Programacion R grupo GJJAB

Baptiste Caillé[a], Gastón Barmat[a], Juan Ignacio Gutiérrez Glielmi[a], Jacinta Calle Monzo[a], Agustina Salvarredi[a]

[a]*Facultad de ingenieria Universidad Nacional de Cuyo - Centro Universitario Edifico 6 M5502KAF*

Abstract

This is the abstract.

It consists of two paragraphs.

Keywords: keyword1, keyword2

1. Generar un vector secuencia

1.1. for

```
A <- vector()
for(i in 0:50000){
  A[i+1] <- i*2
}
tail(A)
```

```
## [1]  99990  99992  99994  99996  99998 100000
```

```
head(A)
```

```
## [1]  0  2  4  6  8 10
```

1.2. seq

```
B <- vector()
B<- seq(0,100000,2)
tail(B)
```

```
## [1]  99990  99992  99994  99996  99998 100000
```

```
head(B)
```

```
## [1]  0  2  4  6  8 10
```

*Corresponding author

Email addresses: baptiste.caille.move@gmail.com (Baptiste Caillé), gastonbarmat@gmail.com (Gastón Barmat), juangglielmi@hotmail.com (Juan Ignacio Gutiérrez Glielmi), jacicalle@hotmail.com (Jacinta Calle Monzo), agusalvarredi@gmail.com (Agustina Salvarredi)

1.3. Performance

uso del algoritmo *Biblioteca tictoc "In general, calls to tic and toc start the timer when the tic call is made and stop the timer when the toc call is made, recording the elapsed time between the calls from proc.time."*

```
library(tictoc)
A <- vector()
B <- vector()

tic.clearlog()

tic
```

```
## function (msg = NULL, quiet = TRUE, func.tic = NULL, ...)
## {
##     stim <- get(".tictoc", envir = asNamespace("tictoc"))
##     smsg <- get(".ticmsg", envir = asNamespace("tictoc"))
##     tic <- proc.time()["elapsed"]
##     if (!is.null(func.tic)) {
##         outmsg <- func.tic(tic, msg, ...)
##         if (!quiet)
##             writeLines(outmsg)
##     }
##     push(stim, tic)
##     push(smsg, msg)
##     invisible(tic)
## }
## <bytecode: 0x0000020745342800>
## <environment: namespace:tictoc>
```

```
for(C in 0:50000){
  A[C+1] <- C*2
}
toc(log = TRUE, quiet = TRUE)

tic
```

```
## function (msg = NULL, quiet = TRUE, func.tic = NULL, ...)
## {
##     stim <- get(".tictoc", envir = asNamespace("tictoc"))
##     smsg <- get(".ticmsg", envir = asNamespace("tictoc"))
##     tic <- proc.time()["elapsed"]
##     if (!is.null(func.tic)) {
##         outmsg <- func.tic(tic, msg, ...)
##         if (!quiet)
##             writeLines(outmsg)
##     }
##     push(stim, tic)
##     push(smsg, msg)
##     invisible(tic)
## }
## <bytecode: 0x0000020745342800>
## <environment: namespace:tictoc>
```

```
B <- seq(0,100000,2)
toc(log = TRUE, quiet = TRUE)

unlist(tic.log())
```

```
## NULL
```

2. Serie de Fibonacci

```
w <- vector()
w[1]<- 0
w[2]<- 1
for(var in 0:15)
{
  w[var+3] <- w[var+2]+w[var+1]
}
head(w)
```

```
## [1] 0 1 1 2 3 5
```

2.1. Performance

uso del algoritmo *Biblioteca rbenchmark.*

"The library consists of just one function, benchmark, which is a simple wrapper around system.time. Given a specification of the benchmarking process (counts of replications, evaluation environment) and an arbitrary number of expressions, benchmark evaluates each of the expressions in the specified environment, replicating the evaluation as many times as specified, and returning the results conveniently wrapped into a data frame"

```
library(rbenchmark)

k <- vector()
w <- vector()

w = function(it){
  k[1] <- 0
  k[2] <- 1
  for(i in 0:it){
    k[i+3] <- (k[i+2]+k[i+1])
  }
  return(k)
}


it=197
benchmark(w(it), replications = 1000)
```

```
##    test replications elapsed relative user.self sys.self user.child sys.child
## 1 w(it)         1000    0.12        1      0.06     0.06         NA        NA
```

- *elapsed*: tiempo acumulado

- *relative*: razon con la prueba mas rapida.
- *user.self*: CPU time spent by the current process
- *sys.self*: CPU time spent by the kernel (the operating system) on behalf of the current process. # Algoritmo para la pesadilla de Gauss

```
for(i in 0:5)
{ a<-i
b <-i+1
c <- a+b

print(c)
}
```

```
## [1] 1
## [1] 3
## [1] 5
## [1] 7
## [1] 9
## [1] 11
```

3. Ordenacion de un vector por Metodo Burbuja

A continuacion estan las lineas de codigo para ordenar, de mayor a menor, por metodo bubble un vector de 150 numeros elegidos aleatoriamente entre el 1 y el 1000000:

```
example <- sample(1:1000000,150)

head(example)
```

```
## [1] 894460 613225 614226 430747 698143  52420
```

```
tail(example)
```

```
## [1]  36985 256690 454503 160040 348643 431433
```

```
bubble = function(example){
  n <- length(example)
  for(i in 1:(n-1)){
    for(j in 1:(n-i)){
      if (example[j] < example[j+1]){
        temporal <- example[j]
        example[j] <- example[j+1]
        example[j+1] <- temporal
      }
    }
  }
  return(example)
}

example_ordenada <- bubble(example)

head(example_ordenada)
```

```
## [1] 987231 982305 980157 961128 958011 955476
```

```
tail(example_ordenada)
```

```
## [1] 26063 25301 22183 21507 20826  6967
```

3.1. Performance

uso del algoritmo *Biblioteca Microbenchmark.*

"Microbenchmark serves as a more accurate replacement of the often seen system.time(replicate(1000, expr)) expression. It tries hard to accurately measure only the time it takes to evaluate expr. To achieved this, the sub-millisecond (supposedly nanosecond) accurate timing functions most modern operating systems provide are used."

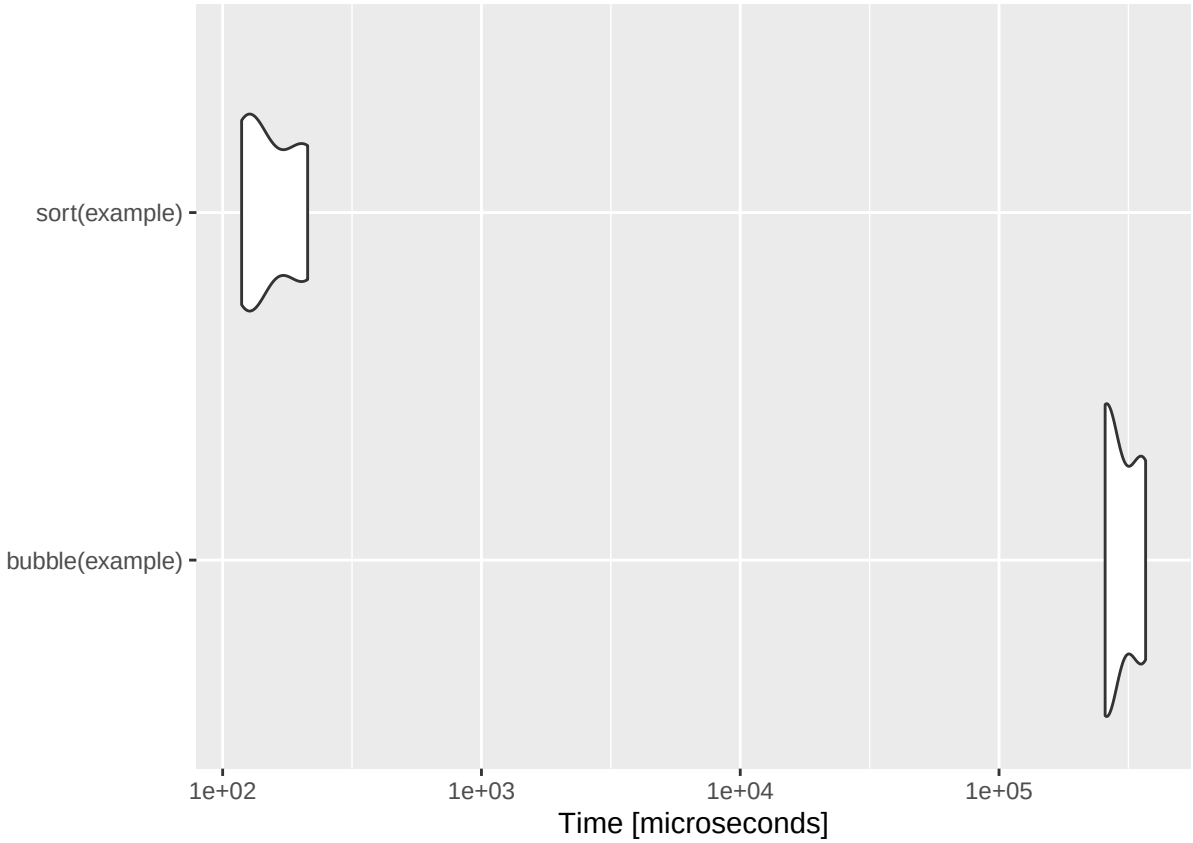
A continuacion vemos las lineas de codigo para calcular el tiempo de ejecucion de el codigo para ordenar por metodo bubble. Tomaremos una muestra de 2000 numeros entre el 1 y el 1000000 Lo compararemos con el comando *sort* de R:

```
library(microbenchmark)

example <- sample(1:1000000,2000)

bubble = function(example){
  n <- length(example)
  for(i in 1:(n-1)){
    for(j in 1:(n-i)){
      if (example[j] < example[j+1]){
        temporal <- example[j]
        example[j] <- example[j+1]
        example[j+1] <- temporal
      }
    }
  }
  return(example)
}


microbenchmark(bubble(example),sort(example), times=5)
```

```
## Unit: microseconds
##             expr      min       lq      mean   median       uq      max neval
##  bubble(example) 275015.4 348202.7 370798.74 379213.1 380950.8 470611.7     5
##    sort(example)    129.1    204.6    251.92    214.4    353.8    357.7     5
```

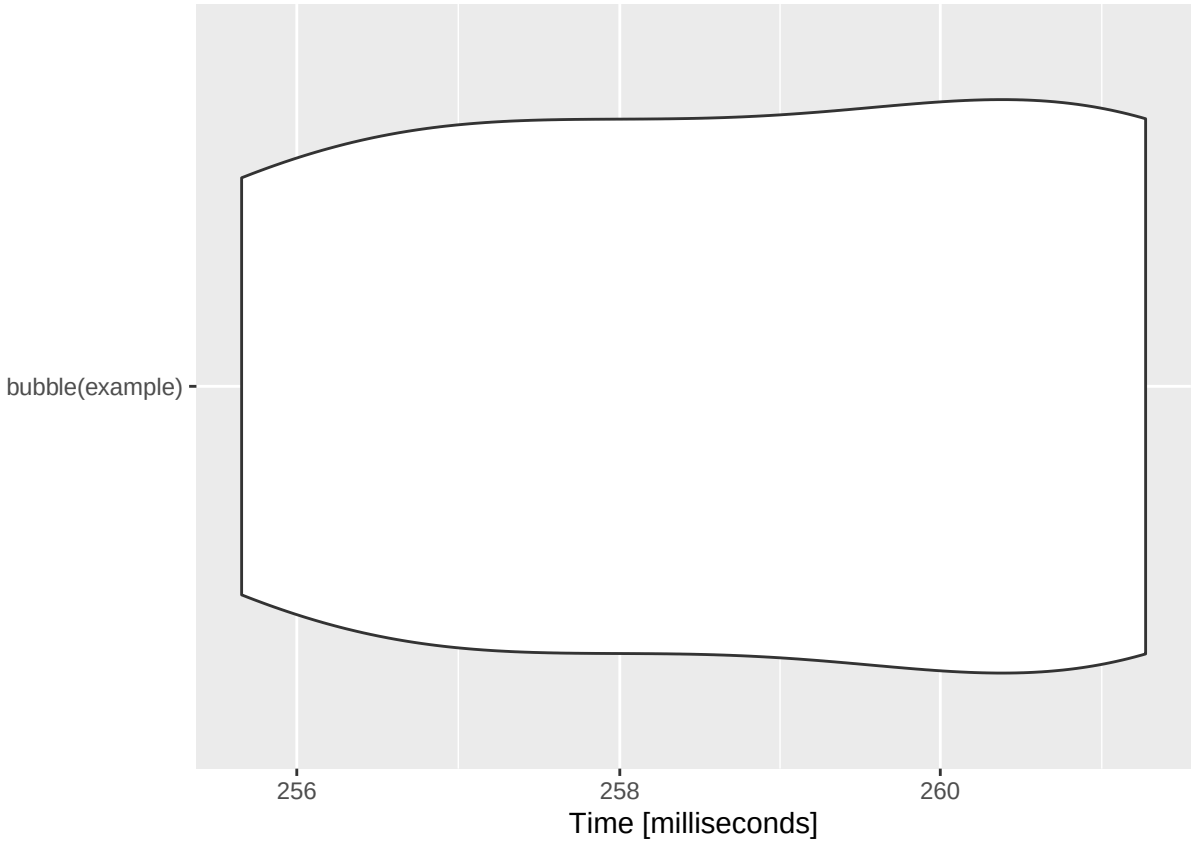
```
library(ggplot2)
autoplot(microbenchmark(bubble(example),sort(example), times=5))
```

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



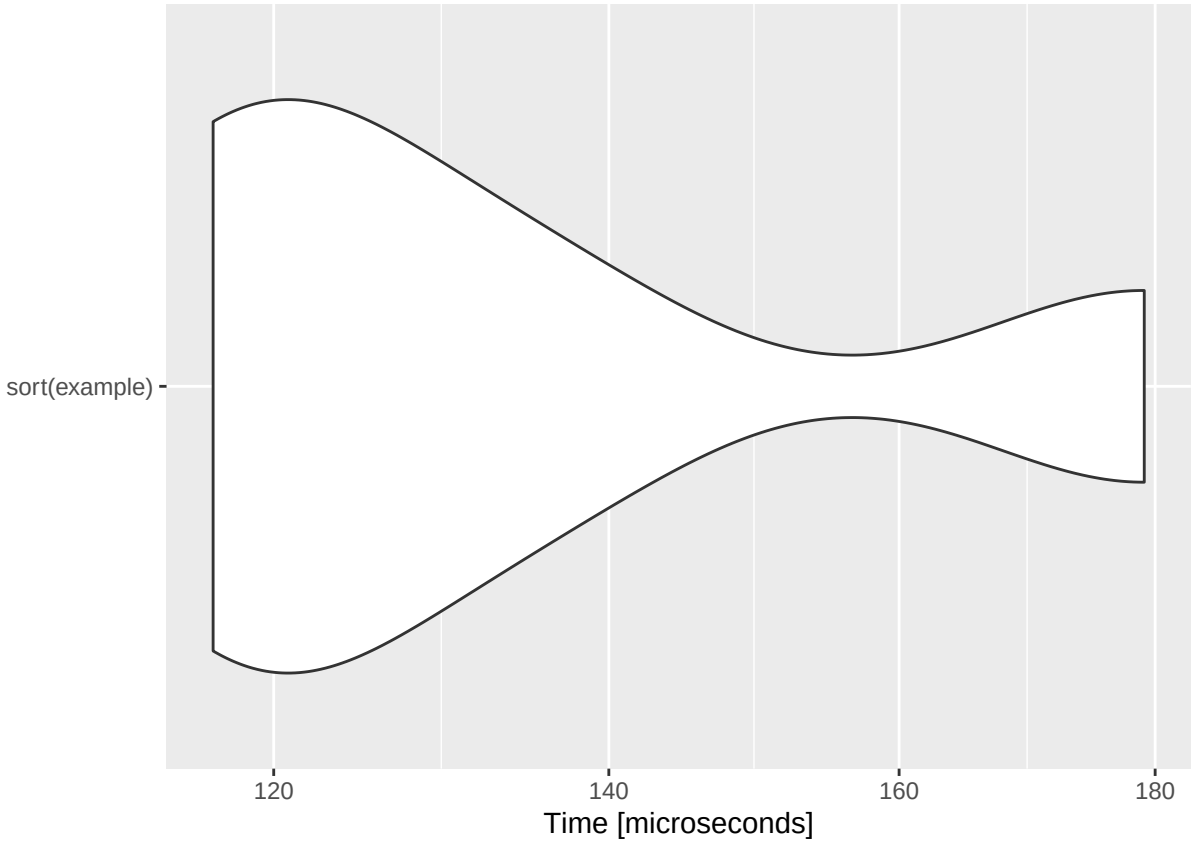

```
autoplot(microbenchmark(bubble(example),times=5))
```

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




```
autoplot(microbenchmark(sort(example),times=5))
```

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



Progresión geométrica del COVid-19 ## Modelado matemático de una epidema

```
library(readr)
location <- getwd()
setwd(location)
casos_A <- read_delim("casos.csv", ";", escape_double = FALSE, trim_ws = TRUE, skip = 1)
```

```
## Rows: 34 Columns: 3
## -- Column specification --------------------------------------------------------
## Delimiter: ";"
## chr (1): Fecha
## dbl (1): Casos
## lgl (1): E_P+1
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
#Estadística de casos
summary(casos_A$Casos)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1.00   36.75  245.50  514.71  995.50 1715.00
```

```r
m <- length(casos_A$Casos)
F <- (casos_A$Casos[2:m])/(casos_A$Casos[1:m-1])
#Estadísticos de F
mean(F,na.rm = TRUE)
```

```
## [1] 1.350739
```

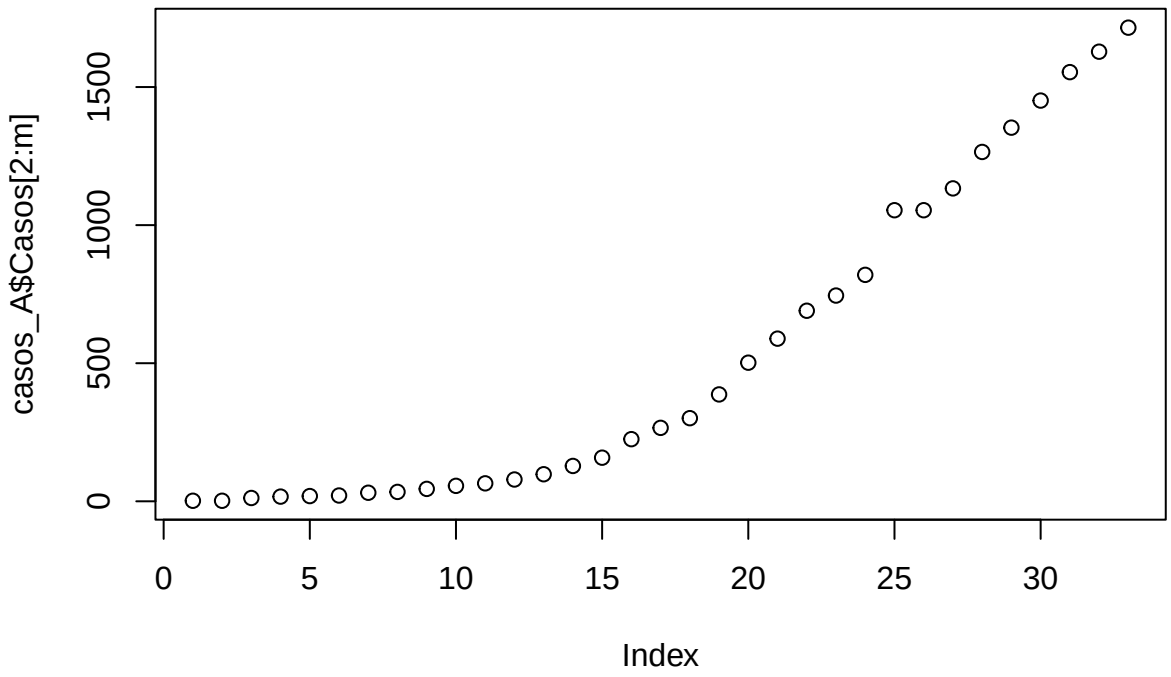
```r
sd(F,na.rm = TRUE)
```

```
## [1] 0.8554107
```

```r
var(F,na.rm = TRUE)
```

```
## [1] 0.7317275
```

```r
#Grafico de casos
plot(casos_A$Casos[2:m]);(casos_A$Casos[1:m-1])
```




```
##  [1]    1    2    2   12   17   19   21   31   34   45   56   65   79   98  128
## [16]  158  225  266  301  387  502  589  690  745  820 1054 1054 1133 1265 1353
## [31] 1451 1554 1628
```

3.2. Accediendo a los datos actualizados del Covid-19

```r
library(readr)
location <- getwd()
setwd(location)
casos_B <- read.csv("time_series_covid19_confirmed_global.csv",header = TRUE, sep = ",", dec=".")
#en nustro caso la base de dato esta muy grande, como una matriz y no un vector,
#asi no se puede utilizar la funcion which.max
# definimos aca los dos variables que necessitamos por la determincaion del modelo
epidemia=4520
fecha <- as.Date(c("01/01/22"),  format = "%m/%d/%y")
head(fecha)
```

```
## [1] "2022-01-01"
```

```r
#determinacion en que fecha se contagiaran 40 millones de personas
library(lubridate)
```

```
##
## Attachement du package : 'lubridate'
```

```
## Les objets suivants sont masqués depuis 'package:base':
##
##     date, intersect, setdiff, union
```

```r
F=1.62
while (epidemia<40000000) {
fecha=fecha+ days(1)
epidemia=epidemia*F
}
"fecha se contagiaran 40 millones de personas:"
```

```
## [1] "fecha se contagiaran 40 millones de personas:"
```

```r
head(fecha)
```

```
## [1] "2022-01-20"
```