Exercice n°1

Variance calculation

Grégoire de Lambertye

2022-10-10

Variance calculation

The aim of this first exercice is to approache the difficulties of computer simulation and to get used to R and R Markdown. In order to illustrate these problems we will use the variance calculation through 4 different algorithms and the "var" function provided by R.

As starting point, we will use theese lines to include libraries and create our datasets

```
library(microbenchmark)#Allows the use of the microbenchmark library

set.seed(11220221)#Create random data
x1 <- rnorm(100)
x2 <- rnorm(100, mean=1000000)
x3 <- rnorm(100, mean=0.0000001)
```

Algorithme n°1: (two-pass algorithme)

The first algorithm follows the traditional variance formula: $s_n^2 = \frac{1}{(n-1)} \sum_{i=1}^{n} (x_i - x_n)^2$. It needs to read all the data twice, once to calculate the mean and once to calculate the variance.

```
precise <- function(x) {
  sum <- 0
  n <- length(x)

  #First pass: mean calculation
  for (i in x) {
    sum <- sum + i
  }
  mean <- sum/n

  variance <- 0
  #Second pass: variance calculation
  for(i in x) {
    variance <- variance + (i - mean)^2
  }
  variance <- variance/(n-1)
  return(variance)
}
```

Algorithme n°2: (one-pass algorithme)

The second algorithm use the Variance Decomposition princips : $s_n^2 = \frac{1}{(n-1)}(\sum_{i=1}^{n} x_i^2 + (\sum_{i=1}^{n} x_i)^2)$. This allows the algorithm to read the data only once.

```
excel <- function(x) {
  P1 <- 0
  P2 <- 0
  n <- length(x)
  variance <- 0

  for (i in x) {
    P1 <- P1 + i^2
    P2 <- P2 + i
  }
  P2 <- (P2^2)/n
  variance <- (P1-P2)/(n-1)
  return(variance)
}
```

Algorithme n°3: (shifted one-pass algorithme)

The third algorithm works with the Scale Invariance property : $s_x^2 = s_{x-c}^2$ with c a constant. That gives us the following formula :

$s_n^2 = \frac{1}{(n-1)}(\sum_{i=1}^{n} (x_i - c)^2 + (\frac{1}{n}\sum_{i=1}^{n}(x_i - c))^2)$

The default c-value is the first value in the dataset

```
shifted <- function(x, c=x[1]) {
  P1 <- 0
  P2 <- 0
  n <- length(x)
  variance <- 0

  for (i in x) {
    P1 <- P1 + (i-c)^2
    P2 <- P2 + i-c
  }
  P2 <- (P2^2)/n
  variance <- (P1-P2)/(n-1)
  return(variance)
}
```

Consider what would be a good value for c ?

Considering the compututation pinciples of a computer, it would be interesting to work with small number (i.e: approaching 0) so giving c the mean value should be interessting.

Algorithme n°4: (online algorithme)

The last algorithm is based on the online calculation of the variance :

$$\bar{s}_n = \frac{(n-1)\bar{x_{n-1}} + x_n}{n} = \bar{x}_{n-1} + \frac{x_n - \bar{x}_{n-1}}{n}$$

$$s_n^2 = \frac{n-2}{n-1} s_{n-1}^2 + \frac{(x_{n-1} - \bar{x}_{n-1})^2}{n}, n > 1$$

```
online <- function(x) {
  #initalisation
  n <- 2
  mean <- (x[1]+x[2])/2
  variance <- (x[1]-mean)^2 + (x[2]-mean)^2

  #Mean and variance are computed after each element in x
  for (i in 3:length(x)) {
    n <- n+1
    variance <- ((n-2)/(n-1)) * variance + ((x[i]-mean)^2/n)
    mean <- mean + (x[i]-mean)/n
  }
  return(variance)
}
```

Comparison

To facilitate the comparison between the different algorithms we will use a wrapper function that call every algorithm

```
variances <- function(x){
  return(c(precise(x), excel(x), shifted(x), online(x),var(x)))
}
```

First we will examine the result obtained by each algorithm on the same two datasets we have set up earlier.

Table 1: Variance calulation

	precise	excel	shifted	online	var
x1	1.1244184	1.1244184	1.1244184	1.1244184	1.1244184
x2	0.9419366	0.9416035	0.9419366	0.9419366	0.9419366

It appears that the excel algorithm isn't robust. For samples with big means it doesn't return a precise value for the variance and differs from the others.

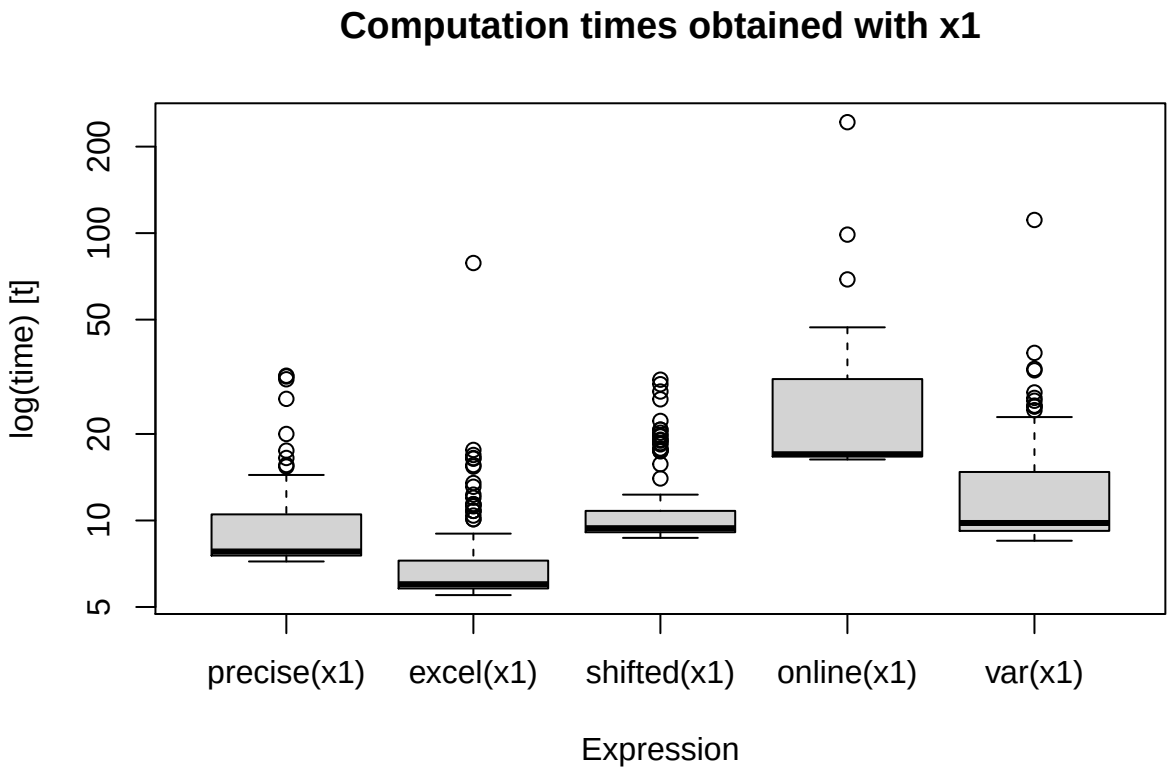
Computation time

Let's focus on the computation time, we will run each algoritm 100 times thank to the microbenchmark function using the x1 dataset.

```
micro <- microbenchmark(precise(x1), excel(x1), shifted(x1), online(x1),var(x1),  times=100)
knitr::kable(summary(micro))
```

expr	min	lq	mean	median	uq	max	neval	cld
precise(x1)	7.2	7.55	10.020	7.8	10.50	31.9	100	ab
excel(x1)	5.5	5.80	8.129	6.0	7.25	78.7	100	a
shifted(x1)	8.7	9.10	11.864	9.4	10.80	30.9	100	ab
online(x1)	16.3	16.70	24.757	17.0	31.05	243.2	100	c
var(x1)	8.5	9.20	14.020	9.8	14.75	111.1	100	b

```
boxplot(micro, main="Computation times obtained with x1")
```

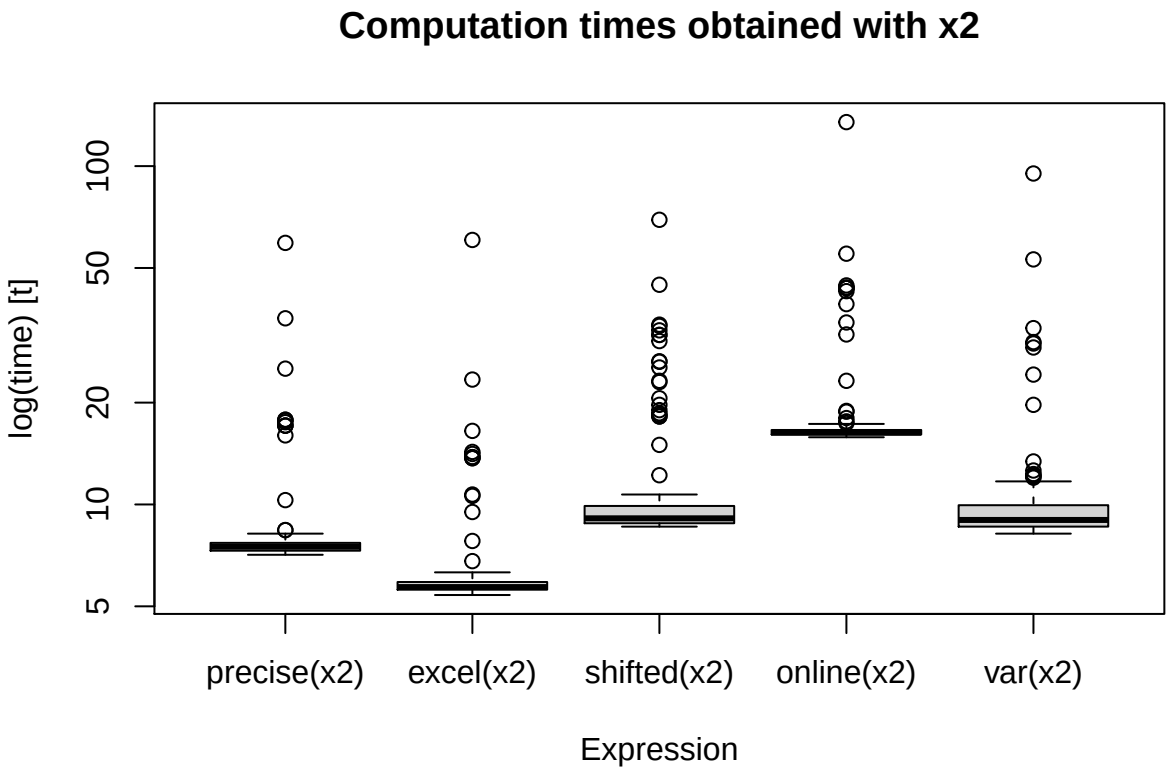


Thank to the boxplot it clearly appears that the excel algorithm is the speediest one and the online one is the worth. By the way, switching x1 dataset to x2 dataset doesn't impact the computational time so much and doesn't change the ranking either.

```
microx2 <- microbenchmark(precise(x2), excel(x2), shifted(x2), online(x2),var(x2),  times=100)
knitr::kable(summary(microx2))
```

expr	min	lq	mean	median	uq	max	neval	cld
precise(x2)	7.1	7.3	9.250	7.5	7.70	59.3	100	ab
excel(x2)	5.4	5.6	7.181	5.7	5.90	60.5	100	a
shifted(x2)	8.6	8.8	12.953	9.1	9.90	69.4	100	b
online(x2)	15.8	16.1	20.180	16.3	16.60	134.9	100	c
var(x2)	8.2	8.6	11.761	9.0	9.95	95.1	100	b

```
boxplot(microx2, main="Computation times obtained with x2")
```



Would you know another way in R to compare computing times?

Recording computing time in R can also be done with the system time :

```
start_time <- Sys.time()
invisible(excel(x1))
end_time <- Sys.time()
computation_time = end_time-start_time
print(computation_time)
```

```
## Time difference of 0.002001047 secs
```

Scale invariance property

Thanks to the scale invariance property, we can assume that $s_x^2 = s_{x+c}^2$ with c a constant. We can investigate this property with the shifted algorithm by changing the c-value. Therefor we will use the *condition number*: $S = \sum_{i=1}^{n}(x_i - x_n)^2 = \frac{1}{(n-1)} * s_n^2$. It gives a idea of how a small change in the inputs will causes a change in the output. The closet is k to 1 the best it is because it would mean our variance remain trustful with some noise errors in the input.

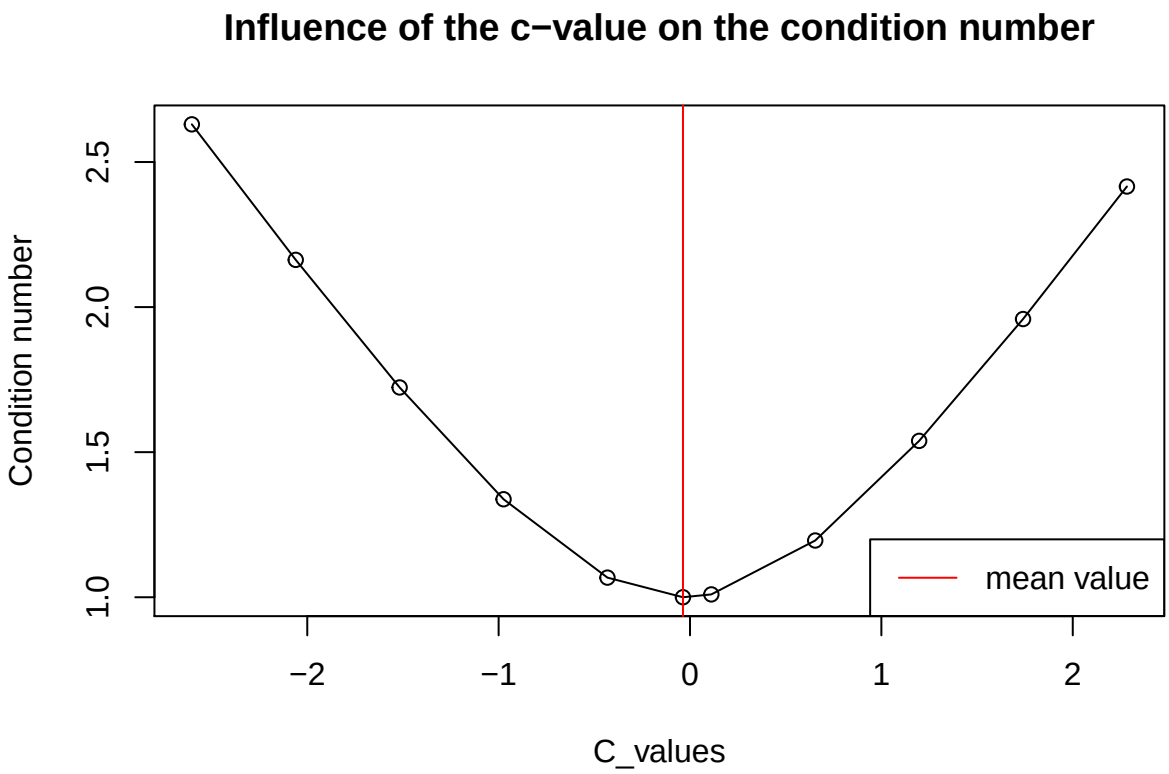
```r
condition_number <- function(mean, n , S){
  return(sqrt(1+(mean^2*n)/S))
}
```

To observe the c_value influence, we will compute the condition number with 10 c-values tooked between the min and the max of the data set and we will also compute the condition number with the mean as c-value.

```r
c_val_influence <- function(x){
  minimum <- min(x)
  maximum <- max(x)
  c_list <- seq(from=minimum, to=maximum, length.out=10)
  c_list <- sort(append(c_list, mean(x1)))
  condition_numb <- matrix(nrow = 2, ncol = 11)
  for(i in 0:length(c_list)){
    mean <- mean(x) - c_list[i]
    n <- 100
    S <- shifted(x,c_list[i])*(n-1)
    condition_numb[1,i] <- c_list[i]
    condition_numb[2,i] <- condition_number(mean, n ,S)
  }
  return(condition_numb)
}

res <- c_val_influence(x1)

plot(res[2,], x=res[1,], main="Influence of the c-value on the condition number", type='o',xlab="C_valu
abline(v=mean(x1), col='red')
legend("bottomright", "mean value", col="red", lty=1)
```



As expected, the best condition number is obtained for the maen as c-value.

/——

ALSO ADD COMPUTATION TIME ?

——/ # Condition number

We will focus on the importance of the dataset's mean for the condition number

```
res <- c()
res <- append(res,condition_number(mean(x1), 100,var(x1)*(100-1)))
res <- append(res,condition_number(mean(x2), 100,var(x2)*(100-1)))
res <- append(res,condition_number(mean(x3), 100,var(x3)*(100-1)))
res <- as.data.frame(res, row.names=c('x1','x2','x3'))
knitr::kable(res , col.names="k", caption="Condition numbers for different dataset mean")
```

Table 4: Condition numbers for different dataset mean

	k
x1	1.000609e+00
x2	1.035551e+06
x3	1.013128e+00

We can assume that the condition number is pretty sensitive. Between the condition number of x1 (mean(x1)=0) and x3 (mean(x3)=0.000 001) we have a difference of 1e-2.