統計模擬 作業一

109354003 統碩一 吳書恆

109354027 統碩一 蔡海蓮

3/24/2021

1. (a) Use the commands “rep” and “seq” to create the vector:

   0 0 0 0 0 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3 4 4 4 4 4

   (b) Similar to (a), create the following vector:

   1 2 3 4 5 2 3 4 5 6 3 4 5 6 7 4 5 6 7 8 5 6 7 8 9

   (c) Use “rep” and “seq” to create the following vector:

   red, yellow, blue, yellow, blue, green

   blue, green, magenta, green, magenta, cyan

此題只要熟悉 `rep` 和 `sep` 指令即可。(b) 和 (c) 其實是一樣的邏輯。

```
> rep(seq(0,4), each = 5)

> seq(1:5) + rep(0:4, each = 5)

> x <- c("red","yellow","blue","green","magenta","cyan")
> i <- seq(1:3) + rep(0:3, each = 3)
> x[i]
```

2. (a) Write a function to calculate the minimum distance between any two points in the region (0, 1)×(0, 1). Randomly generate 20 points from (0, 1)×(0, 1), and then use the function you wrote to calculate the minimum distance.

   (b) Use the function “plot” to create scatter plot for the data in (a), restricting the domain in (0, 1)×(0, 1). Also, divide the region into 4 equal-area sub-regions and plot the 20 points according to which region they lie.

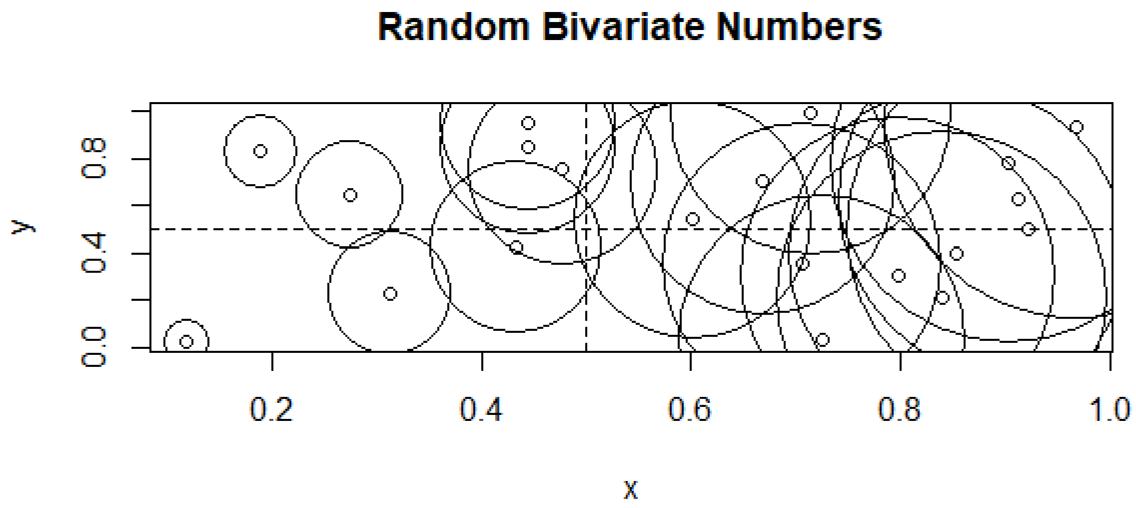
   (c) Explore the function “symbols” and explain what it does. Experiment this function and output the result.

(a) 題只需求 20 個距離中最小的距離，因此透過 `spatstat` 套件的 `nndist()`，最後取 `min`。

```
> mindist <- function(x, y){
+  min(nndist(x, y))
+ }
```

(b) 圖按照老師課堂作法很快就會做出來。(c) 題需要指定 `symbols` 要參照的變數，這變簡單用 `x` 做實驗。兩題結果疊合如下。

```
> plot(x, y, main = "Random Bivariate Numbers")
> abline(h = 0.5, v = 0.5, lty=2)
>
> #2(c)
> symbols(x, y, circles = x, add = T)
```



3. The greatest common divisor of two numbers can be computed via: (Verify!)

   gcd = function(a,b)

   { if (b==0) a else gcd(b,a%%b) }

   Use a similar idea of the function "*gcd*" to create a function "*lcm*" for computing the least common multiplier of two numbers.

   (Bonus: Modify these functions to more than two numbers.)

以下式子成立，

任一整數＝最大公因數 × 最小公倍數

因此，可以簡單寫個函數，

```
> lcm = function(c,d){
+   return(c * d / gcd(c,d))
+ }
```

4. (a) Write a computer program using the Mid-Square Method using 6 digits to generate 10,000 random numbers ranging over [0, 999999]. Use the Kolmogorov-Smirnov Goodness-of-fit test to see if the random numbers that you create are uniformly distributed. (Note: You must notify the initial seed number used, and you may adapt 0.05 as the α value. Also, you may find warning messages for conducting the Goodness-of-fit test, and comment on the Goodness-of-fit test.)

(b) Similar to the above, but consider $X_{i+1} = 69{,}069\, X_i \pmod{2^{32}}$, i.e., the generator used by Vax before 1993. Use both the χ^2 and Kolmogorov-Smirnov Goodness-of-fit tests to check if the data are from U(0,1) distribution.

(c) Consider the combination of 3 multiplicative congruential generators, i.e.,

$$u_i = \frac{x_i}{30269} + \frac{y_i}{30307} + \frac{z_i}{30323} \pmod 1$$

with $x_i = 171\, x_{i-1} \pmod{30269}$, $y_i = 172\, y_{i-1} \pmod{30307}$, $z_i = 170\, z_{i-1} \pmod{30323}$.

Compare the result with those in (a) & (b), and discuss your findings.

(a) Mid-Square Method 方法需要處理取中間位數的問題，透過 %/%（取商數）來消去後三位數值，再用%%（取餘數）取剩於前數值，且這樣的取法能確保數值會介於 [0, 999999] 之間，最後再除以 10^6 使範圍縮減至 [0, 1]。

```
> midsqur <- function(seed,times){
+   numvector <- NULL
+   for(i in 1:times){
+     num <- seed * seed
+     seed <- (num%/%1000) %% 1000000
+     numvector <- c(numvector, seed)
+   }
+   numvector <- (numvector / 10^6)
+   return(numvector)
+ }
```

在用 ks 檢定檢驗隨機變數是否 uniform 時出現以下警告:

```
> ks.test(x1, y = "punif")

    One-sample Kolmogorov-Smirnov test
```

```
data:  x1
D = 0.1969, p-value < 2.2e-16
alternative hypothesis: two-sided

Warning message:
In ks.test(x1, y = "punif") :
  ties should not be present for the Kolmogorov-Smirnov test
```

警告說明有數值出現多次重複，我們發現 0.201 數值出現了 2448 次，占總比例的 24%。

(b) 此題並沒有出現數值重複的狀況，從兩種方式檢驗結果發現沒有足夠證據說明亂數不服從均勻分布。

```
> v <- floor(x2 * 10)
> chisq.test(table(v))

	Chi-squared test for given probabilities

data:  table(v)
X-squared = 7.4, df = 9, p-value = 0.5955

> ks.test(x2, y = "punif")

	One-sample Kolmogorov-Smirnov test

data:  x2
D = 0.0099188, p-value = 0.2788
alternative hypothesis: two-sided
```

(c) 此題結果也是如 (b)。

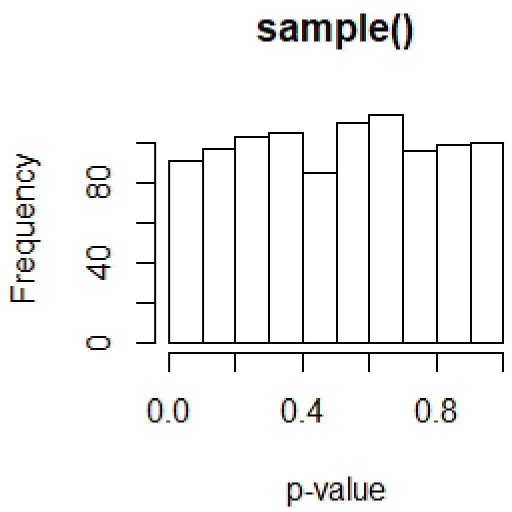
5. (a) In class, we often use simulation tools in R, e.g., “sample” or “ceiling(runif),” to generate random numbers from 1 to k, where k is a natural number. Using graphical tools (such as histogram) and statistical tests to check which one is a better tool in producing uniform numbers between 1 and k. (Hint: You may check if the size of k matters by, for example, assigning k a small and big value.)

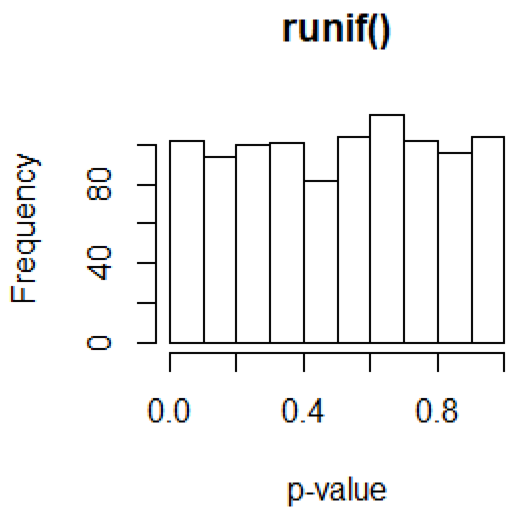
(b) In addition to $U_{n+1} = (\pi + U_n)^5 \ (\mathrm{mod}\ 1)$, we can use $\phi = \dfrac{1+\sqrt{5}}{2}$ (the golden ratio) or other irrational numbers to replace the value of π, to generate random numbers between 0 and 1. Using graphical tools (such as histogram) and statistical tests to check if π or ϕ has a better performance in producing uniform numbers between 0 and 1.

(a) 此題爲了比較這兩種方式，模擬了 1000 次的卡方檢定，每次都會生成 10000 個隨機數字。

```
> t1 <- NULL
> be <- 10000/15
> for (i in 1:1000) {
+   a1 <- sample(c(1:15), 10000, T)
+   a2 <- ceiling(15 * runif(10000))
+   b1 <- table(a1)
+   b2 <- table(a2)
+   c1 <- sum((b1-be)^2 / be)
+   c2 <- sum((b2-be)^2 / be)
+   d1 <- pchisq(c1, 14)
+   d2 <- pchisq(c2, 14)
+   t1 <- cbind(t1,c(d1,d2))
+ }
```

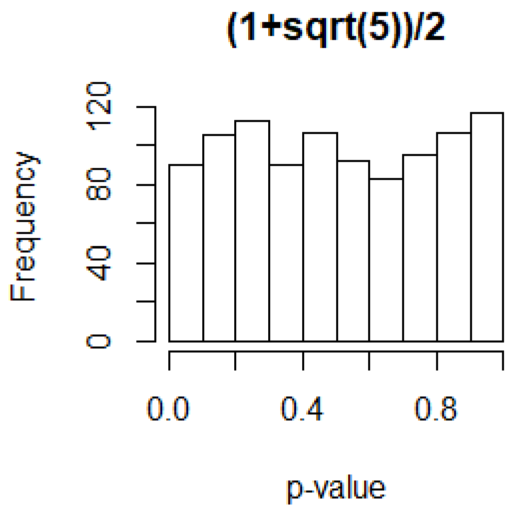
以下是兩種方式 p 值得直方圖。

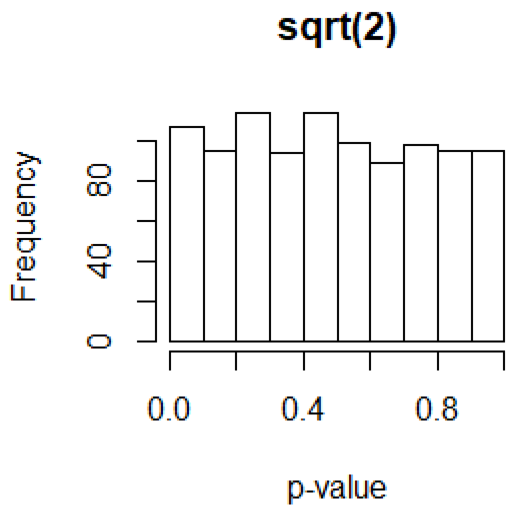




直觀看可能會覺得差不多，我們用卡方檢定比較哪個分布較接近均勻分布，發現 `runif()` 的 p 值會較大一些，說明 `runif()` 較好。

(b) 我們比較 $(1+\sqrt{5})/2$ 和 $\sqrt{2}$ 取代 π 的情形，比較的方式如(a)，不過每次的模擬我們只生成 1000 個亂數。用卡方檢定比較哪個分布較接近均勻分布，發現 $\sqrt{2}$ 的 p 值會較大一些，說明 $\sqrt{2}$ 較好。



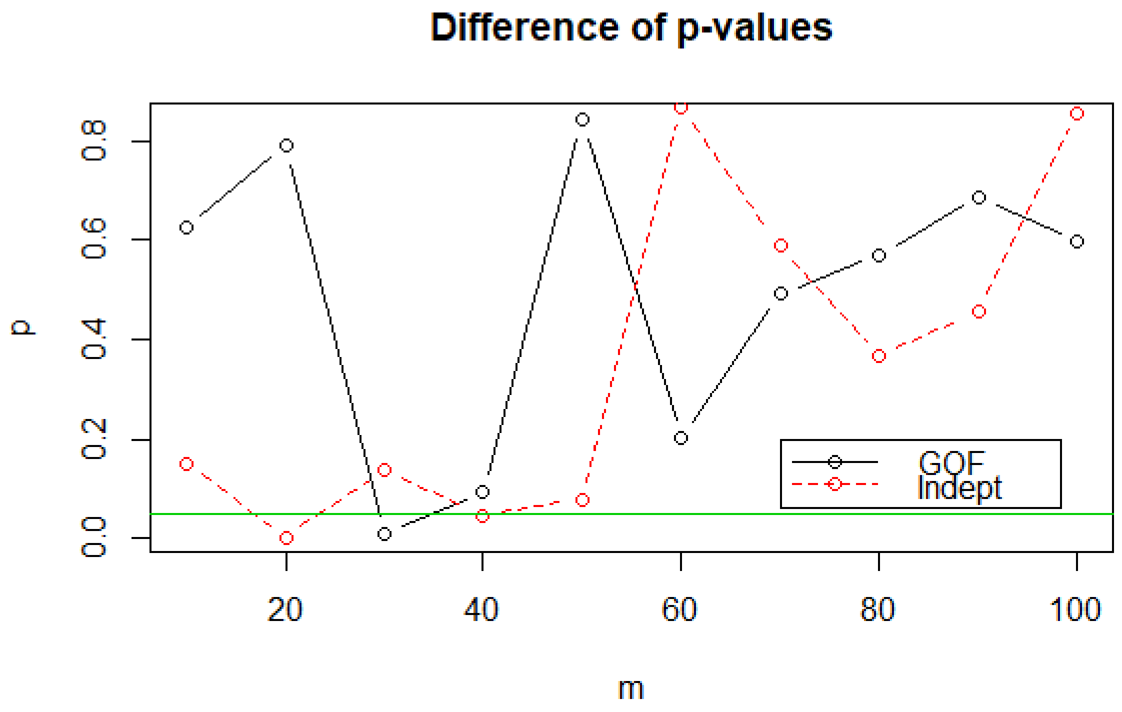


6. (a) Fibonacci numbers, defined as $X_{n+1} = X_n + X_{n-m} \pmod 1$, is another way of generating random numbers. The usual setting is letting $m = 1$ and see if $(X_n)'s$ are a sequence of random numbers from *U(0,1)*. However, $x_n < x_{n+1} < x_{n-1}$ and $x_{n-1} < x_{n+1} < x_n$ never appear under this setting. In general, the performances of Fibonacci numbers would be close to "random" as *m* increases. Write a program to generate Fibonacci numbers and test if they are "good" random numbers given varies choices of *m*. (Note: You could simulate 10,000 random numbers, and use goodness-of-tests & independence tests to evaluate Fibonacci numbers.)

Fibonacci 數列的生成，可以指定要前一筆與前 m 比的總和，且要生成小數所以取 mod1，

```
> fibonacci <- function(seed, n){
+   m <- length(seed) - 1
+   for (j in 1:n) {
+     x <- (seed[j] + seed[j+m]) %% 1
+     seed <- c(seed,x)
+   }
+   return(seed[-c(1:(m+1))])
+ }
```

爲了檢定亂數服從均勻分布且彼此之間獨立，採用卡方檢定與 permutation test，並指定不同的 m:{10, 20, ..., 100}，結果如下。



Code:

```
##-HW1-################
#1(a)
rep(seq(0,4), each = 5)

#1(b)
seq(1:5) + rep(0:4, each = 5)

#1(c)
x <- c("red","yellow","blue","green","magenta","cyan")
i <- seq(1:3) + rep(0:3, each = 3)
x[i]


#2(a)-################
library(spatstat)

mindist <- function(x, y){
 min(nndist(x, y))
}

x <- runif(20)
y <- runif(20)
mindist(x, y)
```

```
#2(b)
grid <- function(x){
  if(x[1] > 0.5 && x[2] > 0.5){index <- 1
  }else if(x[1] < 0.5 && x[2] > 0.5){index <- 2
  }else if(x[1] < 0.5 && x[2] < 0.5){index <- 3
  }else {index <- 4}
}
xy <- rbind(x,y)
index <- apply(xy, 2, grid)

plot(x, y, pch=index)
abline(h = 0.5, v = 0.5, lty=2)

#2(c)
symbols(x, y, circles = index, add = T)


#3-################
gcd = function(a,b){
  if (b==0) a else gcd(b, a %% b)
}

lcm = function(c,d){
  return(c * d / gcd(c,d))
}


#4(a)-################
midsqur <- function(seed,times){
  numvector <- NULL
  for(i in 1:times){
    num <- seed * seed
    seed <- (num%/%1000) %% 1000000
    numvector <- c(numvector, seed)
  }
  numvector <- (numvector / 10^6)
  return(numvector)
}

x <- ceiling(runif(1, 0, 999999))
x1 <- midsqur(x, 10000)
hist(x1)
ks.test(x1, y = "punif")

for(i in 1:length(x1)){
  y <- x1[i] - x1
```

```
  yc <- x1[which(y == 0)]
}
table(yc)


#4(b)
x1 <- 0.6
x2 <- 0
for (i in 1:10000){
  x1 <- (69069*x1) %% 2^32
  x2 <- c(x2, x1)
}
x2 <- x2[-1] / 2^32
hist(x2)

v <- floor(x2 * 10)
chisq.test(table(v))
ks.test(x2, y = "punif")


#4(c)
xi <- rnorm(1, 0, 1)
yi <- rnorm(1, 0, 1)
zi <- rnorm(1, 0, 1)
vector <- NULL

for (j in 1:10000) {
  xi <- (171*xi) %% 30269
  yi <- (172*yi) %% 30307
  zi <- (170*zi) %% 30323
  ui <- ((xi/30269) + (yi/30307) + (zi/30323)) %% 1
  vector <- c(vector, ui)
}

v <- floor(x2 * 10)
chisq.test(table(v))
ks.test(vector, y = "punif")


#5(a)-###############
t1 <- NULL
be <- 10000/15
for (i in 1:1000) {
  a1 <- sample(c(1:15), 10000, T)
  a2 <- ceiling(15 * runif(10000))
  b1 <- table(a1)
```

```
  b2 <- table(a2)
  c1 <- sum((b1-be)^2 / be)
  c2 <- sum((b2-be)^2 / be)
  d1 <- pchisq(c1, 14)
  d2 <- pchisq(c2, 14)
  t1 <- cbind(t1,c(d1,d2))
}
par(mfrow = c(1, 2))
hist(t1[1,], xlab = 'p-value', main = 'sample()')
hist(t1[2,], xlab = 'p-value', main = 'runif()')

v1 <- table(floor(t1[1,] * 10))
chisq.test(v1)
v2 <- table(floor(t1[2,] * 10))
chisq.test(v2)

#5(b)
casio <- function(seed, times){
  phi <- (1+sqrt(5)) / 2
  uvector <- NULL
  for(i in 1:times){
    seed <- ((phi+seed)^5) %% 1
    uvector <- c(uvector,seed)
  }
  return(uvector)
}

casio2 <- function(seed, times){
  uvector <- NULL
  for(i in 1:times){
    seed <- ((sqrt(2)+seed)^5) %% 1
    uvector <- c(uvector,seed)
  }
  return(uvector)
}

l <- runif(1, 0, 1)
k <- casio(l, 10000)
k2 <- casio2(l, 10000)
par(mfrow = c(1, 2))
hist(k)
hist(k2)

v1 <- floor(k * 10)
chisq.test(table(v1))
v2 <- floor(k2 * 10)
```

```
chisq.test(table(v2))

t2 <- NULL
for(i in 1:1000){
  l <- runif(1, 0, 1)
  k <- casio(l, 1000)
  k2 <- casio2(l, 1000)
  v1 <- floor(k * 10)
  u1 <- chisq.test(table(v1))$p.value
  v2 <- floor(k2 * 10)
  u2 <- chisq.test(table(v2))$p.value
  t2 <- cbind(t2, c(u1,u2))
}
par(mfrow = c(1, 2))
hist(t2[1,], xlab = 'p-value', main = '(1+sqrt(5))/2')
hist(t2[2,], xlab = 'p-value', main = 'sqrt(2)')

v1 <- table(floor(t2[1,] * 10))
chisq.test(v1)
v2 <- table(floor(t2[2,] * 10))
chisq.test(v2)


#6-################
fibonacci <- function(seed, n){
  m <- length(seed) - 1
  for (j in 1:n) {
    x <- (seed[j] + seed[j+m]) %% 1
    seed <- c(seed,x)
  }
  return(seed[-c(1:(m+1))])
}


p <- NULL
pid <- NULL
for(i in 1:10){
  k <- runif(10*i, 0, 1)
  k <- fibonacci(k, 10000)
  pvector <- chisq.test(table(ceiling(k*10)/10))$p.value
  p <- c(p, pvector)
  mat <- matrix(k[-1], ncol = 3333, byrow = F)
  mat2 <- apply(mat, 2, rank)
  mat3 <- mat2[1,]*100 + mat2[2,]*10 + mat2[3,]
  pidvector <- chisq.test(table(mat3))$p.value
  pid <- c(pid, pidvector)
```

```
}

plot(seq(10, 100, 10), p, type = "b",
    main = "Difference of p-values", xlab = 'm')
lines(seq(10, 100, 10), pid, type = "b", col = 2)
legend(70, .2, c("GOF", "Indept"), col = c(1, 2), lty = 1, pch =
1)
abline(h = 0.05, col = 3)
```