Lab1

Andreas C Charitos

23 Jan 2019

Question 1

Running the 2 snippets we get the following results

First snippet

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

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

Second snippet

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

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

Evaluating the results of the 2 snippets we see that in the first occasion we get the wrong print of the if-else statement.The problem lies to the fact that float numbers that have infinite numbers of decimals can't be represented exactly in the binary system in computers due to memory storage limitation.Using print(x1-x2,digits=16) and print(1/12,digits=16) we will see that the resulting floats are (0.08333333333333331,0.08333333333333333) respectfully and they are not the same causing the condition of unerflow which leads to the failure of the if statement and evaluation of else. We can adress this problem using the "all_equals()" in the if statement instead of "==" to compare the numbers and we will see that the if statement will be executed and the correctly print message will be outputed. The second statement is evaluated correctly and we get the correct print output because 1/2 has finite numbers of decimals so we don't have the occurence of underflow here.

Question 2

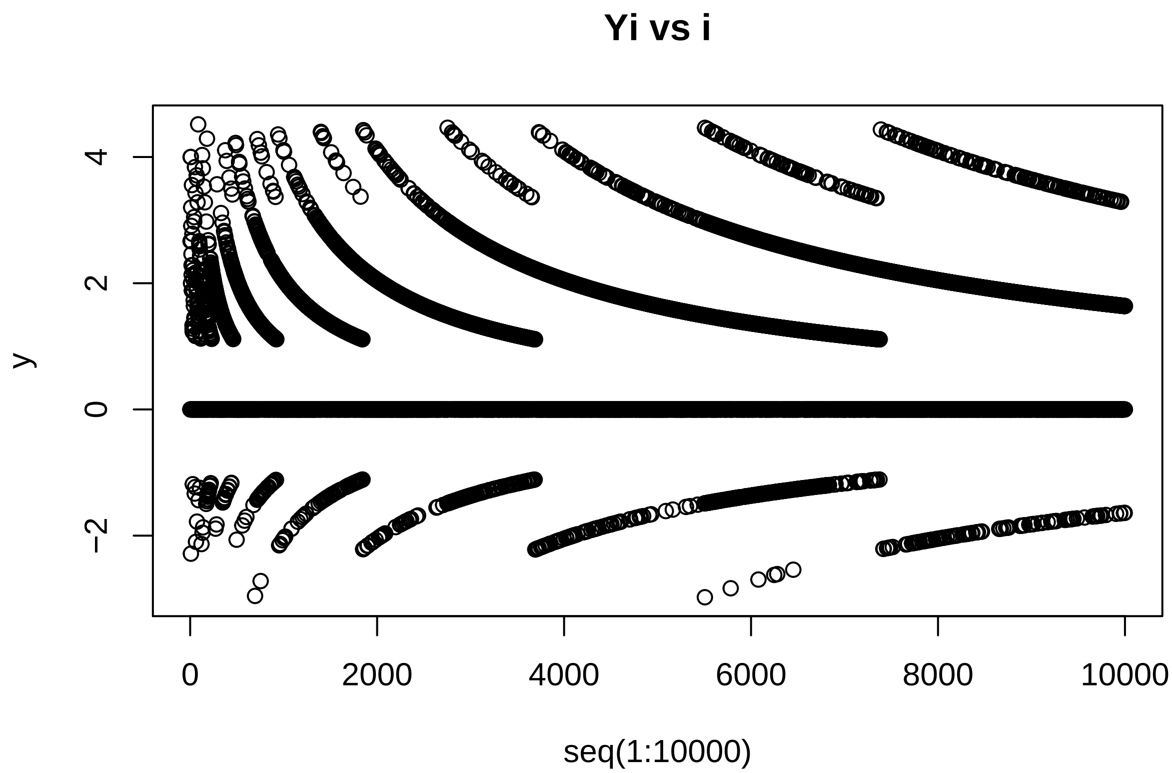
```
derivative <-function(f,epsilon){
  d<-((f+epsilon)-f)/epsilon
  return(d)
}

cat("==============================================\n",
    "The derivative for x=1 is :",derivative(1,10^-15),"\n",
    "The derivative for x=10000 is :",derivative(100000,10^-15))
```

```
## ==============================================
##  The derivative for x=1 is : 1.110223
##  The derivative for x=10000 is : 0
```

The true value for the function using the function $f(x) = x$ is $f'(x) = \frac{f(x+\epsilon)-f(x)}{\epsilon} = \frac{(x+\epsilon)-x}{\epsilon} = 1$ is always constant with value 1.Regarding the result of the derivative function we see that for $x = 100000$ R doesn't take into account the decimals after a specific number of x and rounds the number to the nearest integer which is 100000 due to underflow occurance so the numeretor of the derivative formula becomes 0 leading finally to 0.When instead $x = 1$ the numerator evaluated is 1.1102230246251565e-15 and the devision with epsilon 10^-15 is just discards the last 15 decimals resulting 1.1102230246251565.

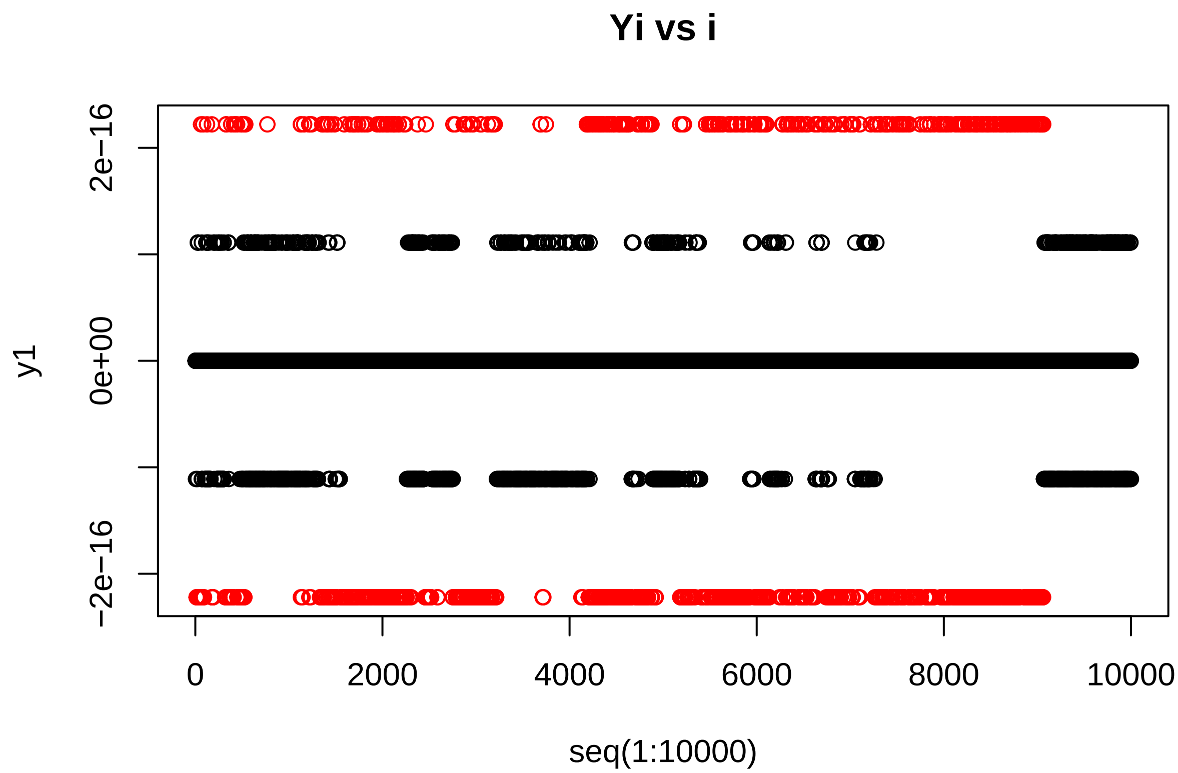
Question 3

```
set.seed(123456)
myvar<-function(vec){

  n<-length(vec)
  variance<-(sum(vec^2)-(sum(vec)^2)/n)/(n-1)
  return(variance)

}

myvec<-rnorm(10000,10^8,1)


y<-double(10000)

for (i in 1:length(myvec)){
  x<-myvec[1:i]
  y[i]<-myvar(x-var(x,na.rm = T))
}

plot(seq(1:10000),y,main="Yi vs i" )
```



The plot above shows the dependence Y_i on i with the formula $Var(x) = \frac{1}{n-1}\left(\sum_{i=1}^{n} x_i^2 - \frac{1}{n}(\sum_{i=1}^{n} x_i)^2\right)$ given. As we can see from the plot we got a lot of curves under and over 0 meaning that we have diffrences in the calculations of the variance using the formula given compared with the var() function. This occurs because if we see the formula the term $\sum_{i=1}^{n} x_i^2$ where we square each value of the vector we tend to lose precision because of arithmetic underflow and all the latter calcucations are affected leading to deviations from the true result.

```
set.seed(12345)
myvar1<-function(v){
  n<-length(v)
  variance<-sum((v-mean(v))^2)/(n-1)
  return(variance)
}


y1<-double(10000)

for (i in 1:length(myvec)){
  x1<-myvec[1:i]
  y1[i]<-myvar1(x1)-var(x1)
}

plot(seq(1:10000),y1,col=ifelse(y1>1.2e-16 | y1< -1.2e-16, "red","black"),
     main="Yi vs i" )
```



The plot above shows the dependence Y_i on i with the formula $Var(x) = \frac{\sum_{i=1}^{n}(x_i-\mu)^2}{n-1}$ where μ is the mean. Using the new formula where we center the points arround the mean we see that we have an improvement in the range of the errors and the deviation of the errors is steady and we can see an upper and a lower band with few errors lie beyond these linear bands represented with red in the plot.Also we can observe that the range of the errors is much smaller with means the formula used almost as good as the var() basic function in R.

Question 4

Using the unscaled data first

```
tecator<-readxl::read_excel("tecator.xls")

tecator<-as.data.frame(tecator)


X<-tecator[,!names(tecator)%in%c("Sample","Protein")]
X$intercept<-1
X<-as.matrix(X)
y<-tecator$Protein

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

try(solve(A,b))
```

```r
cat("The result of solve in the unscaled data is : \n","Error in solve.default(A, b) :
  system is computationally singular: reciprocal condition number = 7.78822e-17")
```

```
## The result of solve in the unscaled data is :
##  Error in solve.default(A, b) :
##   system is computationally singular: reciprocal condition number = 7.78822e-17
```

When we used the unscaled data solve returns an error that the system is computationally singular and we can't solve the linear equation and the function exits.

```r
cat("The value of condition number is :",kappa(A))
```

```
## The value of condition number is : 1.346742e+15
```

Printing the number of kappa for the value of A matrix we see that is very big and that implies that the matrix is said to be ill-conditioned a very small change in matrix A will cause a large error in b and makes the solution unstable.

This happens because the tolerence returned is larger than the default threshold set by the function solve (argument tolerence) so an error returned and we cannot get a solution.The torrelance is related to conditon number by the function $tolerance = \frac{1}{conditionnumber}$ so in our case $tolerance = \frac{1}{kappa(A)} = 7.425326e - 16$ and it is bigger that the threshold of $7.425326e - 17$ that is set by solve function as we see in the printed error resulting the end of execution of the function.

Using the scaled data now

```r
library(knitr)

X1<-as.data.frame(scale(tecator[,!names(tecator)%in%c("Sample","Protein")]))
X1$intercept<-1
X1<-as.matrix(X1)

y1<-tecator$Protein


A1<-t(X1)%*%X1
b1<-t(X1)%*%y1

cat("The result of solve in the scaled data is : \n")
```

```
## The result of solve in the scaled data is :
```

```r
a<-solve(A1,b1)

kable(a,col.names = c("coefficient") ,top.label="Output solve scaled")
```

	coefficient
Channel1	-333.772358
Channel2	-667.261081
Channel3	1140.328097
Channel4	-391.275914
Channel5	1247.152985
Channel6	-240.586840
Channel7	-612.405666
Channel8	249.766533
Channel9	-399.352334

	coefficient
Channel10	771.738979
Channel11	-991.309149
Channel12	-917.524933
Channel13	1882.795303
Channel14	-901.684992
Channel15	122.910023
Channel16	-776.905455
Channel17	510.540926
Channel18	894.939374
Channel19	-980.622563
Channel20	-9.073425
Channel21	1672.935530
Channel22	-4120.880166
Channel23	5612.114025
Channel24	-4272.028011
Channel25	1906.062800
Channel26	-338.001598
Channel27	51.334122
Channel28	-690.612172
Channel29	1340.231072
Channel30	-1802.110398
Channel31	1321.756926
Channel32	950.375437
Channel33	-1055.280644
Channel34	-862.504074
Channel35	1262.728223
Channel36	-238.640566
Channel37	-922.929407
Channel38	857.506529
Channel39	-1313.965854
Channel40	2472.925424
Channel41	-2669.790523
Channel42	979.184570
Channel43	1582.522813
Channel44	-1760.061863
Channel45	-422.852025
Channel46	1741.341147
Channel47	-887.715943
Channel48	-205.361385
Channel49	-272.986941
Channel50	1219.176004
Channel51	-2108.661692
Channel52	3797.657726
Channel53	-5046.106185
Channel54	4483.491188
Channel55	-2450.640297
Channel56	580.698699
Channel57	-99.285353
Channel58	22.248835
Channel59	-267.552168
Channel60	1040.385834
Channel61	-1370.637572

	coefficient
Channel62	1350.331250
Channel63	-595.550474
Channel64	670.721486
Channel65	-1204.430079
Channel66	1100.688386
Channel67	-1107.611458
Channel68	735.836634
Channel69	-230.157669
Channel70	-959.884642
Channel71	988.463914
Channel72	-538.548558
Channel73	359.545861
Channel74	1342.772857
Channel75	-60.372151
Channel76	-1938.978887
Channel77	1114.608556
Channel78	-225.953318
Channel79	-70.848214
Channel80	-2041.879713
Channel81	3057.273399
Channel82	-2684.134845
Channel83	1215.744850
Channel84	1279.331406
Channel85	-2416.655126
Channel86	1975.970793
Channel87	1988.549053
Channel88	-6488.389705
Channel89	5043.324968
Channel90	901.077611
Channel91	-1002.061440
Channel92	-1470.239895
Channel93	840.532795
Channel94	608.353465
Channel95	-1838.695602
Channel96	1705.288725
Channel97	-402.247403
Channel98	-1110.167307
Channel99	718.579767
Channel100	74.336670
Fat	-5.027733
Moisture	-2.817918
intercept	17.682791

Using the scaled data we where able to solve the linear system and get coefficients for every feature value.

```
cat("The value of condition number is :",kappa(A1))
```

```
## The value of condition number is : 490471518993
```

Printing the number of kappa again we can see that is still high but much less that the previous used with the unscaled data and we where able to solve the linear system and get coefficient values.

When we scale the data we see that the linear system did not get any better or worse the linear dependences of the column features are still present but we manage to make the value of condition number smaller with scaling.This is happening because If we look at the definition of the condition number $k(A) = ||A|| * ||A^-1||$ and just by making the range of the columns smaller the magnitude got smaller leading to a smaller value of condition number which is below threshold value of solve function and we manage to get the solution.The tolerence now is $tolerance = \frac{1}{kappa(A1)} = \frac{1}{490471518993} = 2.038854e - 12$ which is smaller than the default $7.425326e - 17$ set by solve so now we are able to get a solution.