Parallel computing with **R** (session 3)

Solution of the exercises

2021-09-28

Exercise 1.1

```
library("parallel")
detectCores(logical = FALSE) # number of cores
```

```
## [1] 40
```

```
detectCores() # number of logical cores or threads
```

```
## [1] 40
```

Exercise 2.1

Solution 1 :

```
B <- 10
res_mean <- numeric(B)
set.seed(123)
for (b in 1:B) {
  samp <- sample(1:nrow(iris), replace = T)
  res_mean[b] <- mean(iris[samp, "Sepal.Length"])
}
res_mean
```

```
##  [1] 5.774000 5.897333 5.924667 5.871333 5.736667 5.793333 5.786667 5.818000
##  [9] 5.824000 5.935333
```

Solution 2 :

```
B <- 10
res_mean <- numeric(B)
for (b in 1:B) {
  set.seed(b)
  samp <- sample(1:nrow(iris), replace = T)
  res_mean[b] <- mean(iris[samp, "Sepal.Length"])
}
res_mean
```

```
##  [1] 5.829333 5.932000 5.839333 5.887333 5.864667 5.916000 5.828000 5.809333
##  [9] 5.812000 5.866000
```

Exercise 2.2

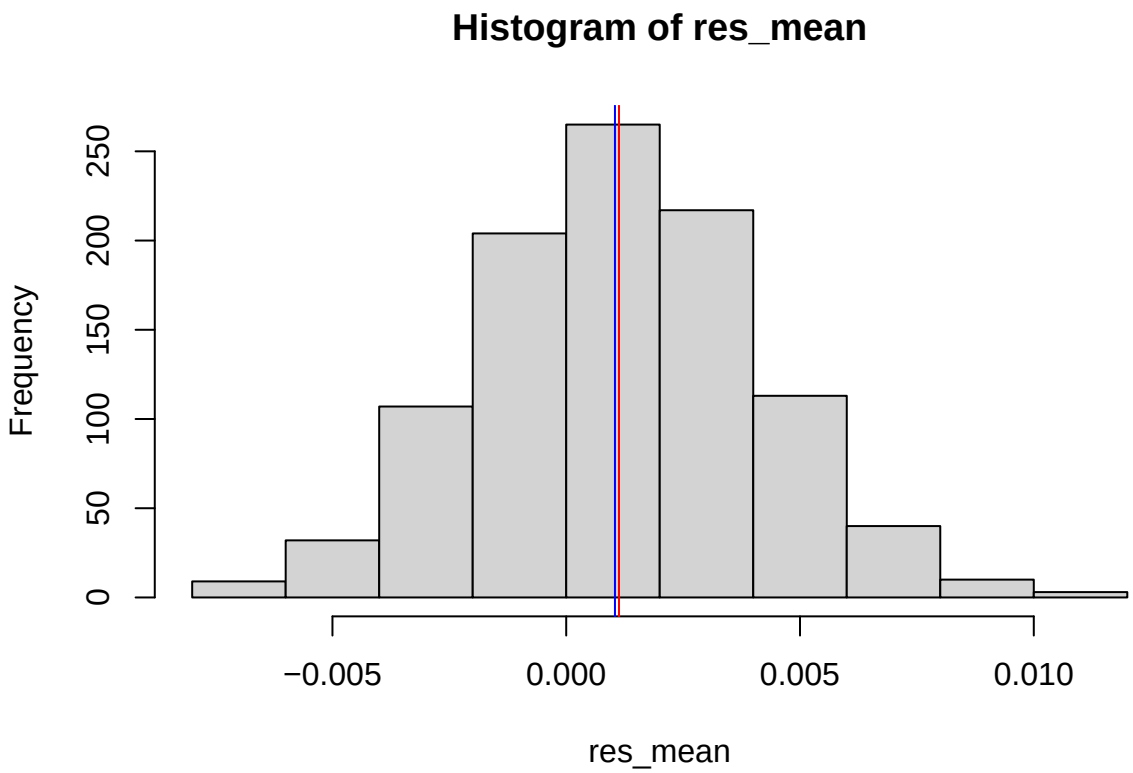
```r
set.seed(200907)
my_vec <- rnorm(100000)
mean(my_vec)
```

```
## [1] 0.001042473
```

We compute the Bootsrapped value:

```r
B <- 1000
res_mean <- numeric(B)
for (b in 1:B) {
  set.seed(b)
  samp <- sample(my_vec, replace = T)
  res_mean[b] <- mean(samp)
}
```

```r
hist(res_mean)
abline(v = mean(res_mean), col = "red")
abline(v = mean(my_vec), col = "blue")
```



Exercise 3.1

We define the data:

```r
set.seed(5656)
id_pred <- c(sample(which(iris$Species == "setosa"), 10, replace = F),
             sample(which(iris$Species == "versicolor"), 10, replace = F),
             sample(which(iris$Species == "virginica"), 10, replace = F))
test_sets <- iris[id_pred, ]
train_sets <- iris[-id_pred, ]
```

We rewrite the function which permits to do bagging :

```
my_bagging <- function(b) {
  set.seed(b)
  train_sets_bootstrap <- train_sets[sample(1:nrow(train_sets), replace = T), ]
  res_rf <- rpart(Species ~ ., data = train_sets_bootstrap)
  res <- predict(res_rf, newdata = test_sets, type = "class")
  return(res)
}
```

We now compute the function which permits to do random forest :

```
my_rf <- function(b) {
  set.seed(b)
  formula <- list(
    Species ~ Sepal.Length + Sepal.Width,
    Species ~ Sepal.Length + Petal.Length,
    Species ~ Sepal.Length + Petal.Width,
    Species ~ Sepal.Width + Petal.Length,
    Species ~ Sepal.Width + Petal.Width,
    Species ~ Petal.Length + Petal.Width
  )
  train_sets_bootstrap <- train_sets[sample(1:nrow(train_sets), replace = T), ]
  res_rf <- rpart::rpart(sample(formula, 1)[[1]], data = train_sets_bootstrap)
  res <- predict(res_rf, newdata = test_sets, type = "class")
  return(res)
}
```

We do parallel computing for each of the method :

```
require(parallel)
require(microbenchmark)
```

```
## Le chargement a nécessité le package : microbenchmark
```

```
P <- 4
cl <- makeCluster(P)
clusterExport(cl, c("test_sets", "train_sets"))
clusterEvalQ(cl, {
  library("rpart")
  })
```

```
## [[1]]
## [1] "rpart"     "stats"     "graphics"  "grDevices" "utils"     "datasets"
## [7] "methods"   "base"
##
## [[2]]
## [1] "rpart"     "stats"     "graphics"  "grDevices" "utils"     "datasets"
## [7] "methods"   "base"
##
## [[3]]
## [1] "rpart"     "stats"     "graphics"  "grDevices" "utils"     "datasets"
## [7] "methods"   "base"
##
## [[4]]
## [1] "rpart"     "stats"     "graphics"  "grDevices" "utils"     "datasets"
## [7] "methods"   "base"
```

```
mbm_1 <- microbenchmark(
  `bagging` = {bag_res <- clusterApply(cl, 1:100, fun = my_bagging)},
   `my_rf` = {rf_res <- clusterApply(cl, 1:100, fun = my_rf)},
  times = 10L
)


stopCluster(cl)
```

We aggregate the results :

```
# Loop 1: aggregate the results on the B simulation
my_matrix_bag <- matrix("0", 100, 30)
my_matrix_rf <- matrix("0", 100, 30)
for (b in 1:100) {
     my_matrix_bag[b, ] <- as.character(bag_res[[b]])
     my_matrix_rf[b, ] <- as.character(rf_res[[b]])
}

# Loop 2: predict by the most observed levels
resultats <- data.frame(Species = iris$Species[id_pred])
resultats$bagging <- apply(my_matrix_bag, 2, function(x) names(which.max(table(x))))
resultats$rf <- apply(my_matrix_rf, 2, function(x) names(which.max(table(x))))
```

We compare the predictions between the two methods :

```
table(resultats$bagging, resultats$rf)
```

```
##
##              setosa versicolor virginica
##   setosa         10          0         0
##   versicolor      0         10         0
##   virginica       0          0        10
```

The two methods give similar predictions.

Exercise 4.1

We vectorize the function: we do a **for** loop and we call the non vectorized function:

```
my_rf_vec <- function(vec_b) {
  length_b <- length(vec_b)
  my_res_vec <- matrix("0", length_b, 30)
  for(k in 1:length_b) {
    b <- vec_b[k]
    my_res_vec[k, ] <- as.character(my_rf(b))
  }
  return(my_res_vec)
}
```

For example, if we apply on a vector of size 2:

```
my_rf_vec(c(1, 2))
```

```
##      [,1]     [,2]     [,3]     [,4]     [,5]     [,6]     [,7]     [,8]
## [1,] "setosa" "setosa" "setosa" "setosa" "setosa" "setosa" "setosa" "setosa"
## [2,] "setosa" "setosa" "setosa" "setosa" "setosa" "setosa" "setosa" "setosa"
```

```
##      [,9]     [,10]    [,11]        [,12]        [,13]        [,14]
## [1,] "setosa" "setosa" "versicolor" "versicolor" "versicolor" "versicolor"
## [2,] "setosa" "setosa" "versicolor" "versicolor" "versicolor" "versicolor"
##      [,15]        [,16]        [,17]        [,18]        [,19]
## [1,] "versicolor" "versicolor" "versicolor" "versicolor" "virginica"
## [2,] "versicolor" "versicolor" "versicolor" "versicolor" "virginica"
##      [,20]        [,21]       [,22]       [,23]        [,24]       [,25]
## [1,] "versicolor" "virginica" "virginica" "versicolor" "virginica" "virginica"
## [2,] "versicolor" "virginica" "virginica" "virginica"  "virginica" "virginica"
##      [,26]       [,27]       [,28]       [,29]       [,30]
## [1,] "virginica" "virginica" "virginica" "virginica" "virginica"
## [2,] "virginica" "virginica" "virginica" "virginica" "virginica"
```

We do parallel computing and for doing this we need to create a list of size the number of jobs. Each element of the list contains a vector of size 25:

```
my_list_vec <- list(
  `job_1` = 1:25,
  `job_2` = 26:50,
  `job_3` = 51:75,
  `job_4` = 76:100
)
```

Please note that we have to export the function *my_rf()*

```
require(parallel)
P <- 4
cl <- makeCluster(P)
clusterExport(cl, c("test_sets", "train_sets", "my_rf"))
clusterEvalQ(cl, {
  library("rpart")
  })
```

```
## [[1]]
## [1] "rpart"     "stats"     "graphics"  "grDevices" "utils"     "datasets"
## [7] "methods"   "base"
##
## [[2]]
## [1] "rpart"     "stats"     "graphics"  "grDevices" "utils"     "datasets"
## [7] "methods"   "base"
##
## [[3]]
## [1] "rpart"     "stats"     "graphics"  "grDevices" "utils"     "datasets"
## [7] "methods"   "base"
##
## [[4]]
## [1] "rpart"     "stats"     "graphics"  "grDevices" "utils"     "datasets"
## [7] "methods"   "base"
```

```
mbm_2 <- microbenchmark(
   `my_rf_vec` = {rf_res_vec <- clusterApply(cl, my_list_vec, fun = my_rf_vec)},
  times = 10L
)

stopCluster(cl)
```

Note that the results are given in a list of size 4 (= number of jobs). To aggregate the data, we do:

```
rf_res_vec_mat <- rf_res_vec[[1]]
for (k in 2:4) {
  rf_res_vec_mat <- rbind(rf_res_vec_mat, rf_res_vec[[k]])
}
resultats$rf_2 <- apply(rf_res_vec_mat, 2, function(x) names(which.max(table(x))))
```

We compare the computational time with the function *randomForest()* from package **randomForest**

```
mbm_3 <- microbenchmark(
  `rf` = {rf <- randomForest::randomForest(
    Species ~ .,
    data = train_sets,
    ntree = 100,
    mtry = 2
  )}, times = 10L
)
```

```
library(tidyverse)
mbm <- rbind(mbm_1, mbm_2, mbm_3)
ggplot(mbm, aes(x = time, y = expr)) +
  geom_boxplot()
```