Computational Statistics: Computer lab 1.

Laura Julià Melis

1/31/2020

Question 1: Be careful when comparing

1. Check the results of the snippets. Comment what is going on.

From the following pice of code we can observe that the result is:

```
x1 <- 1/3
x2 <- 1/4

if(x1-x2 == 1/12){
  print("Subtraction is correct")
}else{
  print("Subtraction is wrong")
}
```

```
## [1] "Subtraction is wrong"
```

But $x_1 - x_2 = 1/3 - 1/4 = 0.08333333$ and also $1/12 = 0.08333333$. So, there is a problem with this because the condition in the if-clause should be true and hence, the printed output should say "Subtraction is correct".

The result of the second snippet is:

```
x1 <- 1
x2 <- 1/2

if(x1-x2 == 1/2){
  print("Subtraction is correct")
}else{
  print("Subtraction is wrong")
}
```

```
## [1] "Subtraction is correct"
```

Which is actually correct because $x_1 - x_2 = 1 - 1/2 = 0.5$ and $1/2 = 0.5$.

2. If there are any problems, suggest improvements.

As commented in subquestion 1, there is a problem with the first snippet. The condition is FALSE instead of TRUE probably due to the fact that the value x_1 has a repeting decimal.

An improvement could be rounding the numbers at a certain number of decimals. Permorming this improvement, we get the correct result:

```
## [1] "Subtraction is correct"
```

See appendix to see an example of this solution to the problem

Question 2: Derivative

From the defintion of a derivative a popular way of computing it at a point x is to use a small ϵ and the formula

$$f'(x) = \frac{f(x+\epsilon) - f(x)}{\epsilon}$$

1. Write your own R function to calculate the derivative of f(x) = x in this way with $\epsilon = 10^{-15}$.

See appendix to see the code

2. Evaluate your derivative function at x = 1 and x = 100000.

- Derivative at $x = 1$:

```
## [1] 1.110223
```

- Derivative at $x = 100000$.:

```
## [1] 0
```

3. What values did you obtain? What are the true values? Explain the reasons behind the discovered differences.

The values obtained for $x = 1$ and $x = 100000$ are 1.110223 an 0, respectively. But the true values should be 0 for both x (rule: derivative of a constant is 0).

For $x = 10000$, R is storing the value in x as $1e^{+05}$ and when computing $(x + \epsilon)$, given that x is a big number and ϵ is quite small, R returns that the sum is x. So then, $(x + \epsilon) - x$ is 0 and of course, the final result is 0. So the problem is that R is not capable of represent (or store) the exact number of the sum. As said in the Lecture 1 slides, there is loss of significant digits.

On the other hand, when $x = 1$, R evaluates $(x + \epsilon)$ as 1 although the decimals are stored because the number is not too big for R to not be represented (here there aren't lossof significant digits). So, when computing the difference $(x + \epsilon) - x$ the result is not 0, but $(1 + 0.000000000000001) - 1 = 1.110223e - 15$. Then, the final result for the derivative is not 0.

Question 3: Variance

A known formula for estimating the variance based on a vector of n observations is

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

1. Write your own R function, `myvar`, to estimate the variance in this way.

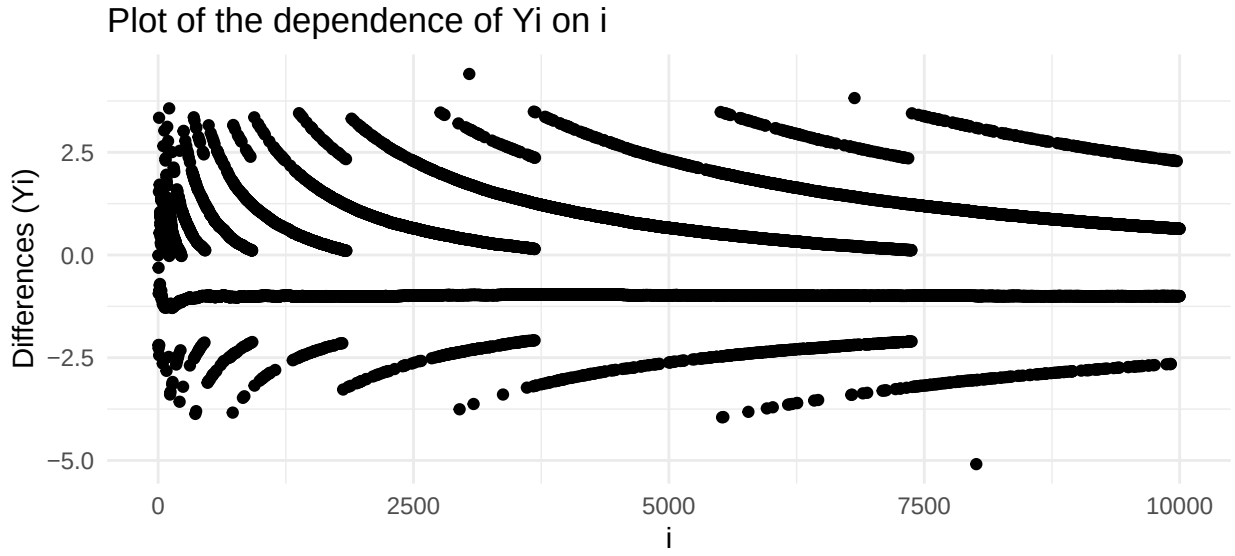
See appendix to see the code

2. Generate a vector $x = (x_1, \ldots, _x 10000)$ with 10000 random numbers with mean 10^8 and variance 1.

The vector x has been generated using the R function `rnorm()`. See appendix to see the code.

3. For each subset $X_i = \{x_1, \ldots, x_i\}, i = 1, \ldots, 10000$ compute the difference $Y_i = myvar(X_i) - var(X_i)$, where $var(X_i)$ is the standard variance estimation function in R. Plot the dependence Y_i on i. Draw conclusions from this plot. How well does your function work? Can you explain the behaviour?

- Plot:



In general, the function `myvar()` estimates the variance with an error of more or less 4 units. We can see this from the plot above: most of the points are between $+4$ and -4.

This makes me think that `myvar()` is not estimating really well the variance. COMPLETE

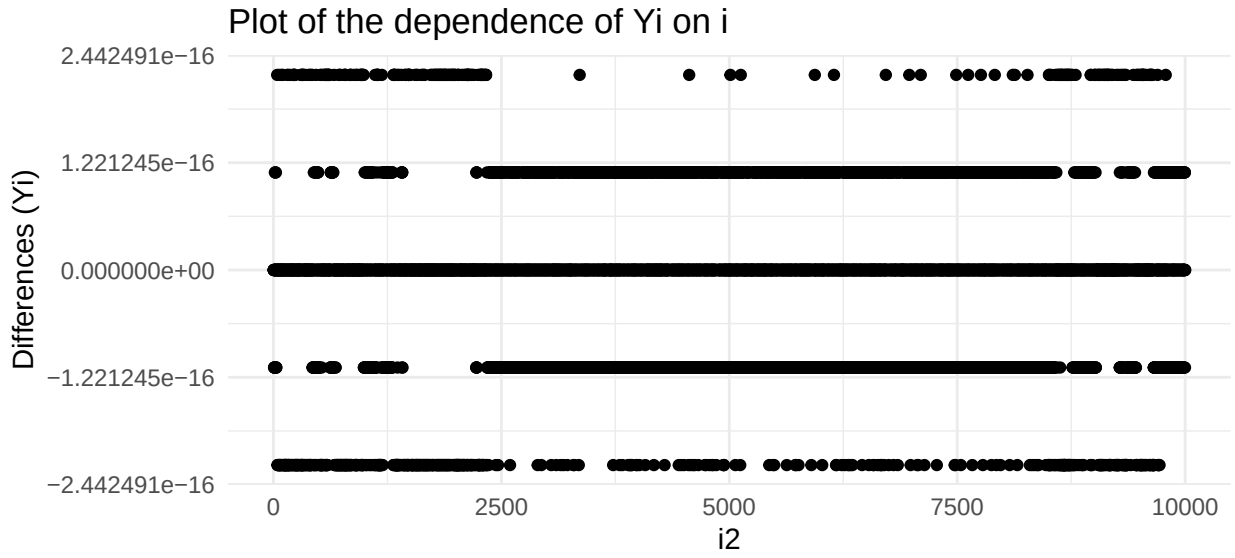
4. How can you better implement a variance estimator? Find and implement a formula that will give the same results as var()?

A better variance estimator has the following formula:

$$\widehat{Var}(\vec{x}) = \frac{1}{n-1} \cdot \sum_{i=1}^{n} (x_i - \bar{x})^2$$

The denominator $n-1$ is used because as it is said in the `var()` documentation, it gives an unbiased estimator of the variance for i.i.d. observations.

So if we plot the new values for Y_i versus i, we obtain:



Now we can observe that many estimations of the variances are equal to the variances calculated with `var()` (a lot of points are in 0). Anyway, the values that are different to 0 are really close to 0 (between $\pm 2.44e^{-16}$) so we can consider that this implementation is better ang in general gives the same results as `var()`.

Question 4: Linear Algebra

The Excel file "tecator.xls" contains the results of a study aimed to investigate whether a near- infrared absorbance spectrum and the levels of moisture and fat can be used to predict the protein content of samples of meat. For each meat sample the data consists of a 100 channel spectrum of absorbance records and the levels of moisture (water), fat and protein. The absorbance is -log10 of the transmittance measured by the spectrometer. The moisture, fat and protein are determined by analytic chemistry. The worksheet you need to use is "data" (or file "tecator.csv"). It contains data from 215 samples of finely chopped meat. The aim is to fit a linear regression model that could predict protein content as function of all other variables.

1. Import the data set to R

See appendix to see the code

2. Optimal regression coefficients can be found by solving a system of the type $\boldsymbol{A}\vec{\beta} = \vec{b}$ where $\boldsymbol{A} = \boldsymbol{X}^T\boldsymbol{X}$ and $\vec{b} = \boldsymbol{X}^T\vec{y}$. Compute $\boldsymbol{A}$ and $\vec{b}$ for the given data set. The matrix $\boldsymbol{X}$ are the observations of the absorbance records, levels of moisture and fat, while $\vec{y}$ are the protein levels.

See appendix to see the code

3. Try to solve $\boldsymbol{A}\vec{\beta} = \vec{b}$ with default solver `solve()`. What kind of result did you get? How can this result be explained?

When we use `solve()` we obtain the following error output:

Error in solve.default(A, b) : system is computationally singular: reciprocal condition number = 7.13971e-17

We are getting this error, probably because the A matrix might not be invertible. Also, there might be linearly dependence among some columns in the A matrix. [1]

Let's check it:

- Determinant of (A):

```
## [1] 0
```

The inverse of a matrix A is calculated as follows:

$$A^{-1} = \frac{1}{det(A)}(\text{Adj}(A))^T$$

Then, given that $A_{102\text{x}102}$ is a square matrix and that its determinant is det(A) = 0, we can conclude that A is singular (not invertible).

- Correlation matrix for the first 7 features:

```
##          Channel1 Channel2 Channel3 Channel4 Channel5 Channel6 Channel7
## Channel1        1        1        1        1        1        1        1
## Channel2        1        1        1        1        1        1        1
## Channel3        1        1        1        1        1        1        1
## Channel4        1        1        1        1        1        1        1
## Channel5        1        1        1        1        1        1        1
## Channel6        1        1        1        1        1        1        1
## Channel7        1        1        1        1        1        1        1
```

We observe that all these features are 100% correlated which means that they are linearly dependent. If all the columns of the A matrix are not linearly independent, then A is not invertible.

[1] Source: https://stats.stackexchange.com/questions/76488/error-system-is-computationally-singular-when-running-a-glm).

4. Check the condition number of the matrix $\boldsymbol{A}$ (function `kappa()`) and consider how it is related to your conclusion in step 3.

From the web page https://en.wikipedia.org/wiki/Condition_number#Matrices:

"If the condition number is not too much larger than one, the matrix is well-conditioned, which means that its inverse can be computed with good accuracy. If the condition number is very large, then the matrix is said to be ill-conditioned".

And the condition number of A, calculated using `kappa()`, is:

```
## [1] 1.157834e+15
```

The condition number of the square matrix A is very high, so this means A is singular and we confirm that, as we said in step 3, is not invertible so this is the reason why `solve(A, b)` can't be performed.

5. Scale the data set and repeat steps 2-4. How has the result changed and why?

We have scaled the data set using the `scale()` function. Then we have computed $\boldsymbol{A} = \boldsymbol{X}^T\boldsymbol{X}$ and $\vec{b} = \boldsymbol{X}^T\vec{y}$.

- Solving $\boldsymbol{A}\vec{\beta} = \vec{b}$ with `solve()`:

We are not receiving an error in this case. The values, $\vec{\beta}$, for the first 10 features are:

```
##                 [,1]
## Channel1  -110.64201
## Channel2  -221.18999
## Channel3   378.00670
## Channel4  -129.70383
## Channel5   413.41803
## Channel6   -79.75199
## Channel7  -203.00600
## Channel8    82.79496
## Channel9  -132.38108
## Channel10  255.82331
```

- Condition number of the scaled matrix A:

```
## [1] 490471520662
```

Now the condition number is much smaller than before although stills being much greater than 1. Also, if we check the determinant of the scaled matrix, its value is 0. But if when performing the inverse matrix, it actually exist.

Maybe the reason why this happens is because there is a loss of significant decimals and although R represents det(A) as 0, it might be not exaclty 0.

Appendix

Question 1

```r
# 1.2. Possible improvement: rounding the elements in the equality inside the if-clause:
x1 <- 1/3
x2 <- 1/4

if(round(x1-x2, 10) == round(1/12,10)){
  print("Subtraction is correct")
}else{
  print("Subtraction is wrong")
}
```

Question 2

```r
# 2.1. Function to calculate the derivative of f(x)=x
derivative <- function(x, epsilon){
  stopifnot(class(x)=="numeric", class(epsilon)=="numeric")

  result <- ((x+epsilon)-x)/epsilon
  return(result)
}

# 2.2. Evaluating the derivative function.
# At x = 1.
epsilon <- 10^(-15)
derivative(1, epsilon)

# At x = 100000.
derivative(100000., epsilon)
```

Question 3

```r
# 3.1. Creating myvar function.
myvar <- function(x){
  stopifnot(class(x)=="numeric")

  n <- length(x)
  result <- ( 1/(n-1) ) * ( sum(x^2) - ( (1/n) * (sum(x)^2)) )
  return(result)
}

# 3.2. Generating random numbers.
set.seed(12345)
x <- rnorm(n= 10000, mean=10^8, sd=1)

## 3.3.a. Computing the differences
y <- vector(length = length(x))
for(i in 1: length(x)){
  x_i <- x[1:i]
  y[i] <- myvar(x[1:i]) - var(x[1:i])
}

## 3.3.b. Plotting Yi vs i.
library(ggplot2)
```

```r
df <- data.frame(i=1:10000, y = y)
df <- df[-1,] # removing first row because (1/n-1) when n=1 is INF.

ggplot(df, aes(x = i, y = y)) + geom_point() +
  ggtitle("Plot of the dependence of Yi on i") +
  ylab("Differences (Yi)") + theme_minimal()

## 3.4.a. Implementing the new variance estimatior function myvar2
myvar2 <- function(x){
  stopifnot(class(x)=="numeric")

  n <- length(x)
  result <- ( 1/(n-1) ) * ( sum( (x - mean(x))^2 ))
  return(result)
}

## 3.4.b. Computing the differences
y2 <- vector(length = length(x))

for(i in 1: length(x)){
  x_i <- x[1:i]
  y2[i] <- myvar2(x[1:i]) - var(x[1:i])
}

## 3.4.c. Plotting Yi vs i.
df <- data.frame(i2=1:10000, y2 = y2)
df <- df[-1,] # removing first row because (1/n-1) when n=1 is INF.

ggplot(df, aes(x = i2, y = y2)) + geom_point() +
  ggtitle("Plot of the dependence of Yi on i") +
  ylab("Differences (Yi)") + theme_minimal()
```

Question 4

```r
# 4.1. Importing the dataset:
data <- read.table("tecator.csv", header= TRUE, sep=",")

# 4.2. Computations:
X <- as.matrix(data[,-c(1,103)])
y <- data$Protein

A <- t(X)%*%X
b <- t(X)%*%y

## 4.3.a. Solving the system:
solve(A, b)

## 4.3.b. Trying to explain the error in solve()
det(A)
cor(A)[1:7,1:7]

# 4.4. Condition number of A
kappa(A)
```

```r
## 4.5.a. Scaling the data
data2 <- scale(data)
X2 <- as.matrix(data2[,-c(1,103)])
y2 <- data2[,103]

## 4.5.b. Computations:
A2 <- t(X2)%*%X2
b2 <- t(X2)%*%y2

betas <- solve(A2, b2)

## 4.5.c. Condition number of scaled A:
kappa(A2)

## 4.5.d. Determinant and inverse matrix
det(A2)
solve(A2)
```