Laboration Assignment 1

Computational Statistics 732A90

Group 4

11/8/2021

Question 1

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

```
##snippet 1
x1<- 1/3
x2<- 1/4
if( x1-x2==1/12){
print ( " Subtraction is correct" )
}else{
print ( " Subtraction is wrong" )
}
```

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

```
##snippet 2
x3<- 1
x4<- 1/2
if( x3-x4==1/2){
    print ( " Subtraction is correct" )
}else{
    print ( " Subtraction is wrong" )
}
```

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

1.1 Comment: The variable x_1 is assigned the value $\frac{1}{3}$ and the variable x_2 is assigned the value $\frac{1}{4}$. x_2 is then subtracted from x_1 and if the resulting difference is equal to exactly $\frac{1}{12}$ the message "Subtraction is correct" is printed, else the message *"Subtraction is wrong"* is printed.
In the second part the variable x_1 is assigned the value 1, while x_2 is assigned the value $\frac{1}{2}$. x_2 is then subtracted from x_1 and if the difference is equal to exactly $\frac{1}{2}$ the message *"Subtraction is correct"* is printed, else the message *"Subtraction is wrong"* is printed.

In another words the main reason of these two different results, that for the first snippet the calculation of $\frac{1}{3}$ = 0.33333333… and $\frac{1}{4}$ = 0.25 and when we convert the decimal fraction to binary fraction we get:
$N_1 = \frac{1}{3} = 0.0333333\ldots = 0.010101010101\ldots$ $N_2 = \frac{1}{4} = 0.25 = 0.01$ By calculating the subtraction between $N_1 - N_2 = (M_1 * 10^{x_1}) - (M_2 * 10^{x2}) = 0.01010 - 0.10000 = 0.00101 = 0.15$ <> 0.08333..
So, what happen is when we subtract $\frac{1}{3}$ from $\frac{1}{4}$ the number $\frac{1}{3}$ is been truncated into 0.10101 instead of 0.101010101… which change the result of the subtraction.
On the other hand, for the second snippets, we can represent 1 as 1.0 and $\frac{1}{2}$ as 0.1 In binary system, the subtraction for the both numbers result as 1.0000 + 0.1000 =0.10000 = .05

1.2 If there are any problems, suggest improvements

1.2 Comment: x_1 - x_2 does not equal $\frac{1}{12}$ in the first part.
Computers use $base_2$ system and as a consequence only numbers whose denominator is a sum of powers of 2 can be represented with a finite number of bits.
12 is not a sum of powers of 2 and hence $\frac{1}{12}$ cannot be represented exactly in neither binary nor decimal form. Because $\frac{1}{12}$ lacks a finite representation, rounding is necessary. If the difference between x_1 and x_2 is rounded differently than $\frac{1}{12}$ the numbers are not going to be exactly equal. A potential solution to this problem is to specify how the numbers are rounded, for example by using the round() function in R:

```
if(round(x1-x2,8) == round(1/12, 8))
```

Question 2: Derivative

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

```
derivative<- function(x){
    e=1e-15
    f=((x+e)-x)/e
    return(f)
}
```

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

x=1

```
derivative(1)
```

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

x=100000

```
derivative(100000)
```

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

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

2.3.1 Comment 1 In both cases the true values should be :

$$f'(x) = \frac{\partial}{\partial x} = 1$$
$$f'(1) = 1$$
$$f'(100000) = 1$$

In the first case when *x=1* the problem could be caused by underflow which may lead to loss of significant digits.
The reason why the value is 0 in the second case could be due to a rounding error of ϵ to 0 which causes $f(x+\epsilon)$ and $-f(x)$ to cancel each other out.

2.3.1.1 Comment 2 The approximation error in $f(x+\epsilon) - f(x)$ decreases as ϵ gets smaller, which says we should take ϵ as small as possible. But as ϵ gets smaller, the error from floating point subtraction increases since the numerator requires subtracting nearly equal numbers. If ϵ is too small, we can loose a lot of precision in the subtraction.

Question 3: Variance

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

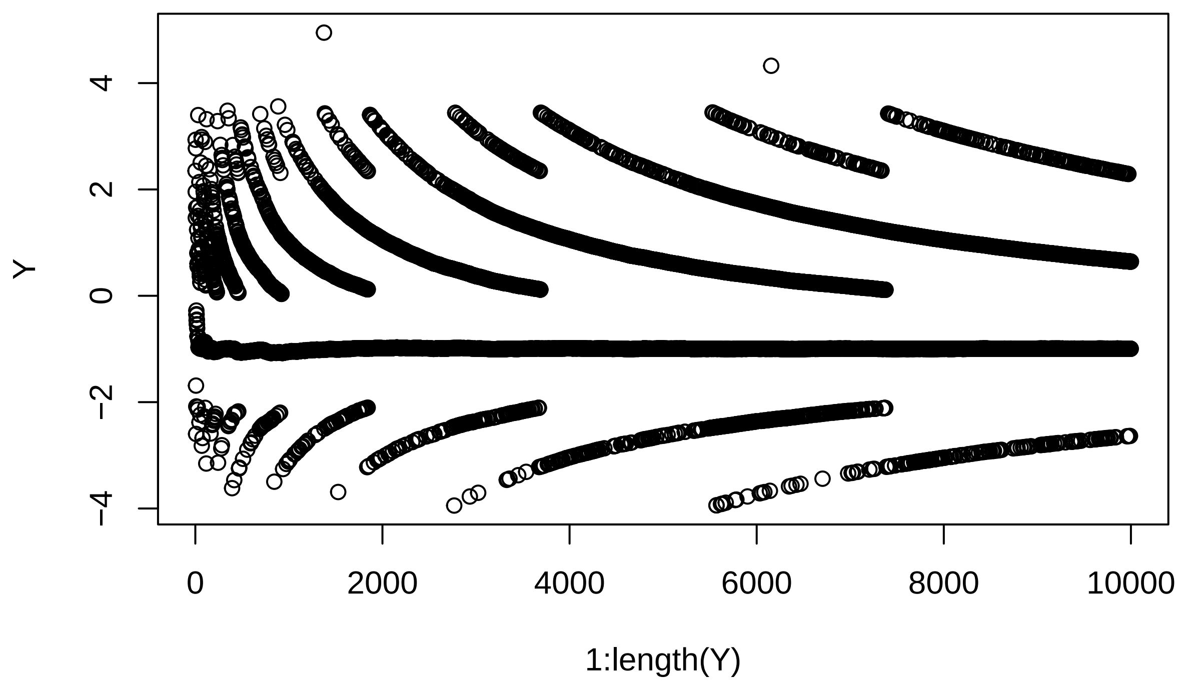
```
myvar <- function(x) {
 n <- length(x)
 var_x <- (1/(n-1)) * (sum(x^2) - (1/n) * ((sum(x))^2))
 return(var_x)
}
```

3.2 Generate a vector x = (x_1,...,$x_1$0000) with 10000 random numbers with mean 10^8 and variance 1

```
x= rnorm(10000,mean = 10^8,sd= 1)
```

3.3 For each subset Xi = {x_1,...,x_i}, i = {1,...,10000} compute the difference Y_i = myvar(X_i)-var(X_i), where var(X_i) is the standard variance estimation function in R. Plot the dependence Y_i on i. Draw conclusions from this plot. How well does your function work? Can you explain the behavior?

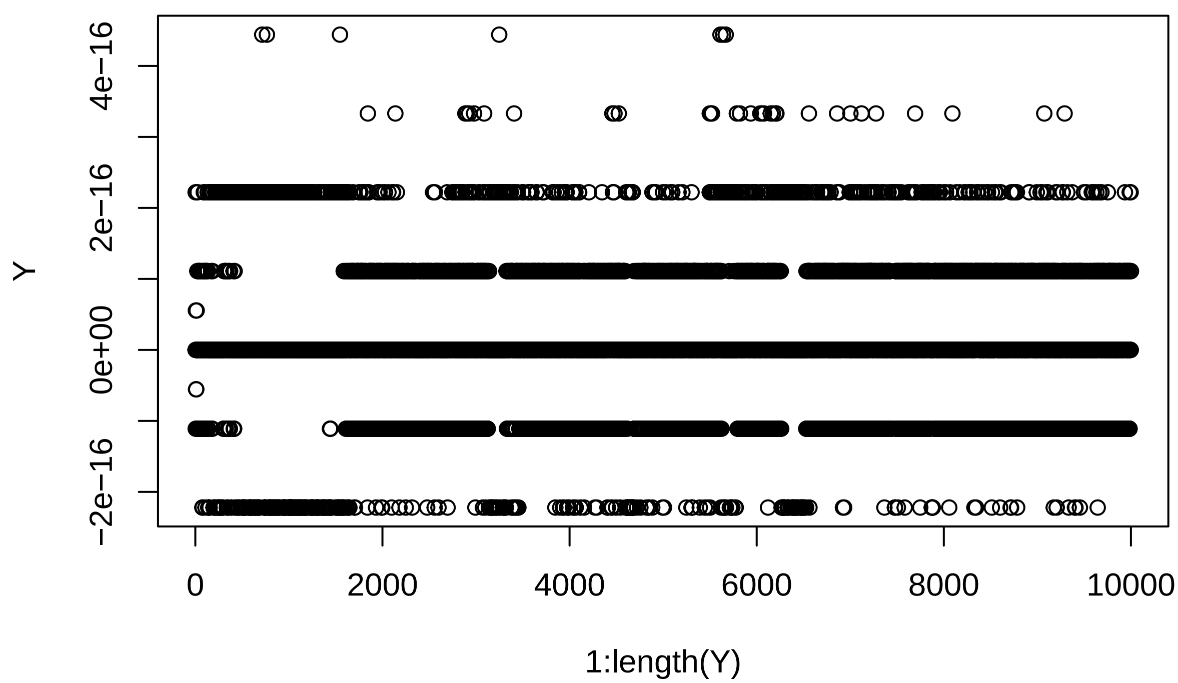
```
Y <- c()
for(i in 1:length(x)) {
 Y[i] <- myvar(x[1:i]) - var(x[1:i])
}
plot(1:length(Y), Y)
```



3.3.1 Comment For large x_i the terms $\sum_{i=1}^{n} x_i^2$ and $\frac{1}{n}\ (\sum_{i=1}^{n} x_i)^2$ will be close to each other, which can lead to precision problems due to loss of significant digits – a phenomenon called catastrophic cancellation.

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

```
myvar2 <- function(x) {
 n <- length(x)
 var_x <- (1/(n-1)) * sum((x-(sum(x)/n))^2)
 return(var_x)
}
Y <- c()
for(i in 1:length(x)) {
 Y[i] <- myvar2(x[1:i]) - var(x[1:i])
}
plot(1:length(Y), Y)
```



Question 4: Binomial coeffcient

4.1 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.

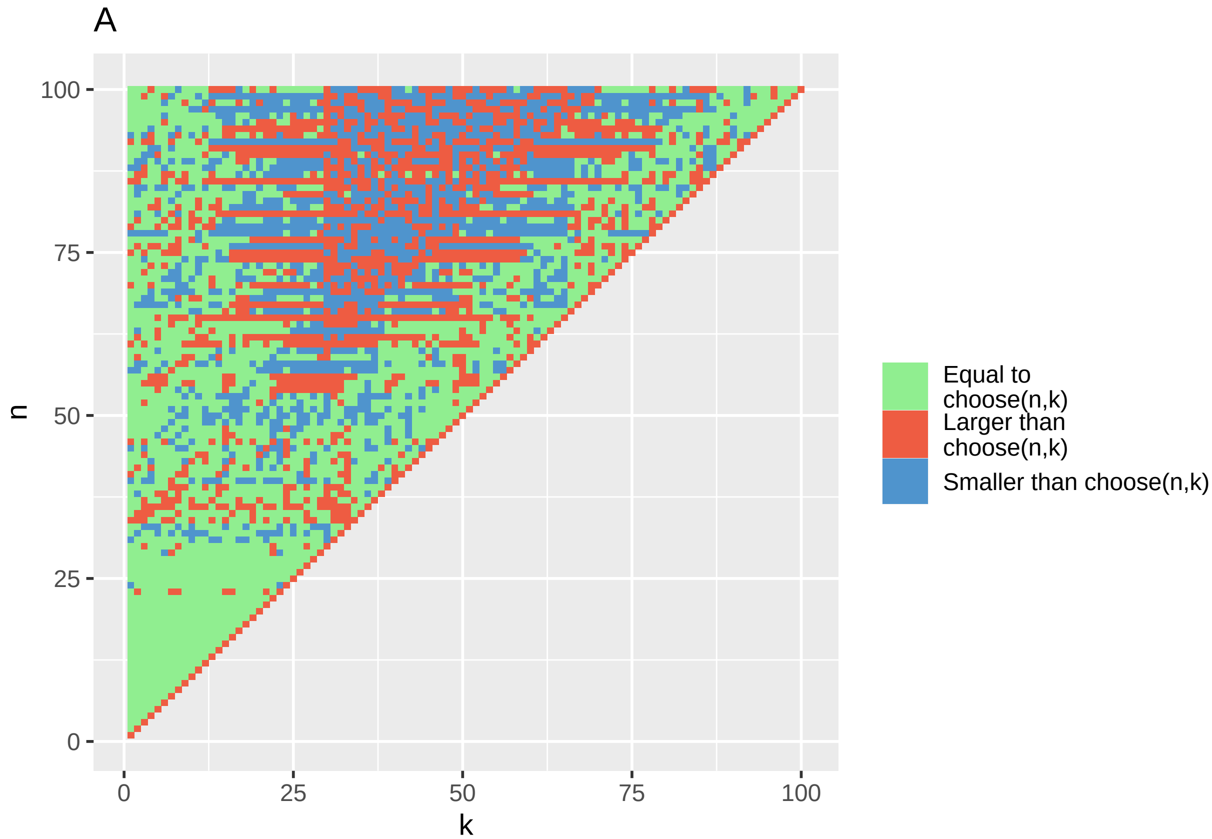
When n=k the denominator becomes 0 in both A, B and C which gives the results Inf instead of 1. Both A, B and C have trouble handling large n and k:s.
In both A, B and C fractions that lacks a finite decimal representation can occur which might lead to rounding errors.

4.2 Experiment numerically with the code of A, B and C, for different values of n and k to see whether overflow occurs. Graphically present the results of your experiments.

```
bin_coef_A <- function(n, k) {
 return(prod(1:n) / (prod(1:k) * prod(1:(n-k))))
}
bin_coef_B <- function(n, k) {
 return(prod((k+1):n) / prod(1:(n-k)))
}
bin_coef_C <- function(n, k) {
 return(prod(((k+1):n) / (1:(n-k))))
}
df <- as.data.frame(matrix(NA, nrow = 0, ncol = 6))
colnames(df) <- c("n", "k", "choose()", "A", "B", "C")
row_counter <- 0
for(k in 1:100) {
 for(n in k:100) {
 row_counter <- row_counter+1
 df[row_counter,1] <- n
 df[row_counter,2] <- k
 df[row_counter,3] <- choose(n, k)
 df[row_counter,4] <- ifelse(bin_coef_A(n, k) == choose(n, k), "Equal to
choose(n,k)", ifelse(bin_coef_A(n, k) > choose(n, k), "Larger than
choose(n,k)", "Smaller than choose(n,k)"))
 df[row_counter,5] <- ifelse(bin_coef_B(n, k) == choose(n, k), "Equal to
choose(n,k)", ifelse(bin_coef_B(n, k) > choose(n, k), "Larger than
choose(n,k)", "Smaller than choose(n,k)"))
 df[row_counter,6] <- ifelse(bin_coef_C(n, k) == choose(n, k), "Equal to
choose(n,k)", ifelse(bin_coef_C(n, k) > choose(n, k), "Larger than
choose(n,k)", "Smaller than choose(n,k)"))
 }
}
```

```
ggplot(df, aes(x = k, y = n, fill = factor(A))) +
 geom_tile() +
 scale_fill_manual(values = c("palegreen2", "tomato2", "steelblue3")) +
 ggtitle("A") +
 labs(fill = "")
```

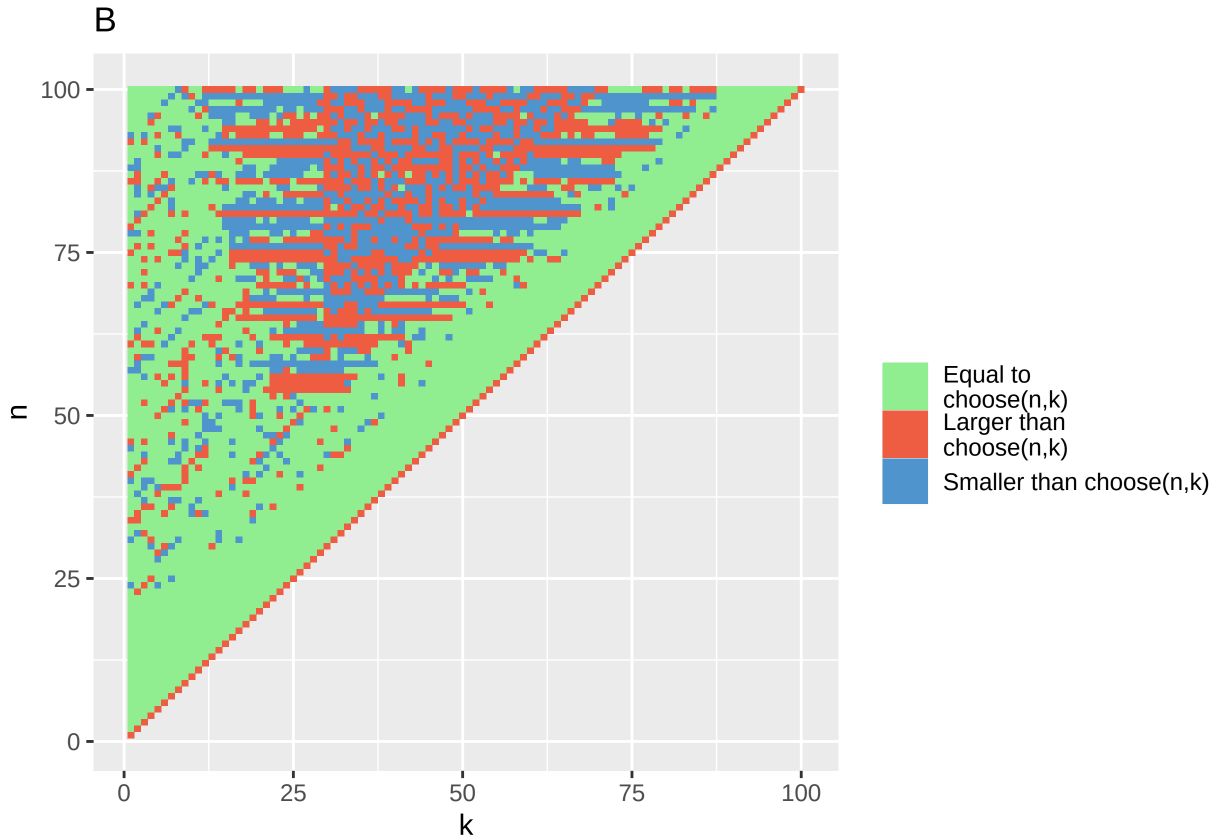
4.2.1 Graph A



A start giving the wrong results when n > 22.

```
ggplot(df, aes(x = k, y = n, fill = factor(B))) +
 geom_tile() +
 scale_fill_manual(values = c("palegreen2", "tomato2", "steelblue3")) +
 ggtitle("B") +
 labs(fill = "")
```

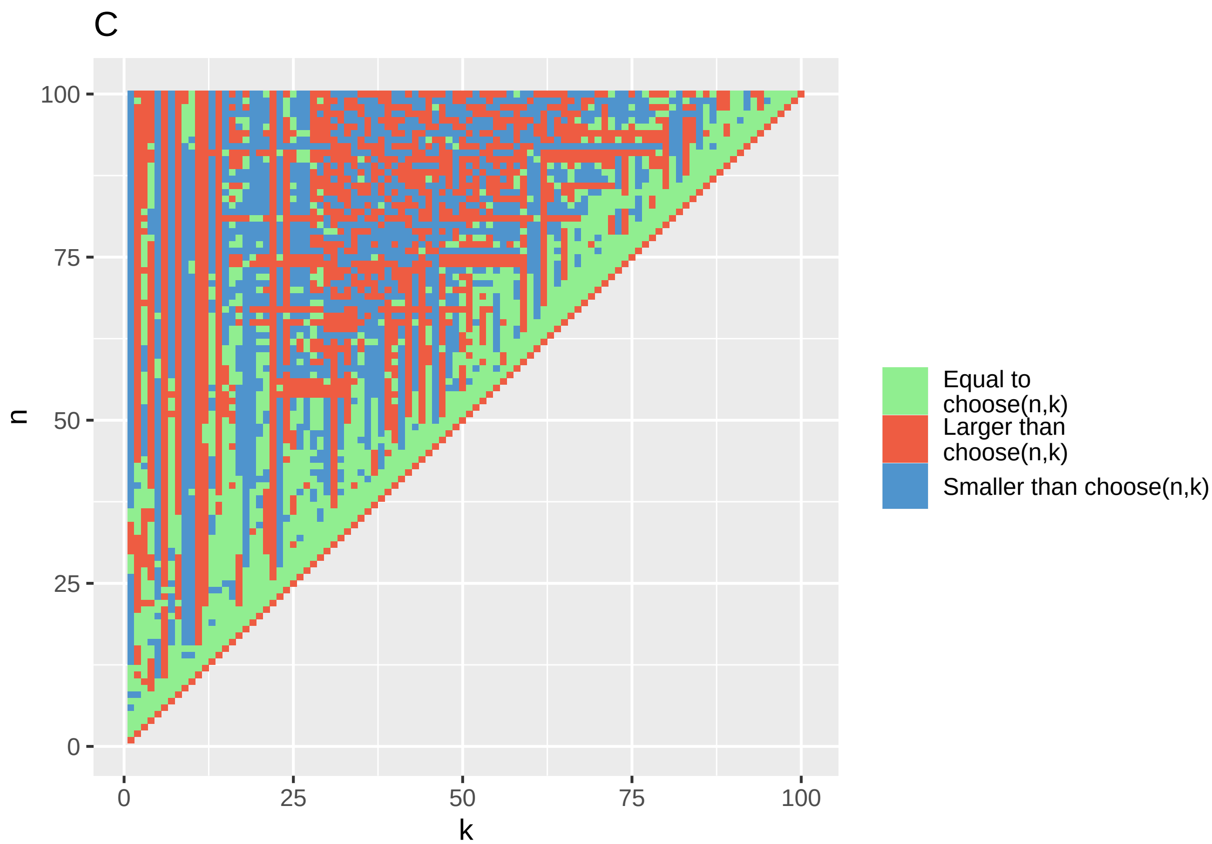
4.2.2 Graph B



B start to give the wrong results when $n > 22$ and k is small relatively to n

```
ggplot(df, aes(x = k, y = n, fill = factor(C))) +
 geom_tile() +
 scale_fill_manual(values = c("palegreen2", "tomato2", "steelblue3")) +
 ggtitle("C") +
 labs(fill = "")
```

4.2.3 Graph C



C start to give the wrong results when $n > 5$ and k is small relatively to n

Which of the three expressions have the overflow problem? Explain why.
Overflow is mainly a problem in A and B.
In A the denominator quickly become very big when n and k grows.
In B the denominator can become very big when n is large and k is much smaller.