Group 12 Computational statistics Lab 02 Report

Karthikeyan Devarajan - Karde799

Question 1: Be careful when comparing

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

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

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

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

For the first snippet, result is “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"
```

For the second snippet, result is “Substraction is correct”.

For the first snippet, X1 value is 1/3. But when we check the value of 1/3 for 22 digits, we got something like this :-

```
x1 <- 1/3
options(digits = 22)
print(x1)
```

```
## [1] 0.33333333333333331
```

So R is going to show 1/3 is equal to 0.333333333333333148296. So this is not true. Since Floats are rounded, so usual mathematical laws do not hold.

But for the second snippet X1 and X2 values do not have any differnce since X1 and X2 values do not effect when the Floats are rounded for 22 digits.

2. If there are any problems, suggest improvements.

We could use R in build function *all.equal*(). This Function is a utility to compare R objects x and y testing 'near equality'. If they are different, comparison is still made to some extent, and a report of the differences is returned. [Reference - R Document]

```
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"
```

Question 2: Derivative

From the defintion of a derivative :-

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

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

Here is the function for the Derivative for $f(x) = x$ where $\epsilon = 10^{-15}$

```
getDer <- function(x) {
  h = 10 ^ -15
  f_dash = ((x + h) - x) / h
  options(digits = 22)
  cat("x + h is ",  x + h , "\n")
  cat("(x + h) - x is ", ((x + h) - x), "\n")
  cat("Derivative is ", f_dash, "\n")
}
```

2. Evaluate your derivative function at x = 1 and x = 100000.

```
getDer(1)
```

```
## x + h is  1.0000000000000011
## (x + h) - x is  1.1102230246251565e-15
## Derivative is  1.1102230246251565
```

```
getDer(100000)
```

```
## x + h is  1e+05
## (x + h) - x is  0
## Derivative is  0
```

What values did you obtain?

We have obtained 1.110223024625156540424 for the $x = 1$ and 0 for $x = 100000$.

What are the true values?

Actually true values should be 1 for both $x = 1$ and $x = 100000$ cases. We could get this value when we solve this equation by mathematically.

Explain the reasons behind the discovered differences.

For the $x = 1$ case :-

Since $1 + 10^{-15}$ is rounded to the possbile nearest value which is 1.000000000000001110223. True value should be 1.000000000000001000000. So there is an error when we do the substraction $(x + h) - x$. That's why we get 1.110223024625156540424 rather than 1

For the $x = 100000$ case :-

Since $100000 + 10^{-15}$ is rounded to the possbile nearest value which is 100000. Decimal value is so small when comparing to the 100000 value. Hence deciaml value is ignored. Therefore the result for the $(x+h)-x$ is 0. and when we devide 0 by any value we get the 0 as the answer.

Question 3: Variance

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

```
myvar <- function(x) {
  xi_squared = sum(x ^ 2)
  x_squared = (1 / (length(x))) * ((sum(x)) ^ 2)
  val = (1 / (length(x) - 1)) * (xi_squared - x_squared)
  return(val)
}
```

2. Generate a vector x = (x1, . . . , x10000) with 10000 random numbers with mean 10^8 and variance 1.

```
RNGversion(min(as.character(getRversion()),"3.6.2"))
set.seed(12345, kind = "Mersenne-Twister", normal.kind = "Inversion")
x = rnorm(10000, mean = 10 ^ 8,sd = 1)
```

3. For each subset Xi = {x1,. . . ,xi},i=1,. . . ,10000 compute the difference Yi = myvar(Xi) - var(Xi), where var(Xi) is the standard variance estimation function in R. Plot the dependence Yi on i. Draw conclusions from this plot. How well does your function work? Can you explain the behaviour?

```
Yi = c()

for (index in 1:10000) {
  Yi[index] = myvar(x[1:index]) - var(x[1:index])
```

```
}

cat("Head Data")
```

```
## Head Data
```

```
print(head(Yi, n = 10))
```

```
##  [1]                    NaN -0.0076802134803717337 -2.1947558863931986295
##  [4] -0.3099219255562846720 -2.2681999862375241683 -0.9394953144239626130
##  [7] -2.1882761193457036519  1.5418321346129513216  3.3410979515208438784
## [10] -2.4398129474863226029
```

```
cat("\n")
```

```
cat("Tail Data")
```

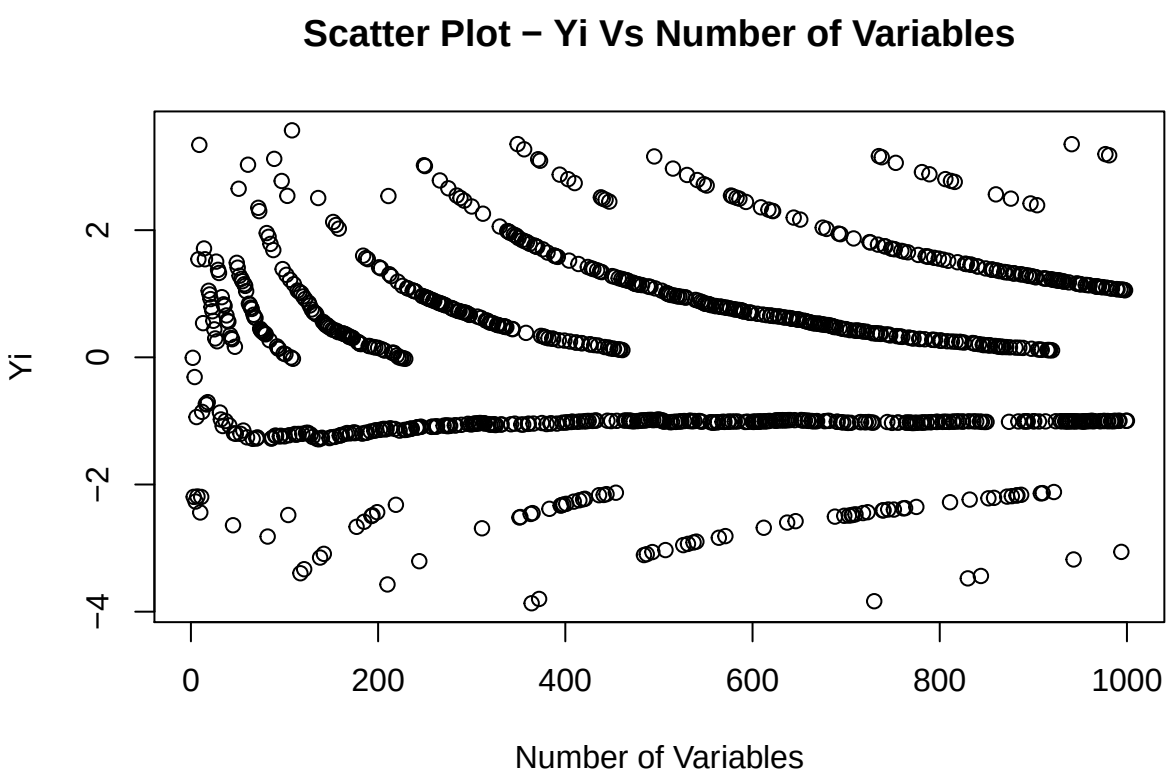
```
## Tail Data
```

```
print(tail(Yi, n = 10))
```

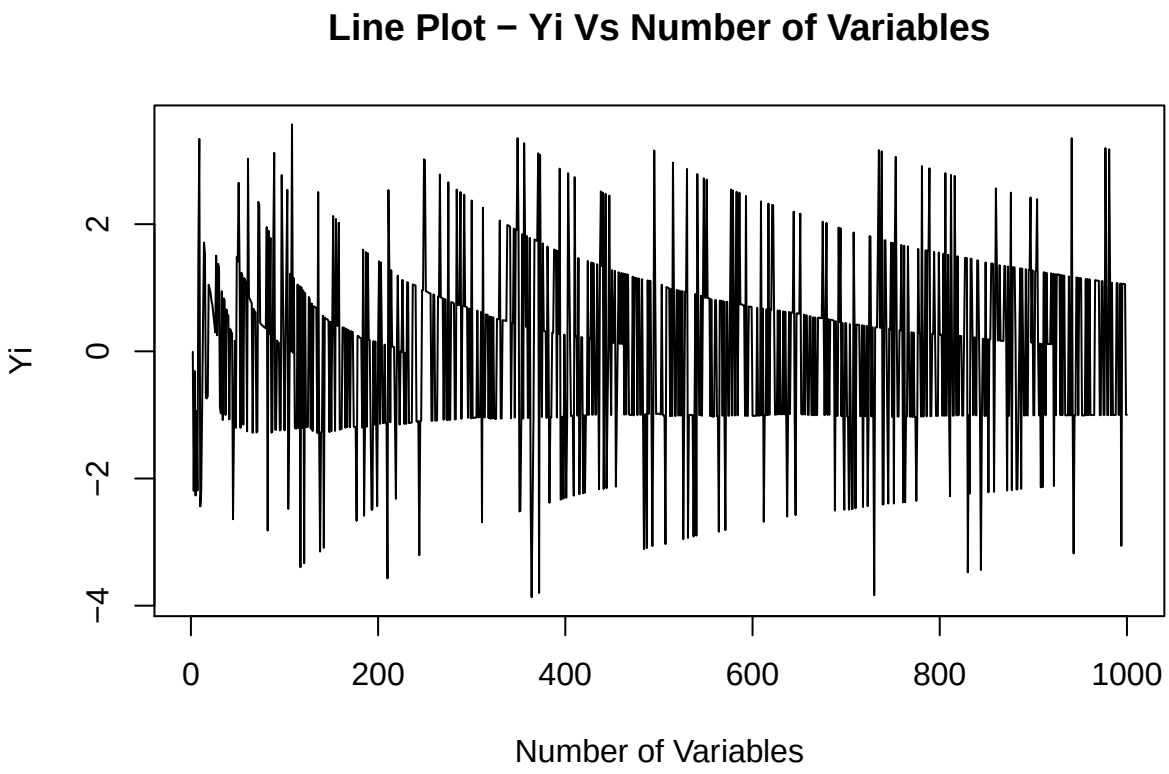
```
##  [1]  0.64035816569026816 -0.99970628355391822 -0.99960780780095226
##  [4]  0.64002312173147335  0.63994098416296485 -0.99953607866712979
##  [7]  0.63937659969458616 -0.99964782429781407 -0.99979985373136504
## [10]  0.63886395715482058
```

```
cat("\n")
```

```
plot(1:1000,
     Yi[1:1000],
     type = 'p',
     xlab = "Number of Variables",
     ylab = "Yi",
     main = "Scatter Plot - Yi Vs Number of Variables")
```




```
plot(1:1000,
     Yi[1:1000],
     type = 'l',
     xlab = "Number of Variables",
     ylab = "Yi",
     main = "Line Plot - Yi Vs Number of Variables")
```



Difference Y_i values are in between -4 to 3. Hence *myvar* function do not have a high accurancy. But when comparing to the vector values with Y_i it's really small error. We are dealing with large numbers in this function.

Adding two large numbers the sign bit can be treated as a high order bit and on some architectures results in a negative number. Hence Overflow would be the reason behind this behaviour.

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

We can use below equation in order to get the variance. We can remove the factors such as squared value of sample value, therfore we could remove the effect of overflows from the equation.

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

```
myvarOpt <- function(x) {

  total = 0
  for (i in 1:length(x)) {
    total = total + x[i]
  }
  vectMean = total / length(x)

  totalDif = 0

  for (index in 1:length(x)) {
```

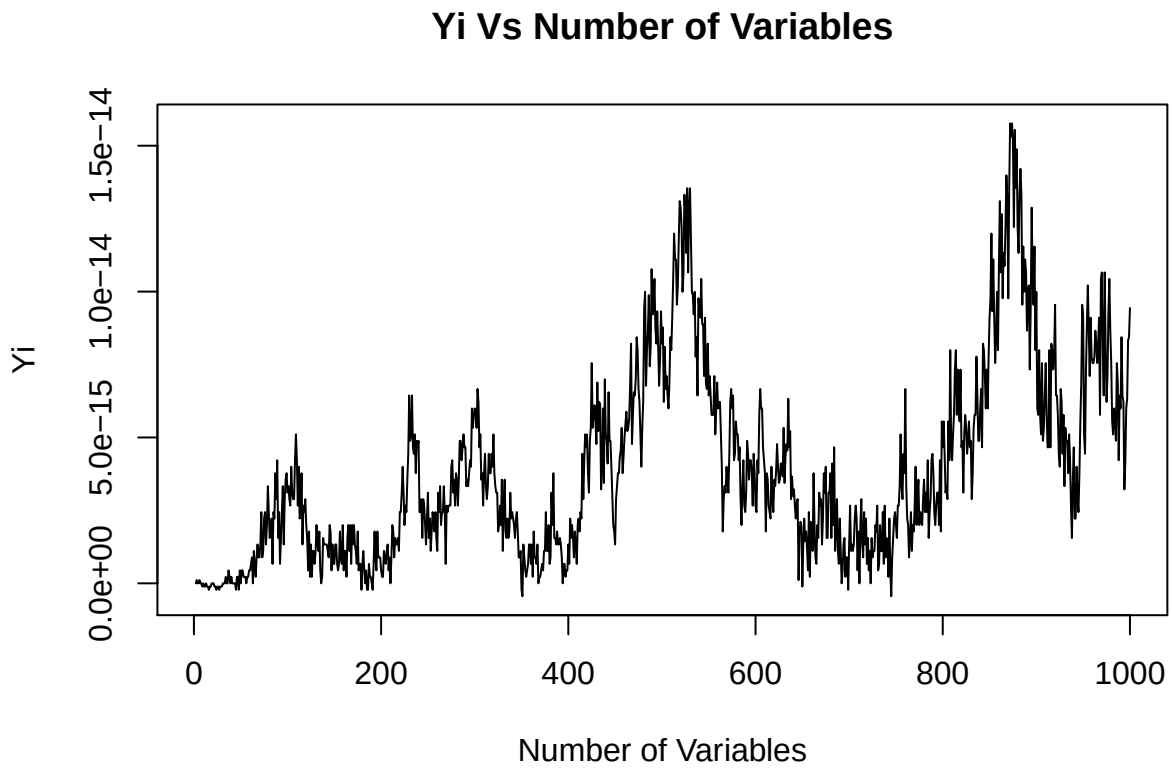
```
    totalDif = totalDif + ((x[index] - vectMean) ^ 2)
  }

  return(totalDif / (length(x) - 1))
}

Yi_Optimal = c()

for (index in 1:10000) {
  Yi_Optimal[index] = myvarOpt(x[1:index]) - var(x[1:index])
}

plot(1:1000,
     Yi_Optimal[1:1000],
     type = 'l',
     xlab = "Number of Variables",
     ylab = "Yi",
     main = "Yi Vs Number of Variables")
```



Difference between $var()$ and $myvar$ values are significantly low. All the values are in between 0 to $1.5 * e-14$. This is a really small value. New equation has a really good accuracy for low number of variables. Accuracy is getting lower(slightly low) when the number of variables are increasing. Hence we could conclude that this function generates same results as $var()$.

Question 4: Linear Algebra

1. Import the data set to R

```
library(readxl)
mydata = read_excel(file.choose())
```

2. Question

```
# Load all data and remove Protein Data
independentData = mydata
independentData$Protein = c()
independentData$Sample = c()

dataLength = nrow(mydata)
dataMatrix = cbind(as.matrix(c(rep(1,dataLength))),
                   independentData)

# Calculate A and b

A_matrix = t(as.matrix(dataMatrix)) %*% as.matrix(dataMatrix)
b_matrix = t(as.matrix(dataMatrix)) %*% as.matrix(mydata$Protein)
```

3. Question

```
beta = solve(A_matrix) %*% b_matrix
```

What kind of result did you get? How can this result be explained ?

" This calculation shows below error:- Error in solve.default(A_matrix) : system is computationally singular: reciprocal condition number = 7.78804e-17 "

This means that design matrix is not invertible , therfore we can't use matrix to develop a regression model. This is because of strongly correlated variables.

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

```
k = kappa(A_matrix)
#print(k)
cat("Kappa(A) = ", k)
```

```
## Kappa(A) =  852351692132074.88
```

What is kappa ?

The condition number of a regular (square) matrix is the product of the norm of the matrix and the norm of its inverse (or pseudo-inverse), and hence depends on the kind of matrix-norm.

kappa() computes by default (an estimate of) the 2-norm condition number of a matrix or of the R matrix of a QR decomposition, perhaps of a linear fit. The 2-norm condition number can be shown to be the ratio of the largest to the smallest non-zero singular value of the matrix. - [R Documentation Help]

So there is a huge difference between largest and smallest values in the data matrix. 100 channel spectrum is calculated by -log10 of the transmittance. The moisture, fat and protein are determined by analytic chemistry. The units of two different features are different, and arbitrary. This is like the case which something gets more influence than it should.

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

```
# Scaled Data

scaledMydata = read_excel(file.choose())
scaledMydata$Protein = c()
scaledMydata$Sample = c()
scaledMydata = scale(scaledMydata)

scaledDataMatrix = cbind(as.matrix(c(rep(1,dataLength))),
                         scaledMydata)


scaled_A_matrix = t(as.matrix(scaledDataMatrix)) %*% as.matrix(scaledDataMatrix)
scaled_b_matrix = t(as.matrix(scaledDataMatrix)) %*% as.matrix(scale(mydata$Protein))

beta = solve(scaled_A_matrix) %*% scaled_b_matrix

cat("Kappa(A) = ", kappa(scaled_A_matrix))
```

```
## Kappa(A) =  490471520661.25275
```

```
print(beta)
```

```
##                                [,1]
##            -1.8730626899087354e-12
## Channel1   -1.1061236724670744e+02
## Channel2   -2.2128735642130778e+02
## Channel3    3.7811936505276162e+02
## Channel4   -1.2972930226498283e+02
## Channel5    4.1331779023151466e+02
## Channel6   -7.9608155641064513e+01
## Channel7   -2.0308049585833214e+02
## Channel8    8.2826571893208893e+01
## Channel9   -1.3242689397116192e+02
## Channel10   2.5584531732433607e+02
## Channel11  -3.2855375763858319e+02
## Channel12  -3.0428247566468781e+02
## Channel13   6.2428100786412415e+02
## Channel14  -2.9901998452872067e+02
## Channel15   4.0828319575084606e+01
## Channel16  -2.5760269072463416e+02
## Channel17   1.6928450858936412e+02
## Channel18   2.9664227791831945e+02
## Channel19  -3.2506039850622983e+02
## Channel20  -3.0061504426994361e+00
## Channel21   5.5455619217338972e+02
```

```
## Channel22  -1.3660306883978774e+03
## Channel23   1.8603712583257948e+03
## Channel24  -1.4161508533517954e+03
## Channel25   6.3185070172154519e+02
## Channel26  -1.1204301426610618e+02
## Channel27   1.7005829239904415e+01
## Channel28  -2.2891699688616791e+02
## Channel29   4.4426528336486081e+02
## Channel30  -5.9737719733979611e+02
## Channel31   4.3814212373719784e+02
## Channel32   3.1504391677968670e+02
## Channel33  -3.4981286282234942e+02
## Channel34  -2.8591300974338083e+02
## Channel35   4.1857943911966868e+02
## Channel36  -7.9106608536123531e+01
## Channel37  -3.0593789922552241e+02
## Channel38   2.8425248303770786e+02
## Channel39  -4.3556960233591963e+02
## Channel40   8.1975667005154537e+02
## Channel41  -8.8501287086447701e+02
## Channel42   3.2458977989817504e+02
## Channel43   5.2458936517138500e+02
## Channel44  -5.8343830385044748e+02
## Channel45  -1.4017674487119075e+02
## Channel46   5.7724094238615362e+02
## Channel47  -2.9427028463303577e+02
## Channel48  -6.8075187109061517e+01
## Channel49  -9.0492777570631006e+01
## Channel50   4.0414626849452907e+02
## Channel51  -6.9900303470273502e+02
## Channel52   1.2588888456710847e+03
## Channel53  -1.6727374520321173e+03
## Channel54   1.4862359578653122e+03
## Channel55  -8.1236473333551839e+02
## Channel56   1.9249586283903045e+02
## Channel57  -3.2910874223634892e+01
## Channel58   7.3739491325250128e+00
## Channel59  -8.8689654237794457e+01
## Channel60   3.4487640252841084e+02
## Channel61  -4.5435188896766977e+02
## Channel62   4.4762035732236109e+02
## Channel63  -1.9741809717740398e+02
## Channel64   2.2233665130467853e+02
## Channel65  -3.9925648038413783e+02
## Channel66   3.6486827825417276e+02
## Channel67  -3.6716351761008264e+02
## Channel68   2.4392384879683959e+02
## Channel69  -7.6295574549993034e+01
## Channel70  -3.1819184864807175e+02
## Channel71   3.2766564283240587e+02
## Channel72  -1.7852323821029859e+02
## Channel73   1.1918538794049527e+02
## Channel74   4.4511553553654880e+02
## Channel75  -2.0013117958318617e+01
```

```
## Channel76  -6.4275088840017270e+02
## Channel77   3.6948107256976073e+02
## Channel78  -7.4901317792726331e+01
## Channel79  -2.3485365417727735e+01
## Channel80  -6.7686150594932042e+02
## Channel81   1.0134537410207558e+03
## Channel82  -8.8976227763647330e+02
## Channel83   4.0300657931517344e+02
## Channel84   4.2408480369417521e+02
## Channel85  -8.0109560823735956e+02
## Channel86   6.5501341977930861e+02
## Channel87   6.5918297365130275e+02
## Channel88  -2.1508325565121195e+03
## Channel89   1.6718088783931162e+03
## Channel90   2.9869771102262894e+02
## Channel91  -3.3217278103512945e+02
## Channel92  -4.8736897021322511e+02
## Channel93   2.7862773509544786e+02
## Channel94   2.0166273255954729e+02
## Channel95  -6.0950814182934118e+02
## Channel96   5.6528517543166527e+02
## Channel97  -1.3334075568348635e+02
## Channel98  -3.6800872869545128e+02
## Channel99   2.3820159905504261e+02
## Channel100  2.4641818107069412e+01
## Fat        -1.6666402848898088e+00
## Moisture   -9.3410994868816033e-01
```

By scaling the all column values we could set a common scale and we could reduced the ratio of the largest to the smallest value of the matrix.
Appendix

```
knitr::opts_chunk$set(echo = TRUE)
x1 <- 1/3
x2 <- 1/4

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

if ((x1 - x2) == 1/2) {
  print("Substraction is correct")
} else {
  print("Substraction is wrong")
}
x1 <- 1/3
options(digits = 22)
print(x1)

x1 <- 1/3
```

```r
x2 <- 1/4

if (isTRUE(all.equal((x1 - x2), (1/12)))) {
  print("Substraction is correct")
} else {
  print("Substraction is wrong")
}
getDer <- function(x) {
  h = 10 ^ -15
  f_dash = ((x + h) - x) / h
  options(digits = 22)
  cat("x + h is ",  x + h , "\n")
  cat("(x + h) - x is ", ((x + h) - x), "\n")
  cat("Derivative is ", f_dash, "\n")
}
getDer(1)
getDer(100000)
myvar <- function(x) {
  xi_squared = sum(x ^ 2)
  x_squared = (1 / (length(x))) * ((sum(x)) ^ 2)
  val = (1 / (length(x) - 1)) * (xi_squared - x_squared)
  return(val)
}

RNGversion(min(as.character(getRversion()),"3.6.2"))
set.seed(12345, kind = "Mersenne-Twister", normal.kind = "Inversion")
x = rnorm(10000, mean = 10 ^ 8,sd = 1)

Yi = c()

for (index in 1:10000) {
  Yi[index] = myvar(x[1:index]) - var(x[1:index])
}

cat("Head Data")
print(head(Yi, n = 10))
cat("\n")
cat("Tail Data")
print(tail(Yi, n = 10))
cat("\n")

plot(1:1000,
     Yi[1:1000],
     type = 'p',
     xlab = "Number of Variables",
     ylab = "Yi",
     main = "Scatter Plot - Yi Vs Number of Variables")

plot(1:1000,
     Yi[1:1000],
     type = 'l',
     xlab = "Number of Variables",
     ylab = "Yi",
```

```
     main = "Line Plot - Yi Vs Number of Variables")
myvarOpt <- function(x) {

  total = 0
  for (i in 1:length(x)) {
    total = total + x[i]
  }
  vectMean = total / length(x)

  totalDif = 0

  for (index in 1:length(x)) {
    totalDif = totalDif + ((x[index] - vectMean) ^ 2)
  }

  return(totalDif / (length(x) - 1))
}

Yi_Optimal = c()

for (index in 1:10000) {
  Yi_Optimal[index] = myvarOpt(x[1:index]) - var(x[1:index])
}

plot(1:1000,
     Yi_Optimal[1:1000],
     type = 'l',
     xlab = "Number of Variables",
     ylab = "Yi",
     main = "Yi Vs Number of Variables")
library(readxl)
mydata = read_excel(file.choose())
# Load all data and remove Protein Data
independentData = mydata
independentData$Protein = c()
independentData$Sample = c()

dataLength = nrow(mydata)
dataMatrix = cbind(as.matrix(c(rep(1,dataLength))),
                   independentData)

# Calculate A and b

A_matrix = t(as.matrix(dataMatrix)) %*% as.matrix(dataMatrix)
b_matrix = t(as.matrix(dataMatrix)) %*% as.matrix(mydata$Protein)
beta = solve(A_matrix) %*% b_matrix
k = kappa(A_matrix)
#print(k)
cat("Kappa(A) = ", k)
# Scaled Data

scaledMydata = read_excel(file.choose())
scaledMydata$Protein = c()
```

```
scaledMydata$Sample = c()
scaledMydata = scale(scaledMydata)

scaledDataMatrix = cbind(as.matrix(c(rep(1,dataLength))),
                         scaledMydata)


scaled_A_matrix = t(as.matrix(scaledDataMatrix)) %*% as.matrix(scaledDataMatrix)
scaled_b_matrix = t(as.matrix(scaledDataMatrix)) %*% as.matrix(scale(mydata$Protein))

beta = solve(scaled_A_matrix) %*% scaled_b_matrix

cat("Kappa(A) = ", kappa(scaled_A_matrix))

print(beta)
```