Statistical Computing and Simulation: Assignment 1

Deparment of Statistics, NCCU
葉佐晨　高崇哲
{112354016,112354020}@nccu.edu.tw

2024-03-13

Statistical Computing and Simulation

Assignment 1, Due March 15/2024

```
library(magrittr)
library(extraDistr)
library(ggplot2)
library(gridExtra)
library(nortest)
library(randtoolbox)
library(knitr)
```

```
SEED <- 123
```

1. (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) 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 $x_i = 171x_{i-1} \pmod{30269}$, $y_i = 172y_{i-1} \pmod{30307}$, $z_i = 170z_{i-1} \pmod{30323}$. Compare the results in (a) and (b), and discuss your findings.

```r
midSquare <- function(size = 1){
  initSample <- function(){sample(0:999999, size = 1)}

  num <- initSample()
  nums <- c(num)

  for (i in seq(size)){
    num <- num^2 %>%
      format(scientific = FALSE) %>%
      substr(start = 4, stop = 9) %>%
      ifelse(. == "", initSample(), .) %>%
      as.integer()

    num <- ifelse(num == nums[length(nums)], initSample(), num)
    nums <- c(nums, num)
  }

  return(nums)
}
```

In the Mid-Square Method, a random six-digit numeral is initially generated and squared, resulting in a numeral 12 digits or fewer. Should the squared numeral possess fewer than twelve digits, zeros are appended to achieve the requisite digit count. Upon obtaining a 12-digit number, digits four to nine are extracted to serve as the newly generated random numeral. In instances, the outcome is 0 or duplicates a previously generated number, do re-sampling procedure. This procedure has been writed within the function `midSquare`.

```r
set.seed(SEED)
rand_seq <- midSquare(size = 10000)
ks.test(rand_seq, "pdunif", 0, 999999)
```

```
## Warning in ks.test.default(rand_seq, "pdunif", 0, 999999): ties should not be
## present for the Kolmogorov-Smirnov test

##
##  Asymptotic one-sample Kolmogorov-Smirnov test
##
## data:  rand_seq
## D = 0.12211, p-value < 2.2e-16
## alternative hypothesis: two-sided
```

Following the simulation of 10,000 trials, the resulting sequence was subjected to the Kolmogorov-Smirnov Goodness-of-Fit Test to evaluate its conformity to a uniform distribution. The test yielded a statistic of 0.12211, with a p-value approaching 0. Consequently, the hypothesis positing that the random number sequence is uniformly distributed was rejected.

The warning message generated by the **ks.test** function is attributed to the presence of ties (replicated numbers) within the data. While ties are incompatible with the assumptions underlying tests for continuous distributions, methodologies exist that permit the asymptotic computation of test statistics despite these irregularities.

```
LCG <- function(size, mod, mul){
  nums <- c()
  xyz <- sapply(mod, function(x){sample(0:x, size = 1)})

  for (i in seq(size)){
    u <- sum(xyz / mod) %% 1
    nums <- c(nums, u)
    xyz <- (xyz * mul) %% mod
  }

  return(nums)
}
```

The Linear Congruential Generator (LCG) employs a composite of three multiplicative congruential generators in a sequential manner. Following the specification of modulus and multiplier parameters, element-wise vector operations are used to produce a new random number.

```
mod <- c(30269, 30307, 30323)
mul <- c(172, 171, 170)
set.seed(SEED)

rand_seq <- LCG(size = 10000, mod = mod, mul = mul)
ks.test(rand_seq, "punif", 0, 1)
```

```
##
##  Asymptotic one-sample Kolmogorov-Smirnov test
##
## data:  rand_seq
## D = 0.0068301, p-value = 0.7393
## alternative hypothesis: two-sided
```

Following the simulation of 10,000 trials, the resulting sequence was subjected to the Kolmogorov-Smirnov Goodness-of-Fit Test to evaluate its conformity to a uniform distribution. The test yielded a statistic of 0.0068, with a p-value approaching 0.7393. Consequently, the hypothesis positing that the random number sequence is uniformly distributed was not rejected.

2. (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 or big value.)
   (b) Hand calculators often use $U_{n+1} = (\pi + U_n)^5 \pmod 1$ to generate random numbers between 0 and 1.Compare the result with those in #1, and discuss your finding based on the comparison.

(a).

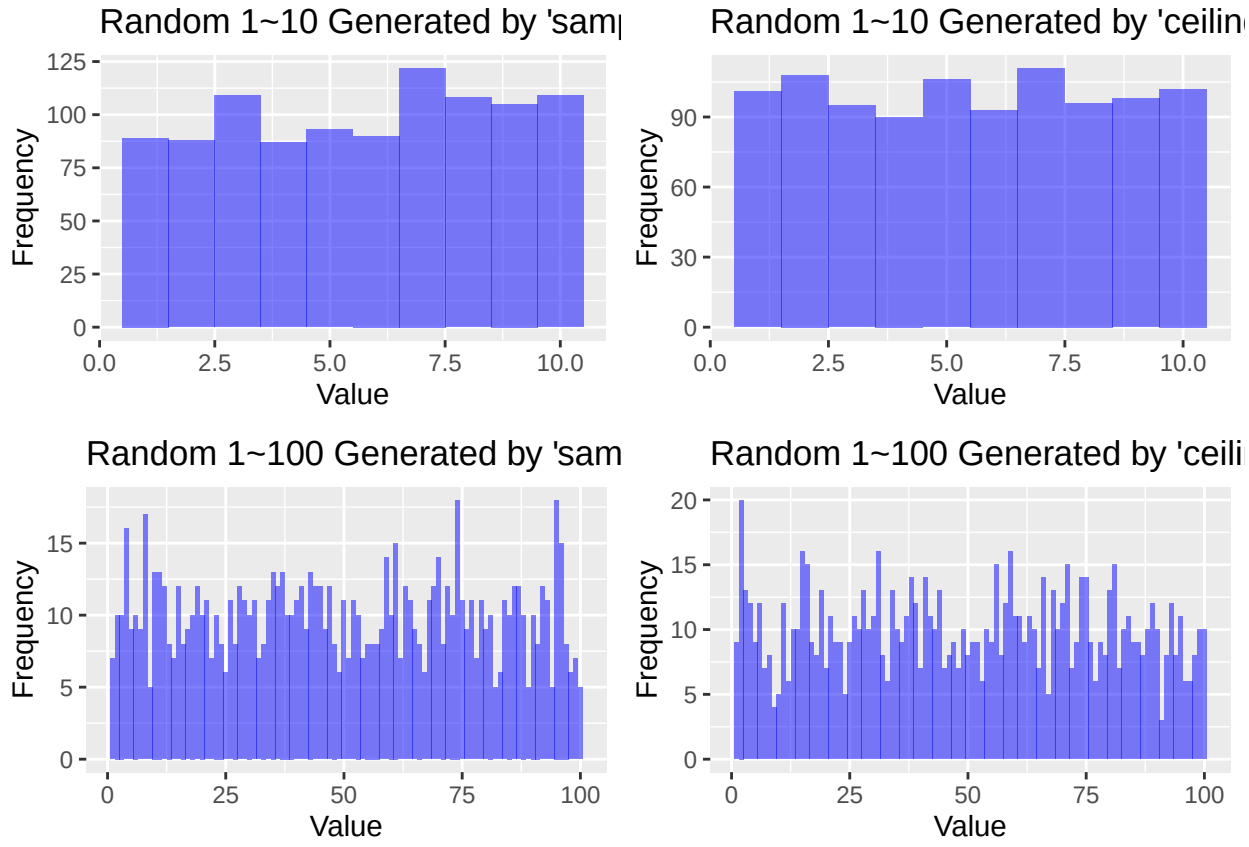
```r
generate_histogram <- function(data, k, title) {
  histogram <- ggplot(data.frame(x = data), aes(x)) +
    geom_histogram(binwidth = 1, fill = "blue", alpha = 0.5) +
    labs(title = title, x = "Value", y = "Frequency")
  # print(histogram)
}

calculate_and_test <- function(data, k) {
  num <- c()
  for (i in 0:(k-1)) {
    c <- data[data <= (1*(i+1)) & data > (1*i)]
    num <- c(num, length(c))
  }
  print(num)
  chisq.test(num, p = rep(1/k, k))
}
```

```r
set.seed(SEED)
sample_data_10 <- sample(1:10, 1000, replace = TRUE)
ceiling_data_10 <- ceiling(runif(1000) * 10)
sample_data_100 <- sample(1:100, 1000, replace = TRUE)
ceiling_data_100 <- ceiling(runif(1000) * 100)

grid.arrange(
  generate_histogram(sample_data_10, 10, "Random 1~10 Generated by 'sample'"),
```

```
  generate_histogram(ceiling_data_10, 10, "Random 1~10 Generated by 'ceiling(runif)'"),
  generate_histogram(sample_data_100, 100, "Random 1~100 Generated by 'sample'"),
  generate_histogram(ceiling_data_100, 100, "Random 1~100 Generated by 'ceiling(runif)'")
)
```



We conducted 1000 random simulations for both the "sample" and "ceiling" functions within the ranges of 1~10 and 1~100, respectively. From the graphs, we observed that regardless of whether it's the sample or ceiling function, when operating within a small range, the generated random numbers are relatively uniform. However, as the range increases, the uniformity decreases accordingly.

```
calculate_and_test(sample_data_10, 10)
```

```
##  [1]  89  88 109  87  93  90 122 108 105 109

##
##  Chi-squared test for given probabilities
##
## data:  num
## X-squared = 13.18, df = 9, p-value = 0.1546
```

```r
calculate_and_test(ceiling_data_10, 10)
```

```
##  [1] 101 108  95  90 106  93 111  96  98 102

##
##  Chi-squared test for given probabilities
##
## data:  num
## X-squared = 4.2, df = 9, p-value = 0.8978
```

```r
calculate_and_test(sample_data_100, 100)
```

```
##   [1]  7 10 10 16  9 10  9 17  5 13 13 12  8  7 12  8  9 10 12 10 11  7 10  8  6
##  [26] 11  8 12 11 10 11  7  8 11 13 12 13 10 10 11 12  9 13 12 12  9 12  8  6 11
##  [51]  7 11 10  7  8  8  8  9 14 10 15  7 12 11  9  8  6 11 12 14  8 12 10 18 11
##  [76]  9 11  7 11  9 10  5  6 11 10 12 12 10  5 10  8 12 11  5 18 15  8  6  7  5

##
##  Chi-squared test for given probabilities
##
## data:  num
## X-squared = 76.2, df = 99, p-value = 0.9569
```

```r
calculate_and_test(ceiling_data_100, 100)
```

```
##   [1]  9 20 13 12  9 12  7  8  4  5 12  6 10 10 16 15  9  8 13  7 11  9  9  5  9
##  [26] 11 10 13 10 11 16  8  6 13 10  9 11 14 12  7 14 11 10 13  7  8  9  7 10  8
##  [51]  9  9  6 10  9 15  8 12 16 11 11  9 11 10  7 14  5 13 10 12 15  7  9 14 14
##  [76]  9  6  9  8 13 15  7 10 11  9  9  8 10 12 10  3  8 12  8 11  6  6  8 10 10

##
##  Chi-squared test for given probabilities
##
## data:  num
## X-squared = 88.2, df = 99, p-value = 0.7732
```

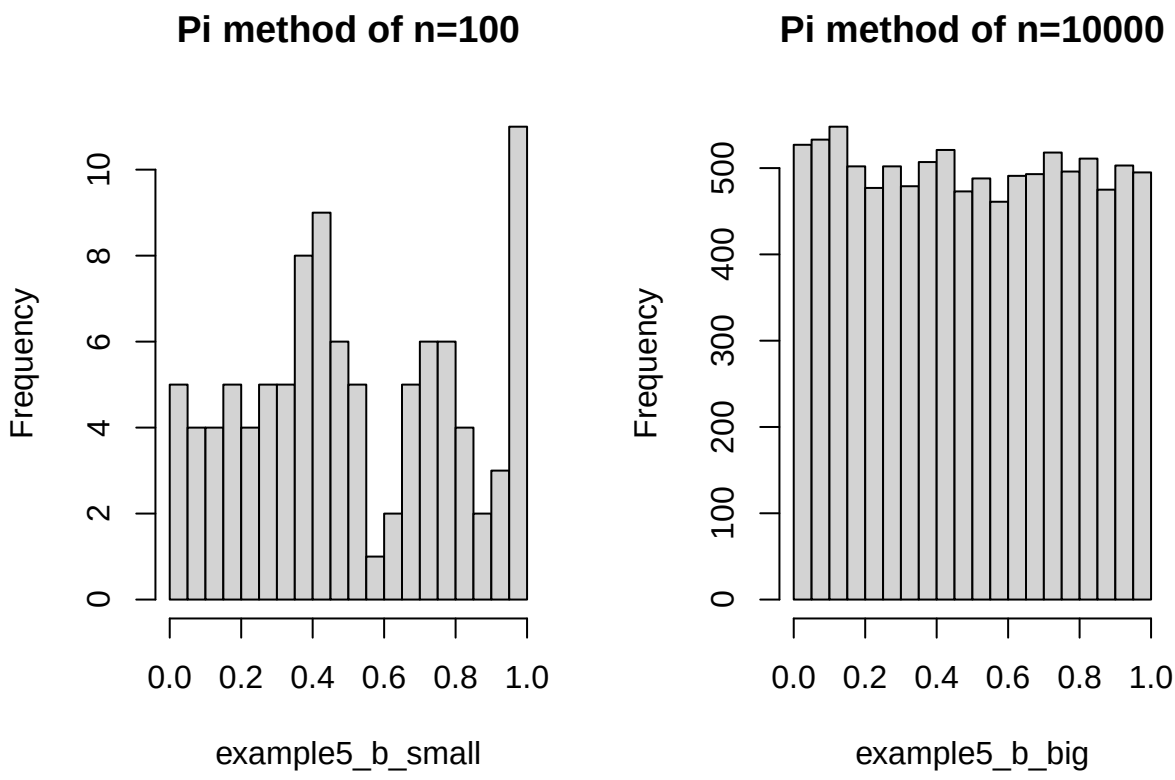
In the simulation with a range of 1~10, we divided the data into 10 blocks for chi-square tests. Similarly, in the simulation with a range of 1~100, we divided the data into 100 blocks for chi-square tests. With a significance level $\alpha = 0.05$, none of our four tests rejected the null hypothesis. Therefore, we conclude that the performance of the "sample" and "ceiling" functions in this experiment is similar.

(b).

```r
pi_fn<-function(seed,n){
  un<-seed
  a<-c()
  for (i in 1:n){
    un<-(un+pi)^5
    un<-un%%1
    a<-c(a,un)
  }
  return(a)
}
```

```r
example5_b_small<-pi_fn(101,100)
example5_b_big<-pi_fn(100,10000)

par(mfrow = c(1, 2))
hist(example5_b_small,main="Pi method of n=100", breaks = 20)
hist(example5_b_big,main="Pi method of n=10000",  breaks = 20)
```



From the graphs, we can observe that when we conduct 100 simulations using this method, it doesn't

quite resemble a uniform distribution. However, after 10,000 simulations, the differences in values for each interval significantly decrease, resembling a uniform distribution.

```
ks.test(example5_b_small, "punif", 0, 1)
```

```
##
##  Asymptotic one-sample Kolmogorov-Smirnov test
##
## data:  example5_b_small
## D = 0.063152, p-value = 0.82
## alternative hypothesis: two-sided
```

```
ks.test(example5_b_big, "punif", 0, 1)
```

```
##
##  Asymptotic one-sample Kolmogorov-Smirnov test
##
## data:  example5_b_big
## D = 0.012056, p-value = 0.1093
## alternative hypothesis: two-sided
```

From the Kolmogorov-Smirnov test results, it is evident that regardless of conducting 100 or 10,000 simulations, when $\alpha = 0.05$, the method does not reject the null hypothesis. This indicates that the method performs well in simulating a uniform distribution from 0~1. Based on the test results, we believe that this method outperforms Method #1.

3. There are several ways for checking the goodness-of-fit for empirical data. In specific, there are a lot of normality tests available in R. Generate a random sample of size 10, 50, and 100 from N(0,1) and t-distribution (with degrees 10 and 20) in R. You may treat testing random numbers from t-distribution as the power. For a level of significance $\alpha = 0.05$ test, choose at least four normality tests in R (`nortest` module) to check if this sample is from N(0,1). Tests used can include the Kolmogorov-Smirnov test and the Cramer-von Mises test. Note that you need to compare the differences among the tests you choose.

```
norTest <- function(testList, n){

  res <- rbind(
    sapply(testList, function(test){test(rnorm(n))[["p.value"]]}),
    sapply(testList, function(test){test(rt(n, df = 10))[["p.value"]]}),
    sapply(testList, function(test){test(rt(n, df = 20))[["p.value"]]})
```

```
  )

  colnames(res) <- c(c("ad", "cvm", "lillie", "pearson", "sf"))
  rownames(res) <- c("N(0,1)", "t(df=10)", "t(df=20)")
  return(res)
}

countReject <- function(n_trial, n, alpha){
   replicate(n = n_trial,
             norTest(testList, n = n) <= alpha) %>%
    apply(MARGIN = c(1, 2), sum)
}
```

```
n <- c(10, 50, 100)
testList <- c(ad.test, cvm.test, lillie.test, pearson.test, sf.test)
n_trial <- 1000
alpha <- 0.05
```

```
set.seed(SEED)
rejList <- lapply(n, function(n){countReject(n_trial = 1000, n = n, alpha = 0.05)})
names(rejList) <- paste0("n=", n)
print(rejList)
```

```
## $`n=10`
##          ad cvm lillie pearson sf
## N(0,1)   60  48     54      61 52
## t(df=10) 86  65     57      73 89
## t(df=20) 60  51     63      70 61
##
## $`n=50`
##           ad cvm lillie pearson  sf
## N(0,1)    55  63     41      58  59
## t(df=10) 128 102    110      62 183
## t(df=20)  83  61     62      66 117
##
## $`n=100`
##           ad cvm lillie pearson  sf
## N(0,1)    58  53     55      48  40
## t(df=10) 177 143    121      74 282
## t(df=20)  79  77     63      49 140
```

In the reported outcomes, the rejection frequency distribution is categorized according to sample size `n` at levels of 10, 50, and 100. The analyses encompass the Anderson-Darling test, Cramer-von Mises test, Lilliefors (Kolmogorov-Smirnov) test, Pearson chi-square test, and Shapiro-Francia test for assessing normality. Theoretically, given a sequence that adheres to a normal distribution, an average rejection frequency of 50 is anticipated across 10,000 trials at a significance level of 0.05.

Observations indicate that for sequences conforming to a normal distribution, the rejection frequency approximates 50 at each sampled size level. In instances where the sequence follows a t-distribution, the Shapiro-Francia and Anderson-Darling tests demonstrate a higher frequency of rejection compared to other tests. It is also noted that an increase in sample size and degrees of freedom correlates with a heightened rejection frequency across all tests.

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

```
gap.test=function(data,a,b){
  n=length(data)
  x=c(1:n)*(a<data & data<b)
  x1=x[x>0]
  y=x1[-1]-x1[-length(x1)]-1
  return(table(y))
}

set.seed(SEED)
uniform_data <- runif(1200)
gap_counts <- gap.test(uniform_data, 0.2, 0.8)
gap_counts
```

```
## y
##   0   1   2   3   4   5   6
## 446 179  63  26  14   3   2
```

We used the `runif` function in R language to generate 1200 values, and then employed Gap test to check the independence of this sample. In Gap test, we set the thresholds to 0.2 and 0.8. The results above represent the frequencies of samples falling within the thresholds and the distances between them. Since we are proceeding with a chi-square test next, we merged the frequencies of gaps 5 and 6.

```
result_vector <- numeric()
for (i in c(0:5)) {
  a = 0.6*0.4^i
```

```
  result_vector <- c(result_vector, a)
}
result_vector[6] <- 1 - sum(result_vector[1:5])
observation <- c(446, 179,  63,  26,  14, 5)
chisq.test(observation, p = result_vector)
```

```
##
##  Chi-squared test for given probabilities
##
## data:  observation
## X-squared = 2.5806, df = 5, p-value = 0.7643
```

The result from the Gap test shows that when $\alpha = 0.05$, we do not reject the null hypothesis.

```
permute.test=function(data,k){
  y=rep(10,k)^c((k-1):0)
  x=matrix(data,ncol=k,byrow=T)
  x1=apply(x,1,rank)
  yy=apply(x1*y,2,sum)
  return(table(yy))
}

permute.test(uniform_data, 3)
```

```
## yy
## 123 132 213 231 312 321
##  54  66  67  80  65  68
```

Next, we utilize a Permutation test. Samples are grouped in sets of 3, each set sorted by magnitude. The results above indicate the quantity of various permutations.

```
observation <- c(54, 66,  67,  80,  65, 68)
chisq.test(observation, p = rep(1/6, 6))
```

```
##
##  Chi-squared test for given probabilities
##
## data:  observation
## X-squared = 5.15, df = 5, p-value = 0.3978
```

The result from the Permutation test shows that when $\alpha = 0.05$, we do not reject the null hypothesis.

```
run.test <- function(data, base_number) {
  n <- length(data)
  observed_runs <- sum(data > base_number)
  expected_runs <- n / 2
  p_value <- 2*pbinom(observed_runs, size = n, prob = 0.5, lower.tail = TRUE)
  return(list(observed_runs = observed_runs, expected_runs = expected_runs, p_value = p_value))
}

run.test(uniform_data, 0.5)
```

```
## $observed_runs
## [1] 591
##
## $expected_runs
## [1] 600
##
## $p_value
## [1] 0.6236232
```

Finally, we conduct a Run test. Similarly, from the test results, it can be observed that when $\alpha = 0.05$, we do not reject the null hypothesis.

```
py_data <- read.csv('random_numbers.csv')
py_data <- py_data[, 1]
gap_counts <- gap.test(py_data, 0.2, 0.8)

result_vector <- numeric()
for (i in c(0:5)) {
  a = 0.6*0.4^i
  result_vector <- c(result_vector, a)
}
result_vector[6] <- 1 - sum(result_vector[1:5])
observation <- c(423, 169,  69,  21,  16, 10)
chisq.test(observation, p = result_vector)
```

```
##
##  Chi-squared test for given probabilities
##
```

```
## data:  observation
## X-squared = 4.8949, df = 5, p-value = 0.4288
```

```
permute.test(py_data, 3)
```

```
## yy
## 123 132 213 231 312 321
##  71  69  58  77  68  57
```

```
observation <- c(71, 69,  58,  77,  68, 57)
chisq.test(observation, p = rep(1/6, 6))
```

```
##
##  Chi-squared test for given probabilities
##
## data:  observation
## X-squared = 4.52, df = 5, p-value = 0.4772
```

```
run.test(py_data, 0.5)
```

```
## $observed_runs
## [1] 573
##
## $expected_runs
## [1] 600
##
## $p_value
## [1] 0.1259869
```

We also generated 1200 random numbers using the `random` package in Python and performed Gap test, Permutation test, and Run test. The results also do not reject the null hypothesis.

5. (a) Use the search engine to download the first one million digits of pi (for example, https://www.piday.org/million/) 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

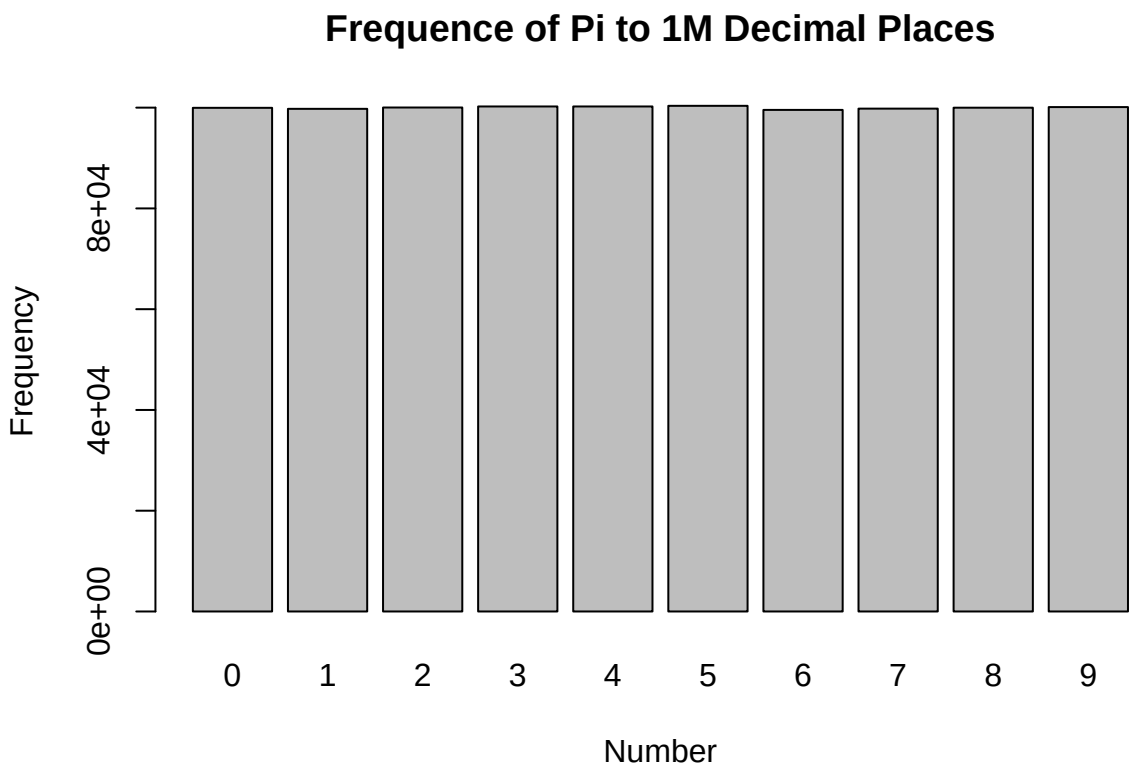
      (a) satisfy the assumption of random numbers.

```
pi1M <- readLines("Pi1MDP.txt", warn = FALSE) %>%
  strsplit(split = "") %>%
  unlist() %>%
  as.integer()
```

```
freq_pi <- table(pi1M)
freq_pi %>%
  t %>%
  kable()
```

0	1	2	3	4	5	6	7	8	9
99959	99758	100026	100229	100230	100359	99548	99800	99985	100106

```
barplot(freq_pi,
        main = "Frequence of Pi to 1M Decimal Places",
        xlab = "Number", ylab = "Frequency")
```



```r
chisq.test(freq_pi)
```

```
##
##  Chi-squared test for given probabilities
##
## data:  freq_pi
## X-squared = 5.5091, df = 9, p-value = 0.7879
```

```r
TEST <- gap.test(pi1M/10, 0.25, 0.75)
result_vector <- numeric()
for (i in c(0:16)) {
  a = 0.5*0.5^i
  result_vector <- c(result_vector, a)
}
result_vector[17] <- 1 - sum(result_vector[1:16])
TEST <- c(TEST[1:16], sum(TEST[17:19]))
chisq.test(TEST, p = result_vector)
```

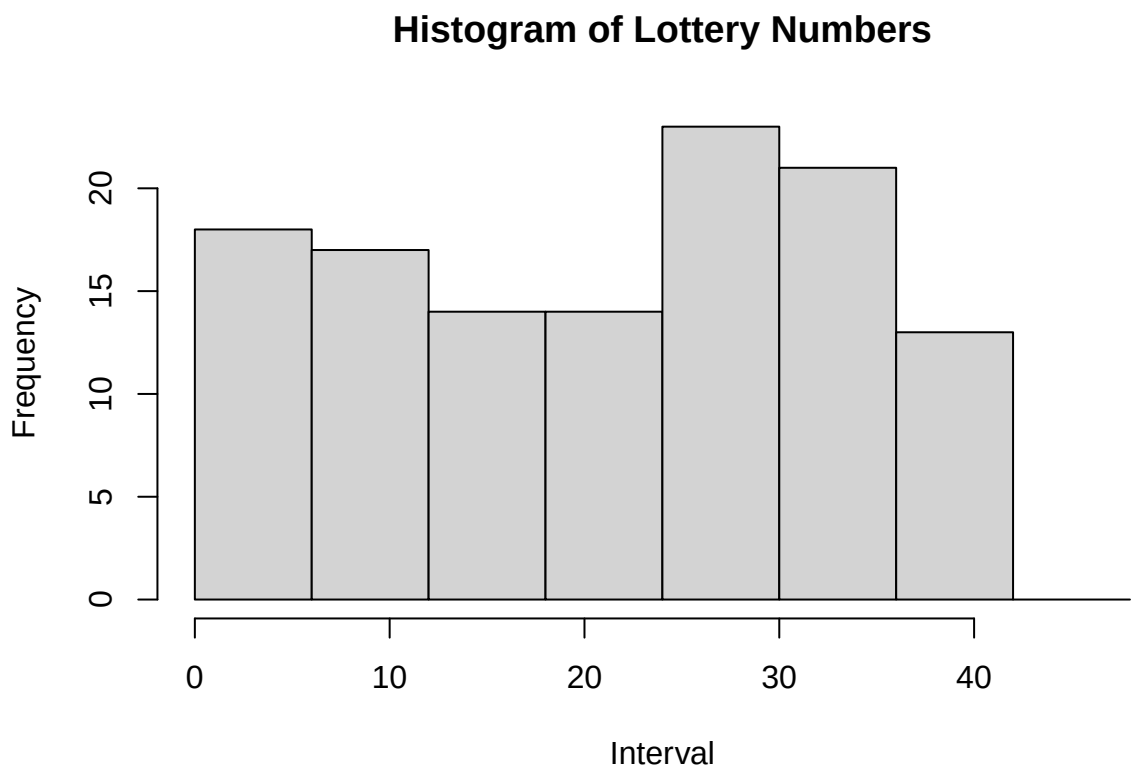
```
##
##  Chi-squared test for given probabilities
##
## data:  TEST
## X-squared = 12.86, df = 16, p-value = 0.6829
```

To test the randomness of numbers, it is important to conduct both uniformity and independence tests. Using the Chi-squared goodness-of-fit test, it was determined that the value of Pi to one million decimal places cannot be rejected as uniformly distributed. Subsequently, the Gap test was applied, which likewise did not lead to the rejection of the hypothesis that the numbers are independent.

6. 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.

```r
lottery_numbers <- c(
  22, 31, 34, 25, 21, 19,
  5, 18, 25, 26, 35, 42,
  32, 21, 9, 27, 31, 6,
  5, 25, 2, 16, 32, 9,
  15, 29, 5, 36, 13, 10,
```

```
  36, 16, 12, 26, 8, 34,
  4, 40, 27, 21, 14, 5,
  29, 4, 10, 23, 39, 14,
  30, 12, 40, 32, 35, 20,
  40, 6, 20, 29, 38, 35,
  32, 10, 15, 2, 30, 23,
  24, 20, 36, 19, 7, 12,
  1, 6, 7, 12, 42, 20,
  25, 39, 20, 38, 29, 37,
  26, 2, 15, 29, 4, 33,
  17, 39, 3, 15, 11, 1,
  13, 39, 28, 30, 25, 29,
  7, 9, 29, 34, 39, 36,
  28, 31, 16, 35, 6, 30,
  10, 32, 13, 4, 9, 33)
interval <- cut(lottery_numbers, breaks = seq(0, 42, by = 6))
hist(lottery_numbers, breaks = seq(0, 50, by = 6), main = "Histogram of Lottery Numbers", xlab = "
```



We divided the lottery numbers into groups of 6 digits each and plotted a histogram. From the results,

we can see that there is not much difference in the quantity between each group.

```r
chisq.test(table(interval), p = rep(1/7, 7))
```

```
##
##  Chi-squared test for given probabilities
##
## data:  table(interval)
## X-squared = 5.0667, df = 6, p-value = 0.5353
```

```r
set.seed(SEED)

sum_groups <- function(numbers, group_size) {
  num_groups <- length(numbers) / group_size
  group_sums <- sapply(1:num_groups, function(i) {
    sum(numbers[((i - 1) * group_size + 1):(i * group_size)])
  })
  return(group_sums)
}

group_sums <- sum_groups(lottery_numbers, 6)
print(group_sums)
```

```
##  [1] 152 151 126  89 108 132 111 119 169 168 112 118  88 188 109  86 164 154 146
## [20] 101
```

```r
dummy_variable_number <- ifelse(group_sums > 129, 1, 0)
first_half <- dummy_variable_number[1:10]
second_half <- dummy_variable_number[11:20]

count_first_0 <- sum(first_half == 0)
count_first_1 <- sum(first_half == 1)
count_second_0 <- sum(second_half == 0)
count_second_1 <- sum(second_half == 1)
matrix_data <- matrix(c(count_first_0, count_second_0, count_first_1, count_second_1), nrow = 2, b

print(matrix_data)
```

```
##      [,1] [,2]
## [1,]    5    6
## [2,]    5    4
```

```r
chisq.test(matrix_data,correct = FALSE)
```

```
## Warning in chisq.test(matrix_data, correct = FALSE): Chi-squared approximation
## may be incorrect
```

```
##
##  Pearson's Chi-squared test
##
## data:  matrix_data
## X-squared = 0.20202, df = 1, p-value = 0.6531
```

Next, we examined the uniformity and independence of the sample using both Chi-square test. For the uniformity, we divided the interval into blocks of 6 digits each, and for the indepedence, We sum up the six numbers of each day to obtain 20 new statistical quantities,then, the horizontal axis of matrix represents the first 10 days and the following 10 days, while the vertical axis represents numbers greater than 129 and numbers less than 129.The results of both tests indicatethat when $\alpha = 0.05$, we do not reject the null hypothesis.Therefore, we conclude that these lottery numbers are random.