Statistical Simulation and Computerintensive Methods

Exercise 1

Valentin Peter (12143831)

2022-10-11

Sample Data

First we need to create some sample data.

```
set.seed(12143831)
x1 <- rnorm(100)
set.seed(12143831)
x2 <- rnorm(100, mean=1000000)
set.seed(12143831)
x3 <- rnorm(100, 10^10)
set.seed(12143831)
x4 <- rnorm(100, 0.0000001)
set.seed(12143831)
```

1. Varicance Calculation

Algorithm 1 (two - pass algorithm - variance calculation in R)

```
two_pass <- function(arr) {
  # Calculate mean
  sum <- 0
  for (x in arr) {
    sum <- sum + x
  }
  mean <- sum / length(arr)

  # Calculate variance
  sum_var <- 0
  for (x in arr) {
    sum_var <-  sum_var + (x - mean)**2
  }

  var <- sum_var / (length(arr) - 1)

  var
}

two_pass(x1)
```

```
## [1] 0.8731751
```

Algorithm 2 (one - pass algorithm - previously variance calculation in Excel)

```
excel <- function(arr) {
  # Calculate mean
  p1 <- 0
  p2 <- 0
  for (x in arr) {
    p1 <- (p1 + x^2)
    p2 <- p2 + x
  }

  p2 <- (p2^2) / length(arr)
  var <- (p1 -  p2) / (length(arr) - 1)
  var
}

excel(x1)
```

```
## [1] 0.8731751
```

Algorithm 3 (shifted one - pass algorithm)

```
shifted <- function(arr, c=arr[1]) {
  # Calculate mean
  p1 = 0
  p2 = 0
  for (x in arr) {
    p1 = p1 + (x - c)**2
    p2 = p2 + (x - c)
  }

  p2 = (p2**2) / length(arr)
  var = (p1 -  p2) / (length(arr) - 1)
  var
}

shifted(x1)
```

```
## [1] 0.8731751
```

Algorithm 4 (online algorithm)

```
online <- function(arr) {
  mean <- (arr[1] + arr[2]) / 2
  var <- ((arr[2] - arr[1])**2) / 2
  for (n in 3:length(arr)) {
    var <- ((n - 2) / (n - 1)) * var + ((arr[n] - mean)**2 / n)
    mean <- ((n - 1)* mean + arr[n]) / n
  }

  var
}

online(x1)
```

```
## [1] 0.8731751
```

R base mehod

```
var(x1)
```

```
## [1] 0.8731751
```

Comparison Function

```
test_methods <- function(x) {
  c(var(x), two_pass(x), excel(x), shifted(x), online(x))
}

test_methods(x1)
```

```
## [1] 0.8731751 0.8731751 0.8731751 0.8731751 0.8731751
```

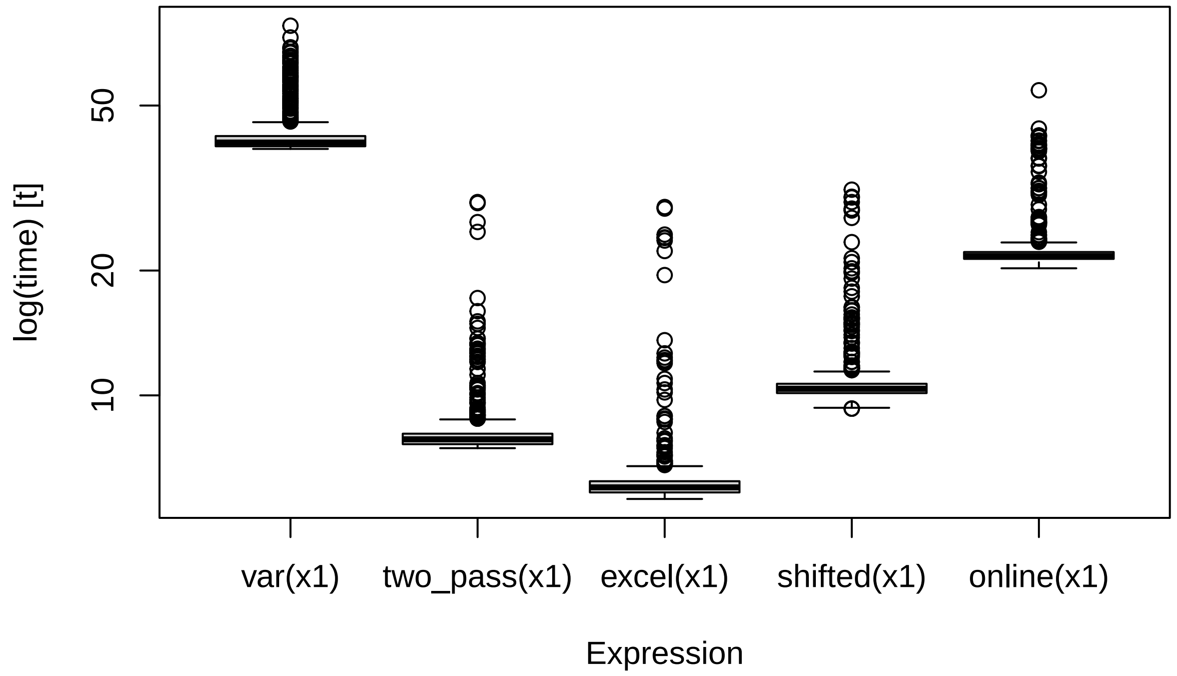
2. Comparison

Each function is executed 1000 times and then the execution times are shown in the boxplot below.

```
library(microbenchmark)

benchmarks <- microbenchmark(var(x1), two_pass(x1), excel(x1), shifted(x1), online(x1),
                              times=1000)

benchmarks
```

```
## Unit: microseconds
##          expr    min     lq      mean median     uq    max neval
##       var(x1) 39.293 39.875 42.027475 40.709 42.209 77.875  1000
##  two_pass(x1)  7.459  7.626  8.113801  7.834  8.084 29.251  1000
##     excel(x1)  5.626  5.834  6.272811  6.000  6.209 28.459  1000
##   shifted(x1)  9.292 10.126 10.750062 10.375 10.667 31.334  1000
##    online(x1) 20.250 21.334 22.325160 21.667 22.167 54.418  1000
```

```
boxplot(benchmarks)
```



The excel algorithm is by far the fastest. Then the shifted and the two pass algorithms come after. As expected the online algorithm was the slowest of the self implemented. Why the actual variance calculation from base R is so slow on my machine (ARM Processor) is unclear to me.

3. Scale Invariance Property

To rewind, let us compare the results of the different methods on different samples.

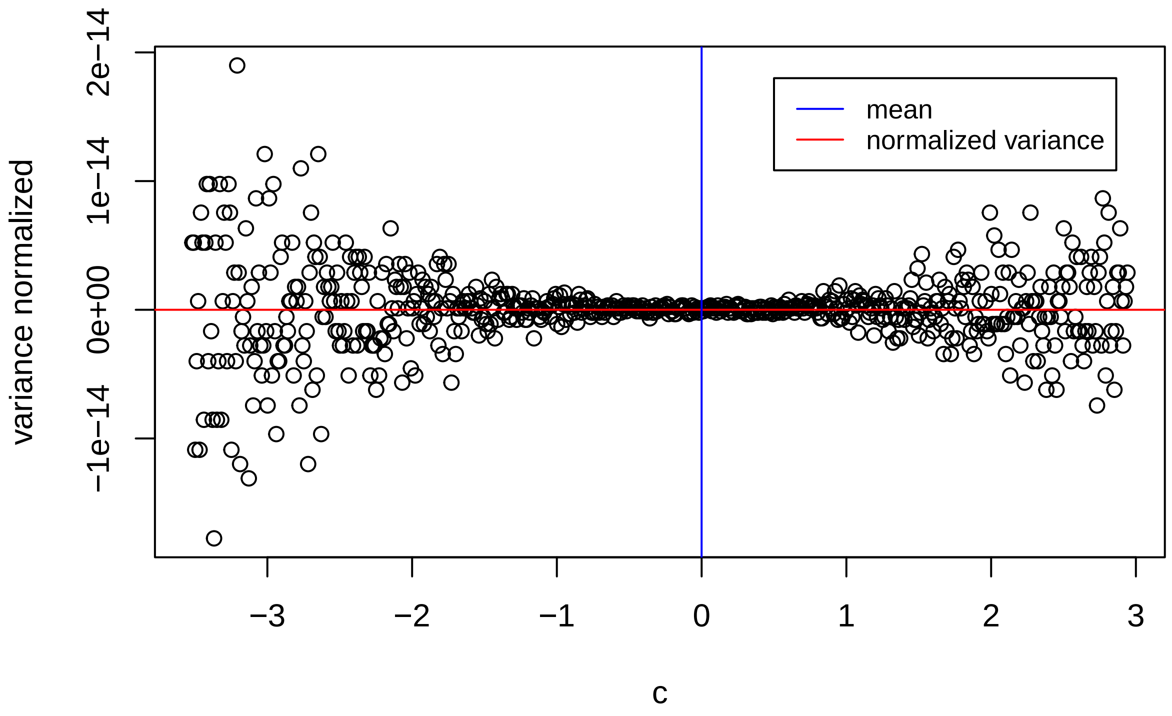
```
res <- rbind(test_methods(x1), test_methods(x2), test_methods(x3), test_methods(x3))
colnames(res) <- c("R var", "R alg", "excel", "shifted", "online")
rownames(res) <- c("mean=0", "mean=1000000", "mean=10^10","mean=0.0000001")

res
```

```
##                     R var     R alg         excel   shifted    online
## mean=0          0.8731751 0.8731751  8.731751e-01 0.8731751 0.8731751
## mean=1000000    0.8731751 0.8731751  8.732639e-01 0.8731751 0.8731751
## mean=10^10      0.8731752 0.8731752 -4.236671e+04 0.8731752 0.8731752
## mean=0.0000001 0.8731752 0.8731752 -4.236671e+04 0.8731752 0.8731752
```

Let us have a close look at the shifted algorithm. The default technique used so far is not calculation the mean but instead just taking the first data entry as a c-value. This works well on all 4 datasets and fixes the problem of the excel algorithm with the mean=1000000 and mean=10^10 sample.

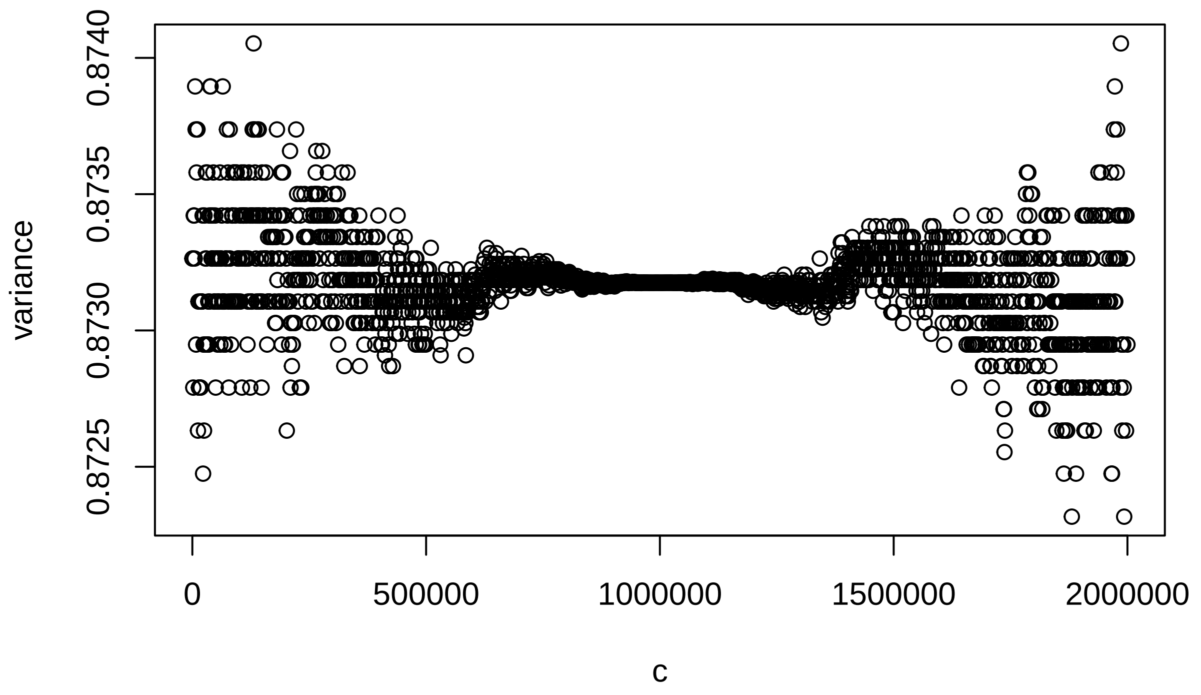
Further i want to showcase which effect different c values could have on the mean=1000000 sample.

```r
# Build sequence from the min and max values with a step size of 0.01.
min_x1 <- min(x1) - 1
max_x1 <- max(x1) + 1
sequence <- seq(from = min_x1, to = max_x1, by=0.01)
length <- length(sequence)
x <- matrix(nrow=length, ncol = 2)
colnames(x) <- c("c", "variance")

res <- c()
i<-0
for (c in sequence) {
  i<-i+1
  x[i, 1] <- c
  x[i, 2] <-  shifted(x1, c)
}

x_ <- x
x_[, 2] <- x[, 2] - var(x1)
plot(x_, ylab="variance normalized")
abline(h=0, col="red")
abline(v=mean(x1), col="blue")
legend(0.5, 1.8e-14,  legend=c("mean", "normalized variance"),
       col=c("blue", "red"), lty=1:1, cex=0.8)
```



As we can see in the two plots above the deviation from the correct variance is the lowest when we use a condition number that is close to our mean.

```r
sequence <- seq(from = 0, to = 2000000, by=1000)
length <- length(sequence)
x <- matrix(nrow=length, ncol = 2)
colnames(x) <- c("c", "variance")
i <- 0
for (c in sequence) {
  i <- i + 1
  x[i, 1] <- c
  x[i, 2] <-  shifted(x2, c)
}

plot(x)
```



The same effect is observed for the x2 data set with a mean of 1000000.

4. Condition Number

The condition number analyzes the robustness of an algorithm. It tell us something about the impact of small changes in the input to the output.

```r
calc_k <- function(x){
  return(sqrt(1 + (mean(x)**2 * length(x)) / sd(x)))
}
```

```
calc_k_shifted <- function(x, c){
 return(sqrt(1 + (length(x) / sd(x)) * (mean(x) - c)**2 ))
}
```

Further the condition numbers are calculated for all 4 samples and first without a shifted parameter then with the method that uses the first data entry as c parameter and then with the mean.

```
condition_numbers <- c("mean=0"=calc_k(x1), "mean=1000000"=calc_k(x2), "mean=10^10"=calc_k(x3),"mean=0.

condition_numbers_shifted_first <- c("mean=0"=calc_k_shifted(x1, x1[1]), "mean=1000000"=calc_k_shifted(

condition_numbers_shifted_mean <- c("mean=0"=calc_k_shifted(x1, mean(x1)), "mean=1000000"=calc_k_shifte

conditionNumbers_comparison <- data.frame("no_shift" = condition_numbers, shifted_first_value=condition_
conditionNumbers_comparison
```

```
##                     no_shift shifted_first_value shifted_mean
## mean=0                     1            3.951386            1
## mean=1000000        10344861            3.951386            1
## mean=10^10      103448606966            3.951385            1
## mean=0.0000001             1            3.951386            1
```

As seen in the table above without a c parameter the condition number for the variance is very high when the sample have high values. When using the first entry in the sample as a c parameter we already achieve a much better result. Only when using the mean as c we geht an optimal condition number of 1 as a result.