CS5331: ASSIGNMENT 1

rapply : rapply, stands for Recursive Apply, and it is one of the more powerful function of the apply group functions. This function is used to implement or apply a function to all elements of a list in a recursive manner. A general view of the function is as follows:

$rapply(x, function() < function\_definition >, class = c(< class\_type >), how = < how\_parameters >)$

The first argument is the list : x . The second argument is the function that is to be applied to the list x. This function has two basic modes, which are *how* and *classes*. Some descriptions of the parameter how is as follows:

- how = replace : The elements in the list which itself is not a list, and has a class included in classes is replaced by the result, generated by applying the function to the elements.
- how = list : All non-list elements whose class is included in classes are replaced by the result, generated by applying the function to the elements. All other elements are replaced by the default result.

After implementing a comparative code between rapply() and its equivalent loop, benchmark was measured, and the performance was plotted. The code is as follows :

```
library("microbenchmark")
#Creating a demo list
demo_list <- list(1:5,list(6:9),10:12,list(13:20))

#Applying rapply to find squares of each element
rapply_function <- function()
{
list_rapply <- rapply(demo_list, function(demo_list){demo_list^2})
}

#Using loops to do the same evaluations

list_loop<- list()
loop_function <- function()
{
for(i in 1:length(demo_list))
{
for(j in 1:length(demo_list[[i]]))
{
if(class(demo_list[[i]][j]) == "list")
```

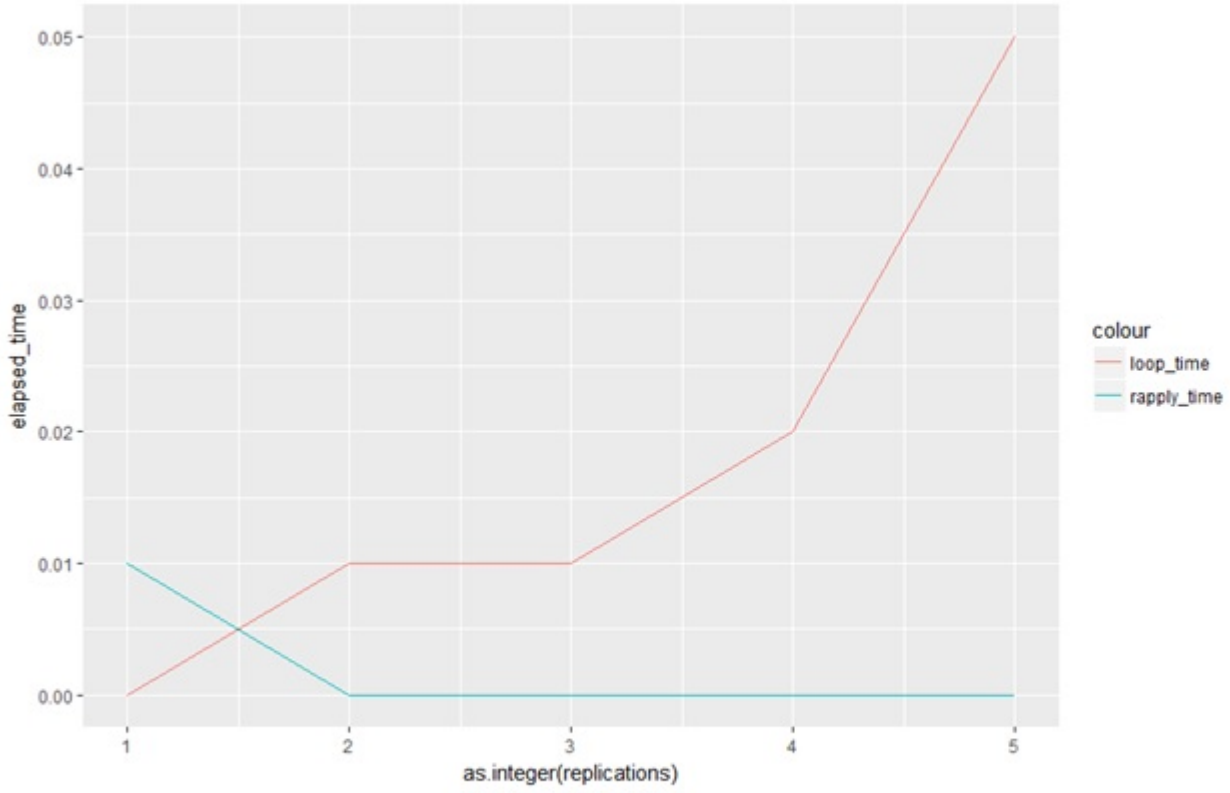
```
{
subList <- demo_list[[i]][j]
for(k in 1:length(subList[[1]]))
list_loop <- c(list_loop,subList[[1]][k]^2)
}
else
{
list_loop <- c(list_loop,demo_list[[i]][j]^2)
}
}
}
}

#Benchmarking both the functions
microbenchmark(rapply_function(),loop_function())

# Performance graph for rapply
performance_rapply <- data.frame(Replications =
character(), Elapsed_Time = numeric())
rep <- c(100,500,1000,1500,2000)
for(i in rep){
tmp <- benchmark(rapply(demo_list,
function(demo_list){demo_list^2}))
performance_rapply <- rbind(performance_rapply,
data.frame(as.character(i), tmp))
}
names(performance_rapply) <- c("replications",
"elapsed_time")

# Performance graph for loop
performance_for <- data.frame(Replications =
character(), Elapsed_Time = numeric())
rep <- c(100,500,1000,1500,2000)
for(i in rep){
tmp <- benchmark(loop_function(), replications = i)
performance_for <- rbind(performance_for,
data.frame(as.character(i), tmp))
}
names(performance_for) <- c("replications","loop_time")

performance_final <- cbind(performance_rapply,
performance_for$loop_time)
names(performance_final) <- "loop_time"


#plot for the performance comparison
```

```
ggplot(performance_final, aes(x = as.integer(replications))) +
geom_line(aes(y = elapsed_time, colour = "rapply_time")) +
geom_line(aes(y = loop_time, colour = "loop_time"))
```

Benchmark was taken for 10 instances, they are as follows:

```
Console ~/
> microbenchmark(rapply_function(),loop_function())
Unit: microseconds
              expr     min      lq      mean median      uq     max neval
 rapply_function()  9.481  9.877 10.47313 10.271 10.666 36.346   100
   loop_function() 11.456 11.852 12.56700 12.247 12.247 31.605   100
> microbenchmark(rapply_function(),loop_function())
Unit: microseconds
              expr     min      lq      mean median      uq     max neval
 rapply_function()  9.086  9.482 10.59956  9.877 10.272 36.741   100
   loop_function() 11.061 11.457 11.97836 11.852 11.852 24.494   100
> microbenchmark(rapply_function(),loop_function())
Unit: microseconds
              expr     min      lq      mean median      uq     max neval
 rapply_function()  9.086  9.482 10.16102  9.876 10.272 33.975   100
   loop_function() 11.061 11.457 12.27465 11.852 12.247 47.013   100
> microbenchmark(rapply_function(),loop_function())
Unit: microseconds
              expr     min      lq      mean median      uq     max neval
 rapply_function()  8.691  9.087 10.50873  9.481  9.482 32.790   100
   loop_function() 10.271 11.061 12.53929 11.062 11.457 72.691   100
> microbenchmark(rapply_function(),loop_function())
Unit: microseconds
              expr     min      lq      mean median      uq     max neval
 rapply_function()  9.086  9.482 10.34675  9.877 10.2720 33.186   100
   loop_function() 11.061 11.457 12.42867 11.852 12.2465 30.025   100
> microbenchmark(rapply_function(),loop_function())
Unit: microseconds
              expr     min      lq      mean median      uq     max neval
 rapply_function()  9.086  9.482 10.22030  9.877 10.2715 37.926   100
   loop_function() 11.061 11.457 11.84799 11.852 11.8520 20.544   100
> microbenchmark(rapply_function(),loop_function())
Unit: microseconds
              expr     min      lq      mean median      uq     max neval
 rapply_function()  9.086  9.876 12.51164  9.877 10.666 41.877   100
   loop_function() 11.061 11.457 15.21782 11.852 12.247 47.408   100
> microbenchmark(rapply_function(),loop_function())
Unit: microseconds
              expr     min      lq      mean median      uq     max neval
 rapply_function()  9.481  9.876 15.07555 10.272 23.5060 44.642   100
   loop_function() 11.061 11.457 15.98824 11.852 12.4445 48.988   100
> microbenchmark(rapply_function(),loop_function())
Unit: microseconds
              expr     min      lq      mean median      uq     max neval
 rapply_function()  9.086  9.482 11.77288  9.877 10.272 50.568   100
   loop_function() 11.061 11.457 13.14380 11.852 12.049 47.408   100
> microbenchmark(rapply_function(),loop_function())
Unit: microseconds
              expr     min      lq      mean median      uq     max neval
 rapply_function()  9.086  9.482 10.77734  9.877 10.2715 65.975   100
   loop_function() 11.061 11.457 12.32600 11.852 11.8520 33.185   100
>
```

A graph was plotted for 5 instances, and the result found are as follows:



From the results, we can see, that rapply, performs the same task in a better time. This can be seen by observing the mean values of both the functions for the benchmarks, and from the graph plotted.

tapply : A Ragged array in R, is an array containing values given by a unique combination of the levels of certain factors. tapply applies a function to each cell of a ragged array. A general view of the function is as follows:

$$tapply(x, INDEX, function() < function\_definition >, default = NA, simplify = TRUE)$$

The description of the arguments are as follows:

- x : An R object for which a split method exists.
- INDEX : a list of one or more factors having the same length as x.
- function : the function that is to be applied to the list.
- default : the value with which the array is initialized as array. Only in case of simplification to an array.
- simplify : if FALSE, tapply will always return an array mode list. If TRUE, then the function always returns an array with the mode of scalar.

After implementing a comparative code between tapply() and its equivalent loop, benchmark was measured, and the performance was plotted. The code is as follows :

```
library(microbenchmark)
library(ggplot2)

# Creating a demo data_frame, assigning student_id and
 randoming marks, and adding labels, based on year
df_demo <- data.frame(student_id = 1:10000,
 marks =rnorm(10000, mean = 60, sd = 10),
year = gl(4, 2500,
labels = c("2017","2016","2015","2014")))

#Assigning grades according to marks obtained
for(i in seq_along(df_demo$student_id))
{
if(df_demo$marks[i] > 90)
{
df_demo$grade[i] <- "O"
}
else if(df_demo$marks[i] > 80)
{
df_demo$grade[i] <- "E"
}
else if(df_demo$marks[i] > 75)
```

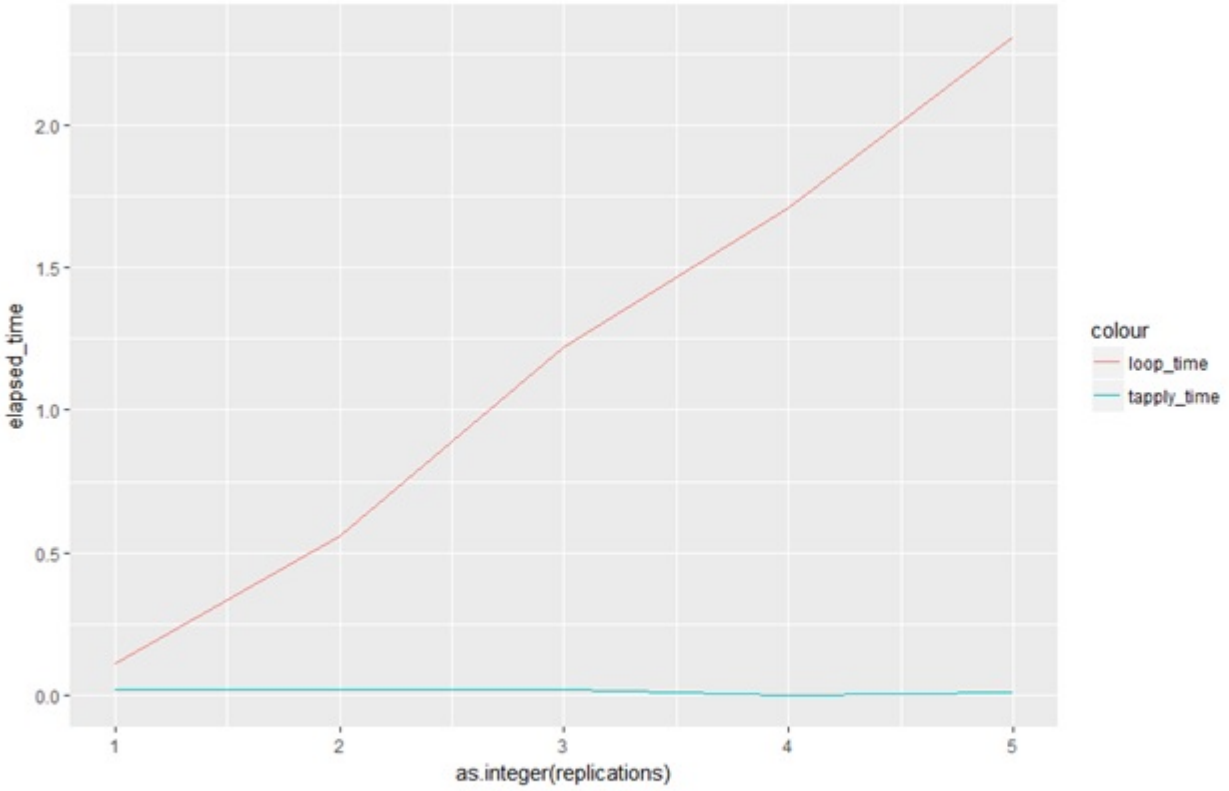
```
{
df_demo$grade[i] <- "A"
}
else if(df_demo$marks[i] > 70)
{
df_demo$grade[i] <- "B"
}
else if(df_demo$marks[i] > 65)
{
df_demo$grade[i] <- "C"
}
else if(df_demo$marks[i] > 60)
{
df_demo$grade[i] <- "D"
}
else
{
df_demo$grade[i] <- "F"
}
}

#Using tapply to find mean of marks based on year
tapply_function <- function()
{
tapply(df_demo$marks, df_demo$year, mean)
}

#Using loop to find the same
loop_function <- function()
{
for(i in unique(df_demo$year))
{
c(mean(df_demo[which(df_demo$year == i),"marks"]),i)
}
}
loop_function()
tapply_function()

#benchmarking
microbenchmark(tapply_function(),loop_function())

# Performance graph for tapply
performance_tapply <- data.frame(Replications = character(),
 Elapsed_Time = numeric())
rep <- c(100,500,1000,1500,2000)
for(i in rep)
```

```
{
tmp <- benchmark(tapply(df_demo$marks, df_demo$year, mean))
performance_tapply <- rbind(performance_tapply,
 data.frame(as.character(i), tmp))
}
names(performance_tapply) <- c("replications","elapsed_time")

# Performance graph for loop
performance_for <- data.frame(Replications = character(),
 Elapsed_Time = numeric())
rep <- c(100,500,1000,1500,2000)
for(i in rep)
{
tmp <- benchmark(loop_function(), replications = i)
performance_for <- rbind(performance_for, data.frame(as.character(i), tmp))
}
names(performance_for) <- c("replications","elapsed_time_for")

performance_final <- cbind(performance_tapply,
 performance_for$elapsed_time_for)
names(performance_final) <- "elapsed_time_for"


#plot for the performance comparison
ggplot(performance_final, aes(x = as.integer(replications))) +
geom_line(aes(y = elapsed_time, colour = "tapply_time")) +
geom_line(aes(y = elapsed_time_for, colour = "loop_time"))
```

The benchmark for 10 instances are as follows :

```
Console ~/
> microbenchmark(tapply_function(),loop_function())
Unit: microseconds
              expr     min      lq      mean  median       uq      max neval
 tapply_function() 128.000 139.654  155.7847 146.963  162.1725  392.296   100
   loop_function() 865.975 913.382 1114.4209 957.234 1007.6045 4119.701   100
> microbenchmark(tapply_function(),loop_function())
Unit: microseconds
              expr     min       lq      mean  median       uq      max neval
 tapply_function() 129.580 137.8760  176.5412 148.543  172.247  568.493   100
   loop_function() 869.135 932.1475 1192.7931 988.444 1142.122 3541.726   100
> microbenchmark(tapply_function(),loop_function())
Unit: microseconds
              expr     min       lq      mean   median       uq       max neval
 tapply_function() 126.420 135.5060  180.2705 146.7650  161.382  2300.048   100
   loop_function() 873.481 919.1105 2021.2212 959.6045 1040.986 94509.358   100
> microbenchmark(tapply_function(),loop_function())
Unit: microseconds
              expr     min       lq      mean   median       uq      max neval
 tapply_function() 125.629 135.7040  169.6987 146.1730  173.827  428.642   100
   loop_function() 843.061 889.2835 1131.8827 943.6045 1069.827 3178.269   100
> microbenchmark(tapply_function(),loop_function())
Unit: microseconds
              expr     min       lq      mean   median       uq      max neval
 tapply_function() 128.791 135.9010  208.6083 147.5555  175.4075 2604.640   100
   loop_function() 872.691 921.4805 1065.9943 958.4190 1016.0980 3181.824   100
> microbenchmark(tapply_function(),loop_function())
Unit: microseconds
              expr    min       lq      mean  median       uq      max neval
 tapply_function() 124.84 137.6795  161.8923 146.963  167.7035  287.604   100
   loop_function() 839.90 909.0360 1122.7527 945.185 1032.6905 3812.343   100
> microbenchmark(tapply_function(),loop_function())
Unit: microseconds
              expr     min       lq      mean   median       uq      max neval
 tapply_function() 128.790 147.3580  235.4053 166.3210  203.852 2872.096   100
   loop_function() 877.826 938.8635 1118.2926 991.9995 1067.653 4865.972   100
> microbenchmark(tapply_function(),loop_function())
Unit: microseconds
              expr     min       lq      mean   median       uq      max neval
 tapply_function() 132.346 149.7285  182.4038  173.827  198.3205  368.592   100
   loop_function() 892.048 980.1475 1233.4923 1062.517 1161.0860 4299.848   100
> microbenchmark(tapply_function(),loop_function())
Unit: microseconds
              expr    min       lq     mean   median       uq      max neval
 tapply_function() 128.79 135.7035  197.673 145.7775  164.938 3287.306   100
   loop_function() 871.11 911.9995 1123.014 947.9500 1061.135 3899.256   100
> microbenchmark(tapply_function(),loop_function())
Unit: microseconds
              expr     min       lq     mean   median       uq      max neval
 tapply_function() 129.185 137.4815  203.109 144.7895 154.667 2531.158   100
   loop_function() 868.345 915.1595 1080.529 941.6295 992.592 3955.750   100
>
```

A graph was plotted for 5 instances, and the result found are as follows:



From the results, we can see, that tapply, performs the same task in a better time. This can be seen by observing the mean values of both the functions for the benchmarks, and from the graph plotted.

mapply : This apply function is a multivariate version of sapply. mapply applies a function to the first elements of each argument, the second elements, the third elements, and so on. The arguments are recycled if necessary. A general view of the function is as follows:

$mapply(function() < functiondefinition >, ..., MoreArgs = NULL, SIMPLIFY = TRUE, USE > NAMES = TRUE)$

The description of the arguments are as follows:

- function : function that is to be applied
- ... : arguments to vectorize over.
- MoreArgs : a list of other arguments to the function.
- SIMPLIFY : logical or character string.
- USE.NAMES : logical values as input. If the first argument has names, or if it is a character vector, the character vector is used as the names.

After implementing a comparative code between mapply() and its equivalent loop, benchmark was measured, and the performance was plotted. The code is as follows :

```
library(microbenchmark)
library(ggplot2)

#using mapply to find sum of respective elements of sub_1, sub_2 and sub_3
demo_list <- list(sub_1 = c(1:10), sub_2 = c(11:20), sub_3 = c(21:30))
mapply_function <- function()
{
results_mapply <- mapply(sum, demo_list$sub_1,
demo_list$sub_2, demo_list$sub_3)
}


#Equivalent for loop
loop_function <- function()
{
temp <- 1
results_loop <- list()
for(i in 1:length(demo_list[[temp]]))
{
results_loop<- c(results_loop,(demo_list$sub_1[i] +
 demo_list$sub_2[i] + demo_list$sub_3[i]))
}
}
```

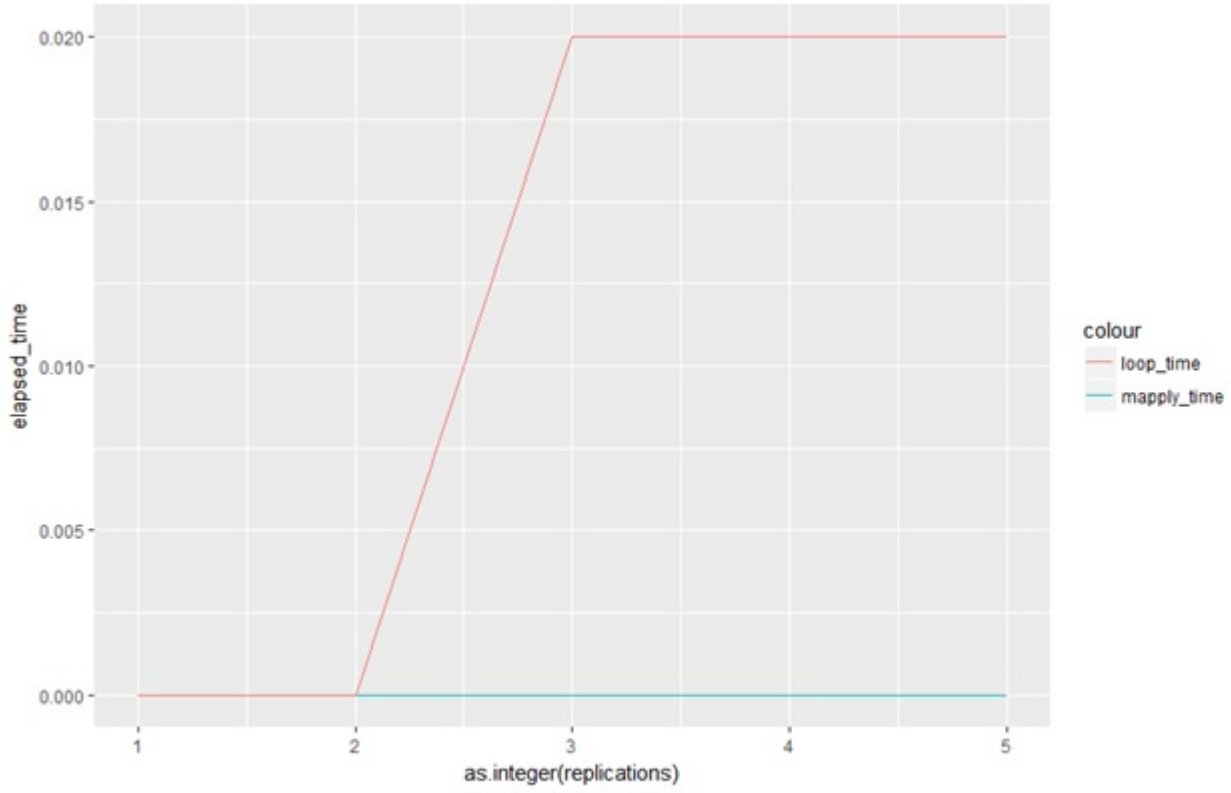
```
loop_function()

microbenchmark(mapply_function(),loop_function())


# Performance graph for mapply
performance_mapply <- data.frame(Replications = character(),
Elapsed_Time = numeric())
rep <- c(100,500,1000,1500,2000)
for(i in rep)
{
tmp <- benchmark(mapply(sum, demo_list$sub_1, demo_list$sub_2,
demo_list$sub_3))
performance_mapply <- rbind(performance_mapply,
 data.frame(as.character(i), tmp))
}
names(performance_mapply) <- c("replications","elapsed_time")

# Performance graph for loop
performance_for <- data.frame(Replications = character(),
 Elapsed_Time = numeric())
rep <- c(100,500,1000,1500,2000)
for(i in rep)
{
tmp <- benchmark(loop_function(), replications = i)
performance_for <- rbind(performance_for, data.frame(as.character(i), tmp))
}
names(performance_for) <- c("replications","elapsed_time_for")

performance_final <- cbind(performance_mapply,
 performance_for$elapsed_time_for)
names(performance_final) <- "elapsed_time_for"


#plot for the performance comparison
ggplot(performance_final, aes(x = as.integer(replications))) +
geom_line(aes(y = elapsed_time, colour = "mapply_time")) +
geom_line(aes(y = elapsed_time_for, colour = "loop_time"))
```

The benchmark for 10 instances are as follows :

Console ~/

```
> microbenchmark(mapply_function(),loop_function())
Unit: microseconds
              expr    min     lq     mean median     uq    max neval
 mapply_function() 20.938 21.728 24.56497 22.124 22.716 58.864   100
   loop_function()  5.530  6.321  6.60945  6.321  6.716 23.704   100
> microbenchmark(mapply_function(),loop_function())
Unit: microseconds
              expr    min     lq     mean median      uq    max neval
 mapply_function() 20.939 21.729 31.29681 22.519 30.8145 79.407   100
   loop_function()  5.530  5.926  9.53680  6.321  6.7160 58.469   100
> microbenchmark(mapply_function(),loop_function())
Unit: microseconds
              expr    min      lq     mean median     uq    max neval
 mapply_function() 20.938 21.7285 26.88792 22.124 23.308 76.642   100
   loop_function()  5.531  5.9260  7.94872  6.321  6.716 25.284   100
> microbenchmark(mapply_function(),loop_function())
Unit: microseconds
              expr    min     lq     mean median     uq    max neval
 mapply_function() 21.333 22.124 30.94916 22.519 37.136 98.765   100
   loop_function()  5.926  6.321  8.44251  6.716  7.111 28.049   100
> microbenchmark(mapply_function(),loop_function())
Unit: microseconds
              expr    min     lq     mean median     uq    max neval
 mapply_function() 19.753 20.543 22.62527 20.939 21.729 60.840   100
   loop_function()  5.530  5.926  6.21040  5.926  6.321 20.149   100
> microbenchmark(mapply_function(),loop_function())
Unit: microseconds
              expr    min     lq     mean median      uq     max neval
 mapply_function() 20.938 21.728 24.52154 22.124 22.9135 110.617   100
   loop_function()  5.926  5.926  7.20597  6.321  6.7160  27.655   100
> microbenchmark(mapply_function(),loop_function())
Unit: microseconds
              expr    min     lq     mean median     uq    max neval
 mapply_function() 21.728 22.518 30.72793 22.914 25.679 84.544   100
   loop_function()  5.925  6.321  8.45829  6.716  7.111 25.284   100
> microbenchmark(mapply_function(),loop_function())
Unit: microseconds
              expr    min     lq     mean median     uq     max neval
 mapply_function() 20.938 21.728 24.79014 22.123 22.519 164.741   100
   loop_function()  5.530  5.926  7.22572  6.321  6.716  51.358   100
> microbenchmark(mapply_function(),loop_function())
Unit: microseconds
              expr    min     lq     mean median     uq    max neval
 mapply_function() 20.938 21.729 23.56944 22.124 22.913 60.840   100
   loop_function()  5.925  5.926  6.83062  6.321  6.716 22.519   100
> microbenchmark(mapply_function(),loop_function())
Unit: microseconds
              expr    min      lq     mean median     uq    max neval
 mapply_function() 20.938 21.7280 23.29291 22.123 22.519 60.839   100
   loop_function()  5.531  6.1235  6.40400  6.321  6.716 12.247   100
> 
```

A graph was plotted for 5 instances, and the result found are as follows:



From the results, we can see, that mapply, performs the same task in a better time. This can be seen by observing the mean values of both the functions for the benchmarks, and from the graph plotted.

REFERENCE

`https://github.com/sm2k2010/R_Fall17_Assignments`