Tarea 3

Augusto Souto-Perez

1. Ejercicio 1

1.1. A

Previamente a realizar el ordenamiento de los datos, se procede a cargar los datos y paquetes necesarios.

```
#VER EN GITHUB, alli se levanta la base con un link.
dir="C:/Users/Usuario/Documents/MEGA/MEGAsync/Maestría/Curso R/"
base=paste(dir, "T3.txt", sep="")
base=read.table(base, header = T)

libs=c("tidyverse", "Rcpp", "microbenchmark", "devtools")
# instalar librerias
lib_nuevas <- libs[!(libs %in% installed.packages()[,"Package"])]
if(length(lib_nuevas)) install.packages(lib_nuevas)

# cargar librerias
load_libs <- function(libraries = libs) for (lib
             in libraries) require(lib, character.only=T)
load_libs(libs)
```

```
## Loading required package: tidyverse
```

```
## -- Attaching packages ---------------------------------------------- tidyverse 1.2.1 --
```

```
## v ggplot2 3.0.0     v purrr   0.2.5
## v tibble  1.4.2     v dplyr   0.7.6
## v tidyr   0.8.1     v stringr 1.3.1
## v readr   1.1.1     v forcats 0.3.0
```

```
## -- Conflicts ------------------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
## Loading required package: Rcpp
```

```
## Loading required package: microbenchmark
```

```
## Loading required package: devtools
```

Luego de ordenar los datos se inspecciona con la función head el data frame para asegurar que los datos están ordenados correctamente.

```
#a. Transformar los datos para que cumplan con las condiciones para estar
#ordenados. El resultado final debe ser un conjunto de datos con 40000
#filas


n_base=base %>% gather(key=repgeno, value=val, REP1.B_b:REP4_MB.total) %>%
  separate(repgeno, c("rep","geno", "alelo"), extra = "merge") %>%
```

```
  spread(data=., alelo, val) %>% mutate(rep=gsub("REP","",rep)) %>%
  arrange(desc(GeneID))

n_base %>% head()
```

```
##          GeneID rep geno b m total
## 1 GRMZM5G899865   1    B 0 0     1
## 2 GRMZM5G899865   1   BM 0 0     1
## 3 GRMZM5G899865   1    M 0 0     2
## 4 GRMZM5G899865   1   MB 0 0     1
## 5 GRMZM5G899865   2    B 0 0     2
## 6 GRMZM5G899865   2   BM 0 0     0
```

1.2. B

Reproducimos la imagen de la tarea. Como no sabemos exactamente que colores toman, los genes del gráfico elegimos de la paleta (mediante inspección visual) a los mas parecidos. El color del alelo B que se elige es "seagreem4", el color del alelo M es "slateblue4" y el de los híbridos es "sienna4".

```
base_graf=n_base %>% mutate(
  logb=log(n_base$b+1),
  logm=log(n_base$m+1)
)

ggplot(base_graf, aes(logm, logb, color=geno,   alpha=0.5))+
  geom_point()+
  facet_wrap(~ rep, scales = 'free',  nrow = 2)+
  labs(x = "Expresión genica de alelo M (en logs)",
       y = "Expresión genica de alelo B (en logs)") +
  theme_bw() +
  scale_color_manual(values = c("seagreen4","sienna3",
                                "slateblue4", "sienna3"))
```

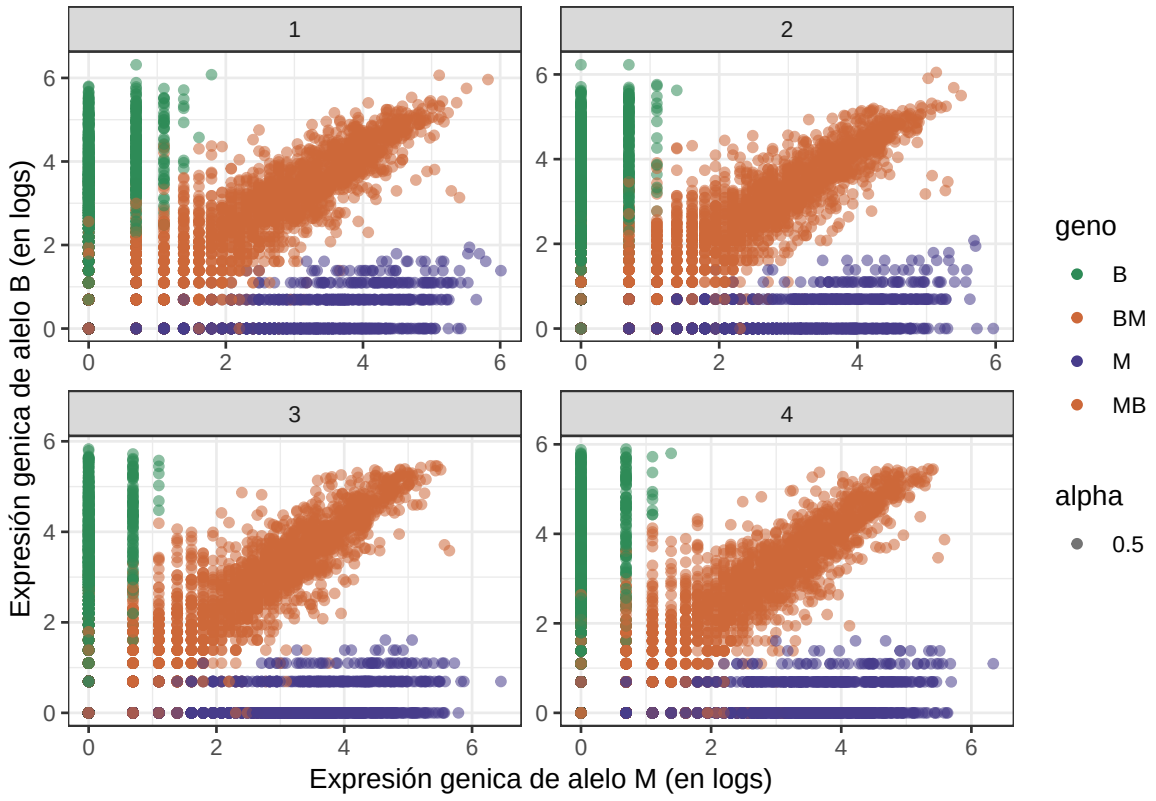

Figura 1: Replica del Grafico 1

1.3. C

Del gráfico se observa que la composición de las diferentes réplicas es muy parecida, dado que los valores (de M y B) en los que toman los alelos son similares para los 4 tipos de planta. Estos valores, en logs, están entre 0 y 6.

También se notan algunas diferencias leves, por ejemplo, se nota que en las replicas 1 y 2, las plantas puras con padres del tipo B, tienen niveles mayores de M que las plantas de tipo 3 y 4 de padres tipo B. Por otro lado, estas últimas muestran mayores niveles rspecto a las replicas 1 y 2 del gen B para los casos en los que ambos padres son tipo M.

2. 2

2.1. A

A continuación se presenta el codigo utilizado para crear las funciones en R (dada en la tarea) y C++. Ambas funciones realizan la misma tarea que el comando prearmado t.test.

```
#COMANDO PARA QUE R DETECTE RTOOLS Y ASI COMPILAR C++ EN RMD
pkgbuild::find_rtools()
```

```
## [1] TRUE
```

```
#FUNCION DE LA TAREA
compara <- function(x, y) {
  m <- length(x)
  n <- length(y)
  # calculo el estadistico de la prueba
  sp <- sqrt(((m-1)*sd(x)^2 + (n-1)*sd(y)^2) / (m+n-2))

  tstat <- (mean(x) - mean(y)) / (sp*sqrt(1/m + 1/n))
   # calculo el p-valor
  2*(1 - pt( abs(tstat), df=n+m-2) )
}
#FUNCION EN CPP
com=cppFunction("

                double comp(NumericVector x, NumericVector y) {
                double m=0;
                double n=0;
                double sp=0;
                double tstat=0;

                m=x.size();
                n=y.size();
                sp=sqrt(((m-1)*pow(sd(x),2) + (n-1)*pow(sd(y),2))/ (m+n-2));


                tstat= (mean(x) - mean(y))/(sp*sqrt((1/m) + (1/n))) ;


                return 2*(1- R::pt(std::abs(tstat), n+m-2,1,0));
                }")
```

Ahora nos aseguramos de que calculen lo mismo:

```
#BUSCO UN GEN ALEATORIO EN LA BASE
n=n_base %>% filter(GeneID == "GRMZM2G105192" )
#PRUEBO LAS 3 FUNCIONES EN ESE GEN
  t.test(n$b, n$m)
```

```
##
##  Welch Two Sample t-test
##
## data:  n$b and n$m
## t = -0.81852, df = 29.17, p-value = 0.4197
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -1.311773  0.561773
## sample estimates:
## mean of x mean of y
##    0.9375    1.3125
```

```
  compara(n$b,n$m)
```

```
## [1] 0.4195148
```

```
  com(n$b, n$m)
```

```
## [1] 0.4195148
```

De los resultados se observa que la funcion escrita en c++ y en R son iguales. Sin embago, la funcion base t.test es ligeramente diferente a las otras dos, lo que se nota en el cuarto digito.

2.2. B

Ahora usamos la libreria microbenchmark para medir el tiempo de computo de cada comando.

Cabe aclarar que las caracterìsticas del laptop utilizado son los siguientes:

-Procesador intel core i 3 M380 2.53 GHz

-Memoria 4 gb ram

-Sistema operativo Windows 7 Ultimate

```
#VELOCIDAD DE LOS COMANDOS "T.TEST"", COMANDO EN R Y COMANDO EN CPP

timer=microbenchmark(
  t.test(n$b, n$m),
  compara(n$b,n$m),
  com(n$b, n$m)
  )

timer
```

```
## Unit: microseconds
##               expr     min       lq      mean   median       uq      max
##   t.test(n$b, n$m) 367.127 376.0420 398.55569 379.2830 385.1595  669.013
##  compara(n$b, n$m) 198.962 204.6345 215.75002 207.8770 213.5500  418.590
##      com(n$b, n$m)  29.986  33.2280  62.92633  40.7245  42.1430 2078.359
##  neval
##    100
##    100
##    100
```

Podemos ver graficamente la performance con el comando autoplot:

```
autoplot(timer)
```

```
## Coordinate system already present. Adding new coordinate system, which will replace the existing c
```

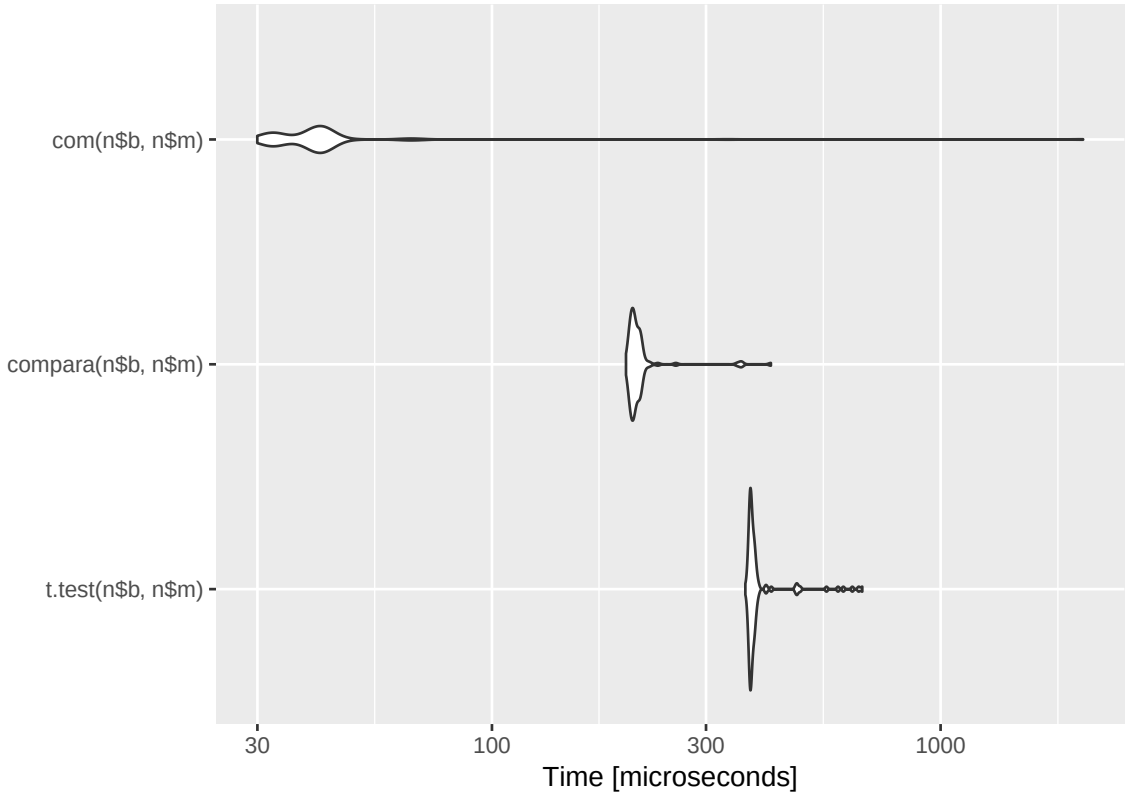

Figura 2: Performance por comando

2.3. C

Como se ve, el comando escrito en cpp es entre 5 y 6 veces mas eficiente en la mediana que el comando escrito en R y entre 9 y 10 veces mas rapido que el comando t.test.

El comando en cpp tambièn tiene una mejor performance en el tiempo minimo de ejecucion, en el cuantil 25, media y cuantil 75. No obstante, el tiempo màximo de ejecucion presenta una reversiòn del orden descrito ya que la mejor performance la tiene t.test, seguido por el comando escrito en R y C++, que tiene un tiempo màximo de ejecución sensiblemente mayor al de los otros dos comandos.

Por lo tanto, se puede ver que los comandos mas eficientes en mediana tienden a ser mas volatiles que los mas lentos. En el caso del comando escrito en c++ hay un valor máximo que es atipico que sube considerablemente el valor de la media.

2.4. D

El problema de hacer la prueba t.test es que probablemente nos llevara mas tiempo ejecutarla dados los valores que mostramos anteriormente. No obstante, como contrapartida de tal desventaja este comando tiene una perfomance mas previsible.

3. 3

3.1. A

La probabilidad de rechazar la hipotesis nula (Error tipo I) en un gen determinado serìa de un 5 %.

3.2. B

Se esperarìa que en el 5 % de esos 2500 genes ocurra el error tipo I. Ergo, en 125 casos se estaria cometiendo dicho error.

3.3. C

Al analizar los p-valores se observa una alta cantidad de NA's, como consecuencia de que en algunos genes la variación de los valores de los genes es nula. La función programada devuelve este resultado, ya que su denominador depende del termino "sp" que es nulo cuando no hay variación en los datos.

La cantidad de NA's es 968, la cantidad de valores por encima de 0.05 es 1282 y la cantidad de valores por debajo del mismo es de 250.

Si se excluyen los NA, y se observa la cantidad de valores que no rechazan la hipótesis nula, observamos que son aproximadamente el 84 % del total. Lo que constituye un valor bajo para afirmar que no hay relacion entre M y B.

```
#GENERO LOS P-VALORES EN TRES PASOS:
#recorto la base en logs de modo que quede segmentada por gen
pieces=base_graf %>% select(-c(2:6)) %>% split(n_base$GeneID)
#genero un vector para alojar los resultados del proximo paso
pval=vector("list", length = length(pieces))
#calculo los p-valores para cada gen
for(i in 1:length(pieces)){
  pval[i]=com(pieces[[i]]$logb, pieces[[i]]$logm)
}

#CUENTA LA CANTIDAD POR ENCIMA Y DEBAJO DEL UMBRAL 0.05 Y NA
pval[which(pval>0.05)] %>% length()
```

```
## [1] 1282
```

```
pval[which(is.na(pval))] %>% length()
```

```
## [1] 968
```

```
pval[which(pval<0.05)] %>% length()
```

```
## [1] 250
```

```
#SACO LOS NA
pval=pval[which(!is.na(pval))]

#PROPORCION DE P-VALORES MAYORES A 0.05
pval[which(pval>0.05)] %>% length()/
```

```
  ( pval[which(pval>0.05)] %>% length()+
   pval[which(pval<0.05)] %>% length() )
```

```
## [1] 0.8368146
```

3.4. D

La corrección de bonferroni implica ajustar el umbral critico de los p-valores a:

$$Pval = \frac{\alpha}{N}$$

Donde α es el nivel de significancia de la prueba y N la cantidad de p-valores analizados.

Para el caso de la tarea, se analizaron los p-valores de la siguiente forma:

```
#GENERO P-VALORES AJUSTADOS POR LA CORRECCION DE BONFERRONI
pval_b=p.adjust(pval, method = "bonferroni")
#CUENTA LA CANTIDAD POR ENCIMA Y DEBAJO DEL UMBRAL 0.05
pval_b[which(pval_b>0.05)] %>% length()
```

```
## [1] 1497
```

```
pval_b[which(pval_b<0.05)] %>% length()
```

```
## [1] 35
```

```
#PORCENTAJE DE P-VALORES MAYORES AL UMBRAL DE BONFERRONI
pval_b[which(pval_b>0.05)] %>% length() /
 ( pval_b[which(pval_b>0.05)] %>% length() +
pval_b[which(pval_b<0.05)] %>% length()     )
```

```
## [1] 0.977154
```

Como se puede ver, usando la corrección de Bonferroni ahora el porcentaje de no rechazos aumenta hasta casi el 98 %.