节选第五章

5.3. Use the inverse transform method to generate a random variable having distribution function

$$F(x) = \frac{x^2 + x}{2}, \quad 0 \le x \le 1$$

翻译: 使用逆变换方法生成具有如上分布函数的随机变量
解: 令 $y = \frac{x^2+x}{2}$, 则 $x = \sqrt{\frac{1}{4} + 2y} - \frac{1}{2}$, 模拟生成 x 即可, 代码如下

```
library(ggplot2)
n <- 188888
U <- runif(n)
X <- sqrt(0.25+2*U)-0.5
#绘制分布函数图像
dataResult <- data.frame(data = X)
p <- ggplot(dataResult, aes(x = data))
p + stat_ecdf(size = 0.1) +
    labs(title = "X的分布函数图像",
         x = "X",
         y = "累积概率") +
    theme(plot.title = element_text(
        color = "blue", face = "bold", hjust = 0.5
        ))
#绘制概率密度图
hist(X, freq = FALSE, breaks = 100,
     col = "yellow", main = "X的密度函数图像")
#拟合理论密度曲线
Y <- seq(0,1,0.001)
lines(Y, Y+0.5, col = "red", lwd = 2)
```

第 3 题运行截图

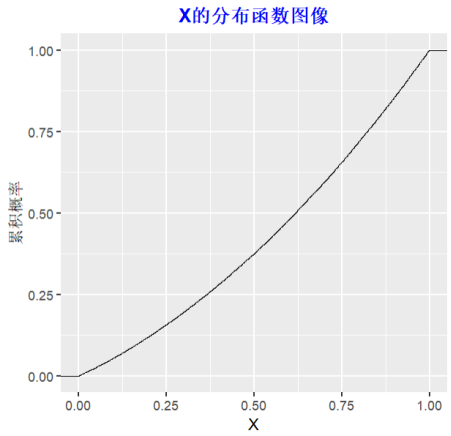

图 2: 题 3: 分布函数

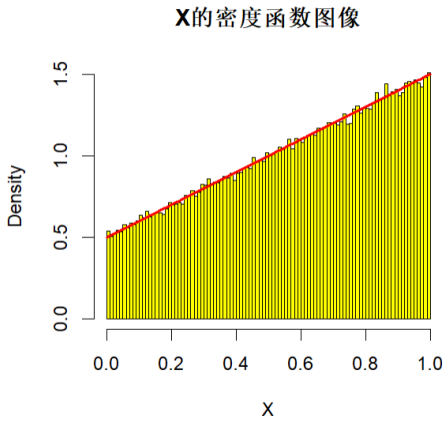

图 3: 题 3: 概率密度函数

5.5. Give a method for generating a random variable having density function

$$f(x) = \begin{cases} e^{2x}, & -\infty < x < 0 \\ e^{-2x}, & 0 < x < \infty \end{cases} \tag{1}$$

翻译: 给出一种生成具有如上密度函数的随机变量的方法

解: 当 $x < 0$ 时,

$$-\int_0^x e^{2t}dt = -\frac{1}{2}e^{2t}\Big|_0^x = \frac{1}{2}(1-e^{2x})$$

令 $x = \frac{1}{2}(1-e^{2y})$, 则 $y = \ln\sqrt{1-2x}$, 则定义域为 $x < \frac{1}{2}$

当 $x > 0$ 时,

$$\int_0^x e^{-2t}dt = -\frac{1}{2}e^{-2t}\Big|_0^x = \frac{1}{2}(1-e^{-2x})$$

令 $x = \frac{1}{2}(1-e^{-2y})$, 则 $y = \ln\sqrt{\frac{1}{1-2x}}$, 则定义域为 $x < \frac{1}{2}$

代码如下

```
n <- 66666
UR <- runif(n, min = 0, max = 0.5)
XR <- log(sqrt(1/(1-2*U)))
UL <- runif(n, min = 0, max = 0.5)
XL <- log(sqrt(1-2*U))
```

```
X <- append(XR,XL)
#绘制概率密度图
hist(X, freq = FALSE, breaks = 100,
     col = "yellow",
     main = "X的密度函数图像")
#拟合理论密度曲线
Y1 <- seq(-5,0,1/100)
Y2 <- seq(0,5,1/100)
lines(Y1, exp(2*Y1), col = "red", lwd = 2)
lines(Y2, exp(-2*Y2), col = "red", lwd = 2)
```

第 5 题运行截图

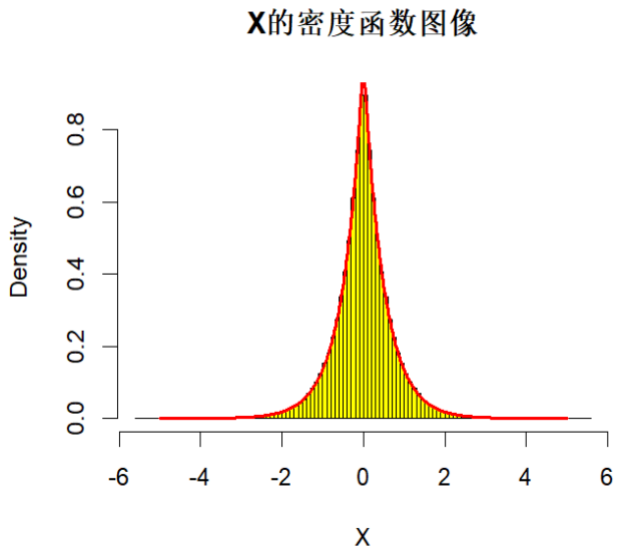

图 3: 题 5: 概率密度函数

5.8.Using the result of Exercise 7, give algorithms for generating random variables from the following distributions.

翻译: 使用练习 7 的结论, 给出从以下分布生成随机变量的算法

(a) $F(x) = \frac{x+x^3+x^5}{3}, \quad 0 \le x \le 1$

4

解 (a): $F(x) = p_1F_1(x) + p_2F_2(x) + p_3F_3(x)$
可以令 $p_1 = p_2 = p_3 = \frac{1}{3}$, $F_1(x) = x$, $F_2(x) = x^3$, $F_3(x) = x^5$
(a) 题代码如下

```
f <- function(n) {
    result <- vector(length = n)
    for(i in 1:n) {
#每次循环要生成两个随机数，一个随机数用来控制概率
#另一个随机数用来生成x变量
      U <- runif(2, min = 0, max = 1)
      if(U[1] < 1/3) {
    #有三分之一的概率会生成y=x
        result[i] <- U[2]
      } else if(U[1] < 2/3) {
    #有三分之一的概率会生成y=x^3,那么x就是y^(1/3)
        result[i] <- U[2]^(1/3)
      } else {
    #有三分之一的概率会生成y=x^5,那么x就是y^(1/5)
        result[i] <- U[2]^(1/5)
      }
    }
    #绘制概率密度图
    hist(result, freq = FALSE, breaks = 50,
         col = "yellow", main = "X的密度函数图像")
    Y <- seq(0,1,0.001)
    #拟合理论密度曲线
    lines(Y, (1+3*Y^2+5*Y^4)/3,
          col = "red", lwd = 2)
}
f(66666)
```

第 8 题 (a) 运行截图

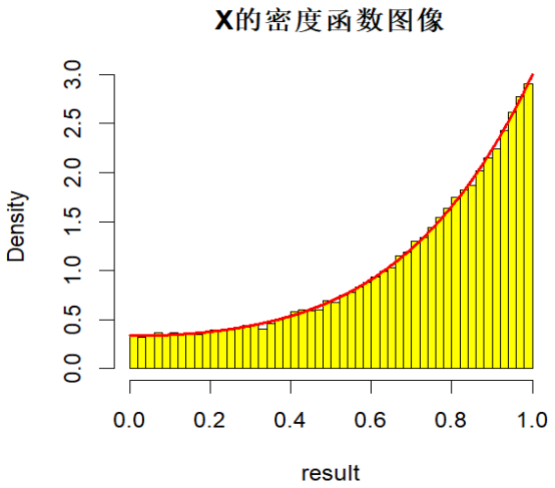

图 4: 题 8(a): 概率密度函数

(b)

$$F(x) = \begin{cases} \dfrac{1 - e^{-2x} + 2x}{3}, & if\ \ 0 < x < 1 \\ \dfrac{3 - e^{-2x}}{3}, & if\ \ 1 < x < \infty \end{cases} \tag{2}$$

解 (b): 可以令 $p_1 = \frac{2}{3}$, $F_1(x) = x$; $p_2 = \frac{1-e^{-2}}{3}$, $F_2(x) = \frac{1-e^{-2x}}{1-e^{-2}}$; $p_3 = \frac{e^{-2}}{3}$, $F_3(x) = 1-e^{-2(x-1)}$ 此时 $p_1+p_2+p_3 = 1$, 且保证了 $F(1) = p_1+p_2 = \frac{3-e^{-2}}{3}$, $F_1(0) = 0$, $F_1(1) = 1$; $F_2(0) = 0$, $F_2(1) = 1$; $F_3(1) = 0$, $F_3(+\infty) = 1$

(b) 题代码如下:

```
f <- function(n) {
    result <- vector(length = n)
    for(i in 1:n) {
  #每次循环要生成两个随机数，一个随机数用来控制概率
  #另一个随机数用来生成x变量
      U <- runif(2, min = 0, max = 1)
      if(U[1] <= 2/3) {
  #当0<U[1]<p1时，生成服从F1(x)的随机变量
        result[i] <- U[2]
      } else if(U[1] < (3-exp(1)^(-2))/3) {
  #当p1<U[1]<(p1+p2)时，生成服从F2(x)的随机变量
```

```
            result[i] <- -0.5*log(1-U[2]*(1-exp(1)^(-2)))
        } else {
    #当(p1+p2)<U[1]<1时，生成服从F3(x)的随机变量
            result[i] <- 1-0.5*log(1-U[2])
        }
    }
    #绘制概率密度图
    hist(result, freq = FALSE, breaks = 100,
         col = "yellow", main = "X的密度函数图像")
    #拟合理论密度曲线
    Y1 <- seq(0,1,0.001)
    lines(Y1, (2+2*exp(1)^(-2*Y1))/3,
          col = "red", lwd = 2)
    Y2 <- seq(1,3,0.01)
    lines(Y2, (2*exp(1)^(-2*Y2))/3,
          col = "red", lwd = 2)
}
f(100000)
```

第 8 题 (b) 运行截图

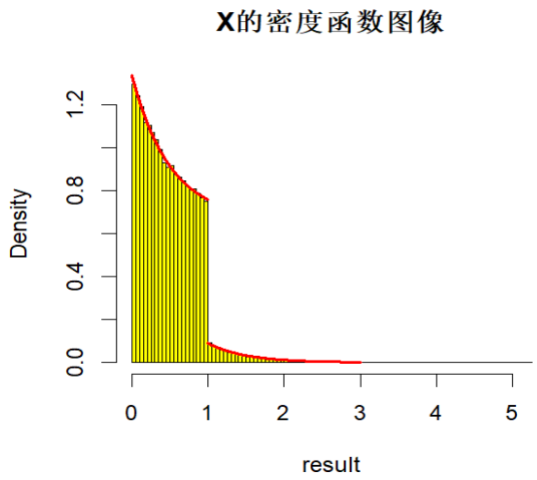

图 5: 题 8(b): 概率密度函数

(c) $F(x) = \Sigma_{i=1}^{n} \alpha_i x^i, 0 \leq x \leq 1, \quad \text{where } \alpha_i \geq 0, \quad \Sigma_{i=1}^{n} \alpha_i = 1$

解 (c): 此题 α_i 可以随机生成, $F_i(x)$ 为幂函数, 可以瞎令, 只需服从指数为正整数且不重复即可

(c) 题代码如下

```
f <- function(m,n) {
    #随机生成每一个α
    #这里的α是为了保证随机性，方便起见可以自己赋值
#此处随机生成n个α的原理:
#首先随机生成n-1个在(0,1)区间的数
    #比如生成了0.49,0.25,0.34,0.08
#对生成的n-1个数进行排序
    #排序后:0.08,0.25,0.34,0.49
# 1 = α1 + (α2-α1) + (α3-α2) + (α4-α3) + (1-α4)
#即生成了:0.08,0.17,0.09,0.15,0.51,加起来刚好是1
    alpha <- vector(length = n)
    R <- runif(n-1, min = 0, max = 1)
    R <- sort(R)
    for(i in 1:length(alpha)) {
      if(i == 1) {
        alpha[i] <- R[i]
      } else if(i == length(alpha)) {
        alpha[i] <- 1-R[i-1]
      } else {
        alpha[i] <- R[i] - R[i-1]
      }
    }
    print(round(alpha,2))
    result <- vector(length = m)
    for(i in 1:m) {
  #每次循环要生成两个随机数，一个随机数用来控制概率
  #另一个随机数用来生成x变量
      U <- runif(2, min = 0, max = 1)
      for(j in 1:n) {
```

```
        #循环，看U[1]属于哪一个累积概率区间
            if(U[1] < cumsum(alpha)[j]) {
              result[i] <- (U[2])^(1/(j))
              break
            }
          }
        }
        #绘制概率密度图
        hist(result, freq = FALSE, breaks = 100,
             col = "yellow")
        #拟合理论密度曲线
        Y <- seq(0,1,0.00001)
        QX <- vector(length = length(Y))
        for(i in 1:n) {
          QX <-  QX + i * alpha[i] * Y^(i-1)
        }
        lines(Y, QX, col = "red", lwd = 2)
    }
    #多组α取值下的概率密度函数拟合效果
    par(mfrow = c(2,3))
    n <- 300000
    for(i in 2:7) {
      f(n,i)
    }
```

第 8 题 (c) 运行截图

```
> for(i in 2:7) {
+     f(n,i)
+ }
[1] 0.39 0.61
[1] 0.21 0.61 0.18
[1] 0.72 0.14 0.10 0.03
[1] 0.36 0.25 0.34 0.01 0.05
[1] 0.39 0.03 0.04 0.28 0.19 0.07
[1] 0.37 0.10 0.13 0.21 0.04 0.08 0.07
```

图 6: 题 8(c): 多组 α 取值结果

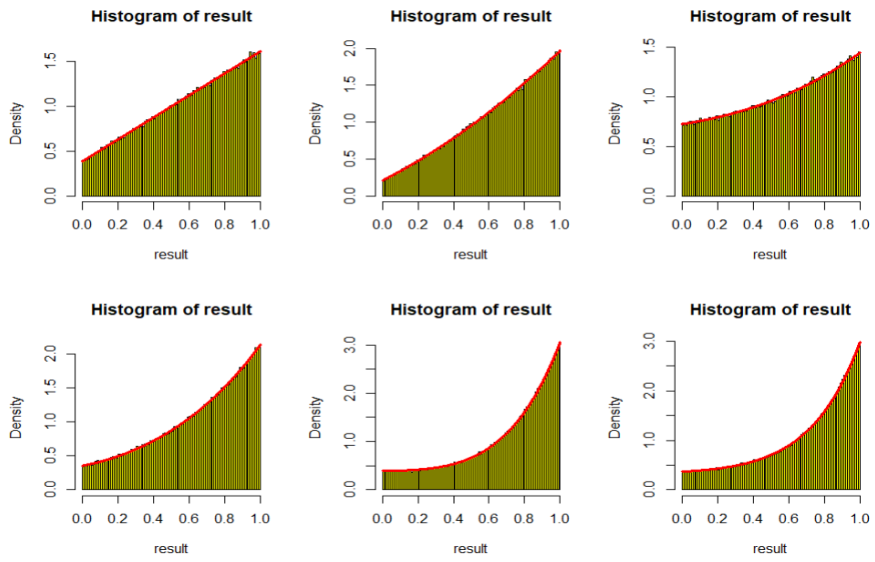

图 7: 题 8(c): 多组 α 取值下的概率密度函数

(c) 题扩充: 可以用与 (c) 题类似的代码模拟出 (a) 题, 代码如下:

```
f <- function(m) {
    alpha <- c(1/3,0,1/3,0,1/3)
    print(round(alpha,2))
    result <- vector(length = m)
    for(i in 1:m) {
      U <- runif(2, min = 0, max = 1)
      for(j in 1:5) {
        if(U[1] < cumsum(alpha)[j]) {
          result[i] <- (U[2])^(1/(j))
          break
        }
      }
    }
    hist(result, freq = FALSE, breaks = 100,
         col = "yellow", main = "8(a)概率密度图")
    Y <- seq(0,1,0.00001)
    QX <- vector(length = length(Y))
```

```
    for(i in 1:5) {
      QX <- QX + i * alpha[i] * Y^(i-1)
    }
    lines(Y, QX, col = "red", lwd = 2)
}
par(mfrow = c(1,1))
f(100000)
```

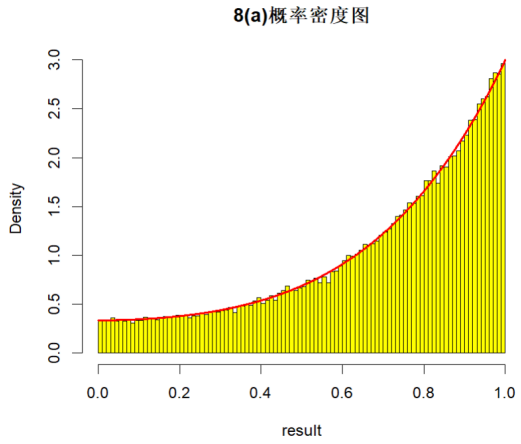

图 8: 8(c) 类似思路代码模拟 8(a)

5.10. A casually insurance company has 1000 policyholders, each of whom will independently present a claim in the next month with probability .05. Assuming that the amounts of the claims made are independent exponential random variables with mean $800, use simulation to estimate the probability that the sum of these claims exceeds $50,000.

翻译: 一家临时保险公司有 1000 名投保人, 每个投保人将在下个月以 0.05 的概率独立提出索赔。假设索赔金额为平均值为 800 美元的独立指数随机变量, 使用模拟来估计这些索赔金额超过 50000 美元的概率

解: 代码如下

```
f <- function(n){
    count <- vector(length = n)
    for(i in 1:n) {
```

```
        r <- sample(c(0,1), 1000, replace = TRUE,
                    prob = c(0.95,0.05))
        U <- rexp(10000, 1/800)
        for(j in 1:1000) {
            if(r[j] == 1) {
              get <- sample(U, 1)
              count[i] <- count[i] + get
            }
        }
    }
    sum(count > 50000)/n
}
f(8888)
```

```
> f <- function(n){
+     count <- vector(length = n)
+     for(i in 1:n) {
+         r <- sample(c(0,1), 1000, replace = TRUE,
+                     prob = c(0.95,0.05))
+         U <- rexp(10000, 1/800)
+         for(j in 1:1000) {
+             if(r[j] == 1) {
+                 get <- sample(U, 1)
+                 count[i] <- count[i] + get
+             }
+         }
+     }
+     sum(count > 50000)/n
+ }
> f(8888)
[1] 0.1065482
```

图 9: 第 10 题模拟结果

结论: 由模拟结果知, 索赔金额超过 50000 美元的概率为 0.1065482

5.14.Let G be a distribution function with density g and suppose, for constants $a < b$, we want to generate a random variable from the distribution function

$$F(x) = \frac{G(x) - G(a)}{G(b) - G(a)}, \quad a \le x \le b$$

(a) If X has distribution G, then F is the conditional distribution of X given what

information?

(b) Show that the rejection method reduces in this case to generating a random variable X having distribution G and then accepting it if it lies between a and b.

翻译: 设 G 是密度为 g 的分布函数, 假设常数 $a < b$, 我们想从分布函数生成一个随机变量 X, 且 X 的分布函数满足上式 F(x)

(a) 如果 X 有分布 G, 那么 F 是给定什么信息的 X 的条件分布?

(b) 在这种情况下, 可简化为用舍选法生成具有分布 G 的随机变量 X, 然后如果它位于 a 和 b 之间, 则接受它

解 (a):

$F(x)$ 是由分布函数 $G(x)$ 生成, 且给定 $F(a) = 0,\ \ F(b) = 1$ 的条件分布, 且需要满足 $G(-\infty) = 0,\ \ G(+\infty) = 1$

解 (b):

”Logistic 函数” 有一个很好的性质刚好满足题目对 $G(x)$ 的要求, 且具有一般性。为方便起见, 令 $G(x)$ 为标准”Logistic 函数”

$$G(x) = \frac{1}{1 + e^{-x}}$$

其满足: $G(-\infty) = 0,\ \ G(+\infty) = 1$, 令其导数为 $g(x)$

$$g(x) = \frac{e^{-x}}{(1 + e^{-x})^2}$$

根据题意, 用” 舍选抽样法” 模拟生成具有密度函数为 $g(x)$ 的 X 随机变量, 如果随机变量 X 位于 a 与 b 之间, 则接受它, 这样就生成了符合条件 (其分布函数为 $F(x)$) 的随机变量 X

代码如下:

```
library(ggplot2)
Fdistribution <- function(n,a,b){
  k = 0
  z = rep(0,n)
    g <- function(x) {
      gx = 0.5*exp(-x)
      fx =  (exp(-x)) / (1+exp(-x))^2
      cmax <- fx/gx
      return(cmax)
    }
```

```
    c <- optimize(g, lower = 0, upper = 100,
                  maximum = TRUE)$objective
    print(c)
  while(k < n) {
    Df = runif(1)
    x <- -log(Df)    #生成范围在(0,+∞)的随机数
    R = runif(1)
    gx = 0.5*exp(-x)
    fx = (exp(-x)) / (1+exp(-x))^2
    if(c * gx * R <= fx){
      r = runif(1)
#符合G(x)分布的随机变量的生成，应当正负概率各占一半
#故当r>0.5时，生成正数，且若该正数位于区间(a,b)，则接受它
      if(r > 0.5) {
        if(x > a && x < b) {
          z[k] = x
          k = k + 1
        }
#当r<0.5时，生成负数，且若该负数位于区间(a,b)，则接受它
      } else {
        if(-x > a && -x < b) {
          z[k] = -x
          k = k + 1
        }
      }
    }
  }
  return(z)
}
a <- -3
b <- 2
resultP <- Fdistribution(188888,-3,2)
#生成密度函数f(x)
```

```
hist(resultP, prob = TRUE, col = "yellow",
     breaks = 100, main = "X的概率密度函数")
#拟合理论密度函数曲线
xx = seq(a,b,1/100)
#yy为F(x)的导数，即变量X的密度函数
yy =   ((1+exp(-a))*(1+exp(-b))*(exp(-xx))) /
       (((1+exp(-xx))^2)*(exp(-a)-exp(-b)))
lines(xx, yy, col = "red",lwd = 2)
dataResult <- data.frame(data = resultP)
#生成分布函数F(x)
ggplot(dataResult, aes(x = data)) +
       stat_ecdf(size = 0.1) +
       labs(title = "X的分布函数图像",
            x = "X",
            y = "累积概率") +
       theme(plot.title = element_text(
             color = "blue",
             face = "bold",
             hjust = 0.5)
            )
```

运行结果如下:

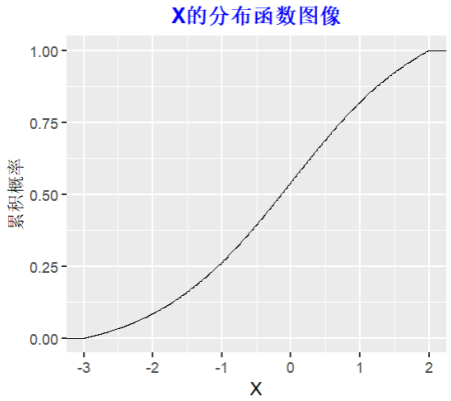

图 11: 题 14: 分布函数

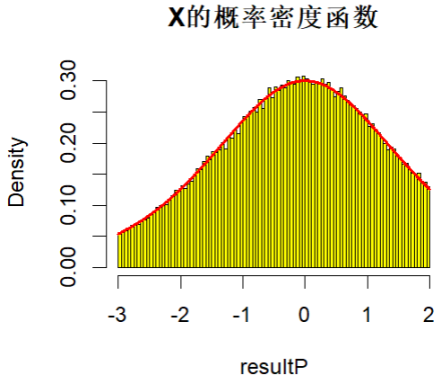

图 12: 题 14: 概率密度函数

5.19. In Example 5f we simulated a normal random variable by using the rejection technique with an exponential distribution with rate 1. Show that among all exponential density functions $g(x) = \lambda e^{-\lambda x}$ the number of iterations needed is minimized when $\lambda = 1$.

翻译: 在示例 5f 中, 我们通过使用速率为 1 的指数分布的舍选抽样法模拟生成了一个正态随机变量。证明在所有的指数密度函数中, 当 $\lambda = 1$ 时, $g(x) = \lambda e^{-\lambda x}$ 所需的迭代次数最小

解: 纯数学证明过程:

$$f(x) = \frac{2}{\sqrt{2\pi}} e^{\frac{-x^2}{2}} \quad 0 < x < \infty$$

$$g(x) = \lambda e^{-\lambda x} \quad 0 < x < \infty$$

$$C = Max \frac{f(x)}{g(x)} = (\frac{\frac{2}{\sqrt{2\pi}} e^{\frac{-x^2}{2}}}{\lambda e^{-\lambda x}})_{max} = (\frac{\sqrt{\frac{2}{\pi}} e^{\lambda x - \frac{x^2}{2}}}{\lambda})_{max}$$

即转化为求解 $F(x) = e^{\lambda x - \frac{x^2}{2}}$ 的最大值, 令其导数为 $f(x)$, 则:

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

故: 当 $\lambda > x$ 时,$F(x)$ 单调递增; 当 $\lambda < x$ 时,$F(x)$ 单调递减; 所以在 $\lambda = x$ 时,$F(x)$ 有最大值, 因此:

$$C = Max \frac{f(x)}{g(x)} = \frac{\sqrt{\frac{2}{\pi}} e^{\frac{\lambda^2}{2}}}{\lambda}$$

现在需要求解当 C 为最小值时,λ 的取值为多少, 令 $C(\lambda) = \frac{e^{\frac{\lambda^2}{2}}}{\lambda}$

则其导数 $c(\lambda)$ 为:

$$c(\lambda) = \frac{(\lambda^2 - 1) e^{\frac{\lambda^2}{2}}}{\lambda^2}$$

故: 当 $\lambda > 1$ 时,$C(\lambda)$ 单调递增; 当当 $0 < \lambda < 1$ 时,$C(\lambda)$ 单调递减; 所以当 $\lambda = 1$ 时,C 有最小值。以下用计算机程序模拟:

```
n <- 120
c <- data.frame(lambda = rep(0,n),
                cMax = rep(0,n))
Df <- seq(0,3,0.025)
f <- function(lambda) {
```

```
    g <- function(x) {
      gx = lambda*exp(-lambda*x)
      fx = (2/pi)^(1/2)*exp(-1/2*x^2)
      cmax <- fx/gx
    }
    #求解在每一个λ水平下的C值(迭代次数)
    result <- optimize(g, lower = 0, upper = 10,
                       maximum = TRUE)$objective
    answerLambda <- optimize(g, lower = 0,
                upper = 10, maximum = TRUE)$maximum
    re <- list(cMax = result,
               aL = round(answerLambda,3))
    return(re)
  }
  for(i in 1:n) {
      c[i,1] <- f(Df[i+1])$aL
      c[i,2] <- f(Df[i+1])$cMax
  }
  plot(x = c$lambda, y = c$cMax, col = "purple")
  #找到迭代次数最小的所在行
  row <- which.min(c$cMax)
  c[row,]
```

运行结果如下:

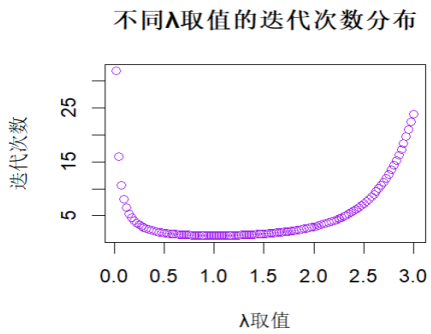

图 12: 19 题: 不同 λ 取值下的迭代次数分布

由上图知, 迭代次数随着 λ 取值的增加, 先是单调递减, 然后是单调递增

```
> row <- which.min(c$cMax)
> c[row,]
   lambda     cMax
40      1 1.315489
```

图 13: 19 题: 迭代次数最小时的 λ 取值

由上图结果知: 迭代次数在 $\lambda = 1$ 时取到最小值, 且最小迭代次数为 1.315489

5.20. Write a program that generates normal random variables by the method of Example 5f.

翻译: 用 5f 的方法编写一个程序生成服从正态分布的随机变量

解: 代码如下

```
#n为模拟次数;mu为均值;sigma为标准差
normalDis <- function(n, mu, sigma){
    k = 0
    z = rep(0,n)
      g <- function(x) {
        gx = exp(-x)
        fx = (1 / ((sqrt(2*pi)) * sigma)) *
             (exp( - (x-mu)^2 / (2*sigma^2)))
        cmax <- fx/gx
      }
      c <- optimize(g, lower = -10, upper = 10,
                    maximum = TRUE)$objective
    while(k < n) {
      Df = runif(1)
      #生成范围在(mu,+∞)的随机数
      x = -log(Df) + mu
      R = runif(1)
      gx = exp(-x)
      fx = (1 / ((sqrt(2*pi)) * sigma)) *
           (exp( - (x-mu)^2 / (2*sigma^2)))
```

```
        if(c * gx * R <= fx){
          r = runif(1)
#生成的随机数应当有一半的概率大于mu,一半的概率小于mu
#故当r<0.5时,生成大于mu的随机数(上面生成的x刚好大于mu)
          if(r < 0.5) {
            z[k] = x
          } else {
#当r>0.5时,生成小于mu的随机数(-x+2mu刚好符合这个条件)
#因为-x属于(-∞,-mu)区间,则-x+2mu属于(-∞,mu)区间
            z[k] = -x + 2*mu
          }
          k = k + 1
        }
      }
      return(z)
    }
    #不同均值和方差的正态分布模拟
    #由于模拟次数太大,且在舍选抽样法中对于方差过大的
    #正态分布模拟效果较低,故只选取方差较小的正态分布
    #进行模拟,否则容易死机
    par(mfrow = c(3,3))
    #变量"i"控制均值;变量"j"控制标准差
    for(i in 1:3) {
      for(j in 1:3) {
        mu <- i
        sigma <- j
        #生成正态分布的密度函数
        hist(normalDis(66666, mu, sigma),
             prob = TRUE, col = "yellow",
             breaks = 100, main = "正态分布图")
        #拟合理论密度函数曲线
        xx = seq(-20,20,1/100)
        yy = (1 / ((sqrt(2*pi)) * sigma)) *
```

```
            (exp ( - (xx-mu)^2 / (2*sigma^2) ) )
        lines(xx, yy, col = "red",lwd = 2)
    }
}
```

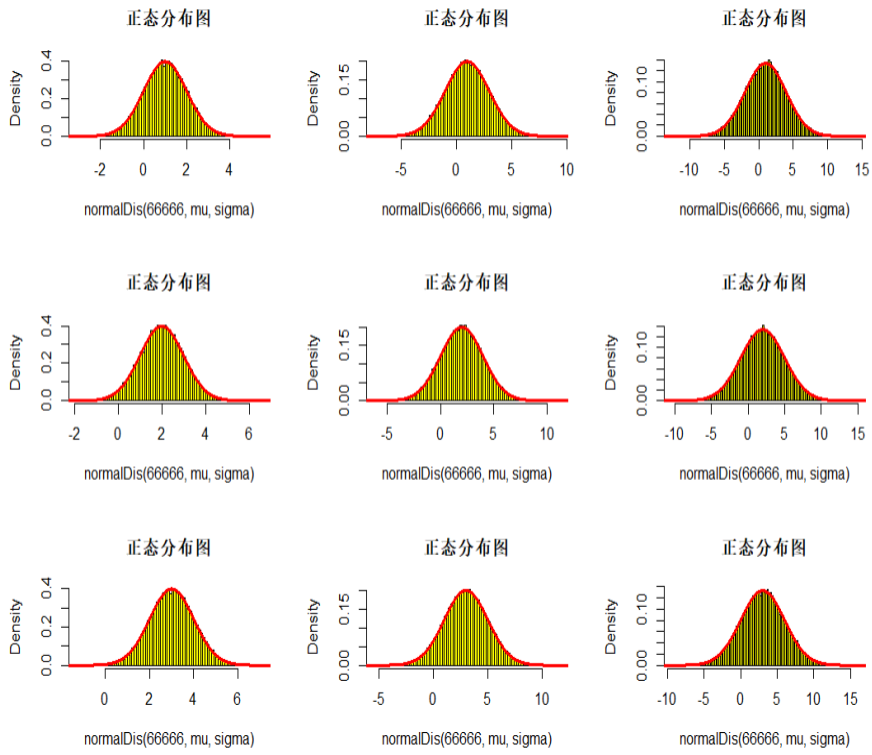

图 14: 20 题: 九组不同均值和标准差取值下的模拟结果

注意: 从左到右, 标准差依次增大; 从上到下, 均值依次增大

5.22. Write a program that generates the first T time units of a Poisson process having rate λ.

翻译: 编写一个程序, 生成具有速率为 λ 的前 T 个时间单位的泊松过程

解: 代码如下:

```
f <- function(lambda, T) {
```

```
g <- function(lambda, T) {
  #S数组用来存储每次事件发生的时间
  S <- vector(length = 10*lambda*T)
  t = 0
  #C为事件发生次数
  C = 0
  for(i in 1:length(S)) {
    Df = runif(1)
    x = -log(Df)  #生成范围在(0,+∞)的随机数
    t = t+x/lambda
    #若时间累加大于既定时间,则退出循环
    if(t > T) {
      break
    }
    C = C+1
    S[C] = t
  }
  re <- list(count = C)
  return(re)
}
n <- 66666
resultCount <- vector(length = n)
for(i in 1:n) {
  re <- g(lambda, T)
  resultCount[i] <- re$count
}
print(mean(resultCount))
#生成次数分布图
hist(resultCount, freq = FALSE,
     col = "yellow", main = "次数分布图",
     xlab = "次数", ylab = "频率",
     breaks = 30)
}
```

```
par(mfrow = c(2,2))
#模拟四组结果
f(3, 6)
f(4, 5)
f(10,3)
f(5, 8)
```

运行结果如下:

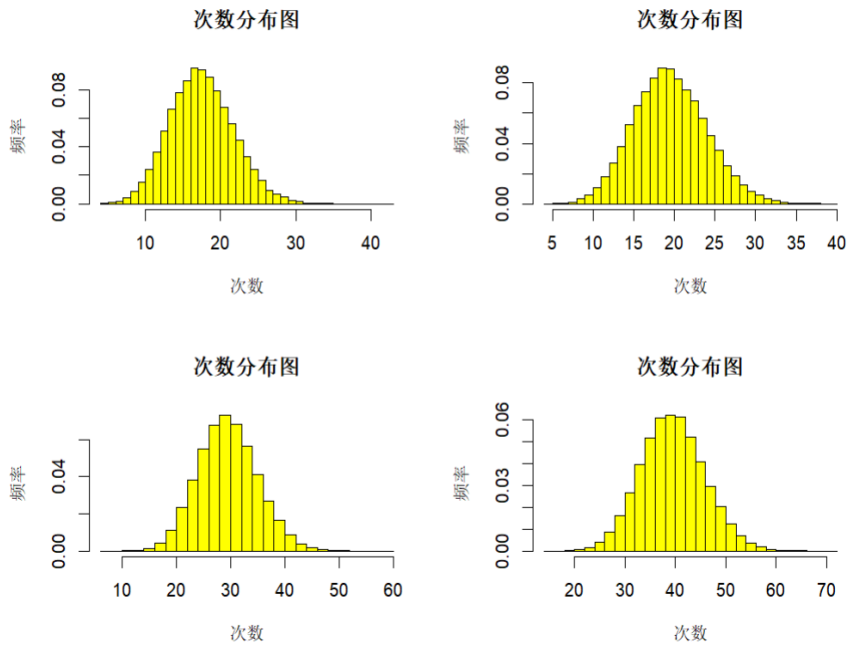

图 15: 22 题: 四组不同 λ 和 T 取值下的泊松过程

```
> #模拟四组结果
> f(3, 6)
[1] 17.99044
> f(4, 5)
[1] 20.01067
> f(10,3)
[1] 29.97472
> f(5, 8)
[1] 40.01446
```

图 16: 22 题: 四组不同 λ 和 T 取值的齐次泊松过程模拟

5.23. To complete a job a worker must go through k stages in sequence. The time to complete stage i is an exponential random variable with rate λ_i, $i = 1, ..., k$. However, after completing stage i the worker will only go to the next stage with probability α_i, $i = 1, ..., k-1$. That is, after completing stage i the worker will stop working with probablity $1 - \alpha_i$, If we let X denote the amount of time that the worker spends on the job, the X is called a Coxian random variable. Write an algorithm for generating such a random variable.

翻译: 要完成一项工作, 工人必须依次经历 k 个阶段。完成阶段 i 的时间是一个指数随机变量, 速率 λ_i, $i = 1, ..., k$。然而, 在完成第 i 阶段后, 工人将仅以概率 α_i, $i = 1, ..., k-1$ 进入下一阶段。也就是说, 在完成第一阶段后, 员工将以概率 $1 - \alpha_i$ 停止工作。如果我们让 X 表示工人在工作上花费的时间量, 则 X 被称为 $Coxian$ 随机变量。编写一个生成这样一个随机变量的算法.

解: 代码如下:

```
#参数k为工作阶段数,参数n为工人人数
g <- function(k,n) {
  #随机生成k个λ:
  lambda <- vector(length = k)
  for(i in 1:k) {
    lambda[i] <- runif(1, min = 0, max = 10)
  }
  print("λ的值为:")
  print(lambda)
  #随机生成k-1个α:
  alpha <- vector(length = k-1)
  for(i in 1:k-1) {
    alpha[i] <- runif(1, min = 0, max = 1)
  }
  print("α的值为:")
  print(alpha)
  #生成m <= k轮服从指数分布的随机变量
  f <- function(k){
      T <- vector(length = k)
      #每个工人都会参与第一阶段的工作
```

```
      T[1] <- rexp(1, rate = lambda[1])
      for(i in 2:k) {
        #判断下一阶段是否工作
        R <- sample(c(0,1), 1,
              prob = c(1-alpha[i-1], alpha[i-1]))
        #不工作直接退出
        if(R == 0) {
          break
        }
        #继续工作的把工时加上
        T[i] <- rexp(1, rate = lambda[i])
      }
      return(round(sum(T),3))
  }
  #重复n次实验(相当于n个工人样本)
  result <- vector(length = n)
  for(j in 1:n) {
    result[j] <- f(k)
  }
  hist(result, main = "工人工作时间分布图",
       xlab = "工作时间", ylab = "频率",
       col = "green", freq = FALSE, breaks = 50)
  print("在该组λ和α取值情况下,工人的平均工作时间如下")
  print(mean(result))
  #计算理论工作时长
  print("在该组λ和α取值情况下,工人的理论工作时间如下")
  sum <- 1/lambda[1]
  for(i in 2:k) {
    sum <-  sum + cumprod(alpha)[i-1] / lambda[i]
  }
  print(sum)
}
#重复6次模拟实验
```

```
par(mfrow = c(3,2))
for(k in 2:7) {
  print("----------------------------------")
  g(k,100000)
}
```

运行结果如下:

```
[1] "----------------------------------"
[1] "λ的值为:"
[1] 6.499465 9.133318
[1] "α的值为:"
[1] 0.4773828
[1] "在该组λ和α取值情况下,工人的平均工作时间如下"
[1] 0.2054208
[1] "在该组λ和α取值情况下,工人的理论工作时间如下"
[1] 0.2061271
[1] "----------------------------------"
[1] "λ的值为:"
[1] 9.892133 1.809626 8.487215
[1] "α的值为:"
[1] 0.8945128 0.1940469
[1] "在该组λ和α取值情况下,工人的平均工作时间如下"
[1] 0.6178789
[1] "在该组λ和α取值情况下,工人的理论工作时间如下"
[1] 0.6158502
```

图 17: 23 题: 前两组不同 λ 和 α 取值下的工人工作时间模拟结果

```
[1] "----------------------------------"
[1] "λ的值为:"
[1] 5.4632249 3.4558876 7.1573199 0.2545739
[1] "α的值为:"
[1] 0.9579377 0.8675252 0.6856702
[1] "在该组λ和α取值情况下,工人的平均工作时间如下"
[1] 2.809569
[1] "在该组λ和α取值情况下,工人的理论工作时间如下"
[1] 2.814654
[1] "----------------------------------"
[1] "λ的值为:"
[1] 4.927234 7.575540 4.509788 8.754706 2.018394
[1] "α的值为:"
[1] 0.6325136 0.1408315 0.9344295 0.7580280
[1] "在该组λ和α取值情况下,工人的平均工作时间如下"
[1] 0.3465257
[1] "在该组λ和α取值情况下,工人的理论工作时间如下"
[1] 0.3469681
```

图 18: 23 题: 中间两组不同 λ 和 α 取值下的工人工作时间模拟结果

```
[1] "λ的值为:"
[1] 1.823820 3.506659 5.766801 8.575364 5.476647 6.464705
[1] "α的值为:"
[1] 0.73354369 0.01645873 0.28551971 0.60591021 0.21939284
[1] "在该组λ和α取值情况下,工人的平均工作时间如下"
[1] 0.7579861
[1] "在该组λ和α取值情况下,工人的理论工作时间如下"
[1] 0.7604334
[1] "---------------------------------"
[1] "λ的值为:"
[1] 9.936085 5.646849 3.647334 8.218913 6.122829 1.235166 9.982415
[1] "α的值为:"
[1] 0.6200187 0.5617920 0.9202131 0.8477977 0.5812738 0.0138175
[1] "在该组λ和α取值情况下,工人的平均工作时间如下"
[1] 0.5141635
[1] "在该组λ和α取值情况下,工人的理论工作时间如下"
[1] 0.5174267
```

图 19: 23 题: 后两组不同 λ 和 α 取值下的工人工作时间模拟结果

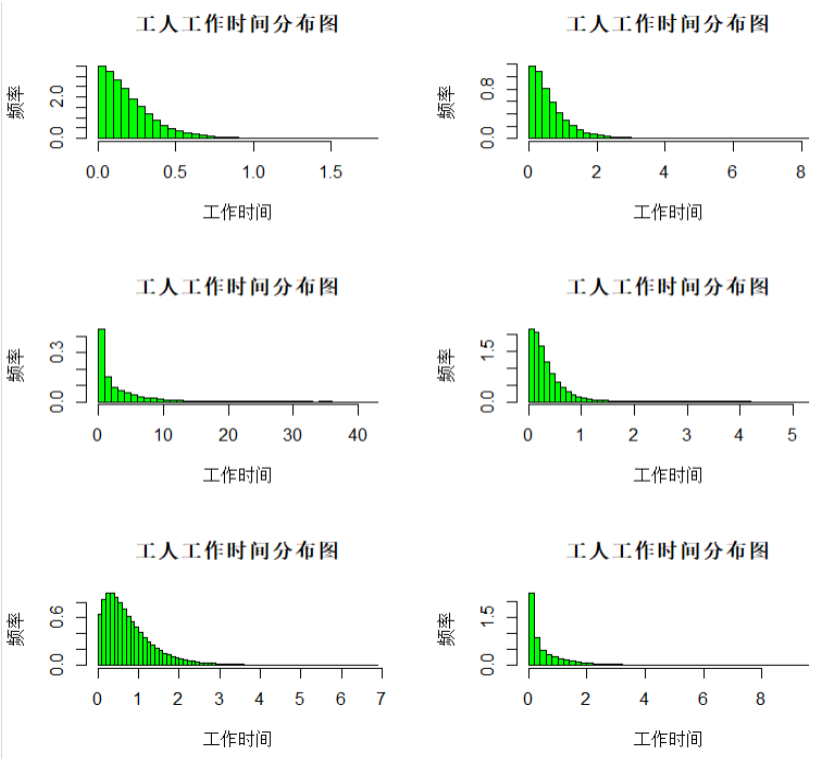

图 20: 23 题:6 组不同 λ 和 α 取值下的工人工作时间分布图

5.24.Buses arrive at a sporting event according to a Poisson process with rate 5 per hour. Each bus is equally likely to contain either 20,21,...,40 fans, with the numbers in the different buses being independent. Write an algorithm to simulate arrival of fans to the event by time $t = 1$.

翻译: 公交车按照泊松过程到达体育赛事, 速度为每小时 5 辆。每个总线都可能包含 20,21,...,40 个风扇, 不同总线中的数字是独立的。编写一个算法, 模拟" 粉丝在时间 $t = 1$ 之前到达" 这个事件.

解: 代码如下:

```
f <- function(n) {
  fansN <- vector(length = n)
  for(i in 1:n) {
    amount <- rpois(1, lambda = 5)
    fansCount <- vector(length = amount)
    for(j in 1:amount) {
      fansCount[j] <- sample(20:40, 1)
    }
    fansN[i] <- sum(fansCount)
  }
  print(mean(fansN))
}
f(16666)
```

运行结果如下:

```
> f <- function(n) {
+     fansN <- vector(length = n)
+     for(i in 1:n) {
+         amount <- rpois(1, lambda = 5)
+         fansCount <- vector(length = amount)
+         for(j in 1:amount) {
+             fansCount[j] <- sample(20:40, 1)
+         }
+         fansN[i] <- sum(fansCount)
+     }
+     print(mean(fansN))
+ }
> f(16666)
[1] 149.8053
```

图 21: 24 题: 模拟事件—粉丝在 T=1 之前到达

5.25. (a) Write a program that uses the thinning algorithm to generate the first 10 time units of a nonhomogeneous Poisson process with intensity function.

$$\lambda(t) = 3 + \frac{4}{t+1}$$

(b) Give a way to improve upon the thinning algorithm for this example.
翻译: (a) 编写一个程序, 使用" 瘦身" 算法生成具有如上强度函数的前 10 个时间单位的非齐次泊松过程
(b) 给出一种改进本例" 瘦身" 算法的方法
解:(a) 题代码如下:

```
fa <- function(T) {
  h <- function(t) {
    return(3+4/(t+1))
  }
  #求解最大的λ值
  hMax <- optimize(h, lower = 0,
                   upper = T, maximum = TRUE)$objective
  g <- function(T) {
    #S数组用来存储每次事件发生的时间
    S <- vector(length = 100)
    t = 0
    #C为事件发生次数
    C = 0
    for(i in 1:100000) {
      Df = runif(1)
      x = -log(Df)   #生成范围在(0,+∞)的随机数
      t = t+x/hMax
      #若时间累加大于既定时间,则退出循环
      if(t > T) {
        break
      }
      R <- runif(1)
      if(R <= h(t)/hMax){
        C = C+1
```

```
            S[C] = t
          }
        }
        #循环结束后返回事件发生次数
        re <- list(count = C)
        return(re)
      }
      n <- 10000
      resultCount <- vector(length = n)
      for(i in 1:n) {
        re <- g(T)
        resultCount[i] <- re$count
      }
      print(mean(resultCount))
      #输出非齐次泊松过程的次数分布图
      hist(resultCount, freq = FALSE,
           col = "yellow", main = "次数分布图",
           xlab = "次数",  ylab = "频率",
           breaks = 30)
}
par(mfrow = c(1,1))
#计算运行时间
ptmf <- proc.time()
fa(10)
print(proc.time() - ptmf)
```

(a) 题运行结果如下:

```
> ptmf <- proc.time()
> fa(10)
[1] 39.5825
> print(proc.time() - ptmf)
用户 系统 流逝
3.22 0.01 3.25
```

图 22: 25(a): 瘦身法模拟非齐次泊松过程—运行结果与运行时间

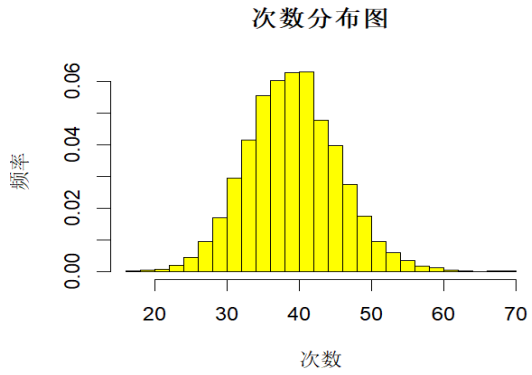

图 23: 25(a): 瘦身法模拟非齐次泊松过程—次数分布图

解:(b) 题代码如下:

```
#参数T为时间，参数k为分段区间个数
fb <- function(T,k) {
  h <- function(t) {
    return(3+4/(t+1))
  }
  #变量r用来存储每个区间的右边界
  #其实为保证随机性，区间分段应是任意的，但这里采用均匀分段
  r <- seq(0,T,len=k+1)
  #求解每个分段区间内的λ的最大值
  hMax <- vector(length = k)
  for(i in 1:k) {
    hMax[i] <- optimize(h, lower = r[i],
                upper = r[i+1],
                maximum = TRUE)$objective
  }
  r <- r[2:length(r)]
  g <- function(T) {
    #S数组用来存储每次事件发生的时间
    S <- vector(length = 10000)
    t = 0
    #C为事件发生次数
```

```
  C = 0
  #变量J用来记录当前事件发生在哪一个区间内
  J = 1
  for(i in 1:length(S)) {
    Df = runif(1)
    x = -log(Df)/hMax[J]
    #当累加时间大于某分区间的右边界时,J应当+1
    while(t+x > r[J] && J!=k+1) {
      x = (x-r[J]+t) * hMax[J]  / hMax[J+1]
      t = r[J]
      J = J + 1
    }
    #如果区间越过规定截止时间,则退出整个循环
    if(J==k+1) {
      break
    }
    t = t+x
    R <- runif(1)
    if(R <= h(t)/hMax[J]) {
      C = C+1
      S[C] = t
    }
  }
  #整个循环结束后返回事件发生次数
  re <- list(count = C)
  return(re)
}
n <- 10000
resultCount <- vector(length = n)
for(i in 1:n) {
  re <- g(T)
  resultCount[i] <- re$count
}
```

```
    print(mean(resultCount))
    #输出非齐次泊松过程的次数分布图
    hist(resultCount, freq = FALSE,
         col = "yellow", main = "次数分布图",
         xlab = "次数", ylab = "频率",
         breaks = 30)
}
ptmf <- proc.time()
fb(10,25)
print(proc.time() - ptmf)
```

(b) 题运行结果如下:

```
> ptmf <- proc.time()
> fb(10,15)
[1] 39.5762
> print(proc.time() - ptmf)
用户  系统  流逝
2.50 0.01 2.52
```

图 24: 25(b): 分段瘦身法模拟非齐次泊松过程—运行结果与运行时间

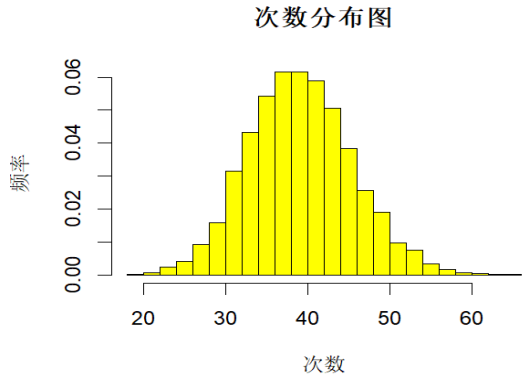

图 25: 25(b): 分段瘦身法模拟非齐次泊松过程—次数分布图

注意: 两种算法的模拟次数必须一致, 否则没有比较意义
且从统计的角度看, 各方法只模拟一次就说哪个算法效率高是不太靠谱的, 因此可以多模拟几次, 此处就不展示了

5.26.Give an efficient algorithm to generate the first 10 times units of a nonhomogeneous Poisson process having intensity function

$$\lambda(t) = \begin{cases} \frac{t}{5}, & 0 < t < 5 \\ 1 + 5(t-5), & 5 < t < 10 \end{cases} \tag{3}$$

翻译: 给出一种有效的算法来生成具有如上强度函数的前 10 个单位的非齐次泊松过程

解:26 题和 25 题思路一模一样, 分别用瘦身法和分段瘦身法模拟, 并比较哪一种算法效率更高

(法一) 瘦身法代码如下:

```
fa <- function(T) {
  h <- function(t) {
    if(t < 5) {
      return(t/5)
    } else if(t < 10){
      return(1+5*(t-5))
    }
  }
  #求解最大的λ值
  hMax <- optimize(h, lower = 0,
          upper = T, maximum = TRUE)$objective
  g <- function(T) {
    S <- vector(length = 100)
    t = 0
    C = 0
    for(i in 1:6666666) {
      Df = runif(1)
      x = -log(Df)   #生成范围在(0,+∞)的随机数
      t = t+x/hMax
      if(t > T) {
        break
      }
      R <- runif(1)
```

```
        if(R <= h(t)/hMax){
          C = C+1
          S[C] = t
        }
      }
      re <- list(count = C)
      return(re)
    }
    n <- 10000
    resultCount <- vector(length = n)
    for(i in 1:n) {
      re <- g(T)
      resultCount[i] <- re$count
    }
    print(mean(resultCount))
    #输出非齐次泊松过程的次数分布图
    hist(resultCount, freq = FALSE,
         col = "yellow", main = "次数分布图",
         xlab = "次数",  ylab = "频率",
         breaks = 30)
  }
  ptmf <- proc.time()
  fa(10)
  print(proc.time() - ptmf)
```

(法二) 分段瘦身法代码如下:

```
  #参数T为时间,参数k为分段区间个数
  fb <- function(T,k) {
    h <- function(t) {
      if(t < 5) {
        return(t/5)
      } else if(t < 10){
        return(1+5*(t-5))
      }
```

```
}
r <- seq(0,T,len=k+1)
#求解每个分段区间内的λ的最大值
hMax <- vector(length = k)
for(i in 1:k) {
  hMax[i] <- optimize(h, lower = r[i],
        upper = r[i+1], maximum = TRUE)$objective
}
r <- r[2:length(r)]
g <- function(T) {
  S <- vector(length = 10000)
  t = 0
  C = 0
  J = 1
  for(i in 1:length(S)) {
    Df = runif(1)
    x = -log(Df)/hMax[J]
    while(t+x > r[J] && J!=k+1) {
      x = (x-r[J]+t) * hMax[J]  / hMax[J+1]
      t = r[J]
      J = J + 1
    }
    if(J==k+1) {
      break
    }
    t = t+x
    R <- runif(1)
    if(R <= h(t)/hMax[J]) {
      C = C+1
      S[C] = t
    }
  }
  re <- list(count = C)
```

```
    return(re)
  }
  n <- 10000
  resultCount <- vector(length = n)
  for(i in 1:n) {
    re <- g(T)
    resultCount[i] <- re$count
  }
  print(mean(resultCount))
  #输出非齐次泊松过程的次数分布图
  hist(resultCount, freq = FALSE,
       col = "yellow", main = "次数分布图",
       xlab = "次数", ylab = "频率",
       breaks = 30)
}
ptmf <- proc.time()
fb(10,30)
print(proc.time() - ptmf)
```

同样的, 可以多模拟几次增强说服力, 这里就不再展示了
运行结果如下:

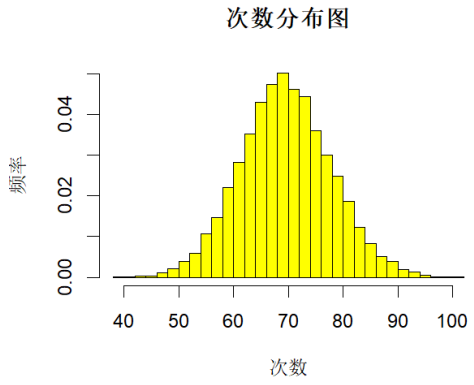

图 27: 题 26: 瘦身法次数分布图

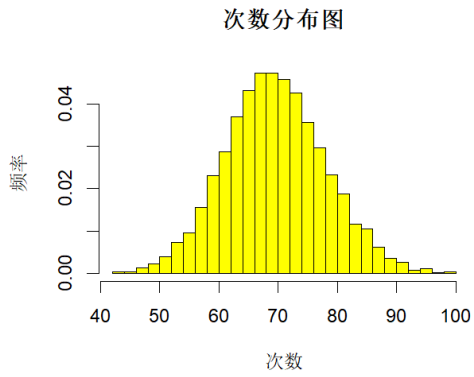

图 28: 题 26: 分段瘦身法次数分布图

```
> ptmf <- proc.time()
> fa(10)
[1] 70.0366
> print(proc.time() - ptmf)
 用户   系统   流逝
11.63  0.02 11.64
```

图 29: 题 26: 瘦身法运行时间

```
> ptmf <- proc.time()
> fb(10,30)
[1] 69.9884
> print(proc.time() - ptmf)
用户 系统 流逝
4.23 0.00 4.27
```

图 30: 题 26: 分段瘦身法运行时间

结论: 显然, 分段瘦身法模型效率更高

注意: 两种算法的模拟次数必须一致, 否则没有比较意义