

Trabajo de programacion en R

Daniel Andolfi^{1,1,*}, Martina Alvarez¹, Felicitas Bascolo^{1,2}, Lisandro Herrera^{1,2}

^aMendoza Argentina

Abstract

En este informe se resolveran los ejercicios planteados por la catedra “Tecnicas y Herramientas Modernas” para la programacion en R

Keywords: Modulo 2, Programacion en R

1. Metodo sys.time

Este metodo para medir el tiempo de ejecucion de un codigo viene ya determinado con la base de R

```
duermete_un_minuto <- function() { Sys.sleep(5) }
start_time <- Sys.time()
duermete_un_minuto()
end_time <- Sys.time()
end_time - start_time

## Time difference of 5.076401 secs
```

2. Metodo biblioteca tictoc

Descargamos e instalamos la biblioteca Tictoc para medir el tiempo de ejecucion del codigo

```
library(tictoc)
tic("sleeping")
A<-20
print("dormire una siestita...")

## [1] "dormire una siestita..."

Sys.sleep(2)
print("...suena el despertador")

## [1] "...suena el despertador"
```

*Corresponding author

Email addresses: daniandolfi@gmail.com (Daniel Andolfi), bob@example.com (Martina Alvarez), cat@example.com (Felicitas Bascolo), derek@example.com (Lisandro Herrera)

¹This is the first author footnote.

²Another author footnote.

```

toc()

## sleeping: 2.1 sec elapsed

```

3. Metodo biblioteca rbenchmark

Descargamos e instalamos la biblioteca rbenchmark para medir el tiempo de ejecucion del codigo

```

library(rbenchmark)
# lm crea una regresion lineal
benchmark("lm" = {
X <- matrix(rnorm(1000), 100, 10)
y <- X %*% sample(1:10, 10) + rnorm(100)
b <- lm(y ~ X + 0)$coef
},
"pseudoinverse" = {
X <- matrix(rnorm(1000), 100, 10)
y <- X %*% sample(1:10, 10) + rnorm(100)
b <- solve(t(X) %*% X) %*% t(X) %*% y
},
"linear system" = {
X <- matrix(rnorm(1000), 100, 10)
y <- X %*% sample(1:10, 10) + rnorm(100)
b <- solve(t(X) %*% X, t(X) %*% y)
},
replications = 1000,
columns = c("test", "replications", "elapsed",
"relative", "user.self", "sys.self"))

##          test replications elapsed relative user.self sys.self
## 3 linear system      1000    0.38    1.000     0.35    0.00
## 1 lm                 1000    2.46    6.474     2.28    0.08
## 2 pseudoinverse      1000    0.47    1.237     0.40    0.05

```

4. Metodo biblioteca microbenchmark

Descargamos e instalamos la biblioteca microbenchmark que nos va a permitir comparar tiempos de ejecucion de múltiples fragmentos de código R. Con este metodo se puede ver de forma grafica el uso de recursos.

```

library(microbenchmark)
set.seed(2017)
n <- 10000
p <- 100
X <- matrix(rnorm(n*p), n, p)
y <- X %*% rnorm(p) + rnorm(100)
check_for_equal_coefs <- function(values) {
tol <- 1e-12
max_error <- max(c(abs(values[[1]] - values[[2]]),
abs(values[[2]] - values[[3]]),

```

```

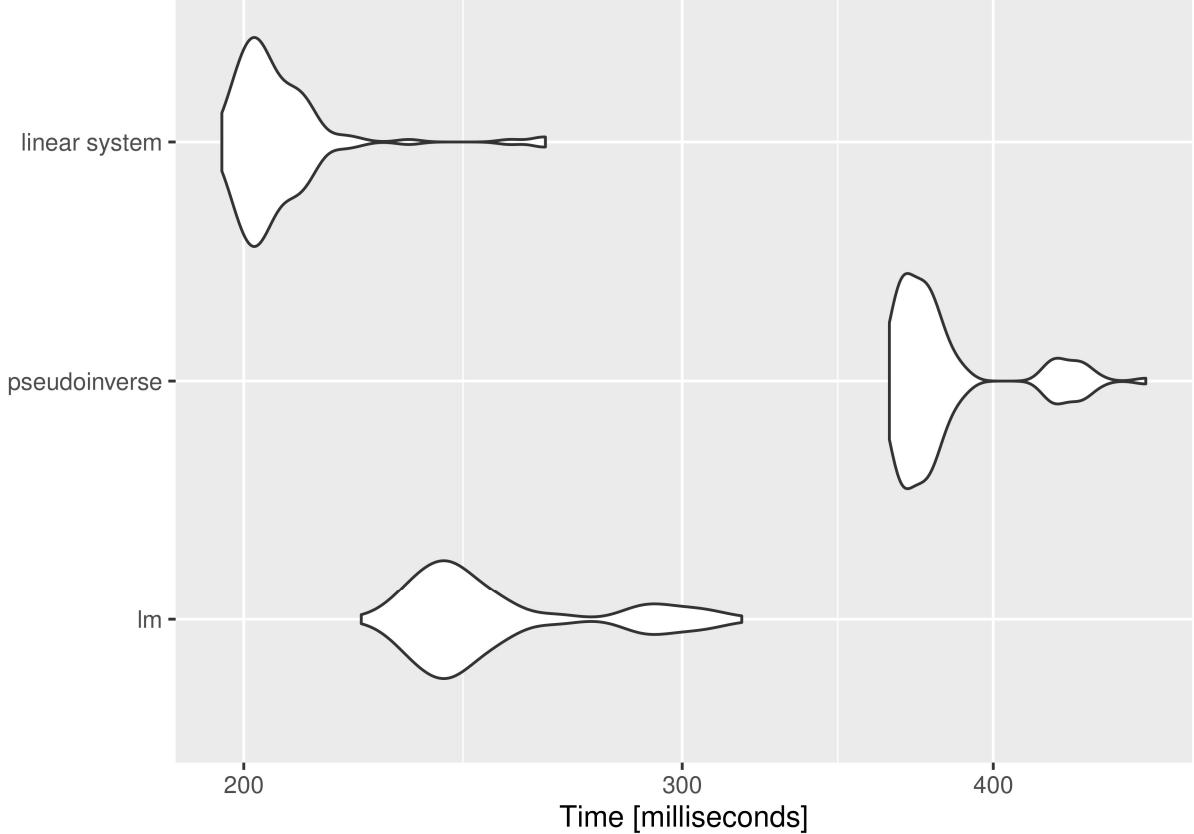
abs(values[[1]] - values[[3]])))
max_error < tol
}
mbm <- microbenchmark("lm" = { b <- lm(y ~ X + 0)$coef },
"pseudoinverse" = {
b <- solve(t(X) %*% X) %*% t(X) %*% y
},
"linear system" = {
b <- solve(t(X) %*% X, t(X) %*% y)
},
check = check_for_equal_coefs)
mbm

## Unit: milliseconds
##          expr      min       lq     mean   median      uq     max neval
##        lm 222.9604 237.5844 253.4529 243.6730 255.8056 316.9943    100
## pseudoinverse 363.3610 368.2667 383.0219 375.4654 382.1097 460.5972    100
## linear system 195.9873 200.6229 206.6459 203.7107 209.1597 264.3538    100

library(ggplot2)
autoplot(mbm)

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

```



5. Trabajo de evaluacion del modulo

5.1. Comparar la generacion de un vector secuencia

5.1.1. Secuencia generada con for

Vamos a generar un vector secuencia usando “for” y vamos a medir el tiempo de ejecucion a traves del metodo Sys.time

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

## [1] 2 4 6 8 10 12

tail(A)

## [1] 99990 99992 99994 99996 99998 100000

end_time <- Sys.time()
end_time - start_time

## Time difference of 0.06995392 secs
```

5.1.2. Secuencia generada con R

Vamos a generar un vector secuencia usando el comando “seq” que viene de base con el RStudio y vamos a medir el tiempo de ejecucion a traves del metodo Sys.time

```
start_time <- Sys.time()
A <- seq(1,1000000, 2)
head (A)

## [1] 1 3 5 7 9 11

tail(A)

## [1] 999989 999991 999993 999995 999997 999999

end_time <- Sys.time()
end_time - start_time

## Time difference of 0.04015899 secs
```

A traves del metodo “Systime” podemos ver que para generar un vector secuencia que contiene numeros del 1 al 100.000 en intervalos de 2 va a requerir menor tiempo de ejecucion el comando para generar secuencias “seq” que ya viene de base con el RStudio

5.2. Implementacion de una serie Fibonacci

Vamos a generar una serie de Fibonacci hasta llegar a 1.000.000 y vamos a medir la cantidad de iteraciones que nos tomo

```

start_time <- Sys.time()
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

end_time <- Sys.time()
end_time - start_time

## Time difference of 0.02047992 secs

start_time <- Sys.time()
f1<-0
f2<-1
N<-0
vec<- c(f1,f2)
f3<-0

while (f3 <= 1000000) {
  N<-N+1
  f3<-f1++f2
  vec<- c(vec,f3)
  f1<-f2
  f2<-f3
  c<-a+b
  i<-i+1
}
N

## [1] 30

vec

## [1] 0 1 1 2 3 5 8 13 21
## [10] 34 55 89 144 233 377 610 987 1597
## [19] 2584 4181 6765 10946 17711 28657 46368 75025 121393
## [28] 196418 317811 514229 832040 1346269

end_time <- Sys.time()
end_time - start_time

## Time difference of 0.05200195 secs

```

Para generar un numero mayor a 1.000.000 en la serie se necesitan 30 iteraciones

5.3. Ordenacion de un vector por metodo burbuja y el comando sort

Vamos a ordenar un vector que contiene 200 numeros aleatorios que van del 1 al 1000. Para esto vamos a usar dos metodos distintos, el metodo burbuja y el metodo sort que viene de base con el RStudio. Vamos a medir el tiempo de ejecucion del codigo y la cantidad de recurso computacional utilizado a traves del metodo Microbenchmark

```
library(microbenchmark)

x<-sample(1:1000,200)

mbm <- microbenchmark(
  "burbuja"={

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)
}
res<-burbuja(x)

},

"sort" = {
  sort(x)
}

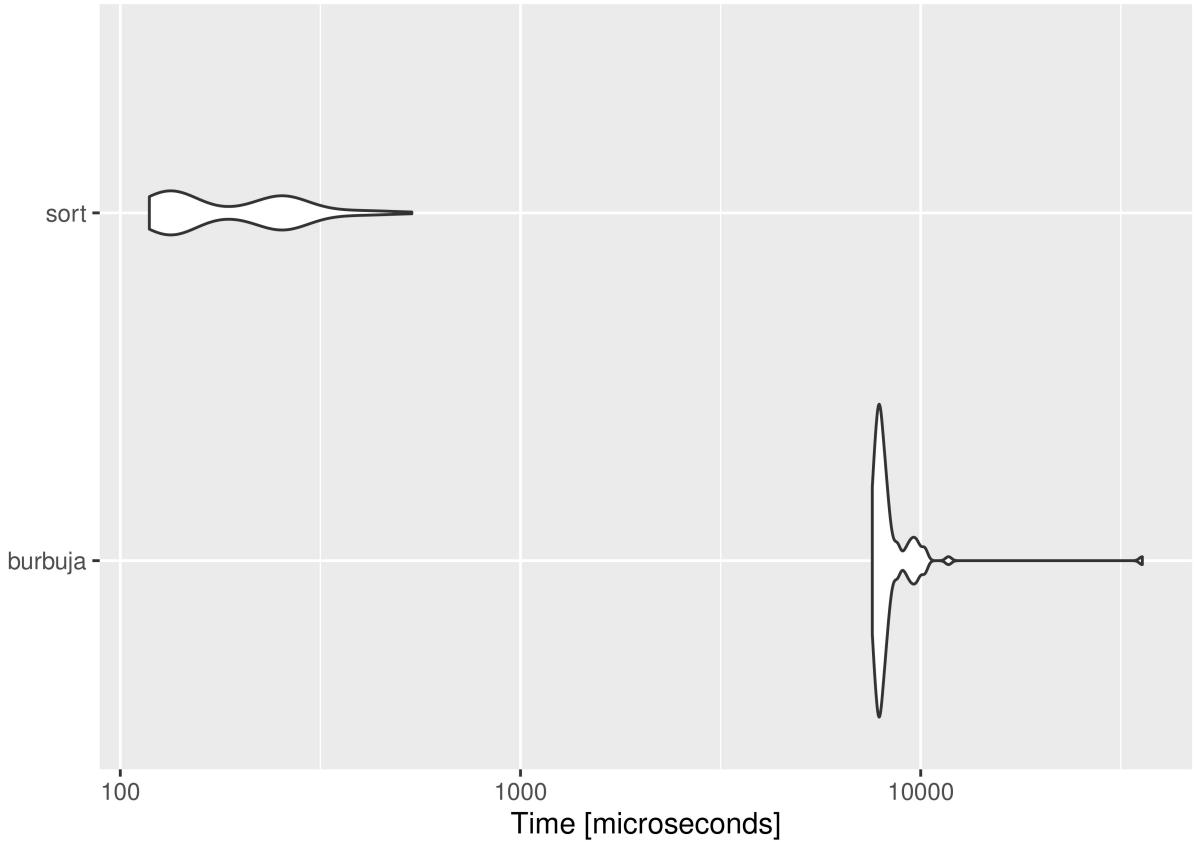
)

mbm

## Unit: microseconds
##      expr    min     lq     mean   median     uq    max neval
##  burbuja 7569.8 7815.65 8540.294 8000.35 8330.9 35794.8   100
##      sort  118.2   134.80  201.773  165.80  255.6   535.2   100

library(ggplot2)
autoplot(mbm)

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



Comparando los dos metodos a traves de microbenchmark podemos notar que el metodo de la burbuja requiere de mucho mas recursos y tiempo de procesamiento que el metodo sort

6. Modelado matemático de una epidemia

Utilizando el numero de casos semanales en Argentina para Junio del 2022 y un factor de contagio constante $F=1.62$, vamos a calcular cuantas semanas tomaria en llegar a los 40.000.000 de contagios

```
# Numeros de casos semanales en Argentina
f1<-25680

N<-0
f3 <- 1.62
f2 <- 0
vec <- c(f1)

while (f1 <= 40000000) {
  f2 <- f1*f3
  f1 <- f2
  N <- N+1
  vec <- c(vec,f2)
}

}
```

```
N
```

```
## [1] 16  
vec  
  
## [1] 25680.00 41601.60 67394.59 109179.24 176870.37 286529.99  
## [7] 464178.59 751969.32 1218190.30 1973468.28 3197018.61 5179170.15  
## [13] 8390255.65 13592214.15 22019386.93 35671406.82 57787679.05
```

Con el numero de casos actuales semanales en Argentina y con el factor de contagio $F=1.62$, tardariamos 16 semanas en llegar a los 40 millones de contagiados.

Este dato se aleja de la realidad y es solo para mostrar como se utilizan las ecuaciones para el modelado de una epidemia. En la realidad el factor de contagio no se mantiene constante y depende de diversos factores como cantidad de contagios y grados de restriccion gubernamental.

7. Importar datos de la red o de excel

Para importar datos de la red o de un excel utilizamos la funcion Import Dataset y seleccionamos From text (readr). Alli vamos a cargar el archivo .csv que tenemos descargado en nuestra computadora y vamos a seleccionar cual va a ser el delimitador.

```
library(readr)  
casos <- read_delim("D:/Descargas/casos.csv",  
  delim = ";", escape_double = FALSE, col_types = cols(...2 = col_number()),  
  trim_ws = TRUE)
```

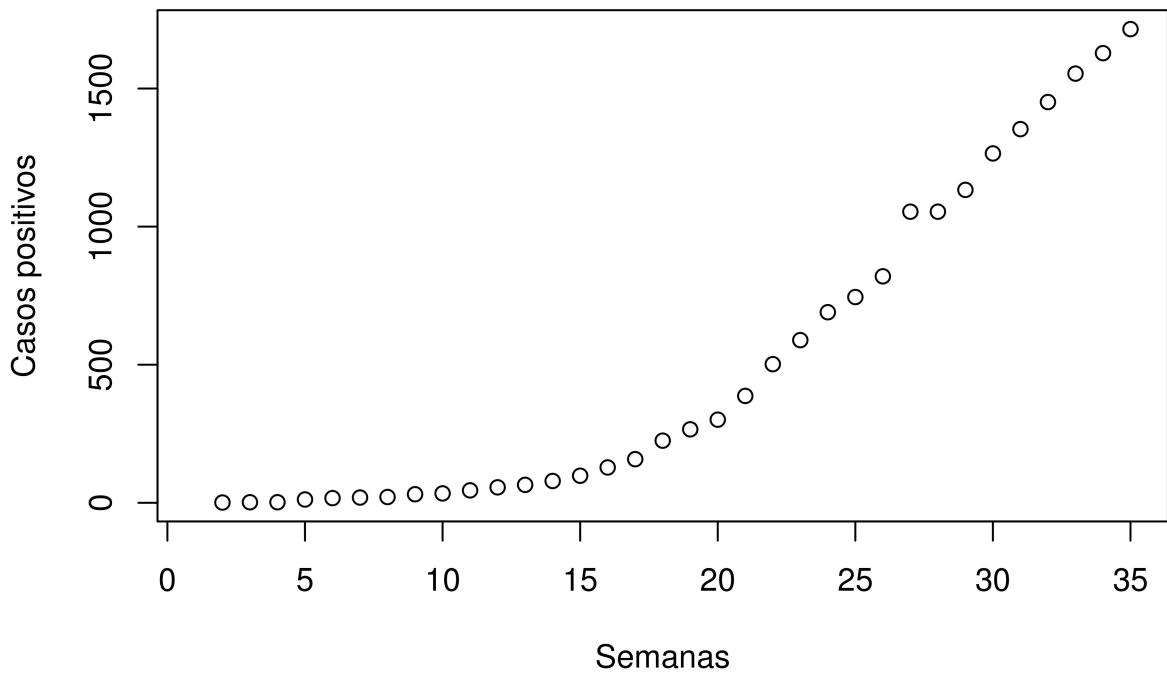
```
## New names:  
## * ` ` -> `...2`  
## * ` ` -> `...3`  
  
## Warning: One or more parsing issues, see 'problems()' for details
```

En este caso importamos el numero de casos covid en la Argentina para poder analizarlo a traves de los distintos graficos que nos permite realizar el RStudio.

Primero realizamos un grafico tipo Plot para ver como fue el incremento de casos en funcion del numero de semanas.

```
plot(casos$...2, main="Contagio 2020", ylab="Casos positivos", xlab="Semanas")
```

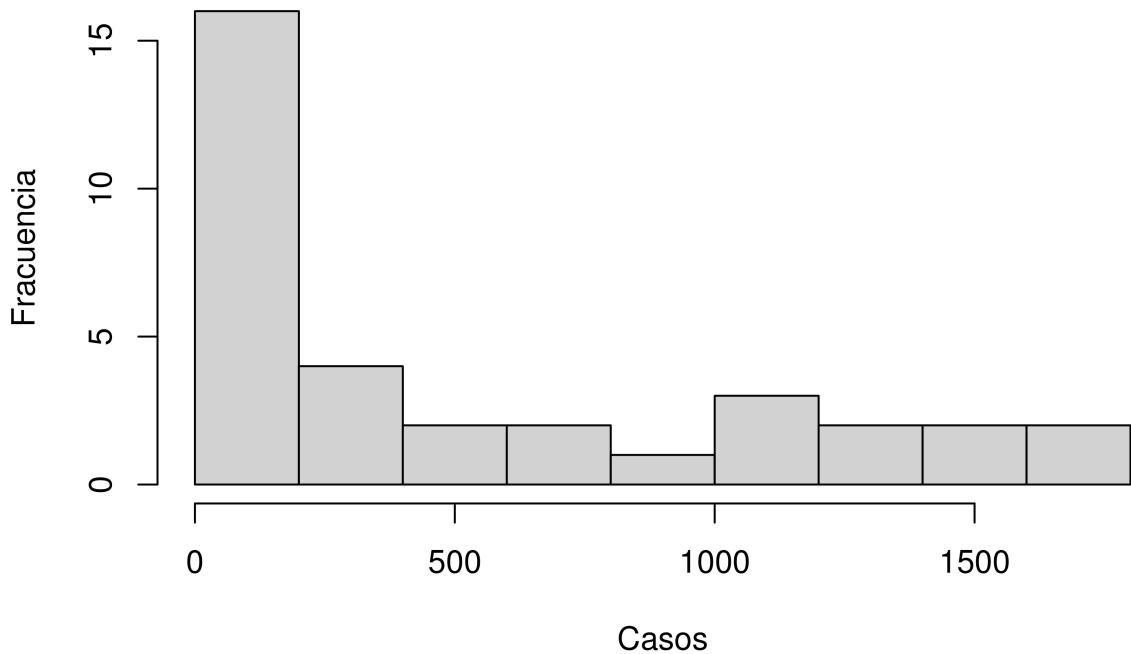
Contagio 2020



Luego realizamos un histograma para ver cual es la frecuencia del numero de contagiados de los datos obtenidos

```
hist(as.numeric(casos$...2), main="Histograma de contagios 2020", ylab="Fracuencia", xlab="Casos")
```

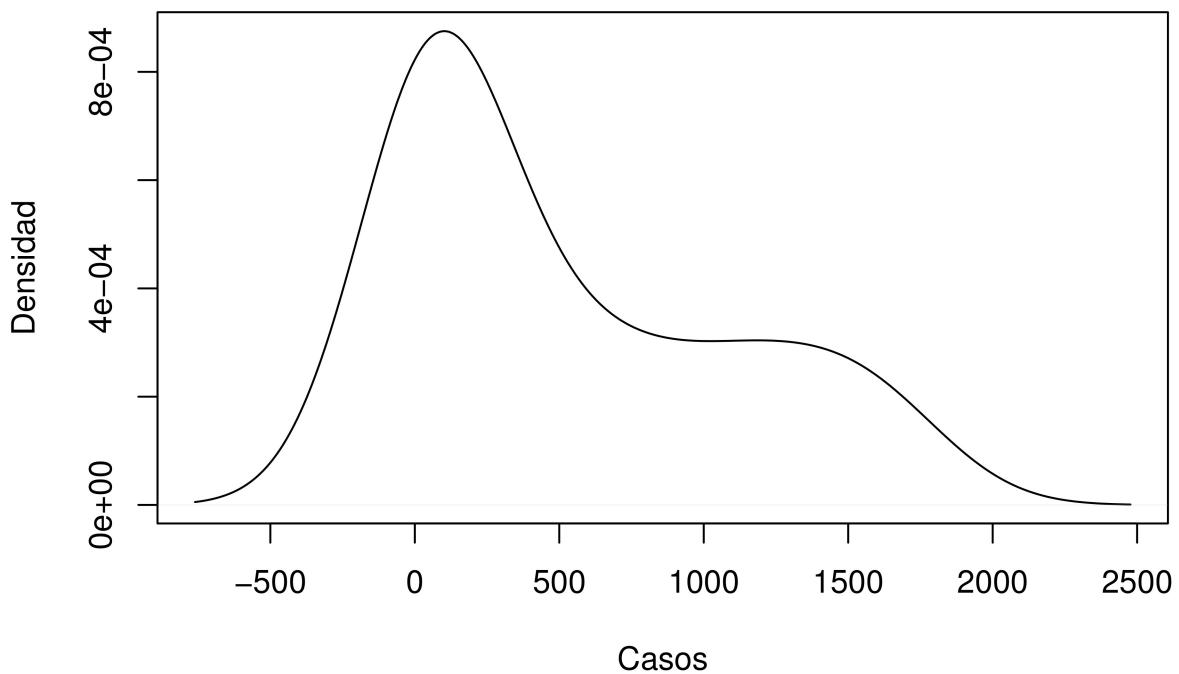
Histograma de contagios 2020



Tambien podemos realizar un grafico de densidad de los casos Covid en Argentina. Este grafico me va a permitir reconocer las distintas cepas de covid que circulan en el pais

```
plot(density(na.omit(casos$...2)), main="Grafico de densidad de contagios", ylab="Densidad", xlab="Casos")
```

Grafico de densidad de contagios



En el grafico de densidad podemos observar que se producen 2 picos. Estos dos picos corresponden a la incidencia de la cepa manaos en primer lugar y de la cepa delta en segundo lugar. Con la aparicion de cada cepa, el virus genera un pico en la tasa de contagios. Esto se debe particularmente a las mutuaciones originadas en cada tipo de cepa.