnorm(1000), x3 = rexp(1000, rate = 1))
p1 <- ggplot(rn, aes(x1, after_stat(density))) +
  geom_histogram(bins = 10, breaks = seq(0, 1, 0.1), color = "black", fill = "white") +
  labs(title = "Uniform")
p2 <- ggplot(rn, aes(x2, after_stat(density))) +
  geom_histogram(binwidth = 0.5, center = 0.25, color = "black", fill = "white") +
  labs(title = "Standard Normal")
p3 <- ggplot(rn, aes(x3, after_stat(density))) +
  geom_histogram(bins = 10, boundary = 0, color = "black", fill = "white") +
  labs(title = "Exponential")
p1 + p2 + p3
```



3 验证大数定律

弱大数定律：如果 X_i 为 i.i.d. 且 $E[X_i] < \infty$，则当 $n \to \infty$ 时，

$$\overline{X}_n = \frac{1}{n}\sum_{i=1}^{n} X_i \xrightarrow{p} E[X_i]$$

$\xrightarrow{p}$ 意为依概率收敛，代表随着样本量的增大随机变量的分布将越来越窄。下面我们通过仿真验证大数定律。

步骤：

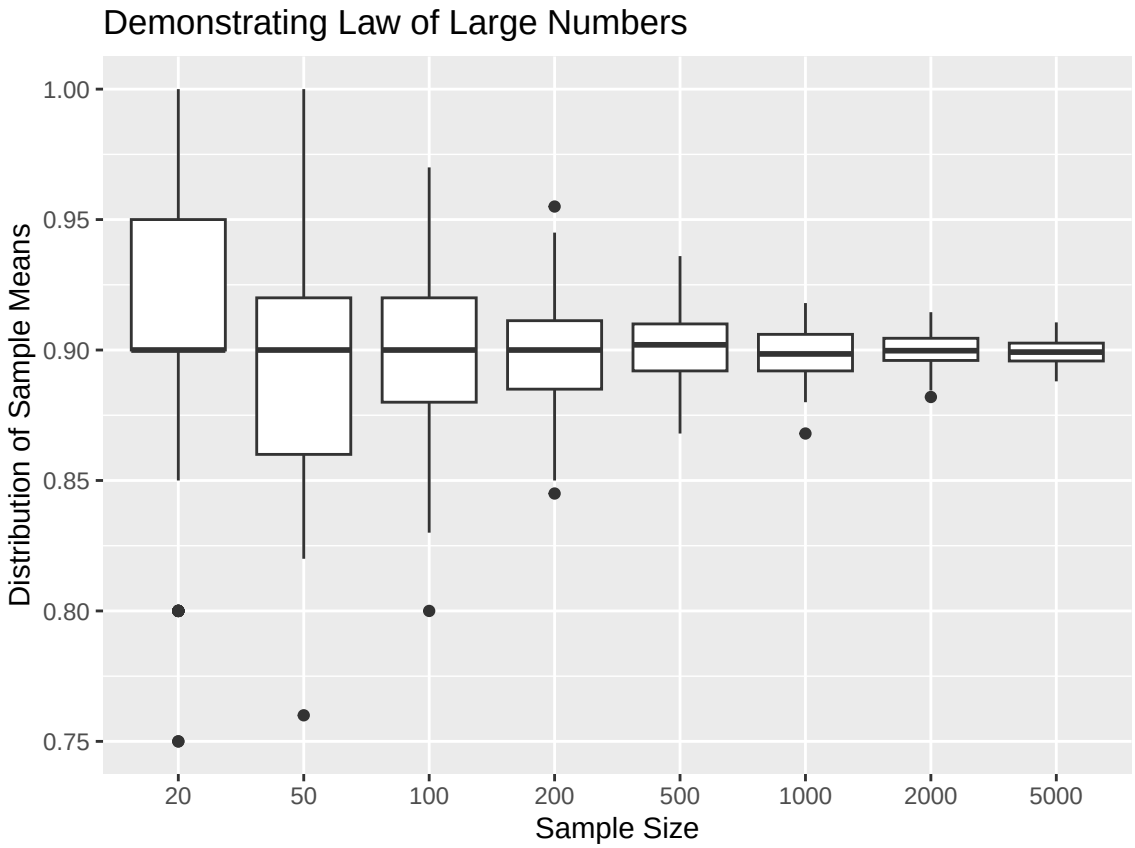
1. 确定样本量 n 的增加序列，例如 20, 50, 100, 200, 500, 1000, 2000, 5000。
2. 确定用于计算样本均值分布的随机样本数 k，例如 100。
3. 选择 X_i 的分布函数，这里我们用 $p = 0.9$ 的 Bernoulli 分布。针对每一个随机样本计算样本均值 $\overline{X}_n$ 并保存结果。

4. 将样本均值的分布可视化，并观察它们和 n 之间的关系。每一个 n 对应 k 组样本，因此可以用计算出的 k 个样本均值画出箱线图。

```r
set.seed(999)
n_values <- c(20, 50, 100, 200, 500, 1000, 2000, 5000)
k_value <- 100 # 固定 n 时的随机样本数
mean_values <- tibble(
  n = rep(n_values, each = k_value),
  k = rep(1:k_value, length(n_values))
) # 这一步是准备好保存样本均值的 tibble 数据

sample_means <- c() # 准备一个空向量
for (i in 1:length(n_values)) {
  for (j in 1:k_value) {
    sample_means[(i - 1) * k_value + j] <-
      mean(rbinom(n_values[i], size = 1, p = 0.9))
    # 将计算出的样本均值依次保存
  }
}
mean_values <- add_column(mean_values, sample_means)
mean_values <- mutate(mean_values, n = as.character(n))

mean_values |>
  ggplot(aes(reorder(n, as.numeric(n)), sample_means)) +
  geom_boxplot() +
  labs(
    title = "Demonstrating Law of Large Numbers",
    x = "Sample Size",
    y = "Distribution of Sample Means"
  )
```



4 验证中心极限定理

Lindeberg-Lévy 中心极限定理：如果 X_i 为 i.i.d. 且 $E[X_i^2] < \infty$，则当 $n \to \infty$ 时，

$$\frac{\sqrt{n}}{\sigma}(\overline{X}_n - \mu) \xrightarrow{d} Z \sim N(0,1)$$

此处 $\mu = E[X_i], \sigma^2 = E[(X_i - \mu)^2]$, $N(a, b^2)$ 是均值为 a 方差为 b^2 的正态分布。$\xrightarrow{d}$ 意为依分布收敛，代表随着样本量的增大随机变量的分布将越来越接近极限分布。中心极限定理也可以表达为 $\overline{X}_n \xrightarrow{d} N(\mu, \sigma^2/n)$。

步骤：

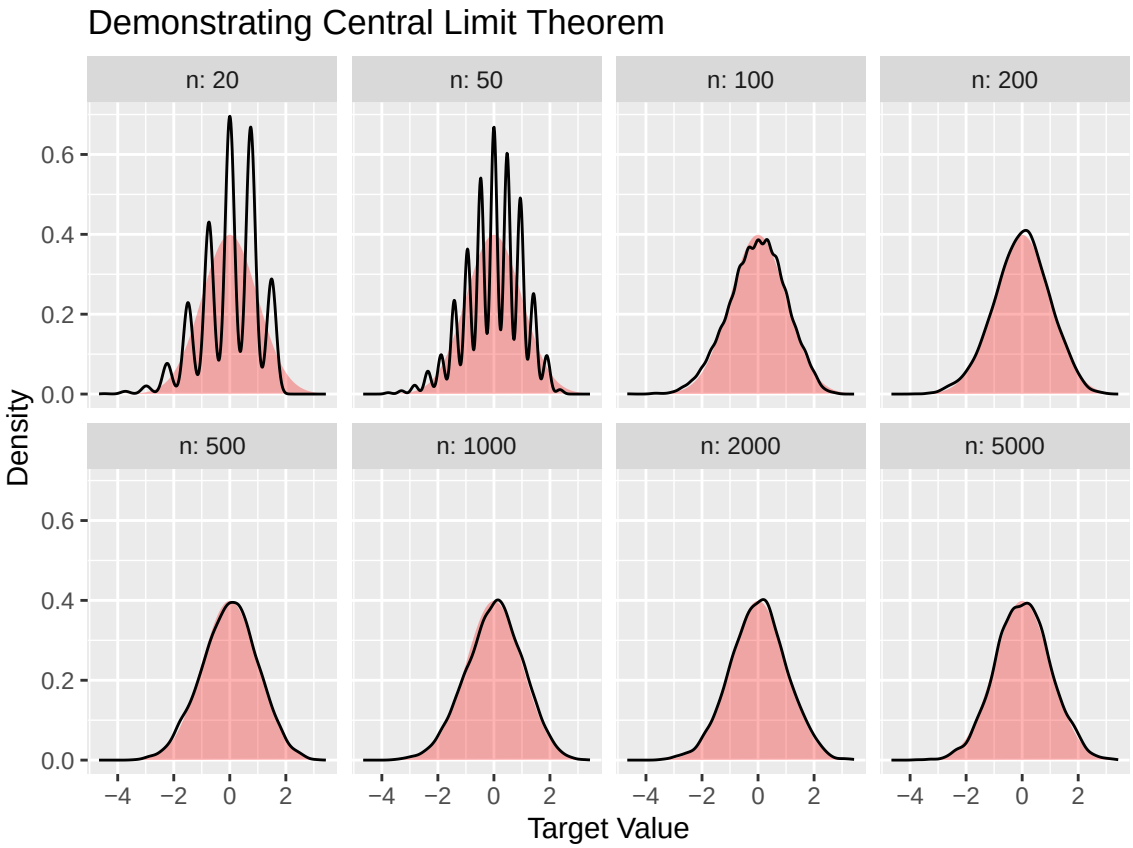
1. 确定样本量 n 的增加序列，例如 20, 50, 100, 200, 500, 1000, 2000, 5000。
2. 确定用于计算样本均值分布的随机样本数 k，为了更好地描绘分布，我们选择 $k = 5000$。
3. 选择 X_i 的分布函数，这里我们依旧用 $p = 0.9$ 的 Bernoulli 分布。针对每一个随机样本计算 $\sqrt{n}(\overline{X}_n - \mu)/\sigma$ 并保存结果。
4. 将样本均值的分布可视化，并对照标准正态分布。这里用样本密度函数的拟合图像与理论标准正态密度函数进行比较。

```
set.seed(999)
n_values <- c(20, 50, 100, 200, 500, 1000, 2000, 5000)
```

```r
k_value <- 5000 # 固定 n 时的随机样本的数量
values <- tibble(
  n = rep(n_values, each = k_value),
  k = rep(1:k_value, length(n_values))
) # 这一步是准备好保存样本均值的 tibble 数据

target_values <- c() # 准备一个空向量
for (i in 1:length(n_values)) {
  for (j in 1:k_value) {
    target_values[(i - 1) * k_value + j] <-
      (mean(rbinom(n_values[i], size = 1, p = 0.9)) - 0.9) *
        sqrt(n_values[i]) / sqrt(0.9 * 0.1)
    # Bernoulli 分布的期望值是 p, 方差是 p(1-p)
  }
}
values <- add_column(values, target_values)

values |>
  ggplot(aes(target_values)) +
  stat_function(fun = dnorm, geom = "polygon", fill = "red", alpha = 0.3) +
  geom_density() +
  facet_wrap(vars(n), nrow = 2, labeller = "label_both") +
  labs(
    title = "Demonstrating Central Limit Theorem",
    x = "Target Value",
    y = "Density"
  )
```


Demonstrating Central Limit Theorem
n: 20
n: 50
n: 100
n: 200
n: 500
n: 1000
n: 2000
n: 5000
Density
0.0
0.2
0.4
0.6
−4
−2
0
2
Target Value