Programación en R:

Creación y manejo de vectores

Autor: Luis Javier Sánchez Martínez
luisja02@ucm.es

Departamento de Biodiversidad, Ecología y Evolución

Unidad de Antropología Física.

Universidad Complutense de Madrid (UCM)

Introducción

El objeto básico en R es el vector, es la manera más elemental de almacenar información. Un vector es una secuencia ordenada de datos. R dispone de muchos tipos de datos, entre los que destacamos:

- logical (lógicos: TRUE, verdadero, o FALSE, falso)
- integer (números enteros)
- numeric (números reales)
- complex (números complejos)
- character (palabras)

El concepto de vector en R es algo diferente a la noción matemática de vector. Un vector es un objeto indexado a partir del índice 1, es decir cada elemento almacenado en el vector estará en una posición concreta del mismo. Se caracteriza, como veíamos antes, por contener objetos que pueden ser numeric, integer,logical o carácter, una restricción fundamental es que todos deben ser del mismo tipo.

En R no existen ni vectores fila ni vectores columna. En general, un vector de R dentro de una expresión matemática se comportará como sea necesario (vector fila o columna) para la expresión en la que se le requiere, aunque a veces hay que obligarlo. Por ejemplo, al aplicar la función mean sobre un vector, este se comporta como un vector columna, mientras que al multiplicar dos vectores estos se comportan como vectores fila.

En R para referirse a un elemento de un vector el índice se encierra entre corchetes. Por ejemplo, los elementos del vector w están indexados de forma que w[3] hace referencia al tercer elemento del vector w.

Creación de vectores

Las formas principales para crear vectores en R son dos:

• por asignación

• como resultado de alguna función.

- **Por asignación**

Vector de longitud 1

En el caso de asignar un único valor se crea un vector unitario que a veces se le llama “escalar”.

```
x <- 7
length(x)
[1] 1
```

```
"Hola" -> y
length(y)
[1] 1
```

para construir vectores de longitud mayor que 1 se utilizan funciones.

- **Con funciones**

Las principales funciones para crear vectores:

• c()

• seq()

• :

• rep()

Función c()

La función c() (combinar o concatenar) permite crear un vector escribiendo los diferentes valores que lo forman. Es totalmente flexible en cuanto al tipo de datos que maneja.

```
x <- c(1,2,3,4,5)
x
[1] 1 2 3 4 5
length(x)
[1] 5

vec1 <- c(1,2,3)
length(vec1)
[1] 3

c("Hola", "Adios") -> saludos
length(saludos)
[1] 2
```

Función seq()

La función seq() permite crear una secuencia de valores numéricos (reales) desde el valor origen from al final to con un incremento determinado por by. Su sintaxis es,

```
seq(from = 1, to = 1, by = ((to - from)/(length.out - 1)),
length.out = NULL, along.with = NULL, ...)
```

Ejemplos:

```
x <- seq(from = 1, to = 5, by = 1)  # El incremento por defecto es uno
x
[1] 1 2 3 4 5

y <- seq(1, 2, 0.1)            # Se permiten incrementos decimales
y
[1] 1.0 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2.0

w <- seq(from = 1, to = 2, length.out = 5)  # Se especifica el número de elementos
w
[1] 1.00 1.25 1.50 1.75 2.00

z <- seq(from = 7, to = -2, by = -2.5)        # Se permiten incrementos negativos decimales
z
[1] 7.0 4.5 2.0 -0.5
```

La función :

La función a:b es parecida a la función seq() pero más limitada. Crea una secuencia desde a hasta b con incremento de 1. El vector generado con : es de enteros. Obsérvese el resultado de estos códigos:

```
# Vector desde 1 hasta 5 de uno en uno y de clase numeric
y <- seq(from=1, to=5, by=1)
y
[1] 1 2 3 4 5
```

```
class(y)
```

```
[1] "numeric"
```

```
# Vector desde 1 hasta 5 de uno en uno y de clase entero
z <- 1:5
z
```

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

```
class(z)
```

```
[1] "integer"
```

```
    # Sin embargo
z == y
```

```
[1] TRUE TRUE TRUE TRUE TRUE
```

```
identical(z,y)
```

```
[1] FALSE
```

La función rep()

La función rep() repite el objeto x el número de veces determinado por el valor de la opción times. Su sintaxis es,

rep(x, times = 1, length.out = NA, each = 1)

```
x <- rep(3, times =4)
x
```

```
[1] 3 3 3 3
```

```
class(x)
```

```
[1] "numeric"
```

```
is.vector(x)
```

```
[1] TRUE
```

```
y <- rep(34, length.out=4)
y
[1] 34 34 34 34
class(y)
[1] "numeric"

s <- rep(c(1,2), length.out=5, each=2)
s
[1] 1 1 2 2 1
class(s)
[1] "numeric"
```

Argumento times (admite recycling pero no en todas las ocasiones)

```
vec <- 1:4
rep(x = vec, times = 3)                # times toma un único valor
[1] 1 2 3 4 1 2 3 4 1 2 3 4
rep(x = vec, times = c(1,2,3,4))               # times toma tantos valores como tiene el vector
[1] 1 2 2 3 3 3 4 4 4 4
rep(x = vec, times = c(1,2))                   # times no admite valores intermedios
```

Argumento each (no admite recycling)

```
rep(c(5,12,13), each=4)                # each toma un único valor
[1] 5 5 5 5 12 12 12 12 13 13 13 13
rep(c(5,12,13), each=c(2,3,5))                 # each toma un vector de valores
Warning in rep(c(5, 12, 13), each = c(2, 3, 5)): first element used of 'each' argument
```

```
[1] 5 5 12 12 13 13
```

En R se permite la definición de vectores sin ningún elemento,

```
x <- c()
length(x)          # Longitud del vector x
[1] 0
```

Cómo modificar vectores en R

Podemos modificar algunas entradas de un vector simplemente declarando sus nuevos valores. Esto se puede hacer entrada a entrada, o para todo un subvector de golpe.

```
x =1:10
x
[1] 1 2 3 4 5 6 7 8 9 10
x [3]=15                    #En la posición 3 escribimos 15
x [11]=25                   #Añadimos en la posición 11 un 25
x
[1] 1 2 15 4 5 6 7 8 9 10 25


x[c(2, 3, 4) ]=x[c(2, 3, 4) ]+10      # Sumamos 10 a las entradas en las posiciones 2, 3 y 4
x
[1] 1 12 25 14 5 6 7 8 9 10 25
```

Una manera rápida de editar un vector

Un vector se puede modificar fácilmente usando el editor de datos que incorpora RStudio. Para hacerlo, se aplica la función fix al vector que queremos editar. R abre entonces el vector en una nueva ventana de edición. Mientras esta ventana esté abierta, será la ventana activa de R y no podremos volver a nuestra sesión de R hasta que la cerremos. Los cambios que hagamos en el vector con el editor de datos se guardarán cuando cerremos esta ventana.

Probadlo. Cread un vector con R y abridlo en el editor. Por ejemplo:

```
x = rnorm(200,20,3)
fix (x)
```

Vectorización

Las funciones sobre vectores son en general elemento a elemento, esto es lo que se conoce como vectorización (en R).

Definimos dos vectores

```
x <- c(1, 2,3,4)
y <- c(10, 20, 30, 40)
x + y                 # Suma de vectores
[1] 11 22 33 44
x*y                   # Multiplicación elemento a elemento
[1] 10 40 90 160
x/y                   # División elemento a elemento
[1] 0.1 0.1 0.1 0.1
y^x                   # Potencia elemento a elemento
[1] 10 400 27000 2560000
```

Cuando se operan vectores de distinta longitud, R repite el más corto hasta que iguala la longitud del más largo. Este proceso se conoce como recycling.

```
# suma de vectores de diferente longitud
c(1,2,3,4) + c(1,2)
[1] 2 4 4 6
2 + c(1,3,5,7)
[1] 3 5 7 9
# Multiplicación de vectores de diferente longitud
2 * c(1,3,5,7)
[1]  2  6 10 14
c(2,2,2,2) * c(1,3,5,7)
[1]  2  6 10 14
```

Vectores de índices. Filtrado de vectores.

Los vectores de índices son de cuatro tipos:

• Vectores de enteros positivos

• Vectores de enteros negativos

• Vectores de cadena

• Vectores lógicos

- **Vectores de enteros positivos**

Los valores que contienen deben corresponderse con los índices del vector, es decir tiene que pertenecer al intervalo [1:length(vector)]. Los elementos correspondientes a los índices se extraen y los índices que no aparezcan serán omitidos.

```
x <- 15:30          # Vector que contiene los elementos
lon <- length(x)          # Longitud del vector que tiene los elementos a ser extraídos
cat(lon)
[1] 16

indice <- c(2, 3, 5, 8, 15)   # Vector de índices que se quieren extraer.
x[indice]                     # Elementos extraídos
[1] 16 17 19 22 29
```

- **Vectores de enteros negativos**

Los valores que contienen deben corresponderse con los índices del vector pero con el signo negativo (-) delante, es decir tiene que pertenecer al intervalo [-length(vector):1]. Los elementos correspondientes a los índices son omitidos.

```
x[-3]                         # Eliminar del vector x el tercer elemento
[1] 15 16 18 19 20 21 22 23 24 25 26 27 28 29 30

x[-c(1,3,6,7)]                # Eliminar del vector x los elementos 1,3,6 y 7
[1] 16 18 19 22 23 24 25 26 27 28 29 30
```

- **Vectores de cadenas**

Únicamente se pueden utilizar si el objeto tiene el atributo names que identifica sus elementos. En este caso un subvector de nombres se emplea de la misma forma que un vector de números enteros positivos.

```
semana <- 1:7                          # Ejemplo semana
names(semana) <- c('Domingo','Lunes','Martes','Miércoles','Jueves','Viernes','Sábado')

semana[c("Jueves","Martes")]           # Seleccionamos jueves y martes
Jueves Martes
5 3
```

- **Vectores lógicos**

También podemos extraer las entradas de un vector (o sus índices) que satisfagan alguna condición. Este proceso se realiza a través de operadores lógicos.

Operador	Igual	Disitnto	Menor	Mayor	Menor o igual	Mayor o igual	Negación	Conjunción	Disjunción
Signo	==	!=	<	>	<=	>=	!	&	|

Han de tener la misma longitud que el vector del que se quieren filtrar elementos. Los valores que corresponden a TRUE se extraen y los que corresponden a FALSE se omiten.

```
x=c(1 ,5 ,6 ,2 ,5 ,7 ,8 ,3 ,5 ,2 ,1)
x[x >3]              # Elementos mayores que 3
[1] 5 6 5 7 8 5
x[!x<4]              # Elementos que NO son menores que 4
[1] 5 6 5 7 8 5
x[x % %4==0]         # Elementos múltiplos de 4
[1] 8

y<-1:10
cond1<-y<5
cond2<-y>2

cond1 & cond2        # Que cumplan las dos condiciones
 [1]  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
y[cond1 & cond2]
[1] 3 4
```

Para obtener los índices de las entradas del vector que satisfacen una condición dada, podemos usar la función **which**. Esta función, aplicada a un vector de valores lógicos, da los índices de las posiciones de los TRUE.

```
x=c(1 ,5 ,6 ,2 ,5 ,7 ,8 ,3 ,5 ,2 ,1)
which (x >3)                    #Índices de los elementos mayores que 3
[1] 2 3 5 6 7 9
```

Función subset()

Tiene el mismo efecto que el filtrado por índices salvo en el tratamiento de los NA

```
x <- c(1, 1:3, NA, 12)
x[x>5]
[1] NA 12
subset(x, x>5)
[1] 12
```

Vectores de caracteres

Un vector de caracteres es aquel en que sus valores son cadenas alfanuméricas.

```
(Dias <- c("lunes", "martes", "miércoles", "jueves", "viernes", "sábado", "domingo"))
[1] "lunes" "martes" "miércoles" "jueves" "viernes" "sábado" "domingo"
```

R tiene algunos vectores de caracteres predefinidos:

```
LETTERS          # Letras mayúsculas (alfabeto inglés)
[1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q"
[18] "R" "S" "T" "U" "V" "W" "X" "Y" "Z"

letters                         # Letras minúsculas (alfabeto inglés)
[1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q"
[18] "r" "s" "t" "u" "v" "w" "x" "y" "z"

month.name                      # Nombre de los meses en inglés
[1] "January" "February" "March" "April" "May"
[6] "June" "July" "August" "September" "October"
```

```
[11] "November" "December"
```

```
month.abb            # Nombre abreviado de los meses en inglés
[1] "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct" "Nov"
[12] "Dec"
```

Manipulación de cadenas de caracteres en R

En esta sección vamos a trabajar con el paquete para manipulación de cadenas llamado stringr (del inglés string, cadena), que es parte de Tidyverse.

```
library(tidyverse)
```

En primer lugar veamos como crear cadenas de caracteres, es muy sencillo:
Puedes crear una cadena utilizando comillas simples o dobles. La recomendación es siempre utilizar ", a menos que quieras crear una cadena que contenga múltiples ".

```
string1 <- "Esta es una cadena de caracteres"
string2 <- 'Si quiero incluir "comillas" dentro de la cadena, uso comillas simples'
```

Si olvidas cerrar las comillas, verás un + en la consola, que es el signo de continuación para indicar que el código no está completo:

```
"Esta es una cadena de caracteres sin comillas de cierre
+
+
+ S.O.S.
```

Si esto te ocurre, ¡presiona la tecla Escape e inténtalo de nuevo!

Lo normal al trabajar con cadenas de caracteres es guardar múltiples cadenas en un vector de caracteres. Puedes crearlo usando c():

```
c("uno", "dos", "tres")
[1] "uno"  "dos"  "tres"
```

R base tiene muchas funciones para trabajar con cadenas de caracteres, pero las evitaremos porque pueden ser incosistentes, lo que hace que sean difíciles de recordar. En su lugar, utilizaremos funciones del paquete stringr, ya que tienen nombres más intuitivos y todas empienzan con str_

- Largo de cadena:
  La función str_length() te dice el número de caracteres de una cadena (length en inglés es largo):

```
str_length(c("a", "Manipulemos cadenas con R", NA, 2))
[1]  1 25 NA  1
```

- Combinar cadenas:
  Para combinar dos o más cadenas utilizamos la función str_c():

```
str_c("Ho", "la")
[1] "Hola"
str_c("Ponemos", "todo", "junto")
[1] "Ponemostodojunto"
```

  Podemos usar el argumento sep para separarlas:

```
str_c("Ponemos", "todo", "junto", sep = "_")
[1] "Ponemos_todo_junto"
```

  En estas funciones también se aprecia la vectorización y el recicling, ya que str_c() automáticamente recicla los vectores más cortos hasta alcanzar la extensión del más largo:

```
str_c("prefijo-", c("a", "b", "c"), "-sufijo")
[1] "prefijo-a-sufijo" "prefijo-b-sufijo" "prefijo-c-sufijo"
```

  Para colapsar un vector de cadenas en una sola, utiliza el argumento collapse:

```
str_c(c("x", "y", "z"), collapse = ", ")
[1] "x, y, z"
```

- Dividir cadenas:

  Puedes extraer partes de una cadena utilizando str_sub(). Al igual que la cadena, str_sub() tiene como argumentos start (inicio) y end (fin), que indican la posición (inclusiva) del subconjunto que se quiere extraer:

```
x <- c("Huelva", "Madrid", "Guadalajara")
str_sub(x, 1, 3)
[1] "Hue" "Mad" "Gua"

str_sub(x, -4, -1)
[1] "elva" "drid" "jara"
```

- Locales:

  Existen otro tipo de modificaciones que podemos hacer sobre nuestra cadena de caracteres como por ejemplo ordenarla alfabéticamente, o bien cambiar mayúsculas a minúsculas etc.

  Las funciones str_to_lower() y str_to_upper() nos sirven para cambiar mayúsculas o minúsculas:

```
str_to_upper(c("a", "o"))
[1] "A" "O"

str_to_lower(str_sub(x, 1, 1))
[1] "h" "m" "g"
```

  Por otro lado, las funciones str_sort() y str_order() nos permiten ordenar nuestro vector de caracteres:

```
str_sort(x)
[1] "Guadalajara" "Huelva"      "Madrid"

str_sort(x, decreasing = TRUE)
[1] "Madrid"      "Huelva"      "Guadalajara"

str_order(x, decreasing = TRUE)
[1] 2 1 3
```

La función paste()

Una de las principales funciones sobre cadenas de caracteres es la función paste(). Su sintaxis es,

paste (..., sep = " ", collapse = NULL)

paste (..., collapse = NULL)

- **. . .** uno o más objetos de R, que serán convertidos a vectores carácter (si no lo son)
- **sep** carácter que servirá para separar las cadenas. No está permitido NA.
- **collapse** fusiona todas las cadenas para crear una única cadena final. El valor del argumento collapse es lo que se inserta entre los elementos

Ejemplos:

```
paste(1:12)          # Convierte números en caracteres
[1] "1" "2" "3" "4" "5" "6" "7" "8" "9" "10" "11" "12"

paste("A",1:6, sep="")       # Une cadenas
[1] "A1" "A2" "A3" "A4" "A5" "A6"

paste("Today is", date())
[1] "Today is Thu Jun 18 10:38:57 2020"

paste(c("Mañana","será", "un","día","histórico"), collapse='--')
[1] "Mañana--será--un--día--histórico"
```

Coerción de tipos

La mayoría de las funciones producen un error cuando el tipo de datos que esperan no coincide con los que especificamos en los argumentos. Ante un error de este tipo disponemos de:

• funciones de comprobación

• funciones de coerción

Las funciones de comprobación chequean si un objeto es de un tipo determinado. Las funciones de comprobación comienzan por la partícula is.*** seguida del tipo que se quiere comprobar. El resultado de este tipo de funciones es un valor lógico TRUE o FALSE.

Las funciones de coerción fuerzan a cambiar el tipo del objeto con el que se está trabajando. Las funciones de coerción comienzan por la partícula as.*** seguida del tipo al que se quiere

transformar. No siempre se puede realizar la transformación, para poder realizar una trasformación es necesario que no se den incompatibilidad entre el tipo de datos origen y destino. La siguiente tabla muestra los tipos más importantes que se pueden comprobar o forzar:

Tipo	Comprobación	Coerción
array	is.array()	as.array()
character	is.character()	as.character()
complex	is.complex()	as.complex()
double	is.double()	as.double()
factor	is.factor()	as.factor()
integer	is.integer()	as.integer()
list	is.list()	as.list()
logical	is.logical()	as.logical()
matrix	is.matrix()	as.matrix()
NA	is.na()	-
NaN	is.nan()	-
NULL	is.null()	as.null()
numeric	is.numeric()	as.numeric()
ts	is.ts()	as.ts()
vector	is.vector()	as.vector()

Funciones de ordenación de vectores

Las dos funciones para ordenar vectores son:

order(x, decreasing=FALSE), #devuelve las posiciones del vector ordenadas según su contenido

order(x,y), #ordena los valores de x y, en caso de empate, utiliza los valores de y en las posiciones correspondientes.

sort (x, decreasing=FALSE), #devuelve el contenido del vector ordenado

rank(x), #devuelve el orden de cada posicion segun su contenido

Ejemplo:

Orden de un vector

```
x <- c(20,80.5,27.9,50,0)    # Vector
order (x, decreasing=FALSE) #devuelve los índices de las posiciones del vector ordenadas según su contenido
[1] 5 1 3 4 2
sort (x, decreasing=FALSE) #devuelve el contenido ordenado del vector
[1] 0.0 20.0 27.9 50.0 80.5
```

Otras Funciones sobre vectores

Nombre	Descripción
length()	longitud
sum()	suma de los valores
prod()	producto de los valores
cumsum(), cumprod()	suma y producto acumulados
max(), min()	máximo y mínimo de los valores
cummax(), cummin()	máximo y mínimo acumulados
sort()	ordena el vector
order()	devuelve los índices de las posiciones del vector ordenado
diff()	calcula la diferencia entre los valores
which(v==2)	devuelve los índices del vector que cumplen la condición
which.max()	índice del elemento mayor
which.min()	índice del elemento menor
range()	valor del mínimo y el máximo
mean()	valor promedio de los elementos
median()	valor de la mediana de los elementos
round(x,n)	redondea los eleemntos de x a n decimales
rank()	rango de los elementos
unique()	vector de los valores únicos del vector
all()	todos los elementos cumplen la condición
any()	algún elemento cumple la condición

¿Practicamos un poco?

1.- Construya los siguientes vectores,

- [2 3 4 5 6 7]
- [7 6 5 4 3 2]
- [2 5 8 11 14]

2.- Dado el vector siguiente, c(1:6,NA,NA,9:12), se pide:

- calcula en que posición están los valores perdidos. Pista utiliza la función which().
- sustituye los NA por el valor 0
- calcula la media del vector original y del trasformado. ¿Son iguales?

3.- Dados los vectores x=(1,2,6,2,23,8), y=(2,4,7,12,8,17), z=(6,8,4,12,14,5). Observa el resultado de max(x,y,z) y de pmax(x,y,z). ¿Qué hace cada función?

4.- Se quiere extraer los elementos ubicados en todas las posiciones múltiplo de 25 de un vector de longitud 1000 proveniente de una distribución normal con media igual a 100 y varianza igual a 100.

Nota: Utiliza la siguiente expresión para obtener el vector proveniente de una normal con media igual a 100 y varianza igual a 100

x <-rnorm(1000,100,10).

Lee la ayuda de la función rnorm() (?rnorm)

5.- Utilizado el vector x del ejercicio anterior, se quiere encontrar el elemento de x más cercano a 108.0

6.- Describe con tus propias palabras la diferencia entre los argumentos sep y collapse de la función str_c().

7.- Utiliza str_length() y str_sub() para extraer el caracter del medio de una cadena. ¿Qué harías si el número de caracteres es par?

8.- ¿Qué hace str_wrap()? (wrap = envolver) ¿Cuándo podrías querer utilizarla?

9.- ¿Qué hace str_trim()? (trim = recortar) ¿Cuál es el opuesto de str_trim()?

Solucionario

1
```
vector1 <- c(2,3,4,5,6,7)
vector1 <- 2:7

vector2 <- c(7,6,5,4,3,2)
vector2 <- 7:2

vector3 <- c(2,5,8,11,14)
vector3 <- seq(2,14,3)
```

2
```
v <- c(1:6,NA,NA,9:12)
which(is.na(v))
vmod <- v
vmod[which(is.na(vmod))] <- 0
mean(v)
mean(vmod)
```

3
```
x <- c(1,2,6,2,23,8)
y <- c(2,4,7,12,8,17)
z <- c(6,8,4,12,14,5)

max(x,y,z)
pmax(x,y,z)
```
max nos devuelve el valor máximo encontrado en cualquiera de los tres vectores. pmax nos devuelve el valor máximo de cada posición de las de los 3 vectores

4
```
x <-rnorm(1000,100,10)
seq <- seq(25,1000,25)
x[seq]
```

5
```
x[which.min(abs(x-108))]
```

6
```
library(tidyverse)
v1 <- c("Hola", "Adiós", "Buen día")
v2 <- c("Pepe", "Jaime", "Luis")

str_c(v1, v2, sep = "_")
```
sep se refiere al caracter que separá los elementos de la misma posición

```
str_c(v1, v2, collapse = "_")
```
collapse se refiere al caracter que unirá los elementos de las distintas posiciones en una sola

7
```
cadena <- "movil"
str_sub(cadena, round(str_length(cadena)/2)+1, round(str_length(cadena)/2)+1)
```

8
str_wrap es útil para formatear un vector de tipo caracter en cuanto a anchura de caracteres, sangrado etc.

9
str_trim elimina los espacios en blanco que pueda haber al principio o final de una cadena. Su función opuesta es str_pad