Assignment 1

Introduction to Simulation with Variance Estimation

Roman Grebnev

2022-10-10

Task 1

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

- Implement all variants of variance calculation as functions.
- Write a wrapper function which calls the different variants.

Import libraries

```
library("ggplot2")
library("microbenchmark")
```

Two-pass algorithm

Implementation of **two-pass algorithm**

```
variance_two_pass <- function(vector){
  n = length(vector)
  sample_mean <- sum(vector)/n
  variance <- 1/(n-1) * sum((vector - sample_mean)^2)
  return(variance)
}
```

Excel algorithm

Implementation of **excel algorithm**

```
variance_excel <- function(vector){
  n = length(vector)
  p1 = sum(vector^2)
  p2 = 1/n * sum(vector)^2
  variance <- (p1 - p2)/(n-1)
  return(variance)
}
```

Shift algorithm

Implementation of **shift algorithm** with shift by the first element of the vector

```
variance_shift <- function(vector){
  n = length(vector)
  c = vector[1]
  p1 = sum((vector-c)^2)
```

```
  p2 = 1/n * (sum(vector-c))^2
  variance <- (p1 - p2)/(n-1)
  return(variance)
}
```

Online

Implementation of **online algorithm**

```
variance_online <- function(vector){

  xmn_prev = vector[1] + (vector[2] - vector[1])/2
  varmn_prev = ((vector[2] - vector[1])^2)/2

  for(i in 3:length(vector)) {
    xmn = xmn_prev + (vector[i] - xmn_prev)/i
    varmn = (i - 2)/(i - 1)*varmn_prev + ((vector[i] - xmn_prev)^2)/i

    xmn_prev = xmn
    varmn_prev = varmn
  }

  return(varmn)
}
```

Wrapper function, which produces the result of variance computation for each variance computation algorithm

```
variance_wrapper <- function(vector){
  var_two_pass <- variance_two_pass(vector)
  var_excel <- variance_excel(vector)
  var_shift <- variance_shift(vector)
  var_online <- variance_online(vector)
  return(c(var_two_pass, var_excel, var_shift, var_online))
}
```

Task 2

Compare the computational performance of the 4 algorithms against R's 'var' function as a gold standard and summarise them in tables and graphically.

Generate vectors with normally distributed values

```
set.seed(12202120)
x1 <- rnorm(100, mean = 0)
set.seed(12202120)
x2 <- rnorm(100, mean=1000000)
set.seed(12202120)
x3 <- rnorm(100, mean=0.000001)
```

Function for benchmark computation

```
micro_benchmark <- function(vect) {
  micro_benchmark_calc = microbenchmark(
    variance_two_pass(vect),
    variance_excel(vect),
    variance_shift(vect),
    variance_online(vect),
```

```
    var(vect)
    )
}
```

Table 1: Comparison of variation computation algorithms for datasets x1 and x2

	two-pass	excel	shift	online
variance_x1	1.254942	1.254942	1.254942	1.254942
variance_x2	1.254942	1.254893	1.254942	1.254942

Table 2: Benchmark summary for generated data with mean = 0

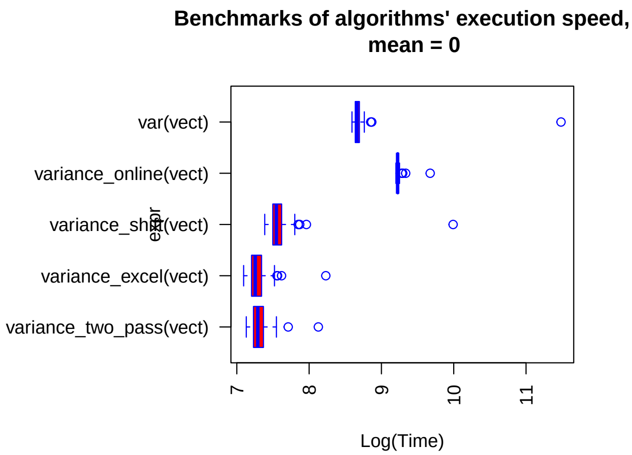
expr	min	lq	mean	median	uq	max	neval
variance_two_pass(vect)	1.250	1.3810	1.51773	1.4655	1.5850	3.386	100
variance_excel(vect)	1.205	1.3445	1.47312	1.4145	1.5440	3.755	100
variance_shift(vect)	1.614	1.8040	2.14572	1.8940	2.0385	21.842	100
variance_online(vect)	9.900	10.0840	10.23886	10.1375	10.2205	15.903	100
var(vect)	5.393	5.6420	6.75919	5.7680	5.9740	97.152	100

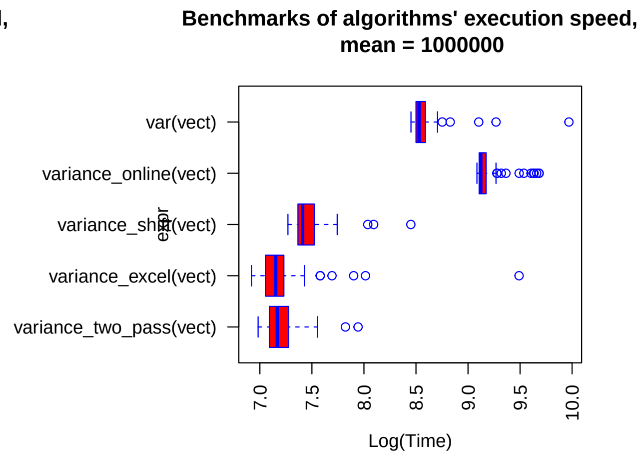
Table 3: Benchmark summary for generated data with mean = 1000000

expr	min	lq	mean	median	uq	max	neval
variance_two_pass(vect)	1.078	1.2010	1.36219	1.2965	1.4460	2.817	100
variance_excel(vect)	1.012	1.1575	1.44981	1.2765	1.3810	13.235	100
variance_shift(vect)	1.436	1.5810	1.76087	1.6535	1.8495	4.678	100
variance_online(vect)	8.829	9.0190	9.80795	9.1590	9.6415	16.055	100
var(vect)	4.684	4.9155	5.43554	5.0645	5.3795	21.369	100

```
knitr::opts_chunk$set(fig.width=20, fig.height=8)
par(mar=c(5, 12, 5, 5))
boxplot(log(time)~expr,
    data=benchmark_x1,
    main="Benchmarks of algorithms' execution speed,
    mean = 0",
    xlab="Log(Time)",
    col="red",
    border="blue",
    horizontal=TRUE,
    las=2
  )

par(mar=c(5, 12, 5, 5))
boxplot(log(time)~expr,
    data=benchmark_x2,
    main="Benchmarks of algorithms' execution speed,
    mean = 1000000",
    xlab="Log(Time)",
```

```
    col="red",
    border="blue",
    horizontal=TRUE,
    las=2
  )
```





Conclusion

Summary for comparison of variance computation algorithms

- Variance computed with all algorithms is the same for dataset x1 with mean = 0
- Variance computed for dataset x2 with mean = 1000000 is the same for algorithms, but results produced by excel algorithm, which has differences starting from 4 decimal place

Summary for benchmarking performed for x1 dataset with mean = 0 and x2 dataset with mean = 1000000

- The ranking of the performed benchmarking is the same for both datasets
- Speed of variance computation for all algorithms is slightly higher for dataset x1 with mean = 0 than for x2
- According to the performed benchmarking, **excel** and **two-pass** algorithms have the highest execution speed.
- Gold-standard implementation of **var** function in R is on the 4th place, while **excel** and **two-pass** outperform it in terms of computation speed
- The least performant algorithm is **online** implementation, because of the iterative approach to updating the variance with each new element of vector

Task 3

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

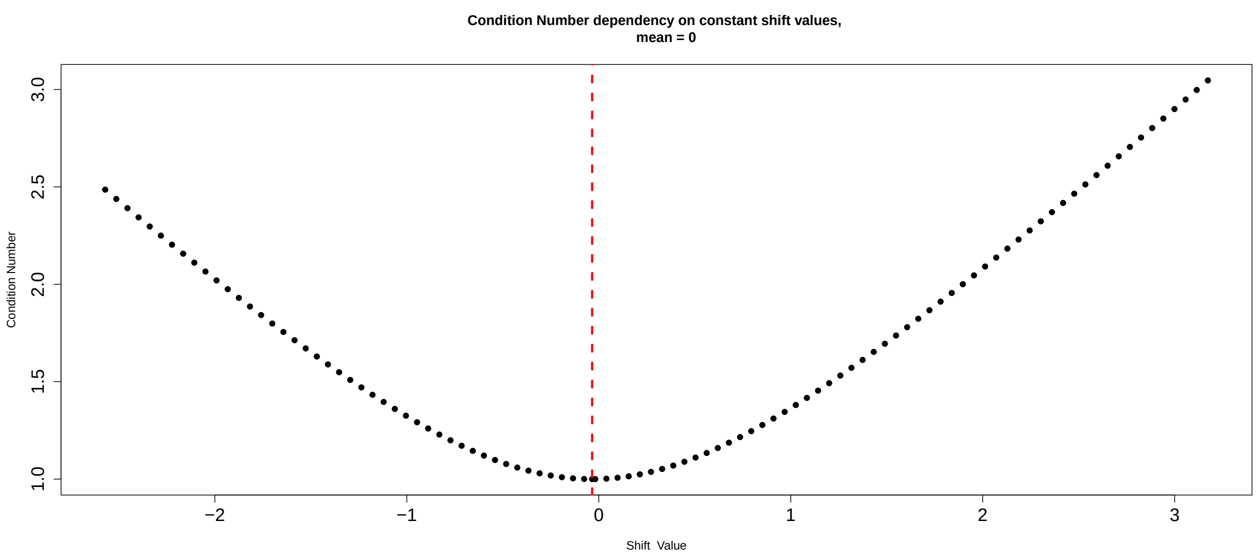
- Compare the results according to the instructions provided by Comparison I and Comparison II of the slides.
- Provide your results in table format and graphical format comparable to the example in the slides.

Description of the approach To investigate the property of scale invariance we can perform the following experiment: - Generate the sequence of values within the range between the minimum and maximum values of distributions - Iteratively compute the condition number for each obtained shift constant - Compare the results and identify, whether condition number is affected by the shift constant c

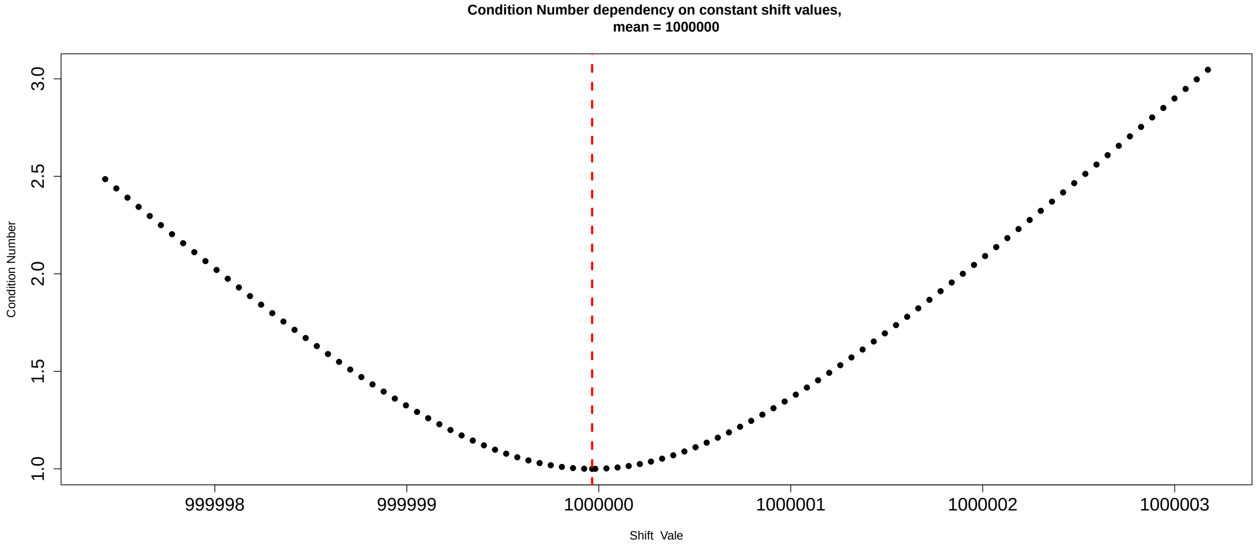
Shift implementation of variance algorithm

```r
func_cond_num <- function(vector, const){
  n = length(vector)
  vector_shifted = vector - const

  mean_shifted <- mean(vector_shifted)

  s_shifted = sum((vector_shifted - mean_shifted)^2)
  cond_number <- sqrt(1 + (mean_shifted^2 * n)/s_shifted)


  return(cond_number)
}


func_compare_cond_numbers <- function(vector){
  cond_numbs <- c()

  consts <- seq(from = min(vector), to = max(vector), length.out = 100)#((max(vector)-min(vector))/100)
  consts <- append(consts, mean(vector))
  consts <- sort(consts)
  for(const in consts) {
    cond_number <- func_cond_num(vector, const)
    cond_numbs <- append(cond_numbs, cond_number)
  }
  df <- data.frame(cond_numbs, consts)
  return(df)
}
```

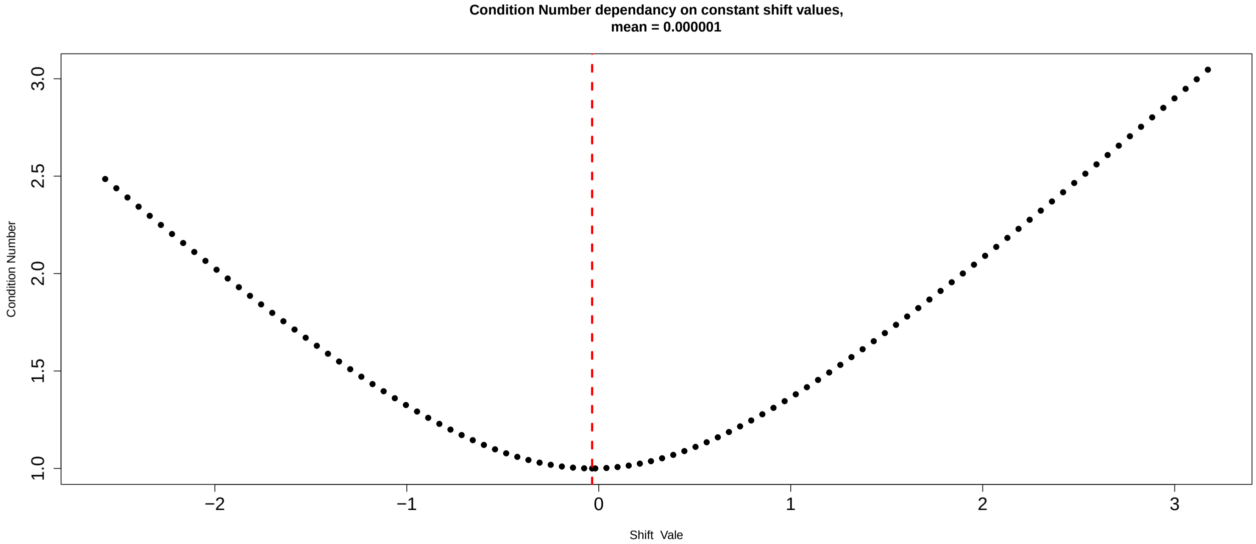
```r
knitr::opts_chunk$set(fig.width=20, fig.height=12)

df_cond_number_x1 <- func_compare_cond_numbers(x1)
par(mar=c(5, 12, 5, 5))
plot(df_cond_number_x1$consts, df_cond_number_x1$cond_numbs, main="Condition Number dependency on consta
     mean = 0",
     xlab="Shift  Value", ylab="Condition Number", pch=19, cex.axis = 1.5)
abline(v = mean(x1), col="red", lwd=3, lty=2)
```



```
df_cond_number_x2 <- func_compare_cond_numbers(x2)
par(mar=c(5, 12, 5, 5))
plot(df_cond_number_x2$consts, df_cond_number_x2$cond_numbs, main="Condition Number dependency on consta
     mean = 1000000",
     xlab="Shift  Vale", ylab="Condition Number", pch=19, cex.axis = 1.5)
abline(v = mean(x2), col="red", lwd=3, lty=2)
```




```
df_cond_number_x3 <- func_compare_cond_numbers(x3)
par(mar=c(5, 12, 5, 5))
plot(df_cond_number_x3$consts, df_cond_number_x3$cond_numbs, main="Condition Number dependancy on consta
     mean = 0.000001",
     xlab="Shift  Vale", ylab="Condition Number", pch=19, cex.axis = 1.5)
abline(v = mean(x3), col="red", lwd=3, lty=2)
```



Conclusion

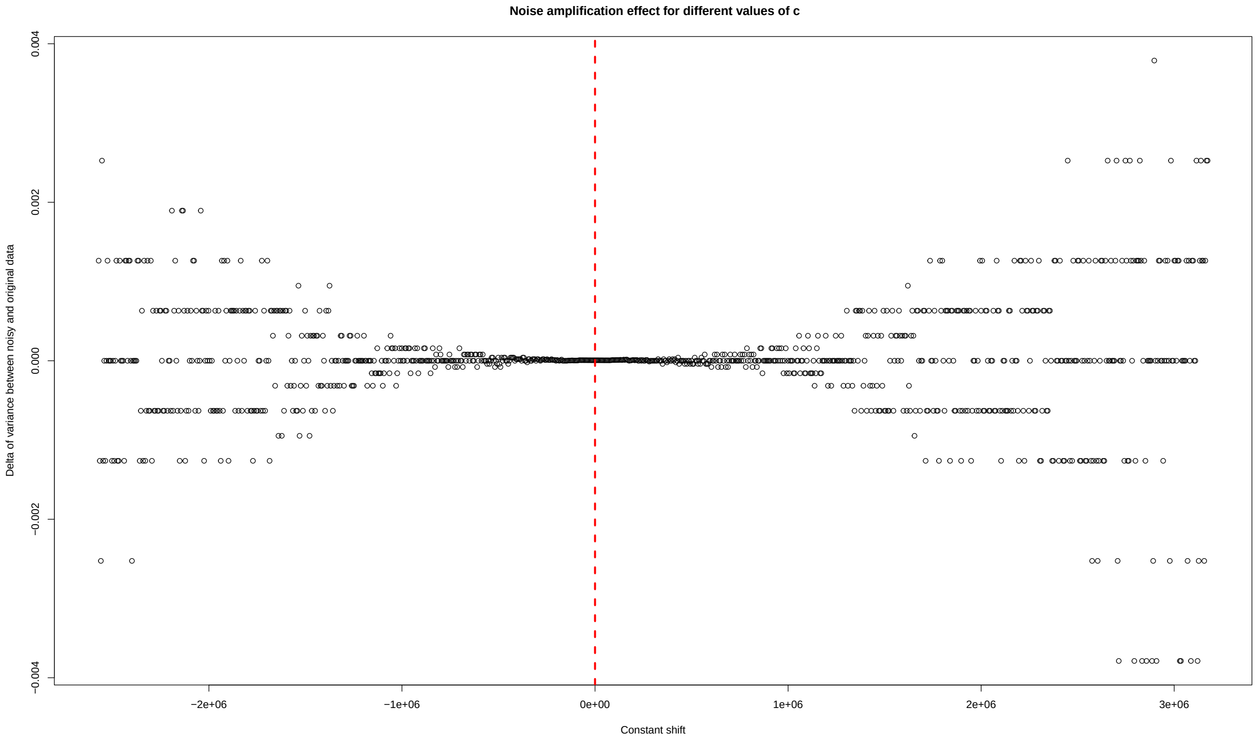
The vertical dashed line highlights the mean of the distribution. The lowest condition number is observed at the dotted line, which indicates that the optimal condition number k ~ 1 is obtained when the shift constant equals the mean of the distribution. It leads to the conclusion that the best condition number k = 1 is achieved for the constant shift of distribution by the mean of the distribution. Change of the shift constant c has the same effect on the condition number for all datasets (with mean = 0, mean = 1000000, mean = 0.000001)

Comparison of variance computation for different values of constant shift c

Description of the approach: In order to demonstrate the impact of constant c on the robustness of the variance computation, we can introduce noise into original dataset x1 with mean = 0. We can iterate over the range of shift constants c from min(vector)*1000000 to max(vector)*1000000 and compare the deltas between the variance, computed for original data and for noisy data.

```r
func_var_const <- function(vector, const){
  n = length(vector)
  vector_shifted = vector - const

  mean_shifted <- mean(vector_shifted)

  variance <- sum((vector_shifted - mean_shifted)^2)/(n-1)

  return(variance)
}

variance_shift_c <- function(vector, const){
  n = length(vector)
  p1 = sum((vector-const)^2)
  p2 = 1/n * (sum(vector-const))^2
  variance <- (p1 - p2)/(n-1)
  return(variance)
}


compute_noisy_variance <- function(vector){

  noise <- rnorm(100, mean = 0) * 0.001
  constant_shifts <- seq(from = min(vector)*1000000, to = max(vector)*1000000, length.out = 1000)

  variances <- c()

  vector_n <- vector + noise

  for(const in constant_shifts) {
    variance <-  variance_shift_c(vector_n, const) - variance_shift_c(vector, const)
    #variance_diff <- func_var_const(vector_n, const) -
    variances <- append(variances, variance)
  }

  df <- data.frame(variances, constant_shifts)
  return(df)
}

df_noisy <- compute_noisy_variance(x1)
```

```r
plot(df_noisy$constant_shifts, df_noisy$variances, xlab = "Constant shift", ylab = "Delta of variance be
abline(v = 0, col="red", lwd=3, lty=2)
```



Summary of shift constant experiment with added noise

Implemented procedure indicates, that variance deltas between original and noisy data has the biggest variation with shift constant c, far from distribution mean, on the contrary computation becomes more robust, when shift constant c approaches the distribution's mean. This illustrates, that shift constant c affects condition number, which in turn creates the effect of noise amplification.

Why shift by c = mean(data) has the best condition number k?

With k computed as follows: $\bar{k} = \sqrt{1 + \frac{n}{S}}\,(\bar{x} - c)$ It means that the most robust variance computation, when $k = 1$ can be obtained with the shift $c = \bar{x}$.

Task 4

Compare condition numbers for the 2 simulated data sets and a third one where the requirement is not fulfilled, as described during the lecture.

```
df_cond_number <- data.frame(rbind(func_cond_num(x1, 0), func_cond_num(x2, 0), func_cond_num(x3, 0)))
colnames(df_cond_number) <- c("Condition number")
rownames(df_cond_number) <- c("x1 (mean = 0)", "x2 (mean = 1000000)", "x3 (mean = 0.000001)")
kable(df_cond_number, caption = "Condition numbers for datasets x1, x2, x3")
```

Table 4: Condition numbers for datasets x1, x2, x3

	Condition number
x1 (mean = 0)	1.000485e+00
x2 (mean = 1000000)	8.971613e+05
x3 (mean = 0.000001)	1.000485e+00

Conclusion

- Condition numbers for datasets x1 and x3 are the same
- For dataset x2 condition number is very high, which means that variance is less robust to the errors. There is a practical sense to perform shift corrections by sample mean to reduce the condition number k, which will result in more robust variance computation.