# 統計模擬 作業二

109354003 統碩一 吳書恆

109354027 統碩一 蔡海蓮

4/8/2021

1. (a) Write your own R programs to perform Gap test and Permutation test. Then use this program to test if the uniform random numbers generated from Minitab (or SAS, SPSS, Excel) and R are independent.

   (b) Write a small computer program to perform Up-and-down test. Then use this program and uniform random numbers generated from R to check if the mean and variance for the number of runs derived by Levene and Wolfowitz (1944) are valid.

(a) 題使用 Excel 和 R 各生成 10000 個介於 $(0, 1)$ 的隨機亂數，分別檢定結果都說明亂數之間互相獨立。

| p-value | Excel | R |
|---|---|---|
| Gap test | 0.4547 | 0.6943 |
| Permutation test | 0.1968 | 0.2227 |

自編 Gap test 函數如下，

```
> gap.test <- function(data, a, b){
+   data <- data/max(data)
+   n <- length(data)
+   x <- c(1:n) * (a < data & data < b)
+   x1 <- x[x>0]
+   y <- x1[-1]-x1[-length(x1)]-1
+   t.y <- table(y)
+   e <- (b-a) * (1-(b-a))^(c(1:dim(t.y))-1) * sum(t.y)
+   e.p <- e / sum(e)
+   c.t <- chisq.test(t.y, p = e.p)
+   return(list("gaps" = t.y, "expected count" = round(e), "chisq.test"
= c.t))
+ }
```

自編 Permutation test 函數如下，

```
> permutation.test <- function(data){
+   cond <- length(data) %% 3
+   if(cond == 0){
+     x1 <- matrix(data, ncol = length(data) %/% 3, byrow = F)
+   }else{
+     x1 <- matrix(data[-c(1:cond)], ncol = length(data) %/% 3, byrow =
```

```
F)
+   }
+   y1 <- apply(x1, 2, rank)
+   y2 <- y1[1,]*100 + y1[2,]*10 + y1[3,]
+   c.t <- chisq.test(table(y2))
+   return(c.t)
+ }
```

(b) 本題生成 10000 個介於 $(0, 1)$ 的隨機亂數，並執行 1000 次，發現有將近 95% 左右的結果是不拒絕虛無假設，說明我們有足夠證據說明 Levene and Wolfowitz 與我們亂數進行 Up-and-Down test 的平均數與變異數相近。

```
> set.seed(2)
> k <- updown.test(10000, 1000)
Numbers of not reject (alpha=0.05):  946
Numbers of reject (alpha=0.05):  54
Proportion of not reject:  0.946
```

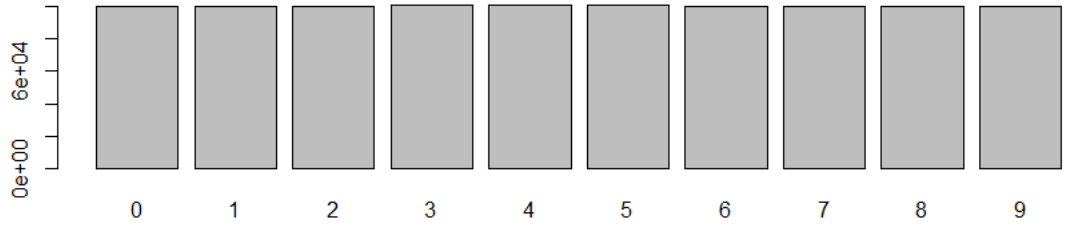
檢定 Up-and-Down test 是否符合 Levene and Wolfowitz 的平均與變異數如下，

```
> updown.test <- function(num,runs){
+   k.vector <- NULL
+   n.r <- 0
+   r <- 0
+   for(i in 1:runs){
+     x <- runif(num)
+     x1 <- (x[-1]>x[-num])
+     x2 <- sum((x1[-1] != x1[-(num-1)]))
+     z <- (x2-(2*num-1)/3)/sqrt((16*num-29)/90)
+     k <- pnorm(z)
+     k.vector <- c(k.vector,k)
+     if(k > 0.025 & k < 0.975){
+       n.r <- n.r + 1
+     }else{
+       r <- r + 1
+     }
+   }
+   cat(paste("Numbers of not reject (alpha=0.05): ", n.r),
+       paste("Numbers of reject (alpha=0.05): ", r),
+       paste("Proportion of not reject: ", n.r/runs),
+       sep = "\n")
+   return(k.vector)
+ }
```

2. (a) Use the search engine to download the first one million digits of pi (for example, http://www.piday.org/million.php), and check via graphic tools if the numbers violate the assumption of random numbers.

(b) Apply the appropriate tools to test if the random numbers from (a) satisfy the assumption of random numbers.

(a) 將 pi 數值存成文字檔，並用 `readLines()`指令去讀取，最後在用 `strsplit()`拆解字串即可得到所有小數點後的數值。0 到 9 數值長條圖如下，發現趨近於均勻分配。




(b) 進一步做統計檢定。卡方檢定結果（p-value＝0.7879）說明分布屬於均勻分布，Gap test 結果說明（p-value＝0.5326）分布是獨立的。(Warning 可忽視)

```
> chisq.test(table(pi.digit))

    Chi-squared test for given probabilities

data:  table(pi.digit)
X-squared = 5.5091, df = 9, p-value = 0.7879

> gap.test(pi.digit, .2, .8)
$chisq.test

    Chi-squared test for given probabilities

data:  t.y
X-squared = 12.923, df = 14, p-value = 0.5326

Warning message:
In chisq.test(t.y, p = e.p) : Chi-squared approximation may be incorrect
```

3. The following table shows the winning numbers of first 20 Taiwan Lottery (starting in 2002), which picks 6 numbers from 42 balls plus a "Power Ball." Choose your tools to check whether these winning numbers are random.

| Date | Winning Numbers | | | | | | Power Ball | Date | Winning Numbers | | | | | | Power Ball |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0329 | 22 | 31 | 34 | 25 | 21 | 19 | 13 | 0222 | 32 | 10 | 15 | 02 | 30 | 23 | 36 |
| 0326 | 05 | 18 | 25 | 26 | 35 | 42 | 29 | 0219 | 24 | 20 | 36 | 19 | 07 | 12 | 26 |
| 0321 | 32 | 21 | 09 | 27 | 31 | 06 | 2 | 0215 | 01 | 06 | 07 | 12 | 42 | 20 | 35 |
| 0319 | 05 | 25 | 02 | 16 | 32 | 09 | 7 | 0212 | 25 | 39 | 20 | 38 | 29 | 37 | 28 |
| 0315 | 15 | 29 | 05 | 36 | 13 | 10 | 1 | 0208 | 26 | 02 | 15 | 29 | 04 | 33 | 39 |
| 0312 | 36 | 16 | 12 | 26 | 08 | 34 | 5 | 0205 | 17 | 39 | 03 | 15 | 11 | 01 | 34 |
| 0308 | 04 | 40 | 27 | 21 | 14 | 05 | 12 | 0201 | 13 | 39 | 28 | 30 | 25 | 29 | 21 |
| 0305 | 29 | 04 | 10 | 23 | 39 | 14 | 36 | 0129 | 07 | 09 | 29 | 34 | 39 | 36 | 16 |
| 0301 | 30 | 12 | 40 | 32 | 35 | 20 | 34 | 0125 | 28 | 31 | 16 | 35 | 06 | 30 | 2 |
| 0226 | 40 | 06 | 20 | 29 | 38 | 35 | 41 | 0122 | 10 | 32 | 13 | 04 | 09 | 33 | 37 |

將所有號碼（不包含日期）存成向量，並畫直方圖，發現分布似乎有點不均勻，進一步檢定的結果發現，當區分成 10 組，卡方檢定結果（p-value＝0.2036）不拒絕虛無假設，說明此分布屬於均勻分布。Gap test 結果說明（p-value＝0.9297）分布是獨立的。

```
> table(v)
v
 0  1  2  3  4  5  6  7  8  9 10
 9 17 10 15  8 12 12 18 17 16  6
> chisq.test(table(v))

    Chi-squared test for given probabilities

data:  table(v)
X-squared = 13.371, df = 10, p-value = 0.2036
```

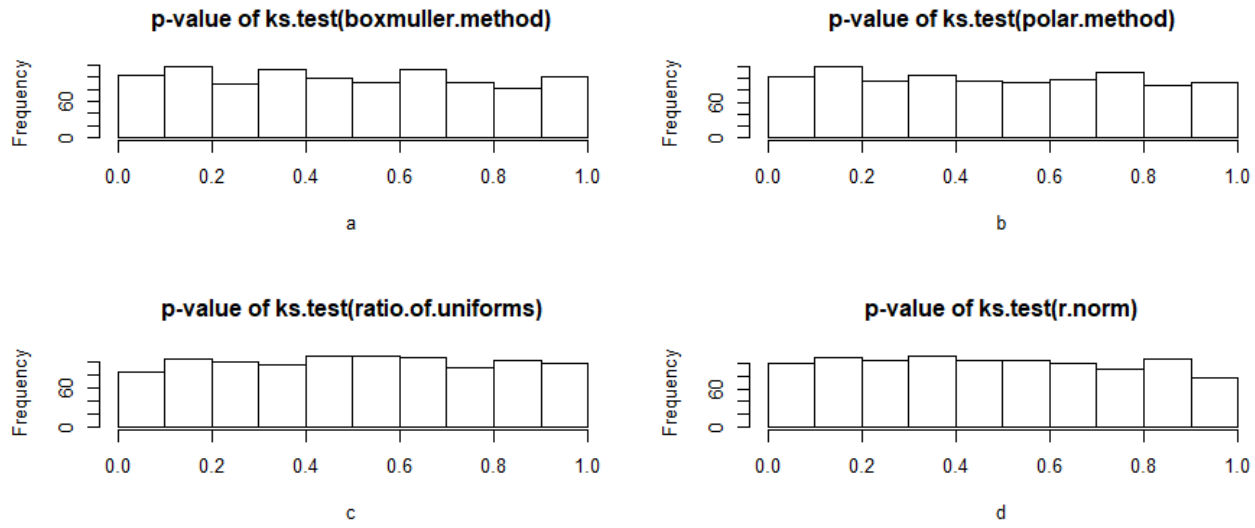
4. (a) Test the generation methods of normal distribution introduced in class, i.e., Box-Muller, Polar, Ratio-of-uniform, and also the random number generators from R. Based on your simulation results, choose the "best" generator.

   (b) In the class we mentioned it is found by several researchers that

   $a$ (multiplier) = 131
   $c$ (increment) = 0
   $m$ (modulus) = 2^35

   would have X ∈ (–3.3,3.6), if plugging congruential generators into the Box-Muller method. Verify if you would have similar results.

(a) 此題需要寫出四個函數去產生常態分配的亂數，由於冗長就不在這列出（可看附錄的程式碼）。四個函數分別執行 1000 次的 10000 個常態分配亂數，並在各自的每一次執行中檢定常態性（`ks.test()`），四個函數分別得到 1000 個 p 值，直方圖結果如下。




再分別檢定以上的分布是否均勻分布，得到結果整理表格，發現 ratio.of.U 最好（這跟 set.seed 有關）。

| | Box.Muller | Polar | ratio.of.U | rnorm |
|---|---|---|---|---|
| p-value | 0.2653 | 0.5121 | 0.6577 | 0.1598 |

(b) 此題發現用 Box.Muller 隨機生成的變數（x, y）用題目給定的同餘法條件，並不會有任何值落在（-3.3, 3.6）之間。

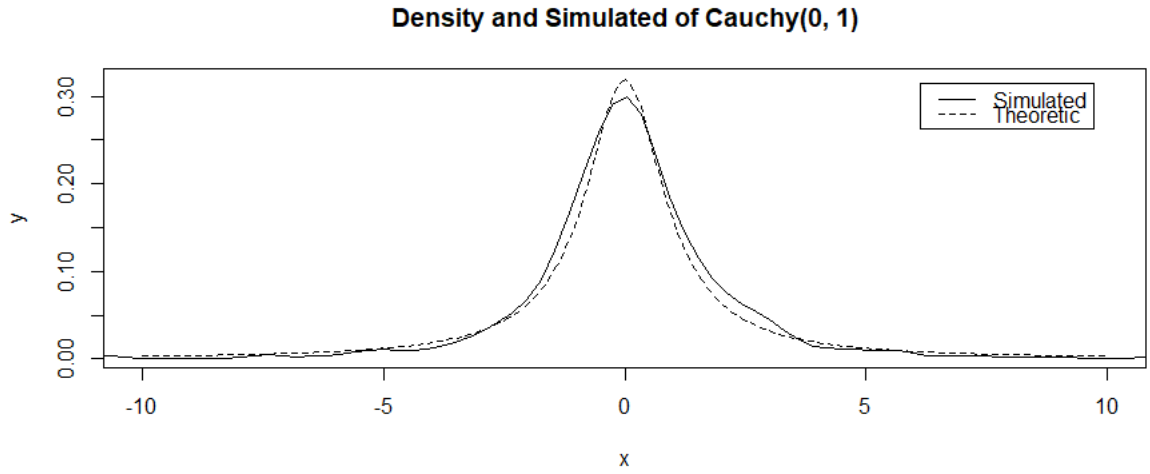
```
> for(i in 1:100){
+   x <- (131*(x)) %% (2^35)
+   y <- (131*(y)) %% (2^35)
+   xy.vector <- c(xy.vector,x,y)
+ }
> sum((-3.3 < xy.vector & xy.vector < 3.6))/length(xy.vector)
[1] 0
```

5. (a) Using Rejection Method (or Inversion) described in the class, to generate random numbers from Cauchy distribution. (Note: You need to check the goodness-of-fit and independence.)
(b) Compare with the methods of "Ratio of Uniform" (in-class) and "Ratio of Normal" and give your suggestion (such as choosing the "best" method) for creating random numbers from Cauchy distribution.

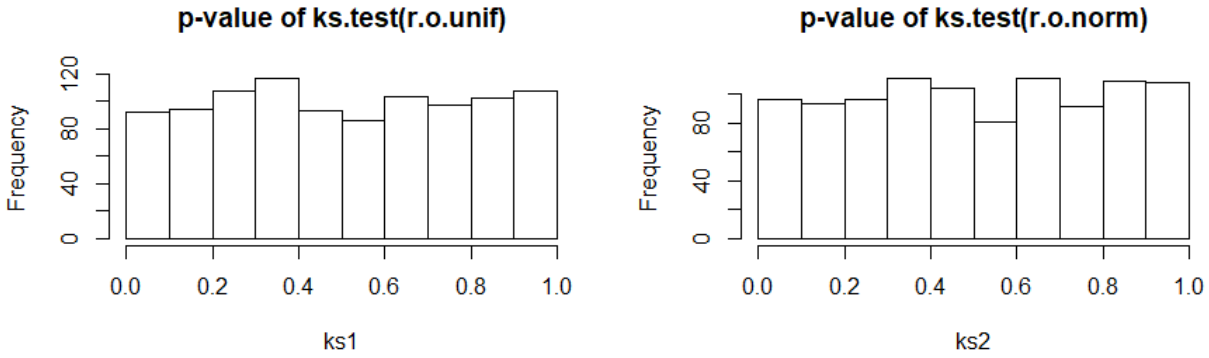
(a) 假設 f(x) ~ Cauchy（0, 1）且根據 inverse F(x)與均勻分布亂數去逆推柯西分布。至於拒絕條件的設定，我們找了與柯西分布相近的 t 分布（自由度 = 0.5）作爲參考分布，可以得到 c = max(f(x)/g(x)) = 1.345。程式碼如下。

```
> x.p <- NULL
> t <- 0
> set.seed(2)
> repeat{
+   u1 <- runif(1)
+   u2 <- runif(1)
+   x <- tan(pi*(u1 - 1/2))
+   k <- dcauchy(x)/dt(x, 0.5)/c
+   if(u2 <= k){
+     x.p <- c(x.p, x)
+   }
+   t <- t + 1
+   if (length(x.p) == 1000) break
+ }
>
> (acc.rate <- 1000/t)
[1] 0.8591065
```

執行 1000 次的亂數生成，發現接受率高達 0.86，說明 inverse F(x) 亂數生成的結果有很高的比例是接受的。透過以下圖形發現分布與理論分布相近，ks 檢定也不拒絕虛無假設 (p-value = 0.0563)。




(b) 此題 Ratio of Uniform 是採用老師講義的做法；但 Ratio of Normal 則沒有設定拒絕條件，可以預見前者效果會較好，我們運行 1000 次 1000 個隨機生成的亂數，分別得到 1000 個 p-value，以下是長條圖，ks 檢定的結果發現 Ratio of Uniform 較好。




```
> ks.test(ks1, "punif")

    One-sample Kolmogorov-Smirnov test

data:  ks1
D = 0.019807, p-value = 0.8276
alternative hypothesis: two-sided

> ks.test(ks2, "punif")

    One-sample Kolmogorov-Smirnov test

data:  ks2
D = 0.025395, p-value = 0.5392
alternative hypothesis: two-sided
```

6. Write a program to check the Table method and the Alias method for generating r.v. from B(3,1/3). Also, compare the speed of generation for the two methods.

兩個方式的程式法如下，

```
> #Table method
> table.method <- function(runs){
+   x.vector <- NULL
+   for(i in 1:runs){
+     u <- runif(1)
+     x1 <- c(rep(0,2),rep(1,4),rep(2,2),rep(3,0))
+     x2 <- c(rep(0,9),rep(1,4),rep(2,2),rep(3,3))
+     x3 <- c(rep(0,6),rep(1,4),rep(2,2),rep(3,7))
+     if(u < 0.8){
+       j <- floor(10*u)+1
+       x <- x1[j]
+     }else if(u < 0.98){
+       j <- floor(100*u)-80+1
+       x <- x2[j]
+     }else{
+       j <- floor(1000*u)-980+1
+       x <- x3[j]
+     }
+     x.vector <- c(x.vector, x)
+   }
+   return(table(x.vector))
+ }
+
> #the Alias method
> alias.run <- function(n) {
+   temp <- NULL
+   for (i in 1:n) {
+     x <- floor(4*runif(1))
+     x1 <- (runif(1) < (27-4)/27)
+     x2 <- (runif(1) < (27-24)/27)
+     x3 <- (runif(1) < (27-25)/27)
+     xx <- c(0,1,2,2)*c(0,x3,x2,x1)
+     y <- x - xx[c(x+1)]
+     temp <- c(temp,y)
+   }
+   cat(x1,x2,x3,"\n")
+   return(temp)
+ }
```

透過 `proc.time()` 發現 Alias method 較 Table method 快些，不過仍差不多，論生成 B(3,1/3) 結果，發現 Alias method 的絕對離差較小。綜合上述，Alias method 表現較佳。

```
> sum(abs(p-c(8/27, 12/27, 6/27, 1/27)*10000))
[1] 183.8889
> sum(abs(table(k)-c(8/27, 12/27, 6/27, 1/27)*10000))
[1] 79.11111
```

Code:

```
##-HW2-################
#1################################
#(a)
gap.test <- function(data, a, b){
  data <- data/max(data)
  n <- length(data)
  x <- c(1:n) * (a < data & data < b)
  x1 <- x[x > 0]
  y <- x1[-1] - x1[-length(x1)]-1
  t.y <- table(y)
  e <- (b-a) * (1-(b-a))^(c(1:dim(t.y))-1) * sum(t.y)
  e.p <- e / sum(e)
  c.t <- chisq.test(t.y, p = e.p)
  return(list("gaps" = t.y, "expected count" = round(e), "chisq.test" =
c.t))
}

permutation.test <- function(data){
  cond <- length(data) %% 3
  if(cond == 0){
    x1 <- matrix(data, ncol = length(data) %/% 3, byrow = F)
  }else{
    x1 <- matrix(data[-c(1:cond)], ncol = length(data) %/% 3, byrow = F)
  }
  y1 <- apply(x1, 2, rank)
  y2 <- y1[1,]*100 + y1[2,]*10 + y1[3,]
  c.t <- chisq.test(table(y2))
  return(c.t)
}

rn.excel  <-  as.matrix(read.csv("StatisticalSimulation/rn10000.csv",
header = F))
set.seed(2)
rn.r <- runif(10000)
gap.test(rn.excel, 0.2, 0.7)
permutation.test(rn.excel)

gap.test(rn.r, 0.2, 0.7)
permutation.test(rn.r)


#(b)
updown.test <- function(num,runs){
  k.vector <- NULL
  n.r <- 0
  r <- 0
  for(i in 1:runs){
    x <- runif(num)
    x1 <- (x[-1]>x[-num])
    x2 <- sum((x1[-1] != x1[-(num-1)]))
    z <- (x2 - (2*num-1)/3) / sqrt((16*num-29)/90)
    k <- pnorm(z)
```

```
    k.vector <- c(k.vector,k)
    if(k > 0.025 & k < 0.975){
      n.r <- n.r + 1
    }else{
      r <- r + 1
    }
  }
  cat(paste("Numbers of not reject (alpha=0.05): ", n.r),
     paste("Numbers of reject (alpha=0.05): ", r),
     paste("Proportion of not reject: ", n.r/runs),
     sep = "\n")
  return(k.vector)
}

set.seed(2)
k <- updown.test(10000, 1000)


#2#################################
#(a)
pidata <- readLines("StatisticalSimulation/pi.txt")
pi.digit   <-   as.numeric(strsplit(as.character(pidata),   "")[[1]][-
c(1:2)])
barplot(table(pi.digit))
chisq.test(table(pi.digit))
gap.test(pi.digit, .2, .8)


#3#################################
q3 <- read.table("StatisticalSimulation/hw2_3.txt")
numbers <- as.vector(as.matrix(q3[, -c(1, 9)]))
hist(numbers, breaks = seq(0, 42, 4.2))

v <- floor(numbers * .25)
chisq.test(table(v))
gap.test(numbers, .2, .8)


#4#################################
#(a)
#Box-Muller
boxmuller.method <- function(runs){
  p.value <- NULL
  p1 <- 0
  p2 <- 0
  p3 <- 0
  for(i in 1:runs){
    u1 <- runif(10000)
    u2 <- runif(10000)
    theta <- 2 * pi * u1
    k <- -log(u2)
    r <- sqrt(2 * k)
    x <- r * cos(theta)
```

```
    y <- r * sin(theta)
    p <- ks.test(c(x,y), "pnorm")$p.value
    if(p <= 0.01){
      p1 <- p1 + 1
    }else if(p <= 0.05){
      p2 <- p2 + 1
    }else if(p <= 0.1){
      p3 <- p3 + 1
    }
    p.value <- c(p.value, p)
  }
  cat("p-value under 0.01 :",p1,"\n",
      "p-value under 0.05 :",p1 + p2,"\n",
      "p-value under 0.1 :",p1 + p2 + p3,"\n")
  return(p.value)
}

#Polar Method
polar.method <- function(runs){
  p.value <- NULL
  p1 <- 0
  p2 <- 0
  p3 <- 0
  for(i in 1:runs){
    v1 <- runif(10000, min = -1, max = 1)
    v2 <- runif(10000, min = -1, max = 1)
    w <- v1^2 + v2^2
    w1 <- which(w < 1)
    w2 <- cbind(v1, v2, w)
    w2 <- w2[c(w1),]
    c <- sqrt(-2 * log(w2[,3]) / w2[,3])
    x <- c * w2[,1]
    y <- c * w2[,2]
    p <- ks.test(c(x,y), "pnorm")$p.value
    if(p <= 0.01){
      p1 <- p1 + 1
    }else if(p <= 0.05){
      p2 <- p2 + 1
    }else if(p <= 0.1){
      p3 <- p3 + 1
    }
    p.value <- c(p.value, p)
  }
  cat("p-value under 0.01 :",p1,"\n",
      "p-value under 0.05 :",p1 + p2,"\n",
      "p-value under 0.1 :",p1 + p2 + p3,"\n")
  return(p.value)
}

#Ratio of uniforms
ratio.of.uniforms <- function(runs){
  p.value <- NULL
  p1 <- 0
```

```
  p2 <- 0
  p3 <- 0
  for(i in 1:runs){
    u1 <- runif(10000)
    u2 <- runif(10000)
    v <- sqrt(2/exp(1)) * (2*u2-1)
    x <- v / u1
    z <- x^2 / 4
    z1 <- which(z <= (0.259/u1) + 0.35 & z <= -log(u1))
    z2 <- cbind(x, z)
    z2 <- z2[c(z1),]
    p <- ks.test(z2[,1], "pnorm")$p.value
    if(p <= 0.01){
      p1 <- p1 + 1
    }else if(p <= 0.05){
      p2 <- p2 + 1
    }else if(p <= 0.1){
      p3 <- p3 + 1
    }
    p.value <- c(p.value, p)
  }
  cat("p-value under 0.01 :",p1,"\n",
      "p-value under 0.05 :",p1 + p2,"\n",
      "p-value under 0.1 :",p1 + p2 + p3,"\n")
  return(p.value)
}

#rnorm()
r.norm <- function(runs){
  p.value <- NULL
  p1 <- 0
  p2 <- 0
  p3 <- 0
  for(i in 1:runs){
    u <- rnorm(10000)
    p <- ks.test(u, "pnorm")$p.value
    if(p <= 0.01){
      p1 <- p1 + 1
    }else if(p <= 0.05){
      p2 <- p2 + 1
    }else if(p <= 0.1){
      p3 <- p3 + 1
    }
    p.value <- c(p.value, p)
  }
  cat("p-value under 0.01 :",p1,"\n",
      "p-value under 0.05 :",p1 + p2,"\n",
      "p-value under 0.1 :",p1 + p2 + p3,"\n")
  return(p.value)
}

set.seed(2)
a <- boxmuller.method(1000)
```

```
b <- polar.method(1000)
c <- ratio.of.uniforms(1000)
d <- r.norm(1000)
par(mfrow = c(2, 2))
hist(a, main = 'p-value of ks.test(boxmuller.method)')
hist(b, main = 'p-value of ks.test(polar.method)')
hist(c, main = 'p-value of ks.test(ratio.of.uniforms)')
hist(d, main = 'p-value of ks.test(r.norm)')
ks.test(a, "punif")
ks.test(b, "punif")
ks.test(c, "punif")
ks.test(d, "punif")

#4(b)
u <- runif(2)
theta <- 2 * pi * u[1]
k <- -log(u[2])
r <- sqrt(2 * k)
x <- r * cos(theta)
y <- r * sin(theta)
xy.vector <- NULL
for(i in 1:100){
  x <- (131*(x)) %% (2^35)
  y <- (131*(y)) %% (2^35)
  xy.vector <- c(xy.vector,x,y)
}
sum((-3.3 < xy.vector & xy.vector < 3.6))/length(xy.vector)


#5#################################
#(a)
x <- seq(-100, 100, 0.1)
c <- max(dcauchy(x) / dt(x, 0.5))

x.p <- NULL
t <- 0
set.seed(2)
repeat{
  u1 <- runif(1)
  u2 <- runif(1)
  x <- tan(pi*(u1 - 1/2))
  k <- dcauchy(x) / dt(x, 0.5) / c
  if(u2 <= k){
    x.p <- c(x.p, x)
  }
  t <- t + 1
  if (length(x.p) == 1000) break
}

(acc.rate <- 1000 / t)
x <- seq(-10, 10, 0.1)
y <- dcauchy(x)
plot(x, y, type="l", lty = 2, main = "Density and Simulated of Cauchy(0,
```

```
1)")
lines(density(x.p), lty = 1)
legend("topright", inset=.05,
     c("Simulated", "Theoretic"), lty = c(1, 2))

ks.test(x.p, "pcauchy")
permutation.test(x.p)

#(b)
leng1 <- NULL
ks1 <- NULL
ks2 <- NULL
set.seed(2)
for(i in 1:1000){
  x1<- NULL
  for(j in 1:1000){
    u1 <- runif(1)
    u2 <- runif(1)
    v <- 2*u2 - 1
    if (u1^2 + v^2 < 1){
      x <- v/u1
      x1 <- c(x1, x)
      }else{
        next
      }
    }

  x2 <- NULL
  for(j in 1:1000){
    u1 <- rnorm(1)
    u2 <- rnorm(1)
    x <- u1/u2
    x2 <- c(x2, x)
    }

  leng1 <- c(leng1, length(x1)/10000)
  ks1 <- c(ks1, ks.test(x1, "pcauchy")$p.value)
  ks2 <- c(ks2, ks.test(x2, "pcauchy")$p.value)
}

par(mfrow = c(1, 2))
hist(ks1, main = 'p-value of ks.test(r.o.unif)')
hist(ks2, main = 'p-value of ks.test(r.o.norm)')
ks.test(ks1, "punif")
ks.test(ks2, "punif")
mean(leng1)



#6################################
#Table method
table.method <- function(runs){
  x.vector <- NULL
```

```
  for(i in 1:runs){
    u <- runif(1)
    x1 <- c(rep(0,2),rep(1,4),rep(2,2),rep(3,0))
    x2 <- c(rep(0,9),rep(1,4),rep(2,2),rep(3,3))
    x3 <- c(rep(0,6),rep(1,4),rep(2,2),rep(3,7))
    if(u < 0.8){
      j <- floor(10*u) + 1
      x <- x1[j]
    }else if(u < 0.98){
      j <- floor(100*u) - 80 + 1
      x <- x2[j]
    }else{
      j <- floor(1000*u) - 980 + 1
      x <- x3[j]
    }
    x.vector <- c(x.vector, x)
  }
  return(table(x.vector))
}

#Alias method
alias.run <- function(n) {
  temp <- NULL
  for (i in 1:n) {
    x <- floor(4*runif(1))
    x1 <- (runif(1) < (27-4)/27)
    x2 <- (runif(1) < (27-24)/27)
    x3 <- (runif(1) < (27-25)/27)
    xx <- c(0,1,2,2)*c(0,x3,x2,x1)
    y <- x - xx[c(x+1)]
    temp <- c(temp,y)
  }
  cat(x1,x2,x3,"\n")
  return(temp)
}

set.seed(2)
p <- table.method(10000)
proc.time()
k <- alias.run(10000)
proc.time()
sum(abs(p-c(8/27, 12/27, 6/27, 1/27)*10000))
sum(abs(table(k)-c(8/27, 12/27, 6/27, 1/27)*10000))
```