Variance Approximation

Sreenand Sasikumar

1/29/2020

Question 1: Be careful when comparing:

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

The code above is giving "Subtraction is wrong" because the computation compares all the decimal points. Both the values has non terminating quotients.

Here x1-x2 = 0.083333333333333315

1/12 = 0.083333333333333329

We can see that after 15 decimal points there is some difference.

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

The code above is giving "Subtraction is correct" because the resulting values from x1-x2 and 1\2 terminated after one decimal point.

To avoid the computational errors like in the first snippet while performing in r we could use system function "all.equal()".

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

Question 2: Derivative

```
## [1] 1.1102230246251565
```

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

Question 2.3:

For x = 1 the derivative is : 1.1102230246251565

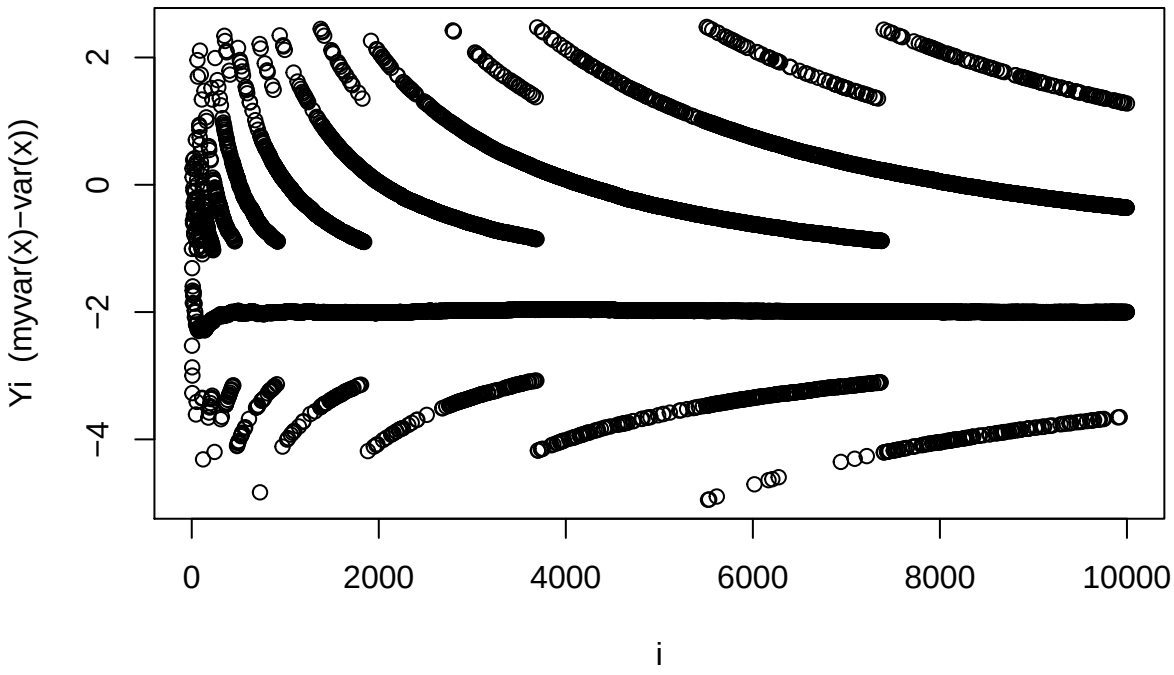
For x = 100000 the derivative is : 0

The true values for the computation of both the derivatives should be : 1

In the case of 1 the tail part after decimal is not neglected as 1 has a very little magnitude.(the left part of the decimal).

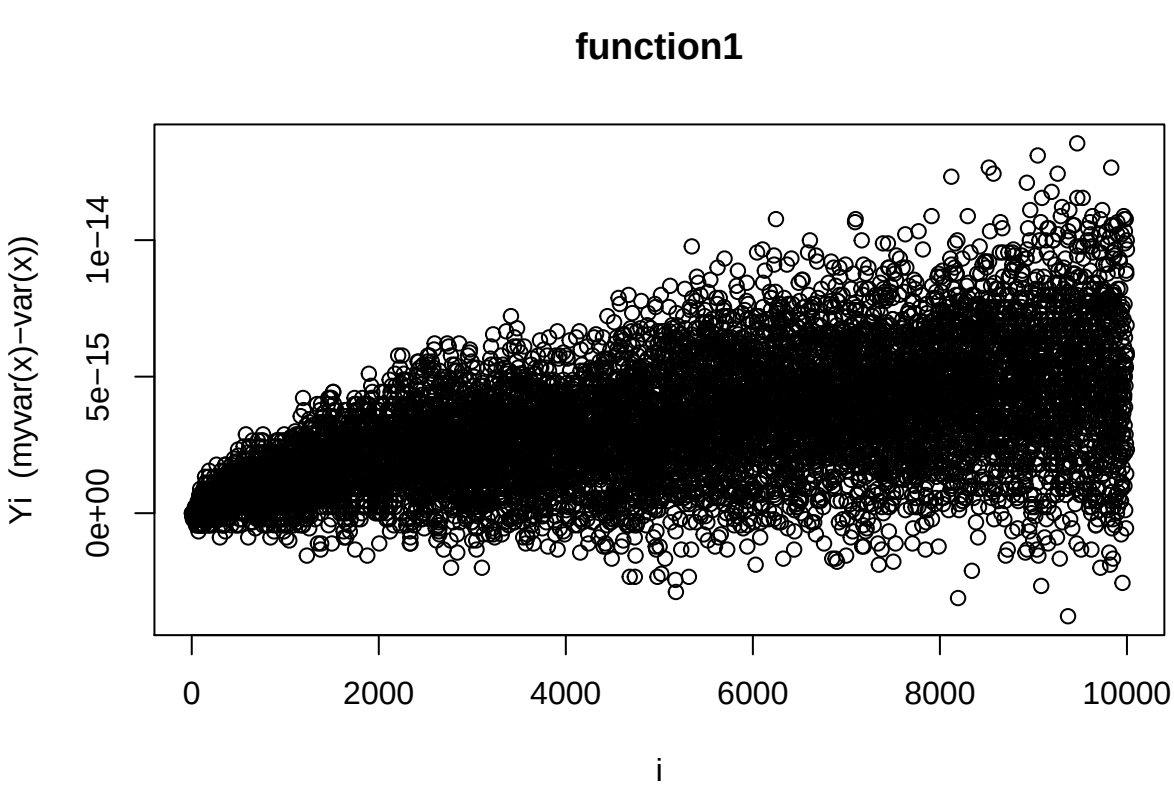
Unlike the case for one 100000 is a large number and the tail part is neglected while subtracting 100000 from it, so there we are getting zero in the numerator resulting the overall zero.

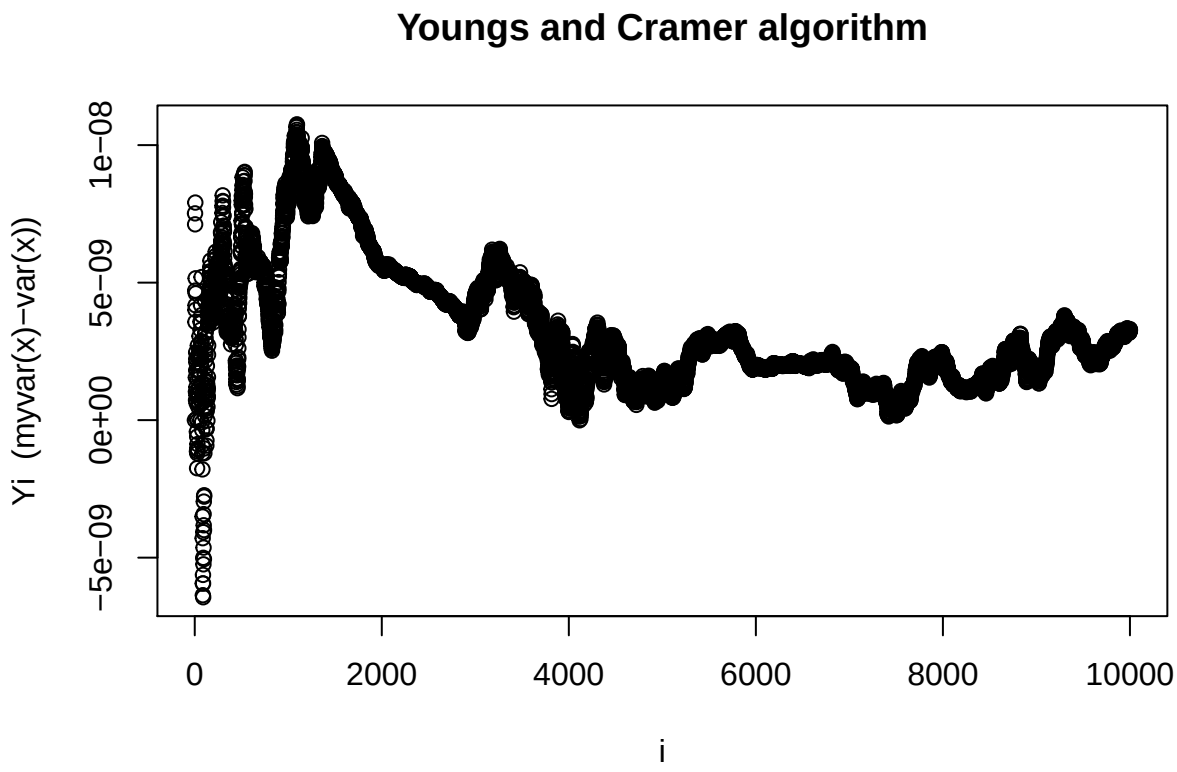
Question 3: Variance



From the above plot we can say that the values obtained by both the functions are so different. we can observe a difference of +2 to -4.

So we can say that the user defined function myvar is not giving the valid results for the variance.


function1
Yi (myvar(x)−var(x))
0e+00
5e−15
1e−14
0
2000
4000
6000
8000
10000
i



Above mentioned are two variance finding functions which are giving the similar results as system function var(). Here we can observe that a very fraction of positive diiference is observed while we take these functions. So we can use these improved functions instead of the myvar() function.

Question 4: Linear Algebra

"system is computationally singular: reciprocal condition number = 7.78804e-17" error is given when tried to calculate the coefficients. The above error could be becuase, the system is treating the A matrix as singular which cannot be invertible in general. The reason could be the linear dependency of multiple variables or values of variable which are changing drastically with respect to each other.

```
## The conditional number for unscaled matrix A is found to be 852351692132074.88
```

kappa function outputs the condition number of product of norm of matrix with it's inverse. Since in the above case there seems to be a singularity in the inverse of A matrix, kappa is resulting in a very high condition number. We can infer that higher the conditional number more prone the matrix to be computationally singular.

```
##                          Protein
## Channel1    -110.61236724670743570
## Channel2    -221.28735642130777705
## Channel3     378.11936505276162279
## Channel4    -129.72930226498283446
## Channel5     413.31779023151466390
## Channel6     -79.60815564106451347
## Channel7    -203.08049585833214223
## Channel8      82.82657189320889302
## Channel9    -132.42689397116191685
```

```
## Channel10    255.84531732433606521
## Channel11   -328.55375763858319260
## Channel12   -304.28247566468780860
## Channel13    624.28100786412414891
## Channel14   -299.01998452872066991
## Channel15     40.82831957508460619
## Channel16   -257.60269072463415796
## Channel17    169.28450858936412260
## Channel18    296.64227791831945069
## Channel19   -325.06039850622983067
## Channel20     -3.00615044269943610
## Channel21    554.55619217338971794
## Channel22  -1366.03068839787738398
## Channel23   1860.37125832579476992
## Channel24  -1416.15085335179537651
## Channel25    631.85070172154519241
## Channel26   -112.04301426610618364
## Channel27     17.00582923990441486
## Channel28   -228.91699688616790809
## Channel29    444.26528336486080661
## Channel30   -597.37719733979611192
## Channel31    438.14212373719783500
## Channel32    315.04391677968669683
## Channel33   -349.81286282234941609
## Channel34   -285.91300974338082597
## Channel35    418.57943911966867745
## Channel36    -79.10660853612353094
## Channel37   -305.93789922552241478
## Channel38    284.25248303770786151
## Channel39   -435.56960233591962606
## Channel40    819.75667005154537037
## Channel41   -885.01287086447700858
## Channel42    324.58977989817503840
## Channel43    524.58936517138499767
## Channel44   -583.43830385044748255
## Channel45   -140.17674487119074911
## Channel46    577.24094238615361974
## Channel47   -294.27028463303577155
## Channel48    -68.07518710906151682
## Channel49    -90.49277757063100580
## Channel50    404.14626849452906754
## Channel51   -699.00303470273502171
## Channel52   1258.88884567108470947
## Channel53  -1672.73745203211728949
## Channel54   1486.23595786531222984
## Channel55   -812.36473333551839460
## Channel56    192.49586283903045114
## Channel57    -32.91087422363489168
## Channel58      7.37394913252501283
## Channel59    -88.68965423779445700
## Channel60    344.87640252841083566
## Channel61   -454.35188896766976541
## Channel62    447.62035732236108743
## Channel63   -197.41809717740397900
```

```
## Channel64    222.33665130467852578
## Channel65   -399.25648038413783070
## Channel66    364.86827825417276472
## Channel67   -367.16351761008263566
## Channel68    243.92384879683959298
## Channel69    -76.29557454999303445
## Channel70   -318.19184864807175472
## Channel71    327.66564283240586519
## Channel72   -178.52323821029858664
## Channel73    119.18538794049527496
## Channel74    445.11553553654880488
## Channel75    -20.01311795831861673
## Channel76   -642.75088840017269831
## Channel77    369.48107256976072676
## Channel78    -74.90131779272633139
## Channel79    -23.48536541772773489
## Channel80   -676.86150594932041713
## Channel81   1013.45374102075584233
## Channel82   -889.76227763647329994
## Channel83    403.00657931517343968
## Channel84    424.08480369417520706
## Channel85   -801.09560823735955637
## Channel86    655.01341977930860594
## Channel87    659.18297365130274557
## Channel88  -2150.83255651211948134
## Channel89   1671.80887839311617427
## Channel90    298.69771102262893692
## Channel91   -332.17278103512944654
## Channel92   -487.36897021322511137
## Channel93    278.62773509544786066
## Channel94    201.66273255954729393
## Channel95   -609.50814182934118435
## Channel96    565.28517543166526593
## Channel97   -133.34075568348634988
## Channel98   -368.00872869545128196
## Channel99    238.20159905504260678
## Channel100    24.64181810706941178
## Fat           -1.66664028488980875
## Moisture      -0.93410994868816033
```

```
## The conditional number for scaled matrix A is found to be 490471520662.05011
```

The conditional number after scaling the data has reduced siginificantly and hence the inverse of A_sc could be calculated by the system. The reason is the raw data containing many features are of different units with respect to each other and the product yeilds in a big number which system cannot handle. Hence the Inverse of A could not be found since there was a very big condition number. Once the data has been scaled, all the feature values are made uniform and inverse of the A matrix could be handled by the system and also the linear dependency between the variables is more siginificant since there is no drastic change in the values of variables as it is restricted within 1.

Appendix

```
knitr::opts_chunk$set(echo = TRUE,warning = FALSE)
# Question 1: Be careful when comparing
```

```r
options(digits = 20)
x1<-1/3
x2<-1/4

if(x1-x2 == 1/12)
{
  print("Subtraction is correct")
} else
{
  print("Subtraction is wrong")
}

# In the first snippet, when the Target value and Current values are compared upto 22 decimal points, t

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

x1<-1/3
x2<-1/4

if(all.equal(x1-x2 , 1/12))
{
  print("Subtraction is correct")
} else
{
  print("Subtraction is wrong")
}
userderivative<-function(x)
{
  epsillon<-10^(-15)
  deriv <- ((x+epsillon)-x)/epsillon
  deriv
}
userderivative(1)
userderivative(100000)


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


set.seed(12345)
x <- rnorm(10000,mean = 10^8, sd = 1)
```

```r
Yi <- vector()
for(i in 1:length(x))
{
  Yi[i] <- myvar(x[1:i]) - var(x[1:i])
}

plot(seq(1:length(x)),Yi,xlab = "i",ylab = "Yi  (myvar(x)-var(x))")



improvedvar<-function(x)
{
  sum<-0
  mean<-mean(x)
  for (i in 1:length(x))
  {
    sum<-sum + (x[i]-mean)^2

  }
  sum<-sum/(length(x)-1)
 return(sum)

}
Yi <- vector()
for(i in 1:length(x))
{
  Yi[i] <- improvedvar(x[1:i]) - var(x[1:i])
}

plot(seq(1:length(x)),Yi,xlab = "i",ylab = "Yi  (myvar(x)-var(x))",main = "function1")



var_YC<-function(v_x){
  ## v_x is a numerical vector of length greater than 2
  ## this function calculates the sample variance
  ## using the Youngs and Cramer algorithm
  T<-v_x[1]
  RSS<-0
  n<-length(v_x)
  for (j in 2:n){
    T<-T+v_x[j]
    RSS<-RSS+((j*v_x[j]-T)^2)/(j*(j-1))
  }
  RSS/(n-1)
}

Yi <- vector()
for(i in 1:length(x))
{
```

```
  Yi[i] <- var_YC(x[1:i]) - var(x[1:i])
}

plot(seq(1:length(x)),Yi,xlab = "i",ylab = "Yi  (myvar(x)-var(x))",main ="Youngs and Cramer algorithm" )



set.seed(12345)

tec_data <- readxl::read_excel(file.choose())
X <- as.matrix(tec_data[,c(-1,-103)])
Y <- as.matrix(tec_data[,c(103)])
X <- cbind(1,X)

A <- t(X) %*% X
b_vec <- t(X) %*% Y

#beta <- solve(A) %*% b_vec

cat("The conditional number for unscaled matrix A is found to be",kappa(A))
# Scaled data
X <- as.matrix(tec_data[,c(-1,-103)])
X_scale <- scale(X)
Y <- as.matrix(tec_data[,c(103)])
Y_scale <- scale(Y)
X <- cbind(1,X)


A_sc <- t(X_scale) %*% X_scale
b_vec_sc <- t(X_scale) %*% Y_scale

beta_sc <- solve(A_sc) %*% b_vec_sc
beta_sc

cat("The conditional number for scaled matrix A is found to be",kappa(A_sc))
```