Statistical Computing and Simulation: Assignment 2

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

2024-03-29

Statistical Computing and Simulation

Assignment 2, Due March 27/2024

Question 1

We can use the command `arima.sim` in R to generate random numbers from ARIMA models.

a. We generate 100 random numbers from AR(2) with parameter values $(\phi_1, \phi_2) = (\theta, \theta)$ and apply correlation between x_i vs. x_{i+1} and x_i vs. x_{i+2} as a tool for verifying independence. You should repeat the simulation at least 1,000 times and try different θ values, such as $\theta = 0$, 0.05, 0.10, 0.15, and 0.20.
b. Using ARIMA random numbers to evaluate the type-1 and type-2 errors of various independence tests, e.g., Gap, Up-and-down, and Permutation tests.

Result (a)

```
## [1] When theta is 0 , the average correlation between x_i and x_i+1 is : -0.01178
## [1] When theta is 0 , the average correlation between x_i and x_i+2 is : -0.01249
## [1] When theta is 0.05 , the average correlation between x_i and x_i+1 is : 0.03442
## [1] When theta is 0.05 , the average correlation between x_i and x_i+2 is : 0.04012
## [1] When theta is 0.1 , the average correlation between x_i and x_i+1 is : 0.09351
## [1] When theta is 0.1 , the average correlation between x_i and x_i+2 is : 0.09608
## [1] When theta is 0.15 , the average correlation between x_i and x_i+1 is : 0.14822
## [1] When theta is 0.15 , the average correlation between x_i and x_i+2 is : 0.15823
## [1] When theta is 0.2 , the average correlation between x_i and x_i+1 is : 0.22834
## [1] When theta is 0.2 , the average correlation between x_i and x_i+2 is : 0.22751
```

我們使用不同的θ值，從AR(2)進行了1000次的100個亂數模擬，從結果中我們可以看出，不管是x_i和x_{i+1}還是x_i和x_{i+2}的平均相關係數，當θ值越大，平均相關係數也就越高。

Result (b)

```
## When theta is 0 , number of rejection is 34.2 % in Gap test
## When theta is 0.05 , number of rejection is 34.3 % in Gap test
## When theta is 0.1 , number of rejection is 37.1 % in Gap test
## When theta is 0.15 , number of rejection is 39.2 % in Gap test
## When theta is 0.2 , number of rejection is 44.8 % in Gap test
```

```
## When theta is 0 , number of rejection is 5.4 % in Permutation test
## When theta is 0.05 , number of rejection is 4.2 % in Permutation test
## When theta is 0.1 , number of rejection is 4.9 % in Permutation test
## When theta is 0.15 , number of rejection is 3.8 % in Permutation test
## When theta is 0.2 , number of rejection is 4.5 % in Permutation test
```

```
## When theta is 0 , number of rejection is 6.6 % in Run test
## When theta is 0.05 , number of rejection is 7.5 % in Run test
## When theta is 0.1 , number of rejection is 12 % in Run test
## When theta is 0.15 , number of rejection is 22.5 % in Run test
## When theta is 0.2 , number of rejection is 37.6 % in Run test
```

Type I error 算法：$\theta = 0$ 資料為獨立,故於此情況下計算拒絕個數。 Type II error 算法：$\theta > 0$ 資料為不獨立,於此情況下計算不拒絕個數。

Gap test :

- $\theta = 0$, Type I error = 0.342
- $\theta = 0.05$, Type II error = 0.657
- $\theta = 0.1$, Type II error = 0.629
- $\theta = 0.15$, Type II error = 0.608
- $\theta = 0.2$, Type II error = 0.552

Permutation test :

- $\theta = 0$, Type I error = 0.054
- $\theta = 0.05$, Type II error = 0.958
- $\theta = 0.1$, Type II error = 0.951
- $\theta = 0.15$, Type II error = 0.962
- $\theta = 0.2$, Type II error = 0.955

Run test :

- $\theta = 0$, Type I error $= 0.066$
- $\theta = 0.05$, Type II error $= 0.925$
- $\theta = 0.1$, Type II error $= 0.880$
- $\theta = 0.15$, Type II error $= 0.775$
- $\theta = 0.2$, Type II error $= 0.624$

從結果比較中我們可以看出，Permutation test對於 θ值的變化敏感程度較低，相反地，Gap test與Run test能夠反映出θ值越大，拒絕$\theta = 0$ 的比例越高。

Question 2

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.

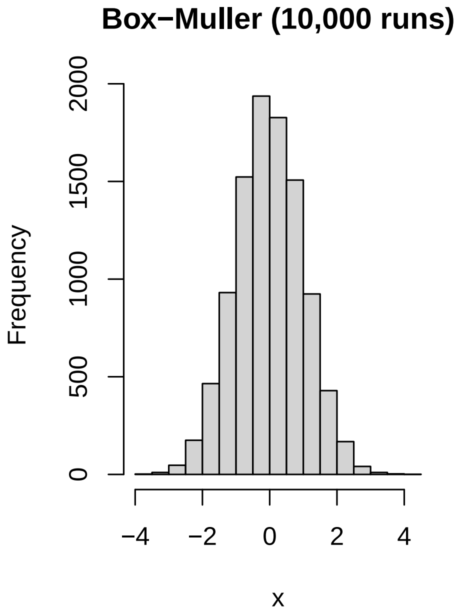
b. In the class we mentioned it is found by several researchers that

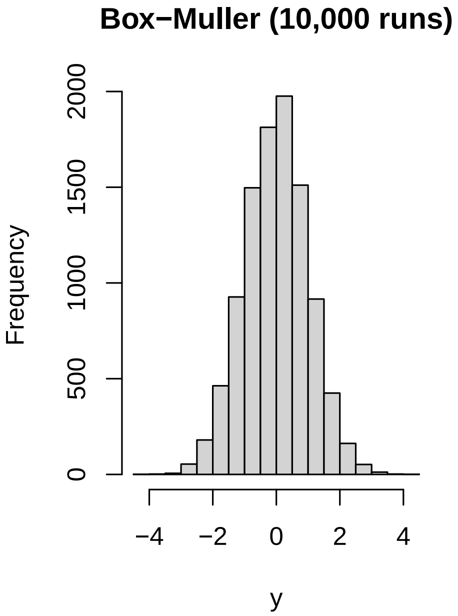
   a (multiplier) = 131
   c (increment) = 0
   m (modulus) = 2ˆ35

would have $X \in (\breve{}3.3, 3.6)$, if plugging congruential generators into the Box-Muller method. Verify if you would have similar results.

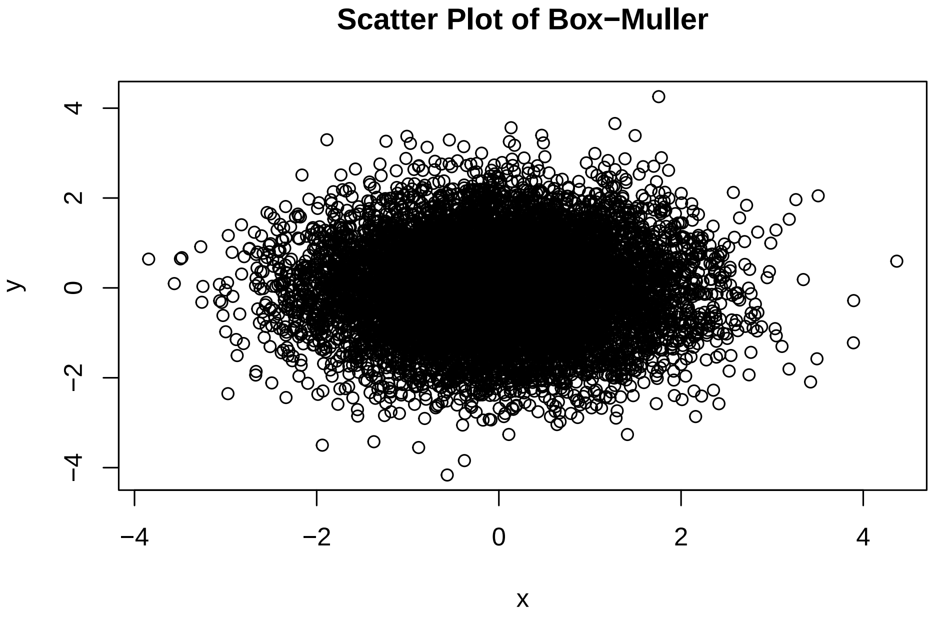
Result (a)

Box-Muller一次可以產生兩個亂數， 並且兩個亂數都符合常態分配且相互獨立。下方是使用Box-Muller產生10,000組亂數，並分別繪製直方圖，可以觀察到亂數的分布很接近常態分佈，具體是否拒絕常態性的假設，還需要透過KS Test進行檢定。

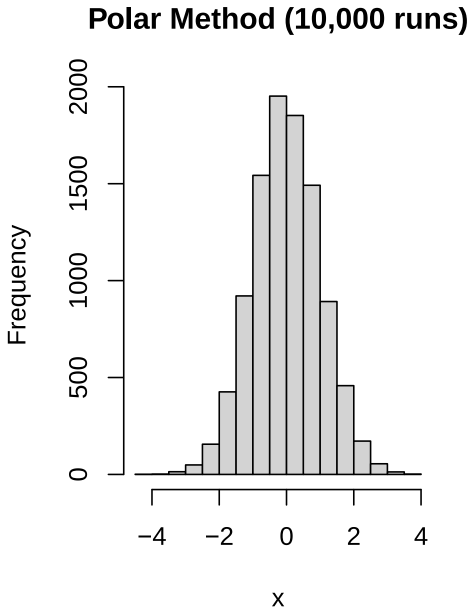


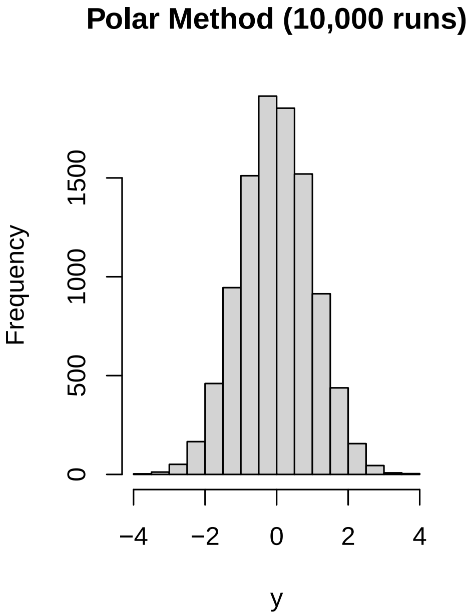


另外由於亂數是二維的，因此我們可以在平面上繪製散佈圖，可以觀察到兩個亂數並沒有明顯相關性。

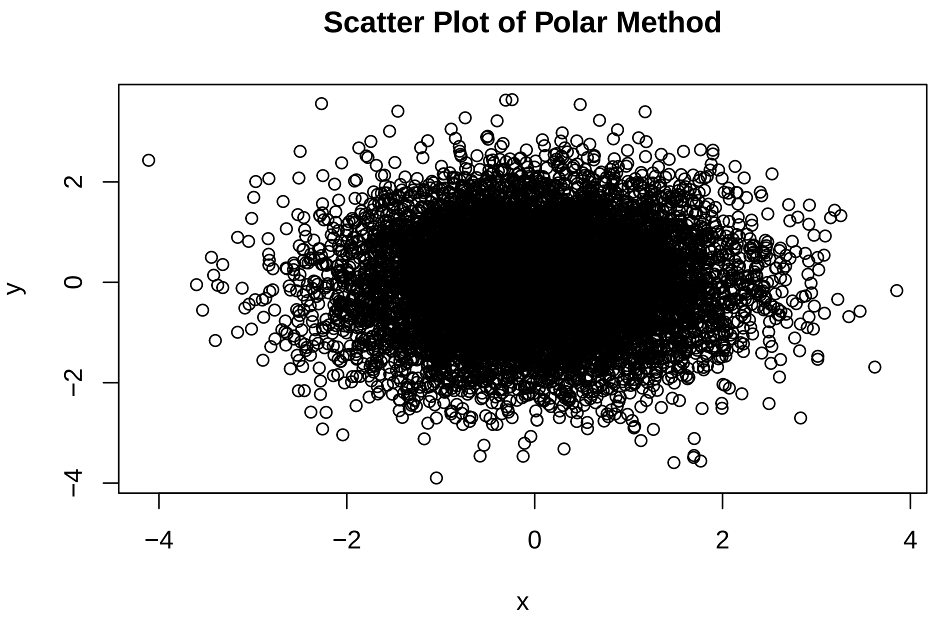


Polar方法一次可以產生兩個亂數，並且兩個亂數都符合常態分配且相互獨立。下方是使用Polar產生10,000組亂數，並分別繪製直方圖，可以觀察到亂數的分布很接近常態分佈。

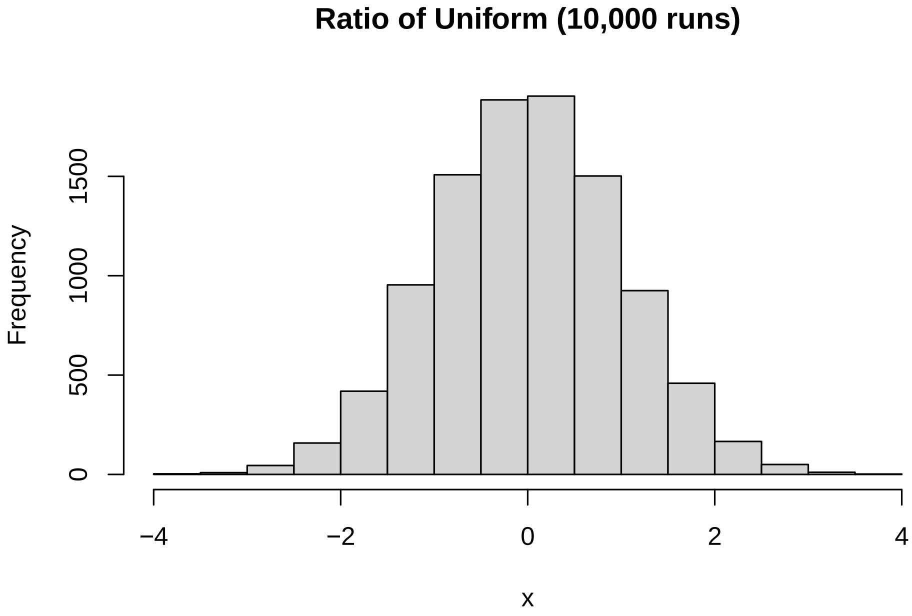




另外由於亂數是二維的，因此我們可以在平面上繪製散佈圖，可以觀察到兩個亂數並沒有明顯相關性。



Ratio-of-uniforms一次可以產生一個亂數，下方是使用Ratio-of-uniforms產生10,000個亂數，並繪製直方圖，可以觀察到亂數的分布很接近常態分佈。



為了比較三種亂數產生的方法，我們將分別進行1,000次KS Test，樣本數為10,000，比較拒絕常態性的假設的次數，由此判斷最好的亂數產生方法。

在表格中我們可以看到在1,000次試驗中， Box-Muller拒絕常態性44次， Polar拒絕45次， Ratio-of-Uniforms拒絕41次。根據這個結果，我們選定Ratio-of-Uniforms為最佳的亂數產生方法，不過因為每個亂數產生方法的結果相當接近，因此不同次試驗可能產生不同的結果。

表 1: Reject of Generators

	Reject	NonReject
BoxMuller	44	956
Polar	45	955
RatioUnif	41	959

Result (b)

在Box-Muller中加入congruential generators後，產生10,000個亂數，再計算樣本的全距。在本次模擬中，全距為$(-3.7, 3.7)$，並不必然如題目所述的$X \in (\breve{}3.3, 3.6)$。

```
## The range of X is from -3.7 to 3.7.
```

Question 3

Write a program to generate random numbers from Poisson distribution. This program has the function for choosing the starting points, such as from starting from 0, mean, or median. In addition, this program can record the numbers of steps needed for generating a random number. Similar to what we saw in class, if $\lambda = 10$, compare the numbers of steps needed if starting from 0 and mean.

Result

```
## Average number of steps starting from 0: 7.77
```

```
## Average number of steps starting from mean: 0.56
```

我們撰寫自定義函數`generatePoisson`，可以將服從0~1的均勻分配亂數轉換成服從 $\lambda = 10$ 的普瓦松分配，並記錄轉換過程所需要的步數，從模擬100次的平均結果可以看出，從期望值出發所需要的步數，明顯少於從0出發，因此當λ很大時，建議從期望值開始轉換。

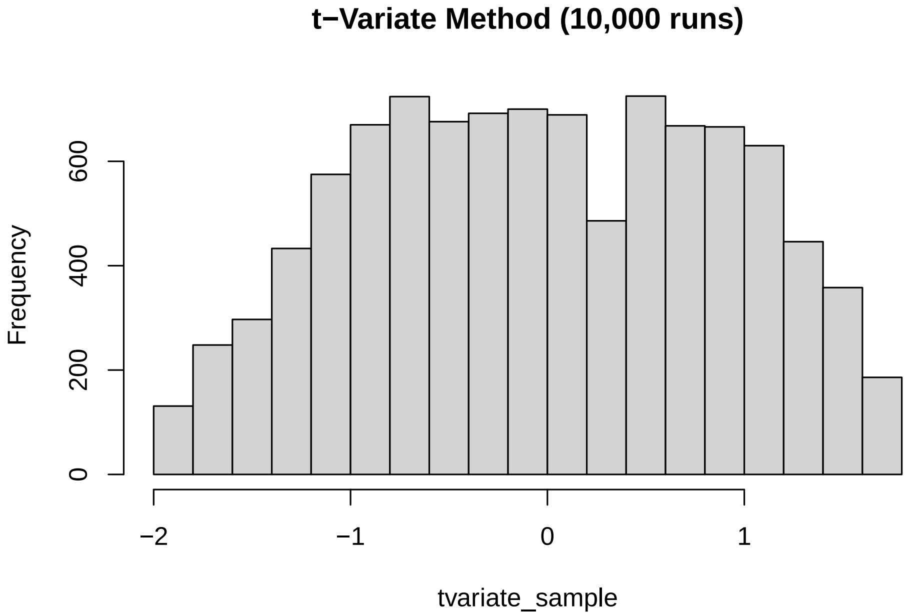
Question 4

Show that the following algorithm generates random numbers from t_v-variate. (It is a rejection algorithm with $g(x) \propto \min(1, 1/x^2)$)

1. Generate U,$U_1 \sim$ U$(0, 1)$.
2. If $U < \frac{1}{2}$ then $X = \frac{1}{4U`1}$,$V = x\text{fl}2U_1$ else $X = 4U_1`3$, $V = U_1$.
3. If $V < 1`\frac{|X|}{2}$ go to 5.
4. If$V > (1 + \frac{X_2}{V})^{-(v+1)/2}$, go to 1.
5. Return X.

Result

t-Variate一次可以產生一個亂數，下方是使用t-Variate產生10,000個亂數，並繪製直方圖，可以觀察到亂數的分布不太接近常態分佈。



和前題的判斷方法相同，我們進行了1,000次試驗，樣本數為10,000。結果發現拒絕的次數高達379次，因此判斷t-Variate不是一個合適產生常態亂數的方式。

表 2: Reject of t-Variate

	Reject	NonReject
tvVariate	379	621

Question 5

Given the following matrix:

$$A = \begin{pmatrix} 1 & 0.5 & 0.25 & 0.125 \\ 0.5 & 1 & 0.5 & 0.25 \\ 0.25 & 0.5 & 1 & 0.5 \\ 0.125 & 0.25 & 0.5 & 1 \end{pmatrix}$$

a. Write a program to compute the Cholesky decomposition of A. To double check your result, use the command “chol” in R to verify the result.

b. Use the commands “eigen”, “qr”, and “svd” on A and check if these commands work properly.

Result (a)

```
##      [,1]      [,2]      [,3]      [,4]
## [1,]    1 0.5000000 0.2500000 0.1250000
## [2,]    0 0.8660254 0.4330127 0.2165064
## [3,]    0 0.0000000 0.8660254 0.4330127
## [4,]    0 0.0000000 0.0000000 0.8660254
```

```
##      [,1]      [,2]      [,3]      [,4]
## [1,]    1 0.5000000 0.2500000 0.1250000
## [2,]    0 0.8660254 0.4330127 0.2165064
## [3,]    0 0.0000000 0.8660254 0.4330127
## [4,]    0 0.0000000 0.0000000 0.8660254
```

我們撰寫自定義函數myChol，可以用來計算一個矩陣的Cholesky decomposition，透過R內建函數chol檢驗，我們帶入題目的矩陣A，可以獲得相同的結果。

Result (b)

```
##       [,1] [,2] [,3]  [,4]
## [1,] 1.000 0.50 0.25 0.125
## [2,] 0.500 1.00 0.50 0.250
## [3,] 0.250 0.50 1.00 0.500
## [4,] 0.125 0.25 0.50 1.000
```

```
##       [,1] [,2] [,3]  [,4]
## [1,] 1.000 0.50 0.25 0.125
## [2,] 0.500 1.00 0.50 0.250
## [3,] 0.250 0.50 1.00 0.500
## [4,] 0.125 0.25 0.50 1.000
```

```
##       [,1] [,2] [,3]  [,4]
## [1,] 1.000 0.50 0.25 0.125
## [2,] 0.500 1.00 0.50 0.250
## [3,] 0.250 0.50 1.00 0.500
## [4,] 0.125 0.25 0.50 1.000
```

我們對A矩陣分別進行了函數eigen, qr和svd的計算，並將計算後的各值帶入公式，結果皆等於原本的矩陣A，這代表這三個函數的運作都是正常的。

Question 6

Figure a way to find the parameters of AR(1) and AR(2) models for the data `lynx` in R. Also, apply statistical software (e.g., R, SAS, SPSS, & Minitab) to get estimates for the AR(1) and AR(2) model and compare them to those from your program.

Result

使用本文的AR模型和R內建的AR函數，進行比較。可以發現估計值相當接近，因此本文所使用的AR模型具有良好的效果。

% Table created by stargazer v.5.2.3 by Marek Hlavac, Social Policy Institute. E-mail: marek.hlavac at gmail.com % Date and time: Fri, Mar 29, 2024 - 9:13:40 PM

表 3: Result of My AR Model and R's ARIMA Model

	Dependent variable:			
	lynx		x	
	ARIMA		*OLS*	
	AR(1)	AR(2)	AR(1)	AR(2)
ar1	0.717***	1.147***	0.720***	1.152***
	(0.065)	(0.074)	(0.066)	(0.077)
ar2		−0.600***		−0.606***
		(0.074)		(0.077)
intercept	1,550.571***	1,545.446***		
	(356.693)	(181.674)		
Constant			454.152***	710.106***
			(145.351)	(121.841)
Observations	114	114	113	112
R^2			0.515	0.690
Adjusted R^2			0.510	0.685
Log Likelihood	−960.495	−935.016		
σ^2	1,210,542.000	768,159.100		
Akaike Inf. Crit.	1,926.991	1,878.032		
Residual Std. Error			1,111.610	893.338
F Statistic			117.668***	121.577***

Note: *p<0.1; **p<0.05; ***p<0.01

Appendix

R code

```
library(stats)
library(randtests)
library(knitr)
library(stargazer)
SEED <- 123
theta_values <- c(0, 0.05, 0.1, 0.15, 0.2)

set.seed(SEED)
for (theta in theta_values) {
  correlation1_values <- c()
  correlation2_values <- c()

  for (i in 1:1000) {
    arima_sim_data <- arima.sim(model = list(order = c(2,0,0), ar = c(theta, theta)), n = 100, sd
    correlation1 <- cor(arima_sim_data[1:99], arima_sim_data[2:100])
    correlation2 <- cor(arima_sim_data[1:98], arima_sim_data[3:100])
    correlation1_values <- c(correlation1_values, correlation1)
    correlation2_values <- c(correlation2_values, correlation2)
  }

  avg_correlation1 <- mean(correlation1_values)
  avg_correlation2 <- mean(correlation2_values)

  print(paste("When theta is", theta, ", the average correlation between x_i and x_i+1 is :", roun
  print(paste("When theta is", theta, ", the average correlation between x_i and x_i+2 is :", roun
}
gapTest <- function(data, a, b) {
  gap <- 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))
  }
  gap_counts <- gap(data, a, b)

  vec <- gap_counts

  while (vec[length(vec)] < 5) {
    vec <- c(vec[1:(length(vec) - 2)], vec[(length(vec) - 1)] + vec[length(vec)])
  }

  result_vector <- numeric()
  for (i in c(0:(length(vec) - 1))) {
    a1 <- (b-a) * (1-b+a)^i
    result_vector <- c(result_vector, a1)
  }

  result_vector[length(result_vector)] <- 1 - sum(result_vector[1:(length(result_vector) - 1)])

  chisq.test(vec, p = result_vector)
}

numRej1 <- function(n, theta) {
  t1err=0
  for (i in 1:n){
    set.seed(1+i)
    data <- arima.sim(model = list(order = c(2,0,0), ar = c(theta, theta)), n = 100, sd = 1)
    data <- data+abs(min(data))
    data <- data/max(data)
    x = gapTest(data,47/100,97/100)
    if ((x$p.value)<=0.05) (t1err=t1err+1)
  }
  results <- (t1err/n)*100
  cat("When theta is", theta, ", number of rejection is",(t1err/n)*100,"% in Gap test\n")
}

set.seed(SEED)
for (theta in theta_values) {
  numRej1(1000,theta)
}

permuteTest = function(data, k) {
  permute <- 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_count <- permute(data, k)
  prob <- rep(1/factorial(k), factorial(k))
  chisq.test(permute_count, p = prob)
}

numRej2 <- function(n,theta) {
  t1err = 0
  for (i in 1:n) {
    set.seed(1 + i)
    data <- arima.sim(model = list(order = c(2,0,0), ar = c(theta, theta)), n = 300, sd = 1)
    if (permuteTest(data, 3)$p.value <= 0.05) {
      t1err = t1err + 1
    }
  }
  results <- (t1err/n)*100
  cat("When theta is", theta, ", number of rejection is",(t1err/n)*100,"% in Permutation test\n")
}

set.seed(SEED)
for (theta in theta_values) {
  numRej2(1000 , theta)
}


numRej3 <- function(n,theta) {
  t1err=0
  for (i in 1:n) {
    data<-arima.sim(model = list(order = c(2,0,0), ar = c(theta, theta)), n = 100, sd = 1)
    x=runs.test(data)
    if ((x$p.value) <= 0.05) (t1err=t1err+1)
  }
  cat("When theta is", theta, ", number of rejection is",(t1err/n)*100,"% in Run test\n")
}

set.seed(SEED)
for (theta in theta_values) {
  numRej3(1000, theta)
}
BoxMuller <- function(n){
  u1 <- runif(n)
  u2 <- runif(n)
  theta <- 2 * pi * u1
  e <- -log(u2)
  r <- sqrt(2 * e)

  x <- r * cos(theta)
  y <- r * sin(theta)
  res <- cbind(x, y)
  return(res)
}

BoxMuller1 <- function(n){
  res <- BoxMuller(n)
  return(res[,1])
}
set.seed(SEED)
box_sample <- BoxMuller(10000)

par(mfrow = c(1,2))
hist(box_sample[,"x"], main = "Box-Muller (10,000 runs)", xlab = "x")
hist(box_sample[,"y"], main = "Box-Muller (10,000 runs)", xlab = "y")
par(mfrow = c(1,1))
plot(box_sample[,"x"], box_sample[,"y"],
     main = "Scatter Plot of Box-Muller", xlab = "x", ylab = "y")
Polar <- function(n){
  v1 <- runif(2*n, min = -1, max = 1)
  v2 <- runif(2*n, min = -1, max = 1)
  w <- v1^2 + v2^2
  v1 <- v1[w < 1][1:n]
  v2 <- v2[w < 1][1:n]
  w <- w[w < 1][1:n]

  c <- sqrt((-2)*log(w) / w)
  x <- c*v1
  y <- c*v2
  return(cbind(x, y))
}

Polar1 <- function(n){
  x <- Polar(n)[, "x"]
  return(x)
}
set.seed(SEED)
polar_sample <- Polar(10000)

par(mfrow = c(1,2))
hist(polar_sample[,"x"], main = "Polar Method (10,000 runs)", xlab = "x")
hist(polar_sample[,"y"], main = "Polar Method (10,000 runs)", xlab = "y")
par(mfrow = c(1,1))
plot(polar_sample[,"x"], polar_sample[,"y"],
     main = "Scatter Plot of Polar Method", xlab = "x", ylab = "y")
RatioUnif <- function(n){
  u1 <- runif(10*n)
  u2 <- runif(10*n)
  v <- sqrt(2*exp(1))*(2*u2 - 1)
  x <- v / u1
  z <- (x^2) / 4
  x <- x[(z <= ((0.259 / u1) + 0.35)) & (z <= ((-1)*log(u1)))][1:n]
  return(x)
}
set.seed(SEED)
ratiounif_sample <- RatioUnif(10000)

par(mfrow = c(1,1))
hist(ratiounif_sample, main = "Ratio of Uniform (10,000 runs)", xlab = "x")
KSTestRep <- function(sim, n = 10000, freq = 1000, dist = "pnorm", alpha = 0.05){
  x <- sapply(1:freq, function(n){sim(n)})
  is_norm <- sapply(x, function(x){ks.test(x, dist)[["p.value"]] >= alpha})
  res <- (table(is_norm, dnn = "reject freq"))
```

```r
  return(res)
}
set.seed(SEED)
reject_tb <- rbind(BoxMuller = KSTestRep(BoxMuller1),
                   Polar = KSTestRep(Polar1),
                   RatioUnif = KSTestRep(RatioUnif))
colnames(reject_tb) <- c("Reject", "NonReject")
kable(reject_tb, caption = "Reject of Generators")
CongGen <- function(n, a, c, m){
  u <- runif(1, max = m)
  res <- c(NULL)
  for(i in 1:n){
    u <- (a*u + c) %% m
    res[i] <- u
  }
  return(res/m)
}

BoxMullerCong <- function(n, a, c, m){
  u1 <- CongGen(n, a, c, m)
  u2 <- CongGen(n, a, c, m)
  theta <- 2 * pi * u1
  e <- -log(u2)
  r <- sqrt(2 * e)
  x <- r * cos(theta)
  y <- r * sin(theta)
  res <- cbind(x, y)
  return(res)
}
set.seed(SEED)
boxcong_sample <- BoxMullerCong(n = 10000, a = 131, c = 0, m = 2^35)
rg <- round(range(boxcong_sample[, "x"]), 1)
cat(sprintf("The range of X is from %s to %s.", rg[1], rg[2]))

generatePoisson <- function(lambda, starting_point) {

  if (starting_point == "mean") {
    i <- lambda
  } else if (starting_point == "0") {
    i <- 0
  } else {
    stop("Invalid starting point. Choose 'mean' or '0'.")
  }

  while (TRUE) {
    U <- runif(1)
    cdf <- ppois(i, lambda)
    if (U >= ppois(i, lambda)) {
      i <- i + 1
      cdf <- ppois(i, lambda)
    } else {
      return(i)
    }
  }
}

set.seed(SEED)
results_from_0 <- replicate(100, generatePoisson(10, "0"))

set.seed(SEED)
results_from_mean <- replicate(100, generatePoisson(10, "mean"))

mean_steps_from_0<- mean(results_from_0)
mean_steps_from_mean <- mean(results_from_mean) - 10

cat("Average number of steps starting from 0:", mean_steps_from_0, "\n")
cat("Average number of steps starting from mean:", mean_steps_from_mean, "\n")
tvVariate <- function(n){
  tv <- c()
  while(length(tv) < n){
    u <- runif(1)
    x <- (u < 0.5)*(1/(4*u - 1)) + (u >= 0.5)*(4*u - 3)
    v <- (u < 0.5)*(u/(x^2)) + (u >= 0.5)*(u)
    if ((v < (1 - abs(x)/2)) | (v > (1 + (x^2)/v)^(-(v+1)/2))){
      tv <- c(tv, x)
    }
  }
  return(tv)
}
set.seed(SEED)
tvariate_sample <- tvVariate(10000)
hist(tvariate_sample, main = "t-Variate Method (10,000 runs)")
set.seed(SEED)
reject_tb4 <- rbind(tvVariate = KSTestRep(tvVariate))

colnames(reject_tb4) <- c("Reject", "NonReject")
kable(reject_tb4, caption = "Reject of t-Variate")
stargazer(reject_tb4, type = "text")
A <- matrix(c(1, 0.5, 0.25, 0.125,
            0.5, 1, 0.5, 0.25,
            0.25, 0.5, 1, 0.5,
            0.125, 0.25, 0.5, 1), nrow = 4, byrow=TRUE)

myChol = function(A) {
  m = nrow(A)
  for (i in 1:(m - 1)) {
    A[i, i]=sqrt(A[i, i] - sum(A[0:(i - 1),i]^2))
    for (j in (i + 1):m) {
      A[i, j] = (A[i, j]-sum(A[0:(i-1),i]*A[0:(i - 1),j]))/A[i, i]
    }
  }
  A[m, m] = sqrt(A[m, m] - sum(A[0:(m - 1), m]^2))
  for (j in 1:m - 1) {
    for (i in (j + 1):m) {
      A[i,j] = 0
    }
  }
  return(A)
}

chol(A)
myChol(A)
eigen_values <- eigen(A)$values
eigen_vectors <- eigen(A)$vectors
eigen_vectors %*% diag(eigen_values) %*% t(eigen_vectors)
```

```r
Q <- qr.Q(qr(A))
R <- qr.R(qr(A))
Q %*% R

U <- svd(A)$u
V <- svd(A)$v
D <- svd(A)$d

U %*% diag(D) %*% t(V)
MyAR <- function(x, order){
  data <- x
  data_colname <- c("x", paste0("ar", 1:order))
  data <- cbind(x = data, x_1 = lag(x, -1))
  order <- order - 1

  while (order > 0) {
    data <- cbind(data, lag(data[,ncol(data)], -1))
    order <- order - 1
  }
  colnames(data) <- data_colname
  data <- na.omit(data)

  md <- lm(x ~ ., data = data)

  return(md)
}

data(lynx)

ar1_md <- arima(lynx, order = c(1, 0, 0))
ar2_md <- arima(lynx, order = c(2, 0, 0))
my_ar1 <- MyAR(lynx, 1)
my_ar2 <- MyAR(lynx, 2)
stargazer(ar1_md, ar2_md, my_ar1, my_ar2,
          title = "Result of My AR Model and R's ARIMA Model",
          type = "latex",
          column.sep.width = "-5pt",
          no.space = TRUE,
          column.labels = c("AR(1)", "AR(2)", "AR(1)", "AR(2)"),
          model.numbers = FALSE,
          df = FALSE)
```