Computational Statistics Lab1

Jooyoung Lee, Vasileia Kampouraki, Weng Hang Wong

2020 1 29

Question 1

1-1.

```
# The first given code
x1 <- 1/3 ; x2 <- 1/4
if (x1-x2 == 1/12) {
  print("Subtraction is correct")
} else {
  print("Subtraction is wrong")
}
```

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

```
# The second given code
x1 <- 1 ; x2 <- 1/2
if (x1-x2 == 1/2) {
  print("Subtraction is correct")
} else {
  print("Subtraction is wrong")
}
```

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

Mathematically, both snippets are expected to return the same result, "Subtraction is correct". However, while the second one prints expected result, the first one returns "Subtraction is wrong" instead.

This is related to rounding problem. In binary computer system, none of fractions, except the ones having a number that is the power of 2 as its denominator, has a finite decimals. Rationals with infinite decimals are rounded towards the nearest computer float so that usual mathematical laws do not hold. However, for the second snippet, since `x2 <- 1/2`, which is of finite decimals, the code returns expected mathematical result.

1-2.

The problem related to the first given snippet can be improved as following method:

```
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"
```

The function `all.equal` is used; according to its description in R Documentation, this function is to 'test if two objects are nearly equal'. By using this function, two values, which are rounded up to certain point, can be compared and return the expected result.

Question 2

2-1.

```
derivative <- function(x) {
  eps <- 10^(-15)
  res <- ((x+eps)-x)/eps
  return(res)
}
```

2-2.

```
derivative(1)
```

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

```
derivative(100000)
```

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

2-3.

By running the code, above results are returned. Since the original function is $f(x) = x$, mathematical results of both are expected to be 1. However, neither of them returns 1.

For `derivative(1)`, expected result is not returned because the computer stores data with binary system, using mentissa and exponent. The calculation in the function is done with the numbers stored in binary system, which leads to the answer that is not mathematically expected.

For `derivative(100000)`, expected result is not returned because of underflow. Since $100,000$ is very big number compare to `eps` in the function, while the computer stores the data to calculate `((x+eps)-x)/eps`, where $x = 100,000$, significant digits of `eps` on numerator are lost so that the numerator becomes 0; this results in final output 0.

Question 3

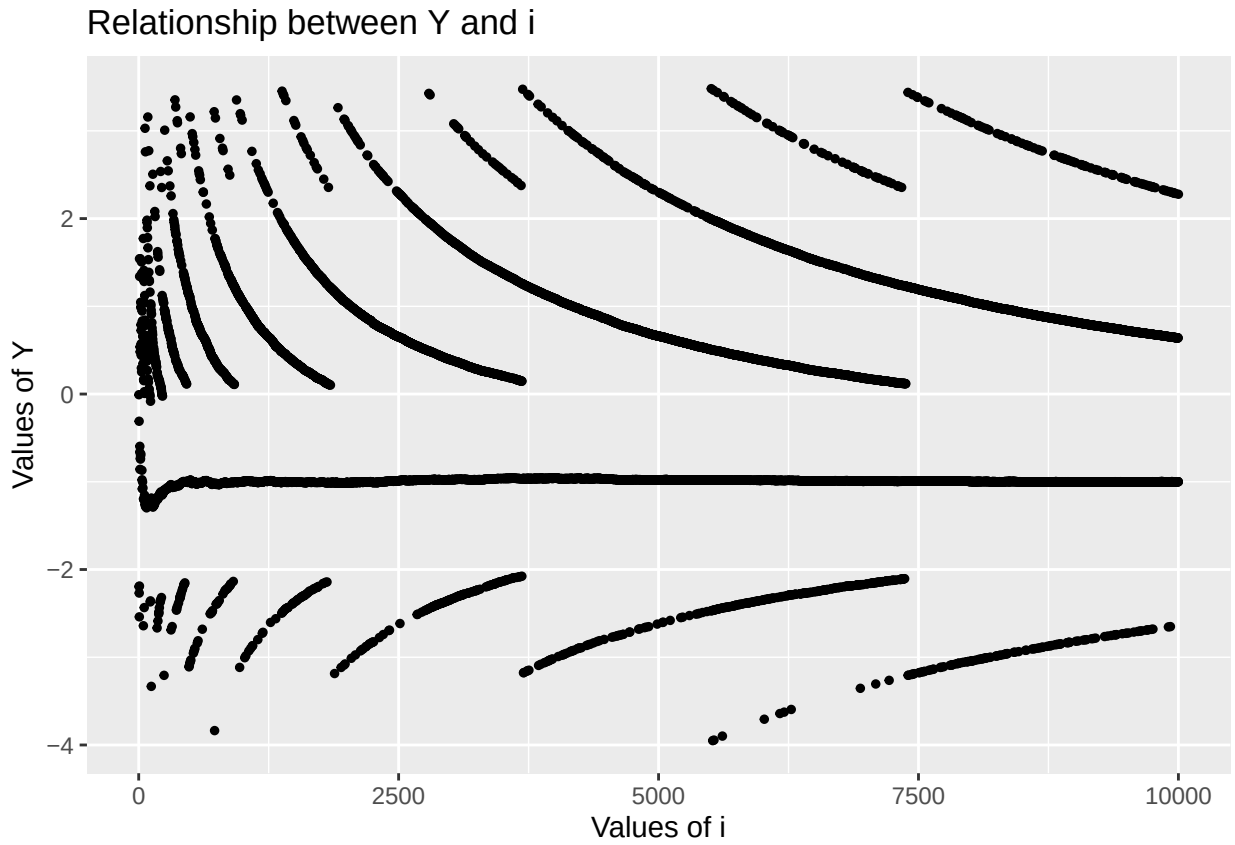
3-1.

```r
myvar <- function(X) {
  n <- length(X)
  res <- (sum(X^2) - ((sum(X)^2)/n))/(n-1)
  return(res)
}
```

3-2.

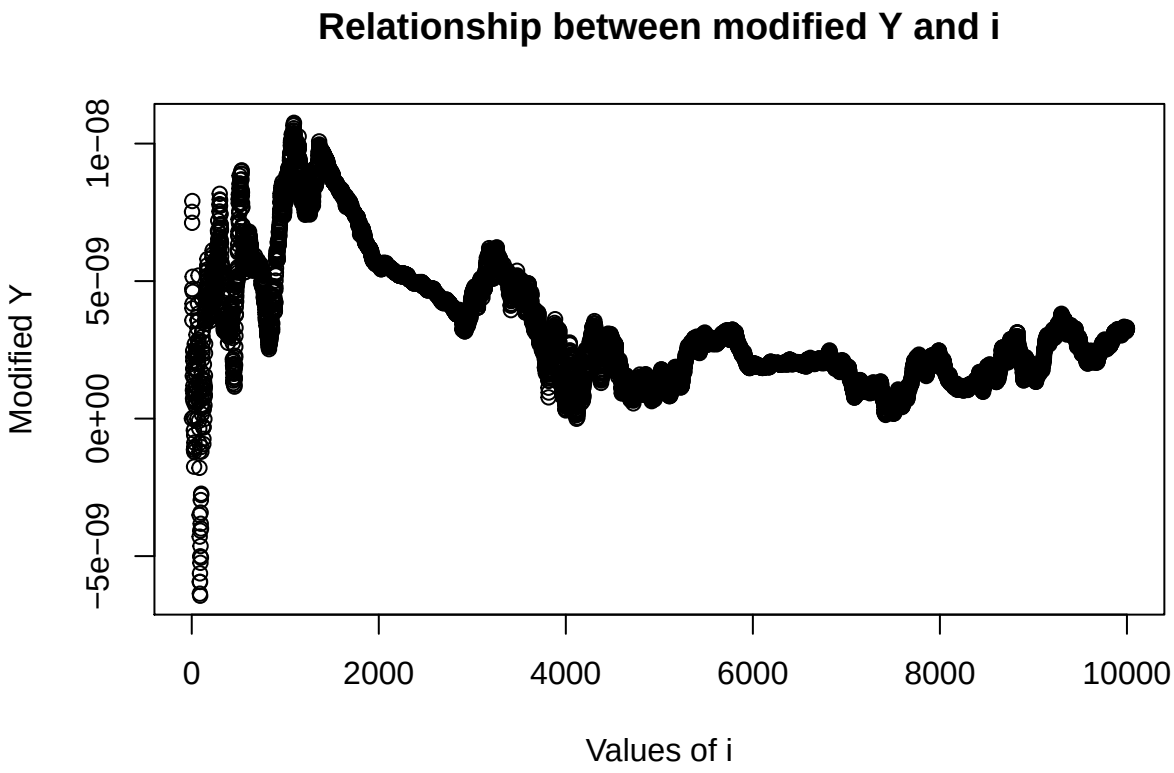
```r
set.seed(12345)
x <- rnorm(10000, mean=10^8, sd=1)
```

3-3.



It is expected that y_i to be 0 for each i as `myvar` for computing the variance theoretically would give the same results as the built-in function `var`. However, the plot shows that the results returned from `myvar` function and `var` function are different. Such differences occur because of the floating point system used in R, where the numbers get rounded and stored in binary system.

3-4.



By using Young-Cramer algorithm, modified variance is manually calculated. It still returns somewhat different value to `var`, but such differences are very close to zero and therefore negligible.

Question 4.

4-1.

```
data <- read.csv("C:/Users/Jooyoung/Documents/Computational Stat/732A90_VT2020_Materials/tecator.csv")
```

4-2.

```
X <- data[-c(1,103)] #observations
X <- as.matrix(X)
y <- data[103]    #target
y <- as.matrix(y)
A <- t(X)%*%X
b <- t(X)%*%y
```

```
## To check if the calculation worked, only parts of A and b are printed as follows.
##
```

```
## First Three Rows and Columns of Computed A:

##          Channel1 Channel2 Channel3
## Channel1 1732.036 1733.816 1735.603
## Channel2 1733.816 1735.600 1737.391
## Channel3 1735.603 1737.391 1739.185

##
##  First Three Rows of Computed b:

##              [,1]
## Channel1 10631.59
## Channel2 10641.63
## Channel3 10651.69
```

4-3.

```
solve(A,b)
```

```
## Error in solve.default(A, b): system is computationally singular: reciprocal condition number = 7.13
```

When `solve` function is used, above error is returned. This is because the matrix A is singular, meaning that some attributes of the data are very likely to be correlated.

4-4.

The function `kappa` returns the condition number of a regular matrix. Condition number represents how much the output value of the function can change for a small change in the input argument; this is used to measure how sensitive a function is to changes or errors in the input. [https://en.wikipedia.org/wiki/Condition_number] κ is calculated by using following function:

$\kappa(A) = \left\|A^{-1}\right\| \|A\|$

According to the result above, condition number of the matrix A is 1.1578342×10^{15}. This indicates the output is massively affected by a small change or error in the matrix A, which means the matrix A is unstable and ill-conditioned that leads to error in using `solve` function. The reciprocal condition number $8.6368149 \times 10^{-16}$, which is very close to zero as well, also explains that the matrix A is ill-conditioned. In this case, huge condition number of A is an indication of multicollinearity.

4-5.

```
## Condition Number of the Scaled Data:

## [1] 490471520662
```

When the data is scaled, `solve` function returns a matrix without any visible error. The condition number of the matrix A is returned to be 4.9047152×10^{11}, which is definitely smaller than the one from unscaled data, explains why the `solve` function worked. With the scaled data, new matrix `A` is not ill-conditioned that the generic function in R could solve the problem with the given inputs. This indicates that multicollinearity can be eliminated by scaling the data.

Appendix

```
knitr::opts_chunk$set(echo = TRUE)
# The first given code
x1 <- 1/3 ; x2 <- 1/4
if (x1-x2 == 1/12) {
  print("Subtraction is correct")
} else {
  print("Subtraction is wrong")
}

# The second given code
x1 <- 1 ; x2 <- 1/2
if (x1-x2 == 1/2) {
  print("Subtraction is correct")
} else {
  print("Subtraction is wrong")
}
x1 <- 1/3 ; x2 <- 1/4
if (isTRUE(all.equal(x1-x2,1/12))) {
  print("Subtraction is correct")
} else {
  print("Subtraction is wrong")
}
derivative <- function(x) {
  eps <- 10^(-15)
  res <- ((x+eps)-x)/eps
  return(res)
}
derivative(1)
derivative(100000)
myvar <- function(X) {
  n <- length(X)
  res <- (sum(X^2) - ((sum(X)^2)/n))/(n-1)
  return(res)
}
set.seed(12345)
x <- rnorm(10000, mean=10^8, sd=1)
y <- c()

for (i in (1:length(x))) {
  ones <- x[1:i]
  res <- myvar(ones) - var(ones)
  y <- append(y, res)
}

y <- y[2:length(y)]
ress <- data.frame(x=2:length(x), y=y)

library(ggplot2)
ggplot(data=ress, mapping=aes(x=x, y=y)) + geom_point(colour="black",
        size=1) + labs(title="Relationship between Y and i",
            x = "Values of i", y="Values of Y")
```

```r
# Young's algorithm

## v_x is a numerical vector of length greater than 2
## this function calculates the sample variance
## using the Youngs and Cramer algorithm
var_YC<-function(v_x){
T<-v_x[1]
RSS<-0
n<-length(v_x)
for (j in 2:n){
  T<-T+v_x[j]
  RSS<-RSS+((j*v_x[j]-T)^2)/(j*(j-1))
}
RSS/(n-1)
}

X=0
for( i in (1:length(x)) ){
  X = x[1:i]
  y[i]=var_YC(X)-var(X)
}

plot(x=(1:length(x)), y=y, xlab="Values of i", ylab="Modified Y", main="Relationship between modified Y

data <- read.csv("C:/Users/Jooyoung/Documents/Computational Stat/732A90_VT2020_Materials/tecator.csv")
X <- data[-c(1,103)] #observations
X <- as.matrix(X)
y <- data[103]    #target
y <- as.matrix(y)
A <- t(X)%*%X
b <- t(X)%*%y
cat("To check if the calculation worked, only parts of A and b are printed as follows.", "\n", "\n")
cat("First Three Rows and Columns of Computed A: ", "\n")
A[1:3, 1:3]
cat("\n","First Three Rows of Computed b: ", "\n")
as.matrix(b[1:3,])
solve(A,b)
kappa(A)
data1 <- as.data.frame(scale(data))
Xnew <- data1[-c(1,103)] #observations
Xnew <- as.matrix(Xnew)
ynew <- data1[103]    #target
ynew <- as.matrix(ynew)
A1 <- t(Xnew)%*%Xnew
b1 <- t(Xnew)%*%ynew
beta1 <- solve(A1)%*%b1
cat("Condition Number of the Scaled Data: ", "\n")
kappa(A1)
```