

Estructura de datos en R

Welton Vieira dos Santos

9/2/2020

Vectores

Un vector es una secuencia ordenada de datos. R dispone de muchos tipos de datos, por ejemplo:

- **logical:** Lógicos (TRUE o FALSE)
- **integer:** Números enteros, \mathbb{Z}
- **numeric:** Números reales, \mathbb{R}
- **complex:** Números complejos, \mathbb{C}
- **character:** Strings o palabras.

En los vectores de R, todos sus objetos han de ser del mismo tipo, es decir, todos números, todos palabras, etc. . .

Cuando queramos usar vectores formados por objetos de diferentes tipos, tendremos que usar **listas generalizadas**, *lists* que se verá más adelante.

Básico

- **c():** Utilizado para definir un vector.

```
c(1,2,3)
```

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

- **scan():** Utilizado para definir un vector a través de un scaneo de una variable, archivo, enlaces de internet o manualmente. Se va insertando datos y con double intro sale de interfaz de inseción del scan.

```
scan()
```

- **fix():** Utilizado para modificar visualmente el vector *x*.
- **rep(a, n):** Utilizado para definir un vector constante que contiene el dato *a* repetido *n* veces.

```
rep("Mates",7)
```

```
## [1] "Mates" "Mates" "Mates" "Mates" "Mates" "Mates" "Mates"
```

```
rep(c(1:10),3)
```

```
## [1] 1 2 3 4 5 6 7 8 9 10 1 2 3 4 5 6 7 8 9 10 1 2 3 4 5
## [26] 6 7 8 9 10
```

Progresiones aritméticas y secuencias

Una progresión aritmética es una secuencia de números tales que la **diferencia**, d , de cualquier par de términos sucesivos de la secuencia es constante.

$$a_n = a_1 + (n - 1) \cdot d$$

- **seq(a, b, by=d)**: Para generar una progresión aritmética de diferencia d que empieza en a hasta llegar a b .

```
x <- seq(5, 60, by = 5)
x
```

```
[1] 5 10 15 20 25 30 35 40 45 50 55 60
```

```
x <- seq(5, 60, by = 3.5)
x
```

```
[1] 5.0 8.5 12.0 15.5 19.0 22.5 26.0 29.5 33.0 36.5 40.0 43.5 47.0 50.5 54.0
[16] 57.5
```

```
x <- seq(60, 5, by = -5)
x
```

```
[1] 60 55 50 45 40 35 30 25 20 15 10 5
```

- **seq(a, b, length.out = n)**: Define progresión geométrica aritmética de longitud n que va de a a b con diferencia d . Por tanto $d = (b - a)/(n - 1)$

```
x <- seq(1, 15, length.out = 5)
x
```

```
[1] 1.0 4.5 8.0 11.5 15.0
```

```
x <- seq(1, 150, length.out = 3)
x
```

```
[1] 1.0 75.5 150.0
```

- **seq(a, by = d, length.out = n)**: Define progresión geométrica aritmética de longitud n y diferencia d que empieza en a .

```
x <- seq(4, length.out = 7, by = 3)
x
```

```
[1] 4 7 10 13 16 19 22
```

```
x <- seq(50, length.out = 3, by = 50)
x
```

```
[1] 50 100 150
```

- **1:20**: Define una secuencia de números enteros (\mathbb{Z}) consecutivos entre números a y b .

```
# Vector de 1 a 5
1:5
```

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

```
# Vector de -2 a 5
-2:5
```

```
[1] -2 -1 0 1 2 3 4 5
```

```
# Vector de -2 a 5
-(2:5)

[1] -2 -3 -4 -5

# Vector de 30 a -2
30:-2

[1] 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6
[26] 5 4 3 2 1 0 -1 -2
```

Concatenar vectores

Para unir dos vectores con R es muy fácil.

```
# Creando un vector de 1 a 7
vector1 <- seq(1,7)
vector1

[1] 1 2 3 4 5 6 7

# Creando un vector de 1 a 10, pero dando saltos de 2 en 2.
vector2 <- seq(1,10, by = 2)
vector2

[1] 1 3 5 7 9

# Concatenando en un sólo vector los dos vectores anteriores.
vectortotal <- c(vector1, vector2)
vectortotal

[1] 1 2 3 4 5 6 7 1 3 5 7 9

# Concatenar un vector de pi con un vector de de 5 a 10 y tambien con el número -7
c(rep(pi, 5), 5:10, -7)

[1] 3.141593 3.141593 3.141593 3.141593 3.141593 5.000000 6.000000
[8] 7.000000 8.000000 9.000000 10.000000 -7.000000
```

Funciones

Cuando queremos aplicar una función a cada uno de los elementos de un vector de datos, la función **sapply** nos ahorra tener que programar con bucles en R:

- **sapply(nombre_de_vector, FUN = nombre_de_funcion))**: Para aplicar dicha función a todos los elementos del vector.

```
x <- c(1:10)
# Aplicando el resultado de una función de callback a cada uno de los elementos
# del vector x.
sapply(x, FUN = function(elemento){
  sqrt(elemento)
})

[1] 1.000000 1.414214 1.732051 2.000000 2.236068 2.449490 2.645751 2.828427
[9] 3.000000 3.162278
```

- **sqrt(x)**: Para calcular un nuevo vector con las raíces cuadradas de cada uno de los elementos del vector *x*.

```
# Para aplicar la raiz en cada uno de los elementos del vector es muy facil con R
vector <- c(1:10)
vector
```

```
[1] 1 2 3 4 5 6 7 8 9 10
```

```
sqrt(vector)
```

```
[1] 1.000000 1.414214 1.732051 2.000000 2.236068 2.449490 2.645751 2.828427
[9] 3.000000 3.162278
```

Dado un vector de datos x podemos calcular muchas medidas estadísticas acerca del mismo:

- **length(x)**: Calcula la longitud del vector x .

```
vector <- c(1:10)
vector
```

```
[1] 1 2 3 4 5 6 7 8 9 10
```

```
# calculando la longitud o el número de elementos del vector.
length(vector)
```

```
[1] 10
```

- **max(x)**: Calcula el máximo valor del vector x .

```
vector <- c(1:10)
vector
```

```
[1] 1 2 3 4 5 6 7 8 9 10
```

```
# Calculando el mayor valor almacenado en el vector.
max(vector)
```

```
[1] 10
```

- **min(x)**: Calcula el menor valor del vector x .

```
vector <- c(1:10)
vector
```

```
[1] 1 2 3 4 5 6 7 8 9 10
```

```
# Calculando el valor del menor elemento del vector.
min(vector)
```

```
[1] 1
```

- **sum(x)**: Calcula el sumatorio ($\sum_{i=1}^n x_i$) de todos los elementos del vector x .

```
vector <- c(1:10)
vector
```

```
[1] 1 2 3 4 5 6 7 8 9 10
```

```
# Calculando la sumatorio del vector.
sum(vector)
```

```
[1] 55
```

- **prod(x)**: Calcula el producto ($\prod_{i=1}^n x_i$) de todos los elementos del vector x .

```
vector <- c(1:10)
vector
```

```
[1] 1 2 3 4 5 6 7 8 9 10
```

```
# Calculando el producto de todos los elementos del vector.  
prod(vector)
```

```
[1] 3628800
```

- **mean(x)**: Calcula la media aritmética ($\frac{1}{n} \cdot \sum_{i=1}^n x_i$) de los elementos del vector x .

```
vector <- c(1:10)  
vector
```

```
[1] 1 2 3 4 5 6 7 8 9 10
```

```
# Calculando la media aritmética de los elementos del vector.  
mean(vector)
```

```
[1] 5.5
```

- **diff(x)**: Calcula el vector formado por las diferencias sucesivas entre entradas del vector original x , es decir, la diferencia que existe entre un elemento y otro del vector.

```
vector <- c(1:10)  
vector
```

```
[1] 1 2 3 4 5 6 7 8 9 10
```

```
# Calculando el diferencial entre los elementos del vector.  
diff(vector)
```

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

```
# Calculando las diferencias de las sumas acumuladas  
diff(cumsum(vector))
```

```
[1] 2 3 4 5 6 7 8 9 10
```

```
# Calculando las diferencias de los productos acumulados  
diff(cumprod(vector))
```

```
[1] 1 4 18 96 600 4320 35280 322560 3265920
```

- **cumsum(x)**: Calcula el vector formado por las sumas acumuladas de las entradas del vector original x .
 - Permite definir sucesiones descritas mediante sumatorios.
 - Cada entrada de **cumsum(x)** es la suma de las entradas de x hasta su posición.

```
vector <- c(1:10)  
vector
```

```
[1] 1 2 3 4 5 6 7 8 9 10
```

```
# Calculando el sumatorio acumulado de todos los elementos del vector.  
cumsum(vector)
```

```
[1] 1 3 6 10 15 21 28 36 45 55
```

- **cummin(x)**: Calcula el vector formado por los mínimos valores de las entradas del vector original x , es decir, va almacenando los valores que van encontrando desde que ese valor no sea mayor que el siguiente

```
vector <- c(1:10)  
vector
```

```
[1] 1 2 3 4 5 6 7 8 9 10
```

```
# Calculando el mínimo acumulado de todos los elementos del vector.
cummin(vector)
```

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

- **cummax(x)**: Calcula el vector formado por los máximos valores de las entradas del vector original x , es decir, va almacenando los valores que van encontrando desde que ese valor no sea menor que el siguiente

```
vector <- c(1:10)
vector
```

```
[1] 1 2 3 4 5 6 7 8 9 10
```

```
# Calculando el máximo acumulado de todos los elementos del vector.
cummax(vector)
```

```
[1] 1 2 3 4 5 6 7 8 9 10
```

- **cumprod(x)**: Calcula el vector formado por los productos acumulados de las entradas del vector original x .

```
vector <- c(1:10)
vector
```

```
[1] 1 2 3 4 5 6 7 8 9 10
```

```
# Calculando el producto acumulado de todos los elementos del vector.
cumprod(vector)
```

```
[1]      1      2      6     24    120    720   5040  40320 362880
[10] 3628800
```

Orden

- **sort(x)**: Ordena el vector en orden natural de los objetos que lo forman: el orden numérico creciente, orden alfabético, etc...

```
v = c(1,7,5,2,4,6,3)
# ordenando los elementos del vector en orden natural.
sort(v)
```

```
[1] 1 2 3 4 5 6 7
```

```
# Ordenando los elementos del vector en orden decreciente.
sort(v, decreasing = TRUE)
```

```
[1] 7 6 5 4 3 2 1
```

- **rev(x)**: Invierte el orden de los elementos del vector x .

```
v = c(1,7,5,2,4,6,3)
# Inviertiendo los elementos del vector.
rev(v)
```

```
[1] 3 6 4 2 5 7 1
```