Variance Calculation: Comparing Algorithms

Mahtab Nahayati

2024-10-15

Introduction

In this document, we are comparing four different algorithms for calculating variance:

- Two-pass algorithm

- One-pass algorithm (Excel method)

- Shifted one-pass algorithm

- Online algorithm

We will compare their results against R's built-in var() function as a gold standard. This analysis will also compare the computational performance of these algorithms.

R's Variance

```r
x<-c(1:20) #Sample vector
var(x) #Bult-in R function for variance
```

```
## [1] 35
```

Implementing the four algorithms for variance calculation and compare them to R's built-in var()functions

Algorithm 1: Two-pass Algorithm

The two-pass algorithm first calculates the mean of the data, and then it computes the variance in a second pass through the data. This method is numerically stable and should provide the same result as R's var() function.

```r
var_two_pass <- function(x){

  #compute the mean
  x_bar <- mean(x)

  #Compute variance using the two-pass formula
  n<- length(x)
  variance <- sum((x - x_bar)^2) / (n - 1)

  return(variance)

}

var_two_pass(x)
```

```
## [1] 35
```

Algorithm 2: One-pass Algorithim (Excel Method)

```r
var_one_pass <- function(x){
  n<- length(x)
  P1 <- sum(x^2)  # sum of squared elements
  P2 <- (sum(x)^2) / n  # squared sum of elements divided by n
  variance <- (P1 - P2) / (n - 1)

  return(variance)
}

var_one_pass(x)
```

```
## [1] 35
```

Algorithm 3: Shifted one-pass Algorithm

```r
var_shifted_one_pass <- function(x) {
  n <- length(x)
  c <- x[1]  # Use the first element as shift

  P1 <- sum((x - c)^2)
  P2 <- (sum(x - c)^2) / n
  variance <- (P1 - P2) / (n - 1)

  return(variance)
}

var_shifted_one_pass(x)
```

```
## [1] 35
```

Algorithm 4: Online Algorithm

```r
var_online <- function(x) {
  n <- length(x)
  x_bar <- x[1]
  s2 <- 0

  for (i in 2:n) {
    new_mean <- x_bar + (x[i] - x_bar) / i
    s2 <- s2 + (x[i] - x_bar) * (x[i] - new_mean)
    x_bar <- new_mean
  }

  variance <- s2 / (n - 1)
  return(variance)
}

var_online(x)
```

```
## [1] 35
```

Wrapper Function

The variance_wrapper() function allows us to run all the algorithms in a single call. It returns a list containing the results from R's var() and each of the custom variance algorithms for easy comparison.

```
variance_wrapper <- function(x) {
  results <- list(
    "R's var()" = var(x),
    "Two-pass" = var_two_pass(x),
    "One-pass" = var_one_pass(x),
    "Shifted One-pass" = var_shifted_one_pass(x),
    "Online" = var_online(x)
  )

  return(results)
}

#Below is the comparison of the results for each algorithm and R's `var()` function. As we
#can see, all algorithms produce the same result, which confirms their correctness.

variance_wrapper(x)
```

```
## $`R's var()`
## [1] 35
##
## $`Two-pass`
## [1] 35
##
## $`One-pass`
## [1] 35
##
## $`Shifted One-pass`
## [1] 35
##
## $Online
## [1] 35
```

Data Preparation and comparison

```
# Data set 1
set.seed(1328781)
x1 <- rnorm(100)

# Data set 2 with a large mean
set.seed(1328781)
x2 <- rnorm(100, mean = 1000000)

# Call the wrapper function
results_x1 <- variance_wrapper(x1)
results_x2 <- variance_wrapper(x2)

# Print results
results_x1
```

```
## $`R's var()`
```

```
## [1] 1.110056
##
## $`Two-pass`
## [1] 1.110056
##
## $`One-pass`
## [1] 1.110056
##
## $`Shifted One-pass`
## [1] 1.110056
##
## $Online
## [1] 1.110056
```

```r
results_x2
```

```
## $`R's var()`
## [1] 1.110056
##
## $`Two-pass`
## [1] 1.110056
##
## $`One-pass`
## [1] 1.109848
##
## $`Shifted One-pass`
## [1] 1.110056
##
## $Online
## [1] 1.110056
```

Compare Computational Performance (Using Microbenchmark)

```r
install.packages("microbenchmark")
```

```r
library(microbenchmark)
```

```
## Warning: Paket 'microbenchmark' wurde unter R Version 4.3.3 erstellt
```

```r
# Benchmark the algorithms for x1
benchmark_x1 <- microbenchmark(
  "R's var()" = var(x1),
  "Two-pass" = var_two_pass(x1),
  "One-pass" = var_one_pass(x1),
  "Shifted One-pass" = var_shifted_one_pass(x1),
  "Online" = var_online(x1),
  times = 100
)

# Print benchmark results
benchmark_x1
```

```
## Unit: microseconds
##              expr   min    lq     mean  median      uq    max neval
##         R's var() 6.701 7.701 11.18201  9.0010 11.4010 30.601   100
##          Two-pass 6.201 7.101 10.48690  8.1515  9.8505 45.200   100
```

```
##          One-pass  1.200  1.451  2.95593  1.6515  2.1510  20.901   100
##  Shifted One-pass  1.500  1.801  3.52898  2.1000  2.4010 113.702   100
##            Online 10.201 10.701 17.89900 12.9515 15.7510  95.800   100
```

```r
# Benchmark the algorithms for x2
benchmark_x2 <- microbenchmark(
  "R's var()" = var(x2),
  "Two-pass" = var_two_pass(x2),
  "One-pass" = var_one_pass(x2),
  "Shifted One-pass" = var_shifted_one_pass(x2),
  "Online" = var_online(x2),
  times = 100
)

# Print benchmark results
benchmark_x2
```

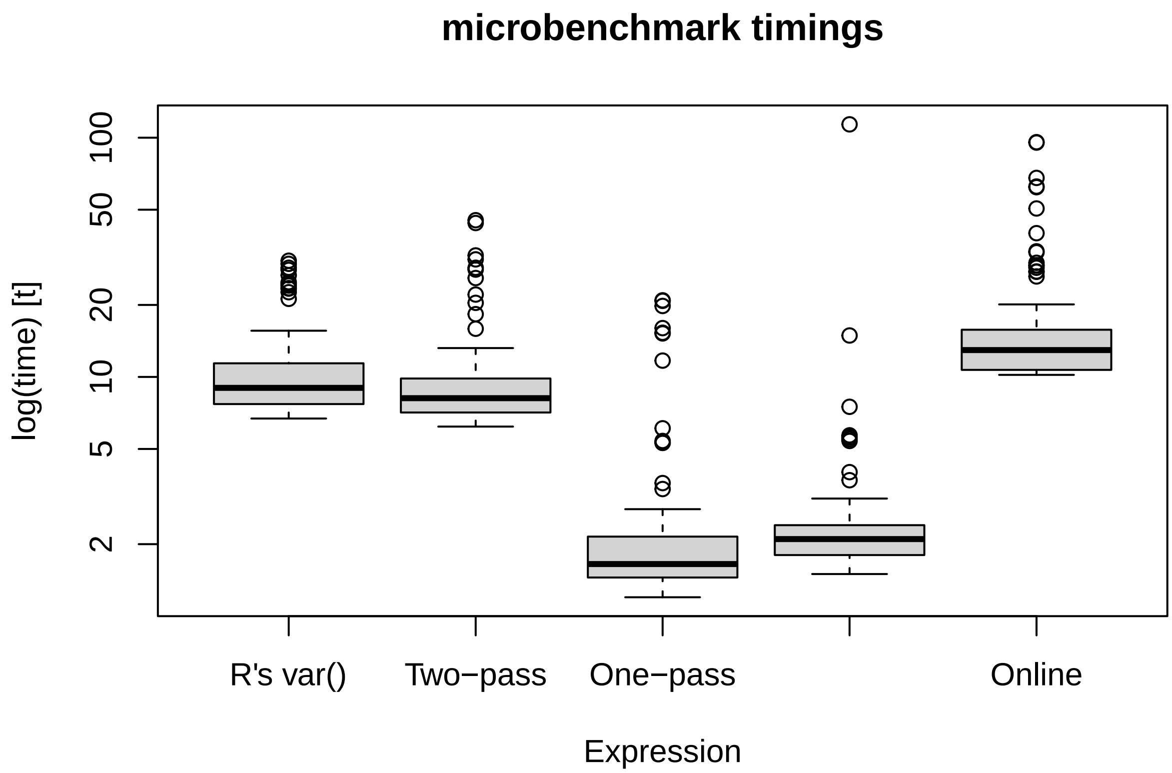
```
## Unit: microseconds
##              expr    min      lq     mean  median      uq    max neval
##         R's var()  7.300  8.6010  9.63502  9.0005  9.6000 60.000   100
##          Two-pass  6.800  7.6510  8.43997  8.0505  8.4010 26.101   100
##          One-pass  1.101  1.5010  1.85406  1.7000  1.8010  7.501   100
##  Shifted One-pass  1.601  1.9010  2.44799  2.0020  2.2020 10.502   100
##            Online 10.001 13.0015 13.44299 13.5015 14.0505 17.101   100
```

Visualize Performance (Boxplot)

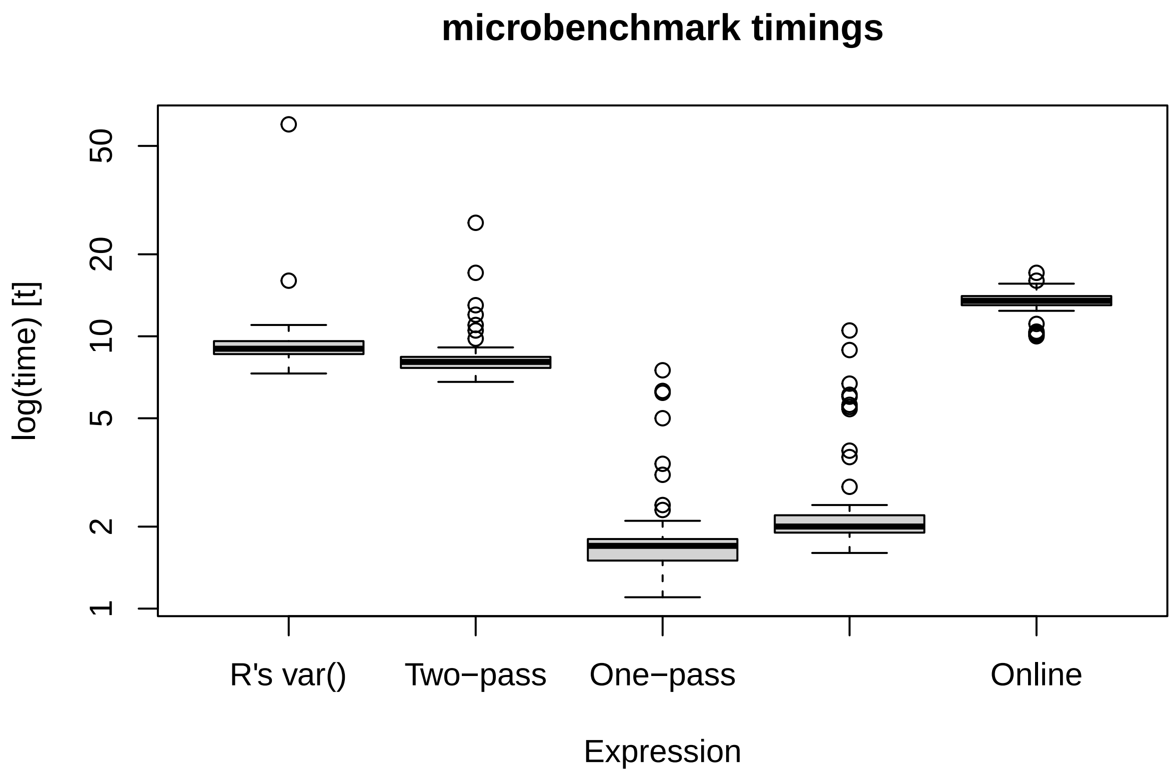
```r
library(ggplot2)
```

```
## Warning: Paket 'ggplot2' wurde unter R Version 4.3.2 erstellt
```

```r
# Boxplot for x1
boxplot(benchmark_x1)
```




```r
# Boxplot for x2
boxplot(benchmark_x2)
```



- From the boxplots, we observe that the One-pass and Shifted One-pass algorithms tend to perform the fastest across both data sets. - The Two-pass and Online algorithms are slightly slower compared to the One-pass methods. - R's built-in var() function performs similarly to the Two-pass algorithm but shows more variability in some cases. - The results align with our expectations: one-pass algorithms tend to be faster since they process the data in a single iteration, while two-pass algorithms require an additional pass over the data.

Investigate Scale Invariance and condition Number

```r
# Original data
x <- c(1:20)

# Shifting the data by adding or subtracting a constant
y <- x + 50   # Shift by adding 50
z <- x - 100  # Shift by subtracting 100

# Calculate variances
var_x <- var(x)
var_y <- var(y)
var_z <- var(z)

# Print results
list(
  "Original Variance" = var_x,
  "Variance after adding 50" = var_y,
  "Variance after subtracting 100" = var_z
```

```
)
```

```
## $`Original Variance`
## [1] 35
##
## $`Variance after adding 50`
## [1] 35
##
## $`Variance after subtracting 100`
## [1] 35
```

Condition Number Calcualtion

The condition number measures the robustness or stability of an algorithm with respect to changes in input. For variance, this condition number is highly dependent on the mean. A well-conditioned variance calculation (using a good mean) will perform more stably than a poorly-conditioned one.

```r
condition_number <- function(x) {
  n <- length(x)
  x_bar <- mean(x)
  S <- sum((x - x_bar)^2)
  kappa <- sqrt(1 + (x_bar^2 * n) / S)

  return(kappa)
}

# Compute condition numbers for different data sets
kappa_x <- condition_number(x)  # Original
kappa_y <- condition_number(y)  # Shifted by +50
kappa_z <- condition_number(z)  # Shifted by -100

# Print the condition numbers
list(
  "Condition number (Original)" = kappa_x,
  "Condition number (+50 shift)" = kappa_y,
  "Condition number (-100 shift)" = kappa_z
)
```

```
## $`Condition number (Original)`
## [1] 2.077448
##
## $`Condition number (+50 shift)`
## [1] 10.53958
##
## $`Condition number (-100 shift)`
## [1] 15.55345
```

the condition number does not change significantly with a shift in data (scale invariance), but a poor choice of mean can lead to a high condition number, indicating instability in the variance calculation.

Why the Mean Performs Best: From the condition number calculation, we can argue,the mean serves as the best shift for stabilizing the variance calculation because it minimizes the condition number. If the data is poorly centered, the condition number increases, indicating that the variance calculation could become unstable or sensitive to small input changes.

Simulate Data for Comparison

```r
# Data set 1 (mean ~ 0)
set.seed(1)
x1 <- rnorm(100)

# Data set 2 (mean = 1,000,000)
set.seed(1)
x2 <- rnorm(100, mean = 1000000)

# Call the wrapper function for both data sets
results_x1 <- variance_wrapper(x1)
results_x2 <- variance_wrapper(x2)

# Print results
results_x1
```

```
## $`R's var()`
## [1] 0.8067621
##
## $`Two-pass`
## [1] 0.8067621
##
## $`One-pass`
## [1] 0.8067621
##
## $`Shifted One-pass`
## [1] 0.8067621
##
## $Online
## [1] 0.8067621
```

```r
results_x2
```

```
## $`R's var()`
## [1] 0.8067621
##
## $`Two-pass`
## [1] 0.8067621
##
## $`One-pass`
## [1] 0.8066604
##
## $`Shifted One-pass`
## [1] 0.8067621
##
## $Online
## [1] 0.8067621
```

Reproduce Results from Comparison1

```r
# Create a table of results for x1 and x2
results_table_x1 <- data.frame(
  Method = names(results_x1),
  Variance = unlist(results_x1)
```

```r
)

results_table_x2 <- data.frame(
  Method = names(results_x2),
  Variance = unlist(results_x2)
)

# Print the tables
results_table_x1
```

```
##                               Method  Variance
## R's var()                  R's var() 0.8067621
## Two-pass                    Two-pass 0.8067621
## One-pass                    One-pass 0.8067621
## Shifted One-pass Shifted One-pass 0.8067621
## Online                        Online 0.8067621
```

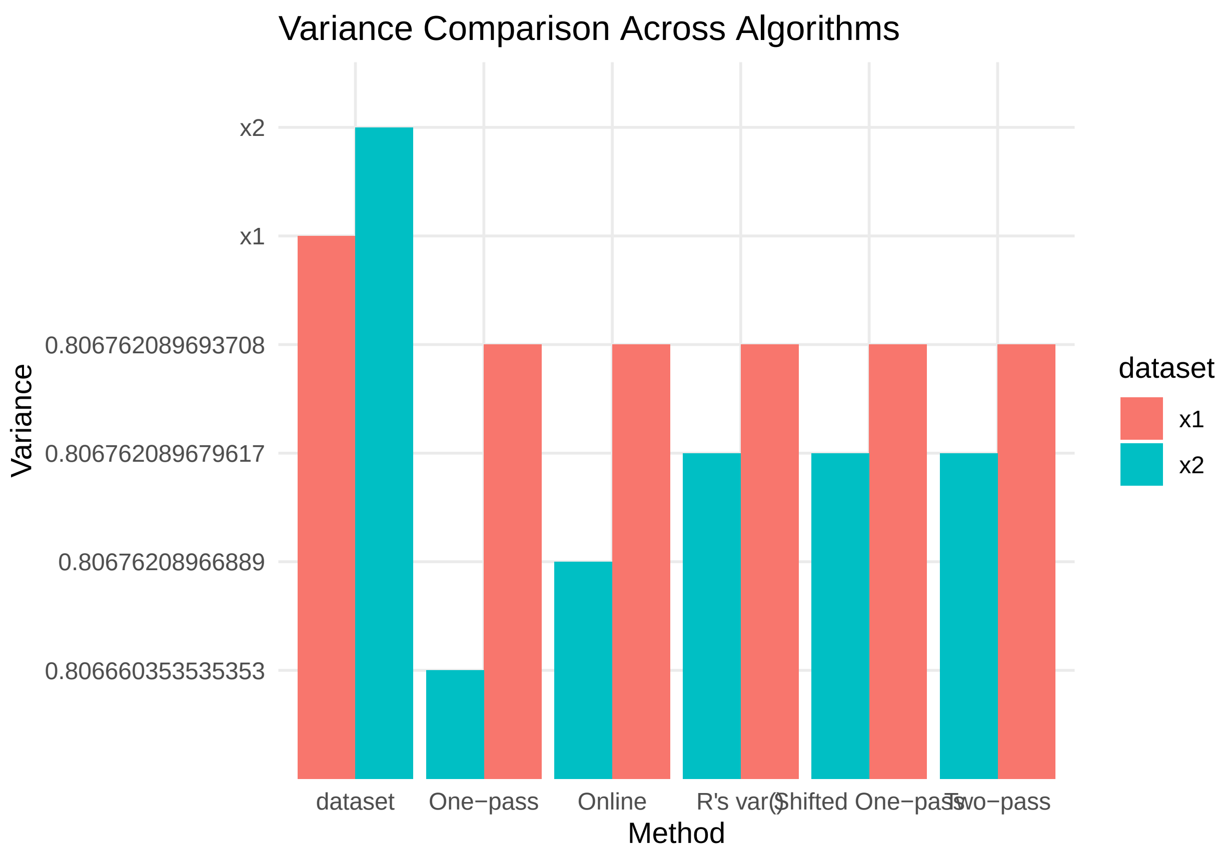
```r
results_table_x2
```

```
##                               Method  Variance
## R's var()                  R's var() 0.8067621
## Two-pass                    Two-pass 0.8067621
## One-pass                    One-pass 0.8066604
## Shifted One-pass Shifted One-pass 0.8067621
## Online                        Online 0.8067621
```

Visualize the Result

```r
# Combine results for plotting
results_x1$dataset <- "x1"
results_x2$dataset <- "x2"
combined_results <- rbind(
  data.frame(Method = names(results_x1), Variance = unlist(results_x1), dataset = "x1"),
  data.frame(Method = names(results_x2), Variance = unlist(results_x2), dataset = "x2")
)

# Plot using ggplot
ggplot(combined_results, aes(x = Method, y = Variance, fill = dataset)) +
  geom_bar(stat = "identity", position = "dodge") +
  labs(title = "Variance Comparison Across Algorithms", x = "Method", y = "Variance") +
  theme_minimal()
```



In `x1` (mean near 0), all algorithms produce nearly identical variance values, showing that each algorithm is stable when the data is well-centered.

However, in `x2` (mean = 1,000,000), we notice a slight difference in the One-pass algorithm's variance calculation. This is likely due to numerical instability when handling very large numbers, as the one-pass algorithm combines all operations in a single loop, which can introduce rounding errors with large inputs.

Condition Number From the condition number calculations, we observe that the condition number increases when the mean is far from zero, indicating that the variance calculation becomes less stable. This further supports the idea that centering the data (choosing the mean as the shift) results in the best performance, as it minimizes the condition number and reduces numerical instability.

Compute Condition Numbers for x1 and x2

```
# Compute condition numbers for the two datasets
kappa_x1 <- condition_number(x1)  # For x1
kappa_x2 <- condition_number(x2)  # For x2

# Print the results
list(
  "Condition number (x1 - mean ~ 0)" = kappa_x1,
  "Condition number (x2 - mean = 1,000,000)" = kappa_x2
)
```

```
## $`Condition number (x1 - mean ~ 0)`
## [1] 1.007395
##
```

```
## $`Condition number (x2 - mean = 1,000,000)`
## [1] 1118947
```

The condition number for x1 is relatively low because the data is well-centered. The condition number for x2 is higher due to the large mean, making the variance calculation less stable.

Create a Third Dataset x3 where the Requirement is Not Fulfilled

```r
# Data set 3: Almost no variance
x3 <- c(rep(1, 99), 100000)  # 99 values are 1, and one value is 100,000

# Compute condition number for the third dataset
kappa_x3 <- condition_number(x3)

# Print the condition number for x3
kappa_x3
```

```
## [1] 1.005048
```

Compute the Condition Number for All Three Datasets

```r
# Create a comparison table for the condition numbers
condition_table <- data.frame(
  Dataset = c("x1 (mean ~ 0)", "x2 (mean = 1,000,000)", "x3 (poorly centered)"),
  Condition_Number = c(kappa_x1, kappa_x2, kappa_x3)
)

# Print the table
condition_table
```

```
##                 Dataset Condition_Number
## 1         x1 (mean ~ 0)     1.007395e+00
## 2 x2 (mean = 1,000,000)     1.118947e+06
## 3  x3 (poorly centered)     1.005048e+00
```

Dataset `x1`: The condition number is very close to 1, indicating a well-centered dataset with a stable variance calculation. Since the mean of this dataset is near 0, the condition number confirms that the data is well-suited for accurate variance calculations.

- **Dataset `x2`**: The condition number is extremely high, around 1.1 million. This shows that when the mean is very large, the variance calculation becomes unstable. The large condition number indicates that small numerical errors in the data could lead to significant inaccuracies in the variance calculation, making this dataset problematic for statistical calculations.
- **Dataset `x3`**: Despite being designed as "poorly centered" with an extreme outlier, the condition number is similar to that of `x1`. This can be explained by the fact that 99 values in the dataset are identical (all 1), leading to a small overall variance. The presence of the outlier (100,000) doesn't drastically affect the condition number because the outlier is a single value among 99 identical values. Thus, the condition number might not always reflect instability in datasets with extreme outliers, as the bulk of the data is uniform.

Conclusion

From these results, we can conclude that: - The condition number effectively identifies datasets with large means (like `x2`) as problematic for variance calculations. - However, datasets like `x3` (with extreme outliers but low overall variance) may not show a high condition number, even though the presence of an outlier could cause numerical instability. Therefore, while the condition number is a useful measure, additional checks may be needed to identify datasets that contain extreme values or other irregularities that could affect variance calculations.