Computational Statistics

Lab 1

Emil K Svensson and Rasmus Holm

2017-01-31

Question 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"
```

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

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

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

```
## all.equal(x1-x2, 1/2)
```

The first part of the snippet says that the subtraction is wrong although if we see it should be the same in regular mathematical arithmatic. This is because the fractions 1/3 and 1/12 can not be represented as correct as 1/2. One should use the function all.equal instead of the logical operator as it takes in consideration the machine epsilon.

1/2 can be represented exact in the binary form. 1/3 and 1/12 are both repeating rational numbers gets rounded to the nearest point float by the machine and since 1/12 is nearer a point in the space of the distribution of computer represented floats.

Question 2

```
f_der <- function(f, eps=10^(-15)){
    function(x){
        (f(x + eps) - f(x)) / eps
    }
}

f <- function(x){ x }
```

```
fprime <- f_der(f)

fprime(1)
```

```
## [1] 1.110223024625156540424
```

```
fprime(100000)
```

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

The true values for both fprime is 1, since the derivate of x in respect to x always is 1.

For 100000 the epsilon is to insignificant/small and gets neglected when added to 100000. f(x + eps) = f(x) and the numerator becomes zero.

In the example with x = 1 the f(x + eps) gets rounded to another floating value that will be a missmatch with the esp in the denominator and therefor not yeild the correct result 1.

Question 3

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



myvar2 <- function(x) {
    n <- length(x)
    (1 / (n - 1)) * sum((x - mean(x))^2)
}

set.seed(1234567890)
x <- rnorm(10000, mean = 10^(8), sd = 1 )
paste("myvar:", myvar(x))
```

```
## [1] "myvar: 0"
```

```
paste("var:", var(x))
```

```
## [1] "var: 0.999195900004603"
```

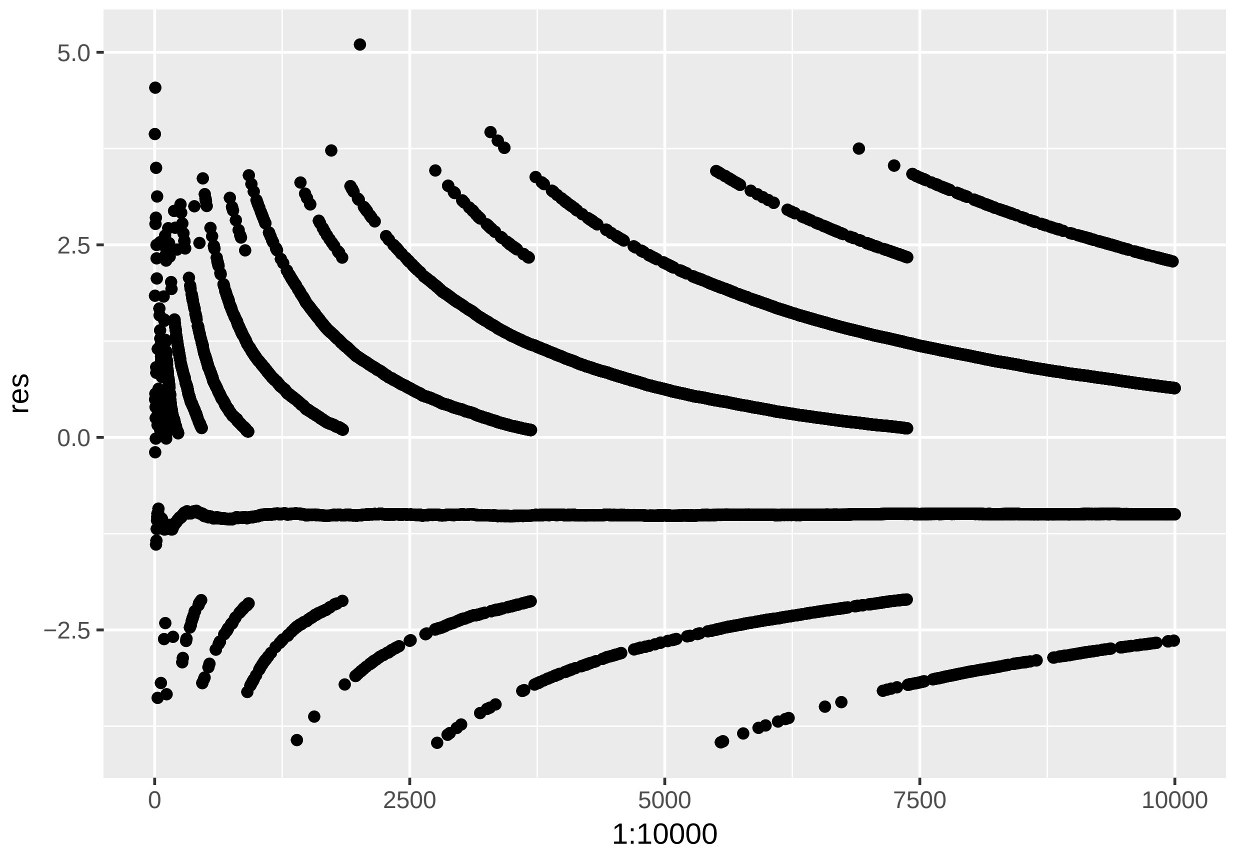
Not good at all, our first function calculates the variance to 0 where it should be around 1.

```
res <- c()
for (i in 1:10000){
    res[i] <- myvar(x[1:i]) - var(x[1:i])
}

library(ggplot2)

ggplot() + geom_point(aes(x = 1:10000, y = res))
```

```
## Warning: Removed 1 rows containing missing values (geom_point).
```



Here we see the errors of my our function myvar compared to the built-in function var. This is because we use squared x that are summed together which results in very large numbers that can't be correctly represented on the floating point scale since it is less dense on larger numbers.

In the next code-chunk we implement a better version that takes the difference between the observations and the mean and after that squares and sums them. This gives us calculations with smaller numbers that can be more accuratly representet in the floating point system since it is more dense there.

```
myvar2 <- function(x) {
  n <- length(x)
  (1 / (n - 1)) * sum((x - mean(x))^2)
}

paste("myvar2:",myvar2(x))
```

```
## [1] "myvar2: 0.999195900004603"
```

```
paste("var:",var(x))
```

```
## [1] "var: 0.999195900004603"
```

Perfect!

Question 4

Unscaled

```
ta <- read.csv("../data/tecator.csv")

X = ta[,-c(1,103)]
Y = ta[,103]

X <- cbind(rep(1,nrow(X)), as.matrix(X))
Y <- as.matrix(Y)

a <- t(X) %*% X
b <- t(X) %*% Y
## solve(a) %*% b
print("Error in solve.default(a) : system is computationally singular:
reciprocal condition number = 7.78804e-17")
```

```
## [1] "Error in solve.default(a) : system is computationally singular:\nreciprocal condition number =
```

The computation can not be preformed. Without scaling the data the A matrix will contain relatively large values which cannot be represented as accurately as smaller numbers in floating point. Gaussian elimination used in solve uses a lot of subtractions and since the values are less accurately represented it can happen that numbers become close to zero which is intepreted as 0 by the solve function resulting in a singular matrix, i.e. non-invertable matrix.

```
paste("kappa off the unscaled a:",kappa(a))
```

```
## [1] "kappa off the unscaled a: 852351692132075"
```

A large kappa is a bad sign.

Scaled

```
tas<- scale(ta)

X = tas[,-c(1,103)]
Y = tas[,103]

Xs <- cbind(rep(1,nrow(X)),as.matrix(X))
Ys <- as.matrix(Y)

as <- t(Xs) %*% Xs
bs <- t(Xs) %*% Ys
head(solve(as) %*% bs)
```

```
##                                  [,1]
##          -1.873062689908735438074e-12
## Channel1 -1.106123672467074356973e+02
## Channel2 -2.212873564213077770546e+02
## Channel3  3.781193650527616227919e+02
## Channel4 -1.297293022649828344584e+02
## Channel5  4.133177902315146639012e+02
```

It works!

```r
paste("kappa off the scaled a:", kappa(as))
```

```
## [1] "kappa off the scaled a: 490471520661.253"
```

A smaller kappa than previously.

Since the kappa of a is a ratio between the largest and smallest eigenvalue the scaling will shrink the eigenvalues in to a smaller ratio.