Computer Lab 1 - Computational Statistics

Phillip Hoelscher

23 1 2019

Contents

Question 1: Be careful when comparing

Consider the following two R code snippets

```r
# snippets 1
x1 = 1/3 ; x2 = 1/4
if(x1 - x2 == 1/12){
print("Subtraction is correct")
} else{
print("Subtraction is wrong")
}
```

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

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

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

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

We can see that the output of the two snippets is different.

- 1- Snippet: x_1 is stored with the value $1/3$, which is a consecutive number in decimal form (0.33333…). Especially floating point number are not stored with the "correct" value, but one that is very similar to the "real" value. This is due to the memory capacity of numbers in 32 or 64 bits. Thus information is lost due to regression, which leads to a rounding error. This is also called underflow.
- 2 - snippet: In this case the output shows that the calculation is correct. In this case, no rounding error occurred due to memory optimization.

2. If there are any problems, suggest improvements.

```r
# snippets 1 - improved
x1 = 1/3 ; x2 = 1/4
if(isTRUE(all.equal(x1 - x2, 1/12))){
  print("Subtraction is correct")
} else{
  print("Subtraction is wrong")
}
```

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

One way to fix the output error of snippet 1 is to use the function *all.equal*. This function tests if two objects are "near equality". We can now see from the output that the calculation is correct.

Question 2: Derivative

From the defintion of a derivative a popular way of computing it at a point x is to use a small ?? 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 $\epsilon = 10^15$

The function:

```r
epsilon = 10**-15
derivate = function(x){
  deriv = ((x + epsilon) - x) / epsilon
  return(deriv)
}
```

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

```
x1= 1
x2= 100000
derivate(x1)
```

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

```
derivate(x2)
```

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

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

Answer: Case : x1 = 1
Overflow

Case: x2 = 1000
Underflow, we recognize the dominator "x + epsilon - x" create already the value of 0. The the value of epsilon (10**-15) is to small, the whole digit does not get stored.

Question 3: Variance

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

$$Var(\overrightarrow{x}) = \frac{1}{n-1}(\sum_{i=i}^{n} x_i^2 - \frac{1}{n}(\sum_{i=1}^{n} x_i)^2)$$

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

```
# write the preseted var function - input vector x
myvar = function(x){
  n = length(x)
  var_calculation = (1/(n-1)) * (sum(x^2)-(1/n)*(sum(x)^2))
  return(var_calculation)
}
```

2. Generate a vector x = (x1, . . . , x10000) with 10000 random numbers

with mean 10^8 and variance 1.

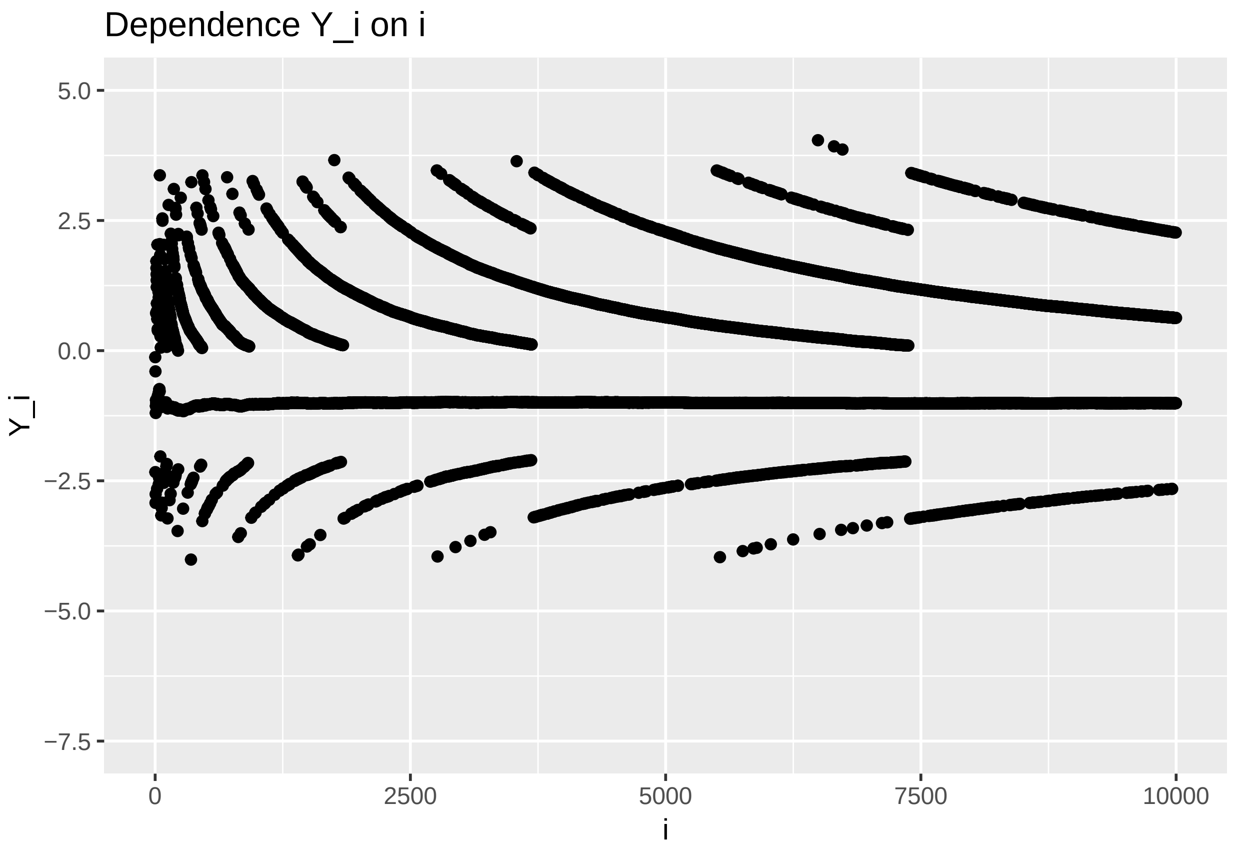
```
# initialize vector
n = 10000
x_vector = rnorm(n, mean = 10^8, sd = 1)
```

3. Compute the difference Yi = myvar(Xi)-var(Xi)

For each subset Xi ={x1,...,xi}, i=1,...,10000 compute the difference $Y_i = myvar(Xi) - var(Xi)$, 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?

```r
# try - 2
y = c()
for (i in 1:length(x_vector)) {
  y[i] = myvar(x_vector[1:i]) - var(x_vector[1:i])
}
# remove the first value - variance of one pont can??t be calculated
y = y[-1]
```

```r
plot_dependence_Y_i
```



At the y-axis we can recognize the Y_i, which can be calculated as follows: Yi = myvar(Xi)- var(Xi). On the x-axis there is the subset with the respective size i. If the functions would output the same output, then all points would be distributed on the 0 horizontal axis. However, we can't see this course. At the y_i value of about -1 there is a continuous line. In the upper part of the visualization it can be seen that the value runs recurrently towards 0. Similar approximate progression can also be seen in the lower part, but against the previously mentioned value of about -1. Since the initialize vector is a very small number, the calculation with the *var()* function may handle these values better than the *myvar* function.

4. Better implementation of variance estimator

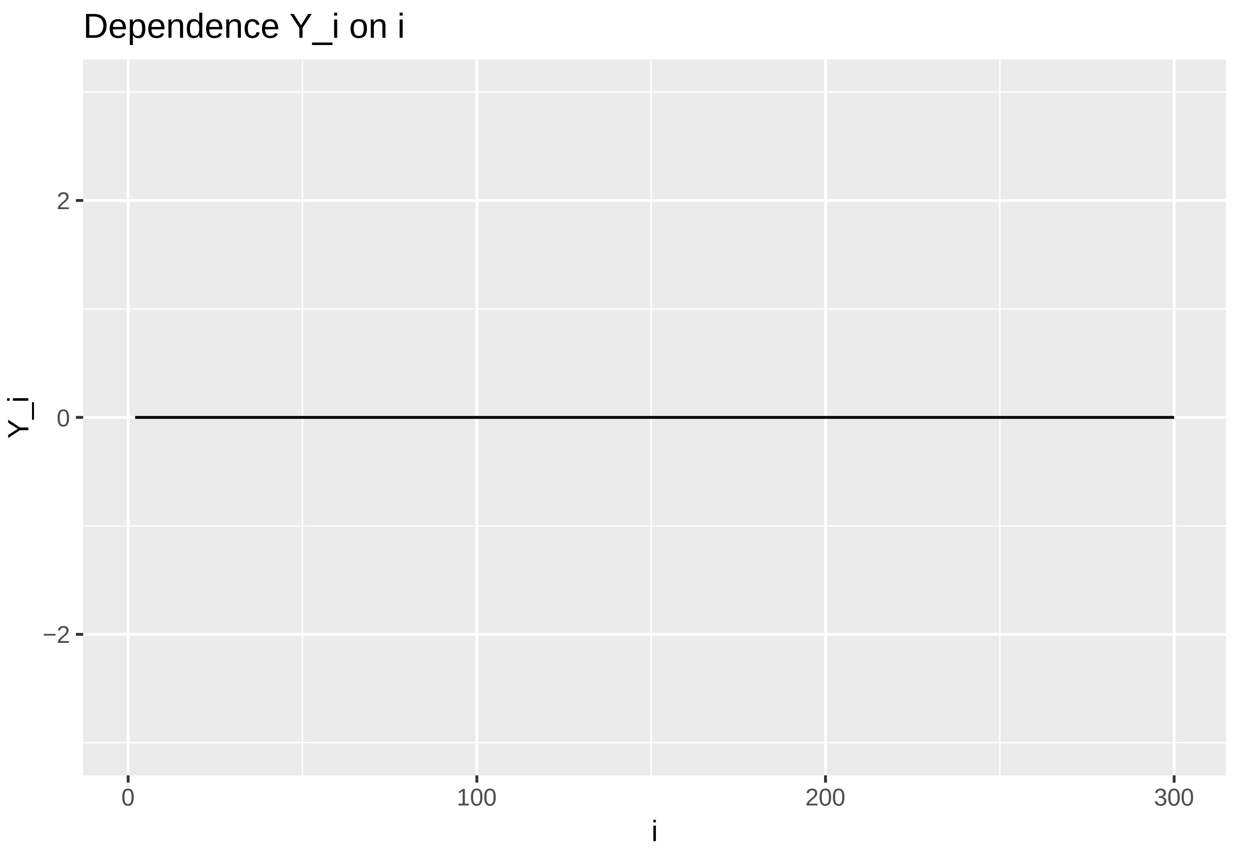
How can you better implement a variance estimator? Find and implement a formula that will give the same results as var()?

```r
myvar_improved <- function(x){
  m <- mean(x)
  sum((x-m)**2)/(length(x)-1)
}
```

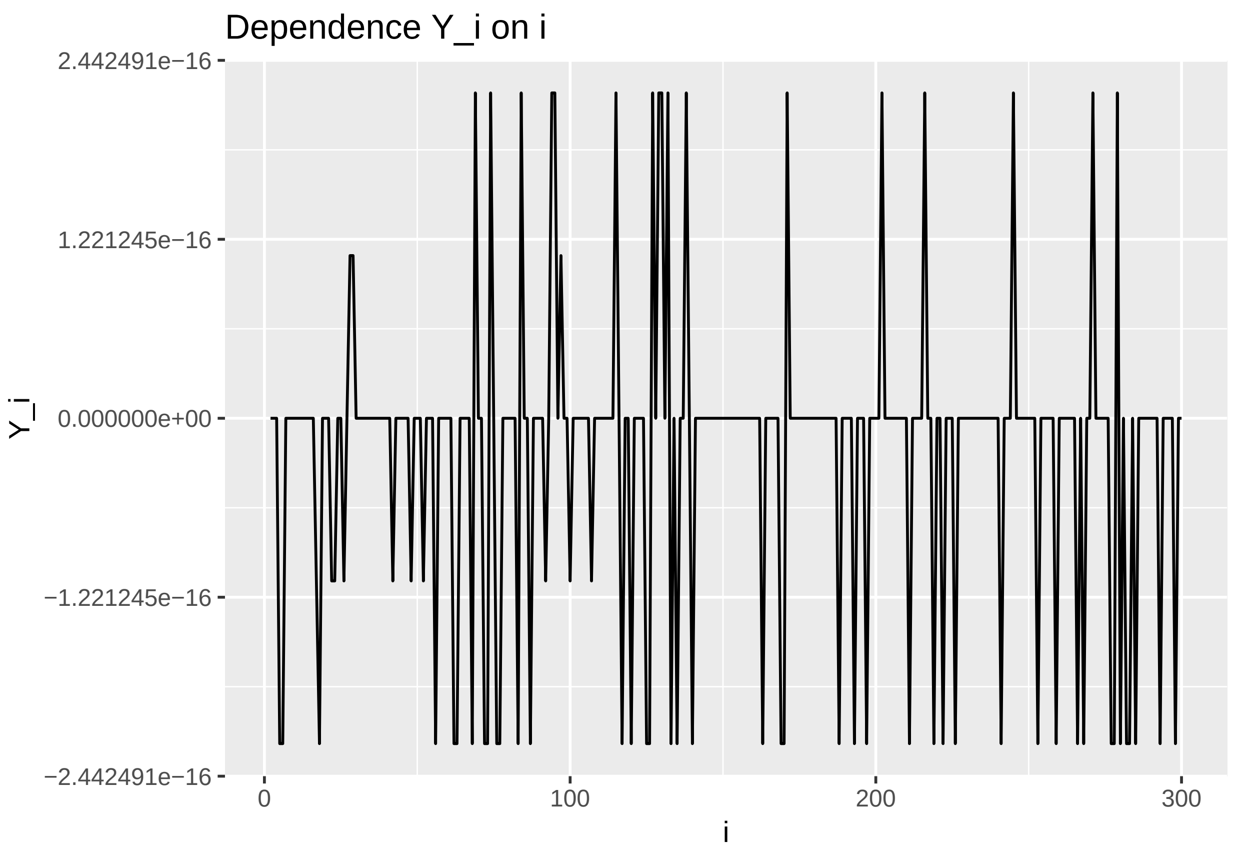
```r
y_improved = c()
for (i in 1:length(x_vector)) {
  y_improved[i] = myvar_improved(x_vector[1:i]) - var(x_vector[1:i])
}

y_improved = y_improved[-1]
```

```r
data_dependence_Y_i_improved = data.frame(y_value = y_improved,
                                iteration = 1:length(y_improved))

plot_dependence_Y_i_2to300_improved_scale3 =
  ggplot(data = data_dependence_Y_i_improved[2:300,]) +
  geom_line(aes(x = iteration, y = y_value)) +
  ggtitle("Dependence Y_i on i") + ylab("Y_i") + xlab("i") +
  ylim(-3,3)

plot_dependence_Y_i_2to300_improved =
  ggplot(data = data_dependence_Y_i_improved[2:300,]) +
  geom_line(aes(x = iteration, y = y_value)) +
  ggtitle("Dependence Y_i on i") + ylab("Y_i") + xlab("i")
```

Improved plot with scale from 3 to -3:



Improved plot :



$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \bar{X})^2$$

Here we use the sample variance formula, which is an unbias estimate of real variance for a finite sequence. Evidently, the result is better and the differnces between estiamted and real variances are much smaller than the previous formula.

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

```r
# set working directory
data = read.csv("tecator.csv")
# remove the column sample
data = data[-1]
```

2. Compute optimal regression coefficients

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

```
# create X and y
# get the index of protein (y)
index_protein = which(colnames(data)== "Protein")
# remove the column names
colnames(data) = NULL
# remove protein from the data & define as matrix X
X = as.matrix(data[-index_protein])

# create y vector
y = as.matrix(data[index_protein])

# compute A
A = t(X) %*% X

# compute b
b_vector = t(X) %*% y
```

3. Solve with default solver *solver()*

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

```
solve(A, b_vector)
#This function creats an error
```

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

We can see the output of the default solver above, because of computationally singularity on the operation does the calcualtion not work.

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

```
kappa(A)
```

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

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

```
# scale the data
data_scale = scale(data)
# create X and y
# get the index of protein (y)

# remove protein from the data & define as matrix X
```

```r
X_scale = as.matrix(data_scale[,-index_protein])

# create y vector
y_scale = as.matrix(data_scale[,index_protein])

# compute A
A_scale = t(X_scale) %*% X_scale

# compute b
b_vector_scale = t(X_scale) %*% y_scale
```

```r
solve(A_scale, b_vector_scale)
```

```r
kappa(A_scale)
```

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

As mentioned before, the function cannot perform the operation without scaled data. The output tells us that the tolerance for the solver in default is not small enough. This error could be corrected manually by changing the tolerance, or as it was done in this part, by scaleing of the data. Now we have a result for the *solver* and a result for the *kappa*.

Appendix

```r
knitr::opts_chunk$set(echo = TRUE, eval = TRUE)
# the lirbraries used in this lab
library(ggplot2)
# snippets 1
x1 = 1/3 ; x2 = 1/4
if(x1 - x2 == 1/12){
print("Subtraction is correct")
} else{
print("Subtraction is wrong")
}
# snippets 2
x1 = 1 ; x2 = 1/2
if(x1 - x2 == 1/2){
print("Subtraction is correct")
} else{
print("Subtraction is wrong")
}
# snippets 1 - improved
x1 = 1/3 ; x2 = 1/4
if(isTRUE(all.equal(x1 - x2, 1/12))){
  print("Subtraction is correct")
} else{
  print("Subtraction is wrong")
}
epsilon = 10**-15
derivate = function(x){
  deriv = ((x + epsilon) - x) / epsilon
  return(deriv)
}
x1= 1
x2= 100000
derivate(x1)
derivate(x2)
# write the preseted var function - input vector x
myvar = function(x){
  n = length(x)
  var_calculation = (1/(n-1)) * (sum(x^2)-(1/n)*(sum(x)^2))
  return(var_calculation)
}
# initialize vector
n = 10000
x_vector = rnorm(n, mean = 10^8, sd = 1)
# test
#myvar(x_vector,10000)
#var(x_vector)

# initialize result vector
# try - 1
y = c()
for (i in 1:length(x_vector)) {
  y[i] = myvar(x_vector[i]) - var(x_vector[i])
}
```

```
# result - NA in every point -> variance of one point does not make sense
# try - 2
y = c()
for (i in 1:length(x_vector)) {
  y[i] = myvar(x_vector[1:i]) - var(x_vector[1:i])
}
# remove the first value - variance of one pont can??t be calculated
y = y[-1]
# create data framem for the plot
data_dependence_Y_i = data.frame(y_value = y,
                                 iteration = 1:length(y))
# create plot dependence Yi on i
plot_dependence_Y_i = ggplot(data = data_dependence_Y_i) +
  geom_point(aes(x = iteration, y = y_value)) +
  ggtitle("Dependence Y_i on i") + ylab("Y_i") + xlab("i") +
  ylim(-7.5,5)
plot_dependence_Y_i
myvar_improved <- function(x){
  m <- mean(x)
  sum((x-m)**2)/(length(x)-1)
}

y_improved = c()
for (i in 1:length(x_vector)) {
  y_improved[i] = myvar_improved(x_vector[1:i]) - var(x_vector[1:i])
}

y_improved = y_improved[-1]

data_dependence_Y_i_improved = data.frame(y_value = y_improved,
                                 iteration = 1:length(y_improved))

plot_dependence_Y_i_2to300_improved_scale3 =
  ggplot(data = data_dependence_Y_i_improved[2:300,]) +
  geom_line(aes(x = iteration, y = y_value)) +
  ggtitle("Dependence Y_i on i") + ylab("Y_i") + xlab("i") +
  ylim(-3,3)

plot_dependence_Y_i_2to300_improved =
  ggplot(data = data_dependence_Y_i_improved[2:300,]) +
  geom_line(aes(x = iteration, y = y_value)) +
  ggtitle("Dependence Y_i on i") + ylab("Y_i") + xlab("i")
plot_dependence_Y_i_2to300_improved_scale3
plot_dependence_Y_i_2to300_improved
# set working directory
data = read.csv("tecator.csv")
# remove the column sample
data = data[-1]
# create X and y
# get the index of protein (y)
index_protein = which(colnames(data)== "Protein")
# remove the column names
colnames(data) = NULL
```

```r
# remove protein from the data & define as matrix X
X = as.matrix(data[-index_protein])

# create y vector
y = as.matrix(data[index_protein])

# compute A
A = t(X) %*% X

# compute b
b_vector = t(X) %*% y
solve(A, b_vector)
#This function creats an error
kappa(A)
# scale the data
data_scale = scale(data)
# create X and y
# get the index of protein (y)

# remove protein from the data & define as matrix X
X_scale = as.matrix(data_scale[,-index_protein])

# create y vector
y_scale = as.matrix(data_scale[,index_protein])

# compute A
A_scale = t(X_scale) %*% X_scale

# compute b
b_vector_scale = t(X_scale) %*% y_scale
solve(A_scale, b_vector_scale)
kappa(A_scale)
```