StatSim & CM - Exercise 1 Stadler

Florian Stadler

2024-10-07

Contents

Exercise 1 - Introduction to Simulation with Variance Estimation

Code for setting up datasets

```r
set.seed(11835945)
data.set.x1 <- rnorm(100)
set.seed(11835945)
data.set.x2 <- rnorm(100,mean=1000000)
```

To initialize this exercise, I define the two datasets of the slides with my matriculation number as random seed. We will see in this exercise, that despite the big difference in mean, the variance of these two datasets will practically be the same.

Compare the 4 algorithms against R's 'var' function as a gold standard regarding the quality of their estimates.

Fore better comparability, we compute the R var function to see our target value

```r
var.x1<- var(data.set.x1)
var.x2<- var(data.set.x2)
```

The variance of the first dataset x1 is 1.1853813 and variance of second dataset x2 is 1.1853813. The variance of both datasets is approximately the same value. We will now implement the 4 methods of the slides.

Implement all variants of variance calculation as functions.

Code for algorithm 1-4

```r
precise <- function(data){
  mean.data <- mean(data)
  n <- length(data)
  variance <- (sum( (data-mean.data)**2) ) / (n-1)
  return(variance)
}
excel <- function(data){
  n <- length(data)
  P1 <- sum(data**2)
  P2 <- 1/n * sum(data)**2
  variance <- (P1 - P2) / (n-1)
  return(variance)
}
shift <- function(data, c =NaN){
  n <- length(data)
  if (is.na(c))    c<- data[1]
  P1 <- sum( (data-c)**2 )
  P2 <- 1/n * sum( data-c )**2
  variance <- (P1-P2) / (n-1)
  return(variance)
}
online.update <- function(n,mean.old,var.old,new.value){
  new.mean <- mean.old + (new.value - mean.old) / n
  new.var <- var.old * (n-2)/(n-1) + (new.value - mean.old)**2 / n
  pair <- c(new.mean, new.var)
  return(c(new.mean, new.var))
}

online <- function(data){
  #step 1
  n<- length(data)
  mean.2 <- sum(data[1:2])/2
  variance.2 <- sum( (data[1:2] - mean.2)**2 )

  #step 2
  mean.update <- mean.2
  var.update <- variance.2
  for (i in 3:n){
    new.value <- data[i]
    result.update <- online.update(i, mean.update,var.update,new.value)
    mean.update<-result.update[1]
    var.update <- result.update[2]
  }
  var.final<- var.update
  return(var.final)
}
```

In this codeblock, we define all 4 algorithms - precise, excel, shift and online. I implemented them according to the definition in the slides. for the online algorithm I outsourced the update function as an additional function, which is used to calculate the elements of the iteration of step 2.

We get following table when we compute all variances with the different methods:

```
table <- data.frame(Dataset = c("x1","x2"), R_Variance = c(var.x1,var.x2),
                    precise = c(precise(data.set.x1),precise(data.set.x2)),
                    excel = c(excel(data.set.x1),excel(data.set.x2)),
                    shift = c(shift(data.set.x1),shift(data.set.x2)),
                    online = c(online(data.set.x1),online(data.set.x2)))
knitr::kable(table)
```

Dataset	R_Variance	precise	excel	shift	online
x1	1.185381	1.185381	1.185381	1.185381	1.185381
x2	1.185381	1.185381	1.185133	1.185381	1.185381

We can see, that all 4 methods achieve very accurate results. It seems, that the excel methods becomes a bit inaccurate when the numbers of the dataset are larger by scale.

Write a wrapper function which calls the different variants.

I want to evaluate, wheter the values are also equal to the var calculation method. Thus I define a wrapper and iterate over all data sets and methods. To compare for equality, I use the comparison methods of "==", "all.equal" and "identical" Code for wrapper

```
var.wrapper <- function(data, type="var") {
  f <- match.fun(type)
  return(f(data))
}
methods<- c("precise","excel","shift","online")
df <- data.frame(dataset=character(),var.method=character(),
                 compare.method=character(),result=character())
colnames(df)<- c("dataset","var.method","compare.method","result")
index<-1
for (method in methods){
    dataset=data.set.x1
    equal<-all.equal(var.wrapper(dataset),var.wrapper(dataset,method))
    operator <- var.wrapper(dataset)==var.wrapper(dataset,method)
    identic <- identical(var.wrapper(dataset),var.wrapper(dataset,method))

    df[index,]<-c("x1",method,"all.equal",equal)
    index=index+1
    df[index,]<-c("x1",method,"==",operator)
    index=index+1
    df[index,]<-c("x1",method,"identical",identic)
    index=index+1
    }

for (method in methods){
    dataset=data.set.x2
    equal<-all.equal(var.wrapper(dataset),var.wrapper(dataset,method))
    operator <- var.wrapper(dataset)==var.wrapper(dataset,method)
    identic <- identical(var.wrapper(dataset),var.wrapper(dataset,method))

    df[index,]<-c("x2",method,"all.equal",equal)
    index=index+1
    df[index,]<-c("x2",method,"==",operator)
    index=index+1
    df[index,]<-c("x2",method,"identical",identic)
    index=index+1
    }

knitr::kable(df)
```

dataset	var.method	compare.method	result
x1	precise	all.equal	TRUE
x1	precise	==	TRUE
x1	precise	identical	TRUE
x1	excel	all.equal	TRUE
x1	excel	==	TRUE
x1	excel	identical	TRUE
x1	shift	all.equal	TRUE
x1	shift	==	TRUE
x1	shift	identical	TRUE
x1	online	all.equal	TRUE
x1	online	==	FALSE
x1	online	identical	FALSE
x2	precise	all.equal	TRUE
x2	precise	==	TRUE
x2	precise	identical	TRUE
x2	excel	all.equal	Mean relative difference: 0.0002097887
x2	excel	==	FALSE
x2	excel	identical	FALSE
x2	shift	all.equal	TRUE
x2	shift	==	TRUE
x2	shift	identical	TRUE
x2	online	all.equal	TRUE
x2	online	==	FALSE
x2	online	identical	FALSE

When comparing with the table before, we can see, that in many cases, equality does not hold despite them appearing in the table. All.equal matches the results from the first table, while the other comparison method do not always hold the equality. We can also see, that the algorithms applied to dataset 2 do not always return true. However, as can be seen in the table before, in practise these difference are so small that in most application they can be neglected, except Dataset 2 with Algorithm excel, which has as only combination a visible difference ## Compare the computational performance of the 4 algorithms against R's 'var' function as a gold standard and summarise them in tables and graphically. For the computational performance we will use the library microbenchmark. This library helps us track the computing times of the different algorithms. Alternatively one could also use the rbenchmark package or measure the time directly with the Sys.time() function to measure the computational performance. We begin by analysing dataset 1

```r
library(microbenchmark)
benchmark.x1 <- microbenchmark(
  var = var.wrapper(data.set.x1),
  precise = var.wrapper(data.set.x1, "precise"),
  excel = var.wrapper(data.set.x1, "excel"),
  shift = var.wrapper(data.set.x1, "shift"),
  online = var.wrapper(data.set.x1, "online")
)
knitr::kable(summary(benchmark.x1), caption = "Dataset x1")
```

Table 3: Dataset x1

expr	min	lq	mean	median	uq	max	neval
var	8.8	9.70	14.042	11.50	13.60	112.7	100
precise	8.1	9.00	11.966	10.25	12.70	52.3	100
excel	3.3	3.80	4.513	4.40	5.00	14.5	100
shift	3.8	4.20	5.768	4.90	6.00	25.6	100
online	149.6	154.15	169.085	160.25	182.25	252.0	100

As can be seen above, the excel method is by far the fastest method, followed by shift and then precise. All these 3 algorithms outperfrom the native var method. Online is significantly the worst algorithm which most likely is due to for loops having poor performance in R and should be avoided if possible. However, online method may be very useful when working with live data.

Now we examine dataset 2's performance

```r
library(microbenchmark)

benchmark.x2 <- microbenchmark(
  var = var.wrapper(data.set.x2),
  precise = var.wrapper(data.set.x2, "precise"),
  excel = var.wrapper(data.set.x2, "excel"),
  shift = var.wrapper(data.set.x2, "shift"),
  online = var.wrapper(data.set.x2, "online")
)
knitr::kable(summary(benchmark.x2), caption = "Dataset x2")
```

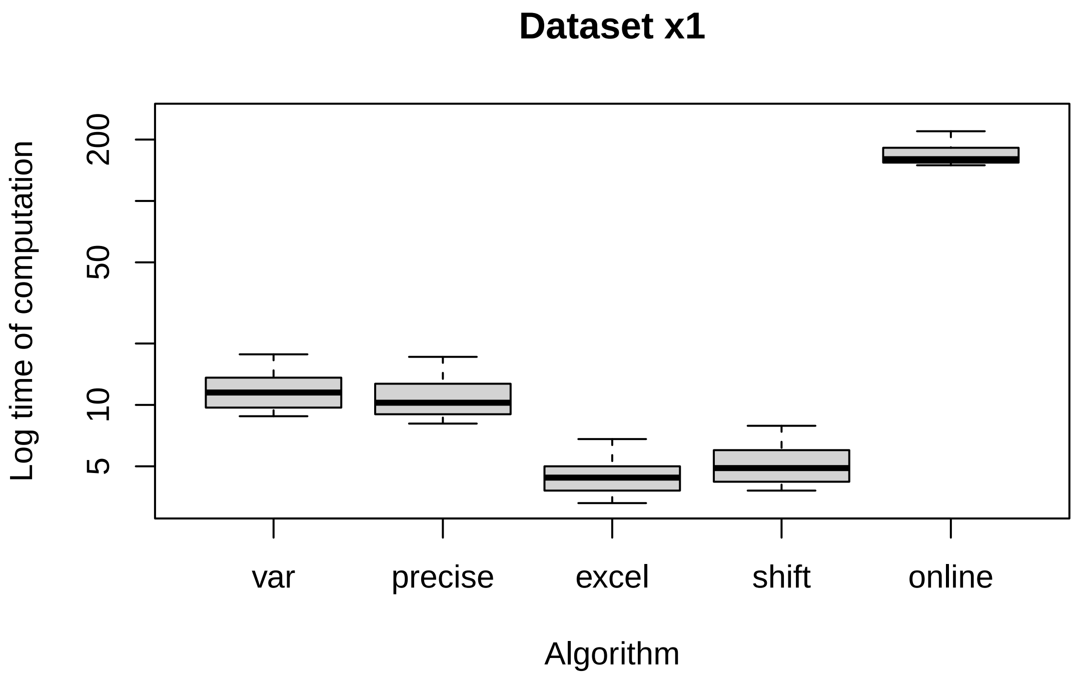
Table 4: Dataset x2

expr	min	lq	mean	median	uq	max	neval
var	9.0	10.00	14.415	11.00	14.75	79.1	100
precise	8.1	9.20	18.021	10.60	13.95	198.9	100
excel	3.3	3.70	6.047	4.20	5.00	67.3	100
shift	3.8	4.35	8.131	4.80	6.15	62.8	100
online	148.9	152.00	166.477	157.95	165.00	253.8	100

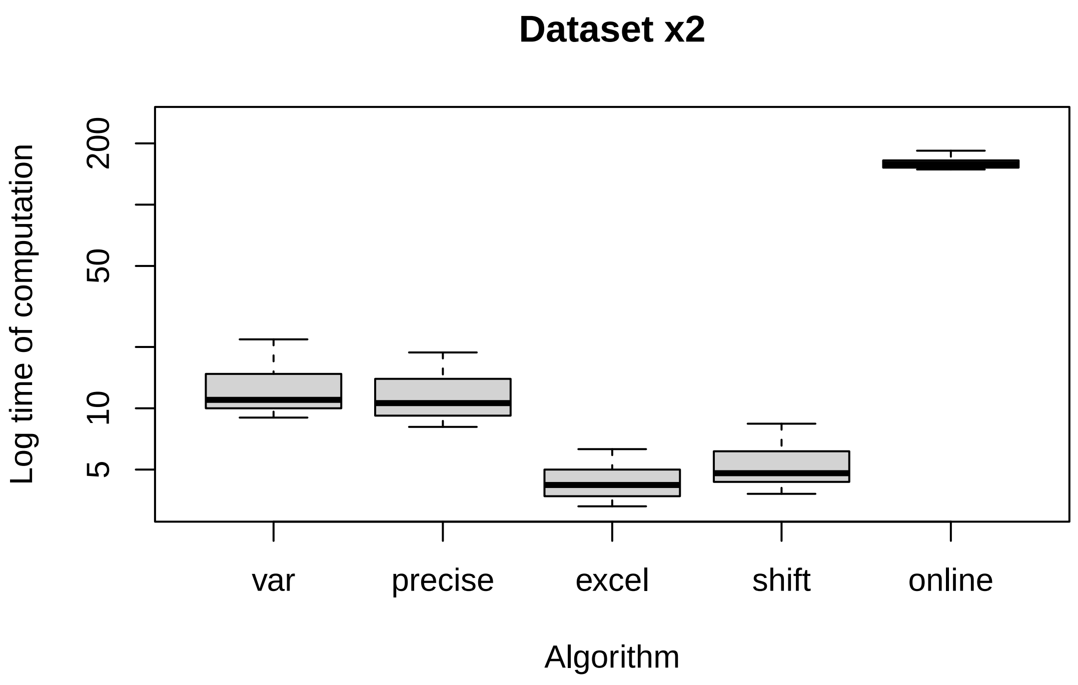
Since the dataset 2 has the same size as dataset 1, I expect the times to not differ alot from dataset 1. This assumption holds true when looking at the table.

When looking at following boxplots, this is easier to spot:

```r
boxplot(benchmark.x1, main="Dataset x1",xlab="Algorithm",
        ylab="Log time of computation",outline=FALSE)
```



```r
boxplot(benchmark.x2, main="Dataset x2",xlab="Algorithm",
        ylab="Log time of computation",outline=FALSE)
```



Investigate the scale invariance property for different values and argue why the mean is performing best as mentioned with the condition number.

As we can see in the results above, arbitrarily increasing the mean does not have a significant effect on computation time. The biggest influence factor on computation speed is the size of the data set, which in this case is 100 for both sets. The scale invariance property is a result of the definition of the Variance (mean is a linear function (Expected value is a linear function):

$$\sigma^2_{x-c} = E((x-c-E(x-c))^2) = E((x-c-E(x)+c)^2) = E((x-E(x))^2) = \sigma^2_x$$

In the shift method using the mean of the dataset is optimal, as this will decrease the size of the sum of the differences in the computation the most on average. Furthermore, the coundition number is smallest when using c as the mean thus making the shift algorithm very robust and stable. Therefore, computing times and especially accuracy are lowered, as c increases. Following example illustrates this effect:
shift with $c = mean(x_1)$: 1.1853813,
shift with $c = 10^7$: 1.1919192.
shift with $c = 10^8$: 2.5858586.

Compare the results according to the instructions provided by Comparison I and Comparison II of the slides.

Comparison Slide 1 and 2 are already described in the task “Compare the computational performance of the 4 algorithms against R’s ‘var’ function as a gold standard and summarise them in tables and graphically.”

Provide your results in table format and graphical format comparable to the example in the slides.

This was already done in the task "Compare the computational performance of the 4 algorithms against R's 'var' function as a gold standard and summarise them in tables and graphically." ## Compare condition numbers for the 2 simulated data sets and a third one where the requirement is not fulfilled, as described during the lecture. First, we define a function that computes the condition number for a given dataset.

```r
get.S <- function(data){
  mean.data <- mean(data)
  S<- sum((data - mean.data)**2)
}
cond.num <- function(data, shift=0) {
  n <- length(data)
  mean.data <- mean(data)
  S<- get.S(data)
  cond<- sqrt(1 + (mean.data-shift)**2 * n/S)
  return(cond)
}
approx.num <- function(data){
  n <- length(data)
  mean.data <- mean(data)
  S<- get.S(data)
  value<- mean.data * sqrt(n/S)
}

set.seed(11835945)
data.set.x3 <- rnorm(100,mean=0.000001)
df.cond <- data.frame(dataset = c("x1","x2","x3"),vars = c(var(data.set.x1),var(data.set.x2),var(data.se
                      `condition n.`=c(cond.num(data.set.x1),cond.num(data.set.x2),cond.num(data.set.x3)
                      `condition approximation`= c(approx.num(data.set.x1),approx.num(data.set.x2),appro
                      S = c(get.S(data.set.x1),get.S(data.set.x2),get.S(data.set.x3)), `shifted conditio
knitr::kable(df.cond)
```

dataset	vars	means	condition.n.	condition.approximation	S	shifted.condition.mean
x1	1.185381	1.356424e-01	1.007809e+00	1.252129e-01	117.3527	1
x2	1.185381	1.000000e+06	9.231100e+05	9.231100e+05	117.3527	1
x3	1.185381	1.356434e-01	1.007809e+00	1.252138e-01	117.3527	1

We added a third data set, with a very small mean of one millionth. This table shows, that the condition number for the variance is high, when we have a higher mean. Furthermore, the gap between the estimation of the condition number to the real condition number increases, if we have small or very small means. For example,
difference in set 1: 0.8825958,
difference in set 3: 0.882595.

The estimation is more accurate for higher means as can be seen in the table for dataset x2. The higher condition number of dataset 2 indicates, that the output changes more significant, if the input is slightly changed. We have also seen that the output of the excel method with input data set 2 differs noticably from the other dataset and methods.

The reason why the sample means of data set x1 and x3 are close to each other is that they are very close to eachs other and therefore similiar in computation. To make the means more accurate to the means by which they were created, one has to increase the sample size.

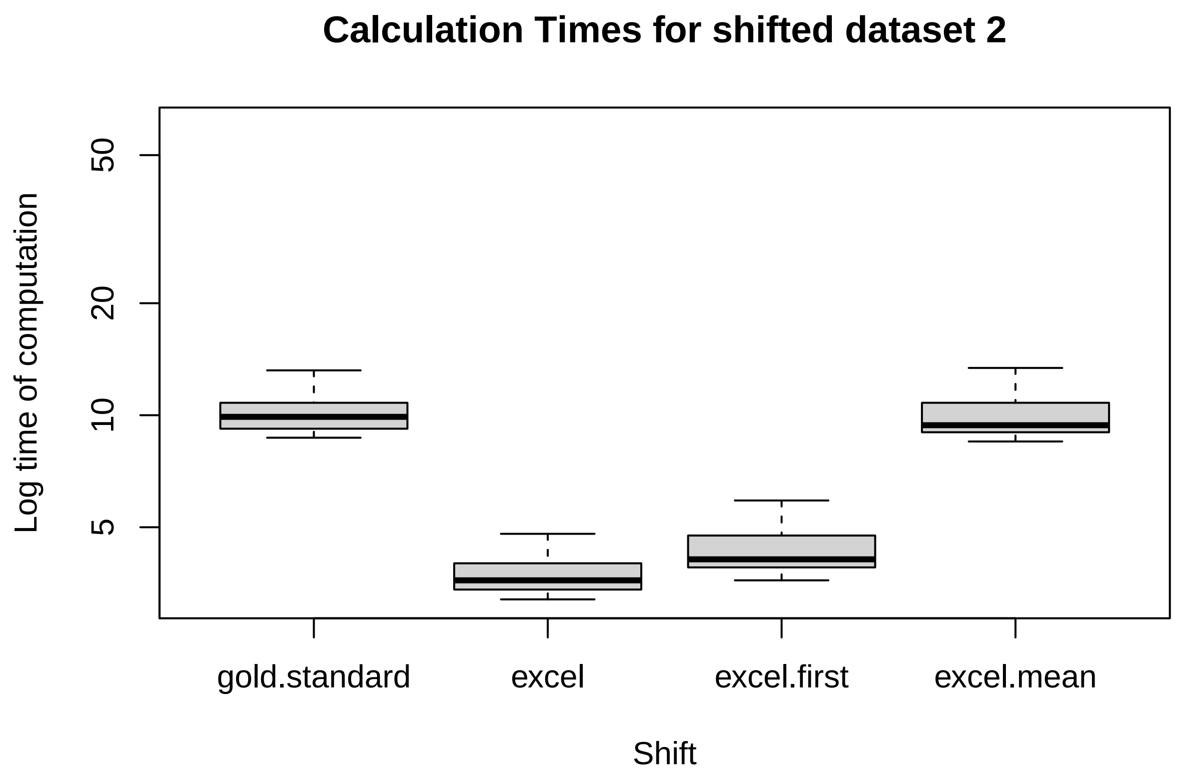
Regarding condition number for shifted variance, shifting by the mean results in the best condition number, 1. This makes sense, when looking at the definition of the condition number for shifted variance. If one looks into the function, we substract the shift of the mean. If they are equal, we get 0 and therefore only 1 remains in the squareroot.

Performance check on excel with shifted dataset 2

At last, I want to check the performance of the excel algorithm when shifting dataset 2 by its mean or other values (since I expect excel to be more correct on the mean-shifted dataset 2 and excel is the fastest method). We know, that the variance for dataset 2 with excel is very close to the real one of we shift by the mean:
"gold standard" variance of dataset 2: 1.1853813
excel variance of mean-shifted dataset 2: 1.1853813

```r
benchmark.x2.shifted <- microbenchmark(
  gold.standard = var.wrapper(data.set.x2),
  excel = var.wrapper(data.set.x2, "excel"),
  excel.first = var.wrapper(data.set.x2-data.set.x2[1], "excel"),
  excel.mean = var.wrapper(data.set.x2-mean(data.set.x2), "excel"))

boxplot(benchmark.x2.shifted,outline=FALSE,xlab="Shift",
        ylab="Log time of computation",main="Calculation Times for shifted dataset 2")
```



We observe, that the computation time takes significantly longer when shifting by the mean instead of other values. However, we also get most accurate results which is favorable. But since we are not really significantly faster than the gold standard r method, I would not recommend using the excel algorithm.