Computational Statistics (732A90) Lab01

Christophoros Spyretos, Marc Braun, Marketos Damigos, Patrick Siegfried Hiemsch & Prakhar

2021-11-30

Question 1

Snippet 1

```r
x1 <- 1/3 ; x2 <- 1/4

if (x1-x2 == 1/12){
  print("Subtraction is correct.")
  }else{
    print("Subtraction is wrong.")
  }
```

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

1.The message we received is that the subtraction is wrong, but in mathematics, we would expect that the subtraction to be correct. The variable x1 has the value of a periodic number in a decimal figure, 0.33333…… . The problem is that the floating-point number, sometimes called "real", does not always naturally correspond to the actual numbers. Thus x1 is not stored correctly in the environment. This happens because of the memory capacity, which stores 32 or 64 bits.

2.A way to improve this code is to use the singif() function, which returns integer values if the number of requested significant digits is less than the number of digits in front of the decimal separator. We could also set the number of digits we want inside the function, e.g. signif(x1-x2, digits = 5).

```r
x1 <- 1/3 ; x2 <- 1/4

if ( signif(x1-x2) == signif(1/12)){
  print("Subtraction is correct.")
  }else{
    print("Subtraction is wrong.")
  }
```

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

Another way to improve this code is to use the all.equal() function, which compares the "near equality" of x1 - x2 and 1/2.

```r
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"
```

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.The message that we received is that the Subtraction is correct, as is expected in mathematics. Thus there is not any problem with the floating-point number. Both 1 and 1/2 do have a maximum of one digit after the separator, which is why no rounding issues occur.

2.There is not any need for improvements.

Question 2

Task 1

Writing your own R function to calculate the derivative of f(x) = x with

$$e = 10^{-15}$$

.

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

Task 2

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

```
my_derivative(1)
```

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

```
my_derivative(100000)
```

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

Task 3

The derivative of $f(x) = x$ is $f'(x) = 1$, which means that regardless the x the derivative should always be 1. However, in this case the function returns different outputs. For the x = 1, the output is 1.110223 and for x = 100000 the output is 0.

For x = 1, the reason that the output is 1.110223 is that the value of epsilon is not small enough. Thus adding epsilon to 1, the environment stores the decimals.

For x = 100000, the reason that the output is 0 is because of the underflow effect. The value of epsilon is minimal, thus adding it to 100000, the result will be 100000; this happens because the environment cannot store the number 100000 with the decimals of epsilon.

Question 3

Task 1

Writing our own R function, myvar.

```
myvar <- function(x){
  n <- length(x)
  s1 <- sum(x^2)
  s2 <- (1/n)*((sum(x))^2)
  variance <- (1/(n-1))*(s1-s2)
  return(variance)
}
```

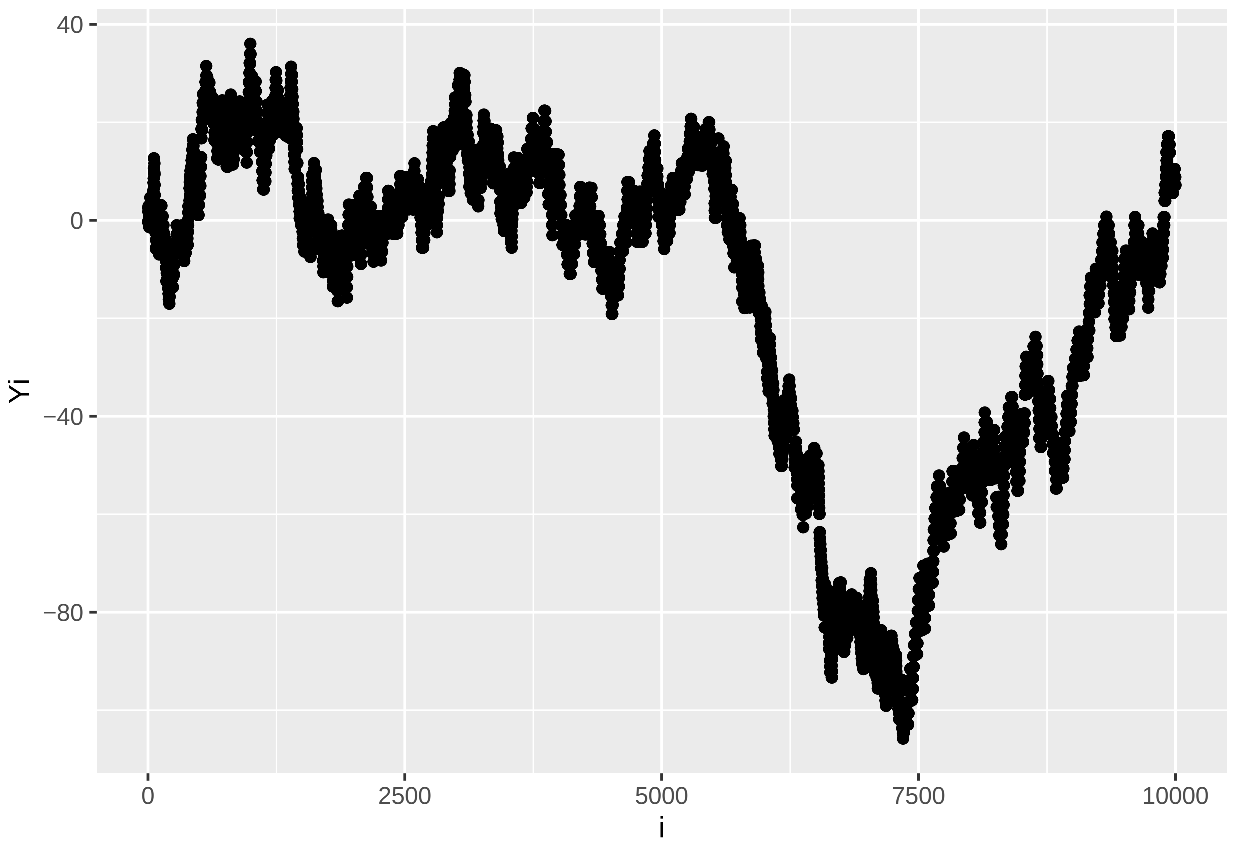
Task 2

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

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

Task 3

```
Y_i <- c()

for ( i in 1:length(x)){
  Y_i[i] <- myvar(x[1:i]) - var(x[1:i])
}

df1 <- data.frame( "i" = 1:length(Y_i), "Yi" = Y_i)

library(ggplot2)

plot1 <- ggplot(df1, aes( x = i, y = Yi)) + geom_point()

plot1
```



We could draw from the plot that the myvar() function is not a good estimator because we would expect that all the Yi values to be equal to 0. However, most of the observations are concentrated around -1, and the rest are either positive or negative.

The reason behind this behaviour is that we generate a vector, in which the values are vast numbers with decimals because of the mean value and the sd, which are equal to 10^8 and 1, respectively. Thus, this leads to an overflow problem in the summations (s1 & s2 in this example) because when squaring a large number, it might lead to a loss of significant digits. Because not all digits of s1 and s2 can be stored and because the two numbers are of the same magnitude, their difference evaluates to zero. As the `var` function returns values of around 1 for our data, the difference which is plotted above ends up centred around -1.

There is also a problem with the formula when $n = 1$, as a division by zero is not defined.

Task 4

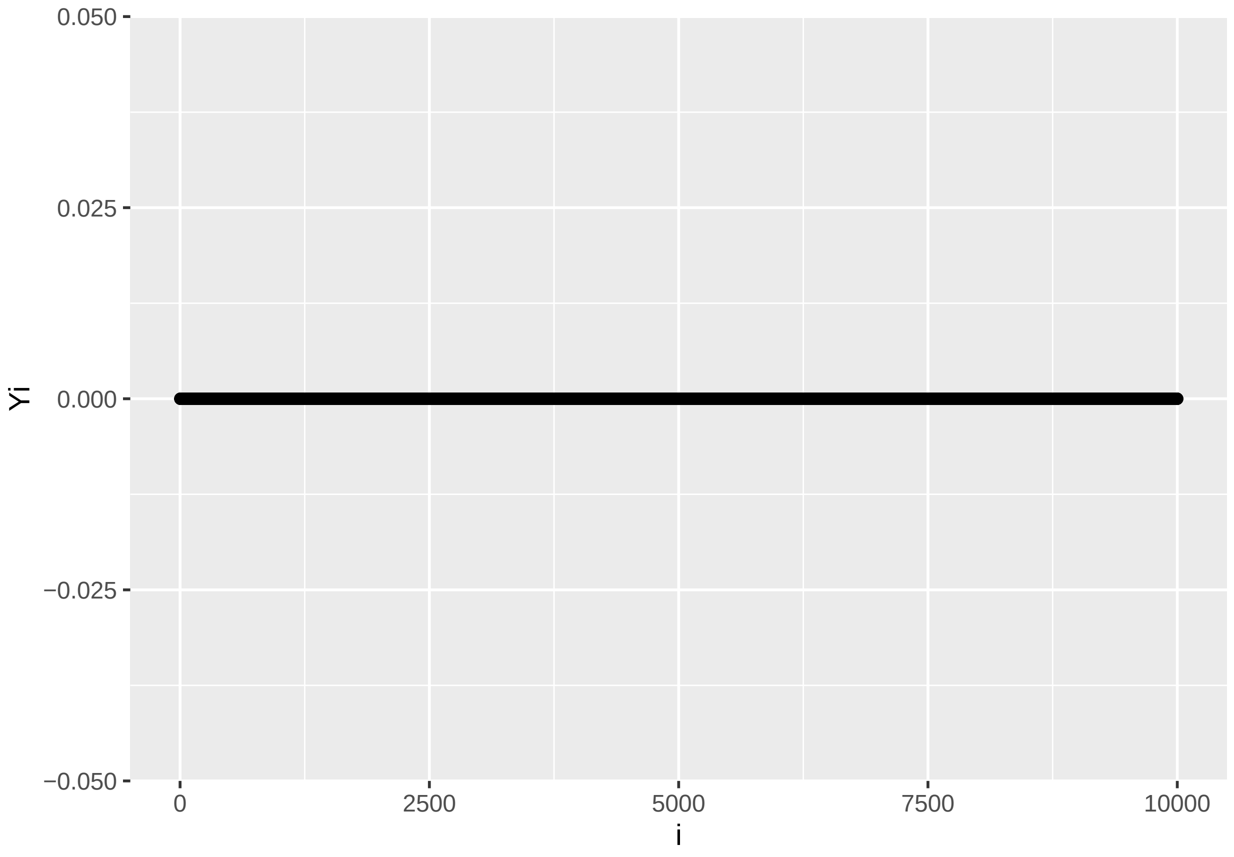
In statistics we generally use the sampling variance for computing the variance. So implementing the sampling variance formula is a reasonable improvement.

Sampling Variance formula: $S^2 = \frac{1}{n-1}\sum_{i=1}^{n}(Xi - \bar{X})^2$

We expect the error to be smaller because before squaring the vast x values, the mean of all the x values is subtracted so the x values get centred around zero. Because the variance is relatively small, the centred values themselves are relatively small and therefore squaring these values will not lead to overflow issues.

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

Y2_i <- c()

for ( i in 1:length(x)){
  Y2_i[i] <- myvar2(x[1:i]) - var(x[1:i])
}

df2 <- data.frame( "i" = 1:length(Y2_i), "Yi" = Y2_i)

plot2 <- ggplot(df2, aes( x = i, y = Yi)) + geom_point()

plot2
```

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



It is evident from this plot is the sampling variance formula is more similar to the var() function compare to the previous function.

Question 4

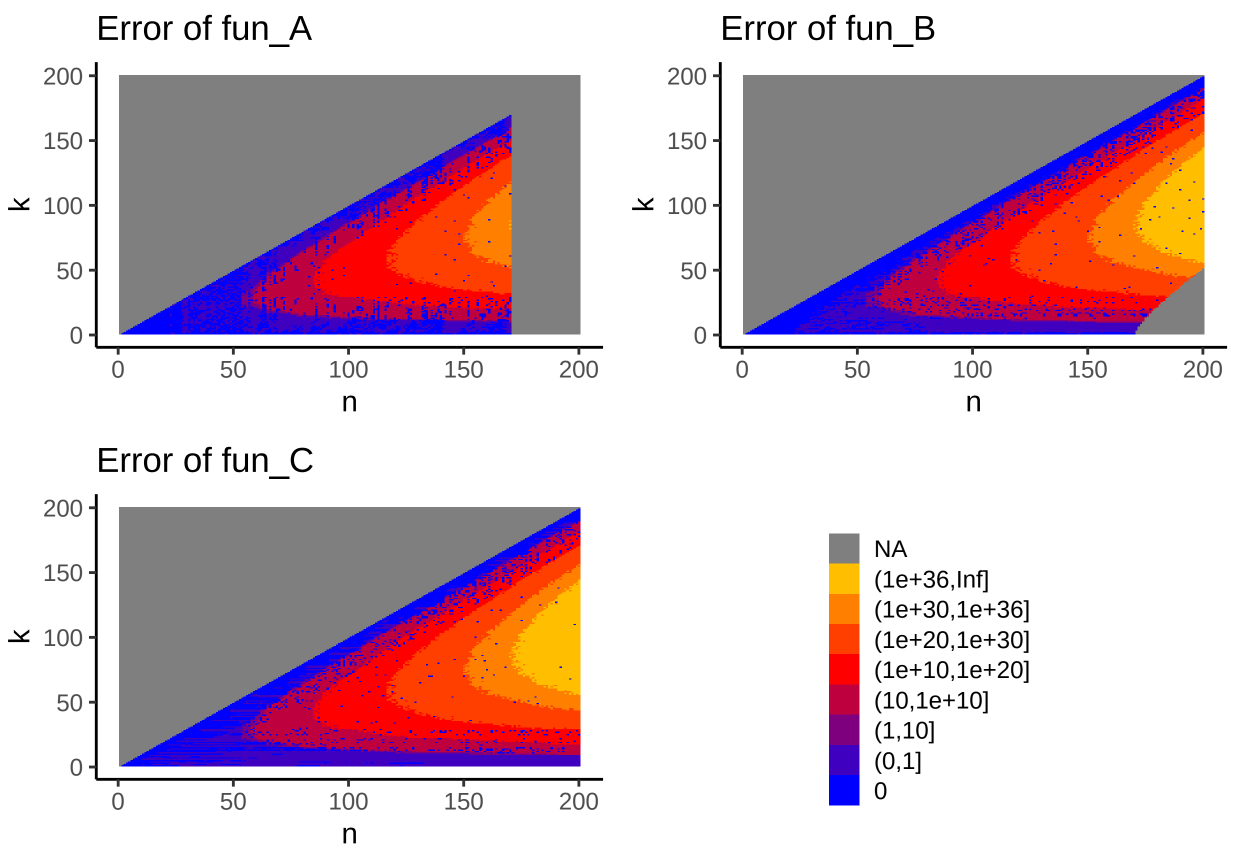
Task 1

The problems occur when $k = 0$ or $n = k$. More specifically, when $k = 0$, the output of the first expression is equal to Inf, but it should be equal to 1. That happens because *prod(1:k)* is equal to 0 instead of 1. Moreover, when $n = k$ all expressions are equal to Inf because *prod(1:(n-k))* is to equal to 0 for A and B; in expression C *1:(n-k)* returns the vector (1,0), thus *((k+1):n) / (1:(n-k))* is a vector, in which the last element is equal to Inf.

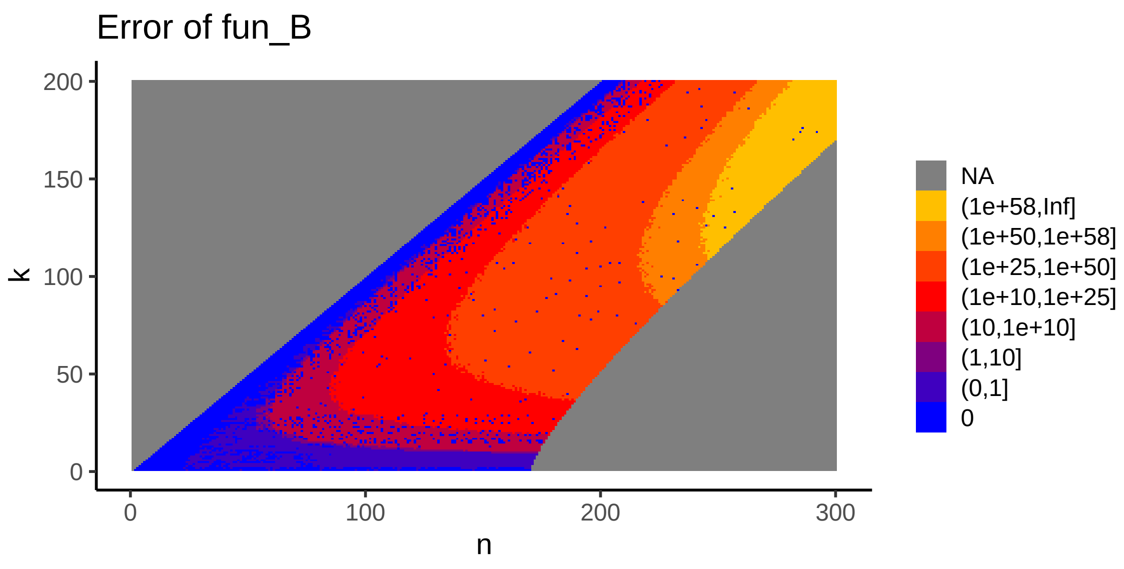
Task 2

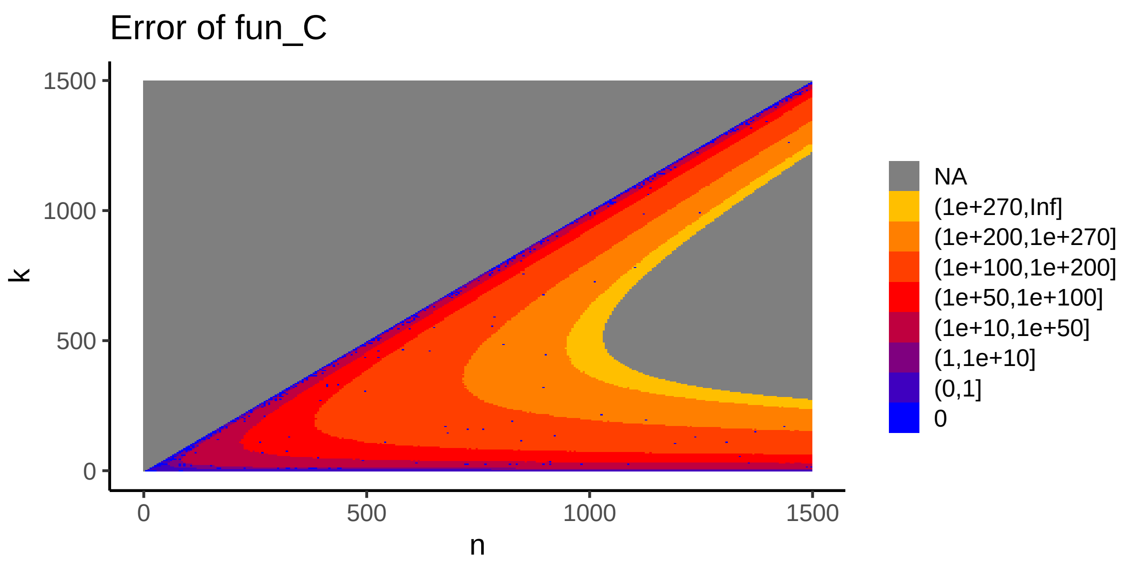
```r
fun_A <- function(n, k){
  return(prod(1:n) / (prod(1:k) * prod(1:(n-k))))
}

fun_B <- function(n, k){
  return(prod((k+1):n) / prod(1:(n-k)))
}

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

The three plots below show the absolute difference between the build-in `choose` function for calculating the binomial coefficient and the functions A, B, and C. Blue points indicate a value of 0, whereas yellow points represent absolute errors in the magnitude of 10^{37}.



It could be seen that for all three approaches A, B, and C, n values smaller than 25 and generally k values which are close to n return the correct result. For both n and k between 0 and 50, all three functions seem to return correct values. It can also be seen that all three functions have points for which they are not defined (other than the trivial ones for which the binomial coefficient is not defined, i.e. $k > n$). The function A seems to only return finite values for $n < 170$.





The colours are scaled differently in the two plots above, which again illustrate blue points as points with an absolute difference between the respective function and the `choose` function of zero. For the Error of fun_B now yellow points represent a magnitude of 10^{60}. Yellow points for fun_C represent an absolute error of magnitude 10^{295}. It can be seen that fun_C returns finite values even for large values of n and k. Nonetheless, the errors become very large and the function starts to return infinite values in the area where $k \approx \frac{1}{3}n$ and $n > 1000$.

Task 3

From the plots above, it could be deducted that all three functions will eventually have overflow problems. More specifically, for function A if $n > 170$ then $n! = Inf$, because R cannot handle the factorial calculation of big values of n. For $n > 170$ the result would be too large for one floating point storage unit (overflow). Moreover, the same problem applies for denominator of function B,if $n - k > 170$ then $(n - k)! = Inf$; also, the numerator of function B will return Inf for large values of n and small values of k. Finally, function C works better than functions A and B because the prod() function is used at the end. Therefore, the divisions are performed before the multiplications, leading to smaller numbers that need to be multiplied. Consequently, the overflow issue occurs for larger values of n. Furthermore, for large values of n and k, the function returns a decimal number with many digits after the separator.

Appendix

R Code to generate the plots

```r
library(tidyverse)
library(gridExtra)
```

```r
n <- seq(1, 200)
k <- seq(1, 200)
fun_values_A = outer(n, k, Vectorize(fun_A)) - outer(n, k, choose)
fun_values_A = apply(fun_values_A,
          1:2, function(x){if (is.infinite(x)) return(NA) else return(abs(x))})
colnames(fun_values_A) = k
rownames(fun_values_A) = n
dat = as.data.frame(fun_values_A) %>%
  rownames_to_column(var="n") %>%
  gather(k, value, -n) %>%
  mutate(k=as.numeric(k),
         n=as.numeric(n),
         value_range = cut(value, c(Inf, 10^36, 10^30, 10^20, 10^10, 10, 1, 0)))
zero_values_indices <- which(dat$value == 0)
prev_levels <- levels(dat$value_range)
dat$value_range <- as.character(dat$value_range)
dat$value_range[zero_values_indices] <- as.character(0)
dat$value_range <- factor(dat$value_range, levels=c("0", prev_levels))
plot_fun_A <- ggplot(dat, aes(n, k, fill=value_range)) +
  geom_raster() +
  scale_fill_manual(values=colorRampPalette(c("blue", "red","yellow"))(9)) +
  theme_classic() +
  theme(legend.position = "none") +
  labs(title = "Error of fun_A")

n <- seq(1, 200)
k <- seq(1, 200)
fun_values_B = outer(n, k, Vectorize(fun_B)) - outer(n, k, choose)
fun_values_B = apply(fun_values_B,
          1:2, function(x){if (is.infinite(x)) return(NA) else return(abs(x))})
colnames(fun_values_B) = k
rownames(fun_values_B) = n
dat = as.data.frame(fun_values_B) %>%
  rownames_to_column(var="n") %>%
  gather(k, value, -n) %>%
  mutate(k=as.numeric(k),
         n=as.numeric(n),
         value_range = cut(value, c(Inf, 10^36, 10^30, 10^20, 10^10, 10, 1, 0)))
zero_values_indices <- which(dat$value == 0)
prev_levels <- levels(dat$value_range)
dat$value_range <- as.character(dat$value_range)
dat$value_range[zero_values_indices] <- as.character(0)
dat$value_range <- factor(dat$value_range, levels=c("0", prev_levels))
plot_fun_B <- ggplot(dat, aes(n, k, fill=value_range)) +
  geom_raster() +
  scale_fill_manual(values=colorRampPalette(c("blue", "red","yellow"))(9)) +
  theme_classic() +
  theme(legend.position = "none") +
  labs(title = "Error of fun_B")

n <- seq(1, 200)
k <- seq(1, 200)
fun_values_C = outer(n, k, Vectorize(fun_C)) - outer(n, k, choose)
fun_values_C = apply(fun_values_C,
          1:2, function(x){if (is.infinite(x)) return(NA) else return(abs(x))})
colnames(fun_values_C) = k
rownames(fun_values_C) = n
dat = as.data.frame(fun_values_C) %>%
  rownames_to_column(var="n") %>%
  gather(k, value, -n) %>%
  mutate(k=as.numeric(k),
         n=as.numeric(n),
         value_range = cut(value, c(Inf, 10^36, 10^30, 10^20, 10^10, 10, 1, 0)))
zero_values_indices <- which(dat$value == 0)
prev_levels <- levels(dat$value_range)
dat$value_range <- as.character(dat$value_range)
dat$value_range[zero_values_indices] <- as.character(0)
dat$value_range <- factor(dat$value_range, levels=c("0", prev_levels))
plot_fun_C <- ggplot(dat, aes(n, k, fill=value_range)) +
  geom_raster() +
  scale_fill_manual(values=colorRampPalette(c("blue", "red","yellow"))(9)) +
  theme_classic() +
  labs(title = "Error of fun_C", fill="") +
  theme(legend.key.size = unit(0.4, 'cm')) +
  guides(fill = guide_legend(reverse=TRUE))




get_legend<-function(myggplot){
    tmp <- ggplot_gtable(ggplot_build(myggplot))
    leg <- which(sapply(tmp$grobs, function(x) x$name) == "guide-box")
    legend <- tmp$grobs[[leg]]
    return(legend)
    }
grid.arrange(plot_fun_A,
             plot_fun_B, plot_fun_C + theme(legend.position = "none"),
             get_legend(plot_fun_C), ncol=2)
```

```
n <- seq(1, 300)
k <- seq(1, 200)
fun_values_B = outer(n, k, Vectorize(fun_B)) - outer(n, k, choose)
fun_values_B = apply(fun_values_B, 1:2,
                function(x){if (is.infinite(x)) return(NA) else return(abs(x))})
colnames(fun_values_B) = k
rownames(fun_values_B) = n
dat = as.data.frame(fun_values_B) %>%
  rownames_to_column(var="n") %>%
```

```r
  gather(k, value, -n) %>%
  mutate(k=as.numeric(k),
         n=as.numeric(n),
         value_range = cut(value, c(Inf, 10^58, 10^50, 10^25, 10^10, 10, 1, 0)))
zero_values_indices <- which(dat$value == 0)
prev_levels <- levels(dat$value_range)
dat$value_range <- as.character(dat$value_range)
dat$value_range[zero_values_indices] <- as.character(0)
dat$value_range <- factor(dat$value_range, levels=c("0", prev_levels))
plot_fun_B_2 <- ggplot(dat, aes(n, k, fill=value_range)) +
  geom_raster() +
  scale_fill_manual(values=colorRampPalette(c("blue", "red","yellow"))(9)) +
  theme_classic() +
  labs(title = "Error of fun_B", fill="") +
  theme(legend.key.size = unit(0.4, 'cm')) +
  guides(fill = guide_legend(reverse=TRUE))

n <- seq(1, 1500, by=5)
k <- seq(1, 1500, by=5)
fun_values_C = outer(n, k, Vectorize(fun_C)) - outer(n, k, choose)
fun_values_C = apply(fun_values_C,
          1:2, function(x){if (is.infinite(x)) return(NA) else return(abs(x))})
colnames(fun_values_C) = k
rownames(fun_values_C) = n
dat = as.data.frame(fun_values_C) %>%
  rownames_to_column(var="n") %>%
  gather(k, value, -n) %>%
  mutate(k=as.numeric(k),
         n=as.numeric(n),
         value_range = cut(value,
                           c(Inf, 10^270, 10^200, 10^100, 10^50, 10^10, 1, 0)))
zero_values_indices <- which(dat$value == 0)
prev_levels <- levels(dat$value_range)
dat$value_range <- as.character(dat$value_range)
dat$value_range[zero_values_indices] <- as.character(0)
dat$value_range <- factor(dat$value_range, levels=c("0", prev_levels))
plot_fun_C_2 <- ggplot(dat, aes(n, k, fill=value_range)) +
  geom_raster() +
  scale_fill_manual(values=colorRampPalette(c("blue", "red","yellow"))(9)) +
  theme_classic() +
  labs(title = "Error of fun_C", fill="") +
  theme(legend.key.size = unit(0.4, 'cm')) +
  guides(fill = guide_legend(reverse=TRUE))

grid.arrange(plot_fun_B_2, plot_fun_C_2, ncol=1)
```