Grupo LDA - Programación en R Cran

Aruani, Juan* De Mezzo, Facundo† López, Emilia‡ López, Bernardo§

2022-06-12

Introducción

En este informe nos adentramos en la programación en R. Comparamos diferentes códigos que utilizan las mismas funciones para compararlos y determinar cuales son más efectivos.

Desarrollo

Generar un vector secuencia

Utilizaremos una secuencia generada con la función for y otra con la función seq() de R. La secuencia generada da lugar a vectores de gran tamaño. Mediremos y compararemos el tiempo utilizado por cada método.

Secuencia generada con for

```
t1_inicial <- Sys.time()
A1 <- 0
for (i in 1:50000) {A1[i]<-(i*2)}
head(A1)
```

```
## [1]  2  4  6  8 10 12
```

```
tail(A1)
```

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

```
t1_final <- Sys.time()
```

Secuencia generada con R

```
t2_inicial <- Sys.time()
A2 <- seq(1,1000000,2)
head(A2)
```

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

```
tail(A2)
```

```
## [1] 999989 999991 999993 999995 999997 999999
```

```
t2_final <- Sys.time()
```

Ahora compararemos el tiempo utilizado por cada método para llevar a cabo la comparativa

*juan.aruani.99@gmail.com
†em.cn.demezzo.facundo@gmail.com
‡emilia.lop49@gmail.com
§bernilopezmorel@gmail.com

```
t_total_for <- (t1_final - t1_inicial)
message("El tiempo utilizado por el método for es: ",t_total_for)
```

```
## El tiempo utilizado por el método for es: 0.202262878417969
```

```
t_total_seq <- (t2_final - t2_inicial)
message("El tiempo utilizado por el método seq es: ",t_total_seq)
```

```
## El tiempo utilizado por el método seq es: 0.0375118255615234
```

Conclusión: Se puede observar claramente la diferencia de velocidad entre ambos códigos. La función "seq" de R es mucho más veloz que el comando "for." Para medir la diferencia de tiempo se utilizó la función "Sys.time"

Serie de Fibonacci

```
A<-0
B <- 1
F[1]<-A
F[2]<-B
for(i in 3:100) {
F[i]<-(F[i-1]+F[i-2])
}
head(F)
```

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

Ahora se averigua la cantidad de iteraciones necesarias para generar un número en la serie que sea mayor a 1.000.000 Utilizamos el comando "remove(F)" para evitar que resultados anteriores o de las diferentes pruebas afecten el resultado, ya que sólo asignamos valores a F, y cualquier valor que se encuentre guardado después del último afectado, simplemente no será modificado, y evitará que podamos trabajar revisando el valor final de la secuencia generada.

```
remove(F)
A<-0
B <- 1
C <- 0
it <- 30
F[1]<-A
F[2]<-B
for(i in 3:(2+it)) {
F[i]<-(F[i-1]+F[i-2])
}
C <- length(F)
message("Para ",it," iteraciones, el penúltimo valor es: \n",F[C-1],"\n","y el último es: \n"
        ,F[C],"\n", "ubicado en la posición ",C," de la secuencia")
```

```
## Para 30 iteraciones, el penúltimo valor es:
## 832040
## y el último es:
## 1346269
## ubicado en la posición 32 de la secuencia
```

Con este resultado se puede concluír que se necesitan 30 iteraciones para superar el número 1.000.000 La razón por la que el número de iteraciones "it" es diferente a la posición del número buscado es que los primeros dos número (0 y 1) son asignados manualmente y no se obtienen en las iteraciones.

Ordenación de un vector por método burbuja

En esta sección se procede a ordenar un vector mediante el método de burbuja, que funciona mediante la revisión de los elementos y la comparación de los mismos. Luego se procede a compararlo con el método "sort" incluído en R usando la librería "microbenchmark"

```
# Tomo una muestra de 10 números ente 1 y 100
xa<-sample(1:100,10)
# Creo una funci?n para ordenar
burbuja <- function(x){
n<-length(x)
for(j in 1:(n-1)){
for(i in 1:(n-j)){
if(x[i]>x[i+1]){
temp<-x[i]
x[i]<-x[i+1]
x[i+1]<-temp
}
}
}
return(x)
}
res1<-burbuja(xa)
#Muestra obtenida
res1
```

```
##  [1]  2  9 19 32 45 88 91 94 95 98
```

```
res2 <- sort(xa)
res2
```

```
##  [1]  2  9 19 32 45 88 91 94 95 98
```

Se muestra el funcionamiento del método aplicandolo a un vector de pocos elementos. A continuación se compara la performance para una muestra mucho más grande (tamaño: 20000)

```
# Tomo una muestra de 20 000 números ente 1 y 1 000 000
xa<-sample(1:1000000,20000)
# Creo una funci?n para ordenar
burbuja <- function(x){
n<-length(x)
for(j in 1:(n-1)){
for(i in 1:(n-j)){
if(x[i]>x[i+1]){
temp<-x[i]
x[i]<-x[i+1]
x[i+1]<-temp
}
}
}
return(x)
}
burbuja<-burbuja(xa)
sortR <- sort(xa)

library(microbenchmark)
mbmk <- microbenchmark(burbuja,sortR)
```

```
mbmk
```

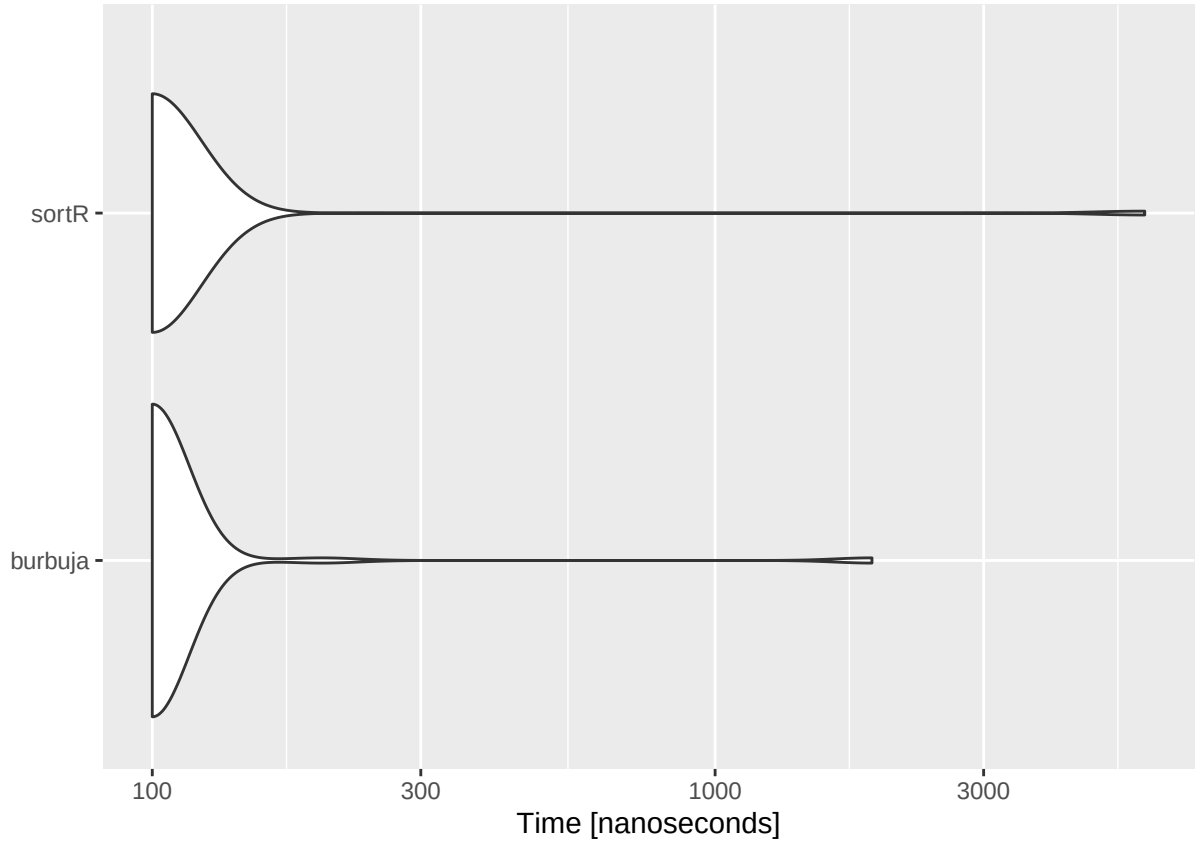
```
## Unit: nanoseconds
##     expr min lq mean median  uq  max neval
##  burbuja   0  0   81    100 100 1900   100
##    sortR   0  0  120    100 100 5800   100
```

```
library(ggplot2)
autoplot(mbmk)
```

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

```
## Warning: Transformation introduced infinite values in continuous y-axis
```

```
## Warning: Removed 75 rows containing non-finite values (stat_ydensity).
```



Se puede observar que el tiempo utilizado por el método burbuja es menor que el del método sort, pero que requiere una utilización de recursos inicial mucho más intensiva.

Progresión geométrica del Covid 19

Ahora se procede a ver como importar datos de un csv, pero no solo desde el disco de la computadora, sino directamente desde internet

Leer datos de un csv desde el disco

Vamso a file > import dataset > readr se instalan paquetes la 1ra vez Se abre un cuadro de dialogo donde nos permite elegir la direccion del archivo donde esta la info y genera el codigo que necesitamos para generar

el dataset[1]

Recuperación de casos de Argentina

```
library(readr)
Arg <- read_delim("~/Facultad/Técnicas y Herramientas Modernas I/R/casos.csv",
    delim = ";", escape_double = FALSE, col_types = cols(`Covid Argentina` = col_date(format = "%m/%d/%
        ...2 = col_integer(), Fecha = col_date(format = "%m/%d/%Y"),
        Casos = col_integer()), locale = locale(),
    trim_ws = TRUE, skip = 1)
```

```
## Warning: The following named parsers don't match the column names: Covid
## Argentina, ...2
```

```
## Warning: One or more parsing issues, see `problems()` for details
```

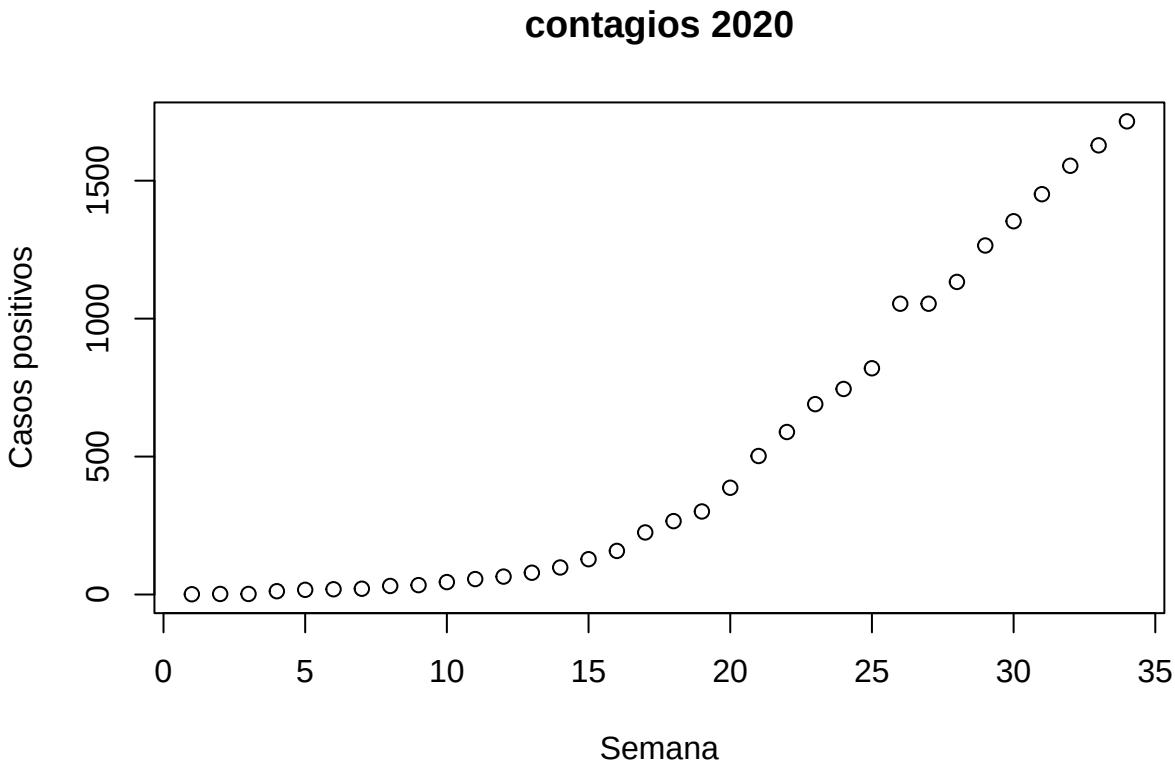
```
Arg
```

```
## # A tibble: 34 x 3
##    Fecha      Casos `E_P+1`
##    <date>     <int> <lgl>
##  1 2020-05-03     1 NA
##  2 2020-06-03     2 NA
##  3 2020-07-03     2 NA
##  4 2020-08-03    12 NA
##  5 2020-09-03    17 NA
##  6 2020-10-03    19 NA
##  7 2020-11-03    21 NA
##  8 2020-12-03    31 NA
##  9 NA            34 NA
## 10 NA            45 NA
## # ... with 24 more rows
```

Gráfico con la cantidad de contagios

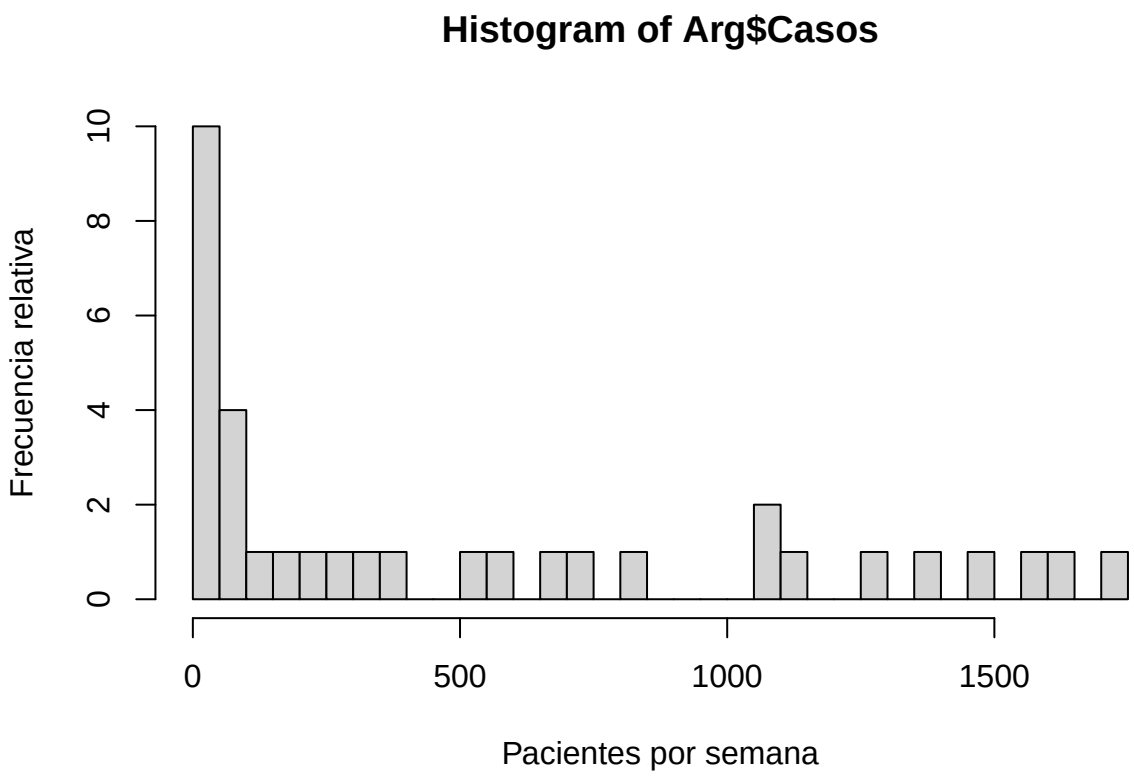
```
#casos$...2
plot(Arg$Casos,main = "contagios 2020",xlab = "Semana",ylab = "Casos positivos")
```

[1]dataset: conjunto de datos que proviene de alguna fuente de la vida real en el que tambien se explica de dónde y cómo se obtuvieron los datos. Suelen estar en repositorios ej: los dataset de estudios de mercado pueden encontrarse en https://www.kaggle.com/datasets El que importamos es una fila de la tabla del hospital Hopkins En el caso del hospital el maximo de excel se alcanzo en FFFF (hexadecimal)



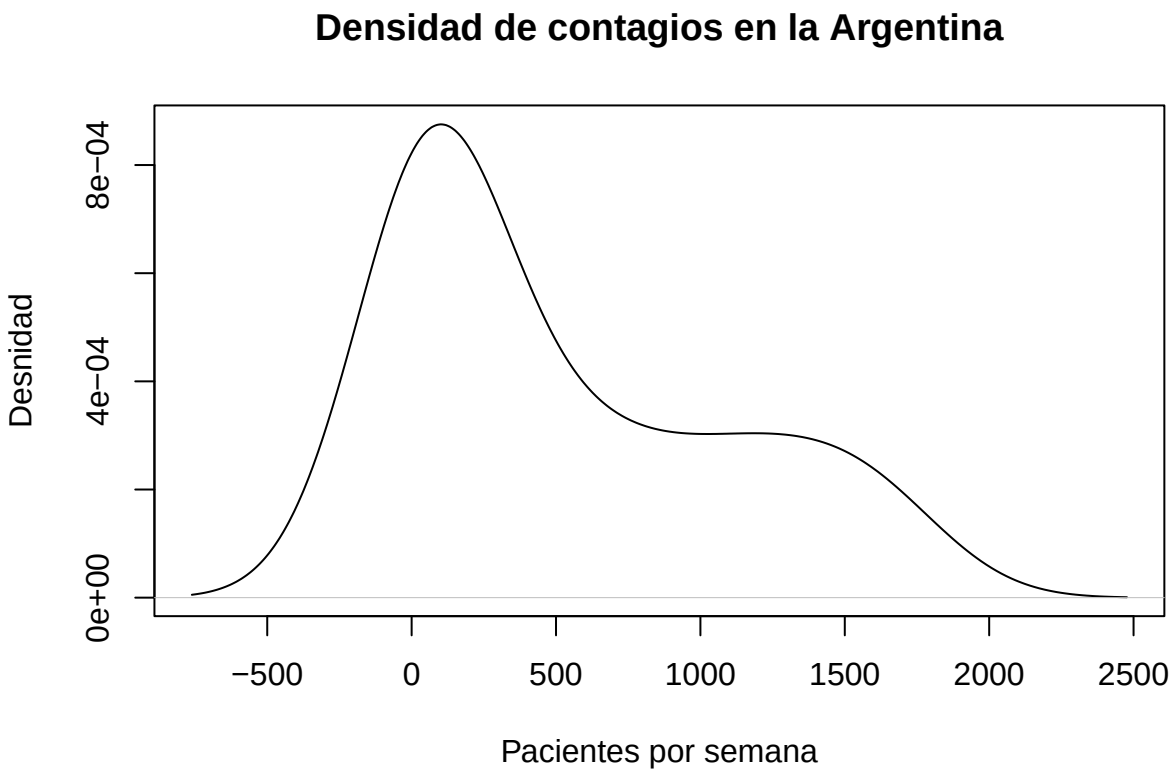
Histograma

```
hist(Arg$Casos,breaks = 50,xlab = "Pacientes por semana",ylab = "Frecuencia relativa")
```



Densidad de contagios

```
plot(density(na.omit(Arg$Casos)),main = "Densidad de contagios en la Argentina", ylab = "Desnidad",xlab
```



Estadística de casos

```
summary(Arg$Casos)
```

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

Cálculo del factor de contagios

Se puede calcular el factor de contagios F mediante el cociente entre los infectados de hoy y los de ayer

$$F = I_{n+1}/I_n$$

De modo que se calcula el factor para cada par de fechas y se entrega el de hoy

```
m <- length(Arg$Casos)
F <- (Arg$Casos[2:m])/(Arg$Casos[1:m-1])
message("El factor de contagios de la última fecha es: ",F[m-1])
```

```
## El factor de contagios de la última fecha es: 1.0534398034398
```

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
```