# Note riguardo R

luciano de falco alfano

07 ott 2022

**Sommario**

Osservazioni riguardo l'ambiente R: "The R Project for Statistical Computing".

## 1 appunti

### 1.1 cos'è

*R* è un ambiente di calcolo originariamente sviluppato per effettuare valutazioni statistiche. Per questo motivo è basato sul *calcolo vettoriale*. Alla base di R, *tutti i dati* sono espressi tramite vettori.

### 1.2 operazioni

Le operazioni di base sono:

- *scrittura* (assegnazione) alla variabile di nome `var`: `var <- espressione`
- *lettura* dalla variabile di nome `var`: `var`

### 1.3 dati semplici

Alcuni *dati semplici* sono espressi con notazioni semplificate:

- *numeri*: interi o a virgola mobile;
  - *interi*, esprimibili in base 10 (ad es. `122`), base 16 (ad es. `0x4E`); [1];
  - *virgola mobile* esprimibili in notazione classica (es. `10.45`) o in notazione esponenziale (es. `1.045e1` che significa $1.045 \cdot 10^1$).
- *testi*: serie di caratteri circondati da virgolette (`"`) o da apici (`'`) [2];
- *valori logici*: `TRUE`, `FALSE`.
- *speciali*: `NULL` e `NA`.

### 1.4 vettori

Un *vettore* [3] è un insieme *ordinato* [4], *monodimensionale* [5], di elementi *omogenei* [6].

[1] in alcune versioni vecchie di R possono essere *semplici* (32 bit) o *doppi* (64 bit); nelle versioni nuove su sistemi operativi a 64 bit, tutto lavora a 64 bit.

[2] attenzione al fatto che una stringa di testo è un singolo elemento di un vettore, non un vettore di caratteri.

[3] è la struttura dati di base di R, utilizzato anche per esprimere i dati semplici.

[4] gli elementi hanno posizioni fisse.

[5] la posizione di un elemento è espressa da un singolo numero: l'indice.

[6] tutti dello stesso tipo.

*Creazione* di un vettore: `c(1, 2, 4, 8, 16)` crea il vettore di 5 elementi contenente i numeri 1, 2, ... 16.

Un vettore con numeri consecutivi crescenti o decrescenti, che inizia da `n1` e termina ad `n2` inclusi, si può creare con: `n1:n2`. ad es. `1:10` crea il vettore di 10 elementi con i numeri 1, 2, ... 10.

*Assegnazione* di un vettore: `v <- c(1, 2, 4, 8, 16)` crea il vettore e lo assegna alla variabile di nome `v`.

La *lettura* degli elementi di un vettore si ottiene utilizzando un vettore che contiene gli indici degli elementi che si vogliono leggere. Ad es. se vogliamo leggere gli elementi alle posizioni 1, 3, 5 possiamo scrivere:

```
v <- c(1,2,4,8,16) # crea e assegna il vett. da leggere
v[c(1,3,5)]        # crea il vett.con le pos. da leggere,
                   #    lo usa su v
```

La *scrittura* degli elementi di un vettore usa la stessa logica della lettura, tramite l'operatore di assegnazione. Vi deve essere armonia tra numero di elementi da scrivere e n.ro di elementi nel vettore degli indici:

```
v <- c(1,2,4,8,16)  # crea e assegna il vett. da leggere
ndx <- c(1,3,5)     # crea il vett.con le pos. da scrivere
v[ndx] <- c(32, 64, 128) # modifica le pos. 1,3,5 in v
```

Per conoscere la *lunghezza* di un vettore: `length(vettore)`.

## 1.5 vettori logici

Sono vettori formati dai soli valori `TRUE` e `FALSE`.

Sono utili per filtrare i vettori, utilizzandoli come indici per la lettura. In questo caso gli elementi in corrispondenza dei valori `FALSE` sono soppressi. Ad es.

```
v <- c(1,2,4,8,16) # crea e assegna il vett. da leggere
ndx <- c(TRUE, FALSE, FALSE, FALSE, TRUE)
v[ndx]             # restituisce c(1, 16)
```

Si possono creare utilizzando operazioni di confronto ad es.

```
v <- c(1,2,4,8,16) # crea e assegna il vett. da leggere
ndx <- v > 4       # c(FALSE, FALSE, FALSE, TRUE, TRUE)
```

In tal modo è possibile manipolare logicamente il vettore sorgente. Ad es. volendo limitare a 4 il valore massimo del vettore precedente:

```
v <- c(1,2,4,8,16) # crea e assegna il vett. da leggere
ndx <- v > 4       # c(FALSE, FALSE, FALSE, TRUE, TRUE)
v[ndx] <- 4        # v==c(1,2,4,4,4)
```

Su questi vettori è possibile effettuare le operazioni booleane `and` (operatore `&`), `or` inclusivo (operatore |), `not` (operatore `!`).

## 1.6 matrici

Una *matrice* è un vettore [7] organizzato per righe e colonne. La priorità (modificabile) dell'assegnazione dei valori in creazione è per colonne.

Per la *creazione* si usa la funzione
`matrix(valori, nrow=num_righe, ncol=num_colonne)` dove `valori` è il vettore con i valori da assegnare alla matrice, `nrow` è il numero di righe della matrice, `ncol` è il numero di colonne della matrice [8]. Ad es. `matrix(1:10, nrow=2)` assegna i numeri da 1 a 10 ad una matrice di 2 righe, e 5 colonne dedotte da R.

Per la *lettura* si usa la notazione `m[righe, colonne]` dove `righe` è il vettore con le posizioni delle righe da accedere, e `colonne` è il vettore con le posizioni delle colonne da accedere. Le posizioni cui si accede sono gli elementi negli incroci tra le righe e le colonne indicate. Ad es. per accedere la riga `1`, colonne `1, 3, 5` possiamo scrivere:

```
M <- matrix(1:10, nrow=2) # crea e assegna matr.da leggere
                          # crea:
#     [,1] [,2] [,3] [,4] [,5]
#[1,]    1    3    5    7    9
#[2,]    2    4    6    8   10
ndxrow <- 1         # crea il vett.con le pos.righe da leggere
ndxcol <- c(1,3,5)# crea il vett.con le pos.col. da leggere
M[ndxrow,ndxcol]  # legge; rende:
#[1] 1 5 9
```

Esistono le notazioni `m[righe,]` che accede alle righe indicate per tutte le colonne esistenti, e la `m[,colonne]` che accede alle colonne indicate per tutte le righe esistenti.

Per la *scrittura* si usa la notazione `m[righe, colonne] <-valori` dove `valori` è il vettore con i valori da assegnare agli elementi agli incroci di `righe` e `colonne`. Ad es. per assegnare la riga `1`, colonne `1, 3, 5` possiamo scrivere:

```
M <- matrix(1:10, nrow=2) # crea e assegna matr.da modif.
                          # crea:
#     [,1] [,2] [,3] [,4] [,5]
#[1,]    1    3    5    7    9
#[2,]    2    4    6    8   10
ndxrow <- 1         # crea il vett.con le pos.righe da scrivere
ndxcol <- c(1,3,5)# crea il vett.con le pos.col. da scrivere
M[ndxrow,ndxcol] <- c(32, 64, 128) # scrive; M modificata:
#     [,1] [,2] [,3] [,4] [,5]
#[1,]   32    3   64    7  128
#[2,]    2    4    6    8   10
```

## 1.7 cbind, rbind

Queste funzioni legano per colonna (`cbind`) o per riga (`rbind`) più vettori o matrici, restituendo una matrice.

[7] quindi vale quanto detto per i vettori: elementi tutti dello stesso tipo.

[8] deve essere: $nrow \cdot ncol == length(valori)$. uno dei parametri `nrow`, `ncol` può essere omesso. In tal caso R lo calcola come modulo tra `length(valori)` e il parametro conosciuto.

## 1.8 liste

Una lista è un insieme *ordinato* [9], *monodimensionale* [10], di elementi [11].

Nel caso delle liste, ogni singolo elemento è una lista. Di conseguenza:

- un elemento di una lista può essere a sua volta una lista;
- quando si legge un elemento con l'espressione `l[indice]` si ottiene in uscita una lista; per ottenere l'elemento è necessario replicare l'operatore per l'accesso, ovvero utilizzare `l[[indice]]`.

La *creazione* avviene con: `list(el1, el2, ...)` dove `el1, el2, ...` è un elenco di dati anche disomogeneo. Ad es. `list(1, "due", 3)`.

La *lettura* avviene con `l[[indice]]`. Ad esempio:

```
l <- list(1, "due", 3)
class(l[2]) # restituisce "list"
l[[2]]      # restituisce "due"
```

## 1.9 nomi

Le posizioni di vettori, matrici e liste, oltre gli indici, possono essere identificate anche tramite nomi.

La *assegnazione del nome* può avvenire al momento della creazione, oppure in un secondo momento. Ad esempio `c("uno"=11, "due"=12, "tre"=13)` crea un vettore in cui l'elemento alla posizione `[1]` ha nome `"uno"`, ... Oppure si può:

```
v <- c(1, 2, 3)
names(v) <- c("uno", "due", "tre")
v[c("due","tre")] # restituisce c(2,3)
```

Nel caso di matrici:

```
m <- matrix(1:6, nrow=2, ncol=3)
rownames(m) <- c("uno", "due")
colnames(m) <- c("uno", "due", "tre")
m["uno", c("due","tre")] # restituisce gli elem.agli incroci di
                         #  riga 1 con le col. 2 e 3
```

## 1.10 dataframe

Un dataframe è una matrice le cui colonne possono avere elementi di tipi diversi l'una dall'altra. R le realizza costruendo ogni colonna con una lista. Tutte le colonne devono avere la stessa lunghezza.

La *creazione* di un dataframe avviene con la funzione `data.frame(cols)` dove `cols` è l'elenco delle colonne che formano il dataframe. Ad es.:

```
temp <- c(20.37, 18.56, 18.4)
humidity <- c(88, 86, 81)
rain <- c(72, 33.9, 37.5)
month <- c("January", "February", "March")
df <- data.frame(temp,humidity,rain,month)
```

[9] gli elementi hanno posizioni fisse.
[10] anche in questo caso la posizione di un elemento è espressa da un singolo numero.
[11] in questo caso il tipo di elemento può variare al variare della posizione.

```
df # ritorna:
#   temp humidity rain    month
#1 20.37       88 72.0  January
#2 18.56       86 33.9 February
#3 18.40       81 37.5    March
```

Per *leggere una colonna* di un dataframe si può usare:

- il nome della colonna, con la forma: `df$temp`;
- l'indice della colonna, con la forma: `df[,1]`;

oppure un gruppo di colonne, ad es. `temp` e `rain`:

- `df[,c("temp","rain")]`;
- oppure `df[,c(1,3)]`;

Per *leggere una riga* di un dataframe si può usare: `df[1,]`. O un gruppo di righe: `df[c(1,3),]`.

La flessibilità di elaborazione dei dataframe è enorme e trascende lo scopo di questi appunti. È possibile aggiungere o togliere righe e/o colonne. O selezionare sottoinsiemi di righe/colonne in base a criteri di selezione. Ordinare, effettuare calcolo vettoriale [12], e così via.

## 1.11 aritmetica dei vettori

L'aritmetica vettoriale è orientata ai singoli elementi dei vettori.

Sono possibli sia operazioni scalari, che vettoriali.

Ad esempio, per le scalari:

```
a <- c(1, 3, 5, 7)
a + 2 # somma; ritorna:
#[1] 3 5 7 9
a - 2 #sottrazione; ritorna:
#[1] -1  1  3  5
a * 5 # moltiplicazione; ritorna:
#[1]  5 15 25 35
a / 2 # divisione; ritorna:
#[1] 0.5 1.5 2.5 3.5
a ^ 2 # potenza; ritorna:
#[1]  1  9 25 49
```

Esempi di operazioni tra vettori (stesse dimensioni):

```
a <- c(1, 3, 5, 7)
b <- c(1, 2, 4, 8)
a + b       # somma
#[1]  2  5  9 15
a * b       # prodotto (elemento per elemento)
#[1]  1  6 20 56
```

Nel caso di vettori con dimensioni diverse, il vettore più corto viene replicato finché non raggiunge la lunghezza del più lungo[13]. Dopo di che si effettua l'operazione. Ad esempio:

[12] Ovvero applicare una elaborazione in parallelo a tutti gli elementi di una riga/colonna.

[13] Questa viene chiamata *regola del riciclo*.

```
a <- c(1, 3, 5)
b <- c(1, 2, 4, 8)
a + b       # somma c(1, 3, 5, 1) + c(1, 2, 4, 8)
#[1] 2 5 9 9
#Warning message:
#In a + b: longer object length is not a multiple of shorter object length
```

## 1.12 aritmetica delle matrici

Anche per le matrici l'aritmetica è basata sui singoli elementi. E vale sia per le operazioni scalari che tra matrici. Attenzione al prodotto: *se indicato con* `*` *è elemento per elemento*; non è il prodotto matriciale[14].

Esempio di prodotto per uno scalare.

```
M <- matrix(data=c(1, 1, 3, 3), nrow=2) # rende:
#     [,1] [,2]
#[1,]    1    3
#[2,]    1    3
M * 5 # moltiplicazione; ritorna:
#     [,1] [,2]
#[1,]    5   15
#[2,]    5   15
```

Esempi di operazioni tra matrici:

```
M <- matrix(data=c(1, 1, 3, 3), nrow=2) # rende:
#     [,1] [,2]
#[1,]    1    3
#[2,]    1    3
N <- matrix(data=c(1, 1, 2, 2), nrow=2) # rende:
#     [,1] [,2]
#[1,]    1    2
#[2,]    1    2
M + N # somma; ritorna:
#     [,1] [,2]
#[1,]    2    5
#[2,]    2    5
M * N # prodotto; ritorna:
#     [,1] [,2]
#[1,]    1    6
#[2,]    1    6
M %*% N # prodotto matriciale; ritorna:
#     [,1] [,2]
#[1,]    4    8
#[2,]    4    8
```

E' possibile trasporre una matrice con la funzione `t(matrice)`. Ad esempio:

```
M <- matrix(data=c(1, 1, 3, 3), nrow=2) # rende:
#     [,1] [,2]
#[1,]    1    3
```

[14] Ovvero ogni elemento del risultato è la somma dei prodotti elemento per elemento di riga per colonna.

```
#[2,]    1    3
t(M)  # trasposizione; ritorna:
#     [,1] [,2]
#[1,]    1    1
#[2,]    3    3
```

---

## 2 osservazioni

n.a.

## Riferimenti bibliografici


[1] R Project “The R Project for Statistical Computing”

[2] R Documentation “Documentation”

[3] R Coder “R Coder”