Ex1

Yelyzaveta Klysa

2023

```
set.seed(12208877)
library(microbenchmark)
```

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

Algorithm 1

```
alg1 <- function(nums){
  m <- mean(nums)
  sum_n <- sum((nums - m)^2)
  return (sum_n / (length(nums) - 1))
}
```

Algorithm 2

```
alg2 <- function(nums){
  p1 <- sum(nums^2)
  p2 <- sum(nums)^2 / length(nums)
  return ((p1 - p2) / (length(nums) - 1))
}
```

Algorithm 3

```
alg3 <- function(nums, c=-1){
  if (c == -1) {
    c <- nums[1]
  }
  p1 <- sum((nums-c)^2)

  p2 <- sum(nums-c)^2 / length(nums)

  return ((p1 - p2) / (length(nums) - 1))
}
```

Algorithm 4

```
cal_mean <- function(xc, xmp, n){
  return (xmp + (xc - xmp) / n)
```

```
}
cal_s <- function(xc, xmp, sp, n){
  return (((n-2)/(n-1)) * sp + ((xc - xmp)^2/n))
}

alg4 <- function(nums){
  m <- mean(nums[1:2])
  s <- sum((nums[1:2] - m)^2)
  for (i in c(3:length(nums))){
    s <- cal_s(nums[i], m, s, i)
    m <- cal_mean(nums[i], m, i)
  }
  return (s)
}
```

Wrapper

```
wrapper <- function(nums){
  print(alg1(nums))
  print(alg2(nums))
  print(alg3(nums))
  print(alg4(nums))
  print(var(nums))
}
wrapper2 <- function(f, nums){
  return (f(nums))
}
```

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.

```
set.seed(12208877)
wrapper(rnorm(100, mean=1000000))
```

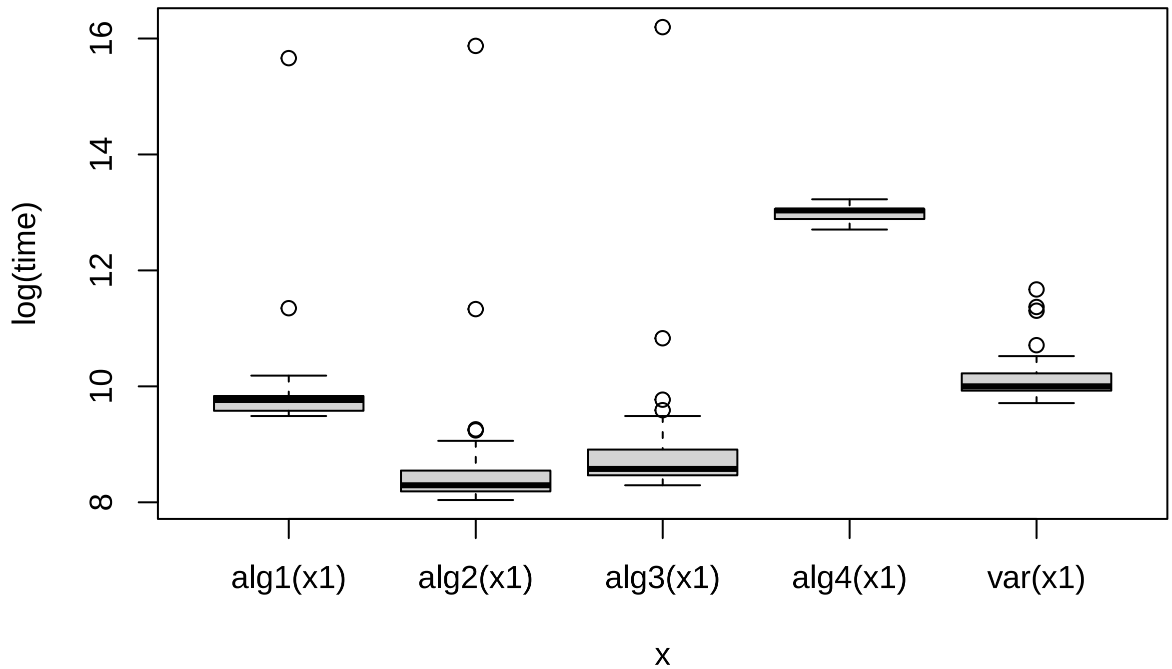
```
## [1] 1.067112
## [1] 1.067235
## [1] 1.067112
## [1] 1.067112
## [1] 1.067112
```

```
#set.seed(12208877)
x1 <- rnorm(100)
x2 <- rnorm(100, mean=1000000)
comparison <- microbenchmark(alg1(x1), alg2(x1), alg3(x1), alg4(x1), var(x1))
comparison
```

```
## Unit: microseconds
##      expr  min    lq    mean median    uq     max neval
##  alg1(x1) 13.2 14.45  80.842  17.40 18.65  6334.5   100
##  alg2(x1)  3.1  3.60  83.708   4.00  5.15  7822.9   100
##  alg3(x1)  4.0  4.75 114.735   5.30  7.40 10813.8   100
```

```
##  alg4(x1) 329.3 394.85 441.100 457.85 469.60   555.0   100
##   var(x1)  16.5  20.45  25.585  22.05  27.55   117.2   100
```

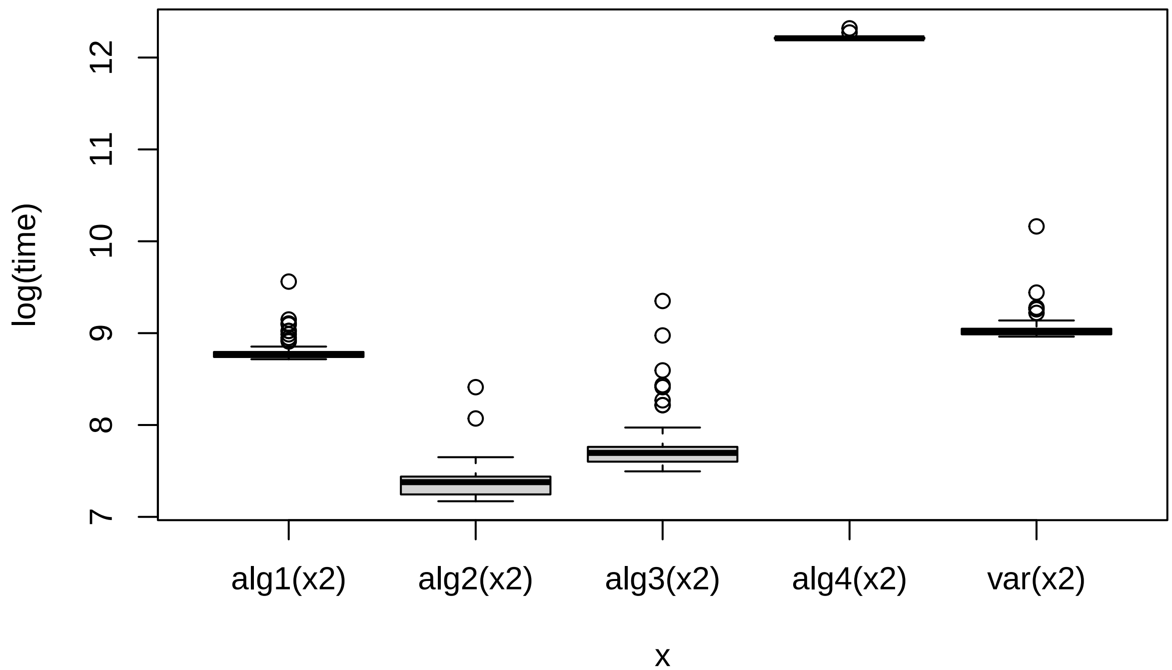
```r
plot(comparison$expr, log(comparison$time), ylab="log(time)")
```




```r
comparison2 <- microbenchmark(alg1(x2), alg2(x2), alg3(x2), alg4(x2), var(x2))
comparison2
```

```
## Unit: microseconds
##      expr   min     lq    mean median     uq   max neval
##  alg1(x2)   6.1   6.25   6.633   6.40   6.60  14.2   100
##  alg2(x2)   1.3   1.40   1.630   1.60   1.70   4.5   100
##  alg3(x2)   1.8   2.00   2.452   2.20   2.35  11.5   100
##  alg4(x2) 197.7 199.35 201.196 200.65 202.35 223.6   100
##   var(x2)   7.8   8.00   8.576   8.20   8.50  25.9   100
```

```r
plot(comparison2$expr, log(comparison2$time), ylab="log(time)")
```



As can be seen the second and third algorithms are the fastest ones, they are even faster than the basic `var`, while the 'online' algorithm (which is the fourth one) needs the longest period of time to calculate the output.

3. Scale invariance property.

```
algs <- c(alg1, alg2, alg3, alg4)

shift_x <- function(x, shift){
  return (x-shift)
}
compare <- function(x_or, shift){
  x <- shift_x(x_or, shift)
  for (i in 1:4) {
    if (i == 3) {
      cat("variance", i, "\n")
      cat("all.equal:", all.equal(alg3(x, shift), wrapper2(var, x_or)), "\n")
      cat("identical:", identical(alg3(x, shift), wrapper2(var, x_or)),  "\n")
      cat("==:", alg3(x, shift) == wrapper2(var, x_or),  "\n\n")
    }
    else {
      cat("variance", i, "\n")
      cat("all.equal:", all.equal(wrapper2(algs[[i]], x), wrapper2(var, x_or)), "\n")
      cat("identical:", identical(wrapper2(algs[[i]], x), wrapper2(var, x_or)),  "\n")
      cat("==:", wrapper2(algs[[i]], x) == wrapper2(var, x_or),  "\n\n")
    }
```

```r
  }
}
compare(x1, 0)
```

```
## variance 1
## all.equal: TRUE
## identical: TRUE
## ==: TRUE
##
## variance 2
## all.equal: TRUE
## identical: TRUE
## ==: TRUE
##
## variance 3
## all.equal: TRUE
## identical: TRUE
## ==: TRUE
##
## variance 4
## all.equal: TRUE
## identical: FALSE
## ==: FALSE
```

```r
compare(x2, 0)
```

```
## variance 1
## all.equal: TRUE
## identical: TRUE
## ==: TRUE
##
## variance 2
## all.equal: Mean relative difference: 0.0001607834
## identical: FALSE
## ==: FALSE
##
## variance 3
## all.equal: Mean relative difference: 0.0001607834
## identical: FALSE
## ==: FALSE
##
## variance 4
## all.equal: TRUE
## identical: FALSE
## ==: FALSE
```

In the setup above we compared the algorithms with `var` without shift. The first and third algorithms seem to give the same results. Now let's test them with a shift.

x1: shift 0.2

```r
compare(x1, 0.2)
```

```
## variance 1
## all.equal: TRUE
## identical: TRUE
```

```
## ==: TRUE
##
## variance 2
## all.equal: TRUE
## identical: FALSE
## ==: FALSE
##
## variance 3
## all.equal: TRUE
## identical: TRUE
## ==: TRUE
##
## variance 4
## all.equal: TRUE
## identical: FALSE
## ==: FALSE
```

x1: shift 2

```r
compare(x1, 2)
```

```
## variance 1
## all.equal: TRUE
## identical: FALSE
## ==: FALSE
##
## variance 2
## all.equal: TRUE
## identical: FALSE
## ==: FALSE
##
## variance 3
## all.equal: TRUE
## identical: FALSE
## ==: FALSE
##
## variance 4
## all.equal: TRUE
## identical: FALSE
## ==: FALSE
```

x1: shift 20

```r
compare(x1, 20)
```

```
## variance 1
## all.equal: TRUE
## identical: TRUE
## ==: TRUE
##
## variance 2
## all.equal: TRUE
## identical: FALSE
## ==: FALSE
##
## variance 3
```

```
## all.equal: TRUE
## identical: FALSE
## ==: FALSE
##
## variance 4
## all.equal: TRUE
## identical: FALSE
## ==: FALSE
```

x2: shift 2

```r
compare(x2, 2)
```

```
## variance 1
## all.equal: TRUE
## identical: TRUE
## ==: TRUE
##
## variance 2
## all.equal: Mean relative difference: 3.870836e-05
## identical: FALSE
## ==: FALSE
##
## variance 3
## all.equal: Mean relative difference: 3.870836e-05
## identical: FALSE
## ==: FALSE
##
## variance 4
## all.equal: TRUE
## identical: FALSE
## ==: FALSE
```

x2: shift 100000

```r
compare(x2, 100000)
```

```
## variance 1
## all.equal: TRUE
## identical: TRUE
## ==: TRUE
##
## variance 2
## all.equal: Mean relative difference: 3.870836e-05
## identical: FALSE
## ==: FALSE
##
## variance 3
## all.equal: Mean relative difference: 3.870836e-05
## identical: FALSE
## ==: FALSE
##
## variance 4
## all.equal: TRUE
## identical: FALSE
## ==: FALSE
```

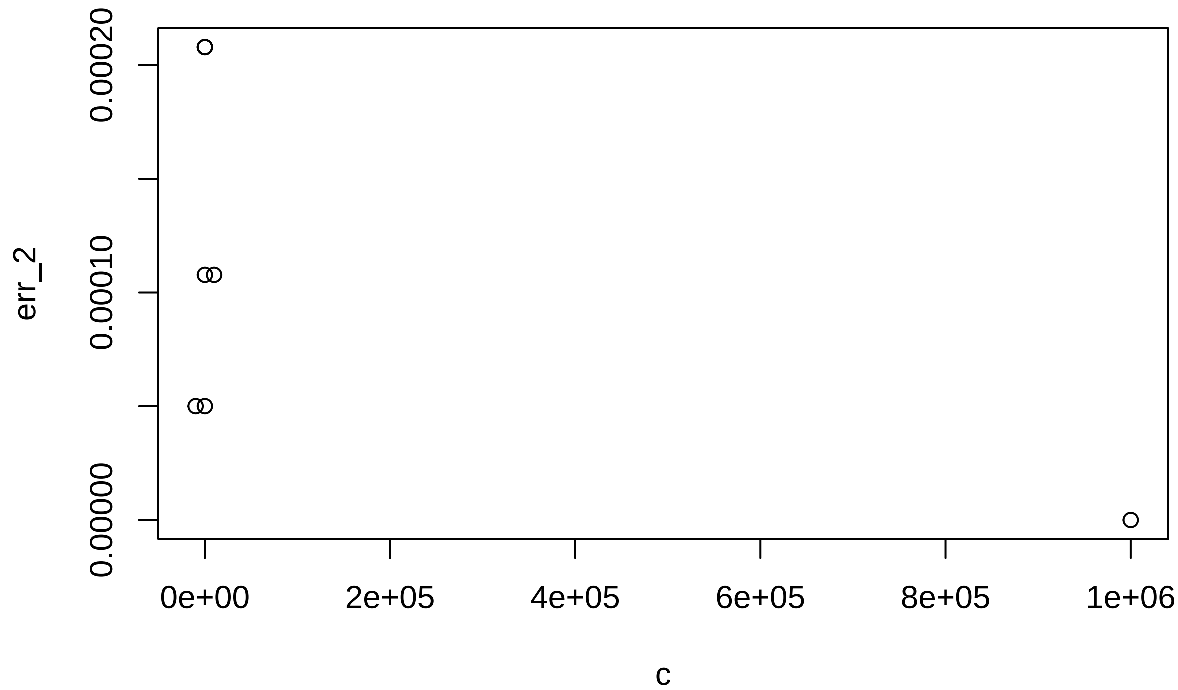
x2: shift 1000000

```r
compare(x2, 1000000)
```

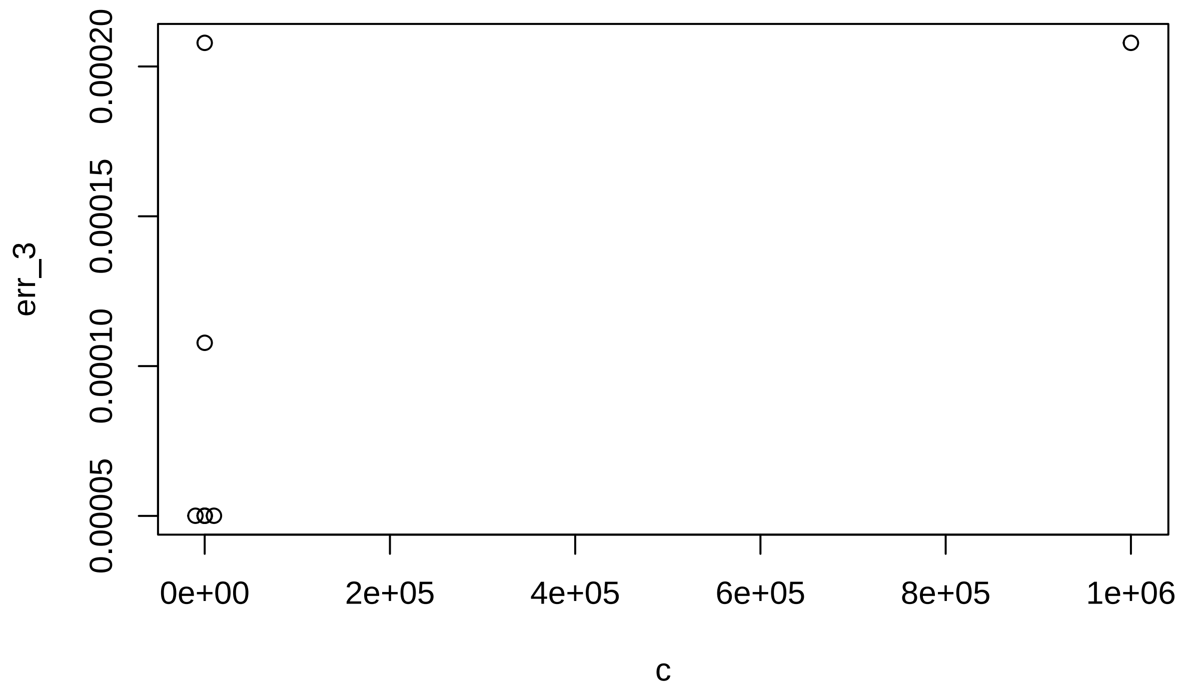
```
## variance 1
## all.equal: TRUE
## identical: TRUE
## ==: TRUE
##
## variance 2
## all.equal: TRUE
## identical: TRUE
## ==: TRUE
##
## variance 3
## all.equal: Mean relative difference: 0.0001607834
## identical: FALSE
## ==: FALSE
##
## variance 4
## all.equal: TRUE
## identical: TRUE
## ==: TRUE
```

We can see from the experiments that the small shift did not affect the results from the algorithm 3 and the algorithm 1. The bigger shift however made bigger difference as can be observed in the results for the first dataset. Using mean value of the dataset gave the best equality result for the algorithm 2.

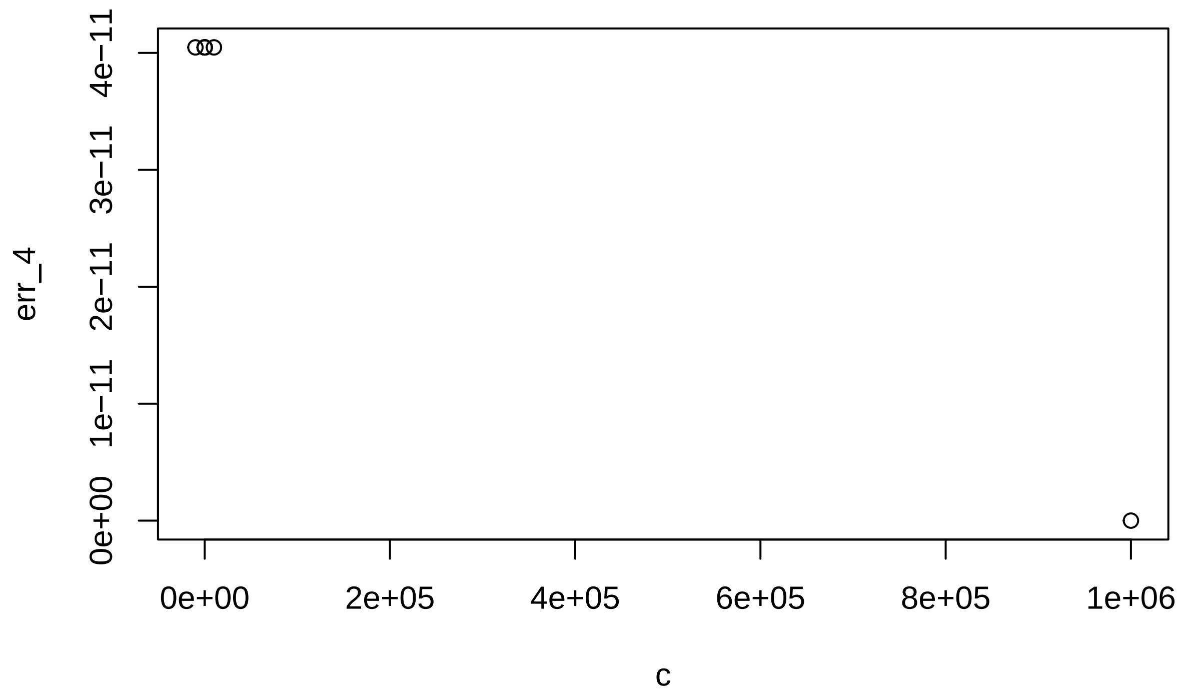
```r
list_c <- c(-10000, -10, -2, 0, 2, 10, 10000, 1000000)
res3 <- c()
res2 <- c()
res1 <- c()
res4 <- c()
for (i in 1:length(list_c)){
  x_new <- shift_x(x2, list_c[i])
  res1[i] <- alg1(x_new)
  res2[i] <- alg2(x_new)
  res3[i] <- alg3(x_new, list_c[i])
  res4[i] <- alg4(x_new)
}
err_3 <-  abs(c(res3) - var(x2))
err_2 <- abs(c(res2) - var(x2))
err_1 <- abs(c(res1) - var(x2))
err_4 <-  abs(c(res4) - var(x2))
plot(list_c,err_2, xlab = "c")
```




```
plot(list_c,err_3, xlab = "c")
```




```
plot(list_c,err_4, xlab = "c")
```




```r
df <- data.frame(list_c, err_1, res1, err_2, res2, err_3, res3, err_4, res4)
df
```

```
##   list_c err_1     res1        err_2     res2        err_3     res3
## 1 -1e+04     0 1.293137 5.005328e-05 1.293087 5.005328e-05 1.293087
## 2 -1e+01     0 1.293137 2.078816e-04 1.292929 5.005328e-05 1.293087
## 3 -2e+00     0 1.293137 2.078816e-04 1.292929 5.005328e-05 1.293087
## 4  0e+00     0 1.293137 2.078816e-04 1.292929 2.078816e-04 1.292929
## 5  2e+00     0 1.293137 5.005328e-05 1.293087 5.005328e-05 1.293087
## 6  1e+01     0 1.293137 1.077750e-04 1.293245 1.077750e-04 1.293245
## 7  1e+04     0 1.293137 1.077750e-04 1.293245 5.005328e-05 1.293087
## 8  1e+06     0 1.293137 0.000000e+00 1.293137 2.078816e-04 1.292929
##          err_4     res4
## 1 4.047207e-11 1.293137
## 2 4.047207e-11 1.293137
## 3 4.047207e-11 1.293137
## 4 4.047207e-11 1.293137
## 5 4.047207e-11 1.293137
## 6 4.047207e-11 1.293137
## 7 4.047207e-11 1.293137
## 8 0.000000e+00 1.293137
```

In the plot above we showed: plot 1 - the difference between the output of algorithms 2 applied to shifted `x2` and `var` plot 2 - the difference between the output of algorithm 3 applied to shifted `x2` comparing to `var` We can see that for the second and the fourth algorithms the best choice of `c` was mean as we claimed previously. Though for the third algorithm the smallest values worked the best. The shift did not affect the results of the first algorithm.

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.

```
calc_cond_number <- function(nums, c=0){
  m <- mean(nums)
  s <- sum((nums-m)^2)
  return (sqrt(1 + (((m-c)^2)*length(nums))/s))
}

calc_cond_number(x1)
```

```
## [1] 1.012698
```

```
calc_cond_number(x1, c=3)
```

```
## [1] 3.230885
```

```
calc_cond_number(x2)
```

```
## [1] 883812.4
```

```
calc_cond_number(x2, c=3)
```

```
## [1] 883809.8
```

```
calc_cond_number(x2, c=1000000)
```

```
## [1] 1.000007
```

```
calc_cond_number(rnorm(1000000, mean = 0.0000000000000000000000001))
```

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

```
calc_cond_number(rnorm(1000000, mean = 0.0000000000000000000000001), c=0.0000000000000000000000001)
```

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

```
calc_cond_number(rnorm(1000000, mean = 0.0000000000000000000000001), c=1)
```

```
## [1] 1.412648
```

The above experiments again confirm the fact that the best shift is mean, since based on the formula `mean-c` will be cancelled if `c=mean`, then `sqrt(1) = 1`, so the condition number will be equal to 1. Additionally, we confirmed that by setting mean to a very small number can give us condition number of `1`, which fulfills the rule that it should be greater or equal to 1.