Computational Statistics

Lab 1

Emil K Svensson and Rasmus Holm

2017-02-03

Question 1

```r
x1 <- 1 / 3
x2 <- 1 / 4
if (x1 - x2 == 1 / 12) { # all.equal(x1 - x2, 1/12)
    print("Subtraction is correct")
} else {
    print("Subtraction is wrong")
}
```

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

```r
x1 <- 1
x2 <- 1 / 2
if (x1 - x2 == 1 / 2) {  # all.equal(x1 - x2, 1/2)
    print("Subtraction is correct")
} else {
    print("Subtraction is wrong")
}
```

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

The first part of the snippet says that the subtraction is wrong although if we see it should be the same in regular mathematical arithmetic. This is because the fractions 1/3 and 1/12 cannot be represented exactly and it will therefore be rounding errors. One should use the function *all.equal* instead of the logical operator as it takes the machine epsilon into consideration.

In the second code snippet the subtraction is correct since 1/2 can be represented exactly in binary form so no rounding errors will occur. When using floating points we suggest to always use *all.equal* to check for equality.

Question 2

```r
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] 1.11
```

```r
fprime(100000)
```

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

The true value of the function is always 1 regardless of input since the derivative of x with respect to x is always 1.

When setting $x = 1$ then $x + \epsilon$ gets rounded to some floating point value so that $(x + \epsilon) - x \approx \epsilon$ which results in $f(x + \epsilon) - f(x) \approx \epsilon$. That is the reason we don't get an answer of 1, the numerator and denominator are not exactly the same.

ϵ is too insignificant when added to 100000 and gets neglected, i.e. no floating point value closer to $100000 + \epsilon$ than 100000. This entails that $f(x + \epsilon) = f(x)$ and the numerator becomes zero.

Question 3

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

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

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

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

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

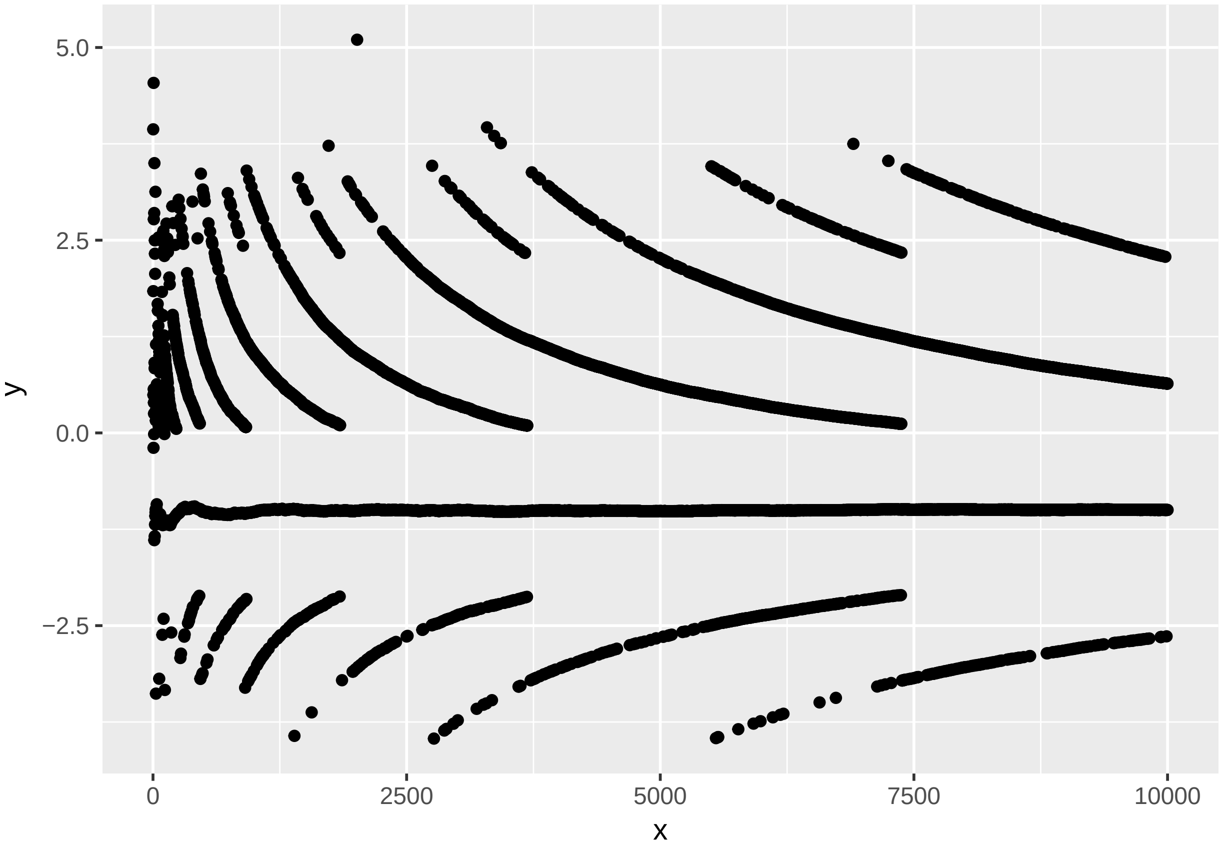
What we can see is that our function does not calculate the variance well at all. It should be approximately 1 but instead returns 0 in this example.

```r
n <- 10000
res <- rep(0, n - 1)
for (i in 2:n){
    res[i - 1] <- myvar(x[1:i]) - var(x[1:i])
}

library(ggplot2)

plot_data <- data.frame(x=2:n, y=res)

ggplot() + geom_point(data=plot_data, aes(x=x, y=y))
```



Here we see the errors of our function *myvar* compared to the built-in function *var*. The problem is that we do sums which results in much larger quantities that are less accurately represented in the floating point system and subtracting two large quantities in order to find a small number is a recipe for disaster in terms of accuracy.

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 quantities that can be more accurately represented in the floating point system.

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

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

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

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

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

The results are equivalent as we would have hoped.

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
writeLines(try(solve(A) %*% b, silent=TRUE)[1])
```

```
## Error in solve.default(A) :
##   system is computationally singular: reciprocal condition number = 7.78981e-17
```

The computation can not be preformed. Taking the determinant $\det(A) = 0$ we see that it is not invertible according to R which indicate that the features are correlated to each other.

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

```
## [1] "kappa off the unscaled A: 852624020599293"
```

A large kappa value is a bad sign, can mean ill-condition, and in this case it gives a bound on how inaccurate the solution x will be. Basically small errors in b may cause large errors in x. However, it does not explain why the matrix turned out to be singular in the solve function.

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]
##          -1.874e-12
## Channel1 -1.106e+02
## Channel2 -2.213e+02
## Channel3  3.781e+02
## Channel4 -1.295e+02
## Channel5  4.131e+02
```

In this case it is possible to find parameters since we scaled the data and the matrix A did not turn out to be singular.

What is interesting is that the determinant is 0 for both the scaled and non-scaled A matrices but it is possible to invert the scaled matrix. We think this has to do with the floating point system and accuracy, the working with smaller numbers in the scaled matrix do prevent instability in the calculations while the non-scaled matrix suffers from large rounding errors causing the instability.

```
paste("kappa off the scaled A:", kappa(As))
```

```
## [1] "kappa off the scaled A: 595262318256.792"
```

A smaller kappa than previously which indicate that the linear system is more stable with scaled data but still a very large number that indicates that the system is volatile.