732A90: Lab 1
Computational Statistics, Group 6

Chih-Yuan Lin, Sarah Walid Alsaadi, Carles Sans Fuentes

February 3, 2017

Assignment 1

The aim of this lab is to

Exercise 1

In this exercise the following snippets must be commented:

```
x1 <-  1/3
x2  <-  1/4
if(x1 - x2 == 1/12){
  print("Substracion is correct")
} else{
  print("Substraction is wrong")}

all.equal((1/3)-(1/4),(1/12))
----------------------------------------
x1<- 1; x2 <- 1/2

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

In the first example, we got ("Substraction is wrong") which is wrong since $x_1 - x_2 = 1/12$ according to the ordinary arithmetics and in the second, we got ("Substraction is correct") which agrees with the usual arithmetics.

The reason why we get different results in the examples is due to the fact that computer numbers are not the same as real numbers. Real numbers are infinite while computer numbers are finite. This force the computer to round numbers like 1/3 to 0.666667 for example to be able to store it in a finite number of bits. The consequence is that we get an error in the calculations. A way to evaluate whether this equality is correct would be by using the function all.equal to test if two objects are nearly equal. The second equality does not have the same problem because $1/2 = 0.5$ without the need to round it.

Exercise 2

In this assignment, we were asked to compute the derivative of $f(x) = x$ at two points, $x = 1$ and $x = 10^5$ using the following formula

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

where $\epsilon = 10^{-15}$.

When $x = 1$, the output is 1.110223 whereas when $x = 10^5$, the output is zero. Nevertheless, the true value for both cases should be 1, since the derivative of $f(x) = x$ is 1.

Computers store numbers in base 2 so the reason why we get 1.110223 and not 1 when $x = 1$ is because of the change from base 10 to base 2. What happens when we change the base is that, a number with a finite decimal expansion in base 10 might no longer have a finite decimal expansion in base 2. In our case, the number 10^-15 in base 2 has to be written as $10^-15 = X2^e \implies -15 = \log X + e(\log 2)$ in the computer where X it the significand and e is the exponent. This number has an infinite decimal expansion in base 2 which results in the same problem as in exercise 1.

In the case where $x = 10^5$ we get a catastrophic cancellation. Since to compute the derivative, we have to first compute $(10^5 + \epsilon) - 10^5$ where ϵ is so small that the number $10^5 + \epsilon$ is rounded to 10^5 in the computer the derivative in that point becomes 0. One way to deal with catastrophic cancellation when adding small numbers and big numbers is to add the small ones first so we get them big enough so they do not get cancelled when added to the big numbers.

Exercise 3

In this exercise, estimation of the variance based on a vector of n observations is done using the formula

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

We generated a vector $x = (x_1, x_2, \ldots, x_n)$ where $n = 10000$ of random numbers from a normal distribution with mean 10^8 and variance 1. We then defined the function *myvar* as the formula above and computed the sample variance. The result we got was 0 which is far from the true variance 1. The reason why we get wrong estimation is due to catastrophic cancellation. Now, for each subset $X_i = (x_1, x_2, \ldots, x_i)$, $i = 1, 2, \ldots, 10000$, we considered the difference between the estimated variance defined as the formula above and the standard variance function built in **R**, that is, $Y_i = myvar(X_i) - var(X_i)$ as a function of i to see how our defined function behave with larger sample compared to the function defined by **R**. The following plot (see figure 1) shows how Y_i changes with i.

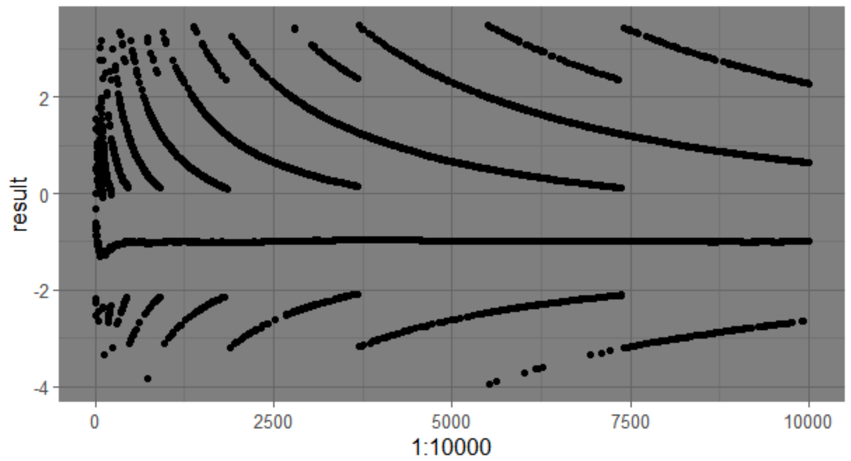

Figure 1: Difference between Yi and i between myvar and the R function var

The function works wrongly in the sense that (1) we got a lot of zero's on our function when the output should be 1. This phenomenon can be observed in the figure 1 by a line at -1 since $(0 - 1 = -1)$, while the result of the subtraction should be 0 since $(1 - 1)$. (2) we got some negative numbers, but variance could never be negative. (3) it shows some periodicity of values. The phenomenon (1) is caused by subtraction of approximately equal numbers. The larger the number is, the less precision it can provide on floating-points. The phenomenon (2) and (3) may be caused by integer overflow, that is, we get a number that is to large to be presented by the storage space.

There are two possible solutions to implement a better variance estimator. The first solution is to centerize each observation by the mean and compute the variance according to the following formula.

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

Figure 2 shows the difference between the output of this function and default var function in **R**. The second solution is Welford's method, which calculates the mean and variance in fly.

$$M_1 = x_1$$

$$S_1 = 0$$

$$M_k = M_{k-1} + \frac{x_k - M_{k-1}}{k}$$

$$Sk = S_{k-1} + (x - M_{k-1}) * (x - M_k)$$

This method requires only one iteration so that we do not have to save all the numbers in the memory. (Though we already have them in memory). Figure 3 illustrates the difference between the output of Welford's method and default var function in **R**, the figure shows that the difference converges to 0 when the number of obervations increases.

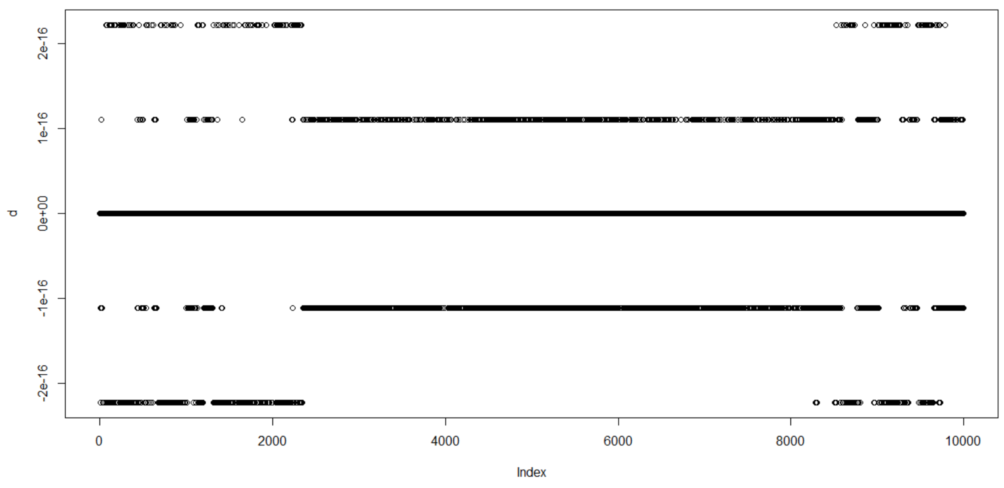

Figure 2: Difference between the formula for variance where each observations is centered and the function var in **R**

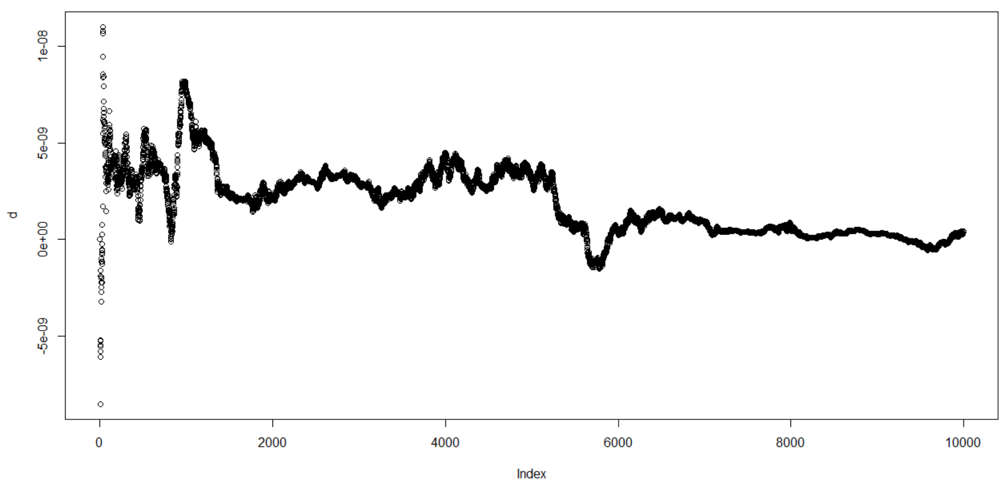

Figure 3: Difference between Welford's method and the function var in **R**

Exercise 4

The datafile **tecator.csv** is used 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 that, a linear regression model that could predict protein content as function of all other variables must be fitted.

To solve $A\beta = b$, **R** has to invert the matrix A. The computation result in the following message: "Error in solve.default(A) : system is computationally singular: reciprocal condition number = 3.29617e-17". So **R** claims that the matrix is computationally singular, note that **R** uses the word computationally which shows that **R** is aware of the fact that the matrix might not be singular, just that enough entries are rounded to zero during computations to make the matrix singular (it could also be that the columns are not linearly independent). Computing the condition number $\kappa(A)$ with respect to inversion we get $\kappa(A) = 490471520662$ which is huge. Getting a κ so large might be due to artificial ill-conditioning which can be solved by scaling the data, see page 207 in the text book. The problem of artificial ill-conditioning in our case is due to the different columns having a very different scale (compare fat, protein and moisture to the other variables). Scaling the data yielded a well-conditioned linear system and we were able to estimate the $\beta : s$.

Contributions

All results and comments presented have been developed and discussed together by the members of the group.

Appendix

Assignment 1

```r
##R script activity 1
###Assignment 1

##1
x1 <-  1/3
x2  <-  1/4
if(x1 - x2 == 1/12){
  print("Substracion is correct")
} else{
  print("Substraction is wrong")}


x1-x2


x1<- 1; x2 <- 1/2

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

##Floats are rounded so usual mathematical laws do not hold floating point arithmetic


##2


der<- function(x, e= 10**-15){

  res<- ((x+e)-x)/e
  return(res)

  }

der(x= 1)
der(x = 100000)

##3


myvar<-function(x){
  res<-(sum(x^2)-(sum(x)^2)/length(x))/(length(x)-1)
  return(res)
}

defvar<- function(x){
  return(sum((x-mean(x))^2)/(length(x)-1))

}

set.seed(12345)
v<-rnorm(10000, 10^8, 1)

d<-integer(0)
d[1]=0

for (i in 2:length(v)){
  #d[i]<-defvar(v[2:i])-var(v[2:i])
  d[i]<-myvar(v[2:i])-var(v[2:i])
}
plot(d)

##calculate in fly
# https://www.johndcook.com/blog/standard_deviation/
# https://www.johndcook.com/blog/2008/09/26/comparing-three-methods-of-computing-standard-
     deviation/
M<-c(v[1])
S<-c(0)
d[1]=0

for(i in 2:length(v)){
  M[i]=M[i-1]+(v[i]-M[i-1])/i
  S[i]=S[i-1]+(v[i]-M[i-1])*(v[i]-M[i])
  d[i]=S[i]/(i-1)-var(v[1:i])
}

plot(d)



```

```r

##4


data<- read.csv("C:/Users/M/Desktop/Statistics and Data Mining Master/Semester 2/Computationa
     Statistics/tecator.csv", sep = ",")
formula <- as.formula(Protein~.)
X <- model.matrix( formula , data= data)
y <-data[,all.vars(formula)[1]]

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


Beta <- solve((t(X)%*%X))%*%t(X)%*%y
#kappa(A)

##scaled data
scaled_data<- as.data.frame(scale(data))

formula <- as.formula(Protein~.)
X <- model.matrix( formula , data= scaled_data)
y <-scaled_data[,all.vars(formula)[1]]

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

Beta <- solve((t(X)%*%X))%*%t(X)%*%y
```