Exercice n°1

Variance calculation

Grégoire de Lambertye

2022-10-17

I Task 1: Implementation

I.1 Algorithms

The aim of this first exercise is to approach 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 these lines to include libraries and create our datasets.

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

set.seed(11220221)#Set the seed
x1 <- rnorm(100)
x2 <- rnorm(100, mean=1000000)
x3 <- rnorm(100, mean=0.000001)
```

I.1.1 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)
}
```

I.1.2 Algorithme n°2: (one-pass algorithme)

The second algorithm use the Variance Decomposition principle : $s_n^2 = \frac{1}{(n-1)}(\sum_{i=1}^{n} x_i^2 - \frac{1}{n}(\sum_{i=1}^{n} x_i)^2)$. 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)
}
```

I.1.3 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 condition number formula, related to the algorithm robustness, it would be interesting to choose a c-value close to the mean.

I.1.4 Algorithme n°4: (online algorithme)

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

$$\bar{x}_n = \bar{x}_{n-1} + \frac{x_n - \bar{x}_{n-1}}{n}, \quad n > 1$$

$$S_n^2 = \frac{n-2}{n-1}S_{n-1}^2 + \frac{(x_n - \bar{x}_{n-1})^2}{n}, \quad n > 1$$

```
online <- function(x) {
  #initialization
  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)
}
```

I.2 Wrapper

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)))
}
```

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.

II Task 2: Comparison

II.1 Computation time

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

```
microx1 <- microbenchmark(precise(x1), excel(x1), shifted(x1), online(x1),var(x1),  times=100)
knitr::kable(summary(microx1), caption ="x1 computation times")
```

Table 2: x1 computation times

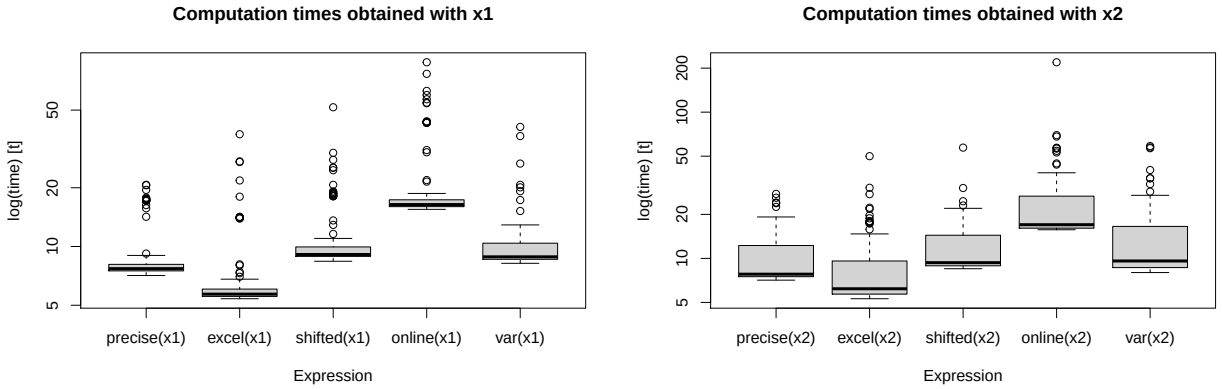
expr	min	lq	mean	median	uq	max	neval	cld
precise(x1)	7.1	7.50	9.362	7.70	8.10	20.7	100	ab
excel(x1)	5.4	5.55	7.340	5.70	6.05	37.6	100	a
shifted(x1)	8.4	8.90	11.644	9.10	9.95	51.7	100	b
online(x1)	15.5	16.00	22.797	16.40	17.35	87.9	100	c
var(x1)	8.2	8.60	10.502	8.85	10.40	41.0	100	b

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

Table 3: x2 computation times

expr	min	lq	mean	median	uq	max	neval	cld
precise(x2)	7.1	7.50	10.230	7.80	12.25	27.6	100	ab
excel(x2)	5.3	5.70	9.172	6.20	9.60	50.0	100	a
shifted(x2)	8.5	8.90	12.279	9.35	14.40	57.2	100	ab
online(x2)	15.7	16.10	25.248	17.00	26.65	219.3	100	c
var(x2)	8.0	8.65	14.729	9.60	16.55	58.6	100	b

```
boxplot(microx1, main="Computation times obtained with x1")
boxplot(microx2, main="Computation times obtained with x2")
```



Thanks 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.

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 0.001998901 secs
```

III 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*:

$$S = \sum_{i=1}^{n}(x_i - x_n)^2 = (n-1) * s_n^2$$

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.

Influence of the c-value on the condition number

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

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

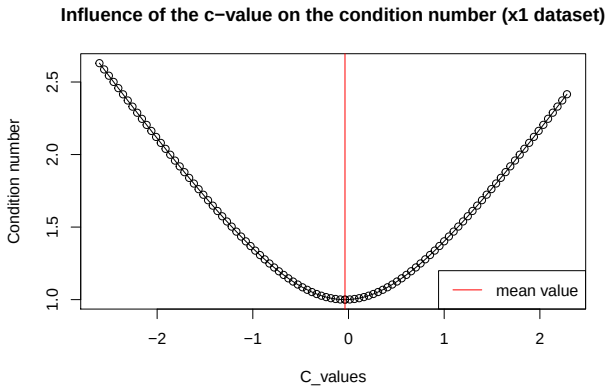
To observe the c_value influence, we will compute the condition number with 100 c-values between the min and the max of the data set and we will also compute the condition number with the mean as c-value.

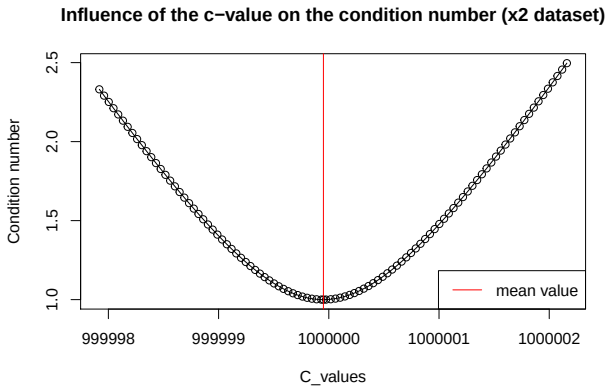
```
c_val_influence <- function(x){
  minimum <- min(x)
  maximum <- max(x)
  c_list <- seq(from=minimum, to=maximum, length.out=100)
  c_list <- sort(append(c_list, mean(x)))
  condition_numb <- matrix(nrow = 2, ncol = 101)
  for(i in 0:length(c_list)){
    S <- var(x)*(length(x)-1)
    condition_numb[1,i] <- c_list[i]
    condition_numb[2,i] <- condition_number_shifted(mean(x), length(x), S, c_list[i])
  }
  return(condition_numb)
}

cond_numb_x1 <- c_val_influence(x1)
cond_numb_x2 <- c_val_influence(x2)

plot(cond_numb_x1[2,], x=cond_numb_x1[1,], main="Influence of the c-value on the condition number (x1 da
abline(v=mean(x1), col='red')
legend("bottomright", "mean value", col="red", lty=1)

plot(cond_numb_x2[2,], x=cond_numb_x2[1,], main="Influence of the c-value on the condition number (x2 da
abline(v=mean(x2), col='red')
legend("bottomright", "mean value", col="red", lty=1)
```





As expected, the smallest condition number is obtained for the mean as c-value. This can be interpreted as: the algorithm output is more stable with this c-value. So, the mean is the best c-value for the algorithm robustness.

III.1 Influence of the c-value on the computation time

We will focus on the computation time of the shifted algorithm with different c-values.

```
micro_shifted <- microbenchmark(shifted(x2,mean(x2)-100000000000),shifted(x2,mean(x2)),shifted(x2,mean(
knitr::kable(summary(micro_shifted), caption ="Computation times with different c-values using x2")
```

Table 4: Computation times with different c-values using x2

expr	min	lq	mean	median	uq	max	neval	cld
shifted(x2, mean(x2) - 1e+11)	11.3	11.8	16.348	12.0	13.45	159.8	100	a
shifted(x2, mean(x2))	11.3	11.7	16.268	11.9	20.50	52.0	100	a
shifted(x2, mean(x2) + 1e+11)	11.2	11.8	17.514	12.0	21.70	80.6	100	a

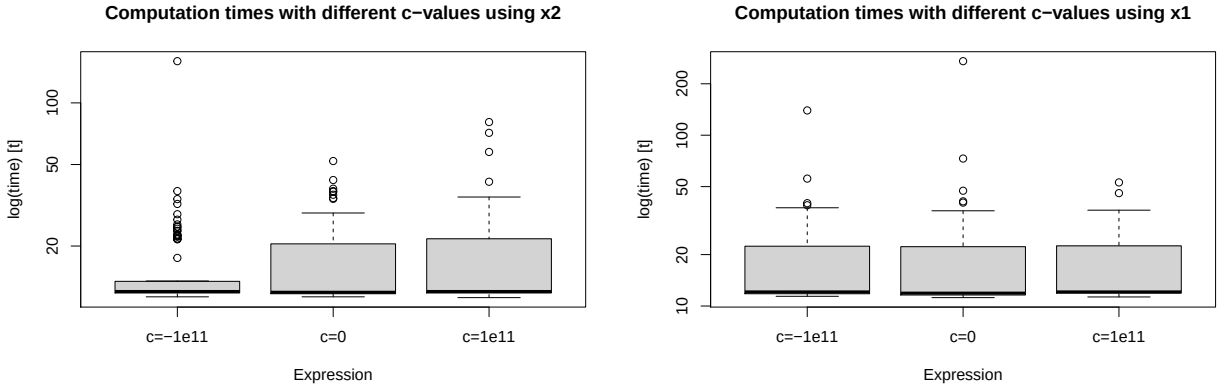
```
boxplot(micro_shifted, main="Computation times with different c-values using x2",names = c("c=-1e11","c

micro_shifted2<- microbenchmark(shifted(x1,mean(x1)-100000000000),shifted(x1,mean(x1)),shifted(x1,mean(
knitr::kable(summary(micro_shifted2), caption ="Computation times with different c-values using x1")
```

Table 5: Computation times with different c-values using x1

expr	min	lq	mean	median	uq	max	neval	cld
shifted(x1, mean(x1) - 1e+11)	11.4	11.8	17.850	12.1	22.40	139.6	100	a
shifted(x1, mean(x1))	11.2	11.6	18.769	11.9	22.25	271.2	100	a
shifted(x1, mean(x1) + 1e+11)	11.3	11.9	16.485	12.1	22.50	52.9	100	a

```
boxplot(micro_shifted2, main="Computation times with different c-values using x1", names = c("c=-1e11",
```



The c-value has a really weak influence on the computation time, Even by using big c-values, it doesn't clearly affect the computation times.

IV Task 4: Impact of the ample mean on the 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 mean=0','x2 mean=1 000 000','x3 mean=0.000 000 1'))
knitr::kable(res , col.names="k", caption="Condition numbers for different dataset mean")
```

Table 6: Condition numbers for different dataset mean

	k
x1 mean=0	1.000609e+00
x2 mean=1 000 000	1.035551e+06
x3 mean=0.000 000 1	1.013128e+00

We can assume that the condition number is very sensitive. Between the condition number of x1 (mean(x1)=0) and x3 (mean(x3)=0.000001), there is a difference of 1e-2. We proved that the best best c-value for the condition number is the average of the dataset, i.e. having a data mean of 0 after the substation of the c-value. Any difference of 1/1000000 in this average has a direct impact on the number of conditions and the stability of the algorithm.