Introduction to Simulation with Variance

Kirill Medovshchikov

2022-10-14

Setup

```
library(microbenchmark)
```

In order to both avoid using R's native mean function and still avoid redundancy in code.

```
my_mean <- function(numbers){
  return(sum(numbers)/length(numbers))
}
```

Algorithms

A simple testing set to see if the algorithms' results match well.

```
test_set = c(1,2,5)
```

Gold standard R var() algorithm

```
var(test_set)
```

```
## [1] 4.333333
```

Two - pass algorithm

This is an implementation of the two-pass algorithm as shown in the lecture.

```
two_pass_algo <- function(numbers){
  squaredSum = 0
  for (n in numbers){
    squaredSum = squaredSum + (n - my_mean(numbers))^2
  }
  return(squaredSum / (length(numbers) - 1))
}

two_pass_algo(test_set)
```

```
## [1] 4.333333
```

One - pass algorithm

This is an implementation of the one-pass algorithm as shown in the lecture.

```
one_pass_algo <- function(numbers){
  p1 = 0
  for (n in numbers){
    p1 = p1 + n^2
  }
  p2 = sum(numbers)^2/length(numbers)
  return((p1 - p2)/(length(numbers)-1))
}

one_pass_algo(test_set)
```

```
## [1] 4.333333
```

Shifted one - pass algorithm

This is an implementation of the shifted one-pass algorithm as shown in the lecture. Here the constant c had been chosen to be represented by the first number in list.

```
shifted_one_pass_algo <- function(numbers){
  c = numbers[[1]][1]
  p1 = 0

  for (n in numbers){
    p1 = p1 + (n - c)^2
  }

  sum_minus_c = 0

  for (n in numbers){
    sum_minus_c = sum_minus_c + (n - c)
  }

  p2 = sum_minus_c^2/length(numbers)

  return((p1 - p2)/(length(numbers)-1))
}

shifted_one_pass_algo(test_set)
```

```
## [1] 4.333333
```

Online algorithm

This is an implementation of the online algorithm as shown in the lecture. For the first two numbers the mean and variance estimates are calculated statically. However starting from the 3rd number, the formulas become dynamic in their calculations.

```
online_algo <- function(numbers){
  n1 = numbers[[1]][1]
  n2 = numbers[[2]][1]
  numbs = c(n1, n2)
  squaredSum = 0
  for (n in numbs){
    squaredSum = squaredSum + (n - my_mean(numbs))^2
  }
  sn = squaredSum / length(numbs) - 1
  xn = my_mean(numbs)
  counter = 1
  for (n in numbers){
    if (counter > 2){
      append(numbs, n)
      xn_new = xn + (n - xn)/counter
      sn = (counter - 2)/(counter - 1) * (sn + n - xn)^2/counter
      xn = xn_new
    }
    counter = counter + 1
  }
  return(sn)
}

online_algo(test_set)
```

```
## [1] 1.260417
```

Algorithm wrapper

Here the function to call the required algorithm is created.

```
algo_wrapper <- function(numbers, name){
  if (name == "two_pass"){
    two_pass_algo(numbers)
  }
  else if (name == "one_pass"){
    one_pass_algo(numbers)
  }
  else if(name == "shifted_one_pass"){
    shifted_one_pass_algo(numbers)
  }
  else if(name == "online"){
    online_algo(numbers)
  }
}
```

Comparison

Data sets

Here three data sets are generated, where x2 has a mean of 1000000, and x3 which is not fulfilling the requirements, having its mean be too small. The seed is set to match the student number as required.

```r
set.seed(12144024)
x1 <- rnorm(100)
set.seed(12144024)
x2 <- rnorm(100, mean=1000000)
set.seed(12144024)
x3 <- rnorm(100, mean=10^1/1000000)
```

Algorithm Results

The following presents the results of all algorithm in numbers, being called one by one on each data set accordingly. The results of algorithms are identical in between the data sets and each other, except for the online algorithm, that happens to be showing a much lower value than its counterparts, though having this value be consistent over the both data sets.

```r
var(x1)
```

```
## [1] 1.113746
```

```r
algo_wrapper(x1, "two_pass")
```

```
## [1] 1.113746
```

```r
algo_wrapper(x1, "one_pass")
```

```
## [1] 1.113746
```

```r
algo_wrapper(x1, "shifted_one_pass")
```

```
## [1] 1.113746
```

```r
algo_wrapper(x1, "online")
```

```
## [1] 0.01037335
```

```r
var(x2)
```

```
## [1] 1.113746
```

```r
algo_wrapper(x2, "two_pass")
```

```
## [1] 1.113746
```

```r
algo_wrapper(x2, "one_pass")
```

```
## [1] 1.113636
```

```
algo_wrapper(x2, "shifted_one_pass")
```

```
## [1] 1.113746
```

```
algo_wrapper(x2, "online")
```

```
## [1] 0.01037335
```

Benchmarks

In this code chunk a benchmark on the first x1 data set is performed with all five algorithms. The results are then presented in a form of a table.

```
benchmark_1 = microbenchmark(var(x1), algo_wrapper(x1, "two_pass"), algo_wrapper(x1, "one_pass"), algo_
benchmark_1
```

```
## Unit: microseconds
##                                  expr     min       lq      mean   median
##                               var(x1)   6.342   7.3525  11.12789  10.3965
##          algo_wrapper(x1, "two_pass")  63.438  65.3045  75.19574  67.2110
##          algo_wrapper(x1, "one_pass")   3.895   4.2625   5.25254   4.6115
##  algo_wrapper(x1, "shifted_one_pass")   6.875   7.3870   9.06555   7.8500
##            algo_wrapper(x1, "online") 100.936 105.6900 117.69498 115.6430
##        uq     max neval
##   11.7185  47.550   100
##   78.0550 148.601   100
##    5.2770  23.776   100
##    8.6015  25.054   100
##  125.0860 181.618   100
```

The same benchmark is then performed with the x2 data set and its results are again stored in a table form.

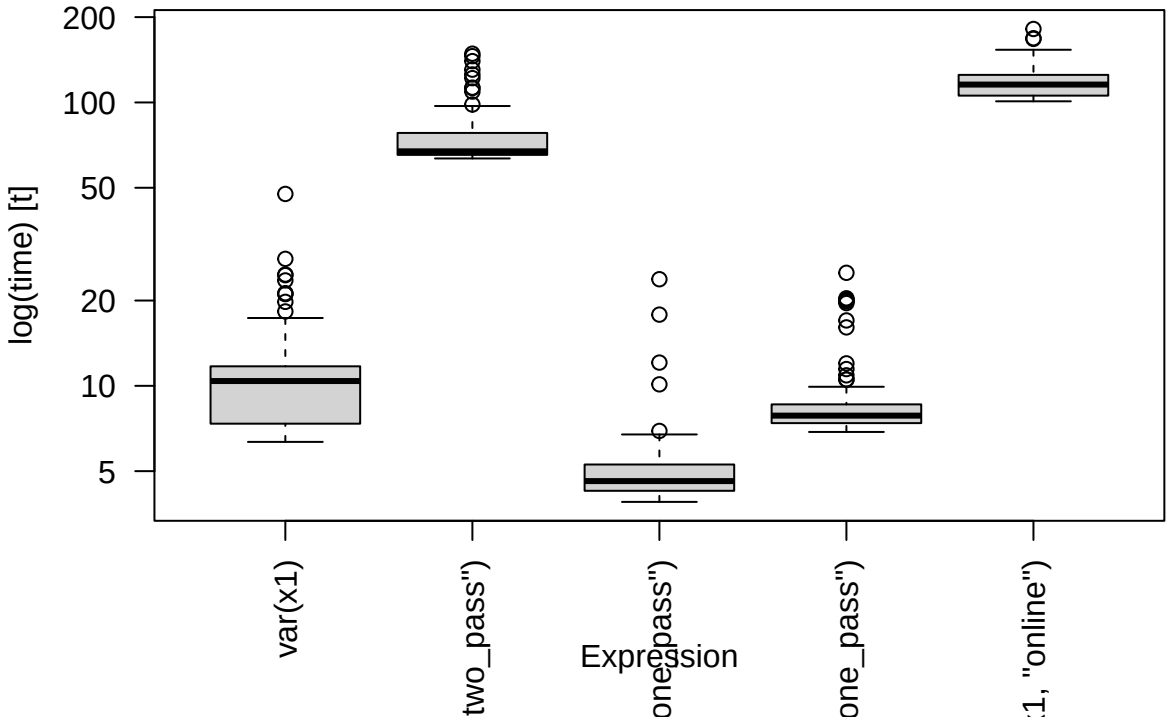
```
benchmark_2 = microbenchmark(var(x2), algo_wrapper(x2, "two_pass"), algo_wrapper(x2, "one_pass"), algo_
benchmark_2
```

```
## Unit: microseconds
##                                  expr     min       lq      mean   median
##                               var(x2)   6.261   7.1835  11.08088  10.2725
##          algo_wrapper(x2, "two_pass")  64.285  66.6355  74.96745  69.1850
##          algo_wrapper(x2, "one_pass")   3.934   4.3460   5.97819   4.7670
##  algo_wrapper(x2, "shifted_one_pass")   6.920   7.3655   9.02210   7.7500
##            algo_wrapper(x2, "online") 101.158 106.6405 117.48983 115.6790
##        uq     max neval
##   11.8240  50.334   100
##   78.9820 137.914   100
##    5.7475  53.233   100
##    8.3480  60.638   100
##  120.9675 182.044   100
```

Box-plots

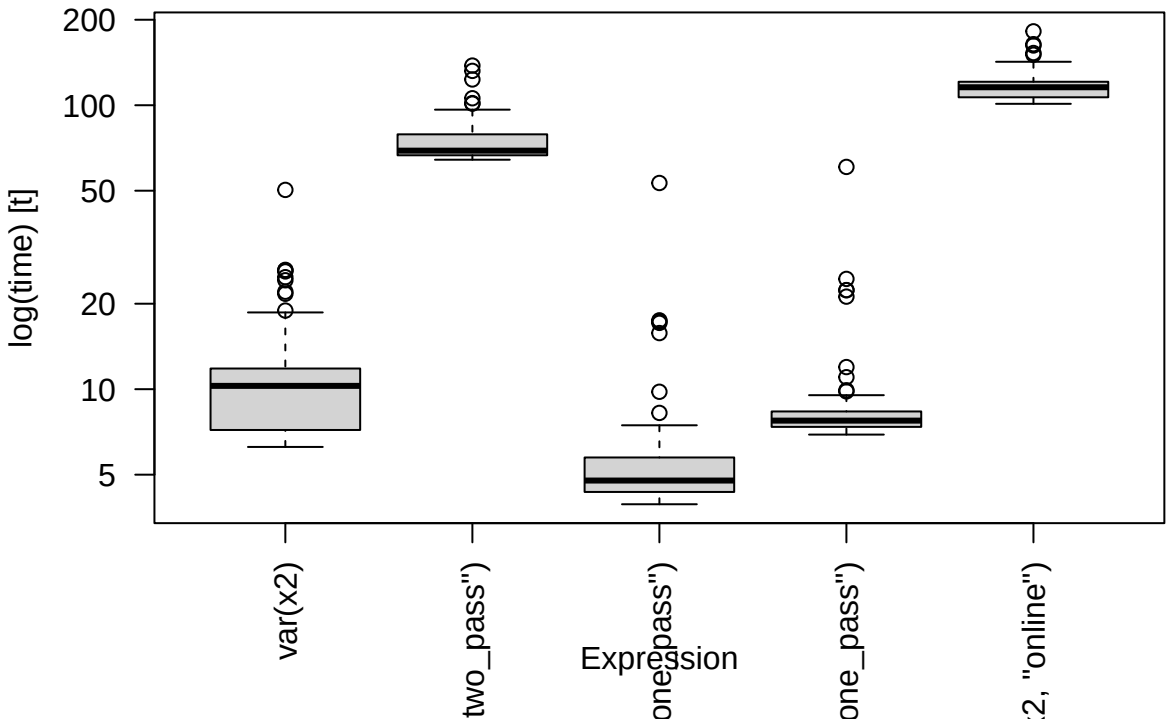
The following box plot shows how well every given algorithm performs. As it can be seen, from the box plot, the online algorithm performed the worst out of them all, with the one pass algorithm performing best on the first x1 data set.

```
boxplot(benchmark_1, las=2)
```



In the x2 data set the performance situation hasn't changed much, the online algorithm being the worst and one pass being the best.

```
boxplot(benchmark_2, las=2)
```



Condition Number for Variance

In the following code chunk a condition number for the variance is calculated by using the formula presented during the lecture. All of the results are bigger than 1, therefore satisfying the requirement.

```
cond_num_var <- function(numbers){
  s = 0
  for (n in numbers){
    s = s + (n - (sum(numbers))/length(numbers))^2
  }
  return(sqrt(1 + (my_mean(numbers)^2 * length(numbers)) / s))
}
```

```
cond_num_var(x1)
```

```
## [1] 1.002775
```

```
cond_num_var(x2)
```

```
## [1] 952334.2
```

```
cond_num_var(x3)
```

```
## [1] 1.002774
```

Condition Number for Shifted Variance

Here now the condition number for shifted variance is calculated by using the formula from the slides. As it is also said there that the c has to be equal to mean, all of the results happen to be 1s, as can be seen below.

```
cond_num_shifted_var <- function(numbers){
  s = 0
  c = my_mean(numbers)
  for (n in numbers){
    s = s + (n - (sum(numbers))/length(numbers))^2
  }
  return(sqrt(1 + (length(numbers)/s)*(my_mean(numbers) - c)^2))
}
```

```
cond_num_shifted_var(x1)
```

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

```
cond_num_shifted_var(x2)
```

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

```
cond_num_shifted_var(x3)
```

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

Comparison

In the following code snippets, every of the condition number calculations is compared against its shifted counterpart with a matching data set.

```
benchmark_x1 = microbenchmark(cond_num_var(x1), cond_num_shifted_var(x1))
benchmark_x1
```

```
## Unit: microseconds
##                       expr    min     lq     mean  median      uq     max neval
##          cond_num_var(x1) 36.749 37.471 42.15238 38.2400 40.1465 128.951   100
##  cond_num_shifted_var(x1) 37.455 38.431 44.30312 39.5695 48.9055  91.936   100
```

```
benchmark_x2 = microbenchmark(cond_num_var(x2), cond_num_shifted_var(x2))
benchmark_x2
```

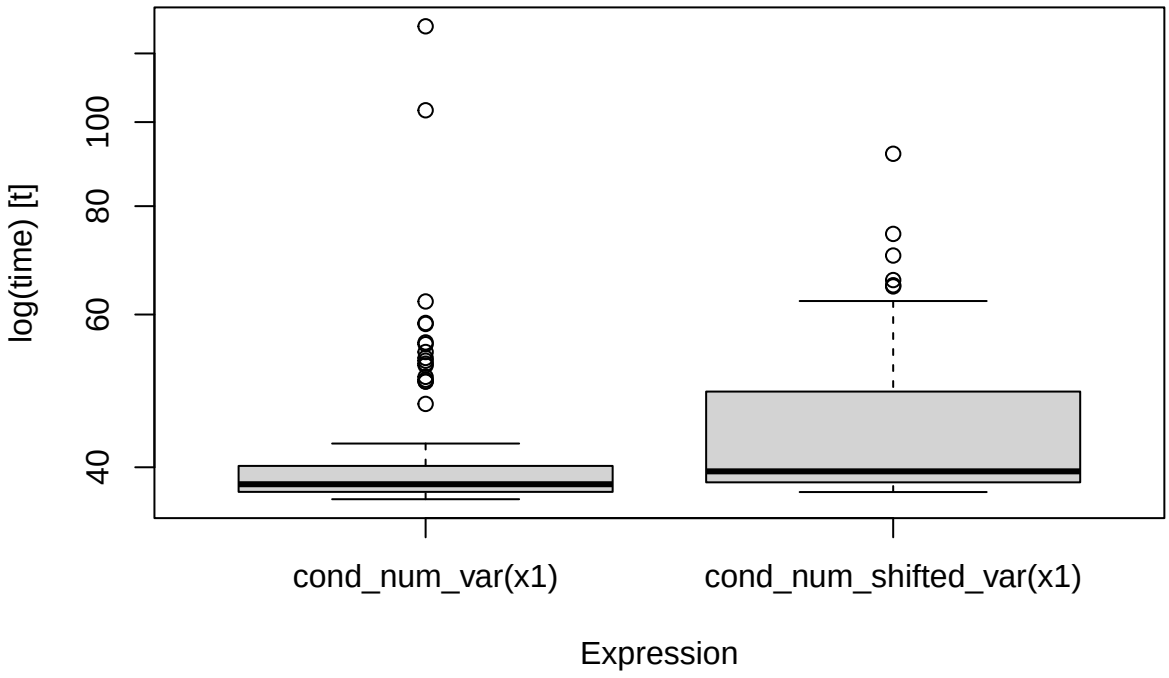
```
## Unit: microseconds
##                       expr    min     lq     mean median      uq    max neval
##          cond_num_var(x2) 36.332 37.667 42.53642 38.715 45.5305 66.239   100
##  cond_num_shifted_var(x2) 37.105 38.402 41.86469 39.351 43.4445 61.453   100
```

```
benchmark_x3 = microbenchmark(cond_num_var(x3), cond_num_shifted_var(x3))
benchmark_x3
```

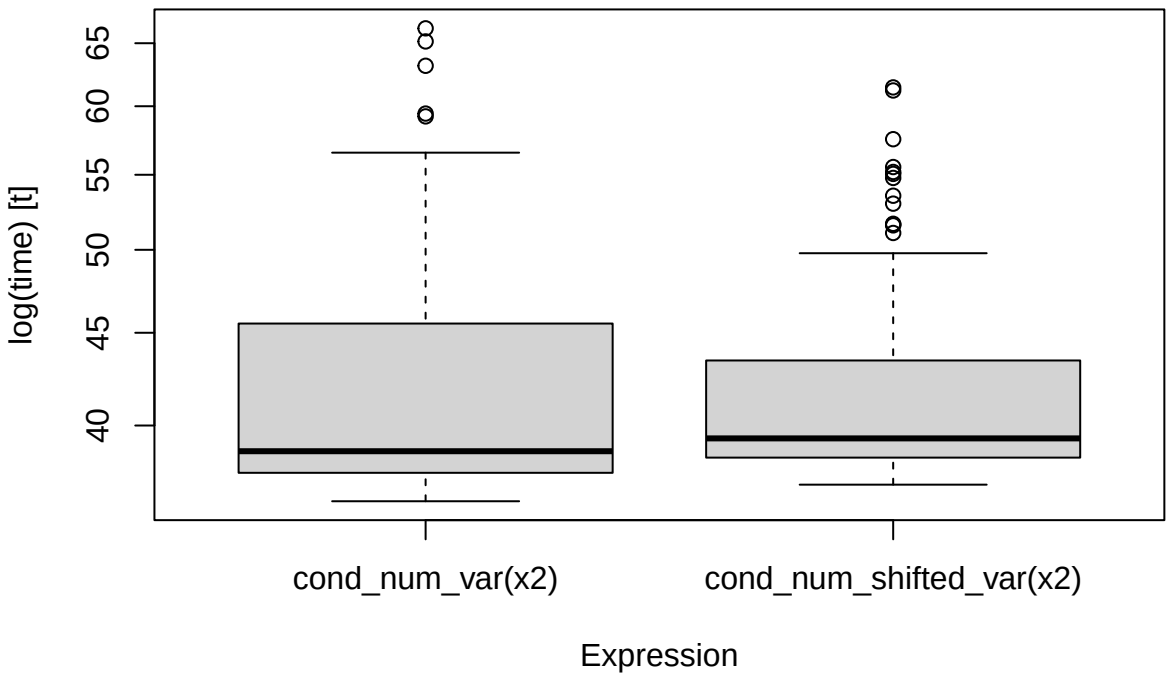
```
## Unit: microseconds
##                       expr    min     lq     mean median      uq     max neval
##          cond_num_var(x3) 36.448 37.933 45.62115 39.196 44.5855 240.388   100
##  cond_num_shifted_var(x3) 37.335 39.124 44.42756 40.342 45.0200  88.019   100
```

As it can be seen from the box plots of the previously mentioned benchmarks, condition number for variance is almost always a tiny bit faster or on par with the shifted variance algorithm.

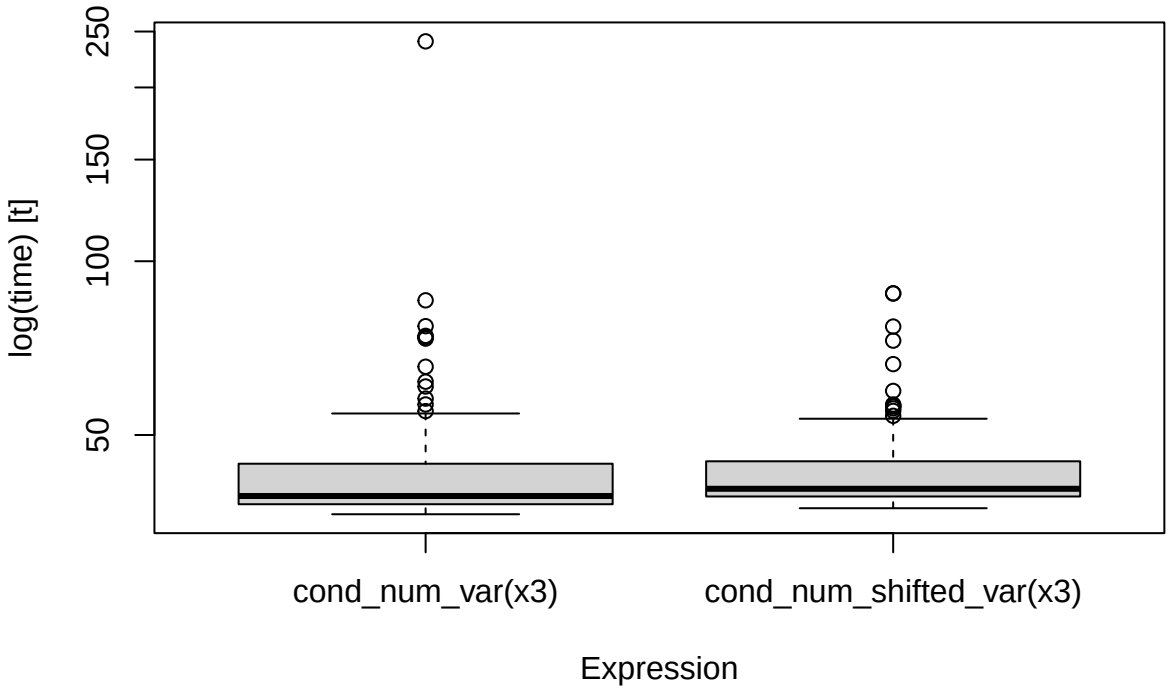
```
boxplot(benchmark_x1)
```




```
boxplot(benchmark_x2)
```




```
boxplot(benchmark_x3)
```


log(time) [t]
250
150
100
50
cond_num_var(x3)
cond_num_shifted_var(x3)
Expression