Statistical Simulation and Computerintensive Methods

Exercise 1

Markus Kiesel | 1228952

11.10.2021

1. Quality of variance calculation algorithms

For the first section we compare four different variants of variance calculation algorithms against R's "var" function as a gold standard regarding the quality of their estimates.

Implementaition of variance calculation algorithms

The four algorithms are implemented as functions in the following subsections.

Algorithm 1

Function for the precise variance algorithm (two - pass algorithm - variance calculation in R).

```
var_precise <- function(samples) {
  mean <- mean(samples)
  n <- length(samples)
  # apply for each sample in the samples vector
  samples <- (samples - mean) ^ 2
  # divide sum by (n-1)
  s <- sum(samples) / (n - 1)
}
```

Algorithm 2

Function for the excel variance algorithm (one - pass algorithm - previously variance calculation in Excel).

```
var_excel <- function(samples) {
  n <- length(samples)
  # sum of squared values
  p1 <- sum(samples ^ 2)
  # squared sum of values divided by n
  p2 <- (sum(samples) ^ 2) / n
  s <- (p1 - p2) / (n - 1)
}
```

Algorithm 3

Function for the shift variance algorithm (shifted one - pass algorithm). As default value for c (shift parameter) we use the mean of samples.

```
var_shift <- function(samples, c = mean(samples)) {
  n <- length(samples)
  # sum of squared samples - shift parameter
```

```
  p1 <- sum((samples - c) ^ 2)
  # squared sum of values (- shift parameter) divided by n
  p2 <- (sum(samples - c) ^ 2) / n
  s <- (p1 - p2) / (n - 1)
}
```

Algorithm 4

Function for the online algorithm.

```
var_online <- function(samples) {
  # calculate mean for first 2 values
  xn <- mean(samples[1:2])
  # calculate sn for first 2 values (reusing the precise algorithm)
  sn <- var_precise(samples[1:2])
  # loop for all further values, update xn and sn for every
  # value in the samples vector
  for (n in 3:length(samples)) {
    sn <- ((n - 2) / (n - 1)) * sn + ((samples[n] - xn) ^ 2 / n)
    xn <- xn + (samples[n] - xn) / n
  }
  sn
}
```

Data sets creation

Next we create four datasets with normal distribution for our experiments. We use different means for all datasets.

```
set.seed(1228952)
# create random dataset with normal distribution of a 100 values with mean 0
ds1 <- rnorm(100)
set.seed(1228952)
# create random dataset with normal distribution of a 100 values with mean 10^6
ds2 <- rnorm(100, mean=10^6)
set.seed(1228952)
# create random dataset with normal distribution of a 100 values with mean 10^10
ds3 <- rnorm(100, mean=10^10)

datasets <- list('mean=0' = ds1,
                 'mean=10^6' = ds2,
                 'mean=10^10' = ds3)
```

Comparison of values

For easier comparison we use a wrapper function to call an algorithm for all datasets.

```
# call an algorithm for all datasets in datasets list
alg_by_ds <- function(algortihm){
  results <- c()
  for (ds in datasets) {
    results <- append(results, algortihm(ds))
  }
  results
}
```

Now we use our wrapper function to calculate all results for the variance by algorithm and dataset. The row gold_standard refers to R's "var" function.

The comparison table shows that almost all results for the variance are the same.

The excel algorithms result for the dataset with a large large (10^6) mean deviates from the gold standard and the algorithm breaks for the dataset with the very large (10^10) mean.

Further the result from the online algorithm deviates slightly from the gold standard for the dataset with the very large (10^10) mean.

```
gold_standard <- alg_by_ds(var)
precise <- alg_by_ds(var_precise)
excel <- alg_by_ds(var_excel)
shift <- alg_by_ds(var_shift)
online <- alg_by_ds(var_online)

comparison_df <- data.frame(gold_standard, precise, excel, shift, online,
                            row.names = names(datasets))
```

Table 1: Comparison of Variance Algorithms

	gold_standard	precise	excel	shift	online
mean=0	1.095721	1.095721	1.095721	1.095721	1.095721
mean=10^6	1.095721	1.095721	1.095802	1.095721	1.095721
mean=10^10	1.095721	1.095721	0.000000	1.095721	1.095720

Comparison functions

Instead of comparing the results of the algorithms by comparing their values visually we next use three different comparison "functions" R offers. For this we use another wrapper function which calls each algorithm for all datasets and compares the result to R's "var" function.

```
compare_alg_by_ds <- function(algortihm, compare_func = NULL){
  results <- c()
  for (ds in datasets) {
    if (is.null(compare_func)) {
      compare <- var(ds) == algortihm(ds)
    } else {
      compare <- compare_func(var(ds), algortihm(ds))
    }
    results <- append(results, compare)
  }
  results
}
```

First we use the "==" operator to compare results.

Note that for numerical and complex values, == and != do not allow for the finite representation of fractions, nor for rounding error.

We see that the precise algorithm always has the exact same result as R's "var" function whereas the online algorithm never returns exactly the same results.

The shift algorithm also hast the exact same results as the gold standard except for the dataset with the very large mean. The excel algorithm only reaches the exact same result as R's "var" function for the dataset with mean 0.

```
precise <- compare_alg_by_ds(var_precise)
excel <- compare_alg_by_ds(var_excel)
shift <- compare_alg_by_ds(var_shift)
online <- compare_alg_by_ds(var_online)

comparison_df <- data.frame(precise, excel, shift, online, row.names = names(datasets))
```

Table 2: Comparison with ==

	precise	excel	shift	online
mean=0	TRUE	TRUE	TRUE	FALSE
mean=10^6	TRUE	FALSE	TRUE	FALSE
mean=10^10	TRUE	FALSE	FALSE	FALSE

Next, we use the identical function to compare the results.

This function tests if two objects are exactly equal and reaches the same results as using the "==" operator.

```
precise <- compare_alg_by_ds(var_precise, identical)
excel <- compare_alg_by_ds(var_excel, identical)
shift <- compare_alg_by_ds(var_shift, identical)
online <- compare_alg_by_ds(var_online, identical)

comparison_df <- data.frame(precise, excel, shift, online, row.names = names(datasets))
```

Table 3: Comparison with identical

	precise	excel	shift	online
mean=0	TRUE	TRUE	TRUE	FALSE
mean=10^6	TRUE	FALSE	TRUE	FALSE
mean=10^10	TRUE	FALSE	FALSE	FALSE

Last, we use the all.equal function to compare the results.

This function tests for "near equality". If they are different, comparison is still made to some extent, and a report of the differences is returned.

Using this function we best see the differences in the results for the different algorithms. The comparison table shows the same results we already discussed when we compared the values visually that the excel algorithm breaks for the dataset with a very large mean and differs already for the dataset with large mean. Also the online algorithms differs for values with a very large mean.

```
precise <- compare_alg_by_ds(var_precise, all.equal)
excel <- compare_alg_by_ds(var_excel, all.equal)
shift <- compare_alg_by_ds(var_shift, all.equal)
online <- compare_alg_by_ds(var_online, all.equal)

comparison_df <- data.frame(precise, excel, shift, online, row.names = names(datasets))
```

Table 4: Comparison with all.equal

	precise	excel	shift	online
mean=0	TRUE	TRUE	TRUE	TRUE
mean=10^6	TRUE	Mean relative difference: 7.403983e-05	TRUE	TRUE
mean=10^10	TRUE	Mean relative difference: 1	TRUE	Mean relative difference: 1.642386e-07

2. Computational performance comparison

For the next experiment we compare the computational performance of the previous implemented algorithms and R's "var" function by using the microbenchmark library.

First, we have a look at the comparison table the microbenchmark library provides. For this we use the dataset with the mean of 0.

```
bench1 <- microbenchmark(var(ds1),
                         var_precise(ds1),
                         var_excel(ds1),
                         var_shift(ds1),
                         var_online(ds1))
```

```
## Unit: microseconds
##              expr    min      lq     mean  median      uq    max neval
##          var(ds1)  9.447 10.1330 11.29611 10.7080 11.0880 61.138   100
##  var_precise(ds1)  4.850  5.2065  5.47543  5.3910  5.6825  8.721   100
##    var_excel(ds1)  1.810  2.1685  2.33984  2.3015  2.4795  4.286   100
##    var_shift(ds1)  5.774  6.1635  6.51628  6.3970  6.7610 11.634   100
##   var_online(ds1) 24.577 25.0165 25.97266 25.3080 25.6475 73.576   100
```

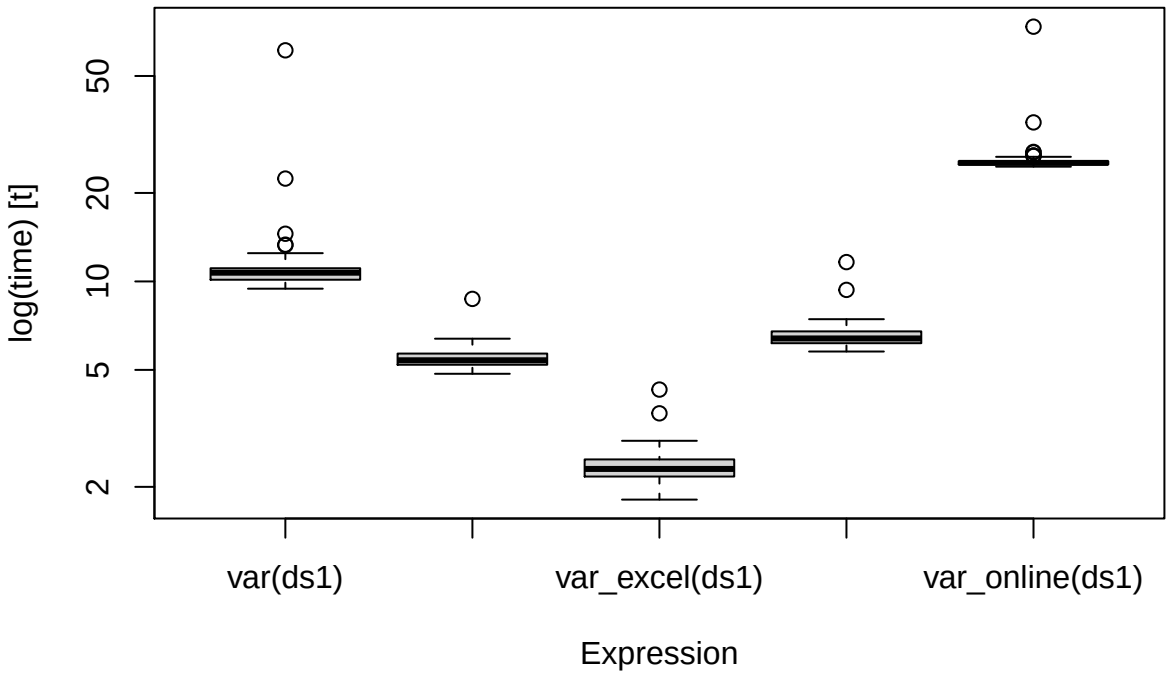
Table 5: Comparison of Computational Performance (microbenchmark)

expr	min	lq	mean	median	uq	max	neval
var(ds1)	9.447	10.1330	11.29611	10.7080	11.0880	61.138	100
var_precise(ds1)	4.850	5.2065	5.47543	5.3910	5.6825	8.721	100
var_excel(ds1)	1.810	2.1685	2.33984	2.3015	2.4795	4.286	100
var_shift(ds1)	5.774	6.1635	6.51628	6.3970	6.7610	11.634	100
var_online(ds1)	24.577	25.0165	25.97266	25.3080	25.6475	73.576	100

We can view these results even better in a boxplot.

The fastest algorithm is the excel algorithm with twice the speed the next algorithm can compute the results. The precise and the shift algorithm are both very fast with results computed in about 6 microseconds. The online algorithm is the slowest of our implemented algorithms. It needs four times as long as the previous mentiond methodes. Surprising is that R's "var" algorithm is the slowest one and needs still longer to compute the results than the online algorithm. The reason for this is possibly factors that are taken into account in R's "var" implementation that makes the algorithm more robust.

```
boxplot(bench1)
```



Using the dataset with the large mean we have almost the same results for the computational performances.

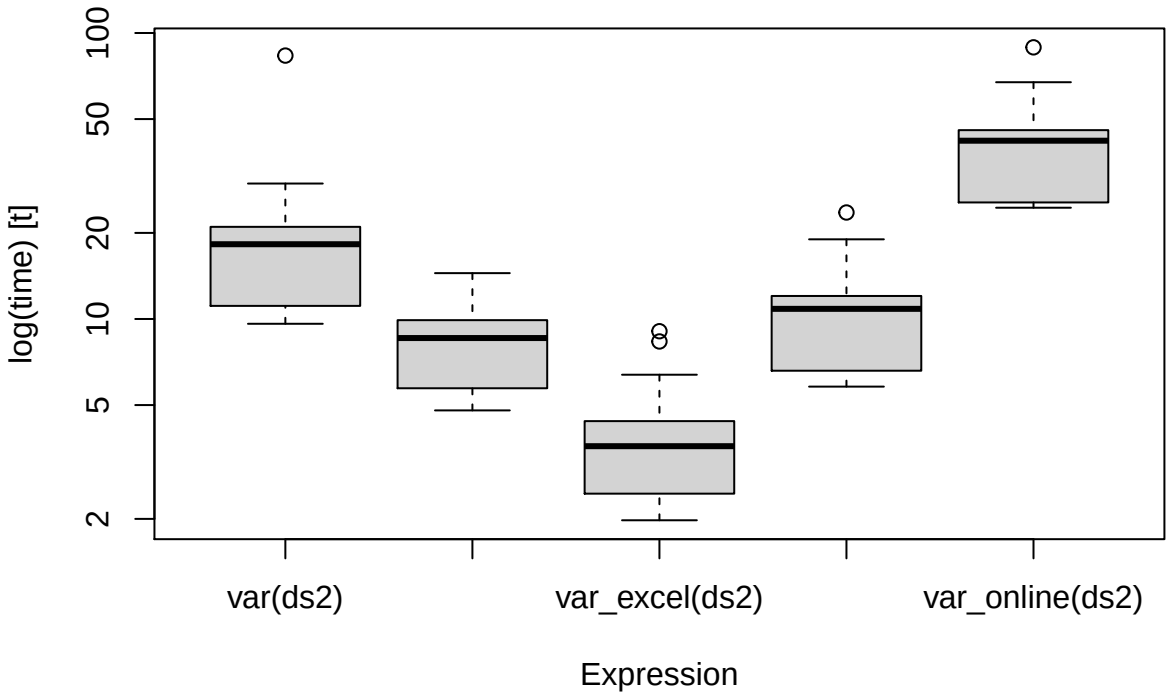
```
bench2 <- microbenchmark(var(ds2),
                         var_precise(ds2),
                         var_excel(ds2),
                         var_shift(ds2),
                         var_online(ds2))
```

Unit: microseconds expr min lq mean median uq max neval var(ds2) 9.621 11.1120 17.76935 18.2550 20.9980 83.371 100 var_precise(ds2) 4.790 5.7245 8.19643 8.5780 9.9065 14.463 100 var_excel(ds2) 1.978 2.4505 3.64663 3.5910 4.3940 9.061 100 var_shift(ds2) 5.802 6.5940 10.00632 10.8500 12.0370 23.569 100 var_online(ds2) 24.491 25.5565 38.16485 42.0115 45.7410 89.080 100

Table 6: Comparison of Computational Performance large mean

expr	min	lq	mean	median	uq	max	neval
var(ds2)	9.621	11.1120	17.76935	18.2550	20.9980	83.371	100
var_precise(ds2)	4.790	5.7245	8.19643	8.5780	9.9065	14.463	100
var_excel(ds2)	1.978	2.4505	3.64663	3.5910	4.3940	9.061	100
var_shift(ds2)	5.802	6.5940	10.00632	10.8500	12.0370	23.569	100
var_online(ds2)	24.491	25.5565	38.16485	42.0115	45.7410	89.080	100

```
boxplot(bench2)
```



3. Scale invariance property

In this experiment we investigate the scale invariance property for different values.

The closer the scale parameter is to the mean the more accurate the results will be but choosing a value inside the samples range will gauratnee the desired stability.

First, we create a wrapper function to compare different shift parameters.

```
compare_shift <- function(ds, shift_pars, compare_func = NULL){
  results <- c()
  for (c in shift_pars) {
    if (is.null(compare_func)) {
      compare <- var(ds) == var_shift(ds, c)
    } else {
      compare <- compare_func(var(ds), var_shift(ds, c))
    }
    results <- append(results, compare)
  }
  results
}
```

Now let us have a look at the range of our values. All values between these two should lead to a stable result.

```
# range for dataset 1
print(min(ds1))
```

```
## [1] -3.374366
```

```
print(max(ds1))
```

```
## [1] 2.722139
```

If we compare the values for different shift parameters we see that only the mean and a value in the range of the dataset produces the exect same result as R's "var" implementatiotn. We further see that the algorithm

breaks down when the shift parameter is much larger than the mean.

```
shift_pars = c(mean(ds1), ds1[10], 10, 10^3, 10^5, 10^7, 10^10)
comp_identical <- compare_shift(ds1, shift_pars)
comp_equal <- compare_shift(ds1, shift_pars, all.equal)

row_names <- c("mean", "value in range", "10", "10^3", "10^5", "10^7", "10^10")
comparison_df <- data.frame(comp_identical, comp_equal, row.names = row_names)
```

Table 7: Comparison of shift parameter c

	comp_identical	comp_equal
mean	TRUE	TRUE
value in range	TRUE	TRUE
10	FALSE	TRUE
10^3	FALSE	TRUE
10^5	FALSE	Mean relative difference: 8.942001e-07
10^7	FALSE	Mean relative difference: 0.004391219
10^10	FALSE	Mean relative difference: 1

4. Condition numbers

In this experiment we compare condition numbers for the simulated data sets and a third one where the requirement is not fulfilled.

First we implement the function to compute the condition number.

```
# compute condition number for dataset
condition_number <- function(samples) {
  mean <- mean(samples)
  s <- sum((samples - mean) ^ 2)
  k <- sum(samples ^ 2) / sqrt(s)
}
```

Next we create a dataset with very small values. The other two datasets that we use are the datasets created for the first experiment.

```
set.seed(1228952)
ds4 <- rnorm(100)
ds4 <- ds4 / 10^6
print(mean(ds4))
```

```
## [1] -8.496327e-08
```

If we compare the condition numbers for the three datasets we see that the condition number for our dataset with very small values is < 1.

```
cn_1 <- condition_number(ds1)
cn_2 <- condition_number(ds2)
cn_3 <- condition_number(ds4)

comparison_df <- data.frame(cn_1, cn_2, cn_3, row.names = "condition number")
```

Table 8: Comparison of condition numbers

	cn_1	cn_2	cn_3
condition number	10.48451	9.601352e+12	1.05e-05