Computer Lab1

Dinesh and Umamaheswarababu Maddela

2023-01-17

Question 1: Be careful when comparing

Consider the following two R code snippets

```
#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
x1<-1  ; x2<-1/2
if (x1-x2==1/2) {
  print ("Subtraction_is_correct" )
}else{
  print ("Subtraction_is_wrong")
}
```

```
## [1] "Subtraction_is_correct"
```

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

Snippet 1: x1 is assigned with 1/3 which is a rational number with infinite digits of 3 in the decimal part of the number (0.3333…). Any number with non-integer datatype is stored as its nearest floating point counterpart. When there are infinite digits after the decimal point of a number, there won't be enough space to store all the (infinite) digits in a computer with finite storage. The computer approximates rational numbers to closest floating point number which results in small inaccuracies. This is the reason the value of x1-x2 is not exactly equal to 1/12 so the output of the snippet is "Subtraction_is_wrong".

Snippet 2: In this case there is no problem in storing the exact value of 1/2 because there is finite number of digits after decimal point(0.5) which can be exactly represented in floating point arithmetic. Hence the output is "Subtraction_is_correct".

2. If there are any problems, suggest improvements.

Problem is there in the snippet 1 which can be improved using the function "all.equal" which tests if two objects are "near equality". The improved version of snippet 1 is as follows

```
#Snippet 1 - Improved
x1<-1 / 3 ; x2<-1 /4;
if(all.equal(x1-x2,1/12)){
  print("Subtraction is correct")
}else{
  print("Subtraction is wrong")
}
```

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

Question 2: Derivative

From the definition of a derivative a popular way of computing it at a point x is to use a small ϵ and the formula

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

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

```
#A function to calculate derivative of f(x) given epsilon = 10^-15
derivative <- function (x){
  return(((x+10^-15)-x)/10^-15)
}
```

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

```
derivative(x=1)
```

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

```
derivative(x=100000)
```

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

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

Case-1: For x= 1

The output of the derivative(x=1) is 1.110223 against the true value 1. The error is caused by the subtraction of two numbers (1+10^-15) and 1 which are very close to each other. The difference between these two numbers is 10^-15, which is very small and can't be represented precisely in floating point representation. Therefore the subtraction results in a small error, which is further increased through the division.

Case-2: For x= 100000

The output of the derivative(x=100000) is 0 against the true value 1. This due to the effect of underflow in the numerator of the expression. We can observe 100000 + 10^-15 - 100000 creates 0 because 10^-15 is too small which is not enough to change the value of (100000 + 10^-15) and it is neglected.

Question 3: Variance

A known formula for estimating the variance based on a vector of n observations is

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

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

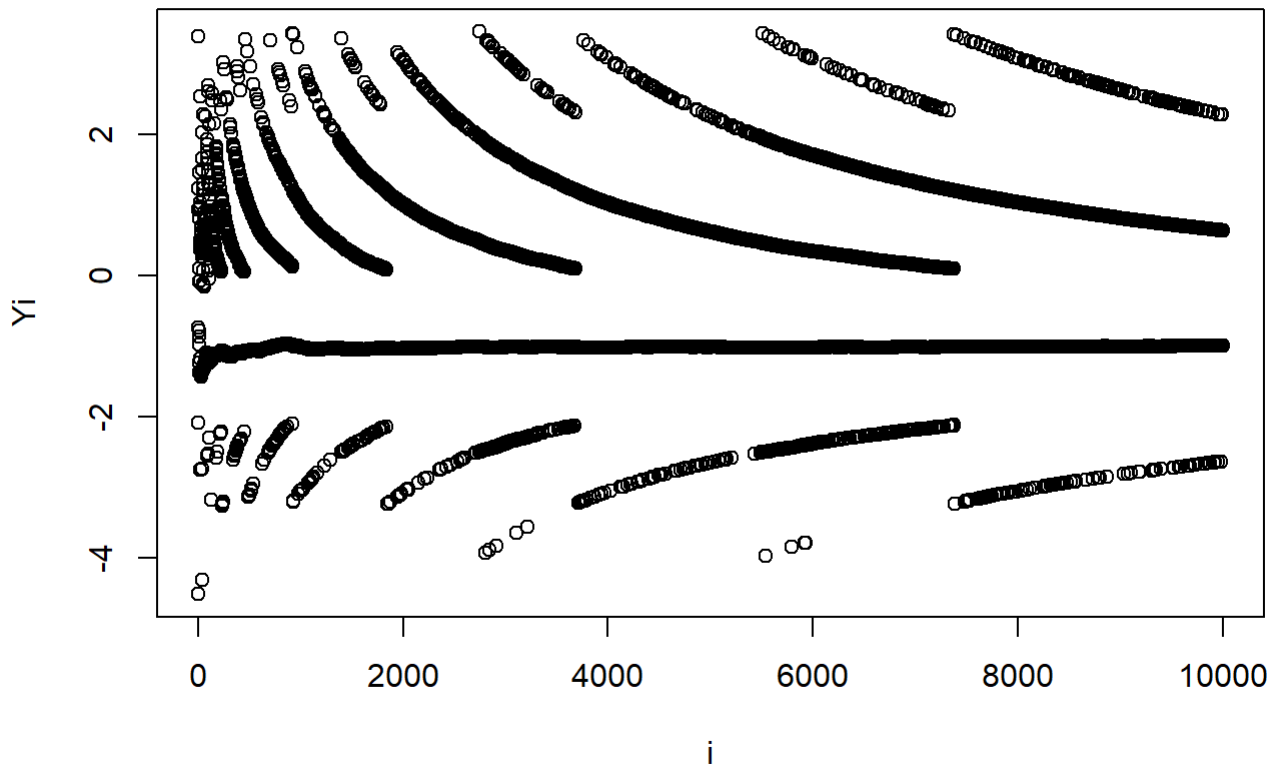
```
#A function to estimate the variance of a input vector x
myvar<- function(x){
  n<- length(x)
  var<-(sum(x^2)-sum(x)^2/n)/(n-1)
  return (var)
}
```

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

```
vector_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() #initializing Yi as vector to store the computed difference myvar(Xi) - var(Xi)
for (i in 1:length(vector_x)){
  Xi<- vector_x[1:i]          #subset of vector_x
  Yi[i]<- myvar(Xi)-var(Xi)
}

#remove Yi[1] because variance of one element cannot be calculated
Yi = Yi[-1]
plot(2:10000, Yi, xlab = "i") #ignoring i=1 because variance cannot be calculated for one elemen
t
```



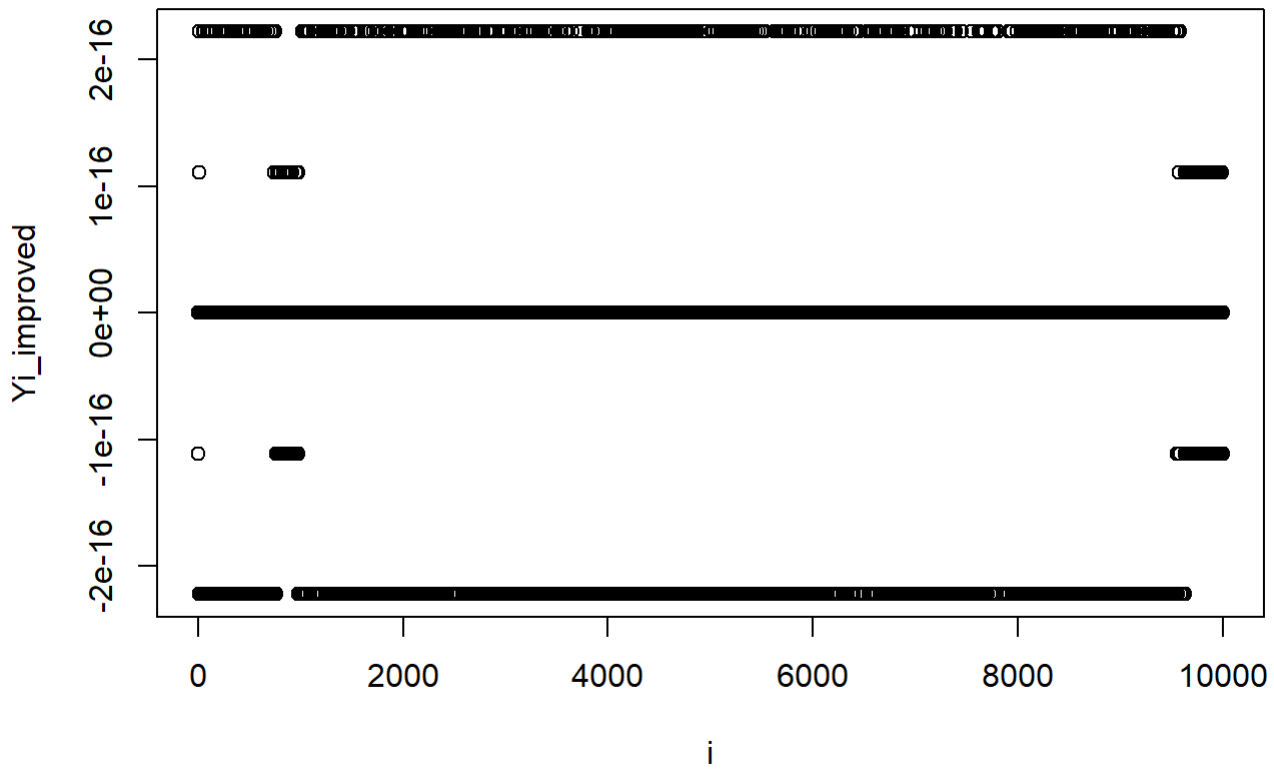
From the above plot we can observe that there is considerable difference between the results of myvar() and var() for most of the cases. The difference varies from -4 to 4. The reason for this difference is “loss of precision” which occurs when cumulative sum(large number) is added to next term which is smaller and when Subtracting numbers of similar magnitudes. So we can conclude that myvar() is not a good function.

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

```
myvar_improved <- function(x){
  mu <- mean(x)  #mean of elements in the vector x
  sum((x-mu)**2)/(length(x)-1)
}
```

Let us compute the difference Yi_improved = myvar_improved(Xi) − var(Xi) and plot dependence Yi_improved on i.

```
Yi_improved<-c()  #initialize Yi_improved to the store values of differences
for (i in 1:length(vector_x)){
  Xi<- vector_x[1:i]          #subset of vector_x
  Yi_improved[i]<- myvar_improved(Xi)-var(Xi)
}

#remove Yi_improved[1] because variance of one element cannot be calculated
Yi_improved = Yi_improved[-1]
plot(2:10000, Yi_improved, xlab = "i") #ignoring i=1 because variance cannot be calculated for o
ne element
```



In the above plot we can see that the difference is almost zero. From this we can say that the function myvar_improved() is giving the same result as var().

Question 4: Binomial coefficient

The binomial coefficient “n choose k” is defined as

$$\binom{n}{k} := \frac{n!}{k!(n-k)!} = \frac{(k+1)(k+2)\dots(n-1)n}{(n-k)!}$$

where n and k are an arbitrary pair of integers satisfying 0 ≤ k ≤ n. Consider the three below R expressions for computing the binomial coefficient. They all use the prod() function, which computes the product of all the elements of the vector passed to it.

A. prod (1 : n) / (prod (1 : k) ∗ prod (1 : (n−k)))
B. prod ((k+1) : n) / prod (1 : (n−k))
C. prod (((k+1) : n) / (1 : (n−k)))

Lets write R functions for each expression

```
A<-function(n,k) prod(1:n)/(prod(1:k)*prod(1:(n-k))) #expression A
B<-function(n,k) prod((k+1):n) / prod(1:(n-k)) #expression B
C<-function(n,k) prod(((k+1):n)/(1:(n-k))) #expression C
```

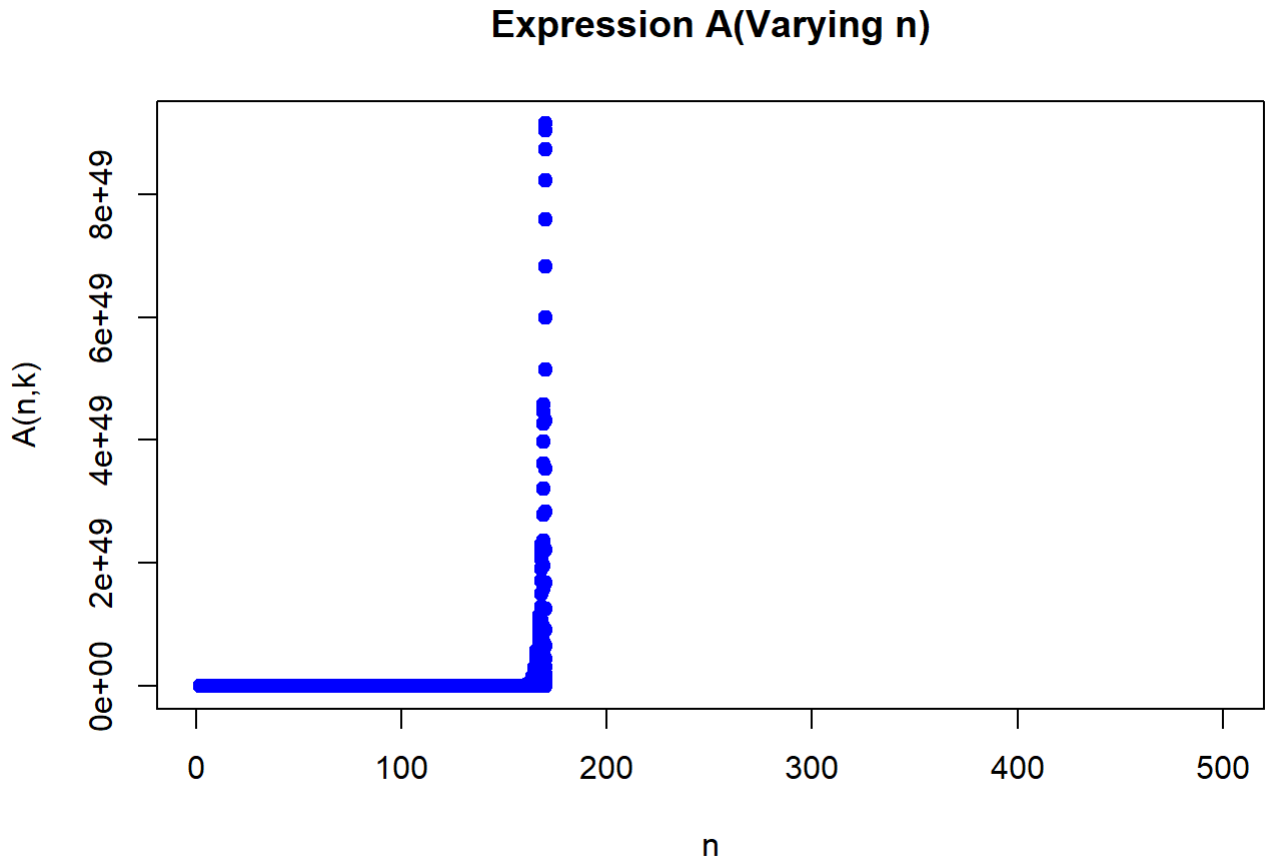
1.Even if overflow and underflow would not occur these expressions will not work correctl for all values of n and k. Explain what is the problem in A, B and C respectively.

In the expressions A and B the problem comes when n=k. When n=k the true value should be 1 but in the expressions the denominator part becomes 0 and output of the expressions will be “Inf”. And additionally expression A fails when k=0 and n!=k too.

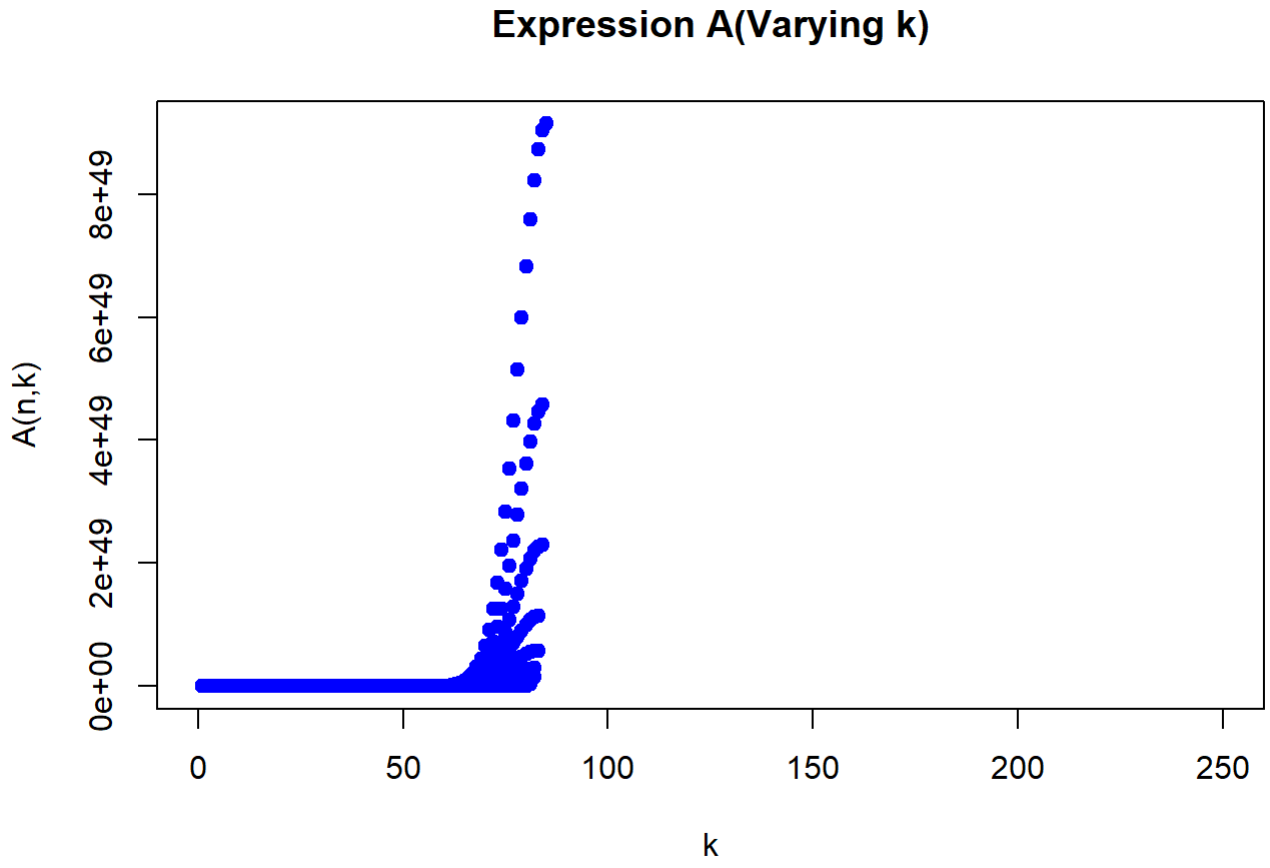
Similarly in the expression C the problem comes when n=k. When n=k, one of the elements in the denominator will become 0 and vector division gives “Inf” and the ultimate output of the expression is “Inf”.

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 overflow occurs. Graphically present the results of your experiments.

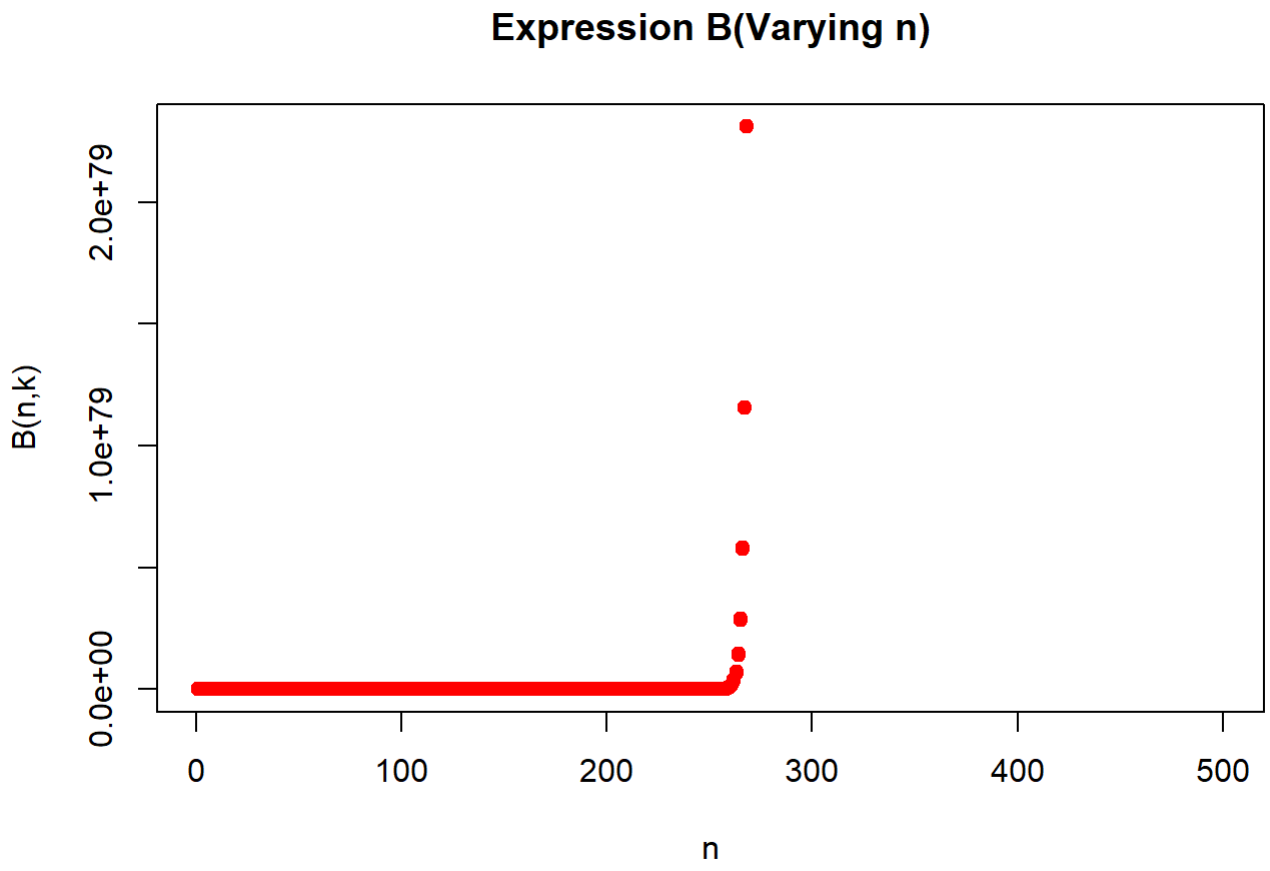
```
#to plot output of three expressions(A,B,C) and to compare with true output (choose(n,k))
A1<-NULL;B1<-NULL;C1<-NULL;T1<-NULL
k<-NULL
n <- NULL
for (i in 1 :500){   #for different values of n = 1 to 500
  for(j in 1:i%/%2){  #for different values of k= 0 to 250
    A1_temp<-A(i,j)
    A1<-c(A1,A1_temp)
    B1_temp<-B(i,j)
    B1<-c(B1,B1_temp)
    C1_temp<-C(i,j)
    C1<-c(C1,C1_temp)
    T1_temp<-choose(i,j)
    T1 <-c(T1,T1_temp)
    k<-c(k,j)
    n<-c(n,i)
  }
}

plot(n,A1, col="blue", pch=19, main = "Expression A(Varying n)", xlab = "n", ylab = "A(n,k)")
```



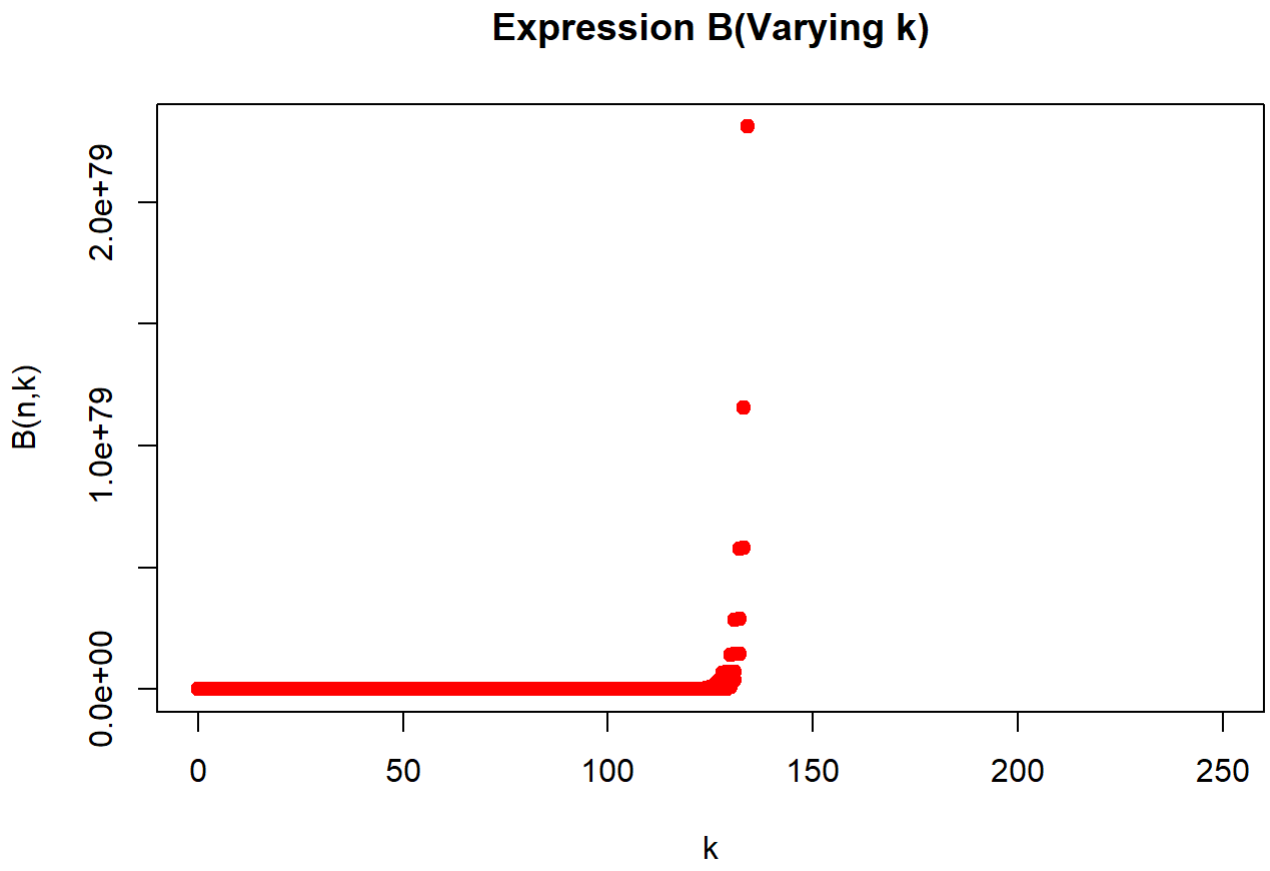

```
plot(k,A1, col="blue", pch=19, main = "Expression A(Varying k)", xlab = "k", ylab = "A(n,k)")
```



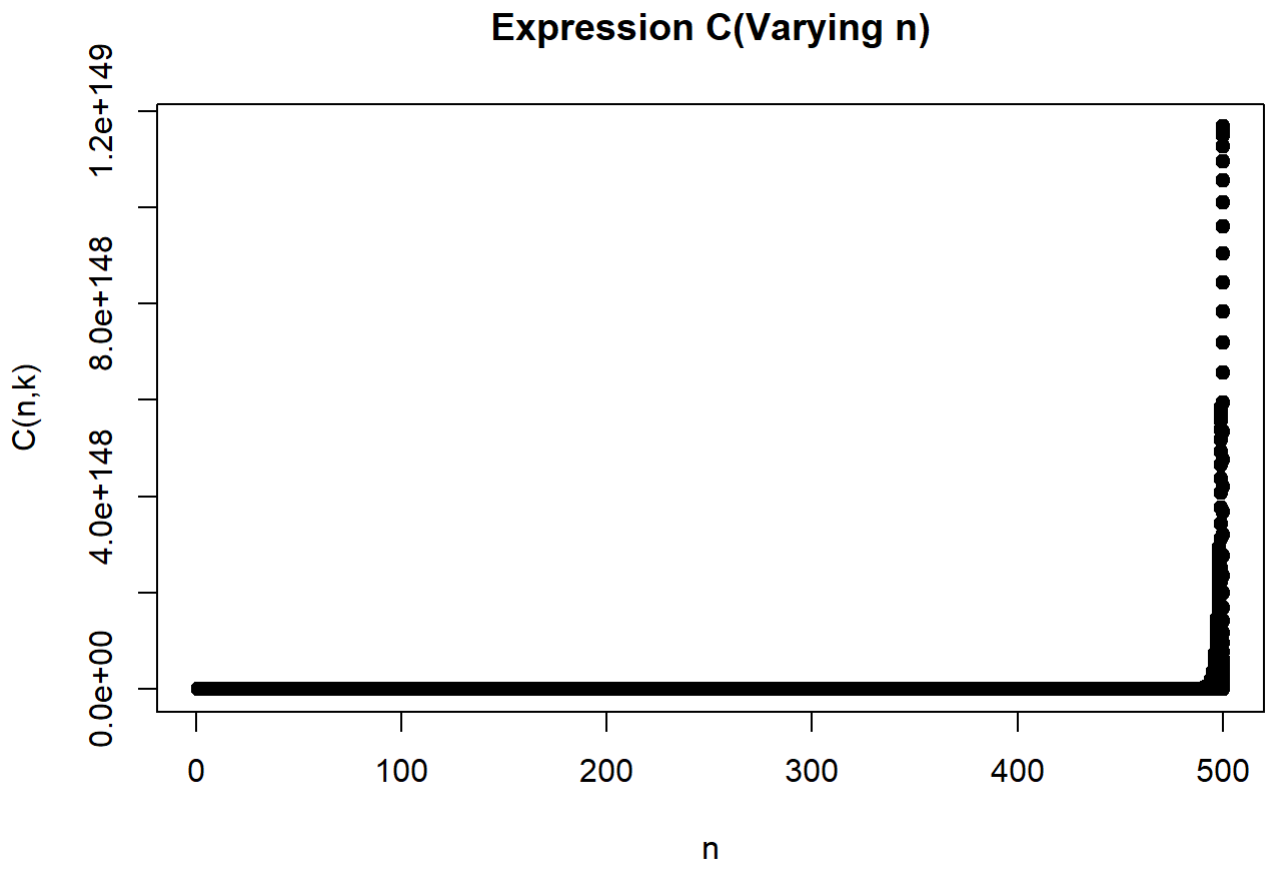

```
plot(n,B1, col="red", pch=19, main = "Expression B(Varying n)", xlab = "n", ylab = "B(n,k)")
```



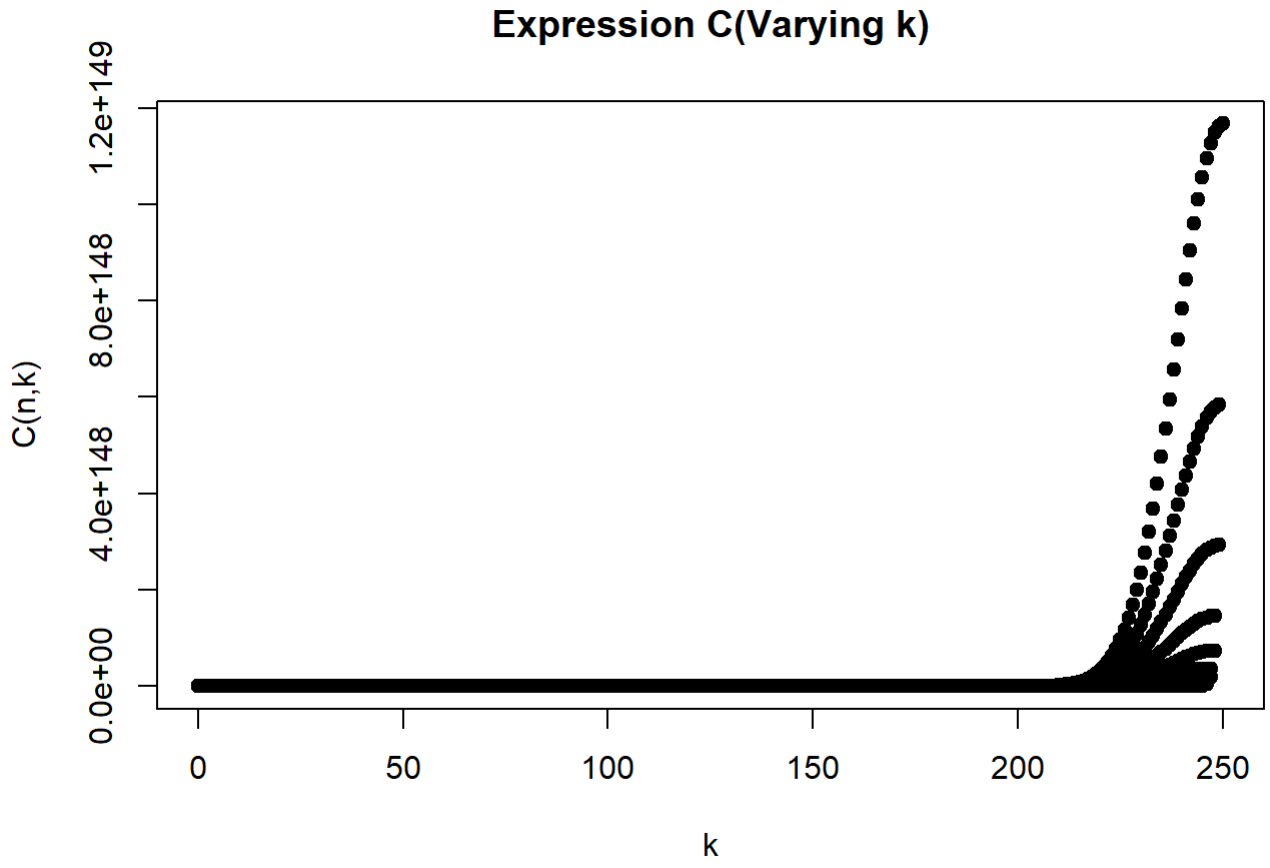

```
plot(k,B1, col="red", pch=19, main = "Expression B(Varying k)", xlab = "k", ylab = "B(n,k)")
```




```
plot(n,C1, col="black", pch=19, main= "Expression C(Varying n)", xlab = "n", ylab = "C(n,k)")
```




```
plot(k,C1, col="black", pch=19, main= "Expression C(Varying k)", xlab = "k", ylab = "C(n,k)")
```



The above plots show graphical representation of results of the three expressions for different values of n and k.

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

All three expressions have overflow. Expression A and B try to calculate the product of integers in the numerator part. For higher value of n the value of product is going beyond the storage limit. Expression C tries to perform vector division which leads a resultant vector with some elements as rational numbers with infinite digits in the fractional part. Expression C also suffers from overflow when n is too large and k is small. Compared to expressions A and B, expression C performs better because it could handle high values of n and k where A and B couldn't.

Appendix

```
knitr::opts_chunk$set(echo = TRUE)
#Snippet 1
x1<-1/3 ; x2<-1/4
if (x1 - x2 == 1/12) {
  print ("Subtraction_is_correct")
}else{
  print ("Subtraction_is_wrong")
}
#Snippet 2
x1<-1  ; x2<-1/2
if (x1-x2==1/2) {
  print ("Subtraction_is_correct" )
}else{
  print ("Subtraction_is_wrong")
}
#Snippet 1 - Improved
x1<-1 / 3 ; x2<-1 /4;
if(all.equal(x1-x2,1/12)){
  print("Subtraction is correct")
}else{
  print("Subtraction is wrong")
}
#A function to calculate derivative of f(x) given epsilon = 10^-15
derivative <- function (x){
  return(((x+10^-15)-x)/10^-15)
}
derivative(x=1)
derivative(x=100000)
#A function to estimate the variance of a input vector x
myvar<- function(x){
  n<- length(x)
  var<-(sum(x^2)-sum(x)^2/n)/(n-1)
  return (var)
}
vector_x<- rnorm(10000,mean=10^8, sd=1)
Yi<-c() #initializing Yi as vector to store the computed difference myvar(Xi) – var(Xi)
for (i in 1:length(vector_x)){
  Xi<- vector_x[1:i]          #subset of vector_x
  Yi[i]<- myvar(Xi)-var(Xi)
}

#remove Yi[1] because variance of one element cannot be calculated
Yi = Yi[-1]
plot(2:10000, Yi, xlab = "i") #ignoring i=1 because variance cannot be calculated for one elemen
t
myvar_improved <- function(x){
  mu <- mean(x)  #mean of elements in the vector x
  sum((x-mu)**2)/(length(x)-1)
}

Yi_improved<-c()  #initialize Yi_improved to the store values of differences
for (i in 1:length(vector_x)){
```

```
  Xi<- vector_x[1:i]          #subset of vector_x
  Yi_improved[i]<- myvar_improved(Xi)-var(Xi)
}

#remove Yi_improved[1] because variance of one element cannot be calculated
Yi_improved = Yi_improved[-1]
plot(2:10000, Yi_improved, xlab = "i") #ignoring i=1 because variance cannot be calculated for o
ne element
A<-function(n,k) prod(1:n)/(prod(1:k)*prod(1:(n-k))) #expression A
B<-function(n,k) prod((k+1):n) / prod(1:(n-k)) #expression B
C<-function(n,k) prod(((k+1):n)/(1:(n-k))) #expression C
#to plot output of three expressions(A,B,C) and to compare with true output (choose(n,k))
A1<-NULL;B1<-NULL;C1<-NULL;T1<-NULL
k<-NULL
n <- NULL
for (i in 1 :500){   #for different values of n = 1 to 500
  for(j in 1:i%/%2){  #for different values of k= 0 to 250
    A1_temp<-A(i,j)
    A1<-c(A1,A1_temp)
    B1_temp<-B(i,j)
    B1<-c(B1,B1_temp)
    C1_temp<-C(i,j)
    C1<-c(C1,C1_temp)
    T1_temp<-choose(i,j)
    T1 <-c(T1,T1_temp)
    k<-c(k,j)
    n<-c(n,i)
  }
}

plot(n,A1, col="blue", pch=19, main = "Expression A(Varying n)", xlab = "n", ylab = "A(n,k)")
plot(k,A1, col="blue", pch=19, main = "Expression A(Varying k)", xlab = "k", ylab = "A(n,k)")
plot(n,B1, col="red", pch=19, main = "Expression B(Varying n)", xlab = "n", ylab = "B(n,k)")
plot(k,B1, col="red", pch=19, main = "Expression B(Varying k)", xlab = "k", ylab = "B(n,k)")
plot(n,C1, col="black", pch=19, main= "Expression C(Varying n)", xlab = "n", ylab = "C(n,k)")
plot(k,C1, col="black", pch=19, main= "Expression C(Varying k)", xlab = "k", ylab = "C(n,k)")
```