HW 6 Random Number Generators & Rshiny

STAT 5400

Due: Oct 11, 2024 9:30 AM

Problems

Submit your solutions as an .Rmd file and accompanying .pdf file. Include all the **relevant** R code and output. Always interpret your result whenever it is necessary.

Problems

1. Generators of normal distributions.

- Write an R function `BoxMuller` that generates a sample of n values from $N(\mu, \sigma^2)$ using the Box-Muller method. This function should have three arguments:
  - `n`: number of observations,
  - `mu`: the value of mean μ,
  - `sigma`: the value of standard deviation σ. This function should return a vector of n normal variates.

```
set.seed(5400)

BoxMuller <- function (n, mu, sigma) {
  U_1 = runif(n/2)
  U_2 = runif(n/2)

  X_1 = sqrt(-2 * log( U_1 )) * cos(2 * pi * U_2)
  X_2 = sqrt(-2 * log( U_1 )) * sin(2 * pi * U_2)

  X_std_norm = c(X_1, X_2)

  X_norm = sigma * X_std_norm + mu

  return(X_norm)

}
```

- Write an R function `PolarMethod` that generates a sample of n values from $N(\mu, \sigma^2)$ using the polar method. This function should have the same arguments and return values as the `BoxMuller` function. When the polar method is used, it might be tricky if you want to control `n` when the accept-reject sampling method is used to generate points in the unit disk. You need a different implementation from the one on the slides.

```
PolarMethod <- function(n, mu, sigma) {

  results = vector()
  for(i in 1:trunc(n/2)) {
    R_2 = 2
    U_1 =0; U_2 = 0
    while(R_2 > 1) {
      U_1 = runif(1, -1, 1)
      U_2 = runif(1, -1, 1)

      R_2 = U_1^2 + U_2^2
      R_2 <- R_2 + .Machine$double.xmin
    }
    Z_1 = U_1 * sqrt(-2 *log(R_2)/R_2)
    Z_2 = U_2 * sqrt(-2 *log(R_2)/R_2)

    results = append(results, Z_1)
    results = append(results, Z_2)
  }
  results = sigma * results + mu
  return(results)
}
```

- Generate two vectors of 30 standard normal variates, by both `BoxMuller` and `PolarMethod`. Run a test to check if the two vectors are from the same distribution. One solution is to use `ks.test` in R to perform a Kolmogorov-Smirnov test.

```
bmDist <- BoxMuller(30, 0, 1)
pmDist <- BoxMuller(30, 0, 1)

ks.test(bmDist, pmDist)
```

```
##
##  Exact two-sample Kolmogorov-Smirnov test
##
## data:  bmDist and pmDist
## D = 0.26667, p-value = 0.2391
## alternative hypothesis: two-sided
```

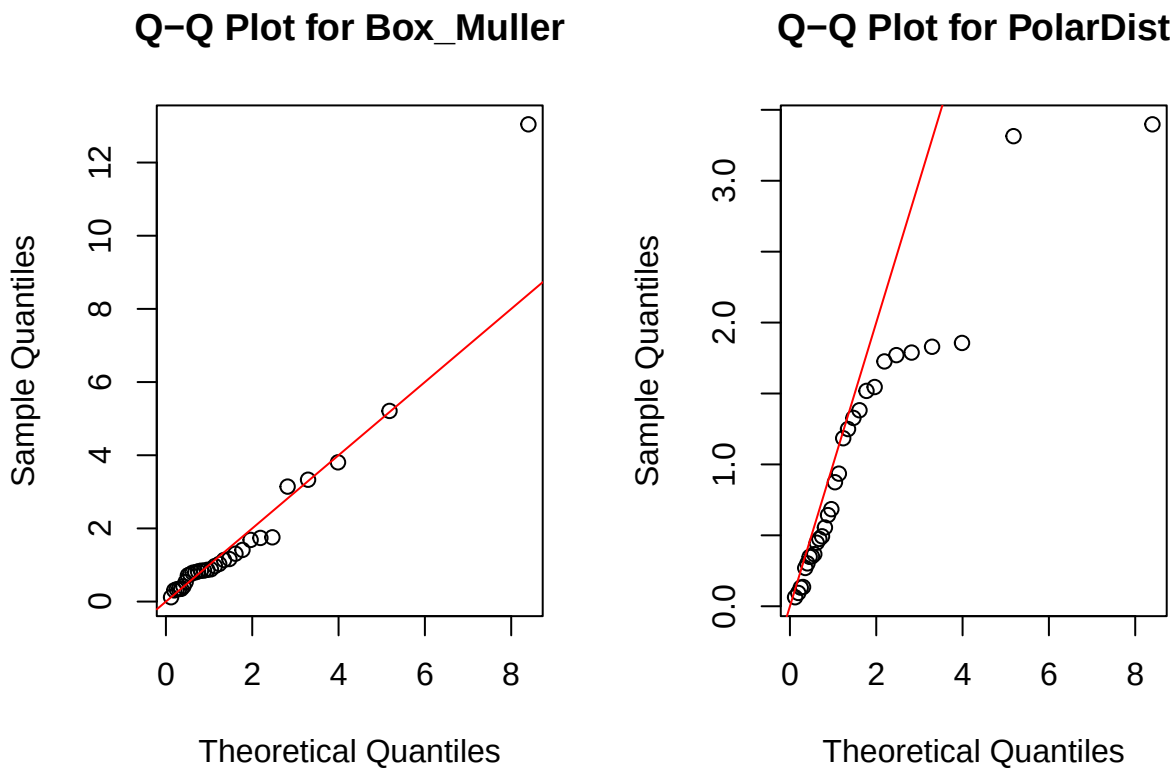
```
Null Hypothesis : The two distributions (bmDist and pmDist) come from the same
underlying population distribution.
Alternative Hypothesis : The two distributions (bmDist and pmDist) come from
different underlying distributions (i.e., at least one point in their cumulative
distribution functions differs).

Test Result:
Since the p-value is greater than the 0.05 significance level, we fail to reject
the null hypothesis.
```

- Generate n values from the log-normal distributions. Note that if X has a normal distribution, then $Y = \exp(X)$ has a log-normal distribution. Thus you may generate log-normal realizations based on the normal variates that you have generated.

```r
lbmDist = exp(bmDist)

lpmDist = exp(pmDist)
```

- Have a Q-Q plot to check whether the sample comforts with the log-normal distribution. You may use `qlnorm` function in R to compute the quantiles of log-normal distributions.

```r
par(mfrow = c(1,2))

qqplot(qlnorm(ppoints(length(lbmDist)), meanlog = 0, sdlog = 1),
       lbmDist,
       main = "Q-Q Plot for Box_Muller",
       xlab = "Theoretical Quantiles",
       ylab = "Sample Quantiles")
abline(0, 1, col = "red") # Add a reference line


qqplot(qlnorm(ppoints(length(lpmDist)), meanlog = 0, sdlog = 1),
       lpmDist,
       main = "Q-Q Plot for PolarDist",
       xlab = "Theoretical Quantiles",
       ylab = "Sample Quantiles")
abline(0, 1, col = "red") # Add a reference line
```




```r
par(mfrow = c(1,1))
```

2. Generators of multivariate normal distributions.

- Write an R function called `mymvrnorm` that generates n random observations from $\mathrm{N}_p(\mu, \Sigma)$. Only calls to R's standard uniform generators `runif` are permitted. This function should have three arguments:
  - `n`, the random sample size,
  - `mu`, the mean vector with p entries,
  - `Sigma`, the variance-covariance matrix, which is symmetric and positive semi-definite.
- This function should return a matrix with n rows and p columns, where the ith row has the realization of Y_i. The `eigen` function should be called in your definition of `mymvrnorm`.

```r
mymvrnorm <- function(n, mu, Sigma) {
  # Get the dimension (p) from the length of the mean vector
  p <- length(mu)

  # Eigen decomposition of the covariance matrix Sigma
  eig <- eigen(Sigma)
  V <- eig$vectors # Matrix of eigenvectors
  D <- diag(sqrt(eig$values)) # Square root of the eigenvalues

  # Compute the square root of Sigma using eigen decomposition
  Sigma.sqrt <- V %*% D %*% t(V)

  # Generate an n × p random matrix Z with i.i.d. N(0, 1) entries
  Z <- matrix(rnorm(n * p), n, p)

  # Generate the matrix X, where each row is an i.i.d. realization from N(mu, Sigma)
  X <- tcrossprod(rep(1, n), mu) + Z %*% Sigma.sqrt

  # Return the resulting matrix
  return(X)
}
```

- Generate 200 random observations from the 3-dimensional multivariate normal distribution having mean vector $\mu = (0, 1, 2)$ and covariance matrix

$$\Sigma = \begin{bmatrix} 1.0 & -0.5 & 0.5 \\ -0.5 & 1.0 & -0.5 \\ 0.5 & -0.5 & 1.0 \end{bmatrix}$$

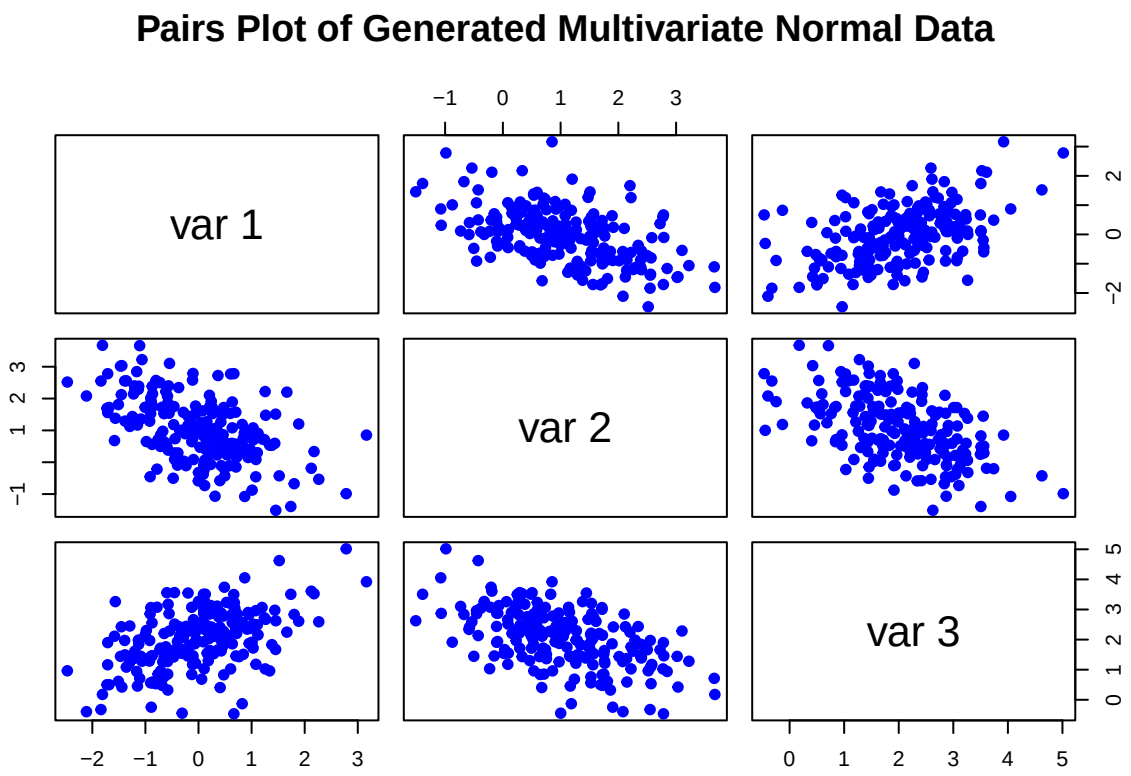
  using your `mymvrnorm` function.

```r
mu <- c(0, 1, 2)
Sigma <- matrix(c(1.0, -0.5, 0.5,
                  -0.5, 1.0, -0.5,
                  0.5, -0.5, 1.0),
                nrow = 3, byrow = TRUE)

# Number of observations
n <- 200

# Generate the random observations using the mymvrnorm function
set.seed(123) # For reproducibility
X <- mymvrnorm(n, mu, Sigma)

# Display the first few rows of the generated matrix
head(X)
```

```
##              [,1]      [,2]      [,3]
## [1,] -1.06402340 3.2225010 1.2802808
## [2,] -0.80180591 2.5670620 0.5345928
## [3,]  1.38244798 0.5322394 1.8314400
## [4,] -0.06835414 1.5023073 1.8613948
## [5,]  0.37763906 0.4207986 2.7604725
## [6,]  1.34019369 0.5357830 0.9603470
```

- Use the R `pairs` plot to graph an array of scatter plots for each pair of variables. For each pair of variables, (visually) check that the location and correlation approximately agree with the theoretical parameters of the corresponding bivariate normal distribution.

```
pairs(X, main = "Pairs Plot of Generated Multivariate Normal Data",
      pch = 19, col = "blue")
```



3. Truncated normal Here we consider sampling the $\mathrm{N}(0,1)$ distribution truncated to the interval $[a,\infty)$ where $a > 0$. Inverting the truncated CDF leads to the formula $X = g_1(U) \equiv \Phi^{-1}(\Phi(a) + (1-\Phi(a))U)$ where $U \sim \mathbf{U}(0,1)$.

- Now find the smallest integer $a \geqslant 1$ for which $X = g_1(U)$ fails to work. For concreteness we can define failure as delivering at least one NaN or $\pm\infty$ when tested with $U \in \{(i-1/2)/1000 \mid, i = 1,2,\ldots,1000\}$.
- Consider $X = g_2(U) \equiv -\Phi^{-1}(\Phi(-a)(1-U))$ that also generates X from the same distribution. Find the smallest integer $a \geqslant 1$ at which g_2 fails using the same criterion as for g_1.

```
library(stats)

# Function to test g1
```

```r
test_g1 <- function(a, U_vals) {
  g1_vals <- qnorm(pnorm(a) + (1 - pnorm(a)) * U_vals)
  any(is.na(g1_vals) | is.infinite(g1_vals))
}

# Function to test g2
test_g2 <- function(a, U_vals) {
  g2_vals <- -qnorm(pnorm(-a) * (1 - U_vals))
  any(is.na(g2_vals) | is.infinite(g2_vals))
}

# Generate U values
U_vals <- (1:1000 - 0.5) / 1000

# Find the smallest a for which g1 fails
a_g1 <- 1
while (!test_g1(a_g1, U_vals)) {
  a_g1 <- a_g1 + 1
}

# Find the smallest a for which g2 fails
a_g2 <- 1
while (!test_g2(a_g2, U_vals)) {
  a_g2 <- a_g2 + 1
}

# Output the results
cat("The smallest a for which g1 fails is:", a_g1, "\n")
```

```
## The smallest a for which g1 fails is: 8
```

```r
cat("The smallest a for which g2 fails is:", a_g2, "\n")
```

```
## The smallest a for which g2 fails is: 38
```

4. Negative binomial distribution The negative binomial distribution with parameters $r \in \{1, 2, \ldots\}$ and $p \in (0, 1)$, denoted $\text{Negbin}(r, p)$, has probability mass function

$$p_k = \mathbb{P}(X = k) = \binom{k+r-1}{r-1} p^r (1-p)^k, \quad k = 0, 1, \ldots$$

It describes the number of failures before the r' th success in a sequence of independent Bernoulli trials with success probability p. For $r = 1$, it reduces to the geometric distribution. The negative binomial distribution has the following compound representation: $X \sim \text{Poi}(\lambda)$ for $\lambda \sim \text{Gam}(r) \times (1-p)/p$. We don't need r to be an integer. Writing

$$\binom{k+r-1}{r-1} = \frac{(k+r-1)!}{(r-1)!k!} = \frac{\Gamma(k+r)}{\Gamma(r)\Gamma(k+1)}$$

yields a valid distribution

$$p_k = \mathbb{P}(X = k) = \frac{\Gamma(k+r)}{\Gamma(r)\Gamma(k+1)} p^r (1-p)^k$$

for $k \in \{0, 1, 2, \ldots\}$ for any real $r > 0$. The Poisson-gamma mixture representation also holds for $r > 0$. Using the compound representation we find that $\mathbb{E}(X) = r(1-p)/p$ and $\text{Var}(X) = r(1-p)/p^2$.

Generate 1000 variables following negative binomial distribution through the compound representation. Show that the sample mean and sample variance are close to the population mean and variance.

```
r <- 5
p <- 0.3

lambda <- rgamma(1000, shape = r, rate = (1 - p) / p)  # Gamma distribution
X <- rpois(1000, lambda)  # Poisson distribution

# Calculate sample mean and variance
sample_mean <- mean(X)
sample_variance <- var(X)

# Calculate theoretical mean and variance
theoretical_mean <- r * (1 - p) / p
theoretical_variance <- r * (1 - p) / p^2

# Output the results
cat("Sample Mean:", sample_mean, "\n")
```

```
## Sample Mean: 2.061
```

```
cat("Theoretical Mean:", theoretical_mean, "\n")
```

```
## Theoretical Mean: 11.66667
```

```
cat("Sample Variance:", sample_variance, "\n")
```

```
## Sample Variance: 2.972251
```

```
cat("Theoretical Variance:", theoretical_variance, "\n")
```

```
## Theoretical Variance: 38.88889
```

5. Rshiny The dataset in counties.rds contains the name of each county in the United States, the total population of the county and the percent of residents in the county who are White, Black, Hispanic, or Asian. During the presentation, we use the percent of residents in the county. In the homework, we hope that you could establish the census app by filling the missing code.

Link of counties.rds: https://shiny.rstudio.com/tutorial/written-tutorial/lesson5/census-app/data/counties.rds

Link of helper.R: https://shiny.rstudio.com/tutorial/written-tutorial/lesson5/census-app/helpers.R

Instructional Codes:

```
# Load packages
library(shiny)
library(maps)
library(mapproj)

# Load data
```

```
counties <- readRDS("censusapp/data/counties.rds")

# Source helper functions
source("helpers.R")

# User interface
ui <- fluidPage(
  titlePanel("censusVis"),

  sidebarLayout(
    sidebarPanel(
      helpText("Create demographic maps with
        information from the 2010 US Census."),

      selectInput("var",
                  label = "Choose a variable to display",
                  choices =  <...missing code...>,
                  selected = "Percent White"),

      sliderInput("range",
                  label = "Range of interest:",
                  min = 0, max = 100, value = c(0, 100))
    ),

    mainPanel(plotOutput("map"))
  )
)

# Server logic
server <- function(input, output) {
  output$map <- renderPlot({
    data <- switch(input$var,
                   <...missing code...>)

    color <- switch(input$var,
                    <...missing code...>)

    legend <- switch(input$var,
                     <...missing code...>)

    percent_map(data, color, legend, input$range[1], input$range[2])
  })
}

# Run app
shinyApp(ui, server)
```

Solution:

```
# Load packages
library(shiny)
library(maps)
library(mapproj)
# Load data
```

```
counties <- readRDS("C:\\Moiyyad\\STAT5400 STAT Computing\\hw6\\counties.rds")
#counties <- readRDS(file.choose())

# Source helper functions
#source("helpers.R")
source("C:\\Moiyyad\\STAT5400 STAT Computing\\hw6\\helpers.R")
# User interface
ui <- fluidPage(
  titlePanel("censusVis"),
  sidebarLayout(
    sidebarPanel(
      helpText("Create demographic maps with
information from the 2010 US Census."),
      selectInput("var",
                  label = "Choose a variable to display",
                  choices = list('Percent White' = 'white',
                                 'Percent Black' = 'black',
                                 'Percent Hispanic' = 'hispanic',
                                 'Percent Asian' = 'asian'
                  ),
                  selected = "Percent White"),
      sliderInput("range",
                  label = "Range of interest:",
                  min = 0, max = 100, value = c(0, 100))
    ),
    mainPanel(plotOutput('map'))
  )
)
# Server logic
server <- function(input, output) {
  output$map <- renderPlot({
    data <- switch(input$var,
                   'white' = counties$white,
                   'black' = counties$black,
                   'hispanic' = counties$hispanic,
                   'asian' = counties$asian);
    color <- switch(input$var,
                    'white' = "blue",
                    'black' = "green",
                    'hispanic' = "red",
                    'asian' = "purple")
    legend <- switch(input$var,
                     'white' = "Percent White",
                     'black' = "Percent Black",
                     'hispanic' = "Percent Hispanic",
                     'asian' = "Percent Asian")
                    #, <...missing code...>)
    percent_map(data, color, legend, input$range[1], input$range[2])
  })
}
# Run app
shinyApp(ui, server)
```