Elementos de programación en R-Cran

PALMA Ricardo R. - Facultad de Ingeniería
<rpalma@uncu.edu.ar>
MASERA Gustavo A. - Facutad de Filosofía y Letras
<gustavo.masera@fing.uncu.edu.ar>
Universidad Nacional de Cuyo

25 de marzo de 2020

Elementos básicos de programación

En este breve tutorial examinaremos algunos elementos del lenguaje de programación R y como valernos de ello para resolver problemas de la vida cotidiana. Apelaremos a ejemplos bien conocidos, pero además mostraremos las soluciones que desarrollaremos contra las misma que ya están implementadas en R. Comparando el costo computacional, medido como tiempo de ejecución. Esto nos permitirá entender la calidad del algoritmo que implementemos. Como excusa para introducirnos propondremos realizar tres experimentos y medir el tiempo ejecución.

Veremos:

- Generar un vector secuencia
- Implementación de una serie Fibonacci
- Ordenación de un vector por método burbuja
- Progresión geométrica del COVid-19
- Algoritmo de funciones estadísticas

1. Algunas ideas de como medir el tiempo de ejecucion

Michos de ustedes están familiarizados con Octave o Matlab. Algunos recordarán que para invertir matrices y saber que método era más eficiente su utilizaman los comandos tic y tac. Por ejemplo se generaba una matriz A, se ejecutaba el comando tic que disparaba una especie de cronómetro interno, luego se invertía seguiendo una algoritmo de determinante y finalmente se ejecutava el tomando toc que detenía el reloj y entregaba el tiempo de ejecución. Luego se repetía el mismo procedimiento, pero en lugar de hacerlo con determitante se usaba un algoritmo de matriz LU.

Una búsqueda rápida en línea nos revela al menos tres paquetes R para comparar performance del código R (rbenchmark, microbenchmark y tictoc). Estos

además de medir el tiempo nos indican porcentaje de memoria y microprocesador utilizados.

Además, la base R proporciona al menos dos métodos para medir el tiempo de ejecución del código R (Sys.time y system.time), que es una aproximación batante útil para un curso como el que desarrollamos.

A continuación, paso brevemente por la sintaxis del uso de cada una de las cinco opciones, y presento mis conclusiones al final.

1.1. Usando Sys.time

El tiempo de ejecución de un fragmento de código se puede medir tomando la diferencia entre el tiempo al inicio y al final del fragmento de código leyendo los registros del RTC (Real Time Clock. Simple pero flexible: como un relojito de arena :.

```
> sleep_for_a_minute <- function() { Sys.sleep(14) }
> start_time <- Sys.time()
> sleep_for_a_minute()
> end_time <- Sys.time()
> end_time - start_time
```

```
Time difference of 14.01499 secs
```

Hemos generado una función que antes no existía y la hemos usado. Deficiencias: Si usas el comando dentro de un documento en R-Studio te demorarás mucho tiempo cuando compiles un PDF o una presentación.

1.2. 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.

La versión de desarrollo más reciente de tictoc se puede instalar desde github:

```
install packages (tictoc)
```

```
> 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"
```

```
> toc()
```

```
sleeping: 2.005 sec elapsed
```

Uno puede cronometrar solamente un fragmento de código a la vez:

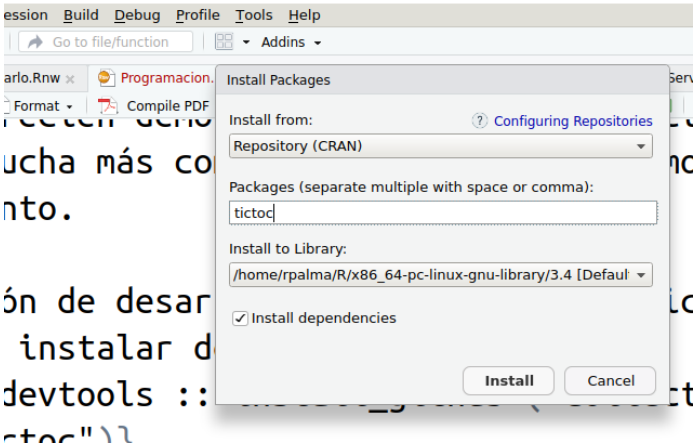


Figura 1: Antes de usar una biblioteca es necesario descargarla de internet

1.3. Biblioteca rbenchmark

La documentación de la función benchmark del paquete rbenchmark R lo describe como ün 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.

Recuerda antes de ejecutar

$$library\ (\ cualquiercosa\ ) \tag{1}$$

debes haber cargado en tu máquina la biblioteca que quieres invocar usando

$$install.packages\ (cualquiercosa)\ . \tag{2}$$

```
> library(rbenchmark)
> # lm crea una regresión 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   0.115    1.000     0.114        0
1            lm         1000   0.721    6.270     0.714        0
2 pseudoinverse         1000   0.120    1.043     0.120        0

>
```

En el informe de salida nos dice que cantidad de tiempo consume cada parte del código.

1.4. Biblioteca Microbenchmark

La versión de desarrollo más reciente de microbenchmark se puede instalar desde github:

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.

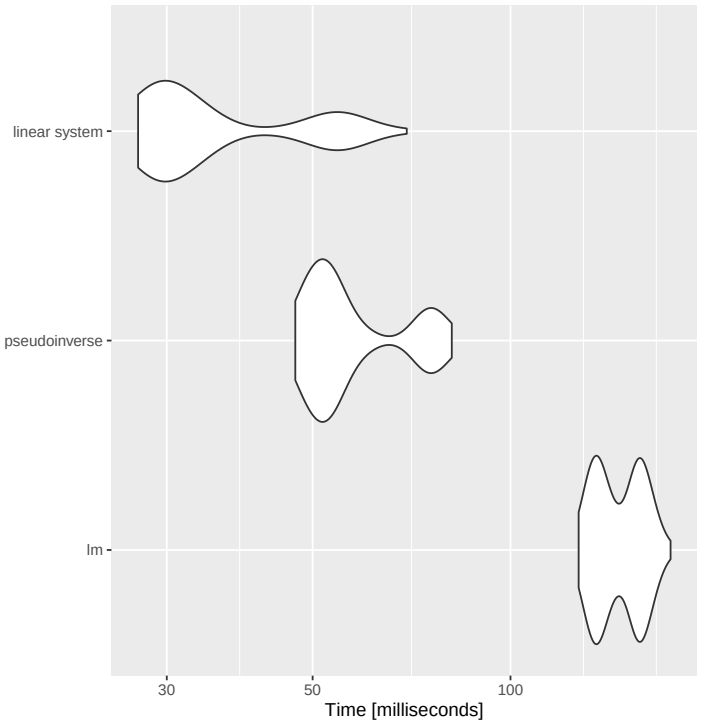
Me parece que una característica particularmente agradable de microbenchmark es la capacidad de verificar automáticamente los resultados de las expresiones de referencia con una función especificada por el usuario. Esto se demuestra a continuación, donde nuevamente comparamos tres métodos que computan el vector de coeficientes de un modelo lineal.

```
> 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
            lm 126.96445 135.37963 146.58844 144.78991 157.41872 175.19859
 pseudoinverse  47.06115  50.34671  57.87653  53.16041  61.37934  81.41464
 linear system  27.13323  29.11787  36.91435  31.00424  47.90309  69.53589
 neval cld
   100   c
   100  b
   100 a

> library(ggplot2)
> autoplot(mbm)
```



1.5. Consigan del trabajo

El trabajo de hay que presentar implica revisar los algoritmos que se presentan a continuación. Deberá ejecutarlos primero en la línea de comando de la consola.

Luego deberá elegir alguno de los métodos vistos para medir la performance y comparar los resultados con otros compañeros que hayan usado otros métodos para medir la performance.

Luego todo deberá entregarse en un informe en formato pdf construido con RStudio, archivo rsweave.

2. Generar un vector secuencia

De echo R. tiene un comando para generar secuncias llamado "seq". Recomendamos ejecutar la ayuda del comando en RStudio.

Pero utilizaremos el clásico método de secuanecias de anidamiento for, while, do , until.

Generaremos una secuencia de números que de dos en dos entre 1 y 100.000.

2.1. Secuencias generada con for

```
> 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
```

2.2. Secuencia generada con R

```
> A <- seq(1,1000000, 2)
> head (A)

[1]  1  3  5  7  9 11

> tail (A)

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

CONSIGNA: Comparar la performance con systime

3. Implementación de una serie Fibonachi o Fibonacci

En matemáticas, la sucesión o serie de Fibonacci es la siguiente sucesión infinita de números naturales:

"0,1,1,2,3,5,8 ... 89,144,233 ..."

La sucesión comienza con los números 0 y 1,2 a partir de estos, ■cada término es la suma de los dos anteriores■, es la relación de recurrencia que la define.

A los elementos de esta sucesión se les llama números de Fibonacci. Esta sucesión fue descrita en Europa por Leonardo de Pisa, matemático italiano del siglo XIII también conocido como Fibonacci. Tiene numerosas aplicaciones en ciencias de la computación, matemática y teoría de juegos. También aparece en configuraciones biológicas, como por ejemplo en las ramas de los árboles, en la disposición de las hojas en el tallo, en las flores de alcachofas y girasoles, en las inflorescencias del brécol romanesco, en la configuración de las piñas de las coníferas, en la reproducción de los conejos y en cómo el ADN codifica el crecimiento de formas orgánicas complejas. De igual manera, se encuentra en la estructura espiral del caparazón de algunos moluscos, como el nautilus.

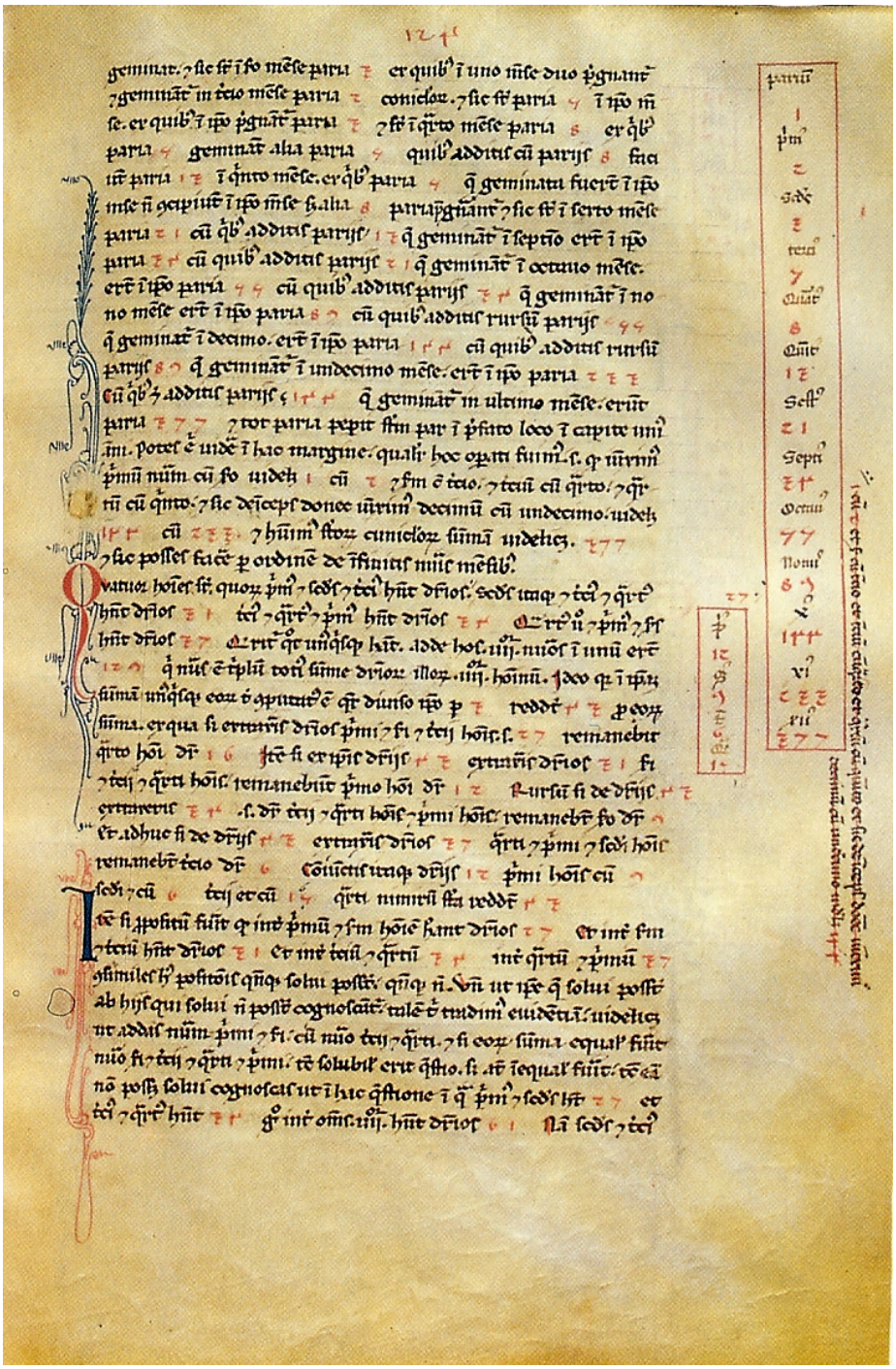


Figura 2: Original de la Biblioteca Uninersidad de Florencia. Liber Abachi - Autor Fibonacci

3.1. Definición matemática recurrente

$$f_0 = 0 \tag{3}$$

$$f_1 = 1 \tag{4}$$

$$f_{n+1} = f_n + f_{n-1} \tag{5}$$

```
> for(i in 0:5)
+ { a<-i
+ b <-i+1
+ c <- a+b
+ # comentar esta línea para conocer el número más grande hallado
+ print(c)
+ }

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

>
> #Descomentar esta línea para saber el número más grande hallado
> # print(c)
```

CONSIGNA: ¿Cuantas iteraciones se necesitan para generar un número de la serie mayor que 1.000.000 ?

4. Ordenación de un vector por método burbuja

La Ordenación de burbuja **(Bubble Sort en inglés)** es un sencillo algoritmo de ordenamiento. Funciona revisando cada elemento de la lista que va a ser ordenada con el siguiente, intercambiándolos de posición si están en el orden equivocado. Es necesario revisar varias veces toda la lista hasta que no se necesiten más intercambios, lo cual significa que la lista está ordenada. Este algoritmo obtiene su nombre de la forma con la que suben por la lista los elementos durante los intercambios, como si fueran pequeñas "burbujas". También es conocido como el método del intercambio directo. Dado que solo usa comparaciones para operar elementos, se lo considera un algoritmo de comparación, siendo uno de los más sencillos de implementada.

```
> # Tomo una muestra de 10 números ente 1 y 100
> x<-sample(1:100,10)
> # Creo una función 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)
> #Muestra obtenida
> x

 [1] 23 73 10 16 22 14 71  7 40 64

> #Muestra Ordenada
> res

 [1]  7 10 14 16 22 23 40 64 71 73

> #Ordanación con el coamando SORT de R-Cran
>
> sort(x)

 [1]  7 10 14 16 22 23 40 64 71 73
```

CONSIGNA: Compara la performance de ordenación del método burbuja vs el método sort de R
Usar método microbenchmark para una muestra de tamaño 20.000

5. Progresión geométrica del COVid-19

Modelado matemático de una epidema

La cantidad delta represent la variación de casos registrados de un dia para otro

$$\Delta_n = I_{n+1} - I_n \tag{6}$$

Esta cantidad está influenciada por dos variables que llamaremos E (exposición) y p (probabilidad de contagio), representaremos a los Infectados por I

$$\Delta_n = I_n * E * p \tag{7}$$

Pero antes habíamos calculado Delta n , de modo que lo reemplazaremos.

$$\Delta_n = I_{n+1} - I_n = I_n * E * p \tag{8}$$

Podemos pronosticar que cantidad de infectados habrá mañana en un país si despejamos

$$I_{n+q} \tag{9}$$

. Despejando tenemos ...

$$I_{n+1} = I_n(E * p) + I_n \tag{10}$$

Luego

$$I_{n+1} = I_n(1 + E * p) \tag{11}$$

Llamaremos factor de contadio F a :

$$F = (1 + E * p) \tag{12}$$

Con estos valores prodemos predecir cuando la epidemia terminará de contagiar a todos los habitantes de un país. F es siempre mayor que 1 y podríamos pensar que E y p son una probabilidad conjunta.

Así tenemos entonces

$$I_{n+1} = I_n * F \tag{13}$$

Ecuación general de una epidemia

Recuperaremos los datos de Argentina entre el inicio de la epidemia y la fecha actual.

```
> library(readr)
> casos_A <-read_delim("/home/rpalma/AAA_Datos/2020/Posgrado/COVID/casos.csv", ";", escape
```

Estadística de casos

```
> summary(casos_A$Casos)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
   1.00   14.50   31.00   40.67   60.50  128.00       1
```

Se puede calcular F dividiendo los infectados de hoy sobre los de ayer

$$F = I_{n+1}/I_n \tag{14}$$

De modo que

```
> 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] 1.62166

> sd(F,na.rm = TRUE)

[1] 1.283397

> var(F,na.rm = TRUE)

[1] 1.647107
```

5.1. Accediendo a los datos actualizados del Covi-19

Los datos más actualizados del avance de la epidemia se pueden encontrar en el sitio :

<https://github.com/CSSEGISandData/COVID-19/raw/master/csse_covid_19_data/csse_covid_19_time_series> GitHub accedido 9 de Marzo.

Prueba aceder al link y guardarlo en una carpeta conocida.

CONSIGNA: usando las ecuaciones de la epidemia determinar en que fecha se contagiarían 40 millones de personas usar los tados de F=1.62