STAT243: PS 4

Xinyue Zhou

October 12, 2015

1.
a) Locate the problem
First of all, I tried repeat the process outside of the function. I find each time I reload the ”tmp.Rda”, the location of the sequence is 1, which is the return value of $.Random.seed[2]$. And each time I run $runif(1)$, the location of the sequence increment by 1.
When it comes to the function, I find even it seems I reload ”tmp.Rda” each time I call the function, which is suppose to give me the same answer everytime. However, it doesn't. So I check the position of the .Random.seed sequence by printing it out. I find out even I load the ”tmp.Rda” time to time, when I run the function, the position of sequence is still there. This makes suspect that run the function actually move the position on the .Random.seed on the global environment. This is proved by print out $.Random.seed[2]$ before load the ”tmp.Rda”. I find that position is change on global environment each time I call the function.

```
set.seed(0)
runif(1)

## [1] 0.8966972

save(.Random.seed, file = 'tmp.Rda')
runif(1)

## [1] 0.2655087

load('tmp.Rda')
.Random.seed[2]

## [1] 1

runif(1)

## [1] 0.2655087

library(pryr)
tmp <- function() {
  print("Location of Random seed 1: ")
  print(where(".Random.seed"))
  cat("position on global env:", .Random.seed[2], "\n")
  load('tmp.Rda')
  print("Location of Random seed 2: ")
  print(where(".Random.seed"))
  cat("position on local env:", .Random.seed[2], "\n")
  a = runif(1)
  cat("position on local env after generating a random number:", .Random.seed[2], "\n")
  return(a)
}
tmp()
```

```
## [1] "Location of Random seed 1: "
## <environment: R_GlobalEnv>
## position on global env: 2
## [1] "Location of Random seed 2: "
## <environment: 0x7fb8992a6788>
## position on local env: 1
## position on local env after generating a random number: 1
## [1] 0.3721239
```

b) Solve the problem
After locating the problem, solving is easy. I just add one more option in load command, to let the it load from global to the local function. See the result, which is favorable.

```
library(pryr)
tmp <- function() {
  print("Location of Random seed 1: ")
  print(where(".Random.seed"))
  cat("position on global env:", .Random.seed[2], "\n")
  load('tmp.Rda', env=.GlobalEnv)
  print("Location of Random seed 2: ")
  print(where(".Random.seed"))
  cat("position on local env:", .Random.seed[2], "\n")
  a = runif(1)
  cat("position on local env after generating a random number:", .Random.seed[2], "\n")
  return(a)
}
tmp()
```

```
## [1] "Location of Random seed 1: "
## <environment: R_GlobalEnv>
## position on global env: 3
## [1] "Location of Random seed 2: "
## <environment: R_GlobalEnv>
## position on local env: 1
## position on local env after generating a random number: 2
## [1] 0.2655087
```

2.
First of all, load all the packages that are needed.

```
require(utils)
require(microbenchmark)

## Loading required package:  microbenchmark
```

a)
In the main function, I create another help function with parameter k, which is used to calculate the denominator. Then I apply the inner function to the sequence of k. Here is a tricky part: I calculate the first and last element in k separately. Since $0 \times log(0)$makes no sense, I calculate 0^0 instead.

```
p = .3
phi = .5
#n = 2
sum_f <- function(p,phi,n) {
  if(n<=0) {return("n is not valid")}
```

```r
  else if(n==1) {return(p^(n*phi)+(1-p)^(n*phi))}
  logf <- function(k){
  coef = lchoose(n,k) #directly return log value of combination
  ret = coef+(phi-1)*(n*log(n)-k*log(k)-(n-k)*log(n-k))+
    k*phi*log(p)+(n-k)*phi*log(1-p)
  return(exp(ret))
  }
  k = seq(0,n)
  ret1 = sum((sapply(k[2:(length(k)-1)], logf)))
  ret2 = p^(n*phi)+(1-p)^(n*phi)
  return(ret1+ret2)
}
microbenchmark(sum_f(p,phi,2000))
```

```
## Unit: milliseconds
##                 expr      min       lq     mean   median       uq      max
##  sum_f(p, phi, 2000) 8.293609 10.01666 11.12888 10.69722 11.46915 41.61709
##  neval
##    100
```

b)
In **b)**, I directly pass k in as a vector and return value is a vector too. In this way, I can implement it without using loop.

```r
sum_f_vec <- function(p,phi,n) {
  if(n==0) {return("n is not valid")}
  else if(n==1) {return(p^(n*phi)+(1-p)^(n*phi))}
  #pass k as vector here
  logf <- function(k){
    coef = lchoose(n,k)
    ret = coef+(phi-1)*(n*log(n)-k*log(k)-(n-k)*log(n-k))+
      k*phi*log(p)+(n-k)*phi*log(1-p)
    return(exp(ret))
  }
  k = seq(0,n)
  #print(exp(logf(k[2:(length(k)-1)])))
  ret1 = sum(logf(k[2:(length(k)-1)]))
  ret2 = p^(n*phi)+(1-p)^(n*phi)
  return(ret1+ret2)
}
microbenchmark(sum_f(p,phi,2000),sum_f_vec(p,phi,2000))
```

```
## Unit: microseconds
##                     expr      min       lq       mean    median         uq
##      sum_f(p, phi, 2000) 8340.626 9746.183 10477.1233 10416.508 11043.7080
##  sum_f_vec(p, phi, 2000)  287.279  306.499   343.3552   318.168   369.4255
##        max neval
##  17852.506   100
##    550.669   100
```

By comparing part a) and part b), the vectorized method can improve the efficiency by two magnitude order.

3.
First of all, load all the packages that are needed:

```r
load("mixedMember.Rda")
require(compiler)

## Loading required package:  compiler

require(microbenchmark)
```

a) This is just a most straightforward way doing so.

```r
#a
sum_a_ret = sapply(1:100000,sum_a<-function(i)
  sum(wgtsA[[i]]*muA[IDsA[[i]]]))
sum_b_ret = sapply(1:100000,sum_b<-function(i)
  sum(wgtsB[[i]]*muB[IDsB[[i]]]))
```

b) Let's improve Case A. *sapply/lapply/apply* are just internal for loops writing in C++. Although it can more efficient than using loops in R, but it's better to avoid loop. One way to do it is convert the data into matrix form, and use Colsum function instead of using for loop.
Here is a tricky part. When convert data into a matrix, as the list in ID or weight are not in the same length, we should find out the longest list. And if the list is short of elements, we complement it using length(ID)+1. For mu vector, I add one more element 0 in the end (like for case A, the empty value is substitude by 1001). Therefore, when I select μ using ID, as it comes to empty one, it automatically selete 0 in weight, which is favored.

```r
#b
transfer_matrix<-function(list){
  n=length(list)
  max_len = max(sapply(list,length))
  max_num = max(sapply(list,max))
  ##
  for (i in 1:n){
    if (length(list[[i]])<max_len)
      list[[i]]=c(list[[i]],rep(max_num+1,max_len-length(list[[i]])))
  }
  mat =array(unlist(list),c(max_len,length(list)))
  return(mat)
}
#transfer_matrix(list)
##change the data object to store IDdata and weight data
IDsA_mat = transfer_matrix(IDsA)
wgtsA_mat = transfer_matrix(wgtsA)
##observe some of the value is larger than one
##those are invalid data and we can just leave it there
muA_mat = c(muA,0)
IDsB_mat = transfer_matrix(IDsB)
wgtsB_mat = transfer_matrix(wgtsB)
##observe some of the value is larger than one
##those are invalid data and we can just leave it there
muB_mat = c(muA,0)
##test the time of them
microbenchmark(sapply(1:100000,
                      sum_a<-function(i) sum(wgtsA[[i]]*muA[IDsA[[i]]]))
               ,colSums(wgtsA_mat*muA_mat[IDsA_mat]))

## Unit: milliseconds
##                                                                    expr
##  sapply(1:1e+05, sum_a <- function(i) sum(wgtsA[[i]] * muA[IDsA[[i]]]))
##                                   colSums(wgtsA_mat * muA_mat[IDsA_mat])
##        min        lq      mean    median        uq      max neval
##  229.84470 264.88785 294.75812 285.56132 319.85728 469.9265   100
##   10.01798  11.27777  18.50054  15.08779  19.75478 131.9282   100
```

c) It is not enough to improve the case B, let's finish it in a more genius way. As in case B, there are only 10 ID numbers. Therefore, we can directly using matrix multiplication.

```
weight_convert<-matrix(0,nr=length(wgtsB),nc=length(muB))
for (i in 1:nrow(weight_convert)){
  weight_convert[i,IDsB[[i]]]<-wgtsB[[i]]
}
microbenchmark(
  ret<-as.vector(weight_convert%*%muB),
      sapply(1:100000,sum_b<-function(i) sum(wgtsB[[i]]*muB[IDsB[[i]]])))

## Unit: milliseconds
##                                                                     expr
##                                 ret <- as.vector(weight_convert %*% muB)
##  sapply(1:1e+05, sum_b <- function(i) sum(wgtsB[[i]] * muB[IDsB[[i]]]))
##        min         lq      mean     median         uq        max neval
##    1.58921   1.746081   1.99138   1.957882   2.164376   2.979487   100
##  214.60059 261.134958 292.37224 291.307492 316.948514 409.975356   100
```

d) It has been included in b) and c)

4.
a) and b)
First of all I set variables we need in test this problem.

```
require(pryr)
set.seed(0)
y=rnorm(1000000)
x1=rnorm(1000000)
x2=rnorm(1000000)
x3=rnorm(1000000)
print(mem_used()) #see the memory used at very beginning.

## 144 MB
```

To write the report in a compressed manager, the inspect processes are all finished in one step. First of all, I set 7 nodes to inspect which part in the function cost the most memory.
Specifically, see the one of the special node called "PART A", which show the memory usage after fitting the model and creating the object z.
Here I will list several weired discoveraries without explanation, which associated with the internal storage and processing of the data.
1. In the node 2, after the function creating two objects called ret.x, ret.y, the usage of memory is actually decreased.
2. In the step creating x and y, it does cost lots of memory, but there is a discrepancy with the object size of x and y. For example, the size of x is 88MB, while the step that creating x just costs 40MB in the RAM.
After several trials, I find that it is in following step exhaust the memory:

- The creating of object mf
- The creating of object x
- The creating of object y

Before I start to revise the lm function, I need to figure out why these three object occupies such a large space on the RAM. Actually, this objects are much more larger than what we expect. Take x for example. x occupies more than 80MB, while itself is just a 1000000×4 matrix. I expect it just used about 32MB memory on RAM. After taking a closed look at internal storage of x, I find there are many unused attributes also stored in object x, which is unecessarily occupying the space. It is the same case for y.
There is a way to improve the function in terms of the structure of x and y. I can set all the tag in x and y to be NULL and delete all the attributes that are unecessary for fitting the model. This will be shown in the next part.

```
lm_my<-function (formula, data, subset, weights, na.action, method = "qr",
                 model = TRUE, x = FALSE, y = FALSE, qr = TRUE, singular.ok = TRUE,
                 contrasts = NULL, offset, ...)
{
  #1
  very_beginning = mem_used()
  cat("1:Memory used at the beginning of the function"
      , very_beginning/1000000, "MB","\n")

  ret.x <- x
  ret.y <- y
  #2
  node2 = mem_used()
  cat("2:", (node2-very_beginning)/1000000, "MB","\n")

  cl <- match.call()
  mf <- match.call(expand.dots = FALSE)
  m <- match(c("formula", "data", "subset", "weights", "na.action",
               "offset"), names(mf), 0L)
  mf <- mf[c(1L, m)]
  mf$drop.unused.levels <- TRUE
  mf[[1L]] <- quote(stats::model.frame)
  mf <- eval(mf, parent.frame())

  cat("mf_size:", object_size(mf)/1000000, "MB", "\n")
  #get the obj_size of mf

  #3
  node3 = mem_used()
  cat("3: memory used after creation of several objs:"
      ,(node3-node2)/1000000, "MB", "\n")

  if (method == "model.frame")
    return(mf)
  else if (method != "qr")
    warning(gettextf("method = '%s' is not supported. Using 'qr'",
                     method), domain = NA)

  #4
  node4 = mem_used()
  #print("4:"); print(node4);print(node4-node3)
  print("check 4--5")
  mt <- attr(mf, "terms"); #print(object_size(mt))
  y <- model.response(mf, "numeric")
  cat("y size:",object_size(y)/1000000, "MB", "\n")
  print("get more info of y to diagnose:")
  print(head(.Internal(inspect(y))))
```

```r
w <- as.vector(model.weights(mf)); #print(object_size(w))

#5
node5 = mem_used()
cat("5: after the creating of mt, y, w objects:"
    , (node5-node4)/1000000, "MB", "\n")

if (!is.null(w) && !is.numeric(w))
  stop("'weights' must be a numeric vector")
offset <- as.vector(model.offset(mf))


if (!is.null(offset)) {
  if (length(offset) != NROW(y))
    stop(
      gettextf("number of offsets is %d, should equal %d (number of observations)"
               ,length(offset), NROW(y)), domain = NA)
}
if (is.empty.model(mt)) {
  x <- NULL
  z <- list(
    coefficients = if (is.matrix(y)) matrix(, 0, 3)
    else numeric(), residuals = y, fitted.values = 0 * y
    , weights = w, rank = 0L, df.residual = if (!is.null(w)) sum(w != 0)
    else if (is.matrix(y)) nrow(y)
    else length(y))

  if (!is.null(offset)) {
    z$fitted.values <- offset
    z$residuals <- y - offset
  }
  #print(head(.Internal(inspect(z))))
}
else {
  #5.1

  x <- model.matrix(mt, mf, contrasts)
  node5.1 = mem_used()
  cat("5.1: the memory usage for creating object x"
      ,(node5.1-node5)/1000000, "MB", "\n")
  cat("x size:",object_size(x)/1000000, "MB", "\n")
  #
  print("get more info of x to diagnose:")
  print(head(.Internal(inspect(x))))

  cat("Memory usage before fitting model"
      ,(node5.1-very_beginning)/1000000, "MB", "\n")
  z <- if (is.null(w)) {
    #mem_fit = mem_used()
    lm.fit(x, y, offset = offset, singular.ok = singular.ok,
           ...)
    #mem_fit <- mem_used()
  }
  else lm.wfit(x, y, w, offset = offset, singular.ok = singular.ok,
               ...)
  print("PART A: after fitting model and creating z")
```

```
    cat("z size:",object_size(z)/1000000,"MB","\n")
    cat("memory used for creating z:"
        ,(mem_used()-node5.1)/1000000, "MB", "\n")
  }
  #6
  node6 = mem_used()
  cat("6: after fit the model"
      ,(node6-node5)/1000000,"MB","\n")

  class(z) <- c(if (is.matrix(y)) "mlm", "lm")
  z$na.action <- attr(mf, "na.action")
  z$offset <- offset
  z$contrasts <- attr(x, "contrasts")
  z$xlevels <- .getXlevels(mt, mf)
  z$call <- cl
  z$terms <- mt
  if (model)
    z$model <- mf
  if (ret.x)
    z$x <- x
  if (ret.y)
    z$y <- y
  if (!qr)
    z$qr <- NULL
  #7
  node7 = mem_used()
  cat("7: the rest of steps", (node7-node6)/1000000, "MB", "\n")
  z
}
ret = lm_my(y~x1+x2+x3)

## 1:Memory used at the beginning of the function 144.4006 MB
## 2: 5.6e-05 MB
## mf_size: 32.00466 MB
## 3: memory used after creation of several objs: 32.00722 MB
## [1] "check 4--5"
## y size: 64.00019 MB
## [1] "get more info of y to diagnose:"
## @119e3b000 14 REALSXP g1c7 [MARK,NAM(2),ATT] (len=1000000, tl=0) 1.26295,-0.326233,1.3298,1.27243,0.414
## ATTRIB:
##   @7fb8a01aa808 02 LISTSXP g1c0 [MARK]
##     TAG: @7fb899819478 01 SYMSXP g1c0 [MARK,LCK,gp=0x6000] "names" (has value)
##     @11a5dd000 16 STRSXP g1c7 [MARK] (len=1000000, tl=0)
##       @7fb89ae7da88 09 CHARSXP g1c1 [MARK,gp=0x61] [ASCII] [cached] "1"
##       @7fb89958ebf8 09 CHARSXP g1c1 [MARK,gp=0x61] [ASCII] [cached] "2"
##       @7fb89a051ad8 09 CHARSXP g1c1 [MARK,gp=0x61] [ASCII] [cached] "3"
##       @7fb899980ed8 09 CHARSXP g1c1 [MARK,gp=0x61,ATT] [ASCII] [cached] "4"
##       @7fb899967728 09 CHARSXP g1c1 [MARK,gp=0x61,ATT] [ASCII] [cached] "5"
##       ...
##          1          2          3          4          5          6
##  1.2629543 -0.3262334  1.3297993  1.2724293  0.4146414 -1.5399500
## 5: after the creating of mt, y, w objects: 83.68998 MB
## 5.1: the memory usage for creating object x 40.05449 MB
## x size: 88.00085 MB
## [1] "get more info of x to diagnose:"
## @11f661000 14 REALSXP g1c7 [MARK,NAM(2),ATT] (len=4000000, tl=0) 1,1,1,1,1,...
## ATTRIB:
##   @7fb8a155a628 02 LISTSXP g1c0 [MARK]
##     TAG: @7fb8998197f8 01 SYMSXP g1c0 [MARK,LCK,gp=0x4000] "dim" (has value)
##     @7fb8a1158d58 13 INTSXP g1c1 [MARK,NAM(2)] (len=2, tl=0) 1000000,4
##     TAG: @7fb899819868 01 SYMSXP g1c0 [MARK,LCK,gp=0x4000] "dimnames" (has value)
##     @7fb89e4f7ed0 19 VECSXP g1c2 [MARK] (len=2, tl=0)
##       @119699000 16 STRSXP g1c7 [MARK] (len=1000000, tl=0)
##  @7fb89ae7da88 09 CHARSXP g1c1 [MARK,gp=0x61] [ASCII] [cached] "1"
##  @7fb89958ebf8 09 CHARSXP g1c1 [MARK,gp=0x61] [ASCII] [cached] "2"
##  @7fb89a051ad8 09 CHARSXP g1c1 [MARK,gp=0x61] [ASCII] [cached] "3"
##  @7fb899980ed8 09 CHARSXP g1c1 [MARK,gp=0x61,ATT] [ASCII] [cached] "4"
##  @7fb899967728 09 CHARSXP g1c1 [MARK,gp=0x61,ATT] [ASCII] [cached] "5"
##  ...
##       @7fb89e507710 16 STRSXP g1c3 [MARK] (len=4, tl=0)
##  @7fb89e4f7e98 09 CHARSXP g1c2 [MARK,gp=0x60,ATT] [ASCII] [cached] "(Intercept)"
##  @7fb899a0e6d8 09 CHARSXP g1c1 [MARK,gp=0x61] [ASCII] [cached] "x1"
##  @7fb89a982e68 09 CHARSXP g1c1 [MARK,gp=0x61] [ASCII] [cached] "x2"
##  @7fb89eb014d8 09 CHARSXP g1c1 [MARK,gp=0x61] [ASCII] [cached] "x3"
##     TAG: @7fb89982bd90 01 SYMSXP g1c0 [MARK,LCK,gp=0x4000] "assign" (has value)
##     @7fb89e4f7e60 13 INTSXP g1c2 [MARK] (len=4, tl=0) 0,1,2,3
##   (Intercept)         x1          x2         x3
## 1           1  1.0536289  1.43764703  0.9394052
## 2           1 -1.3120350  1.51762833 -1.6734425
## 3           1  2.0852002 -0.04439491  1.1289378
## 4           1  0.2929012  0.79577121  2.1111618
## 5           1  0.4976107  0.73851520 -1.4236423
## 6           1  0.1921341  0.23886450 -0.1855364
## Memory usage before fitting model 155.7521 MB
## [1] "PART A: after fitting model and creating z"
## z size: 136.0031 MB
## memory used for creating z: 80.06845 MB
## 6: after fit the model 120.1215 MB
## 7: the rest of steps 0.015216 MB
```

c)
In this part, I revise the lm function by deleting unecessary attributes in x and y. Like in object x, it is the ”dimnames”. It looks like a minor change, while make a big difference.

```
lm_revised<-function (formula, data, subset, weights, na.action, method = "qr",
                 model = TRUE, x = FALSE, y = FALSE, qr = TRUE, singular.ok = TRUE,
                 contrasts = NULL, offset, ...)
{
  #1
  very_beginning = mem_used()
  cat("1:Memory used at the beginning of the function"
      , very_beginning/1000000, "MB","\n")

  ret.x <- x
  ret.y <- y
  #2
  node2 = mem_used()
  cat("2:", (node2-very_beginning)/1000000, "MB","\n")

  cl <- match.call()
  mf <- match.call(expand.dots = FALSE)
```

```
m <- match(c("formula", "data", "subset", "weights", "na.action",
             "offset"), names(mf), 0L)
mf <- mf[c(1L, m)]
mf$drop.unused.levels <- TRUE
mf[[1L]] <- quote(stats::model.frame)
mf <- eval(mf, parent.frame())

cat("mf_size:", object_size(mf)/1000000, "MB", "\n")
#get the obj_size of mf

#3
node3 = mem_used()
cat("3: memory used after creation of several objs:"
    ,(node3-node2)/1000000, "MB", "\n")

if (method == "model.frame")
  return(mf)
else if (method != "qr")
  warning(gettextf("method = '%s' is not supported. Using 'qr'",
                   method), domain = NA)

#4
node4 = mem_used()
#print("4:"); print(node4);print(node4-node3)
print("check 4--5")
mt <- attr(mf, "terms"); #print(object_size(mt))
y <- model.response(mf, "numeric")
attributes(y)$names =NULL
cat("y size:",object_size(y)/1000000, "MB", "\n")
print("get more info of y to diagnose:")
print(head(.Internal(inspect(y))))
w <- as.vector(model.weights(mf)); #print(object_size(w))

#5
node5 = mem_used()
cat("5: after the creating of mt, y, w objects:"
    , (node5-node4)/1000000, "MB", "\n")

if (!is.null(w) && !is.numeric(w))
  stop("'weights' must be a numeric vector")
offset <- as.vector(model.offset(mf))


if (!is.null(offset)) {
  if (length(offset) != NROW(y))
    stop(gettextf(
      "number of offsets is %d, should equal %d (number of observations)",
                  length(offset), NROW(y)), domain = NA)
}
if (is.empty.model(mt)) {
  x <- NULL
  z <- list(
    coefficients = if (is.matrix(y)) matrix(, 0, 3)
    else numeric(), residuals = y, fitted.values = 0 * y
    ,weights = w, rank = 0L, df.residual = if (!is.null(w)) sum(w != 0)
    else if (is.matrix(y)) nrow(y)
    else length(y))

  if (!is.null(offset)) {
    z$fitted.values <- offset
    z$residuals <- y - offset
  }
  #print(head(.Internal(inspect(z))))
}
else {
  #5.1

  x <- model.matrix(mt, mf, contrasts)
  attributes(x)$dimnames=NULL
  node5.1 = mem_used()
  cat("5.1: the memory usage for creating object x"
      ,(node5.1-node5)/1000000, "MB", "\n")
  cat("x size:",object_size(x)/1000000, "MB", "\n")

  #
  print("get more info of x to diagnose:")
  print(head(.Internal(inspect(x))))
  cat("Memory usage before fitting model"
      ,(node5.1-very_beginning)/1000000, "MB", "\n")
  z <- if (is.null(w)) {
    #mem_fit = mem_used()
    lm.fit(x, y, offset = offset, singular.ok = singular.ok,
           ...)
    #mem_fit <- mem_used()
  }
  else lm.wfit(x, y, w, offset = offset, singular.ok = singular.ok,
               ...)
  print("PART A: after fitting model and creating z")
  cat("z size:",object_size(z)/1000000,"MB","\n")
  cat("memory used for creating z:"
      ,(mem_used()-node5.1)/1000000, "MB", "\n")
}
#6
node6 = mem_used()
cat("6: after fit the model"
    ,(node6-node5)/1000000,"MB","\n")

class(z) <- c(if (is.matrix(y)) "mlm", "lm")
z$na.action <- attr(mf, "na.action")
z$offset <- offset
z$contrasts <- attr(x, "contrasts")
z$xlevels <- .getXlevels(mt, mf)
z$call <- cl
z$terms <- mt
if (model)
  z$model <- mf
if (ret.x)
  z$x <- x
if (ret.y)
  z$y <- y
if (!qr)
  z$qr <- NULL
  #7
  node7 = mem_used()
  cat("7: the rest of steps"
      , (node7-node6)/1000000, "MB", "\n")
  z
}
ret = lm_revised(y~x1+x2+x3)

## 1:Memory used at the beginning of the function 331.5755 MB
## 2: 0 MB
## mf_size: 32.00466 MB
## 3: memory used after creation of several objs: 32.00457 MB
## [1] "check 4--5"
## y size: 8.00004 MB
## [1] "get more info of y to diagnose:"
## @124a51000 14 REALSXP g0c7 [NAM(2)] (len=1000000, tl=0) 1.26295,-0.326233,1.3298,1.27243,0.414641,...
## [1]  1.2629543 -0.3262334  1.3297993  1.2724293  0.4146414 -1.5399500
## 5: after the creating of mt, y, w objects: 8.001216 MB
## 5.1: the memory usage for creating object x 32.00054 MB
## x size: 32.00037 MB
## [1] "get more info of x to diagnose:"
## @12e6c4000 14 REALSXP g1c7 [MARK,NAM(2),ATT] (len=4000000, tl=0) 1,1,1,1,1,...
## ATTRIB:
##   @7fb8a1558b18 02 LISTSXP g1c0 [MARK]
##     TAG: @7fb8998197f8 01 SYMSXP g1c0 [MARK,LCK,gp=0x4000] "dim" (has value)
##     @7fb8a73ddbc8 13 INTSXP g1c1 [MARK,NAM(2)] (len=2, tl=0) 1000000,4
##     TAG: @7fb89982bd90 01 SYMSXP g1c0 [MARK,LCK,gp=0x4000] "assign" (has value)
##     @7fb8a73dbb88 13 INTSXP g1c2 [MARK,NAM(2)] (len=4, tl=0) 0,1,2,3
##      [,1]       [,2]        [,3]       [,4]
## [1,]    1  1.0536289  1.43764703  0.9394052
## [2,]    1 -1.3120350  1.51762833 -1.6734425
## [3,]    1  2.0852002 -0.04439491  1.1289378
## [4,]    1  0.2929012  0.79577121  2.1111618
## [5,]    1  0.4976107  0.73851520 -1.4236423
## [6,]    1  0.1921341  0.23886450 -0.1855364
## Memory usage before fitting model 72.00662 MB
## [1] "PART A: after fitting model and creating z"
## z size: 64.00262 MB
## memory used for creating z: 64.00636 MB
## 6: after fit the model 96.0055 MB
## 7: the rest of steps 0.00072 MB
```

Now, let's compare the usage of memory before and after revising the function.

stage	lm_ my	lm_ revised
size of y	64 MB	8 MB
size of x	88 MB	32 MB
size of z	136 MB	64 MB
memory usage for creating z	80 MB	64 MB
memory usage before fitting model	156 MB	72 MB

From the table above, we can find the memory usage is highly saved by revise the lm function.