Computational Statistics 732A90

Lab 1

Albin Västerlund
albva223

Eric Herwin
Erihe068

Contents

Assignment 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"
```

```
# Computer roundning wrong
all.equal(x1-x2,1/12)
```

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

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

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

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

- 1. The values in the first expression is not exact the same. Thats because of the computer try to express a value thats need infinite numers to expressing the value. The values are therefore rounded to the nearest computer float. The function `all.equal()` compare the values if they are equal enough to just be computer float error, which is the case in this case.
- 2. A soloution could be to round the values towards a lesser amount of decimals, but we might lose accuracy.

Assignment 2. Derivative

```
#
Derivate<-function(f,x,epsilon){
  nr1<-f(x+epsilon)
  nr2<-f(x)

  derivatan<-(nr1-nr2)/epsilon
  derivatan
}
epsilon<-10^(-15)

Derivate(sum,1,epsilon) #x = 1
```

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

```
Derivate(sum,23,epsilon) #x = 100000
```

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

In both cases we are supposed to get the value 1, which we dont get in any of the cases.

In the first case when `x=1` there is room to add the small value ε in the binary representation. But we dont add exactly ε becaues the compter have not room for adding the exact number. It adding a value close to ε which is a little bigger then ε. We therefore get a number bigger then 1.

In the second case when we use `x=100000` the function return the value 0. Thats because $f(x+\varepsilon)$ is exact the same as $f(x)$ because in the binary represenation of `x=100000` there is no room for adding a such samll value as ε, and therefore the diffrense betweeen the numbers become 0.

This is called underflow.

Assignment 3. Variance

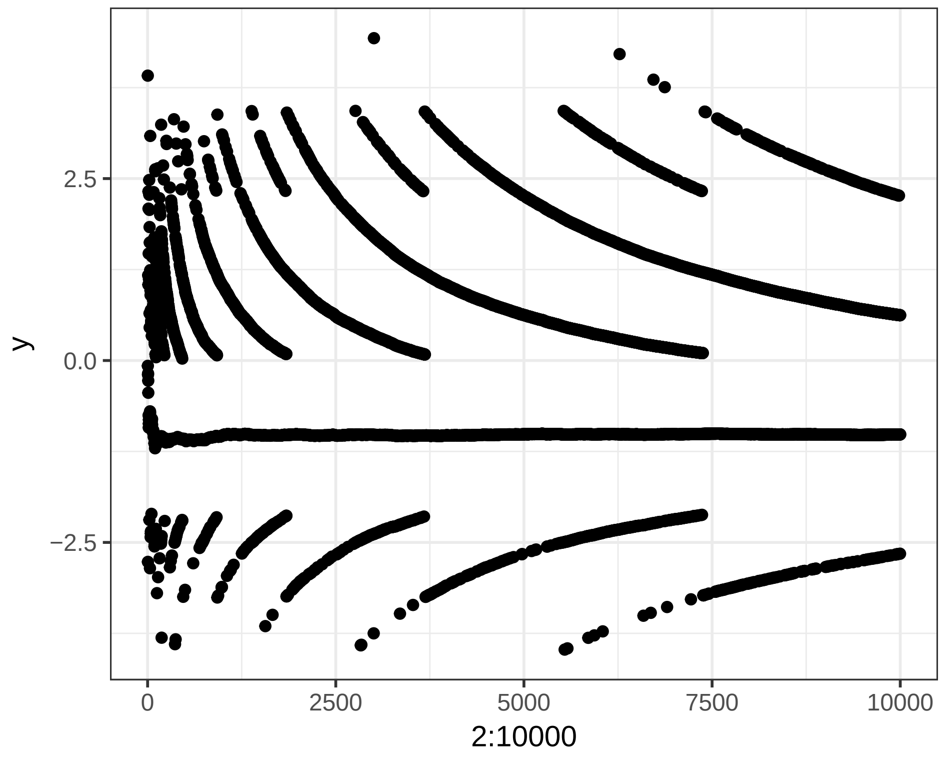
```r
#1
myvar<-function(x){
  n<-length(x)

  varians<-1/(n-1)*(sum(x^2)-(1/n)*sum(x)^2)
  varians
}

#2
x<-rnorm(10000,10^8,1)

#3
y<-sapply(2:10000,
          FUN=function(i){
            myvar(x[1:i])-var(x[1:i])
            })

ggplot(mapping = aes(y=y,x=2:10000))+
  geom_point() + theme_bw()
```



- 3. From the plot we can see that the expected difference is not 0. In our `myvar()` the difference is sometimes 0 because the sums over the squared `x` will become very large and in some cases both sums will be exatly the same. Therefore our function will give back that the variance is 0 and then `myvar(x) - var(x)` will turn out not to be 0. The observations that is far below 0 indicates that some outputs of `myvar()` is negative. This kind of problems are called overflow when a value is to large for addning “normal” size values to it.

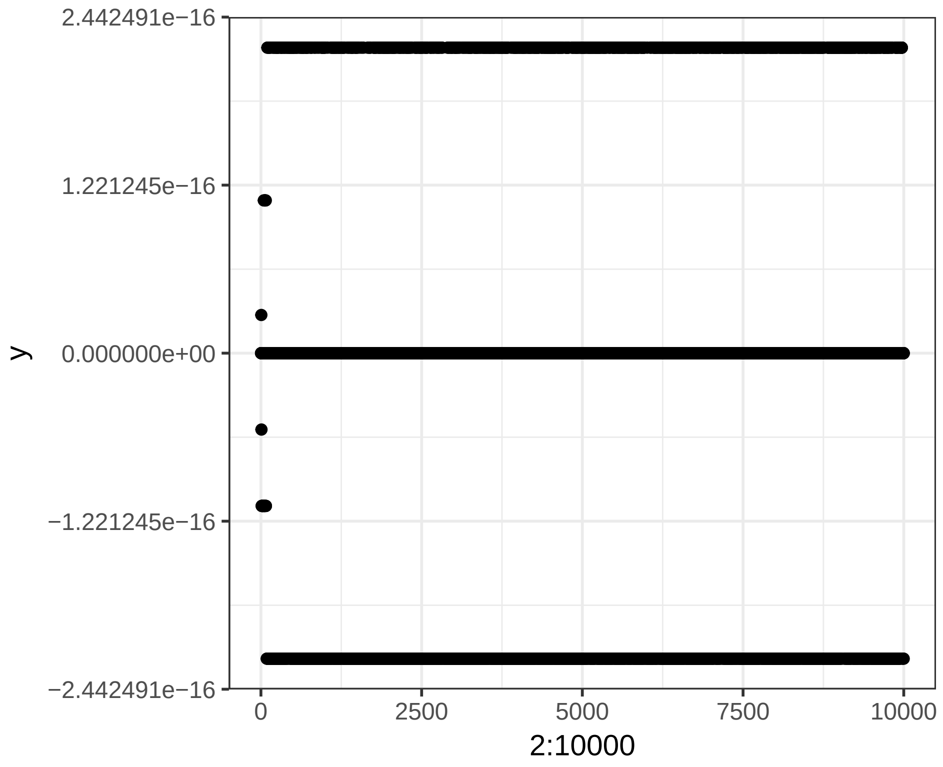
```
#4
myvar_new <- function(x){

  men <- mean(x)
  n <- length(x)

  varians <- 1/(n-1)*sum((x-men)^2)
  varians
}

y<-sapply(2:10000,
          FUN=function(i){
            myvar_new(x[1:i])-var(x[1:i])
            })

ggplot(mapping = aes(y=y,x=2:10000))+
  geom_point() + theme_bw()
```



- 4. We can see that from the plot that the difference is very similar and the values are 0 or very close to 0. So making a computation that avoids handling very large numbers as much as possible will give better accuracy.

Assignment 4. Linear Algebra

```
#1
tecator <- read_csv("tecator.csv")
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   Sample = col_integer()
## )
```

```
## See spec(...) for full column specifications.
```

```
X<-as.matrix(tecator %>% select(-Protein,-Sample))
y<- as.matrix(tecator %>% select(Protein))

#2
A<-t(X)%*%X
b <- t(X) %*% y


#3
#beta <- solve(A, b) #error
```

- 3. When we use `solve` on the matrix A, we get the error that the system is computationally singular.

This is because we have very large numbers in the matrix and when computing for example the determinant there will be a overflow and that might lead to a determinant that is equal to 0.

```
#4
kappa(A)
```

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

- 4. Because the condition number is high, it means that the determinant of matrix **A** is low and that means that there might be that the matrix is not full rank.

```
#5
X_s <- scale(X)
A_s <- t(X_s) %*% X_s
b_s <- t(X_s) %*% scale(y)
beta_s <- solve(A_s, b_s) #works!
```

- 5. When we scale the data, we are taking away all the variablility and centring the data by the mean. Then we will get data that has much lower numbers and when doing the `solve` on the **A** made with the scaled **X**. The determinant will then handle smaller numbers the determinant might not be 0. Scaling is a method for handling non-singular matrices. Also another method that might be useful for inversing is QR-decomposition.