# Elias Washor Homework 6

Elias Washor

2024-10-10

## Q1)

```r
### 1-a
BoxMuller <- function(n, mu, sigma) {
  U1 <- runif(n,0,1)
  U2 <- runif(n,0,1)
  X1 <- sqrt(-2 * log(U1)) * cos(2*pi*U2)
  s <- X1 * sigma + mu
  return(s)
}

#(V <- BoxMuller(50, 10, 3))
# c(mean(V), sd(V))

### 1-b POLAR METHOD ###
unit_disk_pts <- function(n) {
  M <- matrix(NA, n, 2)
  for (i in 1:n) {
    theta <- runif(1, max = 2*pi)
    r <- sqrt(runif(1, max = 1))
    x <- r * cos(theta)
    y <- r * sin(theta)
    M[i,] <- c(x,y)
  }
  return (M)
}

### Polar Method
PolarMethod <- function(n, mu, sigma) {
  V <- unit_disk_pts(n)

  Rsq <- V[,1]^2 + V[,2]^2
  Rsq <- Rsq + .Machine$double.xmin
  m <- sqrt(-(2* log(Rsq)) / (Rsq))

  ## choose one of V1, V2 to transform
  X1 <- m * V[,1]
  X1 <- mu + sigma * X1
  return(X1)
}
```

```r
P <- PolarMethod(1000, 10, 3)
c(mean(P), sd(P))
```

```
## [1] 9.954167 2.967103
```

```r
### 1-c Kolmogorov-Smirnov Test
BM <- BoxMuller(30, 10, 3)
PM <- PolarMethod(30, 10, 3)
(K <- ks.test(BM, PM))
```

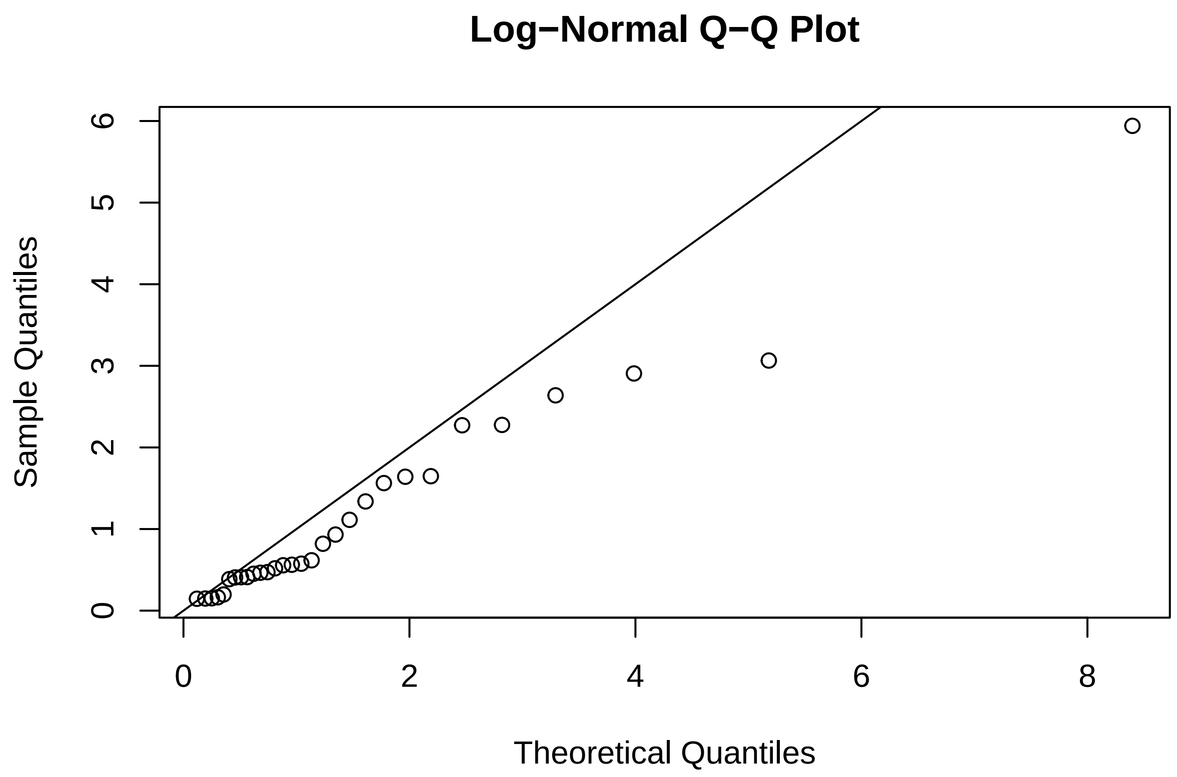
```
##
##  Exact two-sample Kolmogorov-Smirnov test
##
## data:  BM and PM
## D = 0.33333, p-value = 0.07089
## alternative hypothesis: two-sided
```

From the two-sample Kolmogorov-Smirnov test we have two samples, x and y.

$H_0$ : the true distribution of x is equal to the true distribution of y

Since we obtain a p-value of 0.070888 we fail to reject the null hypothesis and conclude that there is not sufficient evidence that the two samples come from different distributions.

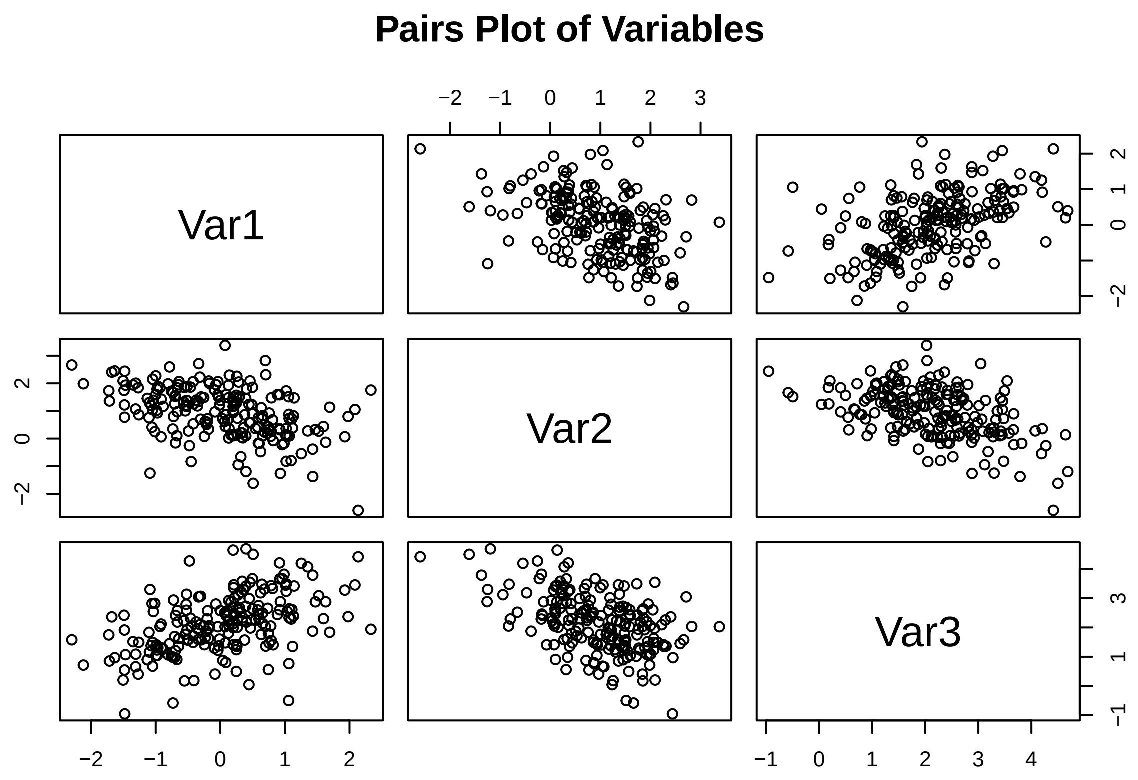
```r
### 1-d

## has log norm dist.
Y1 <- exp(BoxMuller(30,0,1))

## 1-e qqplot
probs <- ppoints(30)
plot(qlnorm(probs), sort(Y1),
     xlab ='Theoretical Quantiles',
     #ylim = c(0,100000),
     ylab ='Sample Quantiles',
     main ='Log-Normal Q-Q Plot'
)

abline(0,1)
```



Based on the Q-Q plot, the sample conforms to the log-normal distribution—as we see most of the points are close to the theoretical quantiles. There is not much deviation from the reference line.

## Q2)

```
mymvrnorm <- function(n, mu, Sigma) {
  rho <- length(mu)

  # Eigen decomposition
  eig <- eigen(Sigma)
  U <- eig$vectors
  D <- diag(sqrt(eig$values))
  Sigma.sqrt <- U %*% tcrossprod(D, U)

  # normal random variables, BoxMuller only calls runif()
  Z <- matrix(BoxMuller(n * rho, 0, 1), n)

  X <- tcrossprod(rep(1,n), mu) + Z %*% Sigma.sqrt
  Xbar <- colMeans(X)
  return(X)
}

cov_m <- matrix(data= c(1, -0.5, 0.5,
                        -0.5, 1, -0.5,
                        0.5, -0.5, 1), 3, 3)

M <- mymvrnorm(200, c(0,1,2), Sigma =cov_m)
MV_df <- as.data.frame(M)

colnames(MV_df) <- (c("Var1","Var2","Var3"))
pairs(MV_df, main='Pairs Plot of Variables' )
```



We know from the Covariance Matrix that the correlation between variables 1 and 3 (with respective means 0 and 2) matches up with the direction from the pairs plot– they have positive directions. The location of each variable matches up with the Mu vector = (0, 1, 2); each variable has a center around their respective mean in the Mu vector.

### Q3)

```
## 3a
g1 <- function(a) {
   U <- c(((1:1000)-0.5)/1000)
   X <- qnorm(pnorm(a) + (1-pnorm(a))* U)
   return(X)
}
## 3b
g2 <- function(a) {
  U <- c(((1:1000)-0.5)/1000)
  X <- -qnorm(pnorm(-a)*(1 - U))
  return(X)
}

###testing ###
#which(is.infinite(g2(37)))

## function to find smallest a
formula_fails <- function(f) {
  a <- 1
  result <- f(a)
  while (! any( is.nan(result)| is.infinite(result))) {
    a <- a + 1
    result <- f(a)
  }
  return(a)}

cat("g1 fails at:", formula_fails(g1), "\ng2 fails at:", formula_fails(g2))
```

```
## g1 fails at: 8
## g2 fails at: 38
```

## Q4)

```
neg_bin <- function(n, r, p) {
  lambda <- rgamma(n, shape = r) * (1-p)/p
  X <- rpois(n,lambda)
  return(X)
}

r <- 3; p <- 0.3;

my_sample <- neg_bin(1000, r, p)
head(my_sample, 10)
```

```
##  [1] 12 14  5  7  0  6  3  6  4  3
```

```
mms <- mean(my_sample)
vms<- var(my_sample)
ppm <- (r*(1-p)/(p))
ppv <- r*(1-p)/p^2

cat("Sample mean:", mms, "\tSample Var: ", vms)
```

```
## Sample mean: 6.915   Sample Var:  21.60538
```

```
cat("\n Pop'n mean:" , ppm, "\t\t Pop'n Var: ", ppv)
```

```
##
##  Pop'n mean: 7         Pop'n Var:  23.33333
```

The sample mean 6.915 from the negative binomial sample I generated was close to the theoretical population mean 7. Also, the sample variance (21.6053804) is close to the population variance 23.3333333.

## Q5 Rshiny)

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

# Load data
counties <- readRDS("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 =  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" = "red",
                    "hispanic" = "darkgreen",
                    "asian" = "purple"
                    )

    legend <- switch(input$var,
                    "white" = "Percent White",
                    "black" = "Percent Black",
                    "hispanic" = "Percent Hispanic",
                    "asian" = "Percent Asian"
                    )

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

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