Advanced Programming with **R** (session 2)

Solution of the exercises

2020-09-05

Exercise 2.1

- The function *my_mean()* by using the **R** recommendation style guide

```
my_mean <- function(x) {

  # verification
  if (!is.numeric(x))
    stop("x must be a numeric vector")

  # initialization
  n <- length(x)
  res <- 0

  for (k in 1:n) {
    res <- res + x[k]
  }

  # return res
  return(res / n)
}
```

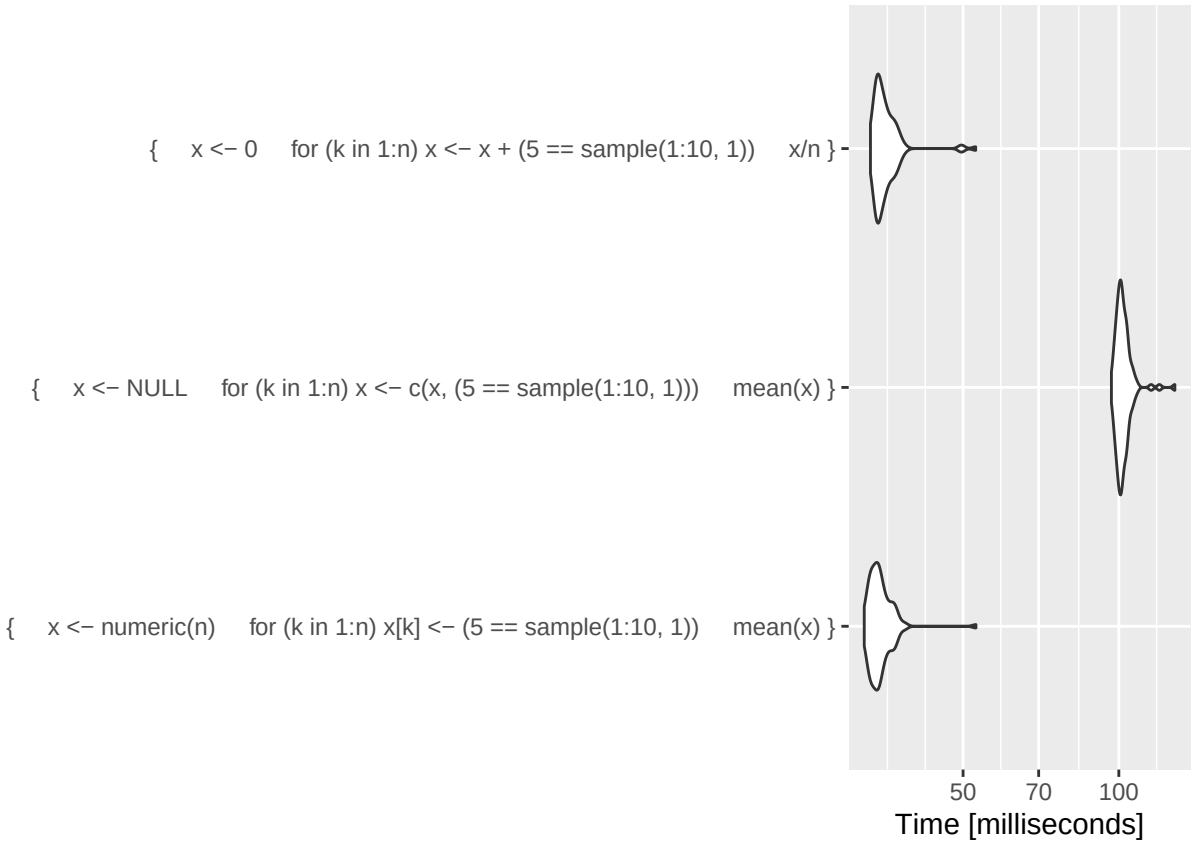
Exercise 2.2

To get the computational time between the three expressions, we use *microbenchmark()* function, and to plot the result, we use *autoplot()*

```
n <- 10 ^ 4

mbm <- microbenchmark::microbenchmark(
  {x <- numeric(n) # expression 1
  for (k in 1:n)
    x[k] <- (5 == sample(1:10, 1))
  mean(x)},
  {x <- NULL       # expression 2
  for (k in 1:n)
    x <- c(x, (5 == sample(1:10, 1)))
  mean(x)},
  {x <- 0          # expression 3
  for (k in 1:n)
    x <- x + (5 == sample(1:10, 1))
  x / n}
```

```
)

ggplot2::autoplot(mbm)
```

```
## Coordinate system already present. Adding new coordinate system, which will replace the existing one
```



In the expression 3, we do not store the result in a vector because we compute directly the result step by step. We can see that the computational time is nearly identical to the 1st case. In other term, use a vector is costly in memory but it does not take time when you fill it.

Exercise 2.3

We use the formula $\sqrt{\frac{1}{n-1}(\sum x_i^2 - (\sum x_i)^2/n}$

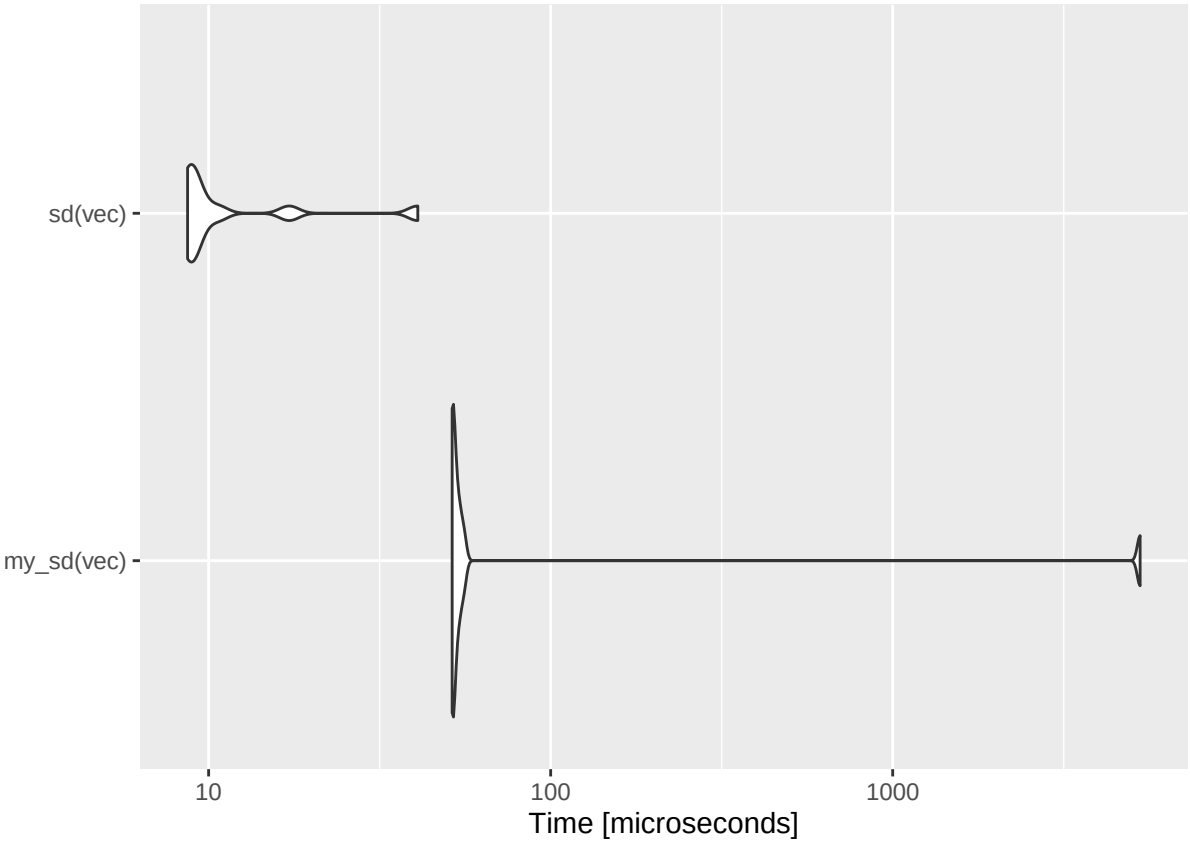
```
my_sd <- function(x) {
  # verification
  stopifnot(is.numeric(x))

  # initialisation
  n <- length(x)
  sum_x <- 0
  sum_square_x <- 0

  # we need to compute the mean and the
  for (k in 1:n) {
    sum_x <- sum_x + x[k]
    sum_square_x <- sum_square_x + x[k] ^ 2
```

```
  }

  return(sqrt(1 / (n - 1) * (sum_square_x - sum_x ^ 2 / n )))
}
```

We compare the computational time on a small example:

```
vec <- rnorm(1000)
ggplot2::autoplot(
  microbenchmark::microbenchmark(
    my_sd(vec),
    sd(vec), times = 10L))
```

```
## Coordinate system already present. Adding new coordinate system, which will replace the existing one
```



Exercise 2.4

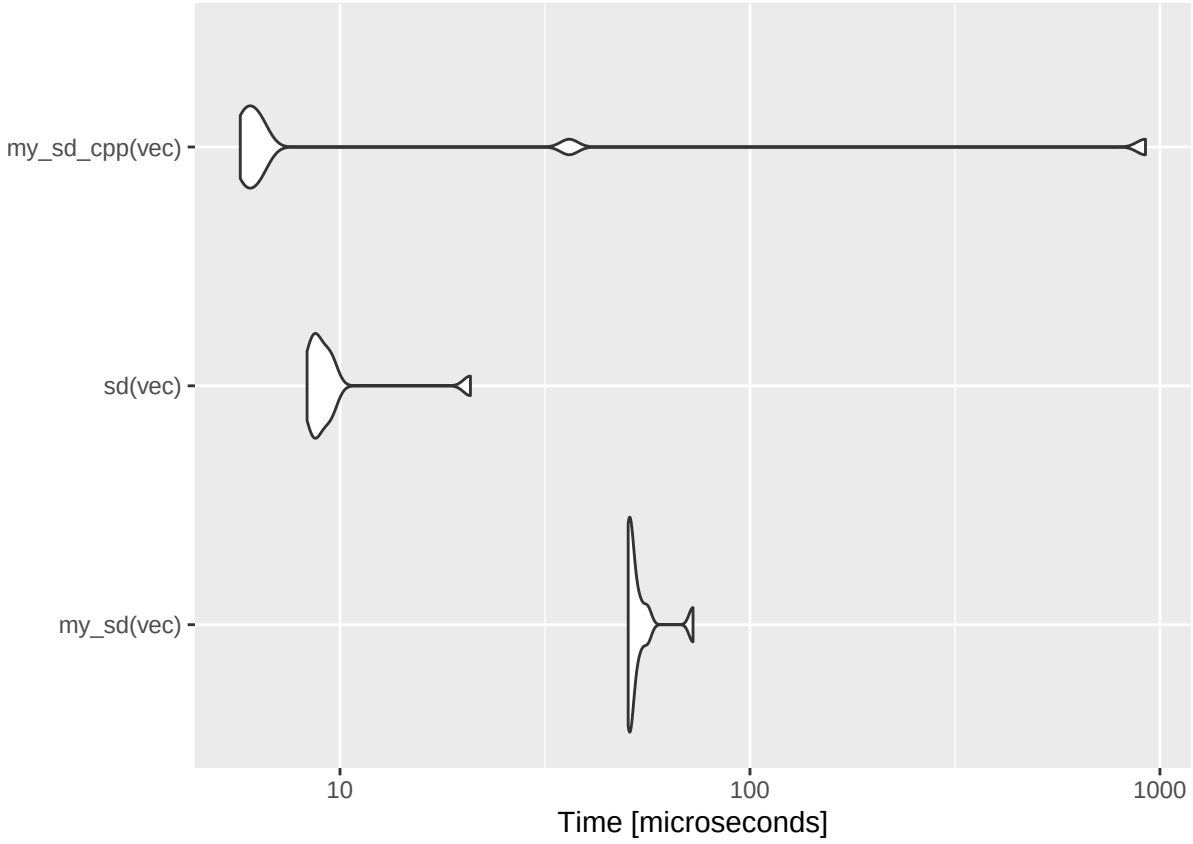
The function in **C++** looks like:

```
#include <Rcpp.h>
using namespace Rcpp;

// [[Rcpp::export]]
double my_sd_cpp(NumericVector x) {
  double res;
  double   sum_x = 0;
  double sum_square_x = 0;
```

```
  int n = x.size();

  for(int i = 0; i < n; i++) {
    sum_x = sum_x + x(i);
    sum_square_x = sum_square_x + pow(x(i), 2.0);
  }
  res = sqrt((sum_square_x - pow(sum_x,  2.0) / n ) / (n - 1));
  return res;
}
```

```
vec <- rnorm(1000)
ggplot2::autoplot(
  microbenchmark::microbenchmark(
    my_sd(vec),
    sd(vec),
    my_sd_cpp(vec),
    times = 10L))
```

```
## Coordinate system already present. Adding new coordinate system, which will replace the existing one
```



Exercise 2.5

- Simulate a list **xs** of 5 samples each of size 10 distributed under a $U_{[0,1]}$ (use if possible function *replicate()*).

```
xs <- replicate(5, runif(10), simplify = FALSE)
```

- Simulate a vector **ws** of size 5 distributed under a binomial $\mathcal{B}(10, 0.5)$ (use function *rbinom()*).

```
ws <- rbinom(5, 10, 0.5)
```

- compute the sum of each element of **xs** and multiply it by the element of **ws** (use **for** loop and *mapply()*).

```
# solution 1
res <- numeric(5)
for (k in 1:5)
  res[k] <- sum(xs[[k]]) * ws[k]
res
```

```
## [1] 12.775780 21.065676 40.998324 21.221033  9.688389
```

```
# solution 2
mapply(function(x, y) sum(x) * y, xs, ws)
```

```
## [1] 12.775780 21.065676 40.998324 21.221033  9.688389
```

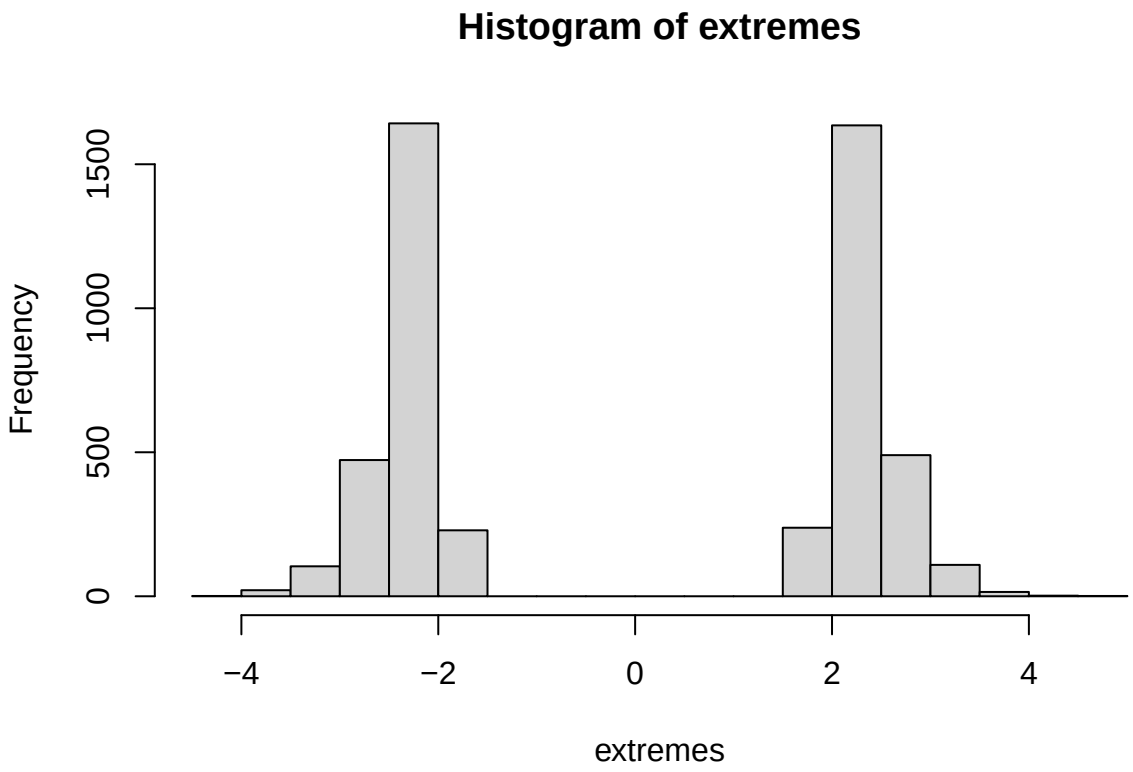
Exercise 2.6

Solution 1: we use **for** loop:

```
hist_extrm.1 <- function(n, B, ...) {
  res <- 0 # au début le phenomene s est produi 0 fois
  extremes <- NULL # ici, on ne sait pas à l'avance la taille finale du vecteur
  for (k in 1:B) {
    x <- rnorm(n)
    if (any(abs(x)>1.96)) {
      res <- res + 1
      extremes <- c(extremes, x[abs(x)>1.96])
    }
  }
  print(paste("Le phenomene s'est produit", res/B*100, "% de fois"))
  hist(extremes,...)
}  # fin de la fonction
```

```
hist_extrm.1(100, 1000)
```

```
## [1] "Le phenomene s'est produit 99.3 % de fois"
```



Solution 2: we do not use **for** loop

```
hist_extrm.2 <- function(n, B, ...) {
  # on créé une fonction qu'on va appliquer à replicate()
  my.func <- function() {
    x <- rnorm(n)
    x[abs(x)>1.96]
  }
  res.ech <- replicate(B, my.func(), simplify = FALSE)
  extremes <- unlist(res.ech) # on récupère les valeurs concernées
  res <- sapply(res.ech,  function(x) length(x) > 0) # on compte le nombre de fois où l'événement s'est
  print(paste("Le phenomene s'est produit", mean(res)*100, "% de fois"))
  hist(extremes,...)
}  # fin de la fonction
```