LAB1_CS

Andreas Stasinakis

January 22, 2019

Contents

Arithmetic

Wrong if because of the decimals

1. Check the results of the snippets. Comment what is going on.

2. If there are any problems, suggest improvements.

```
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"
```

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

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

In the first snippet the printed message is "subtraction is wrong", which of course it is not correct. More specific, we set $x_1 = \frac{1}{3}$ and $x_2 = \frac{1}{4}$ so $x_1 - x_2 = 0.08333333333333315$ but also $\frac{1}{12} = 0.08333333333333329$ which means that the if condition is correct. In the second snippet we set $x_1 = 1$ and $x_2 = \frac{1}{2}$ and the if condition was $x_1 - x_2 = \frac{1}{2}$. The printed message was "Subtraction is correct" which is the correct one. The reason why is this happening is that in the second snippet the are specific numbers of decimals (only 1) while in the first one the number of decimals is infinite. Therefore, both 1/3 - 1/4 and 1/12 is approximations and not specific calculated(R probably round them after a decimal and for that reason we will have slightly different results.

The solution in this case is to use the function all.equal. This function tests the equality between two objects and if they are nearly equal, it makes them equal. Moreover, if they are different, it returns the differences. So if we run this function between the objects $x_1 - x_2$ and $\frac{1}{12}$. The correct answer is below :

```r
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"
```

Derivative

From the definition of a derivative a popular way of computing it at a point x is to use a small ϵ and the formula :

$$f'(x) = \frac{f(x+\epsilon) - f(x)}{\epsilon}$$

Function for calculating the derivative of x

Write your own R function to calculate the derivative of $f(x) = x$ in this way with $\epsilon = 10^{-15}$.

```r
e = 10^(-15)

#function calculates the derivative

der = function(form, epsi){

  #numerator of the derivative
  num = (form + epsi) - form

  #denominarator
  dem = epsi

  #derivative
  deriv = num/dem

  return(deriv)
}
```

Use the function for different values of x

Evaluate your derivative function at $x = 1$ and $x = 100000$

```
#For x = 1, using the derivative function

x = der(1,e)
print(list("The derivative for x=1 is", x))
```

```
## [[1]]
## [1] "The derivative for x=1 is"
##
## [[2]]
## [1] 1.1102230246251565
```

```
# For x =100000

y = der(100000,e)
print(list("The derivative for x=100000 is", y))
```

```
## [[1]]
## [1] "The derivative for x=100000 is"
##
## [[2]]
## [1] 0
```

Analysis

What values did you obtain? What are the true values? Explain the reasons behind the discovered differences.

Using the function we implement in the first task we take the following results. For $x = 1$, the output of the function is 1.11022302462516 while it should be 1. Also for $x = 100000$ the output is 0 while is should be also 1. As mentioned before the derivative of x is always 1.

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.

Variance

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

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

Function for estimating the variance

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 random numbers(normally distributed)

Generate a vector $x = (x_1, ..., x_10000)$ *with 10000 random numbers with mean* 10^8 *and variance 1.*

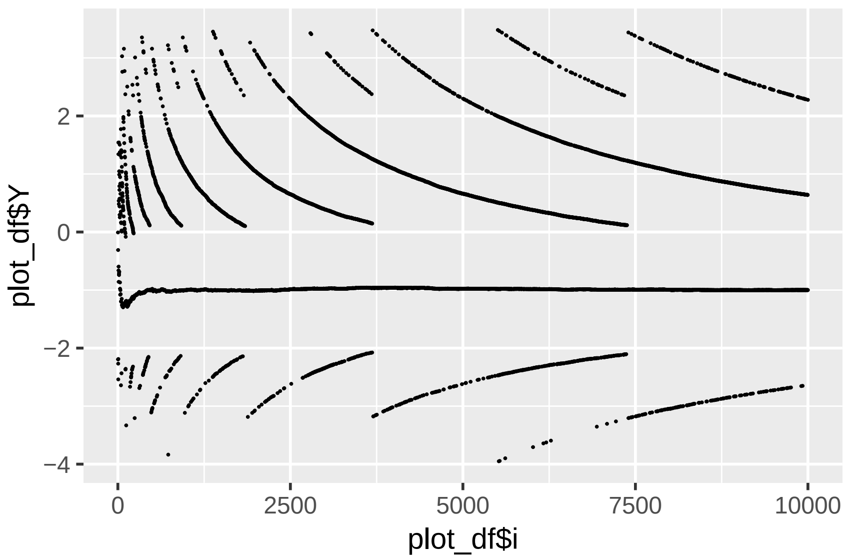
```
set.seed(12345)

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

Calculate the Y = difference between myvar and var(), Plot Y vs i, Analysis

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)
```



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.

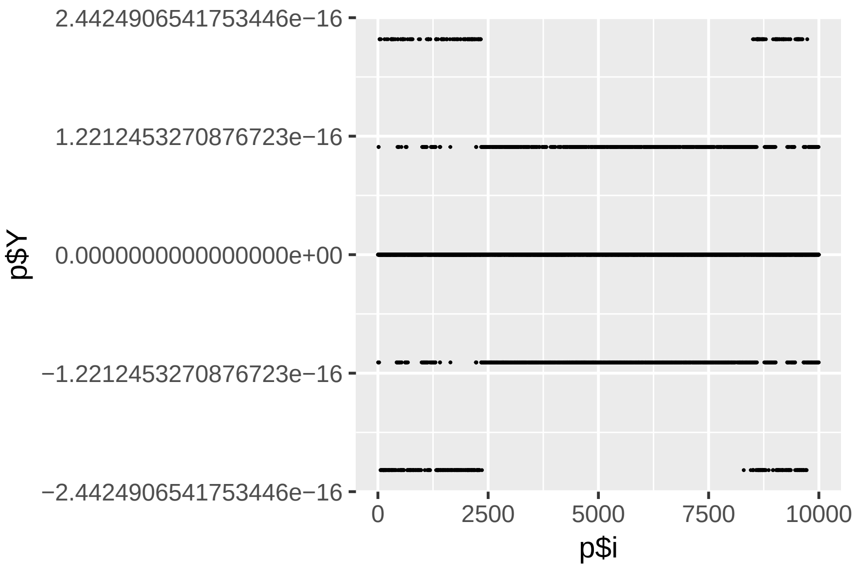
It is obvious from the plot that our function is not giving accurate results. The difference between the two functions should be zero, or at least close to, but we can see that the range of the differences is from approximately -4 to 4. We can also observe two trends. One starting from almost 4, reducing 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).

The reason why is this happening is because both of the formula we use in order to calculate the variance and the computer's problem to 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.

Use different formula for variance

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)
}

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), size = 0.01)
```



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

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

, 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.

Linear Algebra

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.

Importa xl data

Import the data

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

Estimating coefficients

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
```

Function solve(not working) and reason's why

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.

Conditional number and Analysis

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

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

```
## $`Condition number of the matrix A :`
## [1] 1157834236871692.2
```

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 slightly the b 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.

Same procedure with scaled data, Analysis

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.