Contribution Title*

Ricardo Palma[1][0000−1111−2222−3333], Gustavo Masera[2,3][1111−2222−3333−4444], and Cristian Marino[3][2222−−3333−4444−5555]

[1] Universidad Nacional de Cuyo CP 5502 Arg.
[2] Instituto de Ingeniería Industrial
rpalma@uncu.edu.ar
http://themys.sid.uncu.edu.ar
[3] Laboratorio Industria 4.0, Facultad de Ingeniería, Mendoza
ricardo.rpalma@gmail.com

Abstract. Este módulo establece los primeros pasos de programación en R-Cran utilizando la plantilla LNCS de Springer.

Keywords: Programing Languaje · R-Cran · RMarkdown

1 Elementos básicos de programación

En este breve tutorial examinaremos algunos elementos del lenguaje de programacion 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
- La penitencia de Taylor
- Algoritmo de funciones estadísticas

1.1 Algunas ideas de como medir el tiempo de ejecucion

Es muy frecuente la pregunta sobre cuál algoritmo es el mejor para realizar una tarea específica o resolver un problema. Una de las técnicas que suele utilizarse para esto es el benchmarking. No necesariamente un algoritmo es siempre mejor que otro. Por el contrario existen mérticas para hacer el benchmarking que comparan el uso del procesador, la memoria requerida, el tiempo que tarda en

* Universidad Nacional de Cuyo

resolver el problema, la exactitud de la solución, etc. El benchmarking no me dice cual es el mejor, pero me da información sobre cuál es el más conveniente según las características del problema y el recurso físico (máquina) que tengo.

Muchos de ustedes están familiarizados con Octave o Matlab. Algunos recordarán que para invertir matrices y saber que método era más eficiente su utilizaban 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 siguiendo una algoritmo de determinante y finalmente se ejecutaba 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 bastante 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.

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 c?digo 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 14.00634 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.

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 evaluacion 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: 2.013 sec elapsed
```

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

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.

Recuerda antes de ejecutar

```
##library ( cualquiercosa )##
```

debes haber cargado en tu máquina la biblioteca que quieres invocar usando **install.packages (cualquiercosa)** .

«bench_mark,echo=TRUE»=

```
##            test replications elapsed relative user.self sys.self
## 3 linear system          1000   0.114    1.000     0.114    0.000
## 1            lm          1000   0.718    6.298     0.709    0.008
## 2 pseudoinverse          1000   0.140    1.228     0.124    0.016
```

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

1.2 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 ejecuci?n de múltiples fragmentos de cdigo 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.

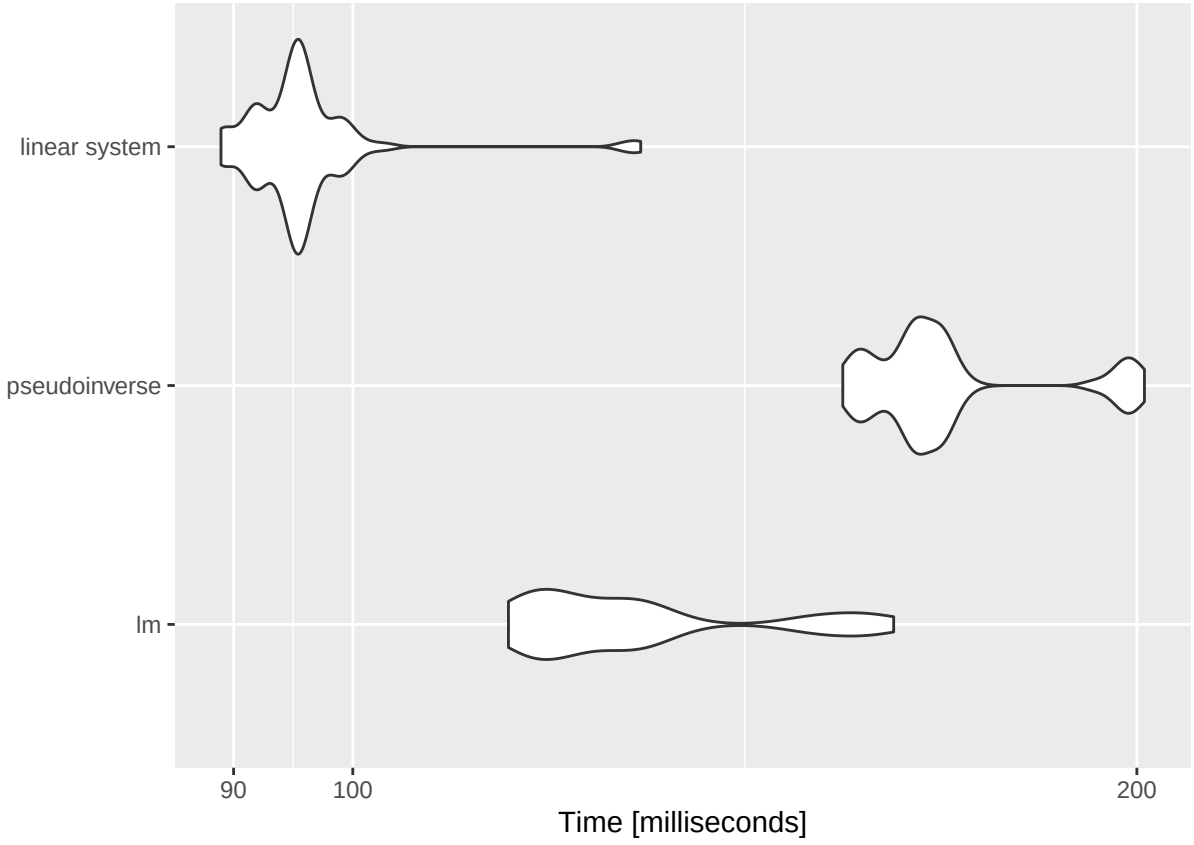
```
library(microbenchmark)

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 neval
```

```
##             lm 114.75118 118.24176 128.92028 124.31368 131.09987 161.3118   100
##  pseudoinverse 154.21624 161.89929 169.44642 165.13217 168.88293 201.3425   100
##  linear system  89.00023  92.78031  95.83777  95.17456  96.27806 128.9860   100
##  cld
##   b
##    c
##  a
```

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

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



1.3 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 RMarkdow.

1.4 Generar un vector secuencia

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

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

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

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

Secuencias generada con for

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

```
tail (A)
```

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

Secuencia generada con R

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

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

```
  CONSIGNA: Comparar la performance con systime
```

1.5 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 como 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.

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

1.6 Definición matemática recurrente

$$f_0 = 0 \tag{1}$$

$$f_1 = 1 \tag{2}$$

$$f_{n+1} = f_n + f_{n-1} \tag{3}$$

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

1.7 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.

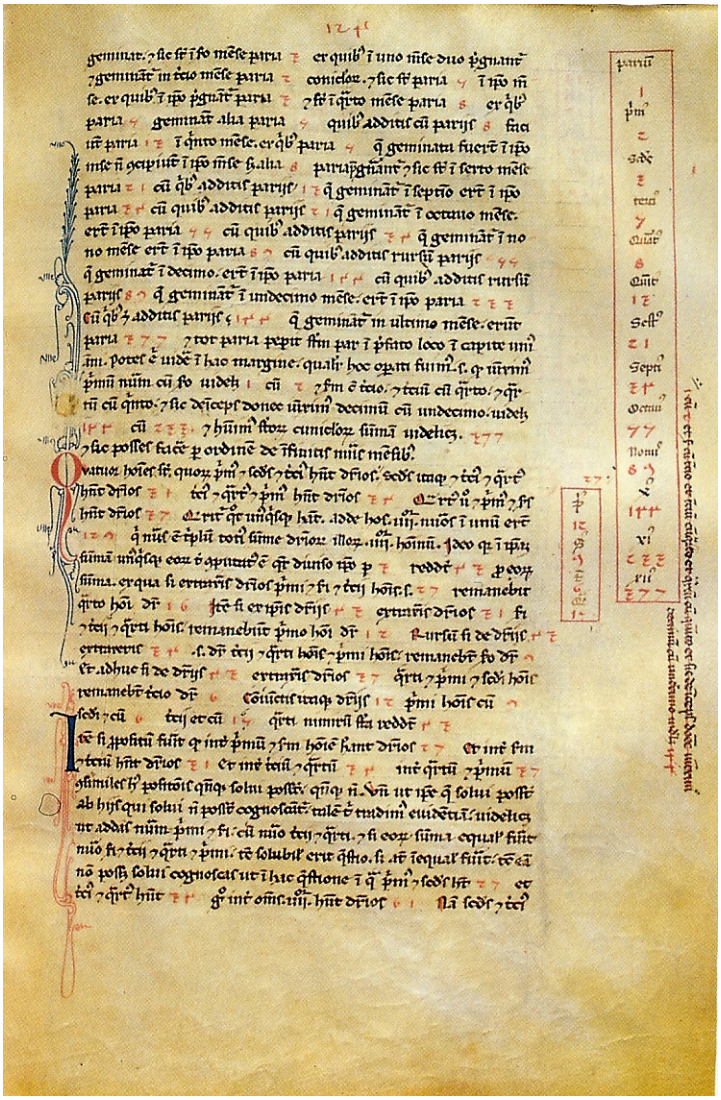


Fig. 1. Fig 3

```
# Tomo una muestra de 10 números ente 1 y 100
x<-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)
}
res<-burbuja(x)
#Muestra obtenida
x
```

```
##  [1]  7 71 10 72 37 28 64 82 19 88
```

```
#Muestra Ordenada
res
```

```
##  [1]  7 10 19 28 37 64 71 72 82 88
```

```
#Ordanaci?n con el coamando SORT de R-Cran

sort(x)
```

```
##  [1]  7 10 19 28 37 64 71 72 82 88
```

```
  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

2 First Section

2.1 A Subsection Sample

Please note that the first paragraph of a section or subsection is not indented. The first paragraph that follows a table, figure, equation etc. does not need an indent, either.

Subsequent paragraphs, however, are indented.

Sample Heading (Third Level) Only two levels of headings should be numbered. Lower level headings remain unnumbered; they are formatted as run-in headings.

Sample Heading (Fourth Level) The contribution should contain no more than four levels of headings. Table 1 gives a summary of all heading levels.

Table 1. Table captions should be placed above the tables.

temperature	pressure
0	0.0002
20	0.0012
40	0.0060
60	0.0300
80	0.0900
100	0.2700

Displayed equations are centered and set on a separate line.

$$x + y = z \tag{4}$$

Please try to avoid rasterized images for line-art diagrams and schemas. Whenever possible, use vector graphics instead (see Fig. 2).

Theorem 1. *This is a sample theorem. The run-in heading is set in bold, while the following text appears in italics. Definitions, lemmas, propositions, and corollaries are styled the same way.*

The environments 'definition', 'lemma', 'proposition', 'corollary', 'remark', and 'example' are defined in the LLNCS documentclass as well.

Proof. Proofs, examples, and remarks have the initial word in italics, while the following text appears in normal font.

For citations of references, use [?]. Multiple citations are grouped [?,?].

Acknowledgements Please place your acknowledgments at the end of the paper, preceded by an unnumbered run-in heading (i.e. 3rd-level heading).

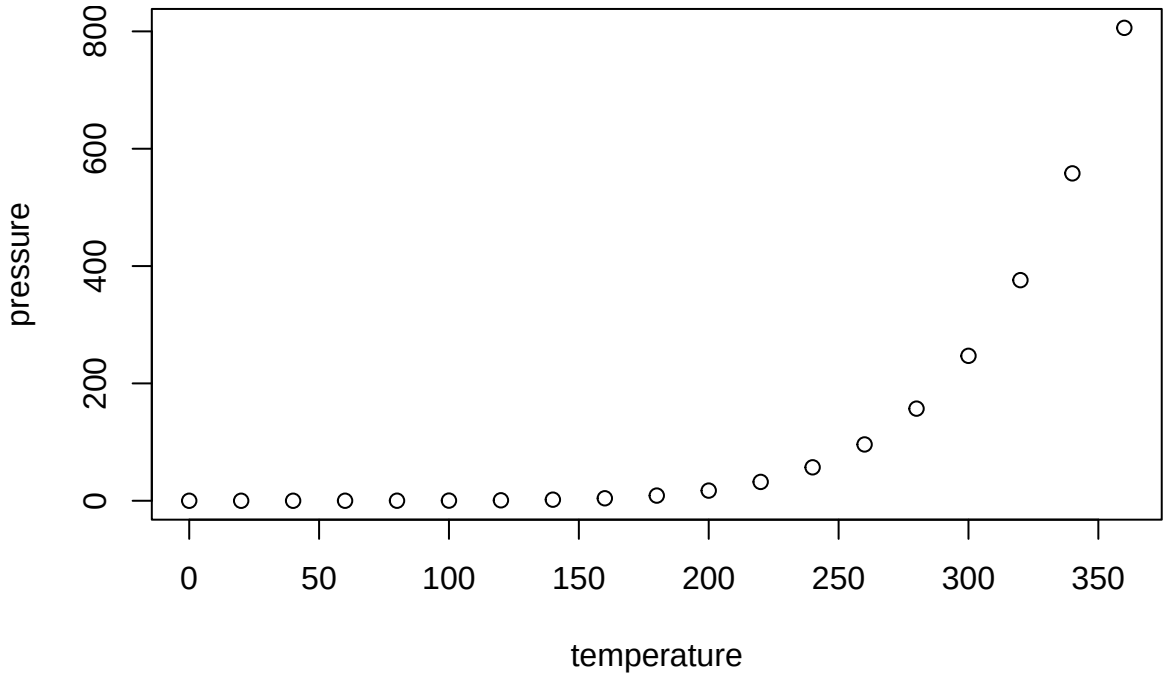


Fig. 2. A figure caption is always placed below the illustration. Please note that short captions are centered, while long ones are justified by the macro package automatically.