Lab 3 - Computational Statistics (732A89)

Helena Llorens Lluís (hllor282), Yi Yang (yiyan338)

QUESTION 1: Sampling algorithms for a triangle distribution

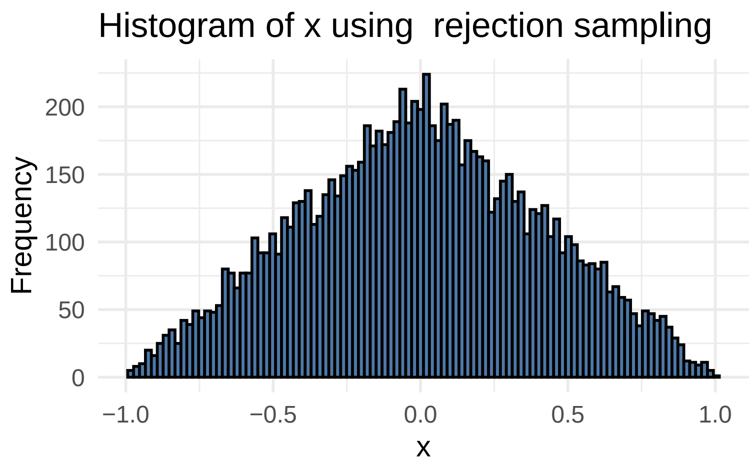
To generate draws of a random variable X with this function:

$$f(x) = \begin{cases} 0, & x < -1 \text{ or } x > 1, \\ x + 1, & -1 \le x \le 0, \\ 1 - x, & 0 < x \le 1. \end{cases}$$

First, we implemented `rejection sampling method`. We choose uniform distribution as the envelope function $e(x)$:

$$e(x) = \begin{cases} 1 & \text{if } -1 \le x \le 1, \\ 0 & \text{otherwise.} \end{cases}$$

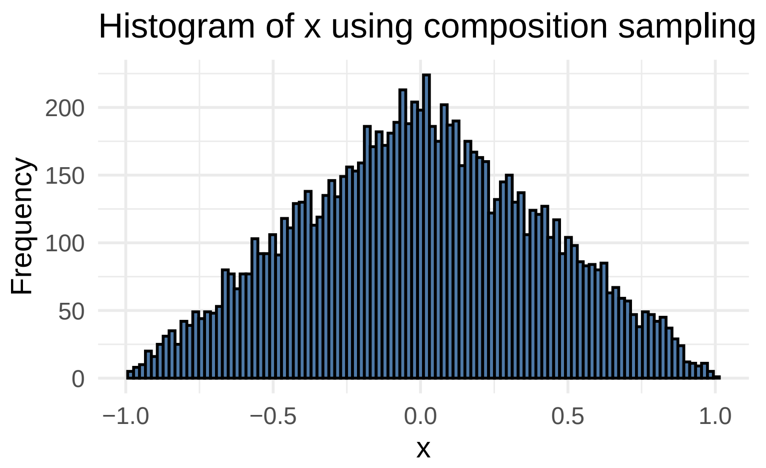
This function always takes the value 1 in the interval[-1,1],which is greater than or eaqual to $f(x)$ for all x. Then, we implemented the rejection sampling algorithm to generate 10,000 random variables. The histogram is shown as follows:



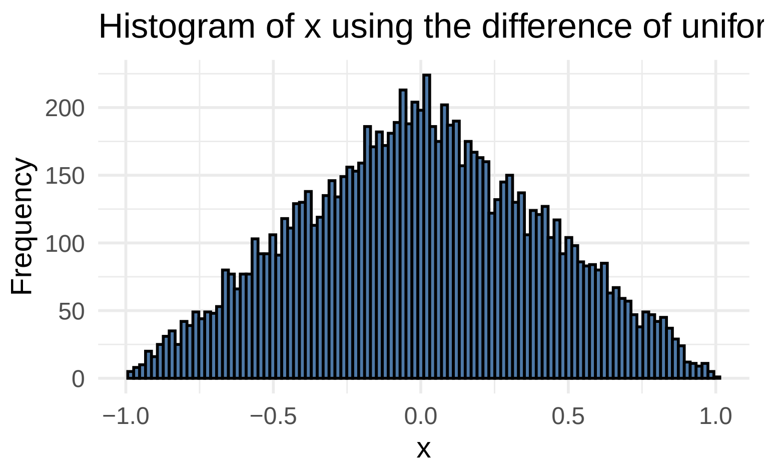
Then, we implemented composition sampling method to generate draws. Let Y be a random variable following this density function:

$$y = \begin{cases} 2 - 2x, & 0 \le x \le 1, \\ 0, & otherwise. \end{cases}$$

$-Y$ has the similar triangle distribution in the interval [-1,0]. The graph of $-Y$ and Y is symmetric about the y-axis. By using `Inverse CDF method`, we generated Y and by the property of symmetry, we could generate $-Y$. The areas under the graphs of Y and $-Y$ are the same, therefore the mixing parameters are both 0.5. Then we could generate 10,000 random variables and the histogram is shown as follows:



When $U1$, $U2$ are two independent `Unif[0,1]` distributed random variables, $U1 - U2$ has the same distribution as X. We used this result and generated random variables and the histogram is shown as follows:



Comparison of Three Sampling Methods

`Rejection Sampling Method` :
1.The amount of programming:
Requires generating random variables for the envelope function and then generating a random number in [0,1], followed by accepting or rejecting samples based on the density function.
2. The number of random value generation and other time consuming:
Two uniform samples are generated for each sample, but samples may be rejected, leading to more random value generation. Additionally, the choice of the envelope function can affect the acceptance rate, resulting in more time-consuming operation.

`Composition Sampling Method`:
1.The amount of programming:
Requires generating Y using `Inverse CDF method`,we should calculate the inverse CDF of Y and then use symmetry to obtain $-Y$ and finally obtain X.
2. The number of random value generation and other time consuming:
Efficient. Only one random variable generation is needed for each sample.Calculating the inverse CDF may lead to additional time consumption.However, it is still more efficient compared to the `Composition Sampling Method`,because it doesn't require generating waste samples.

`Difference of Uniforms`:
1.The amount of programming:
Most efficient. It only requires generating random variables from two uniform distributions and taking their difference.
2. The number of random value generation and other time consuming:
Only two uniform samples generated for each sample.And no need for rejection setps and other time consuming steps.

Based on the analysis above, we choose the third method ***`Difference of Uniforms`*** when generating samples of X. The variance of X is :

```
## Variance of X:  0.1700898
```

QUESTION 2: Bivariate normal and normal mixture distribution

The Box-Muller method generates standard normal random variables from two independent uniform random variables, $U \sim U(0,1)$ and $V \sim U(0,1)$, using the transformations

$$X1 = \sqrt{-2log(U)} \cdot sin(2\pi V)$$
$$X2 = \sqrt{-2log(U)} \cdot cos(2\pi V)$$

These transformations produce two independent standard normal variables, which can then be scaled and shifted to obtain the desired bivariate normal distribution with mean $\mu = (0,0)$ and covariance matrix $\Sigma = \begin{bmatrix} 0.6 & 0 \\ 0 & 0.6 \end{bmatrix}$.

We implemented this method in R and measured the computational time for generating 10000000 random vectors.

To compare efficiency, we also generated the same number of bivariate normal samples using the `rmvnorm` function from the `mvtnorm` package, which is optimized for multivariate normal sampling.

The computation times for both methods are summarized in the table below:

Table 1: Computation time for each method

	Computation time (seconds)
Box-Muller method	3.00
rmvnorm function	1.69

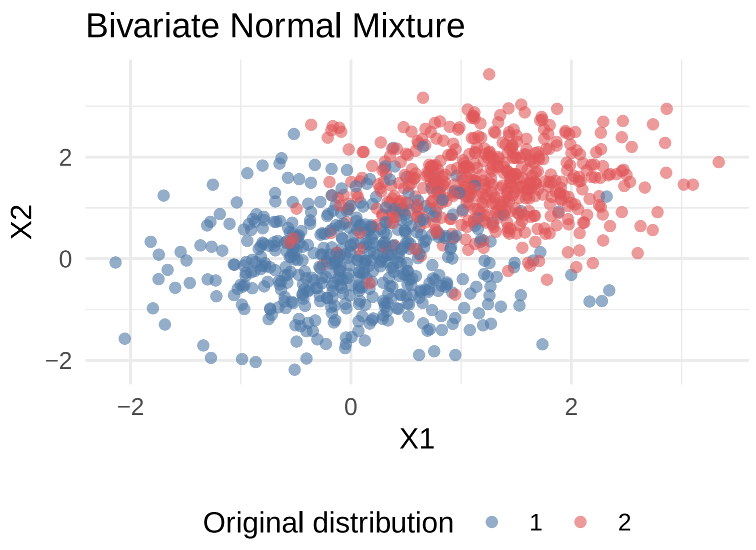
As observed, the `rmvnorm` function is significantly faster than the Box-Muller method. The Box-Muller method requires two uniform random variables per standard normal sample, making it computationally expensive. The `rmvnorm` function, on the other hand, utilizes more efficient matrix decomposition techniques, leading to improved performance.

Now, we consider a bivariate normal mixture model where each observation follows one of two distributions with equal probability (50% each):

$$1.\ N(\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \Sigma = \begin{bmatrix} 0.6 & 0 \\ 0 & 0.6 \end{bmatrix})$$

$$2.\ N(\mu = \begin{bmatrix} 1.5 \\ 1.2 \end{bmatrix}, \Sigma = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix})$$

We generated 1000 random samples from this mixture distribution, assigning each sample to one of the two components based on a random Bernoulli draw. The resulting scatter plot confirms the expected behavior: the data points are drawn from two distinct clusters corresponding to the two normal distributions.



The visualization effectively shows the structure of the mixture, showing two overlapping normal distributions. This confirms that the mixture model is correctly implemented.

Appendix

Question 1

```
#a
library(ggplot2)
fx <- function(x){
  if(x< -1|x > 1){
    return(0)}
  if(-1 <= x&& x <= 0){
    return(x+1)}
  if(0 < x && x <= 1){
    return(1-x)}
}

rejection_sample <- function(n){
  samples <- numeric(n)
  count <- 0

  while (count<n) {
    sample <- runif(1,-1,1) #sampling from uniform distribution
    U <- runif(1)
    f_x <- fx(sample)

    if(U <= f_x){
      count <- count+1
      samples[count] <- sample
    }

  }
  return(samples)

}
set.seed(12345)
x <- rejection_sample(10000)
#hist(x,breaks = 100)
data <- data.frame(x)
ggplot(data,aes(x=x))+
  geom_histogram(bins = 100,fill="#4E79A7",color="black")+
labs(title = "Histogram of x using  rejection sampling", x = "x", y ="Frequency")+ theme_minimal()

#b
com_sample <- function(n){
  U1 <- runif(n/2) #50%
  Y1 <- 1-sqrt(1-U1)

  U2 <- runif(n/2)
  Y2 <- -(1-sqrt(1-U2))

  X <- c(Y1,Y2)
  return(X)

}
set.seed(12345)
composition_samples <- com_sample(10000)
#hist(composition_samples,breaks = 100)
dataCS <- data.frame(composition_samples)
ggplot(dataCS,aes(x=x))+
  geom_histogram(bins = 100,fill="#4E79A7",color="black")+
labs(title = "Histogram of x using composition sampling", x = "x", y ="Frequency")+ theme_minimal()

#c.
set.seed(12345)
U1 <- runif(10000,0,1)
U2 <- runif(10000,0,1)

U3 <- U1-U2
#hist(U3,breaks = 100)
dataDifference <- data.frame(U3)
ggplot(dataDifference,aes(x=x))+
  geom_histogram(bins = 100,fill="#4E79A7",color="black")+
labs(title = "Histogram of x using the difference of uniform distributions", x = "x", y ="Frequency")+

#d
cat("Variance of X: ",var(U3),"\n")
```

Question 2

```r
library(ggplot2)
library(mvtnorm)
library(knitr)

# define n, mu and sigma
n <- 10000000
mu <- c(0, 0)
sigma <- matrix(c(0.6, 0, 0, 0.6), nrow = 2)

# Box Muller method
box_muller_normal <- function(n, mu, sigma){

  # generate U, V as random Uniform
  set.seed(1234)
  u <- runif(n)
  v <- runif(n)

  # Transform to X1 and X2 with the given formula
  x1 <- sqrt(-2*log(u))*cos(2*pi*v)*sqrt(sigma[1, 1]) + mu[1]
  x2 <- sqrt(-2*log(u))*sin(2*pi*v)*sqrt(sigma[2, 2]) + mu[2]

  return(data.frame(x1, x2))
}

# calculate the time to generate n random vectors Box-Muller method
time_box <- system.time({
  dist <- box_muller_normal(n, mu, sigma)
})

# calculate the time to generate n random vectors with the rmvnorm function
set.seed(1234)
time_rmvnorm <- system.time({
  dist2 <- rmvnorm(n, mean = mu, sigma = sigma)
})

# results table
df <- data.frame("Time" = c(time_box["elapsed"], time_rmvnorm["elapsed"]))
rownames(df) <- c("Box-Muller method", "rmvnorm function")
colnames(df) <- c("Time (seconds)")
kable(df, caption = "Computation time for each method")

# set initial distribution paraemters
mu1 <- c(0, 0)
mu2 <- c(1.2, 1.5)
mu <- list(mu1, mu2)

sigma1 <- matrix(c(0.6, 0, 0, 0.6), nrow = 2)
sigma2 <- matrix(c(0.5, 0, 0, 0.5), nrow = 2)
sigma <- list(sigma1, sigma2)

# mixture probabilities
```

```r
prob <- c(0.5, 0.5)

# generate 1000 random vectors from the mixture distribution
n <- 1000
x <- data.frame(matrix(NA, nrow = n, ncol = 3))
set.seed(1234)
for(i in 1:n){
  g <- sample(1:2, 1, replace = TRUE, p = prob)
  x[i, 1:2] <- rmvnorm(n = 1, mean = mu[[g]], sigma = sigma[[g]])
  x[i, 3] <- as.character(g)
}

# plot the mixture distribution
ggplot(x, aes(x = x[, 1], y = x[, 2], color = x[, 3])) +
  geom_point(alpha = 0.6) +
  theme_minimal() +
  scale_color_manual(values = c("#4E79A7", "#E15759")) +
  labs(title = "Bivariate Normal Mixture", x = "X1", y = "X2", color = "Original distribution") +
  theme(legend.position="bottom")
```