Homework 4

M1399.000100, Seoul National University, Spring 2024

Due 23:00 Monday, 2024-06-17

```
library(ggplot2)
```

Append your answer below each question. Submit the modified version of this `Rmd` file and the output `html` file, together with other necessary files such as images and R source code. The submitted version of this `Rmd` file should be `knit` ted to an `html` file ideally identical to the submitted one.

When writing your own R code, do NOT use R packages that implement the functions you are asked to write. i.e., you must write your own code from scratch.

No late submission is accepted.

Q1. Newton-Côtes quadrature

1. Write functions `riemann()`, `trapezoidal()`, and `simpson()`, which evaluates the integral of a given mathematical function taking a single `numeric` argument and returns a `numeric`, using the Riemann rule, the trapezoidal rule, and Simpson's rule, respectively. The three functions should share the following interface. For example, `riemann()` should begin with `riemann <- function(f, a, b, n)` where `f` is the integrand, `a` and `b` are the start and the end points of the interval of integration, and `n` is the number of points in subdivision. In addition, the return value must be the value of the integral.

```
riemann <- function(f, a, b, n) {
    h <- (b-a)/n

    xi <- seq.int(a, b, length.out = n+1)
    xi <- xi[-1]
    xi <- xi[-length(xi)]

    intgrl <- h * (f(a) + sum(f(xi)))

    return(intgrl)
}

trapezoidal <- function(f, a, b, n) {
    h <- (b-a)/n

    xi <- seq.int(a, b, length.out = n+1)
    xi <- xi[-1]
    xi <- xi[-length(xi)]

    intgrl <- h * (0.5 * f(a) + sum(f(xi)) + 0.5 * f(b))

    return(intgrl)
}

simpson <- function(f, a, b, n) {
    h <- (b-a) / n

    xi <- seq.int(a, b, length.out = n+1)
    xi <- xi[-1]
    xi <- xi[-length(xi)]

    intgrl <- (h / 3) *(f(a) + 2*sum(f(xi[seq.int(2, length(xi), 2)])) + 4 * sum(f(xi[se
q.int(1, length(xi), 2)])) + f(b))

    return(intgrl)
}
```

2. Write a function `integral(f, a, b, n, method)` that evaluates the integral of function `f` from `a` to `b` using `n`-point subdivision and numerical integration method `method`. The value of `method` can be either `riemann`, `trapezoidal`, or `simpson`. Your implementation **must not use** `switch`. Instead, use function objects.

```
integral <- function(f, a, b, n, method) {
methods <- list(
    'riemann' = riemann,
    'trapezoidal' = trapezoidal,
    'simpson' = simpson
)

return(methods[[method]](f, a, b, n))
}
```

```
f <- function(x) sin(x)
```

```
integral(f, 0, 1, 100, method = 'riemann')
```

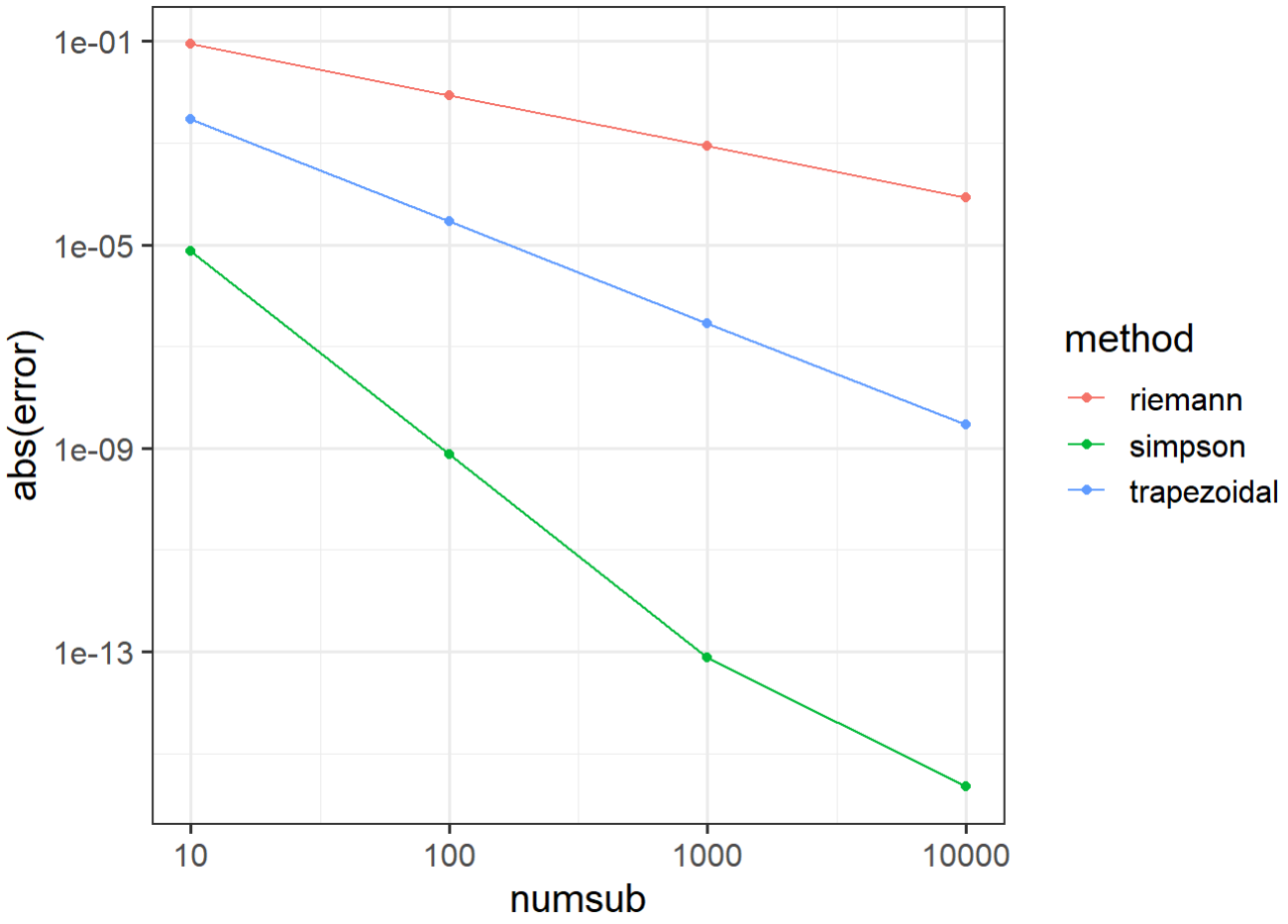
```
## [1] 0.4554865
```

3. Use your function to evaluate $\int_0^2 e^{-x}dx$ and discuss its accuracy. Use the exact value of the integral to evaluate the error.

```
f <- function(x) exp(-x)
truth <- 1-exp(-2)
numsub <- c(10, 100, 1000, 1000, 10000)
err_R <- numeric(length(numsub))
err_T <- numeric(length(numsub))
err_S <- numeric(length(numsub))
for (i in seq(numsub)){
  n <- numsub[i]
  R <- integral(f, 0, 2, n, 'riemann')
  TR <- integral(f, 0, 2, n, 'trapezoidal')
  S <- integral(f, 0, 2, n, 'simpson')

  err_R[i] <- R - truth
  err_T[i] <- TR - truth
  err_S[i] <- S - truth
}
```

```
comp <- data.frame(method = factor(c(rep("riemann", length(numsub)), rep("trapezoidal", length
(numsub)), rep("simpson", length(numsub)))), numsub = c(numsub = rep(numsub, 3)), error = c(err
_R, err_T, err_S))
```

```
ggplot(data = comp, aes(numsub, abs(error))) + geom_point(aes(colour=method)) + geom_line(aes(c
olour=method)) + scale_x_log10() +scale_y_log10() + theme_bw(base_size=15)
```



4. Use your function to evaluate $\int_1^\infty e^{-x}x^{-1/2}dx$. Note the length of the integration interval is infinite. Use the transformation $t = 1/x$ to make the interval finite. Despite of this change of variable, there remains a problem. Specify what it is and how you solved this problem.

```
 f <- function(t) exp(-1/t)*t^(-2/3)
integral(f, 1, 0, 1000, method = "riemann")
```

```
## [1] -0.1905339
```

```
integral(f, 1, 0, 1000, method = "trapezoidal")
```

```
## [1] NaN
```

```
integral(f, 1, 0, 1000, method = "simpson")
```

```
## [1] NaN
```

```
integral(f, 1, 0, 1000, method = "riemann")
```

```
## [1] -0.1905339
```

```
integral(f, 1, 1e-10, 1000, method = "trapezoidal")
```

```
## [1] -0.19035
```

```
  integral(f, 1, 1e-10, 1000, method = "simpson")
```

```
## [1] -0.19035
```

적분구간이 (0,1)이고 `exp(-1/t)` 때문에 적분과정에서 0으로 나누어지는 문제가 발생한다. 따라서 0과 비슷한 매우작은 수로 대치해주면 계산이 가능하다.

Q2. Gauss-Hermite quadrature

Suppose $y_1, \ldots, y_n$ are a random sample from a Poisson distribution with mean $\lambda = \exp(\alpha)$. Suppose the prior on α is $\mathcal{N}(0, 100)$. The observed data are

$$11, 19, 27, 12, 14, 11, 10, 13, 15, 10.$$

Use Gauss-Hermite quadrature to evaluate the mean and variance of the posterior distribution of α. Remember to make a change of variables in the integral, if appropriate, before applying the quadrature formulas. Give some discussion of the accuracy of these calculations. Complete the following R code, using the function `guass.quad()` available in the R package `statmod`.

```
library(statmod)

y <- c(11, 19, 27, 12, 14, 11, 10, 13, 15, 10) # data
ybar <- sum(y)  # sufficient statistic
n <- 10  # sample size
sigma <- 10 # standard deviation of the prior

m <- 40
ghq <- statmod::gauss.quad(m + 1, "hermite")

## Denominator of the posterior
g <- function(alpha, ybar, n)  exp(-exp(alpha)*n + alpha*ybar)

alpha_max <- optimize( function(theta) {g(theta, ybar, n) * exp(-theta^2 / (2*sigma^2))}, c(-
5, 5), maximum = TRUE)$maximum

g_tilde <- function(z, ybar, n, sigma){
  g(z + alpha_max, y, n) * exp(-(z + alpha_max)^2 / 2 / sigma^2 + z^2)
}

denom <- sum(g_tilde(ghq$nodes, ybar, n, sigma) * ghq$weights)
```

```
## Warning in alpha * ybar: 두 객체의 길이가 서로 배수관계에 있지 않습니다
```

```
## Numerator for the 1st moment
## PUT YOUR CODE HERE
numer1 <- sum(g_tilde(ghq$nodes, ybar, n, sigma)*(ghq$nodes + alpha_max)*ghq$weights)
```

```
## Warning in alpha * ybar: 두 객체의 길이가 서로 배수관계에 있지 않습니다
```

```
## Estimate of the posterior mean
posterior_mean <- numer1 / denom

## Numerator for the 2nd moment
## PUT YOUR CODE HERE
numer2 <- sum(g_tilde(ghq$nodes, ybar, n, sigma)*(ghq$nodes + alpha_max)^2 *ghq$weights)
```

```
## Warning in alpha * ybar: 두 객체의 길이가 서로 배수관계에 있지 않습니다
```

```
## Estimate of the posterior variance
posterior_var <- numer2 / denom - (posterior_mean^2)

cat("Posterior mean: ", posterior_mean, "₩n")
```

```
## Posterior mean:  0.3706088
```

```
cat("Posterior variance: ", posterior_var, "₩n")
```

```
## Posterior variance:  0.05735331
```

Q3. Random number generator

1. Write a linear congruential generator function `linearcongruential_gen(m, a, c, seed)` that generates random numbers $u_1, u_2, \ldots$ in the interval $(0, 1)$ using the linear congruential method

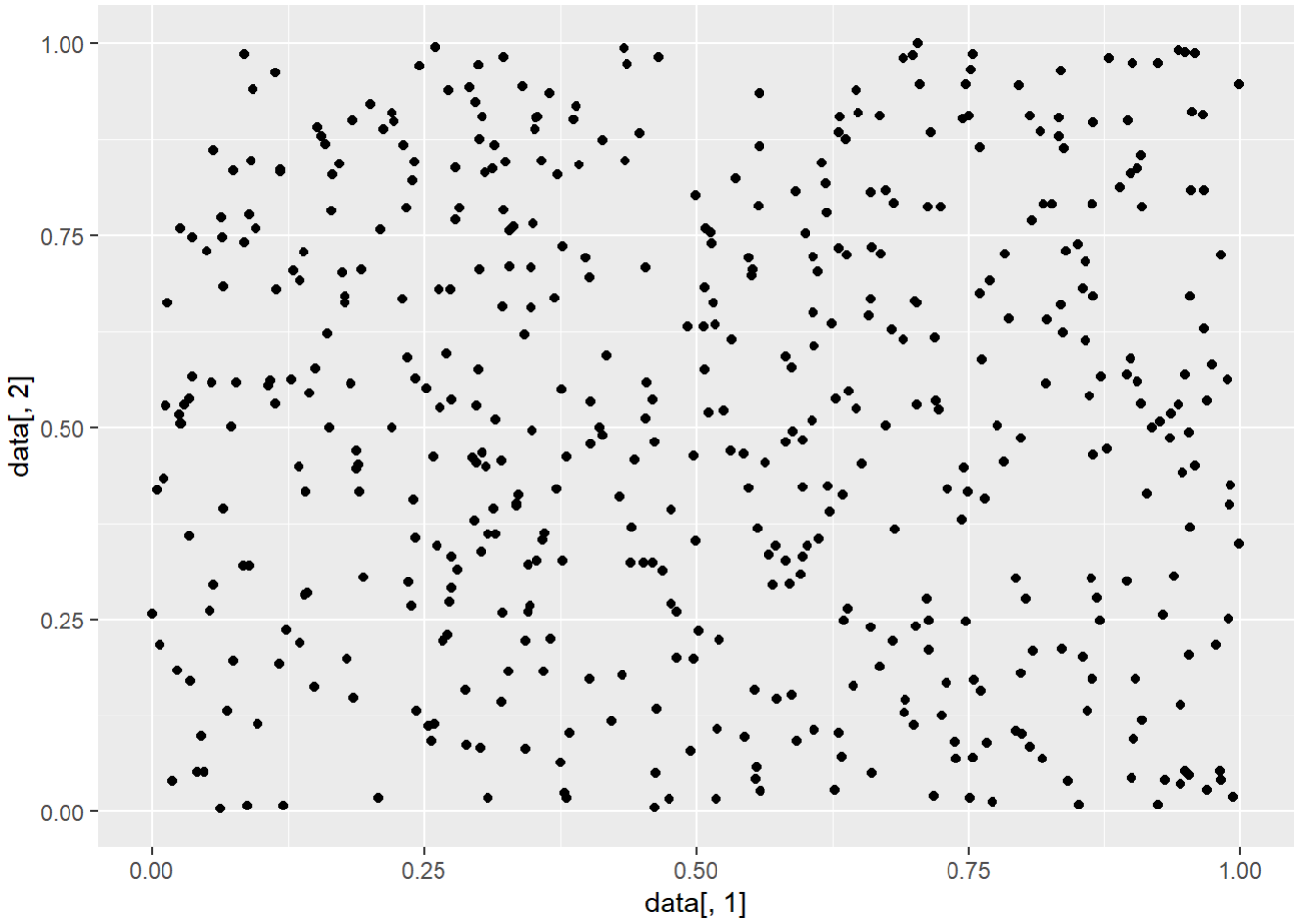
$$x_{i+1} = (ax_i + c) \mod m, \quad i = 1, 2, \ldots$$
$$u_i = x_i/m.$$

   Compare the two random number generators for $m = 2^{32}$, $a = 1103515245$, $c = 12345$, and for $m = 2048$, $a = 1229$, $c = 1$ respectively, in terms of the autocorrelation

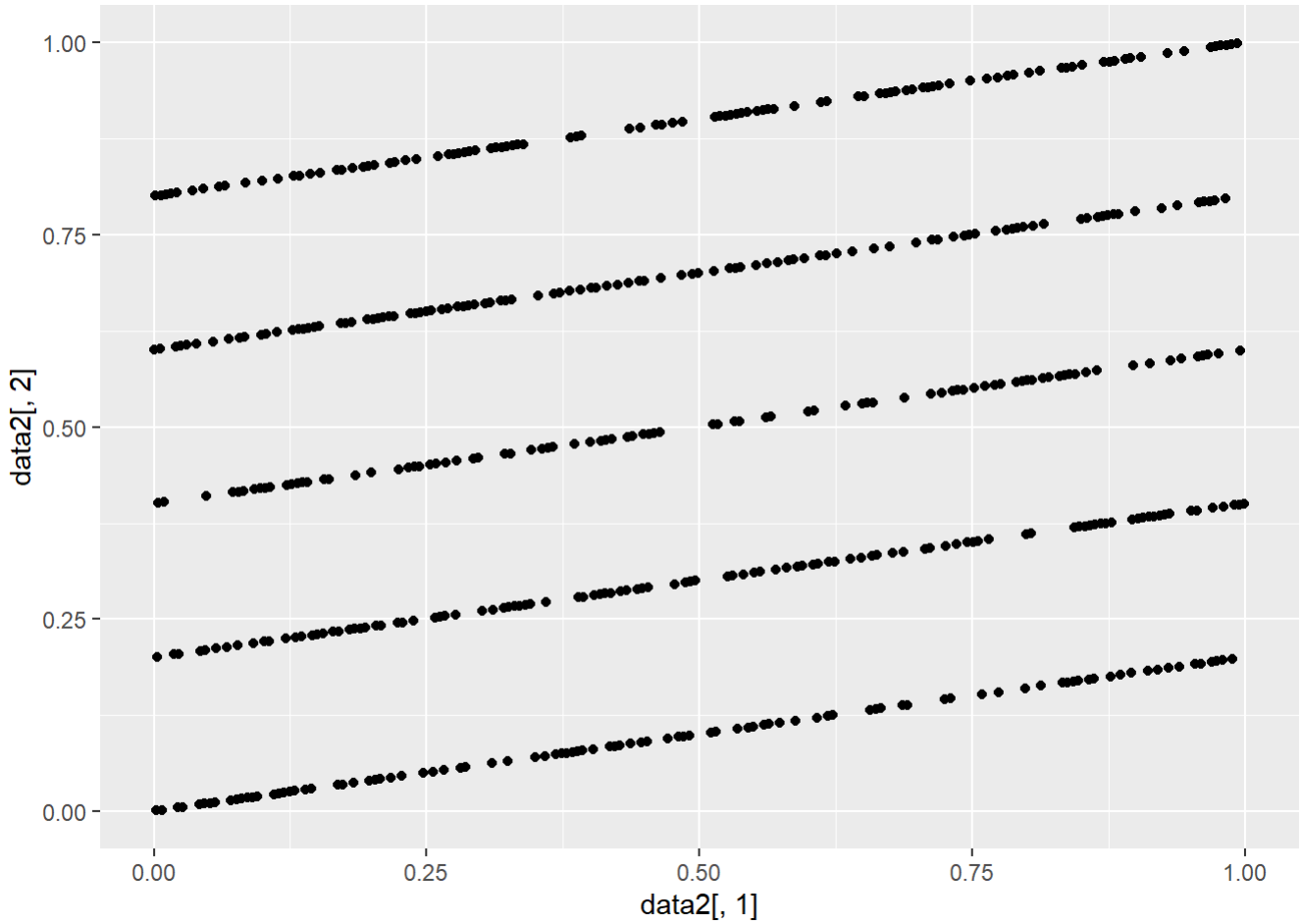
```
#' @param m Modulus
#' @param a Multiplier
#' @param c Intercept
#' @param seed Seed of the RNG
linearcongruential_gen <- function(m, a, c, seed = 1) {
    if (!hasArg(seed)) seed <- (as.numeric(Sys.time()) * 1000) %% m
    ## PUT YOUR CODE HERE
    g <- function(n){
      u <- vector(length = n)
      u[1] <- seed
      for (i in seq_len(n-1)){
        u[i+1] <- (a *u[i] + c) %% m
      }
      seed <<- (a*seed) %% m
      u / m
    }
  return(g)
}
```

```
Randu <- linearcongruential_gen(2^{32}, 1103515245, 12345, seed = 1)
Randu2 <- linearcongruential_gen(2048, 1229, 1, seed = 1)
```

```
u <- Randu(1000)
n <- 1000
data <- matrix(nrow = n / 2, ncol = 2)
data[, 1] <- u[seq(1, n, by =2)]
data[, 2] <- u[seq(2, n, by =2)]

ggplot() + geom_point(aes(x=data[,1], y=data[,2]))
```




```
u2 <- Randu2(1000)

data2 <- matrix(nrow = n / 2, ncol = 2)
data2[, 1] <- u2[seq(1, n, by =2)]
data2[, 2] <- u2[seq(2, n, by =2)]

ggplot() + geom_point(aes(x=data2[,1], y=data2[,2]))
```

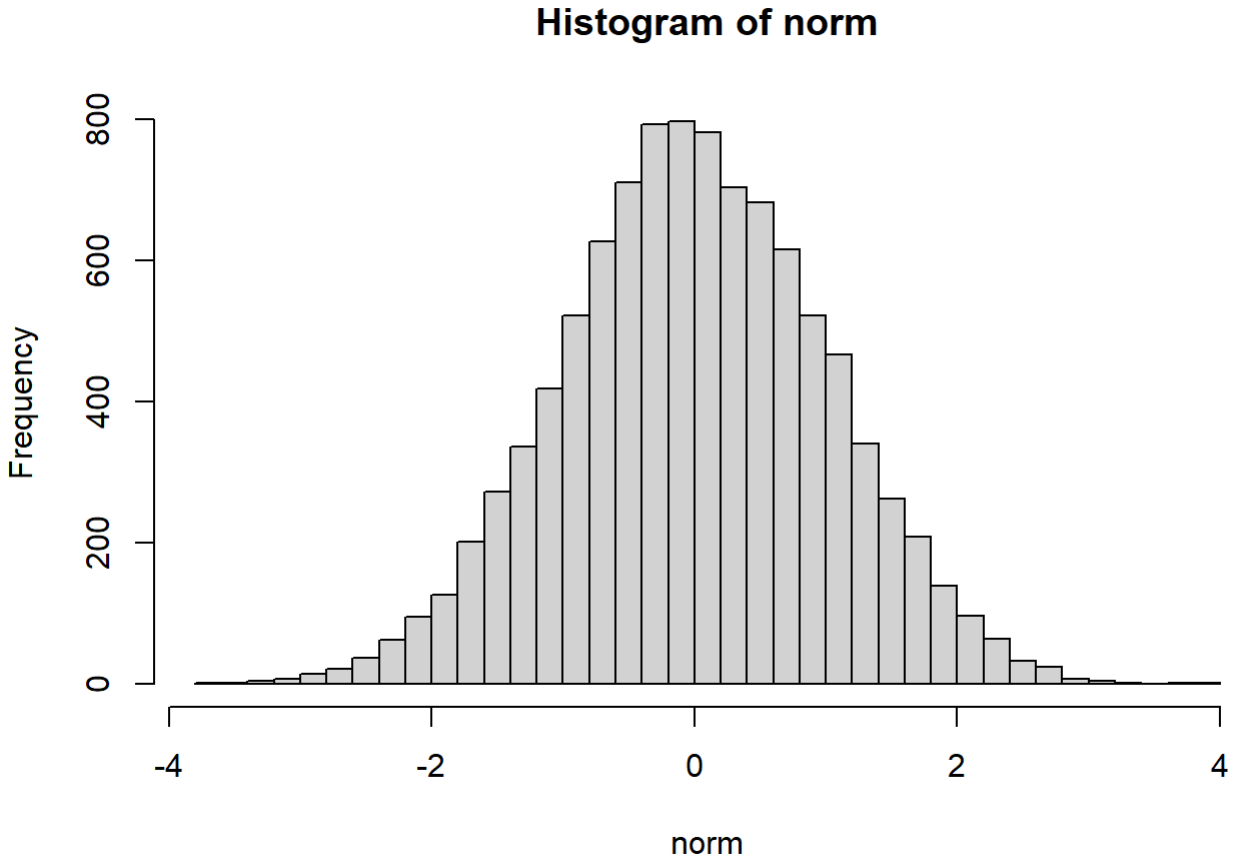


Randu2 에 대해서는 2차원상 그래프에서 1차원으로 모여있는 것으로 보인다. 이는 이전 값을 바탕으로 난수를 생성하기 때문이며, 작은 a와 m에 대해 나타난다.

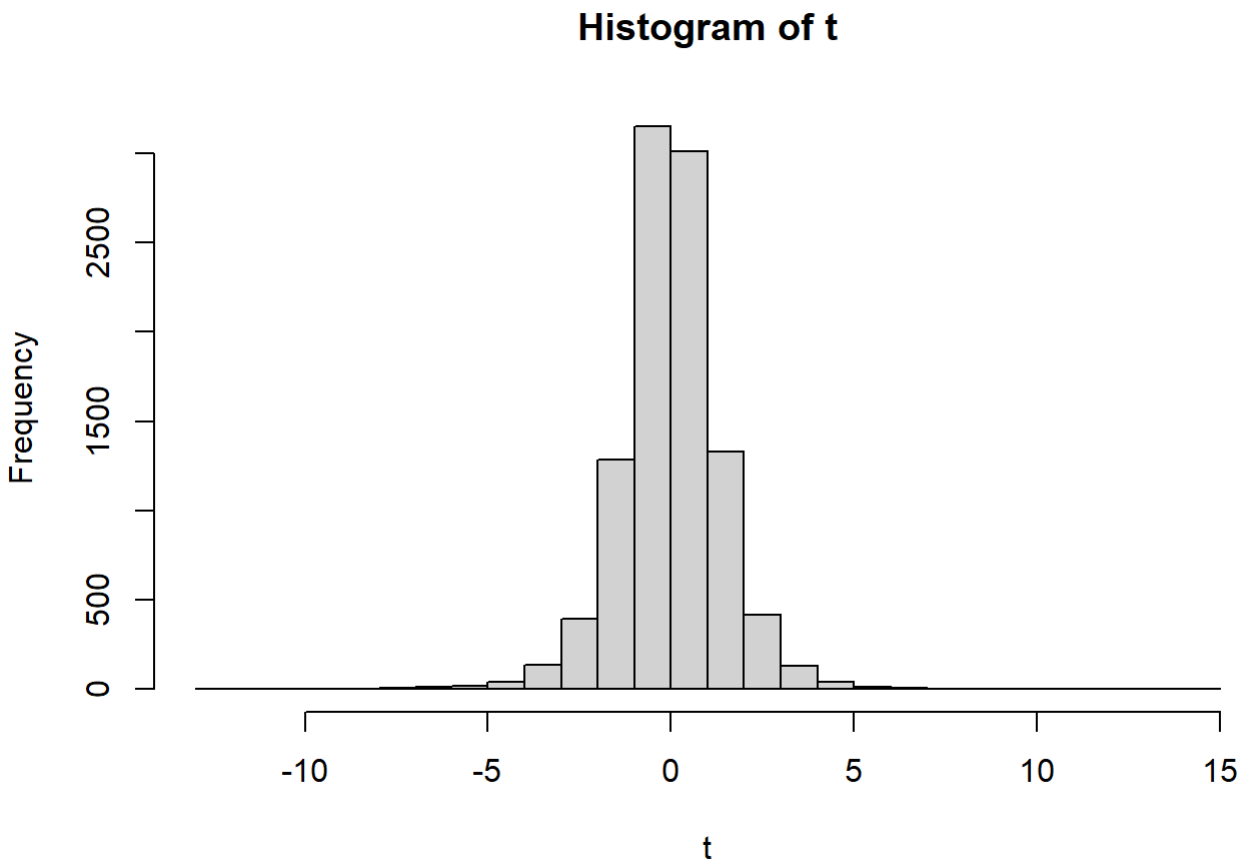
2. In case the inverse CDF F^{-1} is not explicitly available, can you implement the inverse CDF method for random number generation? Assume that the pdf of the distribution exists. Implement your method by completing the following code. Verify your method with standard normal and t distribution with 4 degrees of freedom. (*Hint*. nonlinear equation solve)

```r
#' @param f Density function
#' @param n Sample size
#' @param maxiter Maximum number of iterations
#' @param tol Tolerance for convergence
inverseCDF_numerical <- function(f, n, maxiter=1000, tol=1e-8) {
    Fx <- function(x) integrate(f, -Inf, x)$value
    X <- numeric(n)

    for (i in 1:n) {
      u <- runif(1)
      x <- 0
      iter <- 0
      while (iter < maxiter){
        fx <- Fx(x) - u
        if (abs(fx) < tol) break
        x <- x - fx/f(x)
        iter <- iter +1
      }
    X[i] <- x
    }
  X
}
```

```r
f_norm <- function(x) dnorm(x)
set.seed(1)
norm <- inverseCDF_numerical(f_norm, 10000)
hist(norm, breaks=30)
```



```
f_t <- function(x) dt(x, df=4)
set.seed(1)
t <- inverseCDF_numerical(f_t, 10000)
hist(t, breaks=30)
```



Q4. Acceptance-rejection sampling

As an alternative to the Box-Muller or Marsaglia method, consider the following method of sampling a normal random variable using the absolute value $X = |Z|$ of a standard normal random variable Z.

Q4.

1. $0 \le Z < \infty \rightarrow 0 \le X < \infty$, $X = Z$, $dx = dz$ / $-\infty < z < 0 \rightarrow 0 < x < \infty$, $X = -z$, $dx = -dz$

$$f_z(z) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}z^2} dz$$

$$= \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}z^2} I_{(0 \le z < \infty)} |dz| + \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}z^2} I_{(-\infty < z < 0)} |-dz|$$

$$\Rightarrow \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2} I_{(0 \le x < \infty)} dx + \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2} I_{(0 \le x < \infty)} dx$$

$$f_X(x) = \frac{2}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2} I_{(0 \le x < \infty)}$$

2. $g(x) = e^{-x}$ $x > 0$.

$$\frac{f_X(x)}{g(x)} = \frac{2}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2 + x} = \frac{2}{\sqrt{2\pi}} e^{-\frac{1}{2}(x-1)^2 + \frac{1}{2}} \le c$$

$$c = \frac{2}{\sqrt{2\pi}} e^{\frac{1}{2}}$$

3. Sample a random variable Y distributed according to g
   - Accept Y as a representative of f if $U \le \frac{f_X(Y)}{cg(Y)}$, where U is a uniform random variable on $[0,1]$ drawn independently.
   - otherwise. reject Y. and repeat it.

4. 3번의 과정에서 accept된 값들 중 x 보다 작은 것이 $X = |Z|$ 이고, 같은 확률 분포를 따르므로 그것과 같다.

1. Find the probability density function f_X of X. 2. If g is the probability density function of an exponential random variable with mean 1, find the smallest $c > 0$ such that

$$\frac{f_X(x)}{g(x)} \leq c, \quad \text{for all } x > 0.$$

3. Propose an algorithm that generate X, the absolute value of the standard normal random variable Z.

4. How would you generate Z?

5. Implement your algorithm by completing `myrnorm.R` and test.

```
myrnorm <- function(n) {
    v <- runif(n)

    x <- c()

    for (i in 1:n){
      y <- rexp(1)
     if ( v[i] <= exp((-1/2)*y^2 + y - (1/2)))
      x <- append(x, y)
    }
  z <- x * sample(c(-1,1), length(x), replace = TRUE)
z
}

#### TEST CODE (DO NOT MODIFY THIS PART!!)
set.seed(2023)
my_z <- myrnorm(1000)
# Shapiro-Wilk test
cat("Shapiro-Wilk test for normality of Z: ", shapiro.test(my_z)$p.value, "₩n")
```

```
## Shapiro-Wilk test for normality of Z:  0.9284959
```

6. How efficient is your algorithm?

```
c <- 2/sqrt(2*pi)*exp(1/2)
1/c
```

```
## [1] 0.7601735
```

```
length(my_z)/1000
```

```
## [1] 0.776
```

1000개중 776개가 accept되었고 이는 1/c와 비슷하다.

Q5. Importance sampling

Consider testing the hypotheses $H_0 : \lambda = 2$ versus $H_a : \lambda > 2$ using 25 observations from a Poisson(λ) model. Rote application of the central limit theorem would suggest rejecting H_0 at $\alpha = 0.05$ when $Z > 1.645$, where $Z = \frac{\bar{X}-2}{\sqrt{2/25}}$.

1. Estimate **1) the size of this test** (i.e., the type I error rate) using three Monte Carlo approaches: standard, antithetic, (unstandardized) importance sampling. Denoting the CDF of Poisson(λ) by $F_\lambda(x)$, **2) find the variance of each estimate**. Based on this variance, **3) provide a confidence interval for each estimate**. Discuss the relative merits of each variance reduction technique.
    - For the importance sampling approach, use a Poisson envelope (h in the course notes) with mean equal to the H_0 rejection threshold, namely $\lambda = 2.4653$.
    - Write a function `myMC(method, lam0, nobs, lam, siglevel, nsample, conf)` that evaluates the size of the test using `method`. The value of `method` can be either `standard`, `antithetic`, or `importance`. Your implementation **must not use** `switch`. Instead, use function objects.

```
#' @param lam0 Poisson mean for the null hypothesis
#' @param nobs Observed sample size
#' @param lam Poisson mean for the alternative hypothesis
#' @param siglevel Significance level of the test
#' @param nsample Monte Carlo sample size
#' @param conf Confidence probability for Monte Carlo confidence interval

# Standard Monte Carlo
standard <- function(lam0, nobs, lam=lam0, siglevel=0.05, nsample=1000, conf=0.95) {
    cutoff <- floor((lam0 + sqrt(lam0 / nobs) * qnorm(1 - siglevel)) * nobs)
    y <- rpois(nsample, lam0*nobs)
    lhat <- mean(y > cutoff)
    varhat <- lhat * (1 - lhat) / nsample
    CI <- c(lhat - qnorm((1 - siglevel)/2)*sqrt(varhat), lhat + qnorm((1 - siglevel)/2)*s
qrt(varhat))
    return(list(estimate = lhat, CI = CI))
}

# Antithetic Monte Carlo
antithetic <- function(lam0, nobs, lam=lam0, siglevel=0.05, nsample=1000, conf=0.95) {
    cutoff <- floor((lam0 + sqrt(lam0 / nobs) * qnorm(1 - siglevel)) * nobs)
    y1 <- rpois(nsample/2, lam0*nobs)
    y2 <- lam0*nobs - y1
    y <- c(y1, y2)
    lhat <- mean(y > cutoff)
    varhat <- lhat * (1 - lhat) / nsample
    CI <- c(lhat - qnorm((1 - siglevel)/2)*sqrt(varhat), lhat + qnorm((1 - siglevel)/2)*s
qrt(varhat))
    return(list(estimate = lhat, CI = CI))
}

# Importance-sampled Monte Carlo
importance <- function(lam0, nobs, lam=lam0, siglevel=0.05, nsample=1000, conf=0.95) {
    cutoff <- floor((lam0 + sqrt(lam0 / nobs) * qnorm(1 - siglevel)) * nobs)
    y <- rpois(nsample, lam0*nobs)
    lhat <- mean(( y > cutoff) * dpois(y, lam0*nobs)/dpois(y, 2.4653*nobs))
    varhat <- lhat * (1 - lhat) / nsample
    CI <- c(lhat - qnorm((1 - siglevel)/2)*sqrt(varhat), lhat + qnorm((1 - siglevel)/2)*s
qrt(varhat))
    return(list(estimate = lhat, CI = CI))
}

# Wrapper function for Monte Carlo size calculation
#' @param method Monte Carlo method (standard, antithetic, importance)
myMC <- function(method, lam0, nobs, lam=lam0, siglevel=0.05, nsample=1000, conf=0.95) {
    methods <- list(
    'standard' = standard,
    'antithetic' = antithetic,
    'importance' = importance
)

return(methods[[method]](lam0, nobs, lam=lam0, siglevel=0.05, nsample=1000, conf=0.95))
}
```

```
#### TEST CODE (DO NOT MODIFY THIS PART!!)
set.seed(2023)
lam0 <- 2.0; nobs <- 25; siglevel <- 0.05
cutoff <- floor((lam0 + sqrt(lam0 / nobs) * qnorm(1 - siglevel)) * nobs)
true_size <- ppois(q = cutoff, lambda = nobs * lam0, lower.tail = FALSE)
print(abs(true_size - myMC("standard", lam0, nobs, nsample=1e5)$estimate))
```

```
## [1] 0.009319224
```

```
print(abs(true_size - myMC("antithetic", lam0, nobs, nsample=1e5)$estimate))
```

```
## [1] 0.02668078
```

```
print(abs(true_size - myMC("importance", lam0, nobs, nsample=1e5)$estimate))
```

```
## [1] 0.04961643
```

2. Draw the power curve for this test for $\lambda \in [2.2, 4]$, using the same three techniques. Provide pointwise confidence bands in each case. Discuss the relative merits of each technique in this setting.

   Write a function `myMCpower(lam0, nobs, lam_vec, siglevel, nsample, conf)` that returns a data frame of the power of the test for each value of `lam_vec` for each of the three Monte Carlo methods of Part 1. The data frame must have variables `estimate`, `CI_lower`, `CI_upper` that contain the Monte Carlo estimate of the power and its lower and upper confidence bounds. It must also have `lam`, the corresponding value of λ, and `method` for the used Monte Carlo method. Then the power curve can be drawn based on this data frame.

```
#' @param lam0 Poisson mean for the null hypothesis
#' @param nobs Observed sample size
#' @param lam_vec Poisson mean vector for the alternative hypothesis
#' @param siglevel Significance level of the test
#' @param nsample Monte Carlo sample size
#' @param conf Confidence probability for Monte Carlo confidence interval
myMCpower <- function(lam0, nobs, lam_vec, siglevel=0.05, nsample=1000, conf=0.05) {
    vec_myMC <- Vectorize(myMC, vectorize.args = "lam")
    power_df <- list()
    for (m in c("standard", "antithetic", "importance")) {
        power_lst <- vec_myMC(method = m, lam0, nobs, lam_vec, siglevel, nsample, conf)
        df <- as.data.frame(matrix(unlist(power_lst), ncol = 3, byrow = TRUE))
        colnames(df) <- c("estimate", "CI_upper", "CI_lower")
        df$lam <- lam_vec
        df$method <- m
        power_df <- rbind(power_df, df)
    }
    power_df
}
```

```
lam_vec <- seq(2.2, 4.0, by = 0.2)
nsample <- 100000

set.seed(1)
power_df <- myMCpower(lam0, nobs, lam_vec, siglevel, nsample, conf = 0.95)
power_df <- as.data.frame(power_df)

ggplot(power_df, aes(x = lam, y = estimate, color = method)) +
geom_line()+
labs(title = "Power Curve for Poisson Test",
     x = expression(lambda),
     y = "Power",
     color = "Method",
     fill = "Method")
```