Lab 1

Jesper Lindberg & Andreas Stasinakis

1/24/2019

Contents

Contributors : Jesper Lindberg & Andreas Stasinakis

Question 1 Arithmetic (Jesper Lindberg)

In this question we are supposed to compare two code snippets. We are supposed to analyze why they are giving us the "wrong" (or the correct) output. In the first snippet we are going to take the following equations and see if it's correct. $\frac{1}{3} - \frac{1}{4} = \frac{1}{12}$

```
options(digits = 22)
x1<-1/3
x2<-1/4
if(x1-x2==1/12){
  print("Substraction is correct")
}else{
  print("Substraction is wrong")
}
```

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

The given output is that the equation is wrong, however it is mathematically correct. The problem here is that when handling very large numbers, in this case a lot of decimals, R have to switch a bit, as a result of underflow, which will make small difference in the end results. If we change the output digits to 22, to be able to see the decimals, we can see the following:

$\frac{1}{3} - \frac{1}{4} =$

```
1/3-1/4
```

```
## [1] 0.0833333333333333314829620
```

$\frac{1}{12} =$

```
1/12
```

```
## [1] 0.0833333333333333287074
```

This is because there is no way to represent these numbers in binary without using infinite amount of bits.

In the second code snippet, we are supposed to check $1 - \frac{1}{2} = \frac{1}{2}$ and we get:

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

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

This is because $\frac{1}{2} = 0.5$ and do not have infinite amount of decimal points. The problems, from the first snippet, can be solved by only working with integers and not float points, it can also be done by chaining to a special decimal number type. You can also use the R function "all.equal()", which will "Test if Two Objects are (Nearly) Equal". So by using that function in the first snippet, the output is now:

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

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

Which is correct.

Question 2 Derivative(Jesper Lindberg)

In question number two, we are supposed to make our own derivative function, by using the following formula:

$$f'(x) = \frac{f(x+\epsilon) - f(x)}{\epsilon}, \epsilon = 10^{-15}$$

We are then supposed to try it with $x = 1$ and $x = 100000$ which gives us the following results:

```
myDerivative<-function(x,ep=10^-15){
  returnValue<-(x+ep-x)/ep
  returnValue
}

myDerivative(1)
```

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

```
myDerivative(100000)
```

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

The correct answers should be, in both cases: 1. The reason this is happening is because we first add a very small decimal number (ϵ) to a integer, which makes that integer a float. We then remove the value of the integer, which should just leave us with the value of (ϵ). However, as computers works with binary, there is a risk for overflow and underflow (when a number gets too large for the storage, or too small) which will cause it add or remove numbers to the final results. This was shown in question 1. We then divide it by ϵ.

For the first case, $x = 1$, the numerator of the derivative function is $x + \epsilon - x$. It is obvious that the result is ϵ and the final derivative is $\frac{\epsilon}{\epsilon} = 1$. But from R's perspective $x + \epsilon - x$ is not equal to ϵ. This is happening because in the arithmetic operations $A + X = X$, but $X - X = A$.

For the second case, $x = 100000$, the result of $x + \epsilon - x$ is 0 instead of ϵ. This is happening because x has a really high value and adding ϵ does not change this value. So for the computer is 10000 - 100000 which is equal to 0. So in this case we have underflow, which means that we lose binary information because there are too many decimals.

Question 3 : Variance(Andreas Stasinakis)

A known formula for estimating the variance based on a vector of n observations is

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

Write your own R function, myvar, to estimate the variance in this way.

```
# R function for calculating the variance of a vector

myvar = function(x){

  # the sum of all x components
  sum_x = sum(x)
```

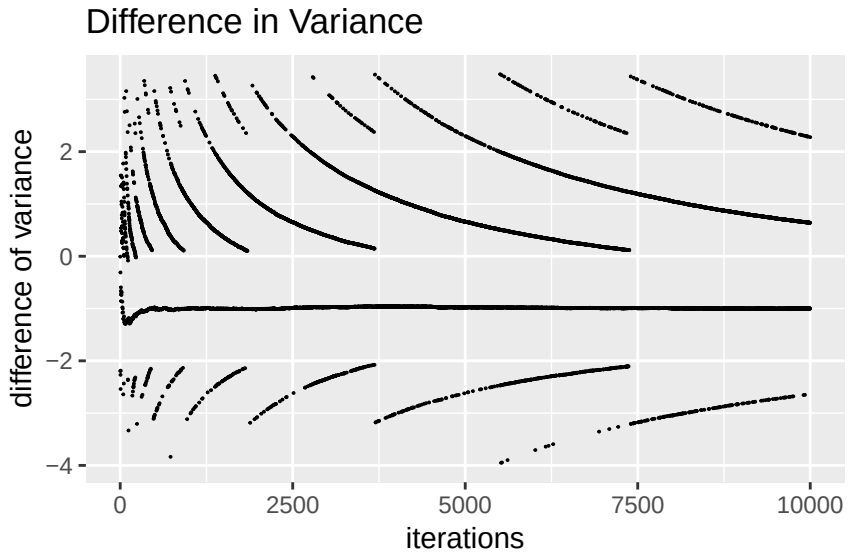
```
  # the square of the sum_x devided by the observations

  s_sum = (sum_x^2)/length(x)

  # The sum of x^2
  square_sum = sum(x^2)

  # the final formula for variance
  var_x = (square_sum - s_sum)/(length(x)-1)

  return(var_x)
}
```

Generate a vector $x = (x_1, ..., x_{10000})$ *with 10000 random numbers with mean* 10^8 *and variance 1.*

```
set.seed(12345)

#generate a sample
dt = rnorm(10000, 10^8, 1)
```

For each subset $X_i = (x_1, ..., x_i), i = 1, ..., 10000$ *compute the difference* $Y_i = myvar(X_i) - var(X_i)$*, where* $var(X_i)$ *is the standard variance estimation function in R. Plot the dependence* Y_i *on* i*. Draw conclusions from this plot. How well does your function work? Can you explain the behaviour?*

```
library(ggplot2)
# a function calculates the Yi for given subset

dif = function(n, data){

  # vector to store the Y's
  y = c()

  # for loop for subset depend on the given n
  for (i in 1:n) {
  y[i] = myvar(data[1:i]) - var(data[1:i])
  }

  return(y)
}

# Using the function for n = 10000 and the given data

 Y = dif(10000,dt)

# plot the dependence Yi and i

plot_df = data.frame(i = c(2:10000), Y = Y[-1])

ggplot(plot_df) +
  geom_point(mapping = aes(x = plot_df$i, y = plot_df$Y), size = 0.01) +
  labs(title = "Difference in Variance", x = "iterations", y = "difference of variance")
```



In this case we calculate the variance for our sample with two ways. One with the function myvar we implement and one with the R function var. Finally we plot this difference vs the i.

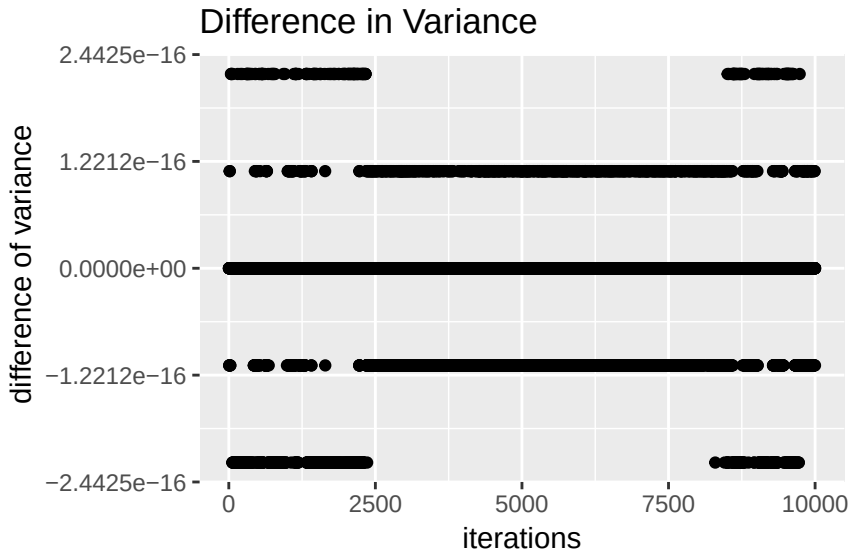
It is obvious from the plot that our function is not giving accurate results. The difference between the two functions should be zero, or at least close to, but we can see that the range of the differences is from approximately -4 to 4. We can also observe two trends. One starting from almost 4, reducing exponentially and stops close to 0, and one other starting close to -4, increasing exponentially before stopping to -2. Moreover, the average difference is an almost straight line close to -1 which is another prove that the differences are quite big(The average line should be close to 0).

There are two reason why is this happening. The first one is the formula we use in order to calculate the variance and the second one is that computer can not store too large numbers or numbers with too many decimals. More specific, in our formula we are calculating the square of the observations and the sum of the square of the x_i's. In this case the computer can not store the float numbers, so it rounded after some decimals. This may look like a small difference but when we sum and square this number becomes big enough in order to have this really different values.

How can you better implement a variance estimator? Find and implement a formula that will give the same results as var()?

```
# Different implementation of variance estimator
myvar2 = function(data){
  df = (data - mean(data))^2

  varr = sum(df)/(length(data)-1)
  return(varr)
}

# anothere function to calculate the difference
dif2 = function(n, data){

  # vector to store the Y's
  y = c()

  # for loop for subset depend on the given n
  for (i in 1:n) {
  y[i] = myvar2(data[1:i]) - var(data[1:i])
  }
```

```
  return(y)
}

options(digits = 5)
Y2 = dif2(10000,data = dt)

p = data.frame(i = c(2:10000), Y = Y2[-1])

ggplot(p) +
  geom_point(mapping = aes(x = p$i, y = p$Y)) +
  labs(title = "Difference in Variance", x = "iterations", y = " difference of variance")
```



In this task we implement a better and more accurate formula for calculating the variance. We chose a simpler formula :

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

, where here $n = 10000$. The results obtained with this formula are approximately the same as the function var. We also plot the dependence of Y_i to i and it is clear that the difference between the two functions is close to 0 for every i which means that the values are almost the same.

Question 4: Linear Algebra(Andreas Stasinakis)

The Excel file "tecator.xls" contains the results of a study aimed 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 each meat sample the data consists of a 100 channel spectrum of absorbance records and the levels of moisture (water), fat and protein. The absorbance is -log10 of the transmittance measured by the spectrometer. The moisture, fat and protein are determined by analytic chemistry. The worksheet you need to use is "data" (or file "tecator.csv"). It contains data from 215 samples of finely chopped meat. The aim is to fit a linear regression model that could predict protein content as function of all other variables.

Import the data

```
library(readxl)
tecator = read_xls("tecator.xls")
```

Optimal regression coefficients can be found by solving a system of the type $Ax = b$ where $A = X^T X$ and $\hat{b} = X^T y$. Compute A and b for the given data set. The matrix X are the observations of the absorbance records, levels of moisture and fat, while y are the protein levels.

```
X = as.matrix(tecator[,-c(1,103)])

y = as.vector(tecator$Protein)

# calculating A and b

A = t(X)%*%X

b = t(X)%*%y
```

Try to solve $A\beta = b$ with default solver solve(). What kind of result did you get? How can this result be explained?

```
# use the function solve for the beta.
#solve(A,b)
```

In this case we try to use the solve function in order to calculate the regression coefficients. The output is the error below :

Error in solve.default(A, b) : system is computationally singular: reciprocal condition number = 7.13971e-17

When you have to solve a linear equation as here we should have at least as many equations as the unknown coefficients we want to calculate. In this case though, some columns of matrix A are highly correlated each other (not independent). The problem is that we have really high float values of the matrix A and R can not store all decimals, as a result R rounds that numbers and we have the equal numbers in many cases. Moreover, after rounding, all the highly correlated columns have the same values and all the small decimal differences disappear. Also the rank of the matrix decreases and finally A seems a singular matrix. For that reason when we try to solve the equation we will have less coefficients than we actually need to solve the equation as a result the equation can not be solved. Moreover, the determinant of matrix A is 0 which means that A is not invertible, but we need A^{-1} in order to calculate the coefficients.

Check the condition number of the matrix A (function kappa()) and consider how it is related to your conclusion in step 3.

```
# use kappa function for matrix A
```

```r
cond_number = kappa(A)
print(list("Condition number of the matrix A :", cond_number))
```

```
## [[1]]
## [1] "Condition number of the matrix A :"
##
## [[2]]
## [1] 1.1578e+15
```

We use the function kappa to calculate the condition number of the matrix A. The condition number of the matrix can be calculated by the following formula :

$$\|A\| \left\|A^{-1}\right\|$$

. We have to mention that the larger this number is, the worse is for the computer system. In this case we can see that the value of the condition number is really high. In general, we want a stable linear system which means that if you change the b slightly we will have smalls changes in the output also. This means that the condition number of the matrix A should be low, otherwise the system will not have this behavior. As mentioned before, the condition number of the matrix A is real big which means that the system is unstabilized and small changes in the input will create real different outputs.

Scale the data set and repeat steps 2-4. How has the result changed and why?

```r
# scale the data
X_scale = scale(X)

y_scale = scale(y)

# calculating A and b

A = t(X_scale)%*%X_scale

b = t(X_scale)%*%y_scale

# solve the equation

beta = solve(A,b)

A_kappa = kappa(A)
```

In this case we firstly scale the data and after that calculate the matrix A and vector b. After this procedure, solve function does not give an error and we can calculate the unknown coefficients. This is happening because despite the fact that some columns are highly correlated each other, R can store them without any rounding. Therefore, during the procedure we will not have any equal columns and we have the number of coefficients we need to solve this equation.

Appendix

```
knitr::opts_chunk$set(fig.width = 4.5, fig.height = 3,
                      fig.align = "center",
                      warning = F, error = F, message = F)
#Library imports
library(ggplot2)
options(digits = 22)
x1<-1/3
x2<-1/4
if(x1-x2==1/12){
  print("Substraction is correct")
}else{
  print("Substraction is wrong")
}

1/3-1/4
1/12
x1<-1
x2<-1/2
if ( x1-x2==1/2 ) {
  print ( "Substraction is correct" )
} else {
  print ( "Substraction is wrong" )
}

x1<-1/3
x2<-1/4
if(isTRUE(all.equal(x1-x2,1/12))){
  print("Substraction is correct")
}else{
  print("Substraction is wrong")
}
myDerivative<-function(x,ep=10^-15){
  returnValue<-(x+ep-x)/ep
  returnValue
}

myDerivative(1)
myDerivative(100000)
# R function for calculating the variance of a vector

myvar = function(x){

  # the sum of all x components
  sum_x = sum(x)

  # the square of the sum_x devided by the observations

  s_sum = (sum_x^2)/length(x)

  # The sum of x^2
  square_sum = sum(x^2)
```

```r
  # the final formula for variance
  var_x = (square_sum - s_sum)/(length(x)-1)

  return(var_x)
}
set.seed(12345)

#generate a sample
dt = rnorm(10000, 10^8, 1)
library(ggplot2)
# a function calculates the Yi for given subset

dif = function(n, data){

  # vector to store the Y's
  y = c()

  # for loop for subset depend on the given n
  for (i in 1:n) {
  y[i] = myvar(data[1:i]) - var(data[1:i])
  }

  return(y)
}

# Using the function for n = 10000 and the given data

 Y = dif(10000,dt)

# plot the dependence Yi and i

plot_df = data.frame(i = c(2:10000), Y = Y[-1])

ggplot(plot_df) +
  geom_point(mapping = aes(x = plot_df$i, y = plot_df$Y), size = 0.01) +
  labs(title = "Difference in Variance", x = "iterations", y = "difference of variance")

# Different implementation of variance estimator
myvar2 = function(data){
  df = (data - mean(data))^2

  varr = sum(df)/(length(data)-1)
  return(varr)
}

# anothere function to calculate the difference
dif2 = function(n, data){

  # vector to store the Y's
  y = c()

  # for loop for subset depend on the given n
  for (i in 1:n) {
```

```
  y[i] = myvar2(data[1:i]) - var(data[1:i])
  }

  return(y)
}

options(digits = 5)
Y2 = dif2(10000,data = dt)

p = data.frame(i = c(2:10000), Y = Y2[-1])

ggplot(p) +
  geom_point(mapping = aes(x = p$i, y = p$Y)) +
  labs(title = "Difference in Variance", x = "iterations", y = " difference of variance")

library(readxl)
tecator = read_xls("tecator.xls")
X = as.matrix(tecator[,-c(1,103)])

y = as.vector(tecator$Protein)

# calculating A and b

A = t(X)%*%X

b = t(X)%*%y

# use the function solve for the beta.
#solve(A,b)

# use kappa function for matrix A

cond_number = kappa(A)
print(list("Condition number of the matrix A :", cond_number))
# scale the data
X_scale = scale(X)

y_scale = scale(y)

# calculating A and b

A = t(X_scale)%*%X_scale

b = t(X_scale)%*%y_scale

# solve the equation

beta = solve(A,b)

A_kappa = kappa(A)
```