732A90: Computational Statistics

Computer lab1 - Group11

Sofie Jörgensen, Oriol Garrobé Guilera, David Hrabovszki

28 January 2020

Question 1: Be careful when comparing

In this task, we are given two R code snippets, each consisting of a comparision of rational numbers. The first code snippet compares $\frac{1}{3} - \frac{1}{4}$ with $\frac{1}{12}$, and the second code snippet compares $1 - \frac{1}{2}$ with $\frac{1}{2}$. For both code snippets, the comparison is performed by the logical operator `==`, and prints an output, which is either "Subtraction is correct" or "Subtraction is wrong". The code snippets can be found in the Appendix.

1.

After running the two code snippets, we observe that the first snippet prints the output "Subtraction is wrong", while the second snippet prints "Subtraction is correct". In both cases, the comparisions should be mathematically correct, but since floats are rounded in R, the values $\frac{1}{3}$ and $\frac{1}{12}$ are not precisely represented. In other words, all the repeating decimals cannot be stored in R so the value is rounded. Since the snippets use the logical operator `==`, corresponding to exactly equality, this implies that the two sides in the first snippet are not precisely equal. Thus it will return "Subtraction is wrong".

2.

The code causes problems as already mentioned in Question 1, which confirmes that comparisons should be carried out carefully. One suggestion of improvement is to use the functions `all.equal()` and `isTRUE()`, instead of the logical operator `==`. The function `all.equal()` tests if two objects are nearly equal, in contrast to `==` which tests exact equality. The funciton `isTRUE()` is necessary to handle the `FALSE` result of `all.equal()` in the `if` statement. This improvement of the code snippet (see Appendix) prints "Subtraction is correct" when comparing $\frac{1}{3} - \frac{1}{4}$ with $\frac{1}{12}$, which is the mathematically correct output.

Question 2: Derivative

We are going to write our own R function of the definition of a derivative at a point x with a small ϵ, given by the formula $f'(x) = \frac{f(x+\epsilon)-f(x)}{\epsilon}$.

1.

We calculate the derivative of $f(x) = x$, with $\epsilon = 10^{-15}$ using our own R function (see Appendix).

2. and 3.

Now we evaluate our derivative function at $x = 1$ and $x = 100000$ and obtain the results $f'(1) = 1.110223$ and $f'(100000) = 0$ from the output. The true values of the derivatives are 1 for every x, and we expect our function to output a value close to 1. The obtained derivative for $x = 1$ is close to the true value, but for large x we get the result 0. If we consider x within the range $[0, 1[$, we obtain approximately 1. However, when using larger values of x, the difference between large numbers, dominates the small epsilon, which gives us zero in the numerator and thus the derivative evaluated at large points is 0. In general, adding small numbers first will give a better accuracy, than adding large number first, which we are doing when using the definition of the derivative.

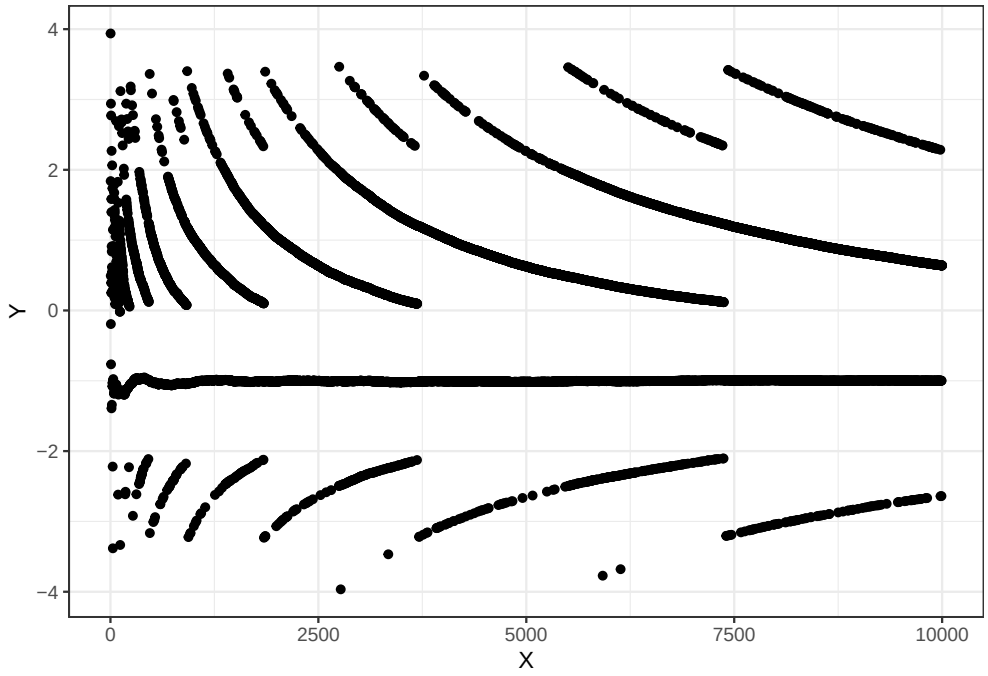

Figure 1: Difference between the functions myvar(X_i) and var(X_i)

Question 3: Variance

The variance based on a vector $\vec{x}$ of n observations can be estimated by using the formula

$$Var(\vec{x}) = \frac{1}{n-1}\left(\sum_{i=1}^{n} x_i^2 - \frac{1}{n}\Big(\sum_{i=1}^{n} x_i\Big)^2\right). \tag{1}$$

1.

We write our own R function, `myvar`, to estimate the variance as given above.

2.

Then we generate 10000 normally distributed random numbers, $x = (x_1, ..., x_{10000})$, with mean 10^8 and variance 1.

3.

The variance can be calculated directly in R by using the standard variance estimation function `var()`. In this task we will compute the difference $Y_i =$`myvar`$(X_i) -$`var`(X_i), for each subset $X_i = \{x_1, ..., x_i\},\ i = 1, ..., 10000$.

From Figure 1 we can observe that our implemented function `myvar()` does not work as expected. When subracting the two functions we would obtain zero if they give the same result, but when we look at the plot we can conclude that they perform differently. The reason for why this is happening is because when adding two large numbers of almost equal magnitude of opposite signs, there will be a cancellation. These cancellations can accumulate, therefore we should use a different function when computing the variance.

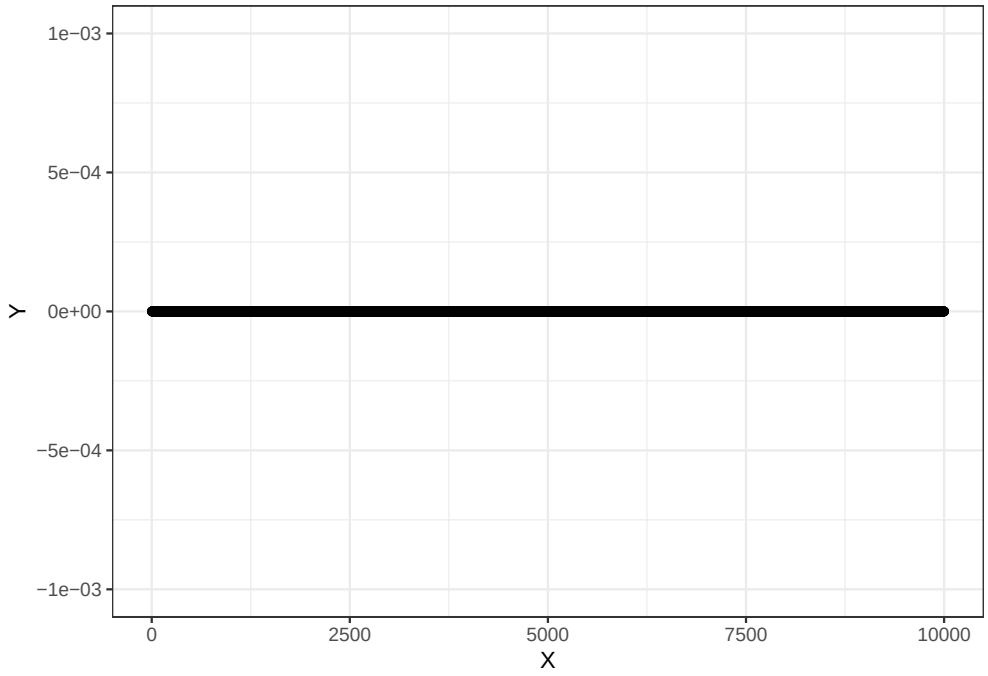

Figure 2: Difference between the functions myvar2(X_i) and var(X_i)

4.

A better way to implement a variance estimator is to use the formula $\text{Var}(\vec{x}) = \frac{\sum_{i=1}^{n}(x_i - \overline{x})^2}{n-1}$, from Definition 1.2 in Mathematical Statistics with Applications by Wackerly et al. We will define this new variance estimator as `myvar2()`.

From Figure 2 we can see that the new implemeted function `myvar2()` is a better variance estimator compared to `myvar()`, because it gives the same results as the function `var()`.

Question 4: Linear algebra

1.

In the last question we will use a data set consisting of a study with the purpose to predict the protein content of 215 meat samples, given the explanatory variables infrared absorbance spectrum, the levels of moisture and fat. Firstly, we import the data set `tecator.csv` to R.

2.

The optimal regression coefficients of a linear regression model can be obtained by solving $\boldsymbol{A}\vec{\beta} = \vec{b}$, where $\boldsymbol{A} = \boldsymbol{X}^T\boldsymbol{X}$ and $\vec{b} = \boldsymbol{X}^T\vec{y}$. The columns in the matrix $\boldsymbol{X}$ are the observations of the absorbance records, levels of moisture and fat, and the response $\vec{y}$ is the protein levels of the meat samples. We compute $\boldsymbol{A}$ and $\vec{b}$ for the data set.

3.

Now we will try to use the function `solve()`, in order to solve $A\vec{\beta} = \vec{b}$.

The `solve()` function returns the error message "Error in solve.default(A, b) : system is computationally singular: reciprocal condition number = 7.13971e-17", which means that matrix $\boldsymbol{A}$ is not invertible. Therefore

the system cannot be solved. We suspect that the columns in $\boldsymbol{A}$ could be linearly dependent or correlated which means that it has less than full rank, and hence not invertible.

4.

The condition number κ of square matrix $\boldsymbol{A}$ is given by $||\boldsymbol{A}||||\boldsymbol{A}^{-1}||$, which gives an indication of how sensitive numerical results are to small errors in the input. Even though $\boldsymbol{A}$ is not invertible, the `kappa()` function still can compute the condition number. This is because it uses psuedo-inverse when inverting is not possible.

The condition number of $\boldsymbol{A}$ is approximately $1.1578 \cdot 10^{15}$. This is a bad sign since the value is very large. A large condition number indicates that the algorithm is numerically unstable, which means that small changes in the inputs results in large changes in the output. However, this does not necessarily lead to ill-conditioning. The problem of solving the system as mentioned in Question 4.3 is due to this observation of a large condition number.

5.

In the final question we will scale the data set and repeat steps 2-4.

When we scaled the data we obtained a smaller condition number, 490471520662, and solving the system with `solve()` is working since the $\boldsymbol{A}$ is now invertible. This can be explained by the fact that the columns of the original $\boldsymbol{A}$ have very different scale, according to Computational Statistics by James E. Gentle. The condition number can be changed when scaling the data, which explains the change in the result.

Appendix

```
knitr::opts_chunk$set(echo = TRUE)
RNGversion('3.5.1')
#1.1
# First code snippet
x1 <- 1/3
x2 <- 1/4
if (x1-x2 == 1/12) {
  print("Subtraction is correct" )
} else {
  print("Subtraction is wrong")
}

# Second code snippet.
x1 <- 1
x2 <- 1/2
if (x1-x2 == 1/2) {
  print("Subtraction is correct")
} else {
  print("Subtraction is wrong")
}
#1.2
# Improvement using all.equal() and isTRUE()
x1 <- 1/3
x2 <- 1/4
if (isTRUE(all.equal(x1-x2, 1/12))) {
  print ("Subtraction is correct" )
} else {
  print ("Subtraction is wrong")
```

```r
}
#2.1
# Function f(x)
f <- function(x) x

epsilon <- 1e-15

# Definition of derivative
derivative <- function(f, x, e){
  (f(x + e) - f(x))/e
}


#2.2 & 2.3
derivative(f, 1, epsilon)
derivative(f,100000, epsilon)
#3.1

# Variance function
myvar <- function(x) {
  n <-length(x)
  1/(n-1) * (sum(x^2) - sum(x)^2/n)
}
#3.2

# Seed
set.seed(1234567890)

# Generate random numbers
x <- rnorm(10000, mean = 10^8 , sd = 1)
#3.3

# Packages
library(ggplot2) # For plotting

# Difference between the different functions of the variance, for each subset
Y <- c()
for (i in seq_along(x)){
  Y[i] <- myvar(x[1:i]) - var(x[1:i])
}

# Create data frame
df <- data.frame(i = seq_along(x), Y)
# Plot the results
ggplot(df, aes(x = i, y = Y)) +
  geom_point() +
  theme_bw() +
  xlab("X")
# 3.4
# Second implementation of the variance
myvar2 <- function(x) {
  n <- length(x)
  avg <- mean(x)
```

```r
  s <- 0

  for (j in 1:n) {
    s <- s + (x[j] - avg)^2
  }

  return(s / (n - 1))
}

# Difference between the different funtions for the variance, for each subset
Y2 <- c()
for (i in seq_along(x)){
  Y2[i] <- myvar2(x[1:i]) - var(x[1:i])
}


# Create data frame
df2 <- data.frame(i = seq_along(x), Y2)

# Plot the results
ggplot(df2, aes(x = i, y = Y2)) +
  geom_point() +
  theme_bw() +
  xlab("X") +
  ylab("Y") +
  ylim(c(-0.001,0.001))
#4.1
# Import data
data <- read.csv(file = "tecator.csv")
#4.2
library(tidyverse)

y <- data$Protein
X <- data %>%
  select(-c(Sample, Protein)) %>%
  as.matrix()

# Computing A and b by matrix multiplication
A <- t(X)%*%X
b <- t(X)*y
#4.3
# Solving the linear system
#solve(A,b)

#4.4
# Check the condition numbers
kappa(A)
#4.5
# Improve the results by scaling
X <- data %>%
  select(-c(Sample, Protein)) %>%
  as.matrix() %>%
  scale()
```

```r
y <- as.vector(scale(data$Protein))

# Computing A and b by matrix multiplication
A <- t(X)%*%X
b <- t(X)*y

# Solving the linear system
solve(A,b)

# Check the condition numbers
kappa(A)
```