Computer Lab 1

Martynas Lukosevicius, Alejo Perez Gomez, Zahra Jalil Pour

03/11/2020

Question 1 (Be Careful When Comparing)

1

It is not possible to represent exact 1/3 and 1/12 in binary. As a result it is rounded towards nearest computer float and in R its equal to 0.33333333333333331 and 0.083333333333333329 respectively. In the first snippet code, the result is "Subtraction is wrong", but the second snippet code the result is "Subtraction is correct". The only numbers that represent exactly in r , are integers and fractions by power of 2 denominator. Hence all other numbers are rounded to 53 binary digits accuracy. When ever floating point operations are done , we should assume that there will be numeric error. 1/3 and 1/12 are repeating decimals that are rounded in R.

2

Instead of writing `if(x1 - x2 == 1/12)` it should be written `if(isTRUE(all.equal(x1-x2,1/12)))`. In this case this equation will return TRUE. We can use `all.equal` function , or we can use `all.equal.numeric` function too.

Question 2 (Derivative)

1

Write your own R function to calculate the derivative of `f(x) = x` in this way with `e = 10^-15`.

```
derivative <- function(x,e){
  f <- function(x){
  return(x)
}
  return((f(x+e)-f(x))/e)
}
```

2

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

```
e <- 10^(-15)
x <- 1

derivative(x,e)
```

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

1 1.110223

```
e <- 10^(-15)
x <- 100000
```

```r
derivative(x,e)
```

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

1 0

###3

when x = 1, derivative = 1.110223, when x = 100000, derivative = 0

However, true values for both cases should be 1.

The smallest positive computer number is epsilon that here we considered it `10^(-15)`When `x=100000` the derivative function showed 0, in equation`((x+e)-x)` , difference between large numbers dominates epsilon, in other words the smallest positive number is added to the large number. Hence the epsilon would be ignored. But when `x=1`, the effect of epsilon can not be ignored the result would be 1.110223.

Question 3 (Variance)

1

```r
myvar <- function(x){
  n <- length(x)
  xSq <- sum(x^2)
  sumXSq <- sum(x)^2
  part2 <- sumXSq/n
  return((xSq - part2)* (1/(n-1)))
}
```

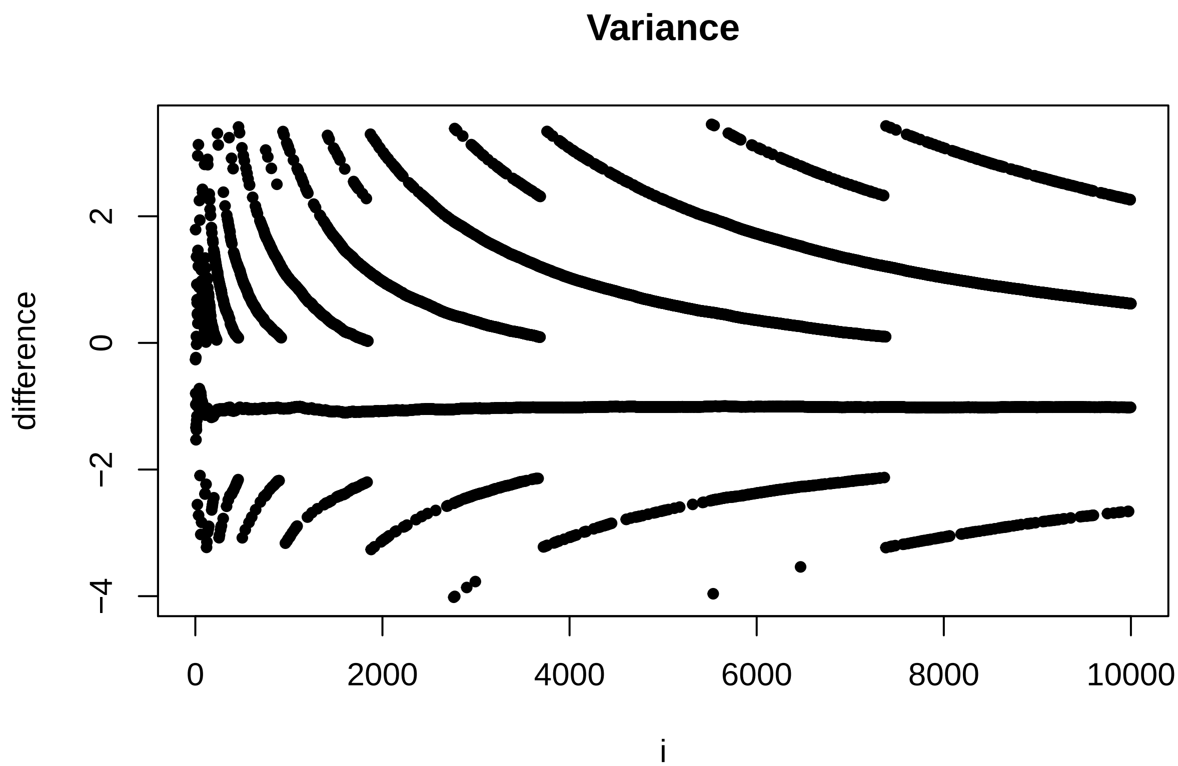
2

```r
x <- rnorm(10000, 10^8, 1)
```

3

```r
result <- list()
options(digits = 22 )
for (i in 1:length(x)) {
  temp <- x[1:i]
  y <- myvar(temp) - var(temp)
  result <- append(result, y)
}

plot(c(1:length(x)), result, main = "Variance", xlab = "i",type = "p",  pch = 20 , ylab = "difference")
```



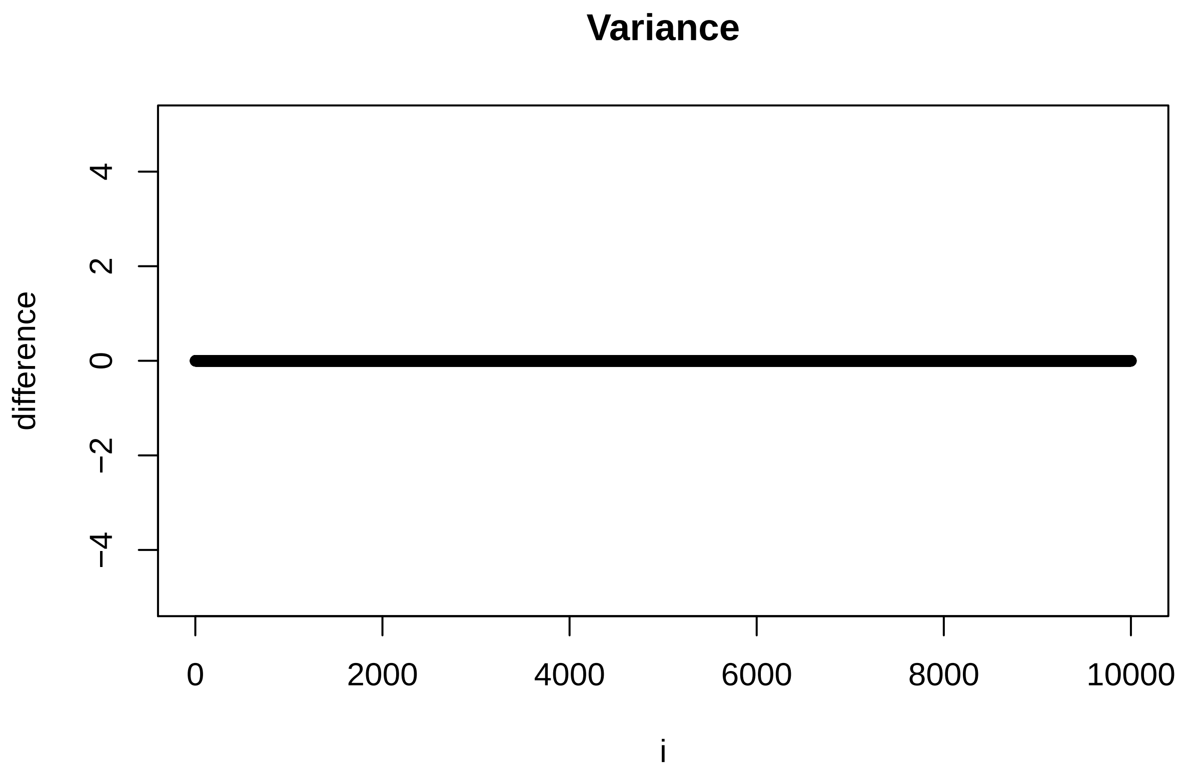
The function does not work good. It can be noted that the oscillation pattern in the produced response (myVar(Xi) - var(Xi)) decreases as the value of terms involved increases. Squaring big numbers result in losing precision (point). As a result first squaring and summing might be less than first summing and later squaring. that is why function will not produce correct answers.

4

```
myvar2 <- function(x){
  n <- length(x)
  return((sum((x - mean(x))^2))/(n-1))
}

result <- list()
options(digits = 22 )
for (i in 1:length(x)) {
  temp <- x[1:i]
  y <- myvar2(temp) - var(temp)
  result <- append(result, y)
}

plot(c(1:length(x)), result, main = "Variance", xlab = "i",type = "p",  pch = 20 , ylab = "difference",
```



Question 4 (Binomial coeficient)

1

A: n , k and n - k cant be zero

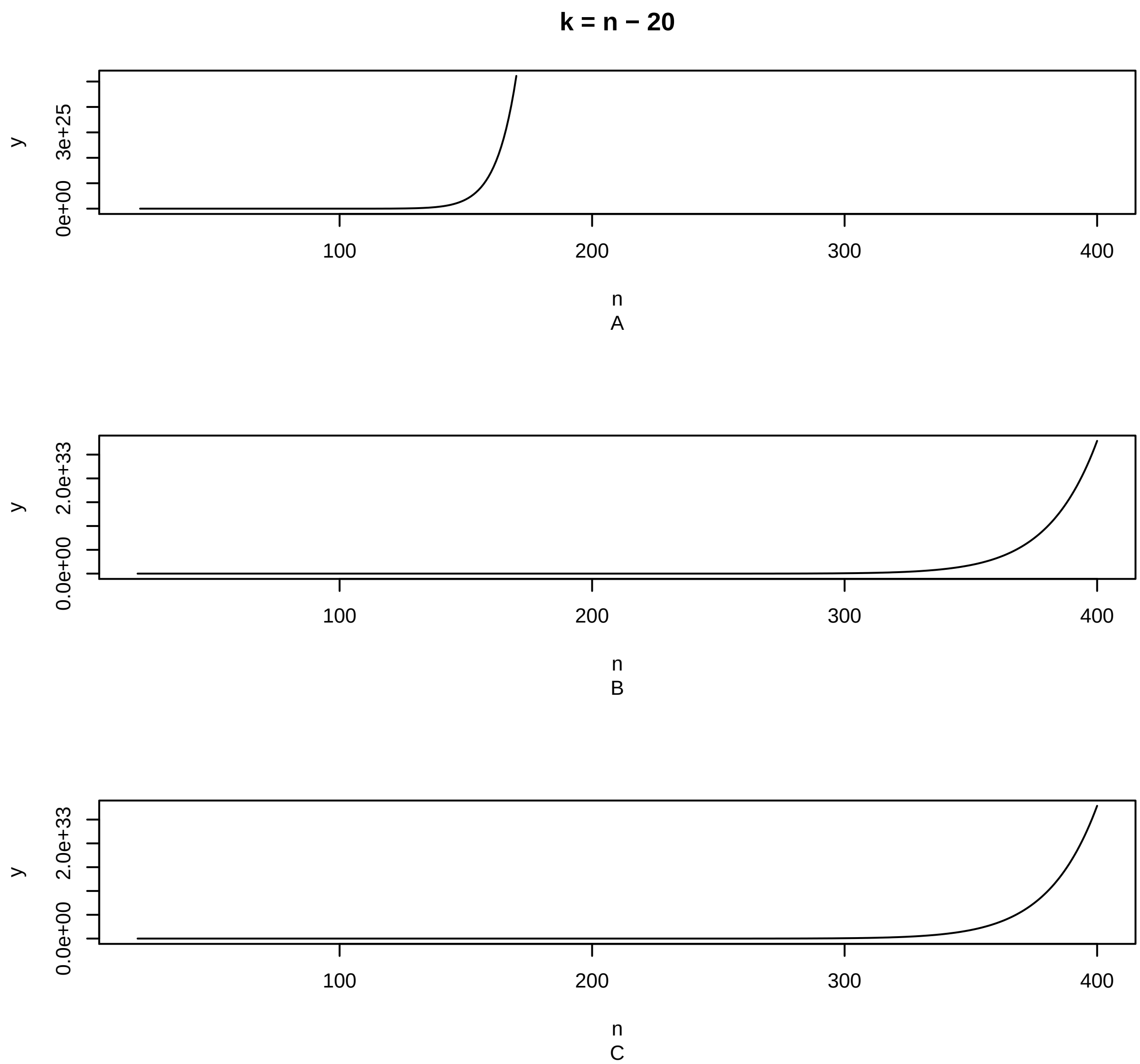
B: n and n - k cant be zero

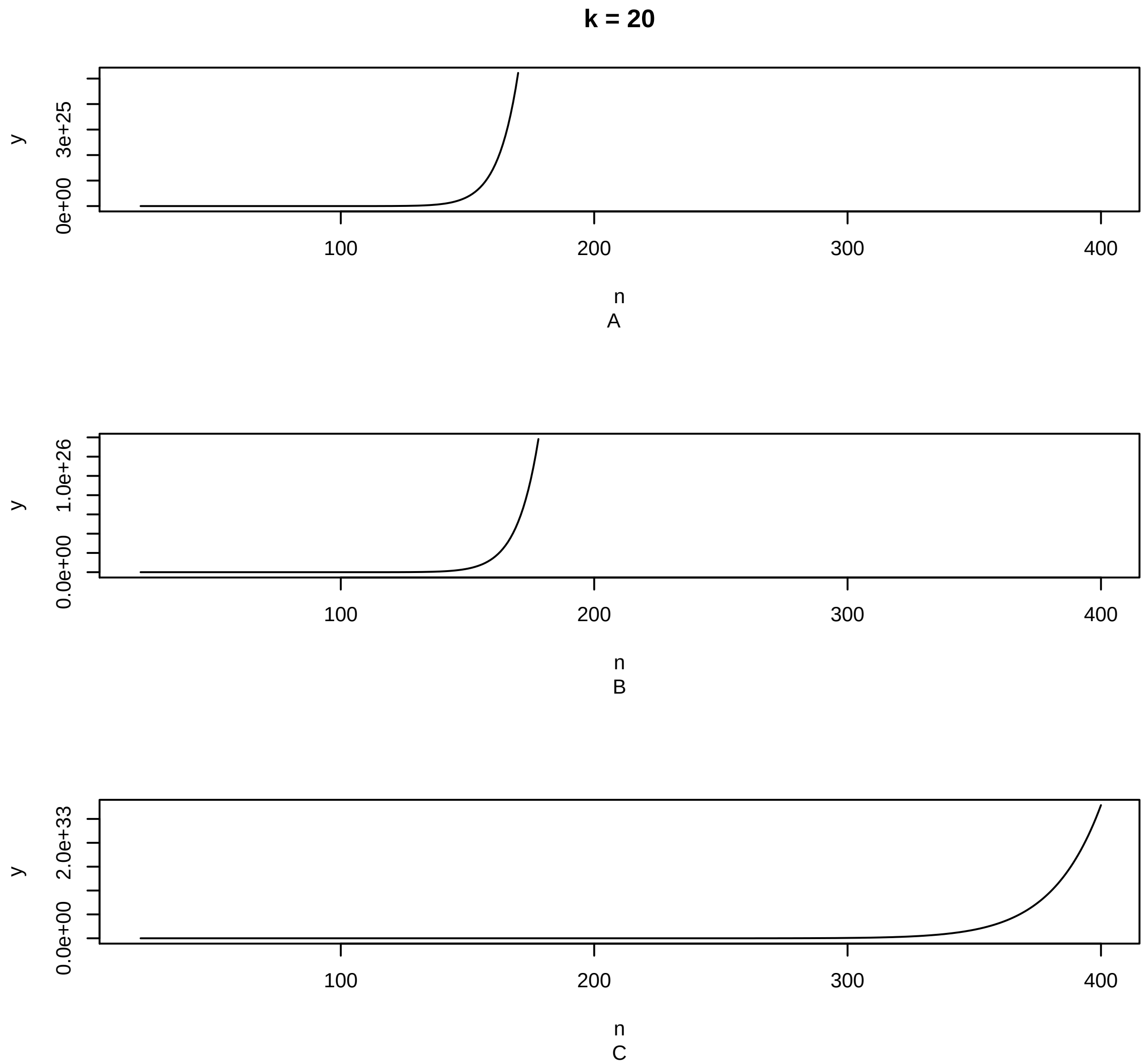
C: same as B

because prod(0) = 0 and 0 / 0 will be NaN

```
## [1] NaN
## [1] NaN
## [1] NaN
```

2


k = n − 20
y
0e+00
3e+25
100
200
300
400
n
A
y
0.0e+00
2.0e+33
100
200
300
400
n
B
y
0.0e+00
2.0e+33
100
200
300
400
n
C



3

expression A and B, because with large numbers method prod() will overflow.

In expression A we calculate product of vector from 1 to n and later divide it by other products with smaller vectors. However in this case first operation (prod(1:n)) will overflow (= Inf) and other operations wont matter as the result will be Inf or Nan (if denominator will be also Inf).

In expression B overflow will depend on k, if k is close to n it wont overflow.

In expression C, as first vectors are divided, the final vector for product will have smaller values and that is why prod() method wont overflow.