ProgM Lab01

Javier Ortín

2026-01-28

Repaso de R

Podemos usar el índice de R (arriba a la derecha en la parte de scripting del IDE). Si estamos en un RScript, podemos añadir "—-" al final de un comentario para generar un título.

Elementos básicos

```
a <- 7
b <- 3
d <-  a+b
```

No funciona como una hoja de cálculo: si cambiamos el valor de b, el valor de d no cambia automáticamente, sino que hay que volve a ejecutar la línea que asigna su valor.

##Vectores en R Los vectores se definien con la función "c" (de concatenar)

```
a <- c(2,4,6,8);a
```

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

Una vez ejecutada la línea anterior, el valor de la variable a cambiará en la tabla "Values" de RStudio.

```
1:4
```

```
## [1] 1 2 3 4
```

```
4:1
```

```
## [1] 4 3 2 1
```

```
a[3]
```

```
## [1] 6
```

```
a[1:3]
```

```
## [1] 2 4 6
```

Podemos usar la función \texttt{seqç de R para generar vectores siguiento el formato "start, step, stop":

```
seq(4,10,2) #Desde 4 hasta 10 en pasos de 2
```

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

Podemos acceder a ciertas posiciones de un vector al indexar por otro vector:

```
a[c(1,3)] #Posiciones 1 y 3
```

```
## [1] 2 6
```

```r
a[-3] # Todos los índices salvo el 3
```

```
## [1] 2 4 8
```

```r
a[-c(1,3)] # Todos menos el 1 y el 3
```

```
## [1] 4 8
```

Hallar mínimos de un vector:

```r
min(a)
```

```
## [1] 2
```

```r
a[which.min(a)] #which.min devuelve los índices de los mínimos
```

```
## [1] 2
```

Matrices en R

```r
?matrix
```

```
## starting httpd help server ... done
```

```r
A = matrix(1:9, ncol=3); A
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

```r
A = matrix(1:9, ncol=3, byrow=T); A #rellenar por filas
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
## [3,]    7    8    9
```

En este caso, RStudio muestra la variable A en la sección de "Data", no de "Values". Además, muestra los contenidos de la matriz por columnas. Veamos cómo se accede a los elementos de una matriz:

```r
A[3,1] #Acceso a un elemento
```

```
## [1] 7
```

```r
A[3,]  #Acceso a una fila
```

```
## [1] 7 8 9
```

```r
A[,1]  #Acceso a una columna
```

```
## [1] 1 4 7
```

```r
A[,-2] #Todo menos la columna 2
```

```
##      [,1] [,2]
## [1,]    1    3
## [2,]    4    6
## [3,]    7    9
```

A diferencia de listas y dataframes, no se pueden añadir filas a las matrices por asignación directa.

```
A = cbind(A, 10:13); A
```

```
## Warning in cbind(A, 10:13): number of rows of result is not a multiple of
## vector length (arg 2)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3   10
## [2,]    4    5    6   11
## [3,]    7    8    9   12
```

```
# Definición de matriz como vector de vectores:
#Por filas
A = rbind(c(1,3,5),
          c(3,4,5),
          c(1,7,8)); A
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    3    4    5
## [3,]    1    7    8
```

```
#Por columnas
A = cbind(c(1,3,5),
          c(3,4,5),
          c(1,7,8)); A
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    1
## [2,]    3    4    7
## [3,]    5    5    8
```

Resolución de sistemas de ecuaciones en R

Resolveremos el siguiente sistema:

$$\begin{cases} 5x_2 + x_3 = 7 \\ 7x_2 - x_3 = 5 \\ 11x_1 + x_2 + x_3 = 14 \end{cases}$$

```
A = rbind(c(0,5,1),
          c(0,7,-1),
          c(11,1,1)); A
```

```
##      [,1] [,2] [,3]
## [1,]    0    5    1
## [2,]    0    7   -1
## [3,]   11    1    1
```

```
b = c(7,5,14); b
```

```
## [1]  7  5 14
```

```
solve(A,b)
```

```
## [1] 1 1 2
```

```
#Alternativamente
solve(A)%*%b
```

```
##      [,1]
## [1,]    1
## [2,]    1
## [3,]    2
```

En la última línea, `solve(A)` halla la matriz inversa de `A`.Finalmente, hacemos el producto de matrices con \texttt{%*%}.

Listas en R

Las listas en R son heterogéneas: podemos almacenar elementos dispares de cualquier dimensión.

```
ejemplo = list(1:13, A, "cadena", c(T,F), list(7,b)); ejemplo
```

```
## [[1]]
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13
##
## [[2]]
##      [,1] [,2] [,3]
## [1,]    0    5    1
## [2,]    0    7   -1
## [3,]   11    1    1
##
## [[3]]
## [1] "cadena"
##
## [[4]]
## [1]  TRUE FALSE
##
## [[5]]
## [[5]][[1]]
## [1] 7
##
## [[5]][[2]]
## [1]  7  5 14
```

La variable `ejemplo` aparece recogida en la sección \texttt{Dataç de RStudio como "`List of 5`". Veamos cómo acceder a sus elementos:

```
ejemplo[[2]] # Accedemos a la matriz A
```

```
##      [,1] [,2] [,3]
## [1,]    0    5    1
## [2,]    0    7   -1
## [3,]   11    1    1
```

```
ejemplo[[2]][3,1] # Elemento (3,1) de A
```

```
## [1] 11
```

```
ejemplo[[5]][[2]][3] #Accedemos al tercer elemento de b
```

```
## [1] 14
```

Para evitar esta sintaxis tan poco amigable, podemos dar nombres a los elementos de la lista y acceder a ellos en base a sus alias usando el signo $.

```r
names(ejemplo) = c("A","B","C","D","E")
names(ejemplo$E) = c("a","b")
ejemplo$E$b[3] #Accedemos al tercer elemento de b
```

```
## [1] 14
```

```r
ejemplo[["A"]] #Accedemos al primer elemento de ejemplo
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13
```

Nótese que aunque los alias son cadenas de texto, accedemos a los elementos sin usar comillas cuando indexamos con "$", pero las usamos con "[]".

Funciones en R

Los corchetes son necesarios si vamos a programar una función cuyo código abarque más de una línea. Veamos un ejemplo muy sencillo: una función que calcule el área de un rectángulo.

```r
#En este caso no es necesario usar "{}"
area = function(lado1, lado2) lado1 * lado2
area(2,3)
```

```
## [1] 6
```

```r
#Otra versión que imprima el área además de devolverla:
area = function(lado1, lado2) {
  a = lado1 * lado2
  print(paste("El área es",a))
  return(a)
}
x = area(2,6)
```

```
## [1] "El área es 12"
```

Podemos asignar valores por defecto a los argumentos de una función para que tomen valores por omisión cuando no se expliciten. Es necesario ordenar los parámetros de modo que aquellos sin valor por defecto precedan a todos los que sí lo tengan.

```r
area = function(lado1, lado2=1) {
  a = lado1 * lado2
  print(paste("El área es",a))
  return(a)
}
x = area(10)
```

```
## [1] "El área es 10"
```

Adicionalmente, podemos usar "..." para aceptar una lista variable de argumentos sin tener que declararlos uno a uno. Ha de estar al final.

Estructuras condicionales

Podemos usar \texttt{ifç para comprobar si se cumple o no una condición.

```r
area = function(lado1, lado2=1) {
  if(lado1 == lado2) {
    print("Es un cuadrado")
  } else {
    print("Es un rectángulo")
```

```
  }
  a = lado1 * lado2
  print(paste("El área es",a))
  return(a)
}

x = area(2,2)
```

```
## [1] "Es un cuadrado"
## [1] "El área es 4"
```

```
x = area(2,3)
```

```
## [1] "Es un rectángulo"
## [1] "El área es 6"
```

Podemos usar \texttt{ifelseç de manera similar a un operador ternario. Aunque puede ejecutar sentencias (no solo devuelve expresiones), no es recomendable usarlo para esto:

Bucles en R

El bucle \texttt{forç ejecuta una serie de instrucciones para una variable que itera por un vector.

```
for(i in 1:5) print(i)
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
```

El bucle \texttt{whileç itera mientras se cumpla una condición:

```
x = 3
while(x <= 5) {
  print(x)
  x= x+1
}
```

```
## [1] 3
## [1] 4
## [1] 5
```