Computational Statistics Lab 1

Muhammad Umair

Usama Nadeem

Question 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"
```

The out put of the above code is “subtraction is wrong”. Because the output of x1-x2 is 0.083333333 and 1/12 is also 0.083333333.But the problem is shown below

```
options(digits = 22)
 x1<-1/3
 x2<-1/4
 print(x1-x2)
```

```
## [1] 0.08333333333333331482962
```

```
 print(1/12)
```

```
## [1] 0.08333333333333332870740
```

we can use round() function or all.equal() function to overcome this problem.As shown below

```
options(digits = 22)
 x1<-1/3
 x2<-1/4

 if(round(x1-x2,7)==round(1/12,7)){
   print("Subtraction is correct")

} else
 {
   print("Subtraction is wrong")
 }
```

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

```
options(digits = 22)
 x1<-1
 x2<-1/2
 if(x1-x2==1/2){
   print("Subtraction is correct")

 } else
 {
   print("Subtraction is wrong")
 }
```

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

The result of this code is fine.

Question 2: Derivative

Write your own R function to calculate the derivative of f(x) = x in this way with E = 10−15

```
derivative<-function(x)
 {
   d<-((x+10^-15)-x)/10^-15
   return(d)
 }
```

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

```
 derivative(1)
```

```
## [1] 1.110223024625156540424
```

```
derivative(100000)
```

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

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

The actual value of derivative of 1 should be 1 but we obtained the derivative of 1 as 1.110223024625156540424 because 10^-15 is very small value and it cause underflow error in addition and division. In second case the derivative of 100000 is 0 but it should be 1. because 100000 is too large value as compare to 10^-15 and due to underflow it is rounded of as zero and loss of percision and numerator becomes zero and finally the derivative is zero.

Question 3: Variance.

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

```
myvar<-function(x)
{ sum1<-0
  n<-(1/(length(x)-1))
  sum1<- sum(x^2)
  ex2<-(sum(x))^2
  n2<-1/length(x)
  varience<-n*(sum1-(n2*ex2))
  print(varience)
}
```

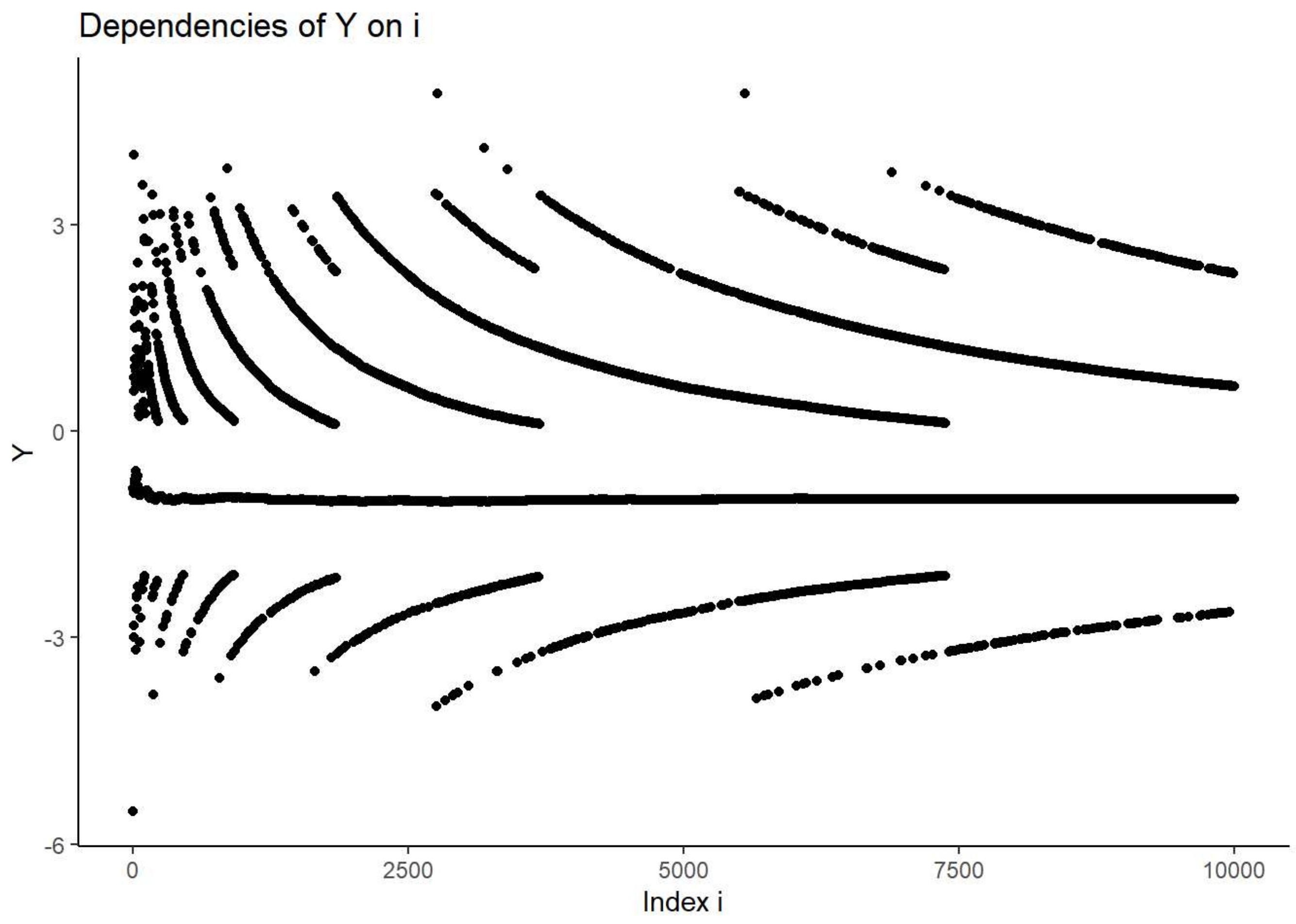
Generate a vector x = (x1,…, x10000) with 10000 random numbers with mean 10^8 and variance 1.

```
std<-1
 varience<-std^2
 options(digits = 22)
rnd<- rnorm(10000, mean=10^8, sd=varience)
```

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?

```r
library(ggplot2)
 ggplot2::ggplot() + geom_point(aes(x = 1:10000, y = y)) + ggplot2::theme_classic() +
 labs(title = "Dependencies of Y on i") + xlab("Index i") + ylab("Y")
```

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

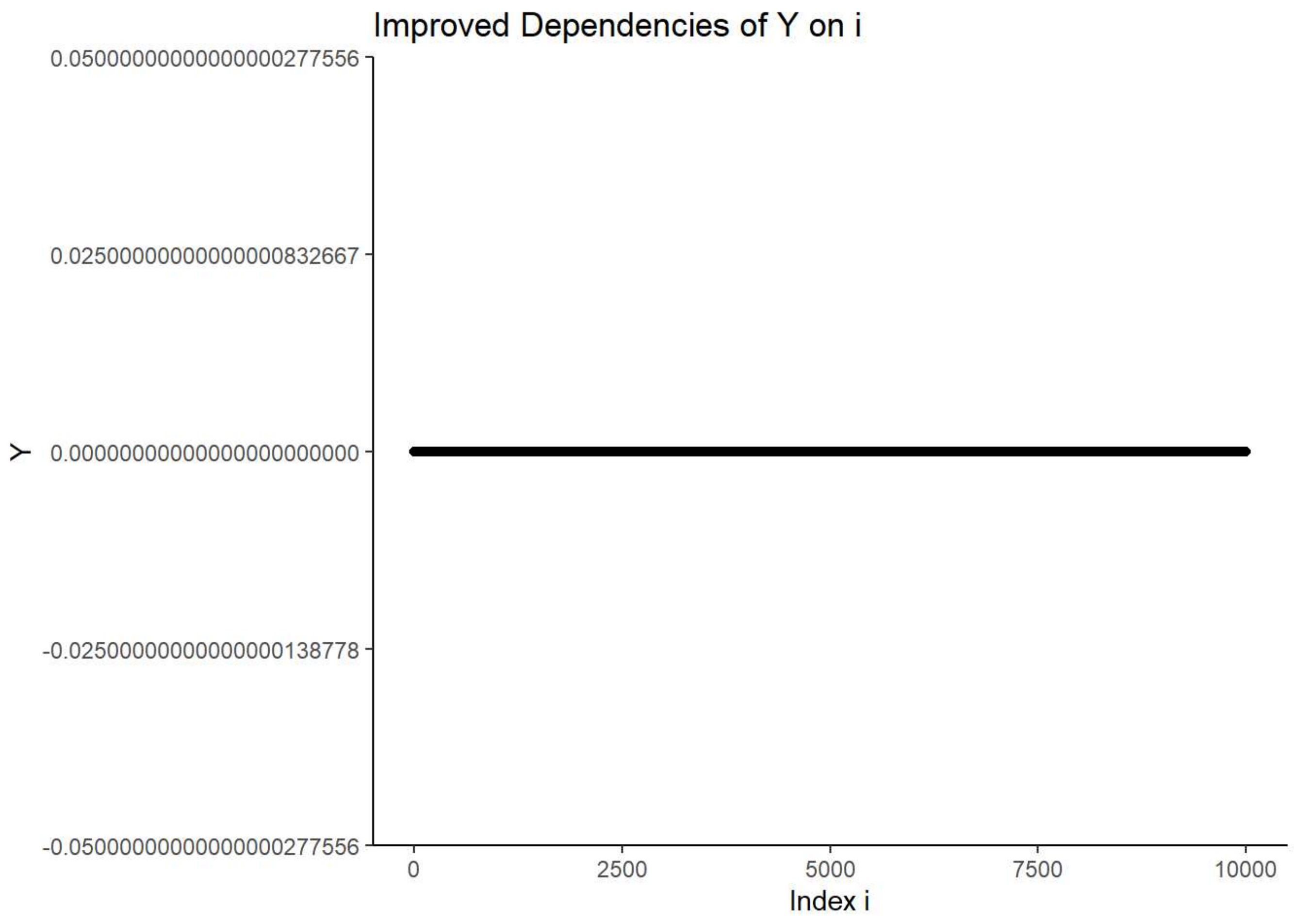


The variance is calculated by a built-in function and our function is different, so when we subtract variance from our calculated variance it causes some loss of precision and it leads towards inaccurate results.

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

```
improvement<-function(new_x)
 {
   d<-length(new_x)-1
   improve<-sum((new_x-mean(new_x))^2)/d
   return(improve)
 }
 new_y<-c()
 for (j in 1:length(rnd)) {
   new_y[j]<-round(improvement(new_x=rnd[1:j])-var(rnd[1:j]),digits=7)
 }
 ggplot2::ggplot() + geom_point(aes(x = 1:10000, y = new_y)) + ggplot2::theme_classic() +
   labs(title = "Improved Dependencies of Y on i") + xlab("Index i") + ylab("Y")
```

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



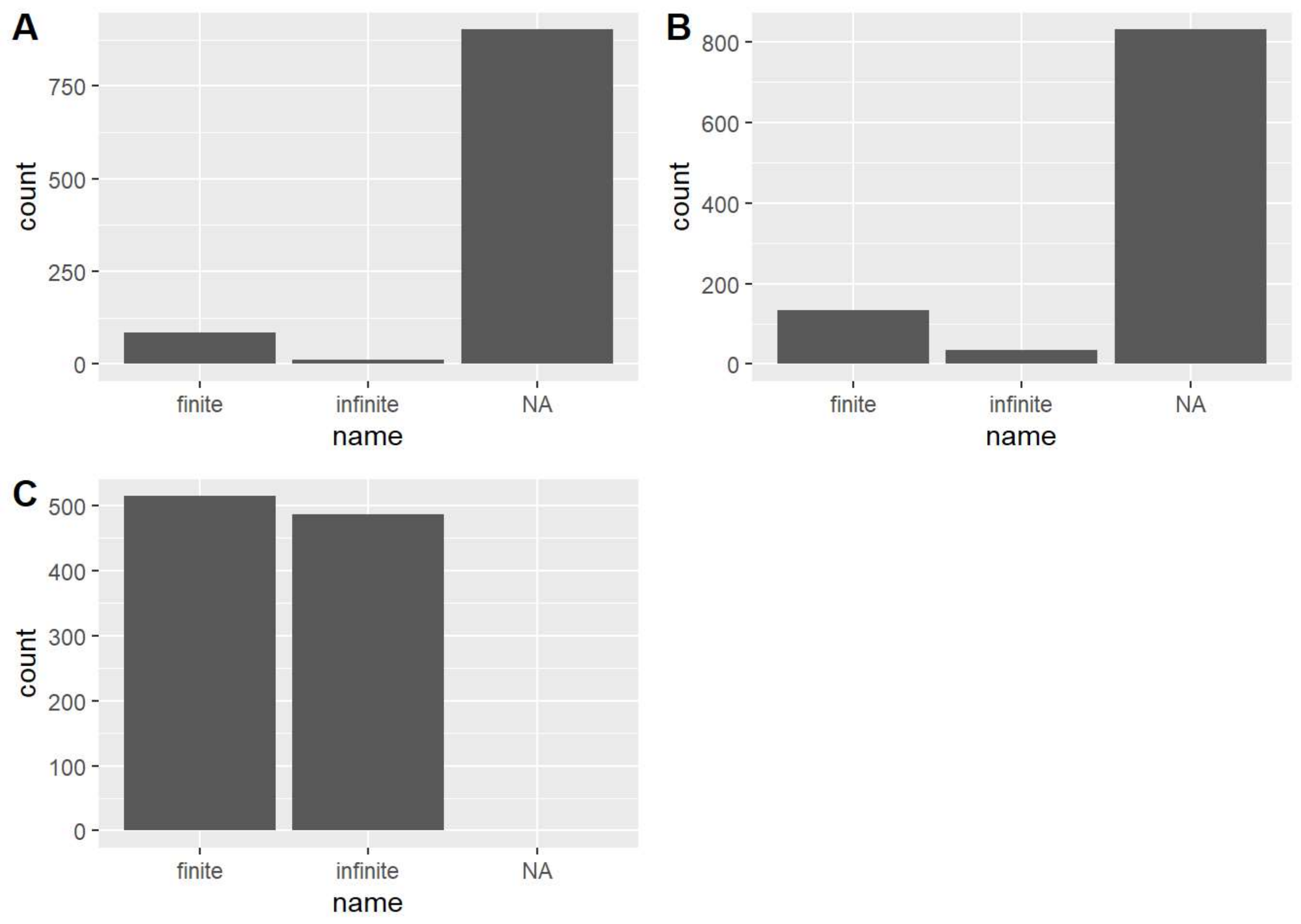
This is improved function and this function gives same varience as var() function gives.

Question 4: Binomial coefficient

```
binomial_coefficient <- function(n, k) {
  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)))
  choose_function <- choose(n,k)
  result_vector <- c(A, B, C, choose_function)
  names(result_vector) <- c("A", "B", "C", "choose_function")
    return(result_vector)
}
```

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.

```
result_plot<- matrix(list(),ncol = 4)
 for(k in 1:1000){
   n<-k*2
  result_plot <- rbind(result_plot, as.vector(binomial_coefficient(n,k)))
 }
 finitesA <- sum(is.finite(unlist(result_plot[,1])))
 finitesB <- sum(is.finite(unlist(result_plot[,2])))
 finitesC <- sum(is.finite(unlist(result_plot[,3])))

 naA <- sum(is.nan(unlist(result_plot[,1])))
 naB <- sum(is.nan(unlist(result_plot[,2])))
 naC <- sum(is.nan(unlist(result_plot[,3])))

 infinitesA <- sum(is.infinite(unlist(result_plot[,1])))
 infinitesB <- sum(is.infinite(unlist(result_plot[,2])))
 infinitesC <- sum(is.infinite(unlist(result_plot[,3])))

dfA <- data.frame(name = c("finite","infinite","NA") ,count = c(finitesA, infinitesA, naA))
dfB <- data.frame(name = c("finite","infinite","NA") ,count = c(finitesB, infinitesB, naB))
dfC <- data.frame(name = c("finite","infinite","NA") ,count = c(finitesC, infinitesC, naC))

p1 <- ggplot2::ggplot(dfA, aes(x = name, y = count)) + geom_bar(stat='identity')
p2 <- ggplot2::ggplot(dfB, aes(x = name, y = count)) + geom_bar(stat='identity')
p3 <- ggplot2::ggplot(dfC, aes(x = name, y = count)) + geom_bar(stat='identity')

cowplot::plot_grid(p1,p2,p3, labels = c("A","B","C"))
```



we used value of k from 1:1000 and n=k*2. we calculated the binomial coefficients of these values and we concluded that none of the three expression gives correct result on all given values of n and k. In some cases if there is zero in denominator then it gives Infinite value and if both nominator and denominator becomes zero then it gives NaN value. because prod function gives zero result for 0 factorial but mathematically it is 1.

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.

From the above graph it can be seen that there is overflow and underflow problem.At some initial values of k and n all three expressions works fine but when k and n increases and goes larger then C expression works fine.

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

Expression A and B both have overflow and underflow problems in many values but on our tested values of n and k Expression C works better then A and B.C can experience more overflow if we pass more larger values.