# Lab1

*Martin, Raymond*

*January 23, 2019*

## Question 1: Be careful when comparing

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

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

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

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

```
options(digits=22)
1/3 # we can see that as 1/3 is a periodic number pc has a problem to put it in a memory.
```

```
## [1] 0.33333333333333331
```

```
1/4
```

```
## [1] 0.25
```

```
1
```

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

```r
1/2
```

```
## [1] 0.5
```

We can see that the first subtraction gives a wrong answer and the second one gives use correct answer. The problem here is that 1/3 is a periodic number with infinite number of digits and that is why the computer is not able to store the exact number and stores it as a closest as possible however not exact 1/3. That is why when subtracting we are getting slightly different result.

## 1.2 Suggested Improvements

We have no specific suggestion but we would like to hint on a realistic direction. This direction must involve the use of the modulus operator and testing for existance of remainders in mathematical manipulations.

# Question 2: Derivative

## 2.1 R function to calculate the derivative of f(x) = x

```
myderiv <- function(x){
   ep <- (10**(-15))
   ((x + ep) - x)/ep
}
```

## 2.2 Evaluate derivative function at x = 1 and x = 100000

```
options(digits = 22)
options(scipen = 999)
myderiv(1)
```

```
## [1] 1.1102230246251565
```

```
myderiv(100000)
```

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

## 2.3

At $x = 1$, I got 1.1102230246251565 while at $x = 100000$, I got 0. The true value should be 1.

```
ep <- (10**(-15))
100000+ep
```

```
## [1] 100000
```

```
1+ep
```

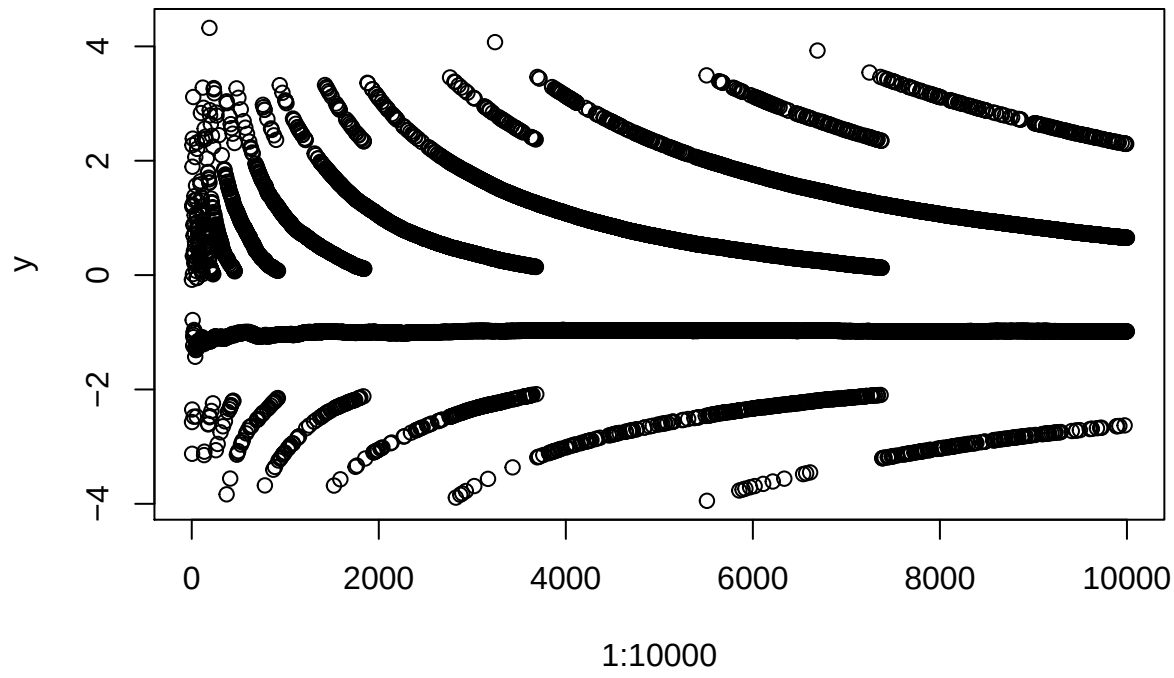
```
## [1] 1.0000000000000011
```

When we add 1 to $\epsilon$ which is 10^-15, the result is a floating point for which all the values can not be stored in memory. When we use options(digits = 22), we see a "one" in the 15th and 16th position. However, when we add the same $\epsilon$ to 100000, the result is 100000. This is because the many extra floating points are disregarded and the closest number which is 100000 is considered. This is a case of underflow.

## Question 3: Variance

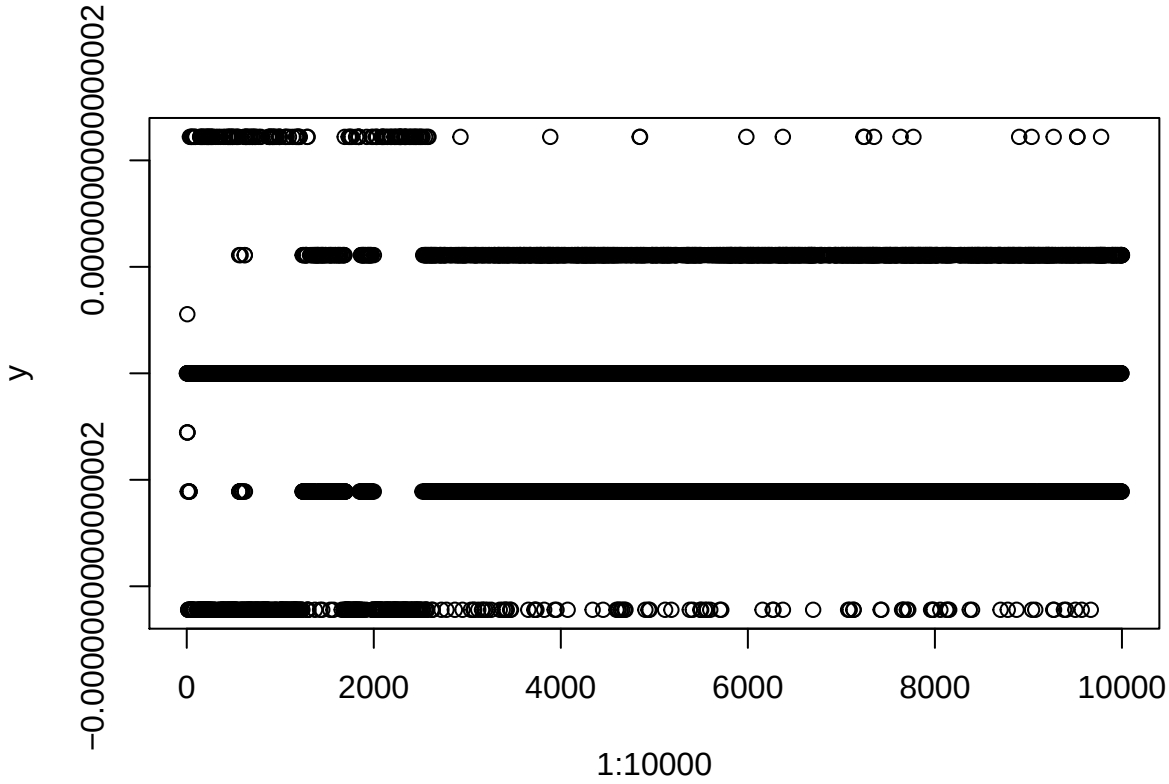
```
myvar <- function(x, n)
{
  variance <- 1/(n-1) * ((sum(x^2)-(1/n)*(sum(x))^2))
  return(variance)
}

x <- rnorm(10000, 10^8, 1)
y <- rep(0, 10000)
for(i in 1:10000)
{
  y[i] <- myvar(x[1:i], i) - var(x[1:i])
}
plot(x=1:10000, y=y)
```




```
newvar <- function(x, n)
{
  K = mean(x)
  variance <- (1/(n-1)) * (sum((x-K)^2)-(1/n)*(sum(x-K))^2)
  return(variance)
}
for(i in 1:10000)
{
```

```
  y[i] <- newvar(x[1:i], i) - var(x[1:i])
}
plot(x=1:10000, y=y)
```




We computed the first graph according to the formula and we can see that it is very different to the 0 that we have expected. This is because the formula that we were given is numerically unstable because the two subtractors can be very similiar. That is why we used a different algorithm in which we used shifted data variance which is numerically stable as the second term in subtracting is always smaller. We are however getting some small errors (appr 10^(-16)) which occurs because the algorithm is not perfectly the same as R is using and some computing error on the low level may occur.

A better formula of how to compute the variance is was obtained from Wikipedia: https://en.wikipedia.org/wiki/Algorithms_for_calculating_variance

$$Var(X - K) = Var(X)$$

with $K$ any constant, which leads to the new formula

$$variance = \frac{\sum_{i=1}^{n}(x_i - K)^2 - (\sum_{i=1}^{n}(x_i - K))^2/n}{n - 1}$$

## Question 4: Linear Algebra

```
tecator <- readxl::read_xls("tecator.xls")
tecator$ones <- rep(1,215)
y <- tecator$Protein
X <- as.matrix(tecator[,c(105,2:102,104)])
A <- t(X)%*%X
b <- t(X)%*%y
#res <- solve(A,b) CAN'T SOLVE!
kappa(A) # big kappa, close to singular.
```

```
## [1] 852351692132074.88
```

```
tecator_scaled <- tecator
for(i in c(2:102,104))
{
  tecator_scaled[i] <- scale(tecator[i])
}
y <- tecator_scaled$Protein
X <- as.matrix(tecator_scaled[,c(105,2:102,104)])
A <- t(X)%*%X
b <- t(X)%*%y
res <- solve(A,b)
#res
kappa(A) #kappa got smaller, got some reasonable result
```

```
## [1] 490471520661.25275
```

We computed A and b shown above. However we were not able to use solve function because the matrix A is "very close"" to the singular matrix (what we can notice because of having high kappa) Which means that it is not possible to solve the equation numerically despite analytically it should be possible. To solve this we had to use scaling with which we got better precision in couting and despite $\kappa$ is still quite high, it is low enough to solve the equation.

## Appendix

```
knitr::opts_chunk$set(echo = TRUE, eval = TRUE, warning = FALSE,
                      message = FALSE)
x1<-1/3
x2<-1/4
if(x1-x2 ==1/12)
{
  print("subtraction is correct")
} else
{
  print("Subtraction is wrong")
}
```

```r
#b)
x1<-1
x2<-1/2
if(x1-x2 ==1/2)
{
  print("subtraction is correct")
} else
{
  print("Subtraction is wrong")
}
options(digits=22)
1/3 # we can see that as 1/3 is a periodic number pc has a problem to put it in a memory.
1/4
1
1/2
myderiv <- function(x){
   ep <- (10**(-15))
   ((x + ep) - x)/ep
}

options(digits = 22)
options(scipen = 999)
myderiv(1)
myderiv(100000)

ep <- (10**(-15))
100000+ep
1+ep
myvar <- function(x, n)
{
  variance <- 1/(n-1) * ((sum(x^2)-(1/n)*(sum(x))^2))
  return(variance)
}

x <- rnorm(10000, 10^8, 1)
y <- rep(0, 10000)
for(i in 1:10000)
{
  y[i] <- myvar(x[1:i], i) - var(x[1:i])
}
plot(x=1:10000, y=y)

newvar <- function(x, n)
{
  K = mean(x)
  variance <- (1/(n-1)) * (sum((x-K)^2)-(1/n)*(sum(x-K))^2)
  return(variance)
}
for(i in 1:10000)
{
  y[i] <- newvar(x[1:i], i) - var(x[1:i])
}
plot(x=1:10000, y=y)
```

```r
tecator <- readxl::read_xls("tecator.xls")
tecator$ones <- rep(1,215)
y <- tecator$Protein
X <- as.matrix(tecator[,c(105,2:102,104)])
A <- t(X)%*%X
b <- t(X)%*%y
#res <- solve(A,b) CAN'T SOLVE!
kappa(A) # big kappa, close to singular.

tecator_scaled <- tecator
for(i in c(2:102,104))
{
  tecator_scaled[i] <- scale(tecator[i])
}
y <- tecator_scaled$Protein
X <- as.matrix(tecator_scaled[,c(105,2:102,104)])
A <- t(X)%*%X
b <- t(X)%*%y
res <- solve(A,b)
#res
kappa(A) #kappa got smaller, got some reasonable result
```