Exercise 1

Yannik Gaebel

2022-10-18

```
library(microbenchmark)

set.seed(12208157)

# create datasets
x1 <- rnorm(100)
x2 <- rnorm(100,mean=1000000)
```

Task 1

In the first exercise, the four algorithms from the lecture on calculating variance are implemented. They are then compared together with r's method of calculating variance.

Algorithm 1 (two - pass algorithm)

```
# function to calculate sample mean
sample_mean <- function(x){
  sum <- 0
  for (i in x){
    sum <- sum + i
  }
  return (sum/length(x))
}

# function to implement algorithm 1
variance <- function(x){
  sample_mean_x <- sample_mean(x)
  sum <- 0
  for (i in x){
    sum <- sum + (i - sample_mean_x)^2
  }
  var <- (1/(length(x1)-1))*sum
  return (var)
}
```

Algorithm 2 (one - pass algorithm)

```
variance_excel <- function(x){
  n <- length(x)
  p1 <- 0
  p2 <- 0

  for (i in x){
    p1 <- p1 + i^2
    p2 <- p2 + i
  }
  p2 <- (p2^2)/n

  var <- (p1-p2)/(n-1)
  return (var)
}
```

Algorithm 3 (shifted one - pass algorithm)

```
variance_shifted <- function(x, c=x[1]){
  p1 <- 0
  p2 <- 0
  len <- length(x)

  for (i in x){
    p1 <- p1 + (i-c)^2
    p2 <- p2 + (i-c)
  }
  p2 <- (p2^2)/len

  var <- (p1-p2)/(len-1)
  return (var)
}
```

Algorithm 4 (online algorithm)

```
variance_batch <- function(x){
  mean <- x[1]
  var <- 0

  for (n in 2:length(x)){
    var <- ((n-2)/(n-1))*var + ((x[n]-mean)^2)/n
    mean <- mean+(x[n]-mean)/n
  }
  return (var)
}
```

Wrapper function

```
variance_wrapper <- function(x){
  return(c(variance(x),variance_excel(x),variance_shifted(x),variance_batch(x),var(x)))
}
```

Task 2

Comparison of variance computations for dataset x1 and x2

series	two - pass	one - pass	shifted one - pass	online	r variance
x1 (mean=0)	0.9328839	0.9328839	0.9328839	0.9328839	0.9328839
x2 (mean=1000000)	1.422059	1.420928	1.422059	1.422059	1.422059

Observations: For the dataset x1 the results of each method are identical. For the dataset x2 the results of each method are identical as well, with the exception of the "one - pass algorithm" wich returns a slightly smaller value.

Comparison of conputation time

In following codechunk with the help of the microbenchmark function for both datasets the variance is calculated 100 times with each of the 4 algotithms from the lecture and r's built in funtion for variance. The method microbenchmark returns statistics on the computation time of each method.

```
m_x1 <- microbenchmark(variance(x1),variance_excel(x1),variance_shifted(x1),variance_batch(x1),var(x1),
m_x2 <- microbenchmark(variance(x2),variance_excel(x2),variance_shifted(x2),variance_batch(x2),var(x2),
```

Table results of microbenchmark method after performing each method 100 times for series x1

```
summary(m_x1)
```

```
##                   expr  min   lq    mean median    uq     max neval
## 1         variance(x1)  7.7  8.1 157.188   8.40 13.05 14629.1   100
## 2   variance_excel(x1)  5.3  5.6 250.770   5.85  9.10 24329.8   100
## 3 variance_shifted(x1)  8.2  8.6 124.388   9.10 14.65 11262.8   100
## 4   variance_batch(x1) 14.3 14.7 257.958  15.10 21.60 23955.1   100
## 5              var(x1)  8.5 12.2  18.016  15.50 19.70   128.7   100
```

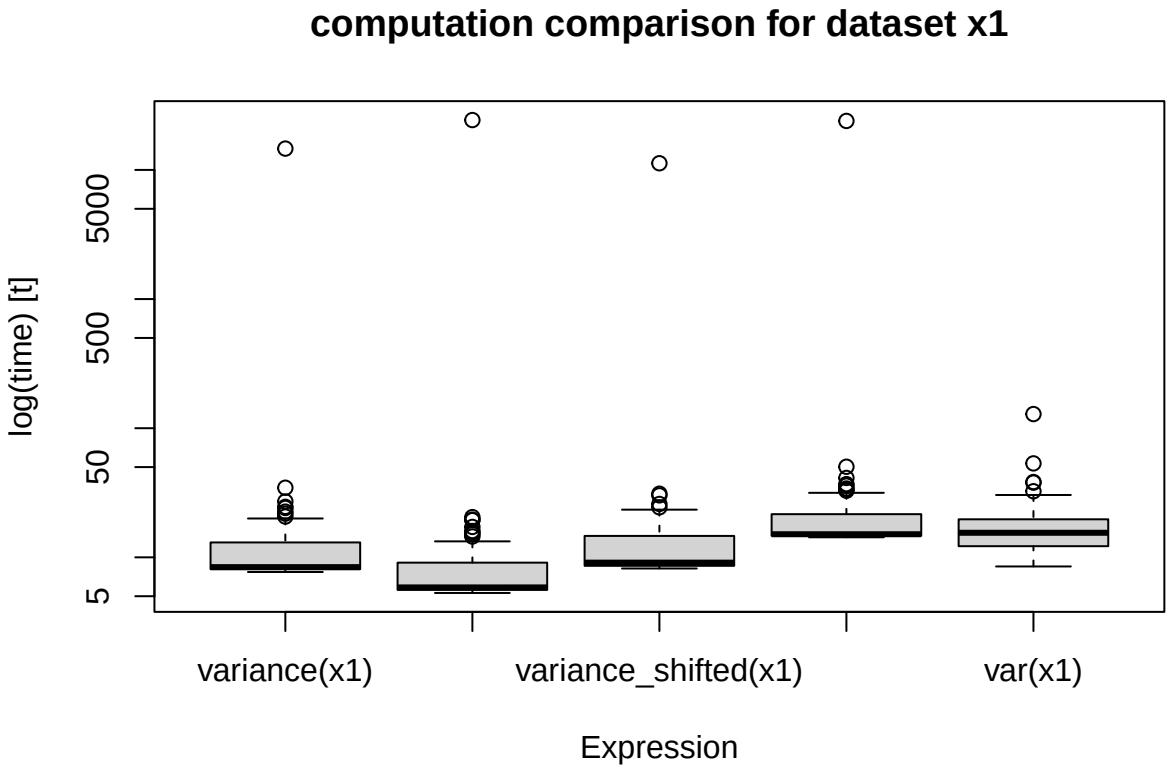
Table results of microbenchmark method after performing each method 100 times for series x2

```
summary(m_x2)
```

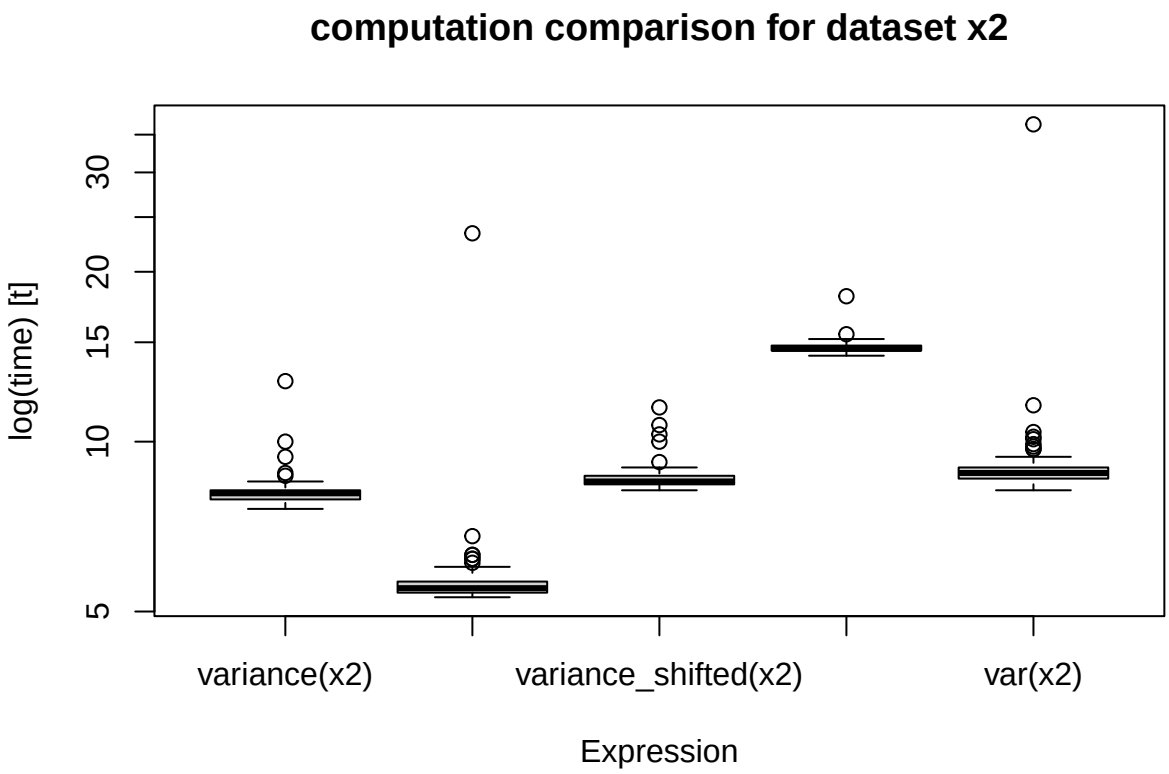
```
##                   expr  min   lq   mean median    uq  max neval
## 1         variance(x2)  7.6  7.9  8.135    8.1  8.20 12.8   100
## 2   variance_excel(x2)  5.3  5.4  5.758    5.5  5.65 23.4   100
## 3 variance_shifted(x2)  8.2  8.4  8.604    8.5  8.70 11.5   100
## 4   variance_batch(x2) 14.2 14.5 14.676   14.6 14.80 18.1   100
## 5              var(x2)  8.2  8.6  9.134    8.8  9.00 36.5   100
```

Visualisation of computation times using boxplots

```
boxplot(m_x1, main="computation comparison for dataset x1")
```




```
boxplot(m_x2, main="computation comparison for dataset x2")
```



Results:

The excel (one - pass algorithm) is the fastest and the batch (online algorithm) is the slowest for both datasets. The order from fastest to slowest does not change for the two datasets. There is also only a small difference in the absolute computation time for the two datasets, so the influce of the mean on computation time is small.

Would you know another way in R to compare computing times?

The duration of a computation can also be measured using the Sys.time() method before and after the computation and calculating the difference.

```
start <- Sys.time()
# perform operations
end <- Sys.time()

duration <- end - start
```

Task 3

The condition number measures how much the output of a function can change as a result of a small change of the input. It provides information about the sensitivity of the function to noise or errors.

Computation of the condition number

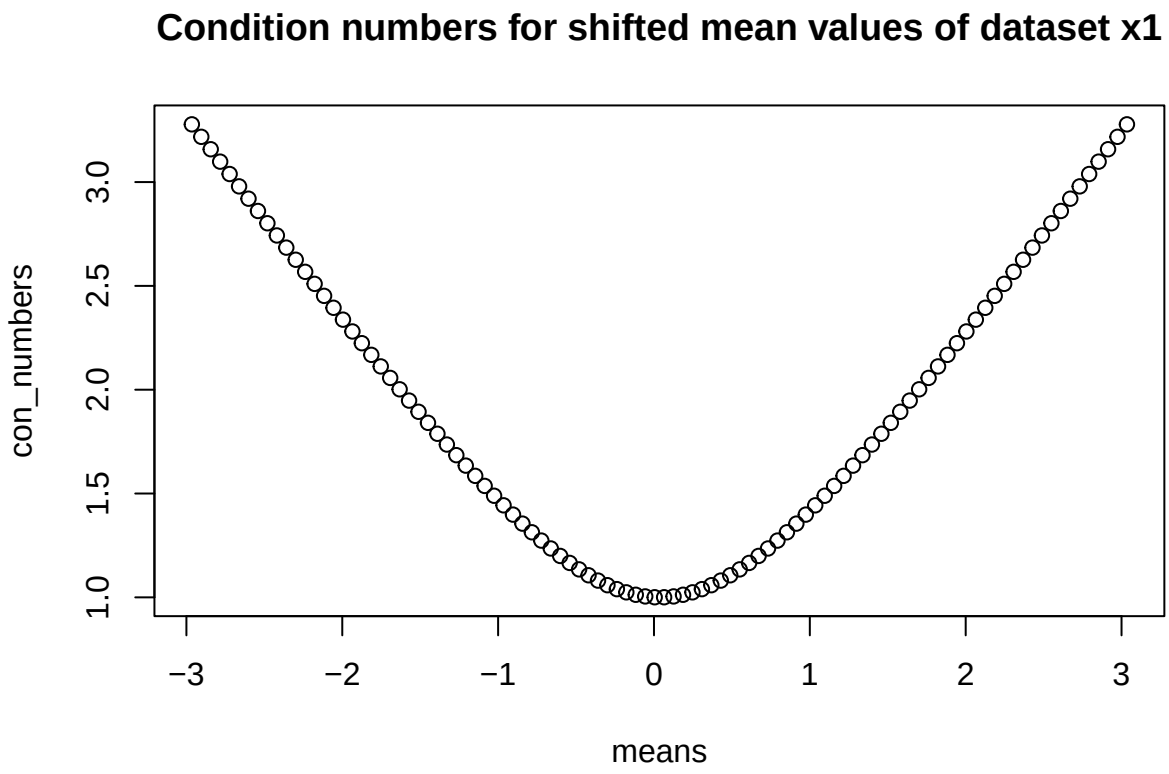
```
condition_number <- function(n,mean,S){
  condition_num <- sqrt(1+(((mean^2) * n)/S))
}
```

Functions that calculates a sequence of values by which to shift the dataset and the corresponding condition numbers

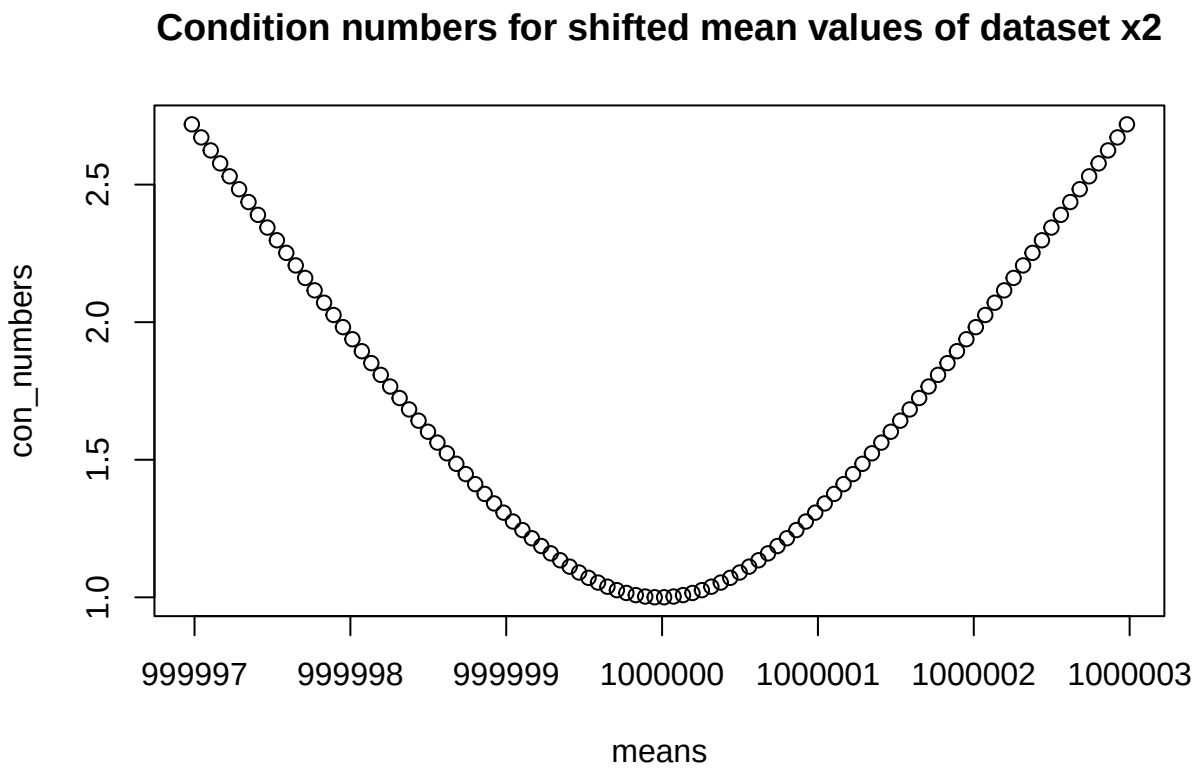
```
compute_condition_nums <- function(x){
  mean <- sample_mean(x)
  n <- length(x)
  means <- list()
  condition_numbers <- list()
  shift_values = seq(from=mean-3,to=mean+3,length.out=100) # generate list of shift values
  for (i in 1:length(shift_values)){
    mean_shifted <- mean - shift_values[i]
    means[i] <- shift_values[i]
    x_shifted <- x - shift_values[i]
    S <- sum((x_shifted - mean_shifted)^2)
    con_number = condition_number(n,mean_shifted,S) # compute condition numbers
    condition_numbers[i] <- con_number
  }
  result <- c(means,condition_numbers)
  return (result) # return concatenated result
}
```

The results of this functions are now plotted.

```
result <- compute_condition_nums(x1)
means <- result[1:100]
con_numbers <- result[101:200]
plot(means,con_numbers,main="Condition numbers for shifted mean values of dataset x1")
```




```
result <- compute_condition_nums(x2)
means <- result[1:100]
con_numbers <- result[101:200]
plot(means,con_numbers,main="Condition numbers for shifted mean values of dataset x2")
```



In both visualisations we can see similar curves. The condition number is best (lowest) at the mean of the dataset for both datasets.

Derived from these results it would be expected to have a lower stability for datasets that are shifted by a large value.

Task 4

Comparison of condition number for different data sets

```
x1 <- rnorm(100)
x2 <- rnorm(100,mean=1000000)
x3 <- rnorm(100,mean=1/1000000,sd=0.000001) # dataset where the requirement is not fulfilled
```

Compute the condition number for each dataset:

```
# compute condition numbers
mean_x1 <- mean(x1)
mean_x2 <- mean(x2)
mean_x3 <- mean(x3)
con_number_x1 = condition_number(100,mean_x1,sum((x1 - mean_x1)^2))
con_number_x2 = condition_number(100,mean_x2,sum((x2 - mean_x2)^2))
con_number_x3 = condition_number(100,mean_x3,sum((x3 - mean_x3)^2))
```

```
con_numbers <- c(con_number_x1,con_number_x2,con_number_x3)
con_numbers
```

```
## [1] 1.000225e+00 1.030265e+06 1.739757e+00
```

Comparison of condition number for different datasets

dataset	condition number
x1 (mean=0,sd=0.97)	1.000655
x2 (mean=1 000 000,sd=1.19)	0.8427983e+06
x3 (mean=0.000 001,sd=0.000 001)	1.425449

For dataset x1 the condition number is really good because it is close to one. Dataset x2 has a large condition number because it has a very high mean value. Dataset x3 where the requirement is not fulfilled has a very small condition number.