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)
x2 <- rnorm(100, mean=1000000)
```

```
library("knitr")
variance_x1 <- variance_wrapper(x1)
variance_x2 <- variance_wrapper(x2)
df_comp <- data.frame(rbind(variance_x1, variance_x2))
names(df_comp) <- c("two-pass", "excel", "shift", "online")
kable(df_comp)
```

	two-pass	excel	shift	online
variance_x1	1.2549423	1.2549423	1.2549423	1.2549423

	two-pass	excel	shift	online
variance_x2	0.7043781	0.7045455	0.7043781	0.7043781

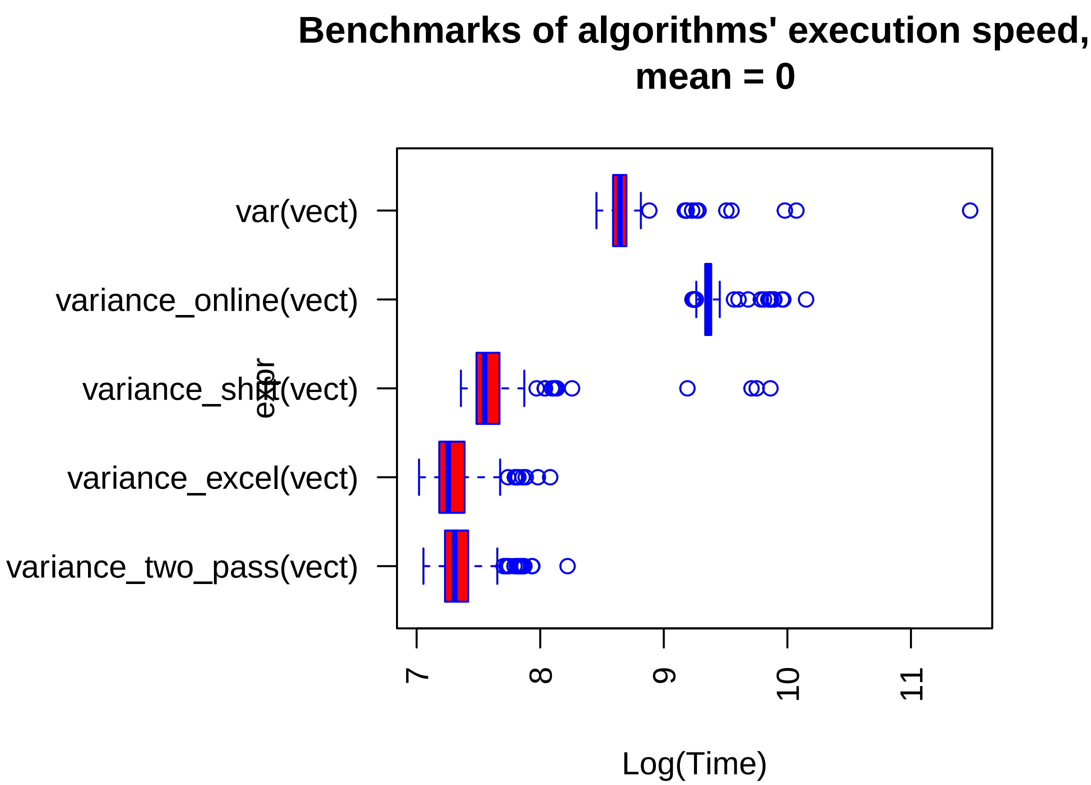
Function for benchmark computation

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

Summary table for generated data with mean = 0

expr	min	lq	mean	median	uq	max	neval
variance_two_pass(vect)	1.158	1.3800	1.63392	1.4895	1.6635	3.719	100
variance_excel(vect)	1.118	1.3170	1.54847	1.4145	1.6150	3.231	100
variance_shift(vect)	1.569	1.7805	2.57303	1.8980	2.1420	19.200	100
variance_online(vect)	10.219	11.3295	12.50750	11.6400	11.8905	25.621	100
var(vect)	4.698	5.3765	7.27003	5.6720	5.9845	96.676	100

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



Summary for data with mean = 0

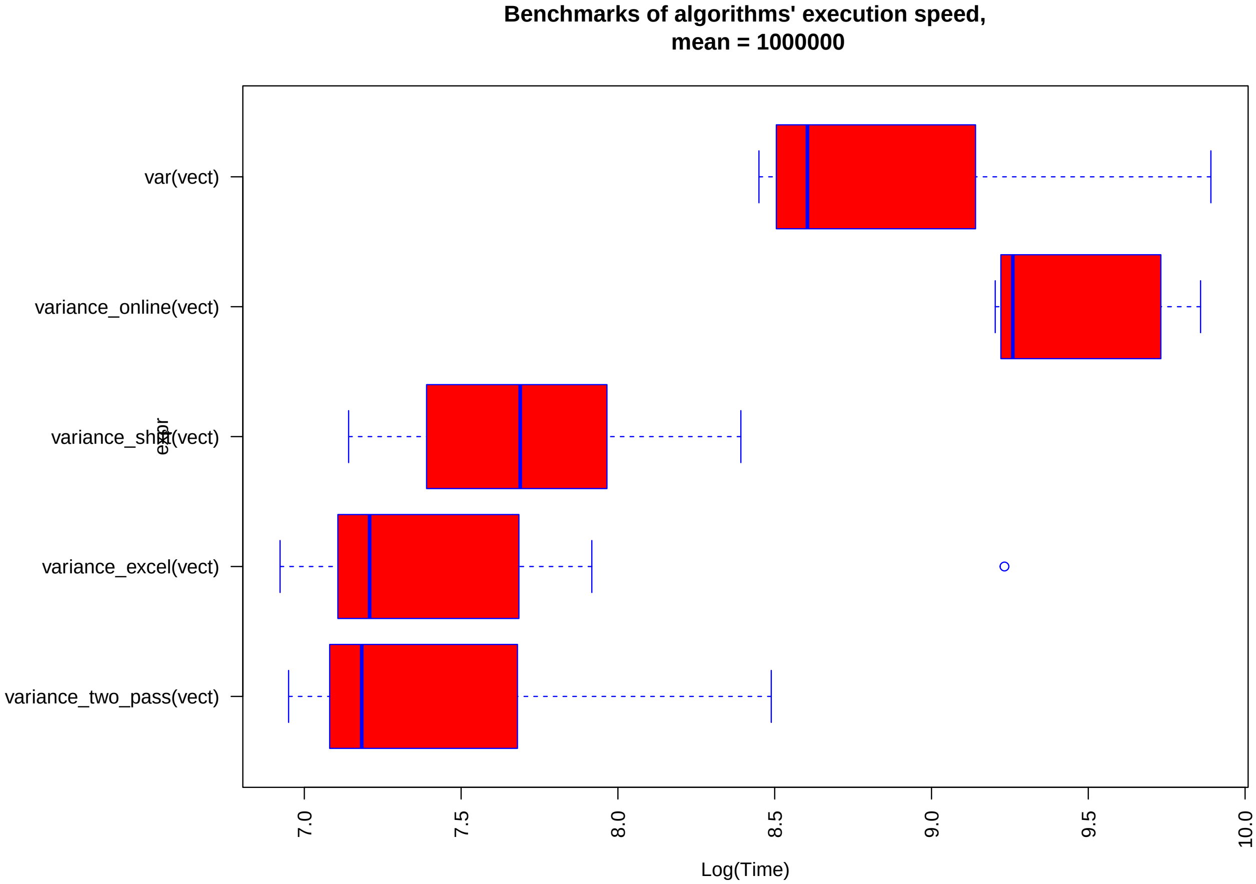
- According to the performed benchmarking, **excel** and **two-pass** algorithms have the highest execution speed.
- Gold-standard implementation of **var** function in R is on the 4th place
- The least performant algorithm is **online** implementation
- There are some outliers, which yield higher execution time

Summary table for generated data with mean = 1000000

expr	min	lq	mean	median	uq	max	neval
variance_two_pass(vect)	1.043	1.1890	1.62202	1.3170	2.1640	4.861	100
variance_excel(vect)	1.015	1.2205	1.70158	1.3505	2.1740	10.226	100
variance_shift(vect)	1.263	1.6195	2.31059	2.1830	2.8785	4.412	100
variance_online(vect)	9.929	10.1095	12.62097	10.5030	16.8385	19.112	100
var(vect)	4.674	4.9405	7.01245	5.4560	9.3245	19.751	100

```
knitr::opts_chunk$set(fig.width=12, fig.height=8)
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
)
```



Summary for data with mean = 1000000

- 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
- The least performant algorithm is **online** implementation

Conclusion

For generated dataset x1 with mean = 0 and for dataset x2 with mean = 10^6 based on the comparison results the speed of the algorithms execution is relatively the same. Which leads to the conclusion that the shift of the mean of the distribution's mean doesn't affect significantly the speed of variance computation.

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.

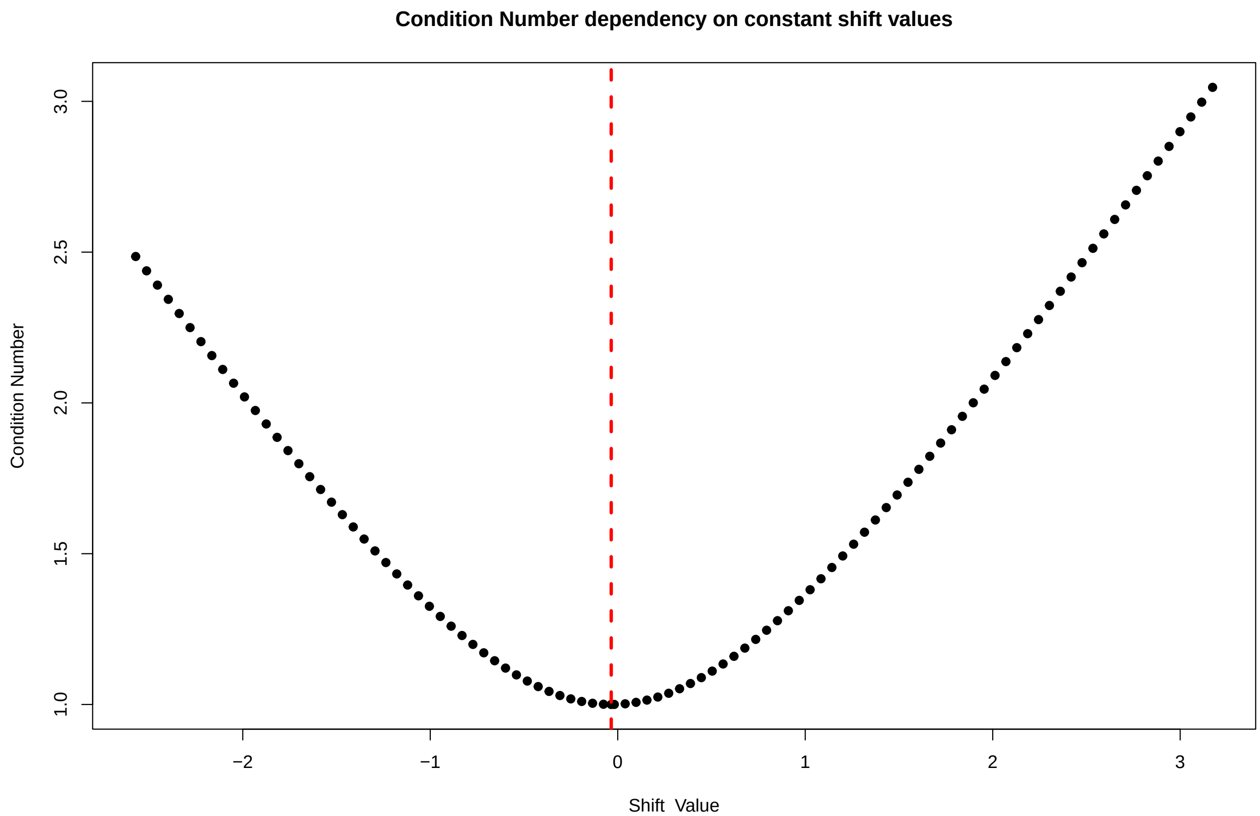
Description of the approach To investigate the property of scale invariance we can perform the following experiment:

- 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

Shift implementation of variance algorithm

```
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)
}
```

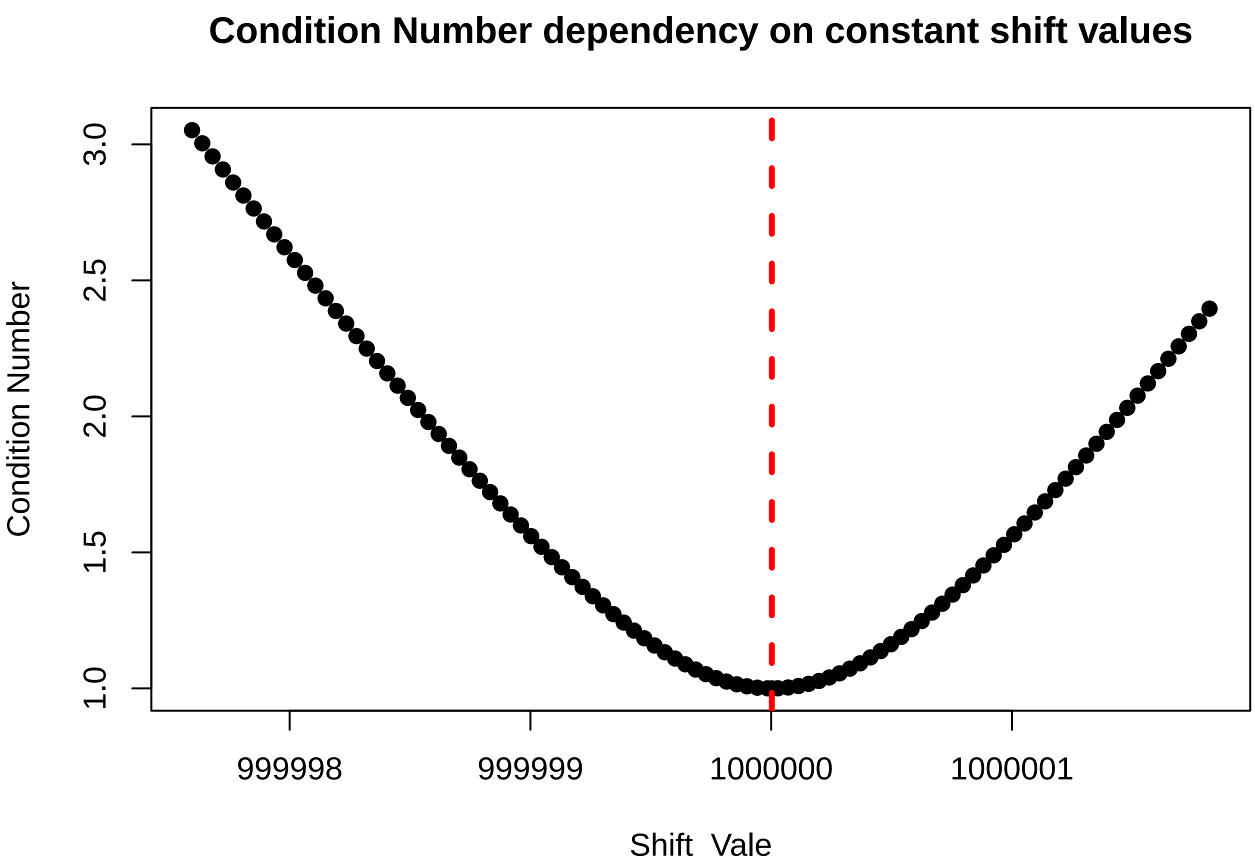
```
knitr::opts_chunk$set(fig.width=7, fig.height=5)
df_cond_number_x1 <- func_compare_cond_numbers(x1)
plot(df_cond_number_x1$consts, df_cond_number_x1$cond_numbs, main="Condition Number dependency on consta
     xlab="Shift  Value", ylab="Condition Number", pch=19)
abline(v = mean(x1), col="red", lwd=3, lty=2)
```



Condition numbers for generated data with mean = 0

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.

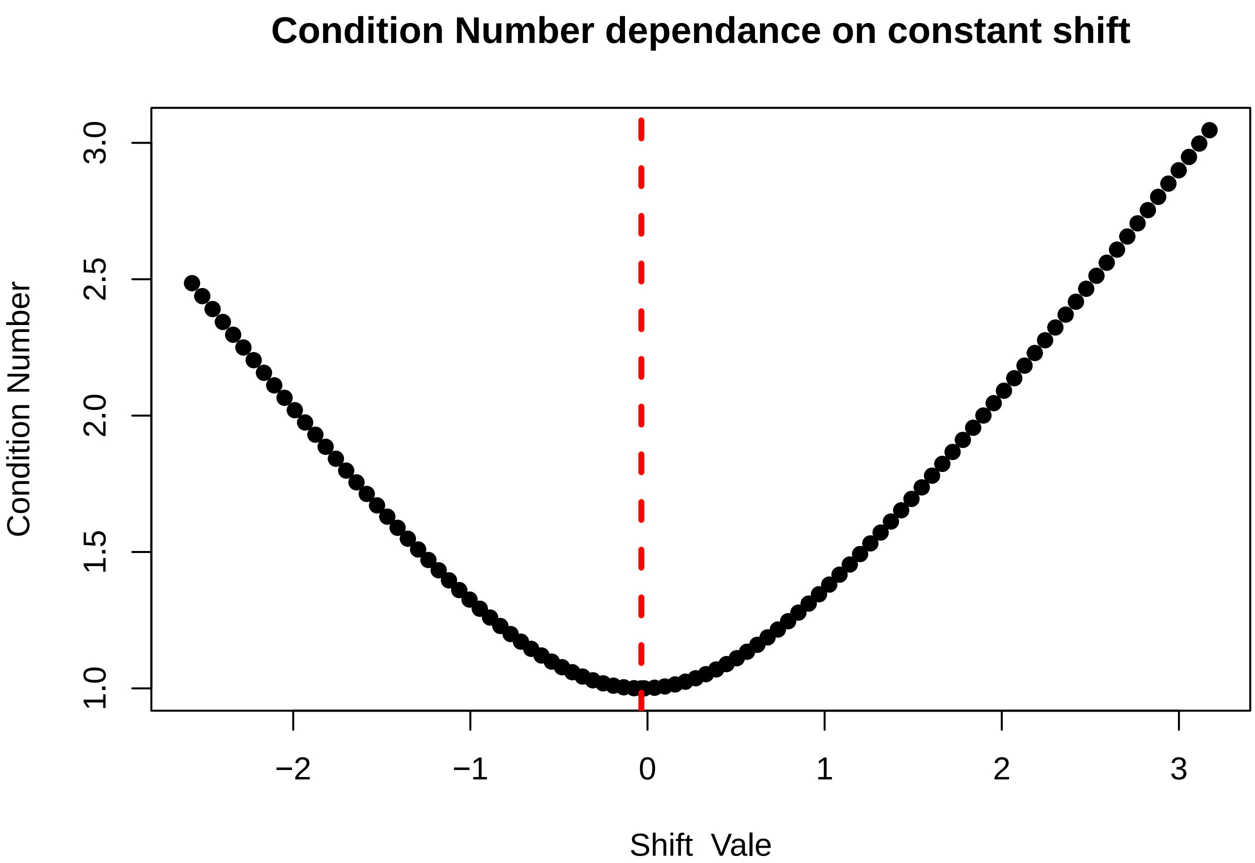
```
knitr::opts_chunk$set(fig.width=7, fig.height=5)
df_cond_number_x2 <- func_compare_cond_numbers(x2)
plot(df_cond_number_x2$consts, df_cond_number_x2$cond_numbs, main="Condition Number dependency on consta
     xlab="Shift  Vale", ylab="Condition Number", pch=19)
abline(v = mean(x2), col="red", lwd=3, lty=2)
```



Condition numbers for generated data with mean = 1000000

As in the previous example the vertical dashed line highlights the mean of the distribution. Respectively 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.

```
knitr::opts_chunk$set(fig.width=7, fig.height=5)
set.seed(12202120)
x3 <- rnorm(100, mean=0.000001)
df_cond_number_x3 <- func_compare_cond_numbers(x3)
plot(df_cond_number_x3$consts, df_cond_number_x3$cond_numbs, main="Condition Number dependance on consta
     xlab="Shift  Vale", ylab="Condition Number", pch=19)
abline(v = mean(x3), col="red", lwd=3, lty=2)
```



Condition numbers for generated data with mean = 0.000001

In the third scenario with the mean = 0.000001 the condition number takes value close to 1 for the shift performed following the same approach.

Conclusion

The best condition number k = 1 is achieved for the constant shift of distribution by the mean of the distribution.

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

The best k can be achieved when mean(data) - c = 0, this can be achieved with c = mean(data)

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.

Condition number of dataset x1 with mean = 0 and shift constant c = 0

```
func_cond_num(x1, 0)
```

```
## [1] 1.000485
```

Condition number of dataset x2 with mean = 1000000 and shift constant c = 0

```r
func_cond_num(x2, 0)
```

```
## [1] 1197511
```

Condition number of dataset x3 with mean = 0.000001 and shift constant c = 0

```r
func_cond_num(x3, 0)
```

```
## [1] 1.000485
```

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 and it leads to higher error. There is a practical sense to perform shift corrections by sample mean to reduce the condition number k