TyHM - Programacion en R 2022

Santiago Luis Blazquez[1], Agostina Lencioni[1], Enzo Viallet[1], Agostina Sosa[1]

[a]*Facultad de ingenieria Universidad Nacional de Cuyo*

Abstract

This is the abstract.
It consists of two paragraphs.

Keywords: keyword1, keyword2

1. Generar un vector secuencia

1.1. for

```
C <- seq(1:200)
for (i in 2:100) {
  C[i] <- (C[i-1]+3)
  }
head (C)
```

```
## [1]  1  4  7 10 13 16
```

```
A <- vector()
for(i in 0:40000){
  A[i+1] <- (i*2)
}
tail(A)
```

```
## [1] 79990 79992 79994 79996 79998 80000
```

```
head(A)
```

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

1.2. seq

Con un paso de 4

```
B <- vector()
B<- seq(1,100000,4)
tail(B)
```

*Corresponding author
Email addresses: santi98b@gmail.com (Santiago Luis Blazquez), agostinalencioni04@gmail.com (Agostina Lencioni), enzo.viallet@enise.fr (Enzo Viallet), agostina.sdo@gmail.com (Agostina Sosa)

```
## [1] 99977 99981 99985 99989 99993 99997
```

```
head(B)
```

```
## [1]  1  5  9 13 17 21
```

1.3. Performance

Uso del algoritmo *Biblioteca tictoc*

" Esto de usar una biblioteca es llamar u cargar una procedimientos que generará comando nuevos en R. Como ya fue comentado, cargar una biblioteca implica ejecutar el comando install.packages() o usar en r-studio el menú de Herramientas y Luego Instalar paquetes. Las funciones tic y toc son de la misma biblioteca de Octave/Matlab y se usan de la misma manera para la evaluación comparativa que el tiempo de sistema recién demostrado. Sin embargo, tictoc agrega mucha más comodidad al usuario y armonía al conjunto"

```
library(tictoc)
tic("inicio")
# Tomo una muestra de 10 números ente 1 y 1000
x<-sample(1:1000,900)
# Creo una funcion 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)
}
res<-burbuja(x)
res
```

```
##   [1]   1   2   3   4   5   6   7   9  11  12  13  15  16  17  18
##  [16]  19  20  21  22  23  25  26  27  28  29  30  31  32  33  34
##  [31]  35  36  37  38  39  40  41  43  44  45  46  47  48  49  50
##  [46]  51  52  54  55  56  57  58  59  60  61  62  63  64  65  66
##  [61]  67  68  69  70  71  72  73  74  75  77  78  79  81  82  83
##  [76]  84  85  86  87  88  89  90  91  93  94  95  96  97  99 101
##  [91] 102 103 104 105 106 107 109 110 111 112 113 114 116 117 118
## [106] 119 120 121 122 124 125 126 127 128 129 131 132 133 134 135
## [121] 136 137 139 140 141 142 143 144 145 146 147 148 149 150 152
## [136] 153 154 155 156 158 159 160 161 162 164 165 166 167 168 169
## [151] 170 171 172 173 174 175 176 177 178 180 181 182 183 184 185
## [166] 187 188 189 190 191 192 193 194 195 198 200 201 204 205 206
## [181] 207 208 209 211 213 214 215 216 217 218 219 220 222 223 224
## [196] 225 226 227 228 230 231 232 233 234 235 236 237 239 240 241
## [211] 242 244 245 246 248 249 250 252 253 254 255 256 257 259 260
```

```
## [226]  261  262  263  264  265  266  267  268  269  270  271  272  273  274  275
## [241]  276  277  278  279  280  281  282  283  284  285  286  287  288  289  290
## [256]  291  292  293  294  295  296  297  299  301  302  303  304  306  307  308
## [271]  309  310  311  312  313  314  316  317  318  320  321  322  323  325  326
## [286]  327  328  329  330  331  332  333  334  335  337  338  339  340  341  342
## [301]  343  344  345  346  349  351  352  353  354  356  357  358  359  360  361
## [316]  362  363  364  365  366  367  368  369  370  371  372  373  374  375  376
## [331]  377  378  379  380  381  382  383  384  385  386  387  388  389  390  391
## [346]  392  393  394  395  396  397  399  400  401  402  403  404  405  406  407
## [361]  408  409  410  411  412  413  414  415  416  417  418  419  420  421  422
## [376]  423  424  425  426  427  428  429  430  431  432  434  435  436  437  438
## [391]  440  441  442  444  445  446  447  448  449  450  451  452  453  454  455
## [406]  456  457  458  459  460  461  462  463  465  466  467  468  469  470  471
## [421]  472  473  474  475  476  477  478  479  480  482  483  484  486  487  488
## [436]  490  492  493  494  495  496  497  498  499  500  501  503  504  505  506
## [451]  507  508  509  510  511  512  513  514  515  516  517  518  519  520  521
## [466]  523  524  526  527  528  529  530  531  532  533  534  535  536  537  538
## [481]  539  540  541  542  543  544  546  547  548  549  550  551  553  554  555
## [496]  556  557  558  559  560  561  562  563  564  565  566  567  568  569  570
## [511]  571  572  573  574  576  578  579  580  582  583  584  585  586  587  589
## [526]  590  591  592  593  594  595  596  597  598  599  600  601  602  603  604
## [541]  605  606  607  609  610  611  612  613  614  615  616  617  618  619  620
## [556]  621  622  623  625  626  627  628  629  630  631  632  633  634  635  637
## [571]  638  639  641  642  643  644  645  646  647  648  649  650  651  652  653
## [586]  654  655  656  657  658  659  660  661  662  663  664  665  666  667  668
## [601]  669  670  671  672  673  674  675  676  677  678  679  680  681  682  683
## [616]  684  685  686  687  688  689  690  692  693  694  695  697  698  699  700
## [631]  701  702  703  704  705  706  707  708  709  710  711  712  713  714  715
## [646]  716  717  718  719  721  722  723  724  725  726  727  729  730  733  734
## [661]  735  736  737  738  739  740  741  742  743  744  745  746  747  748  750
## [676]  751  752  753  754  755  756  758  759  760  761  762  763  764  765  766
## [691]  767  768  769  770  771  772  773  775  776  777  778  779  781  782  783
## [706]  784  785  786  787  788  789  790  791  792  793  794  795  796  797  798
## [721]  799  800  801  802  803  804  805  808  809  810  811  812  813  814  815
## [736]  817  819  820  822  823  825  826  827  829  830  831  832  833  834  835
## [751]  836  837  838  839  840  841  842  843  844  846  847  848  849  850  851
## [766]  853  854  855  856  857  858  859  860  862  863  864  865  866  867  869
## [781]  870  871  872  873  874  875  876  877  878  879  881  882  884  885  886
## [796]  888  889  891  892  893  894  895  897  899  900  901  902  903  904  905
## [811]  906  907  908  909  910  911  912  913  914  915  916  917  918  919  920
## [826]  921  922  923  924  925  926  927  928  929  930  931  932  933  934  935
## [841]  936  937  938  939  940  941  942  943  944  945  947  948  949  950  951
## [856]  952  953  954  955  956  957  958  959  960  961  962  963  964  965  966
## [871]  967  968  969  971  972  973  974  976  977  978  979  980  981  982  983
## [886]  984  985  987  988  989  990  992  993  994  995  996  997  998  999 1000
```

```
toc()
```

```
## inicio: 0.08 sec elapsed
```

2. Serie de Fibonacci

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

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

2.1. Performance

uso del algoritmo *Biblioteca rbenchmark.*

"La documentación de la función benchmark del paquete rbenchmark R lo describe como "un simple contenedor alrededor de system.time". Sin embargo, agrega mucha conveniencia en comparación con las llamadas simples a system.time. Por ejemplo, requiere solo una llamada de referencia para cronometrar múltiples repeticiones de múltiples expresiones. Además, los resultados devueltos se organizan convenientemente en un marco de datos"

```
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    0.11        1      0.11        0         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 50 numeros elegidos aleatoriamente entre el 1 y el 1000:

```
x <- sample(1:1000,50)

head(x)
```

```
## [1]  55 457 374 630 460 377
```

```
tail(x)
```

```
## [1] 841 978 417 490  98 304
```

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

muestraordenada <- bubble(x)

head(muestraordenada)
```

```
## [1] 996 981 978 977 928 923
```

```
tail(muestraordenada)
```

```
## [1] 93 91 63 58 55 41
```

3.1. Performance

Uso del algoritmo *Biblioteca Microbenchmark.*

"Al igual que el punto de referencia del paquete rbenchmark, la función microbenchmark se puede usar para comparar tiempos de ejecución de múltiples fragmentos de código R. Pero ofrece una gran comodidad y funcionalidad adicional. Es más "beta" (inestable), pero como todo lo que hoy es nuevo poco a poco se hará más estable y no complicará tanto las cosas para el usuario final.Una cosa interesante es que se puede ver la salida gráfica del uso de recursos. Ver líneas finales del código. Me parece que una característica particularmente agradable de microbenchmark es la capacidad de verificar automáticamente los resultados de las expresiones de referencia con una función especificada por el usuario. Esto se demuestra a continuación, donde nuevamente comparamos tres métodos que computan el vector de coeficientes de un modelo lineal."

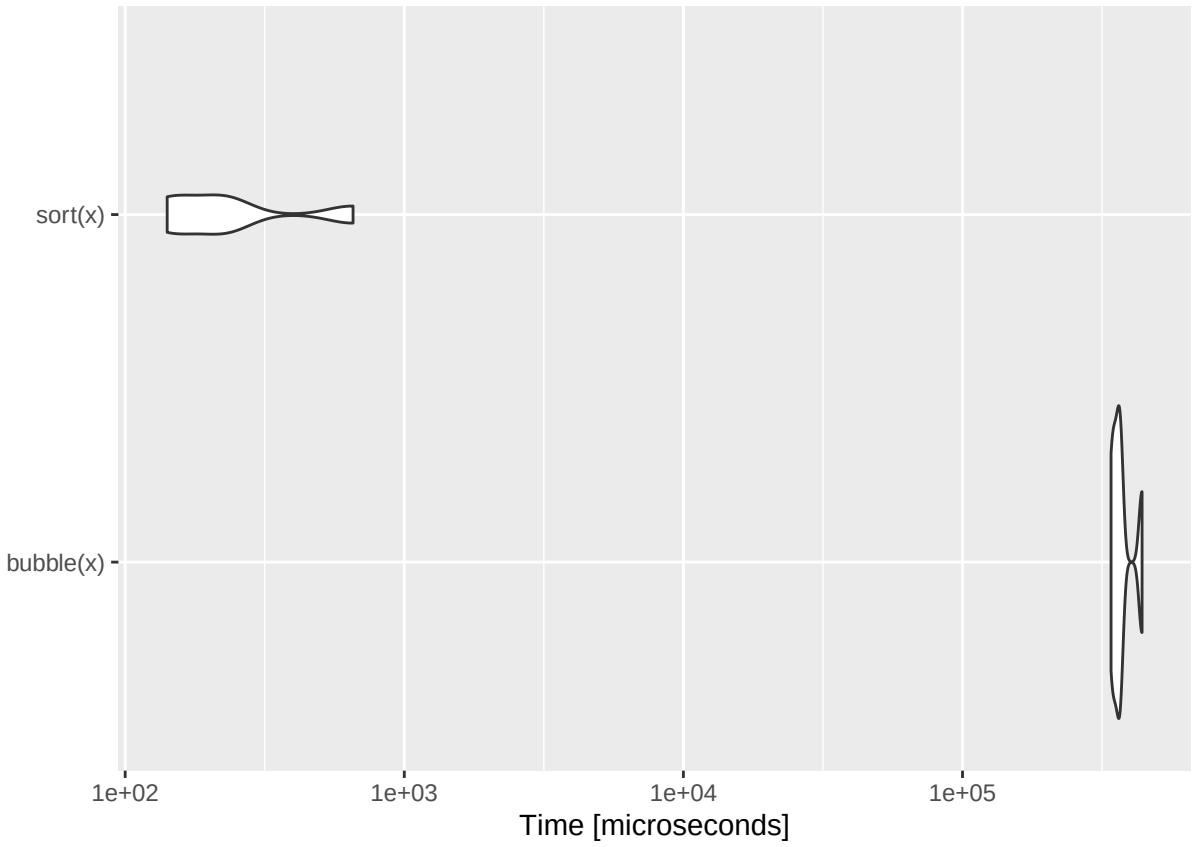
"A continuacion vemos las lineas de codigo para calcular el tiempo de ejecucion de el codigo para ordenar por metodo bubble. Tomaremos una muestra de 2000 numeros entre el 1 y el 1000000 .Lo compararemos con el comando *sort* de R:" "Utilizaremos como ejemplo x"

```
library(microbenchmark)

x <- sample(1:1000000,2000)

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


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

```
## Unit: microseconds
##       expr        min       lq        mean     median       uq        max neval
##  bubble(x) 354755.201 369027.3 448074.2010 439604.901 485585.1 591398.502     5
##    sort(x)    200.701    217.5    303.9606    317.301    376.8    407.501     5
```

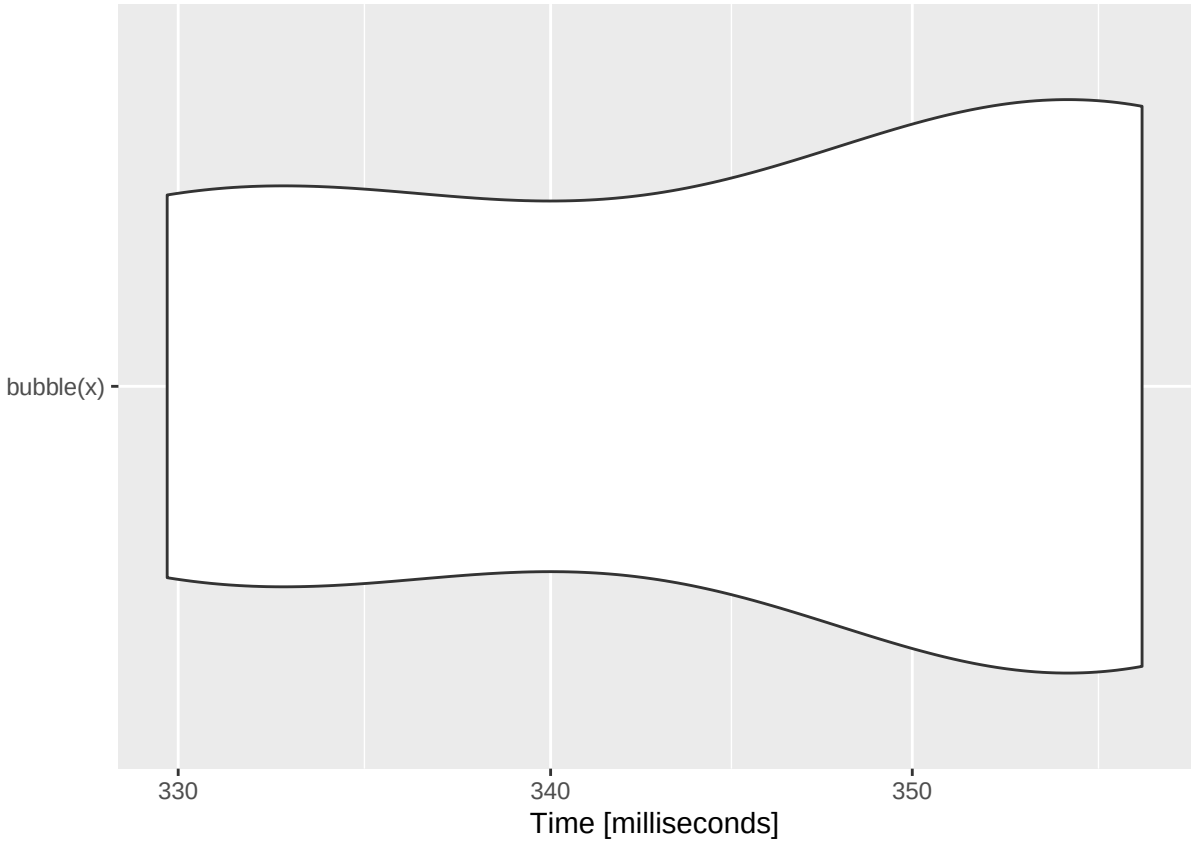
```
library(ggplot2)
autoplot(microbenchmark(bubble(x),sort(x), times=5))
```

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



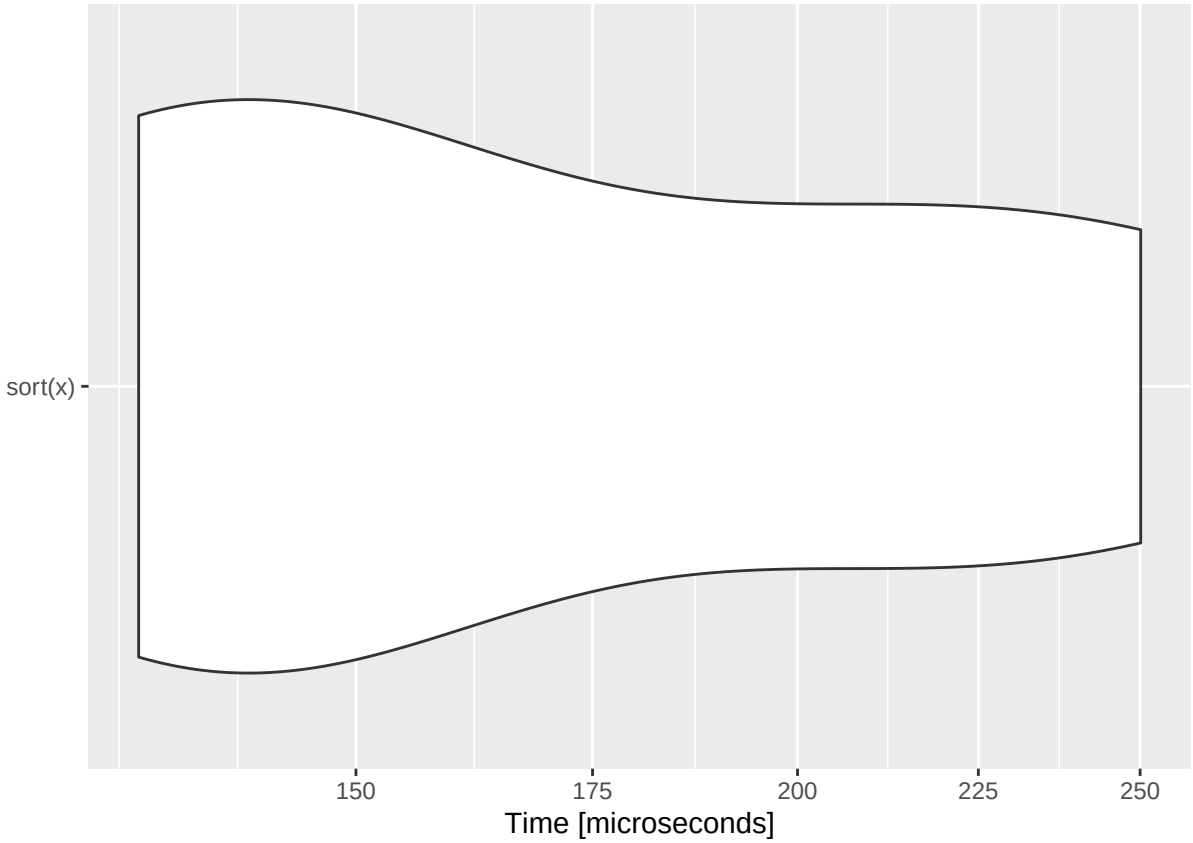

```
autoplot(microbenchmark(bubble(x),times=5))
```

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




```
autoplot(microbenchmark(sort(x),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)
casos_3_ <- read_delim("C:/Users/MI EQUIPO/Downloads/casos (3).csv",
    delim = ";", escape_double = FALSE, col_types = cols(...2 = col_number()),
    trim_ws = TRUE, skip = 3)
```

```
## New names:
## * `` -> `...3`
```

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

```
#Estadística de casos
summary(casos_3_$`2`)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    2.00   53.25  283.50  546.78 1054.00 1715.00
```

```
  m <- length(casos_3_$`2`)
 F <- (casos_3_$`2`)/(casos_3_$`2`)
#Estadísticos de F
mean(F,na.rm = TRUE)
```

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

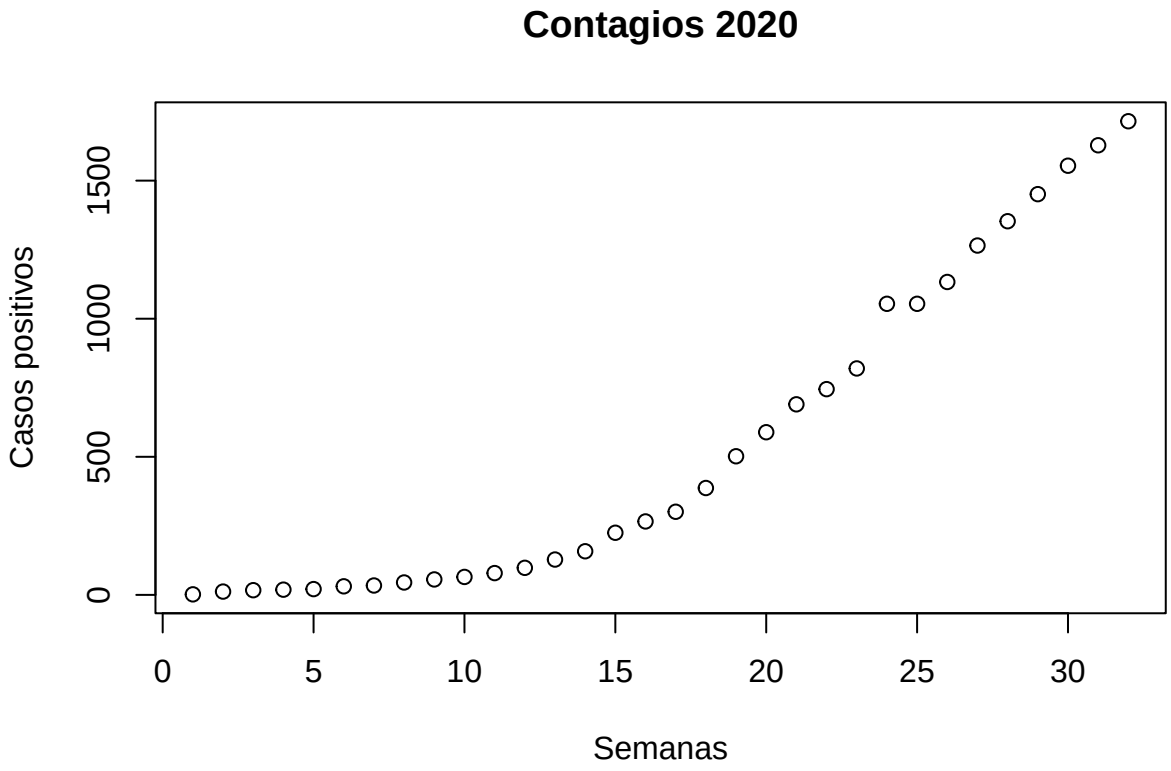
```r
sd(F,na.rm = TRUE)
```

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

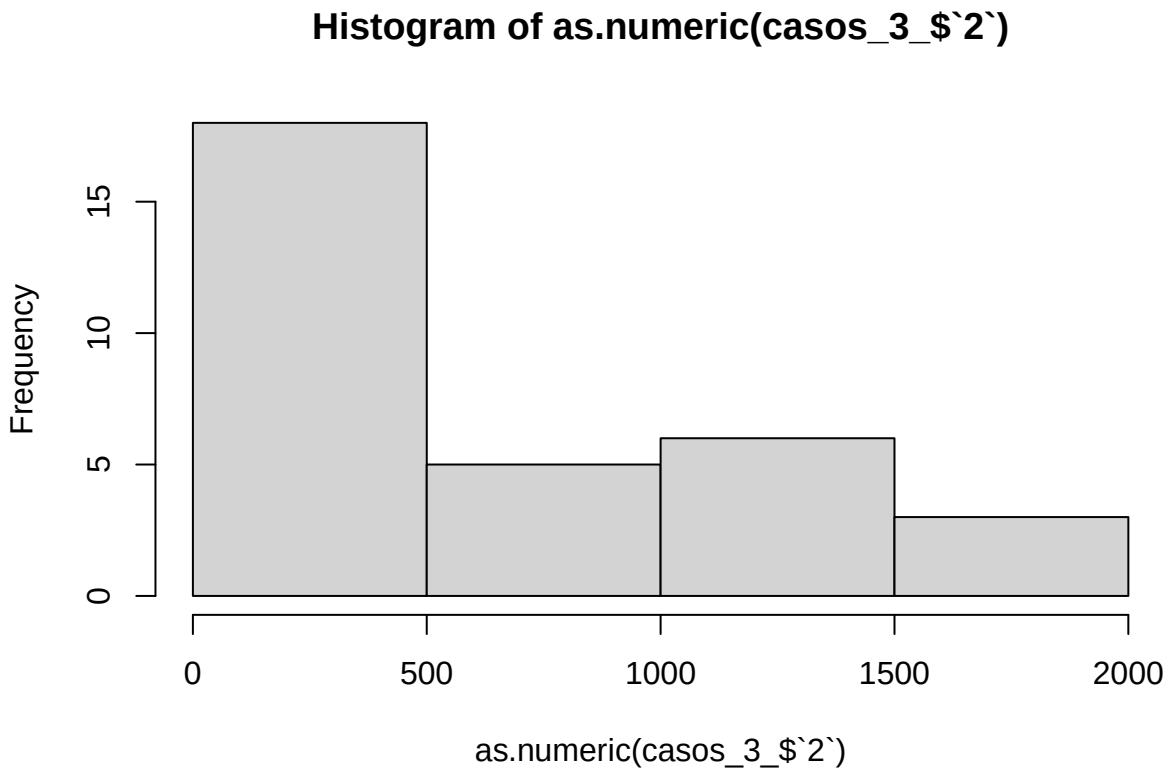
```r
var(F,na.rm = TRUE)
```

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

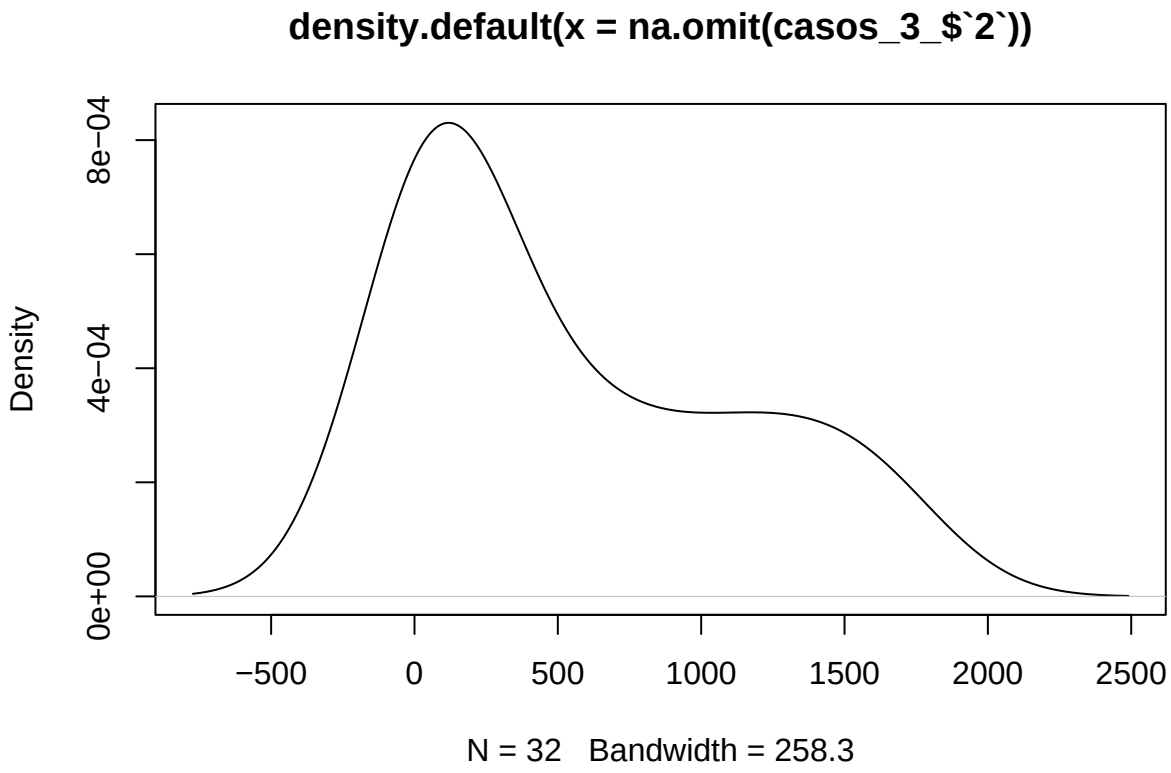
```r
#Grafico de casos
plot(casos_3_$`2`,main="Contagios 2020", ylab="Casos positivos", xlab="Semanas")
```




```r
hist(as.numeric(casos_3_$`2`))
```




```
plot(density(na.omit(casos_3_$`2`)))
```



3.2. Accediendo a los datos actualizados del Covid-19————————

#determinacion en que fecha se contagiaran 40 millones de personas

```
# Numeros de casos semanales en Argentina
f1<-51778

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] 14
```

```
vec
```

```
##  [1]    51778.00    83880.36   135886.18   220135.62   356619.70   577723.91
##  [7]   935912.74  1516178.64  2456209.39  3979059.21  6446075.93 10442643.00
## [13] 16917081.66 27405672.29 44397189.11
```

Con el numero de casos actuales semanales en Argentina y con el factor de contagio F=1.62, tardariamos 14 semanas en llegar a los 44 millones de contagiados

“‘