# 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, which in R is equal to 0.33333333333333331 and 0.083333333333333329 respectively. In the first snippet code, the result is "Subtraction is wrong", but in the second snippet code the result is "Subtraction is correct". A rational number , of any size may or may not have an exact representation by a floating point number. This is the familiar situation where fractions such as 1/3 have no finite representation in base 10. The only numbers that can be represented exactly in R are integers and powers of two. Hence all other numbers are rounded to 53 binary digits accuracy. Whenever 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 $if(x_1 - x_2 == 1/12)$ it should be written $if(isTRUE(all.equal(x_1 - x_2, 1/12)))$. In this case this equation will return TRUE. We can use `all.equal` function , or we can use `all.equal.numeric` function too, which will test if the compared numbers are nearly equal.

## Question 2 (Derivative)

### 1

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

```
  f <- function(x){
  return(x)
  }

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

### 2

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

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

derivative(x,e)
```

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

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

derivative(x,e)
```

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

#### 3

When $x = 1$, $derivative = 1.110223$ and 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 to be $10^{-15}$. When $x = 100000$ the derivative function showed 0, in the equation $((x+e)-x)$, the difference between large numbers dominates over epsilon. In other words, the smallest positive number is added to the large number. Hence the epsilon would be ignored. However, when $x = 1$, the effect of epsilon cannot 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

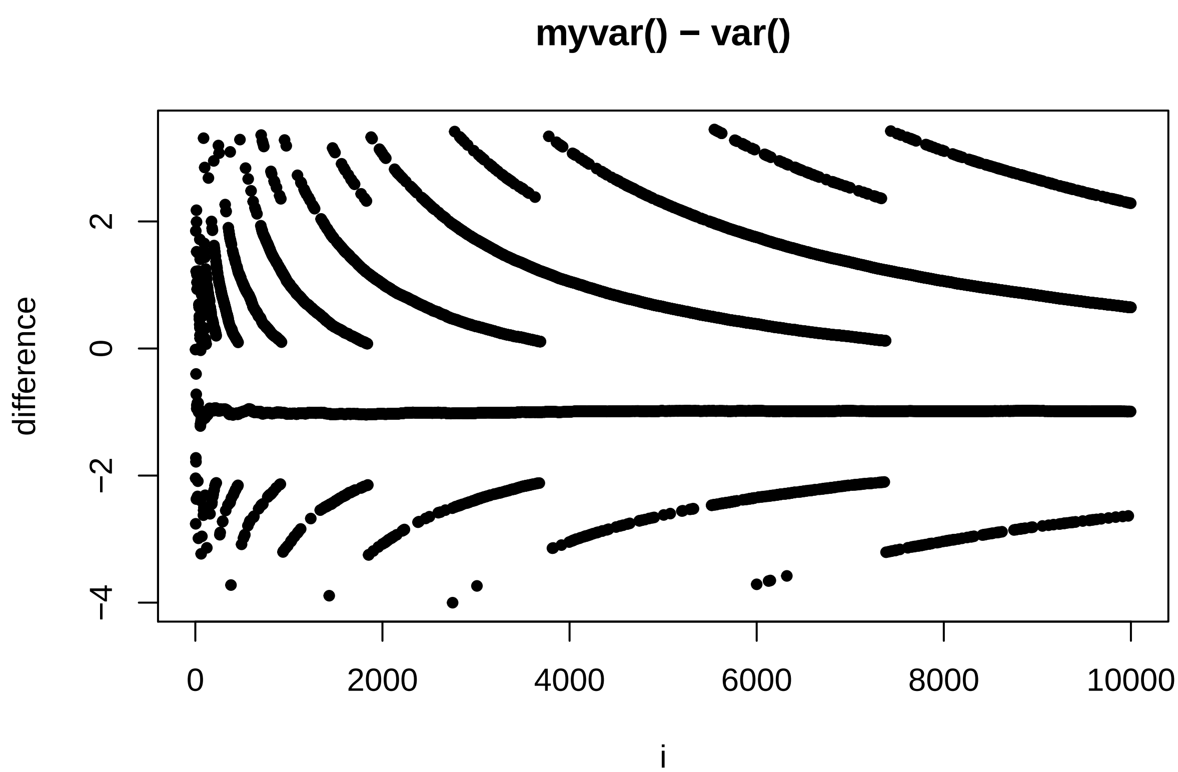
```r
x <- rnorm(10000, 10^8, 1)
```

#### 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 = "myvar() - var()",
     xlab = "i",type = "p",  pch = 20 , ylab = "difference")
```




The function does not work properly. It oscillates primarily when a smaller set of values is involved in the variances calculation. It can also be noted that the oscillation pattern in the produced response $(myVar(X_i) - var(X_i))$ decreases as the value of terms involved increases. Squaring big numbers result in overflow, since the computer is not able to handle such a large number correctly. That is because the numbers are so big that the computer cannot destine the right amount of bytes for them. Moreover, first squaring and summing may lead to a smaller result than first summing and later squaring. This is why our function will not produce correct answers.

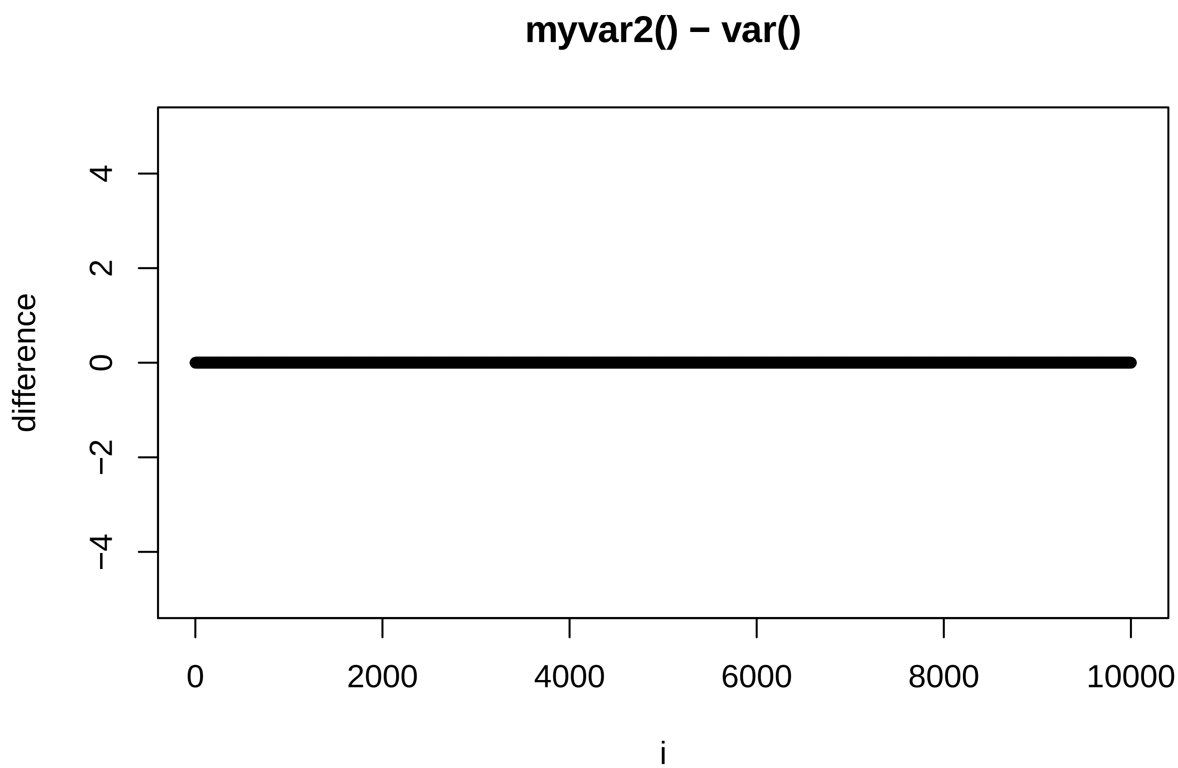
**4**

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

result <- list()
options(digits = 22 )
```

```r
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 = "myvar2() - var()", xlab = "i",type = "p",
     pch = 20 , ylab = "difference", ylim = c(-5,5))
```




## Question 4 (Binomial coeficient)

### 1

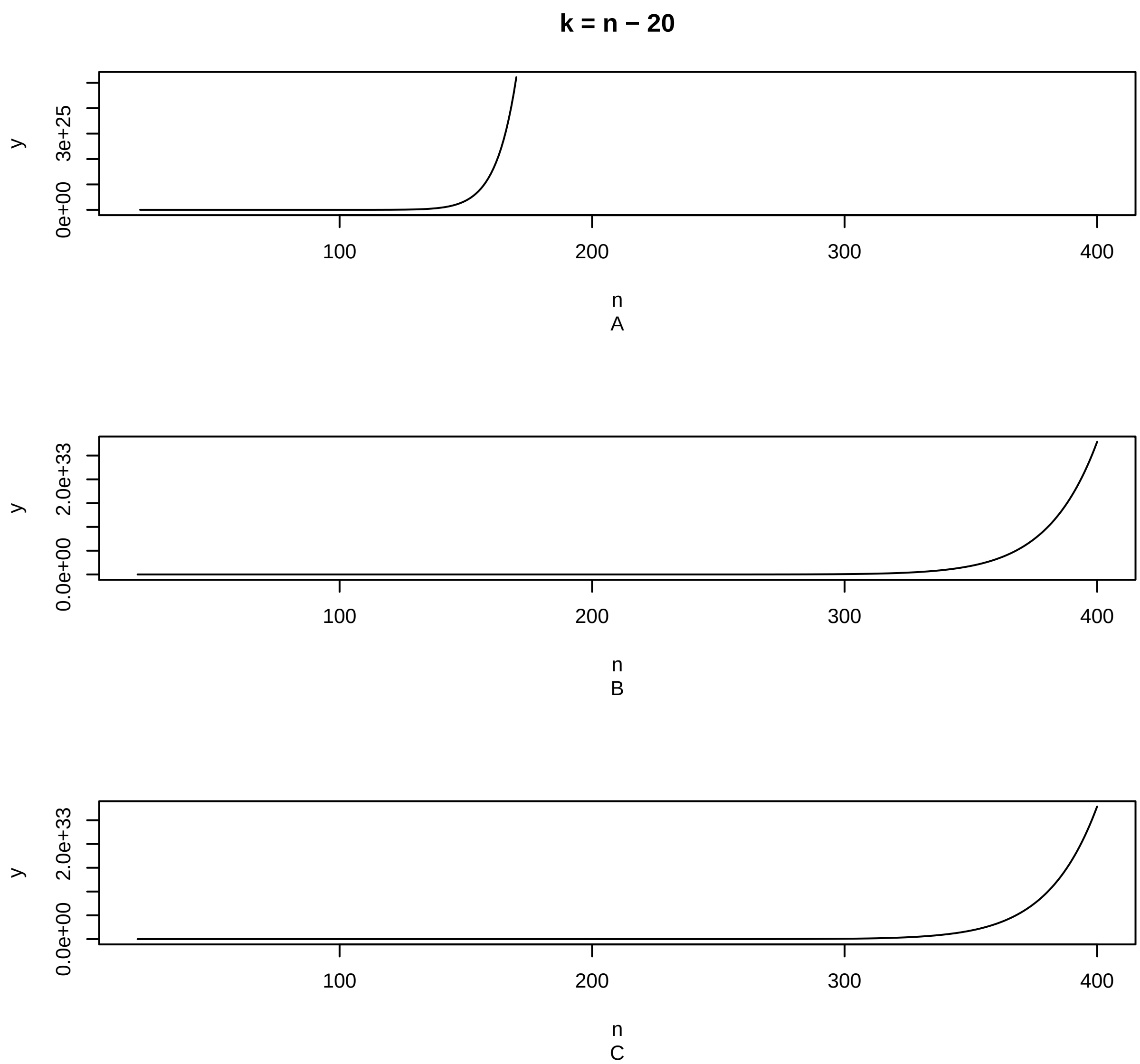
A: $n$ , k and $n - k$ cant be zero

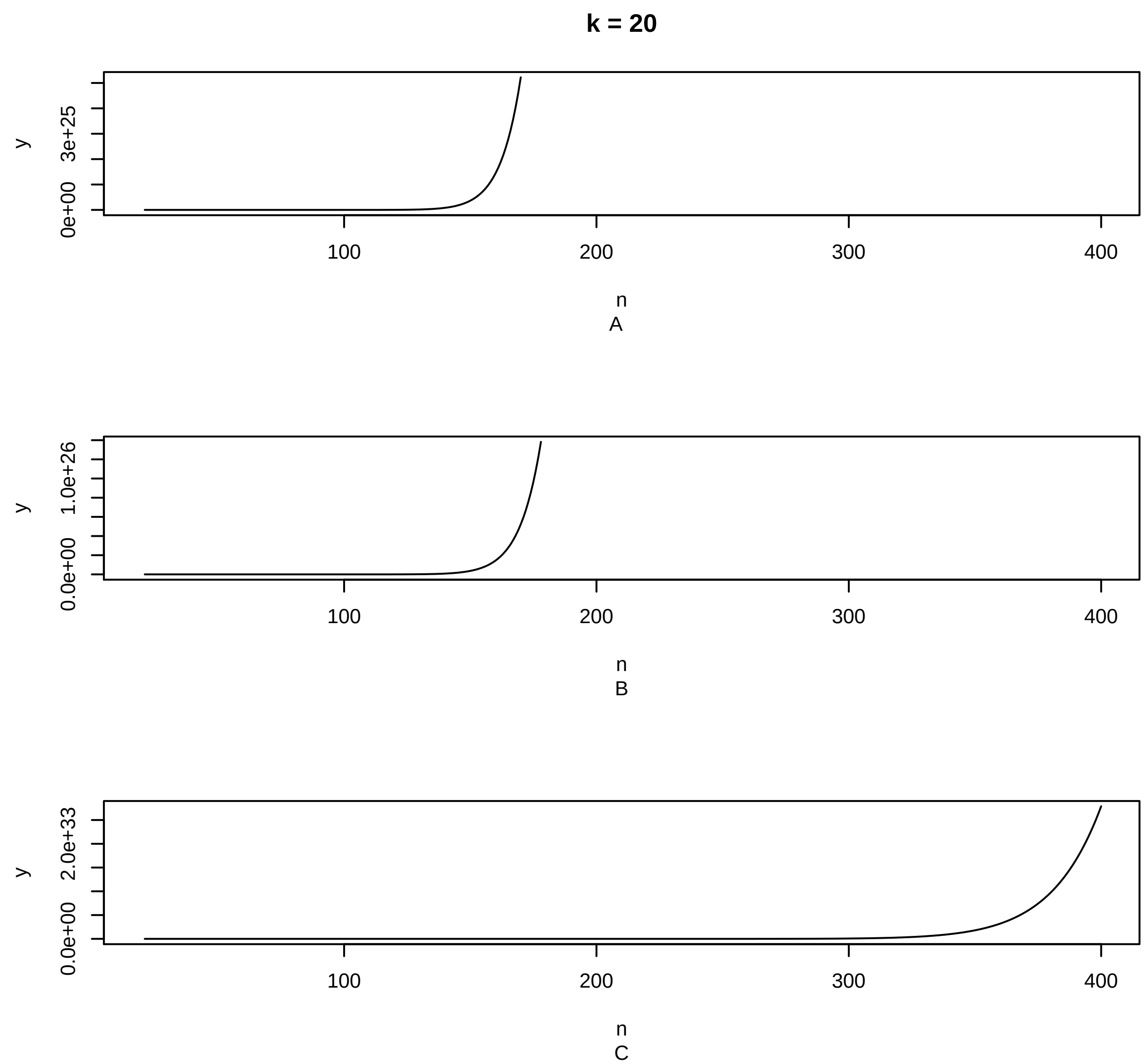
B: $n$ and $n - k$ cant be zero

C: same as B

because $prod(0) = 0$ and $\frac{0}{0}$ will be 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**

The expressions A and B, because with large numbers method *prod*() will overflow.

In expression A we calculate the product of a vector from 1 to n and later divide it by other products with smaller vectors. However, in this case, the first operation ($prod(1 : n)$) will overflow ( = Inf) and other operations won't 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 won't overflow.

In expression C, as first vectors are divided, the final vector for product will have smaller values and that is why *prod*() method won't overflow.

# Appendix

```r
#1
# all.equal.numeric() and isTRUE() function
x1 <- 1/3
x2 <- 1/4
if (isTRUE(all.equal.numeric(x1-x2, 1/12))) {
  print ("Subtraction is correct" )
} else {
  print ("Subtraction is wrong")
}

#2

f <- function(x){
  return(x)
}

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

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

derivative(x,e)

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

derivative(x,e)

#3
myvar <- function(x){
  n <- length(x)
  xSq <- sum(x^2)
  sumXSq <- sum(x)^2
  part2 <- sumXSq/n
  return((xSq - part2)* (1/(n-1)))
}

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

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 = "myvar() - var()",
     xlab = "i",type = "p",  pch = 20 , ylab = "difference")
```

```r
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 = "myvar2() - var()", xlab = "i",type = "p",
     pch = 20 , ylab = "difference", ylim = c(-5,5))

#4

n <- 0
k <- 0
prod(1:n) / (prod(1:k) * prod(1:(n-k)))
prod((k+1):n) / prod(1:(n-k))
prod(((k+1):n) / (1:(n-k)))


#n <- c(20:100)

calc1 <- function(n,k){
k <- n - k
 return(prod(1:n) / (prod(1:k) * prod(1:(n-k))))
}
calc2 <- function(n,k){
  k <- n - k
 return(prod((k+1):n) / prod(1:(n-k)))
}
calc3 <- function(n,k){
  k <- n - k
 return(prod(((k+1):n) / (1:(n-k))))
}

n <- c(20:400)
k <- 20

y <-  sapply(n, calc1, k = k)
y2 <-  sapply(n, calc2, k = k)
y3 <-  sapply(n, calc3, k = k)
par(mfrow=c(3,1))
plot(n,y, type = "l", main = "k = n - 20", sub =  "A"  , ylab = "y")
plot(n,y2, type = "l", sub =  "B" , ylab = "y")
plot(n, y3, type = "l", sub =  "C" , ylab = "y")

calc1 <- function(n,k){
 return(prod(1:n) / (prod(1:k) * prod(1:(n-k))))
```

```r
}
calc2 <- function(n,k){
 return(prod((k+1):n) / prod(1:(n-k)))
}
calc3 <- function(n,k){
 return(prod(((k+1):n) / (1:(n-k))))
}

n <- c(20:400)
k <- 20

y <-  sapply(n, calc1, k = k)
y2 <-  sapply(n, calc2, k = k)
y3 <-  sapply(n, calc3, k = k)
par(mfrow=c(3,1))

plot(n,y, type = "l", main = "k = 20", sub =  "A  " , ylab = "y")
plot(n,y2, type = "l", sub = "B", ylab = "y")
plot(n, y3, type = "l", sub = "C", ylab = "y")
```