节选第四章

4.3.Give an efficient algorithm to simulate the value of a random variable X such that

$$P\{X=1\}=0.3,\quad P\{X=2\}=0.2$$
$$P\{X=3\}=0.35,\quad P\{X=4\}=0.15$$

翻译: 给出一个有效的算法来模拟随机变量 X 的值, 使得随机变量每个值的概率满足上式
解: 代码如下

```
#法一:sample抽样函数
n <- 500000
r <- sample(1:4, n, replace = TRUE,
            prob = c(0.3,0.2,0.35,0.15))
for(i in 1:4) {
    print(sum(r==i)/n)
}
#法二:逆变换法
n <- 100000
x <- seq(1, 4, 1)
p <- c(0.3, 0.2, 0.35, 0.15)
F <- c(0,cumsum(p))  #累加概率得到分布函数
result <- vector(length = n)
r <- runif(n, min = 0, max = 1)
for(i in 1:4){
    T <- r > F[i] & r <= F[i+1]
    #大于F[i]且不大于F[i+1]的就为TRUE,否则为FALSE
    #print(T) #放开注释以便理解过程
    result[T] <- x[i]
    #上一步为FALSE的此次循环不赋值,等后面的循环
    #print(result)  #放开注释以便理解过程
}
table(result)/n
```

第 3 题 (法一) 运行截图

```
> #法一:sample抽样函数
> n <- 500000
> r <- sample(1:4, n, replace = TRUE,
+             prob = c(0.3,0.2,0.35,0.15))
> for(i in 1:4) {
+     print(sum(r==i)/n)
+ }
[1] 0.3004
[1] 0.198906
[1] 0.350086
[1] 0.150608
```

图 1: 第三题 (法一)R 代码

第 3 题 (法二) 运行截图

```
> #法二:逆变换法
> n <- 100000
> x <- seq(1, 4, 1)
> p <- c(0.3, 0.2, 0.35, 0.15)
> F <- c(0,cumsum(p))  #累加概率得到分布函数
> result <- vector(length = n)
> r <- runif(n, min = 0, max = 1)
> for(i in 1:4){
+     T <- r>F[i] & r<= F[i+1]
+     #大于F[i]且不大于F[i+1]的就为TRUE,否则为FALSE
+     #print(T) #放开注释以便理解过程
+     result[T] <- x[i]
+     #上一步为FALSE的此次循环不赋值,等后面的循环
+     #print(result)  #放开注释以便理解过程
+ }
> table(result)/n
result
      1       2       3       4
0.29855 0.20125 0.34942 0.15078
```

图 2: 第三题 (法二)R 代码

4.7. A pair of fair dice are to be continually rolled until all the possible outcomes 2,3,...12 have occurred at least once. Develop a simulation study to estimate the expected number of dice rolls that are needed.

翻译: 一对质量均匀的骰子不断投掷, 直到所有可能的结果 2,3,...,12 至少出现一次. 通过随机模拟研究估计预期的掷骰次数

解: 代码如下

```
n <- 6666
count <- vector(length = n)
for(i in 1:n) {
    k <- 2:12
    while(sum(k==0) < length(k)) {
        r1 <- sample(1:6, 1)
        r2 <- sample(1:6, 1)
        k[r1+r2-1] = 0
        count[i] = count[i] + 1
    }
}
mean(count)
```

第 7 题运行截图

```
> n <- 6666
> count <- vector(length = n)
> for(i in 1:n) {
+     k <- 2:12
+     while(sum(k==0) < length(k)) {
+         r1 <- sample(1:6, 1)
+         r2 <- sample(1:6, 1)
+         k[r1+r2-1] = 0
+         count[i] = count[i] + 1
+     }
+ }
> mean(count)
[1] 61.15287
```

图 3: 第七题 R 代码

结论: 平均需要 61 次, 才可以保证 2-12 每种结果都至少出现一次.

4.13. Let X be a binomal random variable with parameters n and p. Suppose that we want to generate a random variable Y whose probability mass function is the same as the conditional mass function of X given that $X \geq k$, for some $k \leq n$. Let $\alpha = P\{X \geq k\}$ and suppose that the value of α has been computed.

(a) Give the inverse transform method for generating Y

(b) Give a second method for generating Y

(c) For what values of α, small or large, would the algorithm in (b) be inefficient?

翻译: 假设 X 是一个具有参数 n 和 p 的二项式随机变量. 现在我们想要生成一个随机变量 Y, 其概率质量函数与给定 $X \geq k$ 的 X 的条件质量函数相同, 对于某些 $k \leq n$. 假设 $\alpha = P\{X \geq k\}$ 的值已经计算出来.

(a) 给出生成 Y 的逆变换方法

(b) 给出第二种生成 Y 的方法

(c) 对于 α, 无论是小值还是大值,(b) 中的算法是否效率低下?

解:(a) 题代码如下

```
f <- function(n, k, p) {
    num <- 100000
    x <- seq(k, n, 1)
    prob_k <- vector(length = n-k+1)
    r <- rbinom(num, size = n, prob = p)
    alpha <- sum(r >= k)/num #P{X>=k}
    for(i in 1:length(prob_k)) {
      prob_k[i] <- sum(r == k+i-1)/num/alpha
      #P{X=k}/P{X>=k}
    }
    FF <- c(0,cumsum(prob_k))
    result <- vector(length = num)
    R <- runif(num, min = 0, max = 1)
    for(i in 1:n-k+1){
      T <- R > FF[i] & R <= FF[i+1]
      result[T] <- x[i]
    }
    print(table(result)/num)
  }
  f(12,5,0.4)
```

第 13 题 (a) 运行截图

```
> f <- function(n, k, p) {
+     num <- 100000
+     x <- seq(k, n, 1)
+     prob_k <- vector(length = n-k+1)
+     r <- rbinom(num, size = n, prob = p)
+     alpha <- sum(r >= k)/num #P{X>=k}
+     for(i in 1:length(prob_k)) {
+         prob_k[i] <- sum(r == k+i-1)/num/alpha
+         #P{X=k}/P{X>=k}
+     }
+     FF <- c(0,cumsum(prob_k))
+     result <- vector(length = num)
+     R <- runif(num, min = 0, max = 1)
+     for(i in 1:n-k+1){
+         T <- R>FF[i] & R<= FF[i+1]
+         result[T] <- x[i]
+     }
+     print(table(result)/num)
+ }
> f(12,5,0.4)
result
      5       6       7       8       9      10      11      12
0.40667 0.31395 0.17574 0.07497 0.02335 0.00488 0.00041 0.00003
```

图 4: 第十三题 (a)R 代码

解:(b) 题代码如下

```
g <- function(n, k, p) {
    num <- 100000
    prob_k <- vector(length = n-k+1)
    r <- rbinom(num, size = n, prob = p)
    alpha <- sum(r >= k)/num   #P{X>=k}
    for(i in 1:length(prob_k)) {
      prob_k[i] <- sum(r == k+i-1)/num/alpha
      #P{X=k}/P{X>=k}
    }
    R <- sample(k:n, num, replace = TRUE,
                prob = prob_k)
    print(table(R)/num)
  }
  g(12,5,0.4)
```

第 13 题 (b) 运行截图

```
> g <- function(n, k, p) {
+     num <- 100000
+     prob_k <- vector(length = n-k+1)
+     r <- rbinom(num, size = n, prob = p)
+     alpha <- sum(r >= k)/num  #P{X>=k}
+     for(i in 1:length(prob_k)) {
+         prob_k[i] <- sum(r == k+i-1)/num/alpha
+         #P{X=k}/P{X>=k}
+     }
+     R <- sample(k:n, num, replace = TRUE, prob = prob_k)
+     print(table(R)/num)
+ }
> g(12,5,0.4)
R
      5       6       7       8       9      10      11      12
0.40554 0.31093 0.18115 0.07463 0.02283 0.00429 0.00057 0.00006
```

图 5: 第十三题 (b)R 代码

解:(c) 题代码如下

```
#必须设置固定随机数，否则没有对比价值
set.seed(100)

#(a)算法
f <- function(n, k, p) {
  num <- 20222022
  #必须保证num(实验次数)足够大，否则结论不太准确
  x <- seq(k, n, 1)
  prob_k <- vector(length = n-k+1)
  r <- rbinom(num, size = n, prob = p)
  alpha <- sum(r >= k)/num #P{X>=k}
  print(alpha)
  for(i in 1:length(prob_k)) {
    prob_k[i] <- sum(r == k+i-1)/num/alpha
    #P{X=k}/P{X>=k}
  }
  FF <- c(0,cumsum(prob_k))
  result <- vector(length = num)
  R <- runif(num, min = 0, max = 1)
  for(i in 1:n-k+1){
```

```
    T <- R>FF[i] & R<= FF[i+1]
    result[T] <- x[i]
  }
}

#(b)算法
g <- function(n, k, p) {
    num <- 20222022
    prob_k <- vector(length = n-k+1)
    r <- rbinom(num, size = n, prob = p)
    alpha <- sum(r >= k)/num  #P{X>=k}
    for(i in 1:length(prob_k)) {
      prob_k[i] <- sum(r == k+i-1)/num/alpha
      #P{X=k}/P{X>=k}
    }
    R <- sample(k:n, num, replace = TRUE,
                prob = prob_k)
  }

#改变k的值相当于改变alpha的值
aTime <- vector(length = 6)
bTime <- vector(length = 6)
for(i in 1:6) {
  ptmf <- proc.time()
  f(12,i,0.4)
  aTime[i] <- (proc.time() - ptmf)[3]
  print(proc.time() - ptmf)
  ptmg <- proc.time()
  g(12,i,0.4)
  bTime[i] <- (proc.time() - ptmg)[3]
  print(proc.time() - ptmg)
}
```

```
#以下绘制(a)和(b)两种算法运行时间对比折线图
library(ggplot2)
dataR <- data.frame(
  number = c(seq(1,6,1),seq(1,6,1)),
  data = c(aTime, bTime),
  belong = c(rep("a",6), rep("b",6))
  )
p <- ggplot(data = dataR,
  mapping = aes(
    x = number,
    y = data,
    ))
p + geom_line(mapping = aes(color = belong)) +
  geom_point(
    size = 2,
    shape = 21,
    fill = "purple"
    ) +
  labs(
    title = "(a)和(b)两种算法运行时间对比",
    x = "alpha的值(越往右越小)",
    y = "运行时间"
  ) +
  theme(plot.title = element_text(color = "blue",
                                  face = "bold",
                                  hjust = 0.5
                                  ),
        axis.text.x = element_text(
          size = 10, color = "black"
          ),
        axis.text.y = element_text(
          size = 10, color = "black"
          ),
```

```
  axis.title.x = element_text(
    size = 10, color = "black"
    ),
  axis.title.y = element_text(
    size = 10, color = "black"
    )
)
```

第 13 题 (c) 运行截图 ——(a) 和 (b) 两种算法运行数据

```
[1] 0.9978249
用户 系统 流逝
6.14 1.17 7.39
用户 系统 流逝
2.94 0.35 3.28
[1] 0.9803611
用户 系统 流逝
6.05 1.26 7.31
用户 系统 流逝
3.06 0.39 3.45
[1] 0.9166162
用户 系统 流逝
5.78 1.00 6.80
用户 系统 流逝
2.84 0.30 3.14
[1] 0.7746419
用户 系统 流逝
5.55 1.09 6.63
用户 系统 流逝
2.83 0.22 3.04
[1] 0.5617519
用户 系统 流逝
5.39 1.03 6.43
用户 系统 流逝
2.62 0.24 2.85
[1] 0.3348113
用户 系统 流逝
5.44 0.81 6.25
用户 系统 流逝
2.59 0.19 2.79
```

图 6: 第十三题 (c) 两种算法运行数据

——(a) 和 (b) 两种算法运行时间对比

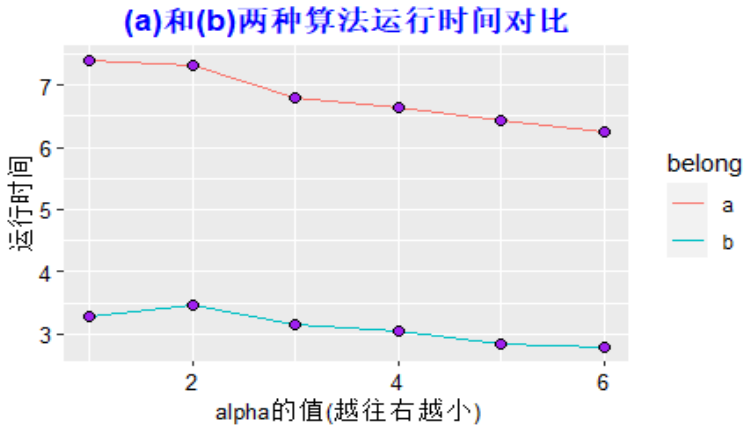

图 7: 第十三题 (c) 两种算法运行时间对比

结论: 对比 (a) 和 (b) 两种算法, 发现无论 α 的取值是大是小,(a) 算法的运行时间总是大于 (b) 算法, 因此 (b) 中的算法效率并不低下, 其运行效率要高于 (a) 算法; 而在算法内 (无论对于 (a) 还是 (b) 算法), 随着 α 取值的减小, 运行时间也有所减少

注意: 要确保在程序运行过程中电脑没有其他耗费 CPU 的程序在运行, 否则会严重影响到测算结果

4.16.Present a method to generate the value of X, where

$$P\{X=j\}=(\frac{1}{2})^{j+1}+\frac{(\frac{1}{2})2^{j-1}}{3^j},\quad j=1,2,...$$

Use the text's random number sequence to generate X.

翻译: 提出一种生成 X 值的方法, 使用随机模拟的方法生成满足上式中每个取值概率的随机变量

解: 代码如下

```
num <- 100000
prob_N <- vector(length = num)
count <- 0;
for(j in 1:length(prob_N)) {
```

```
    prob_N[j] <- 0.5^(j+1) + 0.5*2^(j-1)/3^j
    if(sum(prob_N[1:j]) >= 1) {
      break
    }
    count <- count + 1
  }
  R <- sample(1:count, num, replace = TRUE,
              prob = prob_N[1:count])
  print(table(R)/num)
```

第 16 题运行截图

```
> num <- 100000
> prob_N <- vector(length = num)
> count <- 0;
> for(j in 1:length(prob_N)) {
+     prob_N[j] <- 0.5^(j+1) + 0.5*2^(j-1)/3^j
+     if(sum(prob_N[1:j]) >= 1) {
+         break
+     }
+     count <- count + 1
+ }
> R <- sample(1:count, num, replace = TRUE,
+             prob = prob_N[1:count])
> print(table(R)/num)
R
      1       2       3       4       5       6       7       8       9      10      11
0.41825 0.23686 0.13448 0.08113 0.04894 0.02857 0.01933 0.01133 0.00731 0.00502 0.00298
     12      13      14      15      16      17      18      19      20      21      22
0.00186 0.00125 0.00083 0.00068 0.00041 0.00019 0.00022 0.00014 0.00005 0.00008 0.00003
     23      24      28
0.00002 0.00003 0.00001
```

图 8: 第十六题 R 代码

4.19.A random selection of m balls is to be made from an urn that contains n balls, n_i of which have color type i, $\Sigma_{i=1}^{r} n_i = n$. Discuss efficient procedures for simulating X_1,..., X_r, where X_i denotes the number of withdrawn balls that have color type i.

翻译: 从包含 n 个球的箱中随机抽取 m 个球, n_i 代表颜色种类为 i 的球的个数, 且 $\Sigma_{i=1}^{r} n_i = n$, 探究如何设计一个有效的程序模拟 X_1,...,X_r, 其中 X_i 代表颜色种类为 i 抽取的个数

解: 代码如下

```
#n是箱中球的总个数,m是抽取的球的数量,r是球的颜色种类
f <- function(n, m, r) {
    p <- vector(length = r)
    #随机生成每种颜色的球的概率
    R <- runif(r-1, min = 0, max = 1)
    R <- sort(R)
    for(i in 1:length(p)) {
      if(i == 1) {
        p[i] <- R[i]
      } else if(i == length(p)) {
        p[i] <- 1-R[i-1]
      } else {
        p[i] <- R[i] - R[i-1]
      }
    }
    print(round(p,3))
    #根据每种球的概率模拟生成箱中的n个球
    all <- sample(1:r, n, replace = TRUE, prob = p)
    #从包含n个球的箱中随机抽取m个(不放回)
    result <- sample(all, m, replace = FALSE)
    #输出m个球中不同颜色球的概率
    print(round(table(result)/m,3))
  }
```

第 19 题运行截图

```
> f(100000,66666,5)
[1] 0.143 0.297 0.032 0.493 0.035
result
    1     2     3     4     5
0.142 0.300 0.032 0.492 0.034
> f(100000,6666,5)
[1] 0.415 0.276 0.043 0.106 0.161
result
    1     2     3     4     5
0.414 0.272 0.047 0.104 0.163
> f(100000,666,5)
[1] 0.385 0.291 0.254 0.054 0.015
result
    1     2     3     4     5
0.422 0.282 0.221 0.063 0.012
```

图 9: 第十九题 R 代码

结论: 由输出结果可知: 总体数量不变时, 样本量越大, 抽取到的每种颜色的球的概率就越接近总体中每种颜色的球的概率