S611 HW5 Woojeong Kim

Q1.

```
library(R6)
 normal_model <- R6Class(
 classname = "normal_model",
 public = list(initialize = function(y = NULL, mu_0 = NULL, tau_0 = NULL, s2_0 = NULL, nu_0 = NULL,
    initial_mu = NULL, initial_s2 = NULL, buffer_length = NULL){
     self <- new.env(parent = parent.frame())
     self$y <- y
     n <- max(rank(self$y))
     self$y_bar <- mean(y)
     self$y_var <- var(y)
     self$initial_mu <- mu_0 #rnorm(1, mean = mu_0, sd = {s2_0}^{1/2})
     self$initial_s2 <- s2_0 #(rgamma(1, shape = nu_0/2, rate = (nu_0 * s2_0)/2))^{-1}
     self$mu <- mu_0
     self$tau <- tau_0
     self$s2 <- s2_0
     self$nu <- nu_0
     self$buffer_length <- buffer_length
     self$buffer_mu <- self$buffer_s2 <- numeric(buffer_length)
     self$buffer_mu <- initial_mu########
     self$buffer_s2 <- initial_s2######
     self$iteration <- 0
     #return(e)
 },
 update = function(self = NULL){
         n <- max(rank(self$y))
         mu <- rnorm(1, mean = (((n * self$y_bar)/ self$s2) + (self$initial_mu/(self$tau)^2))/((n/self$s2) +
             (1/(self$tau)^2)), sd = 1/(n/(self$s2) + 1/(self$tau)^2))
         s2 <- (rgamma(1, shape = (n+ self$nu)/2, rate = ((n-1) * (self$y_var) + n * (self$y_bar - self$mu)^2 +
                                                          self$nu * self$initial_s2)/2))^{-1}
         self$mu <- mu
         self$s2 <- s2
         self$buffer_mu <- self$mu
         self$buffer_s2 <- self$s2
         self$iteration <- self$iteration + 1
         invisible(self)
     }
 )
 )
```

First, we use R6 class to make class “normal_moel”. In initializing function “public”, we initialize the fields y, mu_0, …, initial_s2 and buffer_length as mutable objects. Since they are not determined, the values coming out by applying this function are not specified values. As HW4 Q1, the fields should be labeled to the environment variable “self” from y_var to buffer_s2. The prefix “self” is used in the initializing function of the R6 class to share the variables in the part of initializing and updating. Also, this initializing function contains iteration as 0, which implies counting number in the next updating function. The “public” function in R6 class have fields as variables and the variables in public function can be changed outside of the class environment.

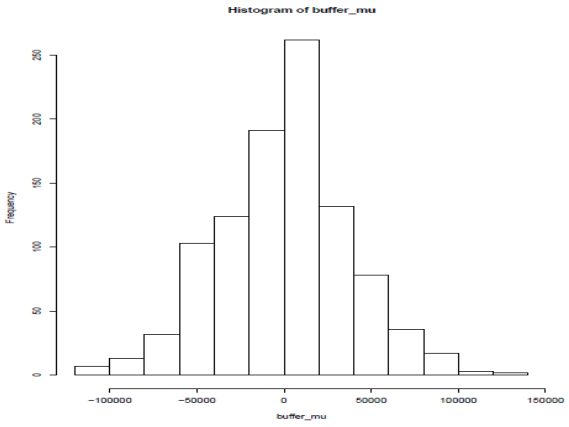
Second, we “update” the public function in the next component. For the computing next mu and s2, set the formulas by using the variables “self” in the previous component function “public”. After the computing we set the self$buffer_mu and self$buffer_s2 as new mu and s2. Also, self$iteration is renewed by +1 from the previous one. By setting this, iteration label can be the counter number of applying update function. Since the mutable variables are used to set the initializing function “public”, the results from this update function should be mutable, too.

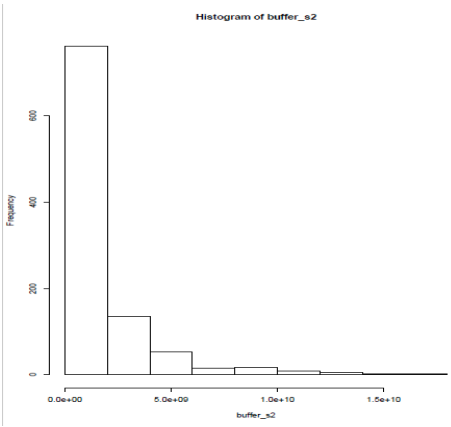
Third, the results are as follows.

```
sampler <- normal_model$new(y = c(24, 43, 58, 71, 43, 49, 61, 44, 67, 49, 53, 56, 59, 52, 62, 54, 57, 33, 46, 43, 57),
                            mu_0 = 42, tau_0 = 200, s2_0 = 400, nu_0 = 1, initial_mu = 1, initial_s2 = 1, buffer_length = 1000)
buffer_mu <- buffer_s2 <- numeric(1000)
for (sampler_iteration in 1:1000){
sampler$update()
buffer_mu[i] <- sampler$buffer_mu # or whatever NULLme you've used
buffer_s2[i] <- sampler$buffer_s2 # (idem)
}
```

By setting the “sampler” as the R6 class as above, we can get the results mean, standard variation and histograms.

```
update_sampler(sampler, iterations = 1000)
mean(sampler$buffer_mu)
[1] 493.2865
sd(sampler$buffer_mu)
[1] 34560.32
mean(sampler$buffer_s2)
[1] 1259379447
sd(sampler$buffer_s2)
[1] 2328320798
hist(sampler$buffer_mu)
hist(sampler$buffer_s2)
```





Q2.

```
gamma_mcmc_extended <- R6Class(
 classname = 'normal_model_extended',
 inherit = normal_model,
 private = list(buffer = NULL, buffer_length=100),
 public = list(
 initialize = function(y = c(24, 43, 58, 71, 43, 49, 61, 44, 67, 49, 53, 56, 59, 52, 62, 54, 57, 33, 46, 43, 57),
                       mu_0 = 42, tau_0 = 200, s2_0 = 400, nu_0 = 1, initial_mu = 1, initial_s2 = 1, buffer_length = 1000){
 private$buffer_length <- buffer_length
 private$buffer <- numeric(buffer_length)
 private$buffer_mu[i] <- sampler$buffer_mu # or whatever name you've used
 private$buffer_s2[i] <- sampler$buffer_s2 # (idem)
 super$initialize(y = c(24, 43, 58, 71, 43, 49, 61, 44, 67, 49, 53, 56, 59, 52, 62, 54, 57, 33, 46, 43, 57),
                       mu_0 = 42, tau_0 = 200, s2_0 = 400, nu_0 = 1, initial_mu = 1, initial_s2 = 1, buffer_length = 1000)
 },
 update = function(iterations = 1){
 n <- self$iteration
 for (i in seq(from = n, length = iterations)){
 super$update()
 if (i <= private$buffer_length){
 private$buffer_mu[i] <- private$buffer_mu # or whatever name you've used
 private$buffer_s2[i] <- private$buffer_s2 # (idem)
 }
 }
 if (i > private$buffer_length) warning('Buffer full! extra samples discarded')
 }
 ),
 active = list(history = function() private$buffer)
 )
```

First, we use the R6 class interitance to get the exactly same functionality of R6 class “normal_model”. The inheritance can be used to get same functionality but modify the method. We modify the previous class to get the buffer, which has the buffer size as we want. Therefore, in the “private” of list, we call the buffer and buffer_length as unmodified from outside by using private. As in the public initializing function in Q1, we give the initializing variables. Also, we add the buffer_length as the number we want. By using prefix “super”, we can succeed the functionality of class “normal_model” in simpler way.

Second, in the if statement, updating of the initializing values is computed by only using update function of “normal_model” only for the iteration less or equals to the private$buffer_length. If the iteration exceed the buffer_length, the warning is printed.

Third, the results are as follows.

```
sampler <- normal_model_extended$new(y = c(24, 43, 58, 71, 43, 49, 61, 44, 67, 49, 53, 56, 59, 52, 62, 54, 57, 33, 46, 43, 57),
                                     mu_0 = 42, tau_0 = 200, s2_0 = 400, nu_0 = 1, initial_mu = 1, initial_s2 = 1, buffer_length = 1000)
buffer_mu <- buffer_s2 <- numeric(1000)
for (sampler_iteration in 1:1000){
sampler$update()
buffer_mu[i] <- sampler$buffer_mu # or whatever NULLme you've used
buffer_s2[i] <- sampler$buffer_s2 # (idem)
}

 update_sampler(sampler, iterations = 1000)
 mean(sampler$buffer_mu)
[1] 493.2865
 sd(sampler$buffer_mu)
[1] 34560.32
 mean(sampler$buffer_s2)
[1] 1259379447
 sd(sampler$buffer_s2)
[1] 2328320798
 hist(sampler$buffer_mu)
 hist(sampler$buffer_s2)
```