Zhao_Yanxiang_Project1

Question 1

```r
##### Question 1 #####
# a) LGM random number generation
runif <- function(n){
  # set seed
  seed <- as.numeric(Sys.time())*1000
  # initiate the random number series
  x <- c(seed)
  # LGM parameters
  m <- 2^31-1
  a <- 7^5
  b <- 0
  # random number generation
  for(t in 1:n){
    x <- c(x, (a*x[t]+b) %% m)
  }
  # drop the seed
  x <- x[-1]
  # scale to U[0,1]
  return(x/m)
}
q1Unif1 <- runif(1000)
# mean and standard deviation
c(mean = mean(q1Unif1), sd = sd(q1Unif1))
```

```
##      mean        sd
## 0.4989403 0.2885661
```

```r
# b) random numbers with build-in function
q1Unif2 <- stats::runif(10000, 0, 1)
c(mean = mean(q1Unif2), sd = sd(q1Unif2))
```

```
##      mean        sd
## 0.4994601 0.2906776
```

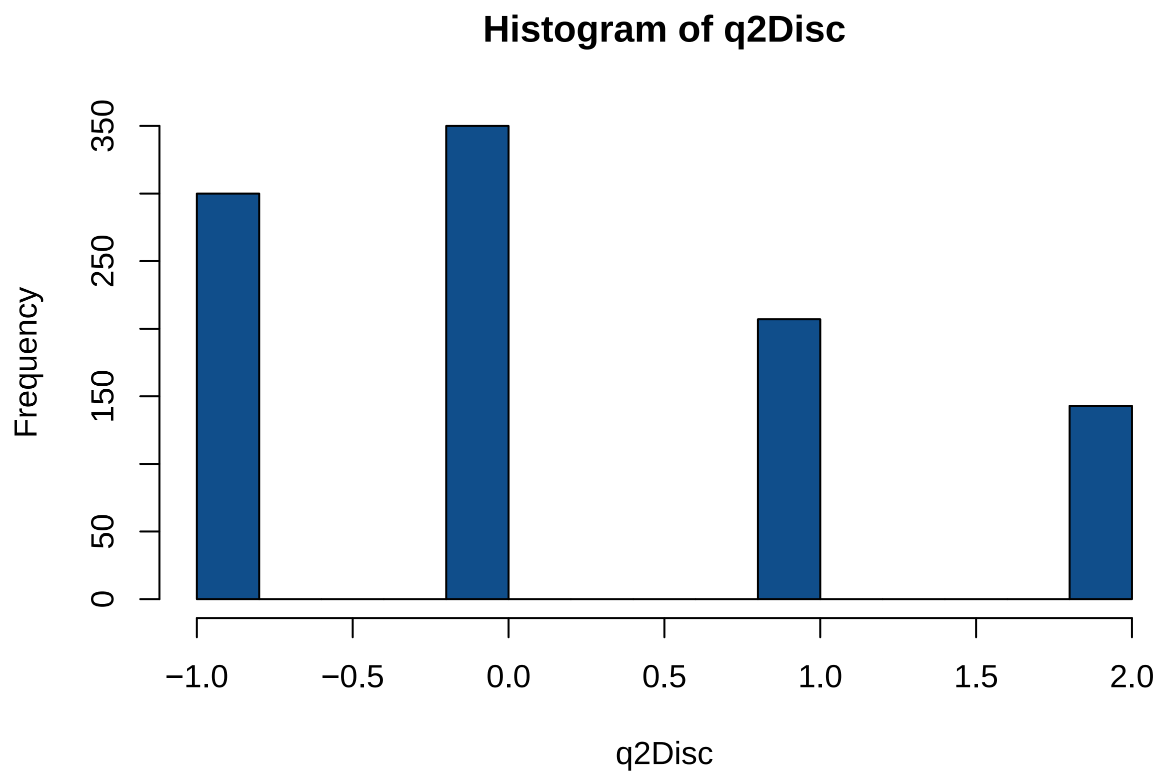
```r
# c) compare
table1 <- cbind(mean = c(mean(q1Unif1), mean(q1Unif2)), sd = c(sd(q1Unif1), sd(q1Unif2)))
rownames(table1) <- c("a)", "b)")
table1
```

```
##         mean        sd
## a) 0.4989403 0.2885661
## b) 0.4994601 0.2906776
```

Comparing with the build-in uniformly distributed random number generator, the LGM method generate a uniform distribution that has similar mean and standard deviation.

Question 2

```r
##### Question 2 #####
# a) generate random discrete variables
# random discrete distribution function
rdisc <- function(rn, val, p){
  # args: rn is a unified randomly distributed series U[0,1],
  #       val is a list of values,
  #       p is a list of probailities, sum to 1
  stopifnot(sum(p)==1) # check if probabilities sum to 1
  stopifnot(length(val)==length(p)) # check if value and prob match
  # generate 0 series
  output <- rep(0, times = length(rn))
  # initial lower-bound
  lb <- 0
  for (i in 1:length(p)){
    # rewrite upper-bound
    ub <- sum(p[1:i])
    # assign discrete value
    output[rn<=ub & rn>lb] <- val[i]
    # rewrite lower-bound
    lb <- ub
  }
  return(output)
}
# use numbers from 1 to generate discrete distribution
values <- c(-1, 0, 1, 2)
prob <- c(0.3, 0.35, 0.2, 0.15)
q2Disc <- rdisc(q1Unif1, values, prob)
# b) draw histogram and compute mean and standard deviation
# histogram
hist(q2Disc, col = "dodgerblue4")
```

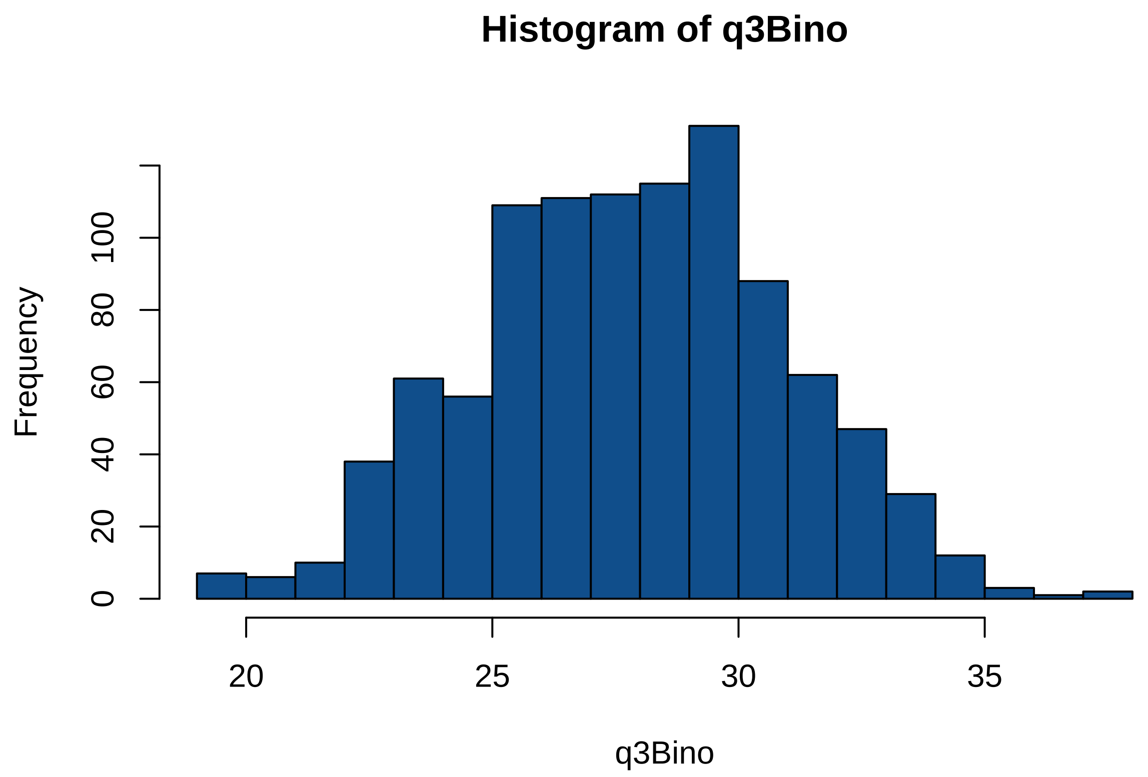


```
# mean and standard deviation
c(mean = mean(q2Disc), sd = sd(q2Disc))
```

```
##     mean       sd
## 0.193000 1.021173
```

Question 3

```
##### Question 3 #####
# a) generate 1000 binomial(n = 44, p = 0.64) random variables
# step 1: generate 44*1000 U[0,1]
q3Unif <- matrix(runif(44000), 44, 1000)
# step 2: generate bernoulli(0.64) column-wise
q3Bern <- apply(q3Unif, 2, rdisc, c(0, 1), c(0.36, 0.64))
# step 3: sum each column to gernerate 1000 binomial(n = 44, p = 0.64) random variables
q3Bino <- apply(q3Bern, 2, sum)
# b) histogram
hist(q3Bino, col = "dodgerblue4", breaks = 20)
```



```
# find P(X>=40)
paste0("P(X>=40) = ", sum(q3Bino>=40)/1000)
```

```
## [1] "P(X>=40) = 0"
```

The exact probability for P(X>= 40) = 0.00004823664. We can expect 0.048 observations to be greater than or equal to 40 in 44000 trials, which is close to 0.

Question 4

```
##### Question 4 #####
# a) generate 10000 exp distributed random number
rexp <- function(n, rate){
  output <- -rate*log(1-runif(n))
  return(output)
}
q4Exp <- rexp(10000, 1.5)
# b) compute P(X>=1) and P(X>=4)
paste0("P(X>=1) = ", sum(q4Exp>=1)/10000)
```

```
## [1] "P(X>=1) = 0.5072"
```

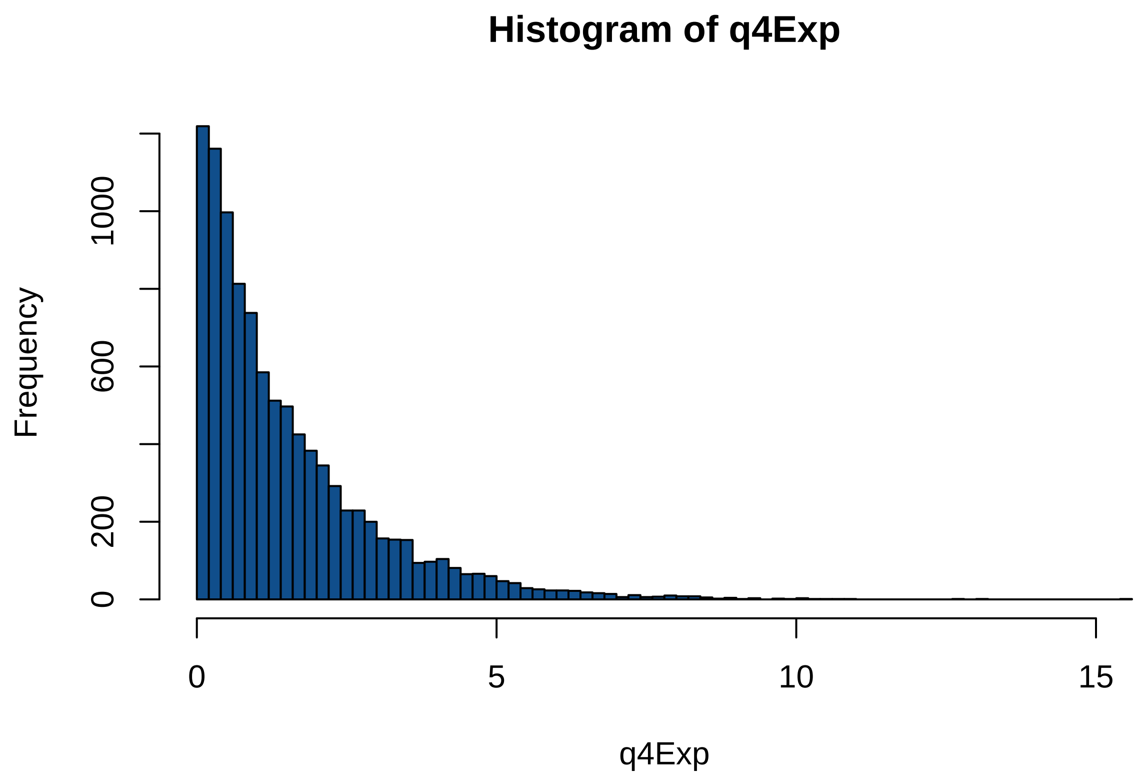
```r
paste0("P(X>=1) = ", sum(q4Exp>=4)/10000)
```

```
## [1] "P(X>=1) = 0.072"
```

```r
# c) compute empirical mean and standard deviation of part (a)
c(mean = mean(q4Exp), sd = sd(q4Exp))
```

```
##     mean       sd
## 1.500398 1.490818
```

```r
# draw the histogram
hist(q4Exp, col = "dodgerblue4", breaks = 100)
```



Question 5

```r
##### Question 5 #####
# a) generate 5000 U[0, 1]
q5Unif <- runif(5000)
# b) N(0,1): Box-Muller method
rnorm_bmm <- function(n){
  m <- n + n %% 2 # to overwirte odd number
  u0 <- runif(m)
  # split into 2 vectors
  u <- split(u0, rep(1:2, m/2))
  # generate N(0,1)
  z1 <- sqrt(-2*log(u[[1]]))*cos(2*pi*u[[2]])
  z2 <- sqrt(-2*log(u[[1]]))*sin(2*pi*u[[2]])
  output <- c(z1, z2)[1:n]
  return(output)
}
q5Norm1 <- rnorm_bmm(5000)
# c) compute empirical mean and standard deviation of part (b)
cat("Box-Muller Method: \n")
```

```
## Box-Muller Method:
```

```r
c(mean = mean(q5Norm1), sd = sd(q5Norm1))
```

```
##       mean         sd
## 0.01643096 0.99576636
```

```r
# d) N(0,1): Polar-Marsaglia
rnorm_pmm <- function(n){
  # define the output vector
  output <- c()
  while(length(output)<n){
    m <- n - length(output)
    u0 <- runif(m)
    u <- split(u0, rep(1:2, m/2))
    # Define v1, v2, w
    vw <- as.data.frame(matrix(c(2*u[[1]]-1, 2*u[[2]]-1), m/2, 2))
    vw <- cbind(vw, vw[,1]^2 + vw[,2]^2)
    # select w <= 1
    vw <- vw[vw[,3]<=1,]
    # generate N(0,1)
    z1 <- vw[,1]*sqrt((-2*log(vw[,3]))/vw[,3])
    z2 <- vw[,2]*sqrt((-2*log(vw[,3]))/vw[,3])

    output <- c(output, z1, z2)
  }
  return(output)
}
q5Norm2 <- rnorm_pmm(5000)
# e) compute empirical mean and standard deviation of part (d)
cat("Polar-Marsaglia Method: \n")
```

```
## Polar-Marsaglia Method:
```

```r
c(mean = mean(q5Norm2), sd = sd(q5Norm2))
```

```
##         mean           sd
## -0.005702229  1.001860972
```

```r
# f) compare run-time efficiency
# Box-Muller
start1 <- Sys.time()
test1 <- rnorm_bmm(20000)
end1 <- Sys.time()
timer1 <- end1 - start1
# Polar-Marsaglia
start2 <- Sys.time()
test2 <- rnorm_bmm(20000)
end2 <- Sys.time()
timer2 <- end2 - start2
# comparison
if (timer1<timer2){
  cat("Box-Muller Method is more efficient!\n")
} else {
  cat("Polar-Marsaglia Method is more efficient!\n")
}
```

```
## Polar-Marsaglia Method is more efficient!
```

```r
# result: Polar-Marsaglia Method is more efficient.
```