Assignment 1: Variance Estimation

Statistical Simulation and Computerintensive Methods

Joan Salvà

October 2022

The goal of the assignment is to implement and compare the performance and accuracy of four diferent methods to calculate the finite variance of a sample.

1. Precise

We will compute the variance directly from the formal definition by precomputing the mean and then iterating over the sample to sum the difference squares.

The mathematical formula would be:

$$S^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2,$$

where $\bar{x}$ corresponds to the mean of the sample.

```
precise <- function(sample) {
  n <- length(sample)

  # Mean
  res <- 0
  for (i in 1:n) {
    res <- res + sample[i]
  }
  mean <- res / n

  # Variance
  res <- 0
  for (i in 1:n) {
    res = res + (sample[i] - mean)^2
  }
  return(res / (n - 1))

}
```

2 & 3: Excel and Shifted

We will code a one-pass algorithm that uses the fact that we can write

$$S^2 = \frac{P_1 - P_2}{n-1}, \quad \text{where} P_1 = \sum_{i=1}^{n} x_i^2 P_2 = \frac{1}{n}\left(\sum_{i=1}^{n} x_i\right)^2$$

Since we know that the variance estimate is invariant under translations, i.e., $\sigma_x^2 = \sigma_{x+c}^2$, the following alternative definitions for P_1 and P_2 would also work for any $c \in \mathbb{R}$:

$$P_1 = \sum_{i=1}^{n}(x_i - c)^2 P_2 = \frac{1}{n}\left(\sum_{i=1}^{n} x_i - c\right)^2$$

The *excel* to estimate the variance corresponds to taking $c = 0$, and the *shift* corresponds to testing different values for the parameter c. We will first code *variance_3* to then define *variance_2* as a particular case.

```
shifted <- function(sample, c = NULL) {
  n <- length(sample)
  if (is.null(c)) {
    c <- sample[1]
  }

  p_1 <- 0
  p_3 <- 0
  for (i in 1:n) {
    p_1 = p_1 + (sample[i] - c)^2
    p_3 = p_3 + sample[i] - c
  }
  p_2 <- p_3^2 / n
  return((p_1 - p_2) / (n - 1))

}

excel <- function(sample) {
  return(shifted(sample, c=0))
}
```

4. Online

In this case, we assume that the data points arrive sequentially and that we want to have an estimate of the variance for the available data at all times.

The recursive formula works as follows:

$$S_1^2 = 0 \bar{x_1} = x_1 \bar{x}_n = \bar{x}_{n-1} + \frac{x_n - \bar{x}_{n-1}}{n}, \quad n > 1 S_n^2 = \frac{n-2}{n-1} S_{n-1}^2 + \frac{(x_n - \bar{x}_{n-1})^2}{n}, \quad n > 1$$

In the implementation, we are going to use the formal definition of variance for the cases $n \leq 2$ and the general formula for the other cases.

```
online <- function(sample) {
  n <- 2
  mean <- 0.5 * (sample[1] + sample[2])
  s_squared <- (sample[1]-mean)^2 + (sample[2] - mean)^2

  for (i in 3:length(sample)) {
    n <- n + 1
    s_squared <- (n - 2) / (n - 1) * s_squared + 1/n * (sample[i] - mean)^2
    mean <- mean + 1/n * (sample[i] - mean)
  }
  return(s_squared)
}
```

Comparison

It makes sense to define a wrapper function that estimates the variance using the five different methods:

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

We now define the three different sample vectors we will test our calculations on. We fix the sample size to 100 and set the means to 0, 10^3, and 10^6.

```
library(microbenchmark)
matrikelnummer <- 12223411
set.seed(matrikelnummer)
x1 <- rnorm(100)
x2 <- rnorm(100, mean=1e3)
x3 <- rnorm(100, mean=1e6)
```

Table 1: Variance calulation

	precise	excel	shifted	online	var
x1	1.0537376	1.0537376	1.0537376	1.0537376	1.0537376
x2	0.8719542	0.8719542	0.8719542	0.8719542	0.8719542
x3	0.8543821	0.8540088	0.8543821	0.8543821	0.8543821

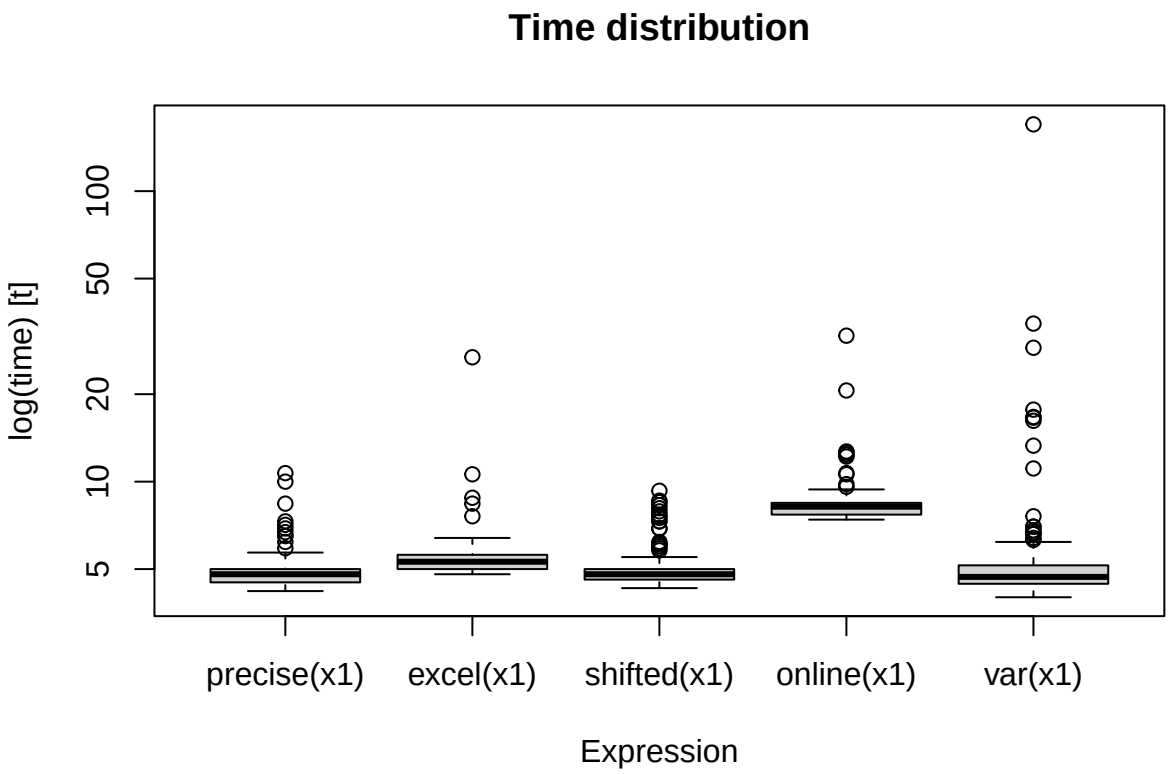
The table shows that the *excel* method is not robust as it returns values that differ from all the other ones as the mean of the sample grows.

Computation time

The *microbenchmark* library provides an easy way of comparing the runtime of different function calls. For this example, we will call the functions 200 times each.

Table 2: Summary table

expr	min	lq	mean	median	uq	max	neval
precise(x1)	4.2	4.50	4.8990	4.8	5.00	10.7	200
excel(x1)	4.8	5.00	5.5015	5.3	5.60	26.8	200
shifted(x1)	4.3	4.60	5.0120	4.8	5.00	9.3	200
online(x1)	7.4	7.70	8.4640	8.2	8.45	31.8	200
var(x1)	4.0	4.45	6.2555	4.7	5.15	169.8	200



The results of the test show that the fastest implementation is the *shifted* one, closely followed by the *precise* and the *excel*. The slowest one is the *online*, but we have to acknowledge that it is not a batch method so it can't exploit the good properties of batch data.

Note: we could create a custom function to compute the runtime of a function using the `Sys.time()` method. The code below captures the idea. the libraries *tictok* or *rbenchmark* provide similar resources as *microbenchmark*.

```
print_runtime <- function(f) {
  a <- Sys.time()
  f()
  print(Sys.time() - a)
}
```

Condition number and numerical stability

The condition number for the variance is defined as

$$\kappa = \sqrt{\frac{\sum_{i=1}^{n} x^2}{S}} = \sqrt{1 + \frac{\bar{x}^2 n}{S}} S = \sum_{i=1}^{n} (x_i - \bar{x})^2$$

Note that for the case of the shifted variance, it becomes

$$\sqrt{1 + \frac{(\bar{x} - c)^2 n}{S}}$$

The *precise*, *excel*, and *online* implementations share the same condition number so we cannot infer qualities of the different versions from the condition number.

Finding the most stable version of the *shifted* algorithm goes through finding the value c that minimizes the condition number. The global minimum of the function is $c = \bar{x}$. Taking an estimate of the mean as c, the sample becomes perfectly conditioned and the *shifted* algorithm should outperform the others in terms of stability.

Stability experiment

Despite the fact that we analytically know the optimal value of c, let's test the stability of the *shifted* algorithm for different values of c. We will test it on the sample with mean 10^6.

We define a range of values for c and for each of them, we will compare a 500 times the algorithm estimation of the variance against the same computation on a perturbed sample. Taking the mean relative difference for each c gives the following plot.

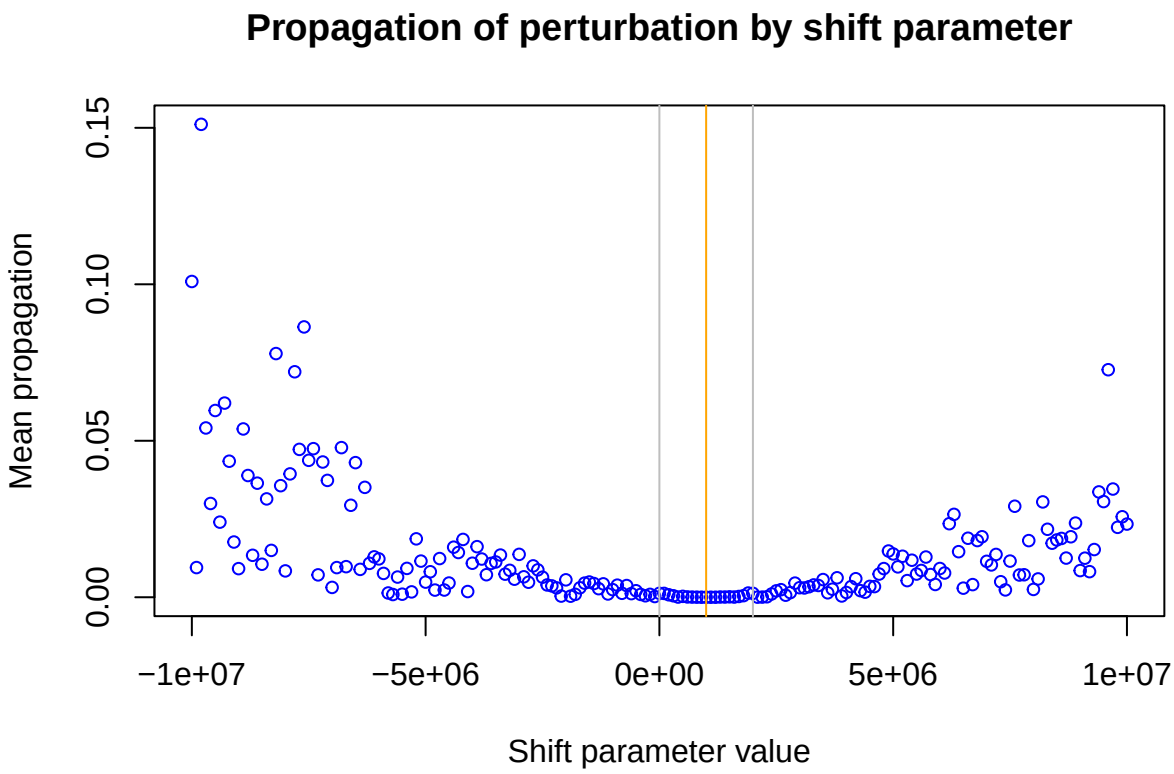
```
set.seed(matrikelnummer)
perturbation <- 0.000001
no_tests_per_c <- 500
x2 <- rnorm(100, mean=1e6)
res <- c()

# Domain of values to try
c_values <- seq(-10 * mean(x2), 10 * mean(x2), 0.1e6)


for (c in c_values) {
  var_value_wout_perturbation <- shifted(x2, c)
  changes <- c()
  for (i in 1:1000) {
    x <- x2 + perturbation * rnorm(length(x2))
    var <- shifted(x, c)

    changes <- c(changes, var / var_value_wout_perturbation - 1)

  }
  res <- c(res, mean(changes))
}
```



It is clear that large absolute values of the shift parameter lay worse stability. The plotted function, despite having a random component, should respect symmetry. Using that and visual inspection, one could sense that the minimum is close to the orange line, that happens to be the estimated mean of the sample (10^6). Our results support the analytic solution to the stability problem discussed before.

Note, however, that in a fairly wide interval of values for c also provide a very stable version of the algorithm. In the plot, the vertical grey lines define the region $|c - \bar{x}| \leq 10^6$. As pointed by Chan T, Et al. in *Algorithms for computing the sample variance*, despite not optimal, any value of c in the range of the sample would provide a very similar and close to optimal stability.