Group 12 Lab Report

Aswath Mandakath Gopinath, Varun Singapura Ravi

2022-11-21

Question 1

Sub Question 1

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

Answer: In the first code snippet, x1(1/3) is represented as 0.333…(recurring decimal) in floating point system and it has been approximated(underflow) in such a way that the mantissa will hold only 23 3's (as per IEEE754 standard). This results in a value 0.33333333333333331482 96 which is less than the actual value of 1/3 due to underflow. Subtracting 1/4 from this, results in yet another underflow and an approximated value. Doing an exact comparison with between this approximated value and 1/12 will always be false.

In the second code snippet, as the values are represented with finite number of decimal places and as the subtraction also does not lead to underflow, the exact comparison yields the expected result.

Sub Question 2

If there are any problems, suggest improvements.

Answer: When doing comparison of numbers in floating point system close to 0, it is recommended that we use all.equal() instead of an exact comparison. all.equal() will manage correct comaprison of computer approximated values generated due to underflow. See correct implementation in Appendix section.

Question 2

Sub Question 1

Write your own R function to calculate the derivative of f(x) = x with ϵ = 10^-15.

Answer: See Appendix for code

Sub Question 2

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

Answer: For x = 1, the function evaluates the following:

```
## [1] 1.110223
```

For x = 100000, the function evaluates the following:

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

Sub Question 3

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

Answer: The values obtained are shown above.

The true value in both the cases should be 1.

In the first case, since the magnitudes are comparable, adding epsilon to 1 yields a computer approximation for the addition i.e, 1 + (10^-15) = 1.000000000000001110223. This approximation is greater than the expected approximation, i.e. 1.00000000000000100000000000000077629(from 10^-15). Hence, subtracting epsilon from this results in a value $>$ 0, as the first term is slightly greater than the 2nd term, hence numerator $>$ denominator, hence a value greater than 1.

In the second case, x is of a very high magnitude in comparison to epsilon and adding epsilon to it doesn't make a difference. For e.g.

```
100000 + 10^-15 + 10^-15 + 10^-15 == 100000
```

```
## [1] TRUE
```

Question 3

Sub Question 1

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

Answer: See Appendix for code.

Sub Question 2

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

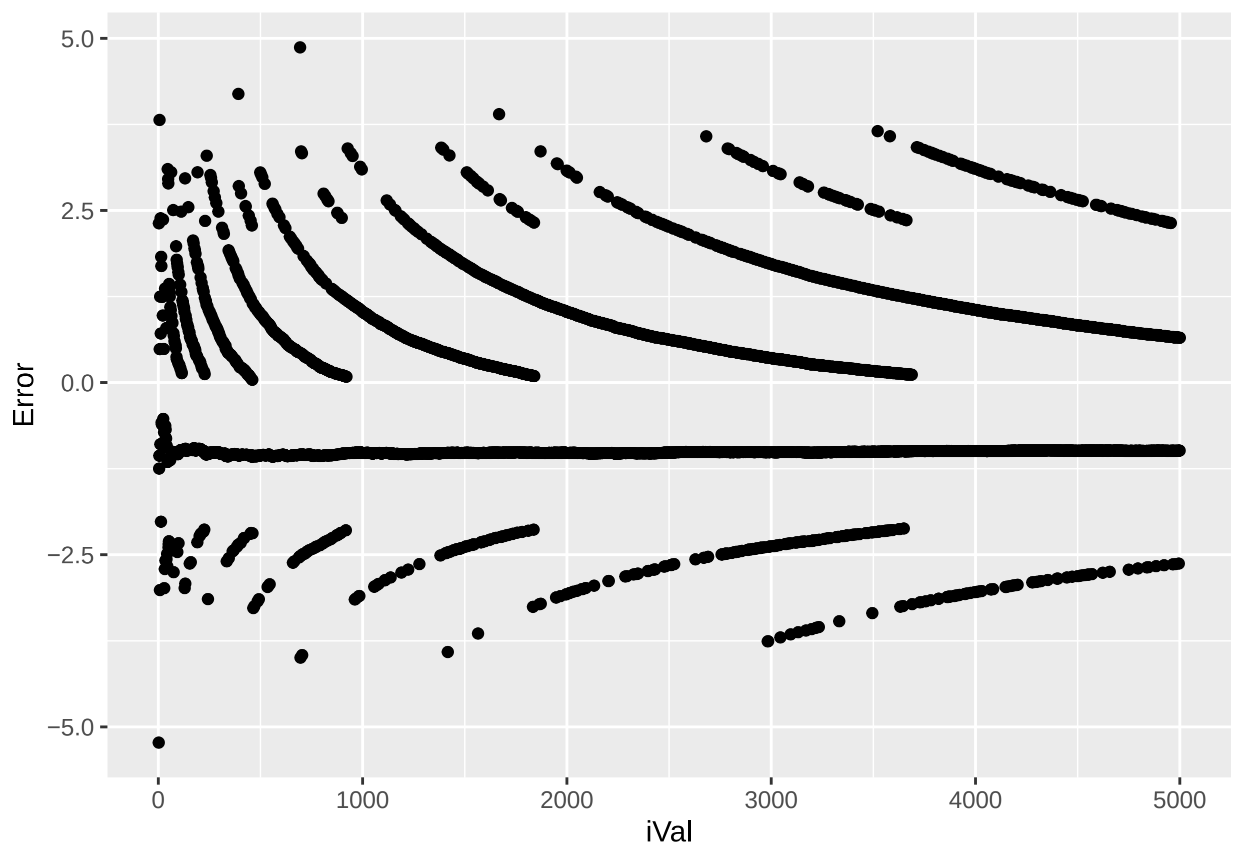
Answer: See Appendix for code.

Sub Question 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?

Answer: As seen in the graph, for all subsets of Xi, the variance is incorrectly calculated. This is because the x values are of high magnitude and sum of square of x values results in high values and the cancellation results in approximated values. Because of this approximation, there is a difference in comparison to the variance calculated using the built-in var().

```
## Warning: Removed 1 rows containing missing values (geom_point).
```



Sub Question 4

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

Answer: See Appendix for code.

The values below are shown for comparing values from a better implementation of the variance estimator with the in-built function for variance in R.

```
## Result of the new implementation for variance calculation: 0.7504276
```

```
## Variance calculated using the inbuilt var(): 0.7504276
```

As seen above, the values are an exact match.

Question 4

Sub Question 1

Even if overflow and underflow would not occur these expressions will not work correctly for all values of n and k. Explain what is the problem in A, B and C respectively.

Answer: For formula A - k can't have 0 value and k can't be equal to n. In these cases the prod results in incorrect results.

For formula B - k cannot be equal to n. This will cause incorrect calculation of prod(1:(n-k))

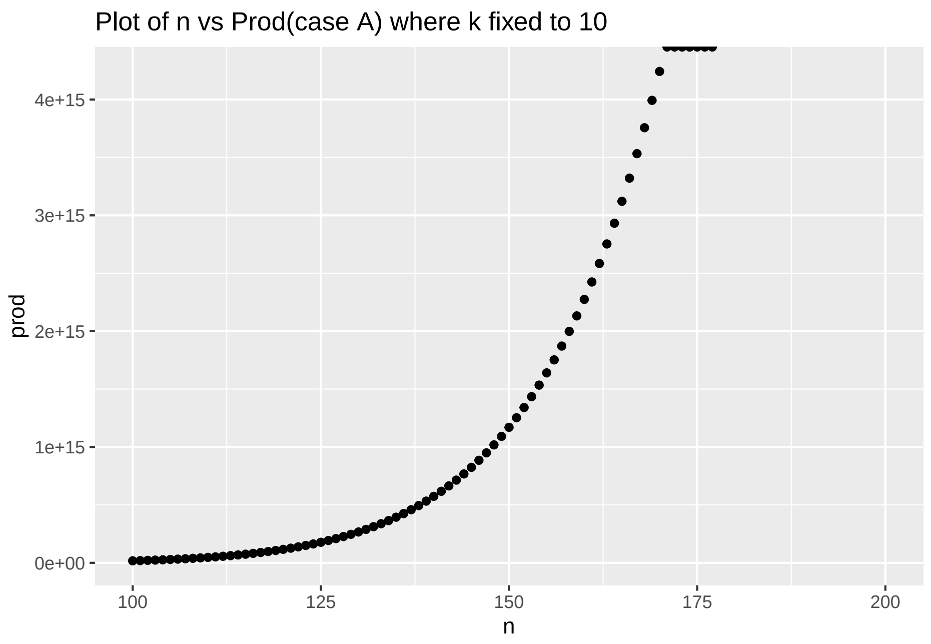
For formula C - In C, we do vector division. This sometimes yields numbers with decimal numbers with recurring floating point. And with the outer prod() we are multiplying these approximated values which finally yields a wrong product.For eg. n = 37, k = 25.

Sub Question 2

In mathematical formulae one should suspect overflow to occur when parameters, here n and k, are large. Experiment numerically with the code of A, B and C, for different values of n and k to see whether overlow occurs. Graphically present the results of your experiments.

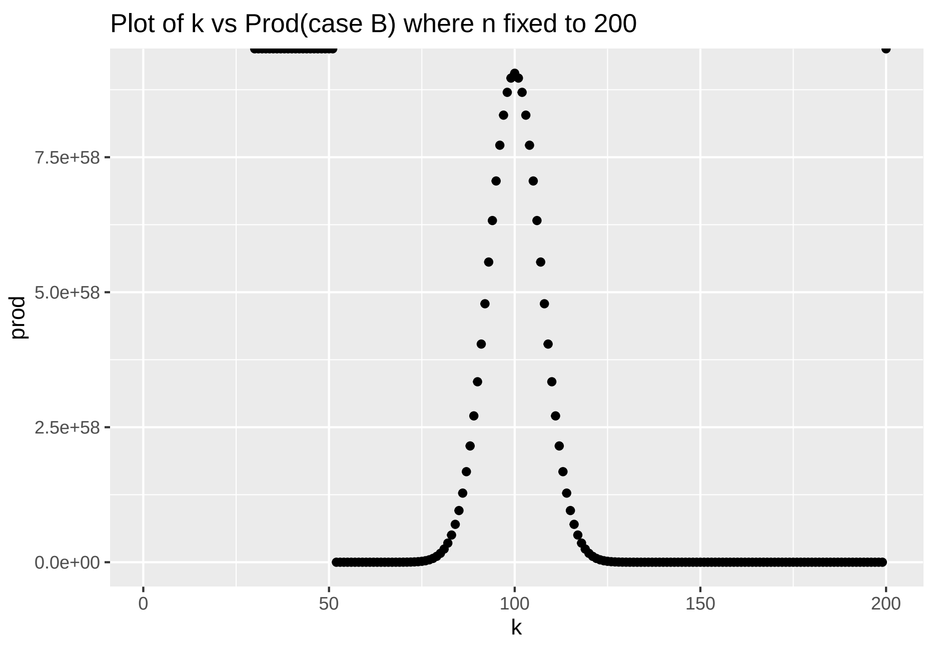
Answer: See Appendix for code.

```
## Warning: Removed 23 rows containing missing values (geom_point).
```

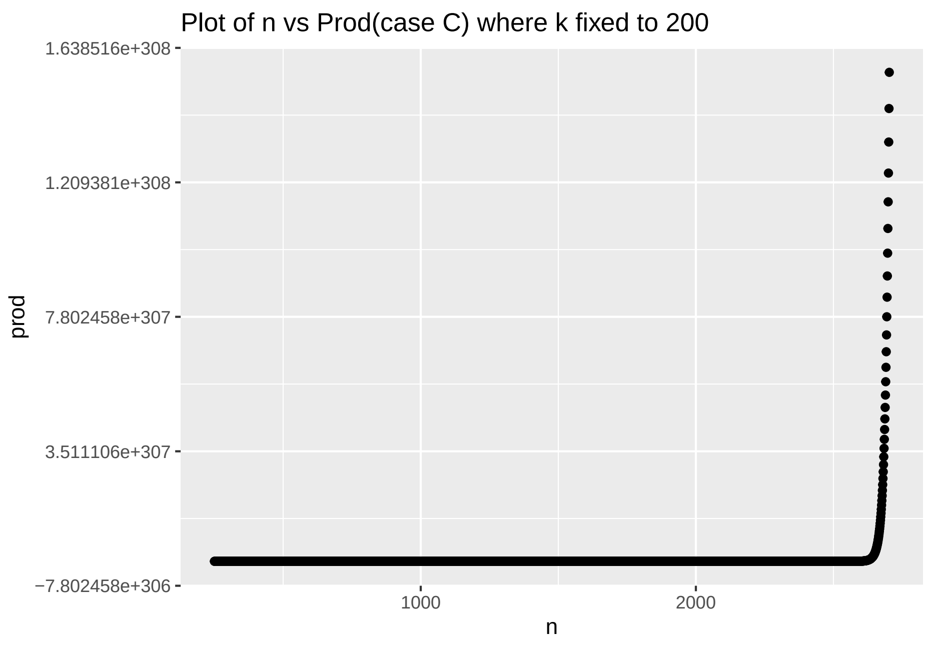


As seen in the above plot for case A, as n is nearing 175, there is overflow.

```
## Warning: Removed 29 rows containing missing values (geom_point).
```



As seen in the above plot for case B, as k decreases from 50 to 0, there is overflow.



As seen from the plot above, for values of n approximately greater than 2703 while keeping k fixed to 200, we begin to experience overflow.

Sub Question 3

Which of the three expressions have the overflow problem? Explain why.

Answer: For case A, as seen in the graph, there will be overflow when n crosses 175. This is because of the numerator prod(1:n). As seen below, the prod of 1:171 gives an overflow:

```
prod(1:171)
```

```
## [1] Inf
```

For case B, as seen in the graph, overflow is observed for k values between 1-50. This is because of the term prod(k+1:n) in the numerator. This causes the overflow for values like shown below:

```
prod(52:200)
```

```
## [1] Inf
```

For case C, as seen in the graph above and in the formula below, overflow is observed for value of k at 200 and values of n after approximately 2704.

```
prod(((200+1):2704)/(1:(2704-200)))
```

```
## [1] 1.685132e+308
```

At n = 2705, we see overflow:

```
prod(((200+1):2705)/(1:(2705-200)))
```

```
## [1] Inf
```

Appendix : All code for this report.

```
library(ggplot2)
library(tidyr)
library(dplyr)
knitr::opts_chunk$set(echo = TRUE)

# Question 1, sub-question 2:
x1 <- 1/ 3 ; x2 <- 1/4
if(all.equal((x1-x2), 1/12)){
  print("Subtraction is correct")
}else{
  print("Subtraction is wrong")
}
# Question 2 sub question 1:
der_fn <- function(x, epsilon){
  return(((x+epsilon) - x)/epsilon)
}
# Question 2 sub question 2:
# x = 1
der_fn(x = 1, epsilon = 10^-15)
# x = 100000
der_fn(x = 100000, epsilon = 10^-15)
100000 + 10^-15 + 10^-15 + 10^-15 == 100000
```

```r
# Question 3 sub question 1:
myvar <- function(x){
 n <- length(x)
 return((sum(x^2) - (1/n)*sum(x)^2)/(n-1))
}
# Question 3 sub question 2:
x_pop <- rnorm(10000, mean = 10^8, sd = 1)
# Question 3 sub question 3:

y_df <- as.data.frame(matrix(ncol=2, nrow = 0))
colnames(y_df) <- c('iVal', 'yVal')
for (i in 1:5000) {
  y_df <- rbind(y_df, data.frame(iVal = i,
                                 yVal = myvar(x_pop[1:i]) - var(x_pop[1:i]))
                )
}
ggplot(y_df) +
  geom_point(aes(x = iVal, y = yVal)) +
  ylab('Error')

# Question 3 sub question 4:

myvar_new <- function(x){
  imp_var <- sum((x - mean(x))^2)/(length(x)-1)
  return(imp_var)
}
cat('Result of the new implementation for variance calculation:', myvar_new(x_pop[1:9]))
cat('Variance calculated using the inbuilt var():', var(x_pop[1:9]))
# Question 4 sub question 2:
prod_df <- as.data.frame(matrix(nrow=0,ncol=3))
colnames(prod_df) <- c('n', 'k', 'prod')
k <- 10
for (n in 100:200) {
  prod1 <- prod(1:n)/(prod(1:k) * prod(1:(n-k)))
  prod_df <- rbind(prod_df ,
                   data.frame(n=n, k=k,
                              prod = prod1))
}
ggplot(prod_df, aes(x = n, y = prod)) +
  geom_point() +
  ggtitle('Plot of n vs Prod(case A) where k fixed to 10')
prod_df <- as.data.frame(matrix(nrow=0,ncol=3))
colnames(prod_df) <- c('n', 'k', 'prod')
n <- 200
for (k in 1:200) {
  prod2 <- prod((k+1):n)/prod(1:(n-k))
  prod_df <- rbind(prod_df ,
                   data.frame(n=n, k=k,
                              prod = prod2))
}
ggplot(prod_df, aes(x = k, y = prod)) +
  geom_point() +
  ggtitle('Plot of k vs Prod(case B) where n fixed to 200')
```

```r
prod_df <- as.data.frame(matrix(nrow=0,ncol=3))
colnames(prod_df) <- c('n', 'k', 'prod')
k <- 200
for (n in 250:2703) {
  prod3 <- prod(((k+1):n)/(1:(n-k)))
  prod_df <- rbind(prod_df ,
                   data.frame(n=n, k=k,
                              prod = prod3))
}
ggplot(prod_df, aes(x = n, y = prod)) +
  geom_point() +
  ggtitle('Plot of n vs Prod(case C) where k fixed to 200')

prod(1:171)
prod(52:200)
prod(((200+1):2704)/(1:(2704-200)))
prod(((200+1):2705)/(1:(2705-200)))
```