

2.3_ListeMatriciArray

Strutture Dati

- vettori
- fattori
- *liste*
- matrici
- array
- dataframe

Liste

Le liste sono strutture flessibili che possono contenere oggetti di tipo differente e di differenti dimensioni (ogni elemento può essere a sua volta una lista)

```
1 my_list = list("ciao!", 1:3, c(TRUE,FALSE),
2               list("ciao!", 1:3, c(TRUE,FALSE)))
3 str(my_list)
```

List of 4

```
$ : chr "ciao!"
$ : int [1:3] 1 2 3
$ : logi [1:2] TRUE FALSE
$ :List of 3
..$ : chr "ciao!"
..$ : int [1:3] 1 2 3
..$ : logi [1:2] TRUE FALSE
```

Liste

La lista pur essendo unidimensionale, si sviluppa in profondità

```
1 str(my_list)
```

```
List of 4
```

```
$ : chr "ciao!"
```

```
$ : int [1:3] 1 2 3
```

```
$ : logi [1:2] TRUE FALSE
```

```
$ :List of 3
```

```
..$ : chr "ciao!"
```

```
..$ : int [1:3] 1 2 3
```

```
..$ : logi [1:2] TRUE FALSE
```

Liste

Primo Livello: lista

```
1 str(my_list[2])
```

```
List of 1  
$ : int [1:3] 1 2 3
```

Secondo Livello: vettore numerico

```
1 str(my_list[[2]])
```

```
int [1:3] 1 2 3
```

Terzo Livello: numero intero

```
1 str(my_list[[2]][1])# numero intero
```

```
int 1
```

Liste

E' possibile assegnare dei nomi agli elementi della lista...

```
1 my_list_nam = list(parola = "ciao!", numeri = 1:3,  
2                   logico = c(TRUE, FALSE),  
3                   lista = list("ciao!", 1:3, c(TRUE, FALSE)))  
4 my_list_nam
```

```
$parola
```

```
[1] "ciao!"
```

```
$numeri
```

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

```
$logico
```

```
[1] TRUE FALSE
```

```
$lista
```

```
$lista[[1]]
```

```
[1] "ciao!"
```

```
$lista[[2]]
```

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

Liste

ed accerdervi chiamandoli per “nome” attraverso l’operatore **\$**

```
1 my_list_nam$parola
```

```
[1] "ciao!"
```

```
1 my_list_nam$lista #lista creata come elemento della lista
```

```
[[1]]
```

```
[1] "ciao!"
```

```
[[2]]
```

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

```
[[3]]
```

```
[1] TRUE FALSE
```

Liste - Attributi

Come per i vettori anche le liste hanno una lunghezza (**length()**) ed eventualmente dei nomi (**names()**). Il comando **str()** (struttura) è molto utile per le liste perchè fornisce una visione sulla struttura:

```
1 length(my_list_nam); names(my_list_nam)
```

```
[1] 4
```

```
[1] "parola" "numeri" "logico" "lista"
```

```
1 str(my_list_nam)
```

```
List of 4
```

```
$ parola: chr "ciao!"
```

```
$ numeri: int [1:3] 1 2 3
```

```
$ logico: logi [1:2] TRUE FALSE
```

```
$ lista :List of 3
```

```
..$ : chr "ciao!"
```

```
..$ : int [1:3] 1 2 3
```

```
..$ : logi [1:2] TRUE FALSE
```


Liste - Indicizzazione

La differenza tra le parentesi quadre riguarda il fatto se vogliamo fare un subset della lista ottenendo un'altra lista oppure se vogliamo accedere direttamente all'elemento interno:

```
1 my_list_nam[1] #ottenendo un'altra lista
```

```
$parola  
[1] "ciao!"
```

```
1 my_list_nam[[1]] # accedo all'elemento interno
```

```
[1] "ciao!"
```

Liste - Indicizzazione

Se vogliamo selezionare più elementi (quindi fare un vero e proprio subset della lista) dobbiamo sempre usare le parentesi quadre singole:

```
1 #lista composta dai primi due elementi della my_list_nam  
2 my_list_nam[1:2]
```

```
$parola  
[1] "ciao!"
```

```
$numeri  
[1] 1 2 3
```

Liste - Indicizzazione Nested

Le liste hanno una struttura unidimensionale ma che si può sviluppare in profondità. Per selezionare elementi nested si possono concatenare più parentesi:

```
1 my_list_nam[[4]]
```

```
[[1]]
```

```
[1] "ciao!"
```

```
[[2]]
```

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

```
[[3]]
```

```
[1] TRUE FALSE
```

```
1 my_list_nam[[4]][[1]]
```

```
[1] "ciao!"
```

```
1 my_list_nam[[4]][[2]]
```

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

Liste - Perché?

Sono flessibili, permettono di conservare/organizzare dati/informazioni complesse e varie. Per esempio la funzione `lm()` fornisce in output una lista contenente varie informazioni e risultati sul modello testato.

```
1 N = 30; b0 = 0; b1 = 0.3; sigma = 1
2 x = rnorm(N, 0, 1); y = b0 + b1*x + rnorm(N, 0, sigma)
3 mod = lm(y ~ x)
4 str(mod)
```

List of 12

```
$ coefficients : Named num [1:2] 0.233 0.556
  ..- attr(*, "names")= chr [1:2] "(Intercept)" "x"
$ residuals    : Named num [1:30] -0.432 0.171 -0.55 -0.118 0.454 ...
  ..- attr(*, "names")= chr [1:30] "1" "2" "3" "4" ...
$ effects      : Named num [1:30] -0.839 2.387 -0.42 -0.106 0.571 ...
  ..- attr(*, "names")= chr [1:30] "(Intercept)" "x" "" "" ...
$ rank         : int 2
$ fitted.values: Named num [1:30] 0.206 -0.143 0.878 -0.467 0.727 ...
  ..- attr(*, "names")= chr [1:30] "1" "2" "3" "4" ...
$ assign       : int [1:2] 0 1
```

```
$ qr      :List of 5
..$ qr    : num [1:30, 1:2] -5.477 0.183 0.183 0.183 0.183 ...
.. ..- attr(*, "dimnames")=List of 2
.. .. ..$ : chr [1:30] "1" "2" "3" "4" ...
```

Strutture Dati

- vettori
- fattori
- liste
- *matrici*
- array
- dataframe

Matrici

Le matrici sono una struttura dati **bidimensionale** (caratterizzate da 2 dimensioni **dim()**) dove il numero di righe rappresenta la dimensione 1 e il numero di colonne la dimensione 2.

```
1 my_mat = matrix(data = 1:10, nrow = 2, ncol = 5, byrow = FALSE)
2 my_mat
```

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

```
1 matrix(data = 1:10, nrow = 2, ncol = 5, byrow = TRUE)
```

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

Matrici - Attributi

```
1 attributes(my_mat)
```

```
$dim  
[1] 2 5
```

```
1 my_mat = matrix(data = 1:10, nrow = 2,  
2                 ncol = 5, dimnames = list(c("row1", "row2"),  
3                                           c("col1", "col1",  
4                                           "col3", "col4", "col5")))  
5 attributes(my_mat)
```

```
$dim  
[1] 2 5
```

```
$dimnames  
$dimnames[[1]]  
[1] "row1" "row2"
```

```
$dimnames[[2]]  
[1] "col1" "col1" "col3" "col4" "col5"
```


Matrici - Caratteristiche

- Possono contenere **una sola tipologia** di dati
- Essendo **bidimensionali**, abbiamo bisogno di due indici di posizione (righe e colonne) per identificare un elemento
- Possono essere viste come un **insieme** di singoli **vettori**

Matrici - Caratteristiche

Il numero di righe e colonne non deve essere lo stesso necessariamente (matrice quadrata) ma il numero di righe deve essere compatibile con il vettore data:

```
1 matrix(data = 1:10, ncol = 3, nrow = 3)
```

Warning in matrix(data = 1:10, ncol = 3, nrow = 3): data length [10] is not a sub-multiple or multiple of the number of rows [3]

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

Matrici

Cosa fa R di default?

```
1 matrix(data = 1:10, ncol = 3, nrow = 3)
```

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

```
1 matrix(data = 1:2, ncol = 3, nrow = 3)
```

	[,1]	[,2]	[,3]
[1,]	1	2	1
[2,]	2	1	2
[3,]	1	2	1

warnings: la funzione ci informa di qualcosa di potenzialmente problematico, ma (circa!!) tutto liscio

Matrici - Indicizzazione

Per identificare uno o più elementi nella matrice abbiamo bisogno di indici/e di riga e/o colonna separati da virgola, sempre con le parentesi quadre: `matrice[riga, colonna]`

```
1 my_mat
```

	col1	col1	col3	col4	col5
row1	1	3	5	7	9
row2	2	4	6	8	10

```
1 my_mat[1,1]
```

```
[1] 1
```

Matrici - Indicizzazione

E' possibile anche selezionare un'intera riga o colonna

```
1 my_mat[1,]
```

col1	col1	col3	col4	col5
1	3	5	7	9

```
1 my_mat[,1]
```

row1	row2
1	2

Matrici - Indicizzazione

Come per i vettori, anche in questo caso possiamo applicare l'indicizzazione logica:

```
1 my_mat>2
```

```
      col1 col1 col3 col4 col5  
row1 FALSE TRUE TRUE TRUE TRUE  
row2 FALSE TRUE TRUE TRUE TRUE
```

```
1 # Tutti gli elementi maggiori di due  
2 my_mat[my_mat>2]
```

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

Vettori e Matrici

I vettori si creano attraverso la funzione `c()` e possono essere concatenati tra loro sempre attraverso la stessa funzione:

```
1 my_vect1 = c(1:4); my_vect2 = c(5:10)
2 my_vect12 = c(my_vect1,my_vect2)
3 my_vect12
```

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

Matrici

Similarmente, le matrici possono essere unite tra loro attraverso i comandi **`cbind()`** e **`rowbind()`** :

```
1 my_mat = matrix(1:4,2,2)
2
3 cbind(my_mat, my_mat)
```

	[,1]	[,2]	[,3]	[,4]
[1,]	1	3	1	3
[2,]	2	4	2	4

```
1 rbind(my_mat, my_mat)
```

	[,1]	[,2]
[1,]	1	3
[2,]	2	4
[3,]	1	3
[4,]	2	4

Operazioni con le matrici

Come per i vettori, anche alle matrici si possono applicare operazioni matematiche:

```
1 # elemet-wise  
2 my_mat*my_mat
```

```
      [,1] [,2]  
[1,]     1     9  
[2,]     4    16
```

```
1 # Prodotto matriciale  
2 my_mat%%my_mat
```

```
      [,1] [,2]  
[1,]     7    15  
[2,]    10    22
```

```
1 # (1*1 + 3*2) , (1*3 + 3*4)  
2 # (2*1 + 4*2) , (2*3 + 4*4)
```

Strutture Dati

- vettori
- fattori
- liste
- matrici
- *array*
- dataframe

Array

Gli array sono degli oggetti n-dimensional (es., se la matrice è un quadrato un array è un cubo). Valgono le stesse proprietà della matrice scalate alle n dimensioni

```
1 my_array = array(1:6, dim = c(2,3,2)) # esempio tridimensionale
2 my_array
```

, , 1

	[,1]	[,2]	[,3]
[1,]	1	3	5
[2,]	2	4	6

, , 2

	[,1]	[,2]	[,3]
[1,]	1	3	5
[2,]	2	4	6

Array - Indicizzazione

L'indicizzazione avviene allo stesso modo delle matrici aggiungendo una dimensione

```
1 my_array
```

```
, , 1
```

	[,1]	[,2]	[,3]
[1,]	1	3	5
[2,]	2	4	6

```
, , 2
```

	[,1]	[,2]	[,3]
[1,]	1	3	5
[2,]	2	4	6

```
1 my_array[1,1,1] # prima riga, prima colonna, prima "fetta"
```

```
[1] 1
```

```
1 my_array[1,2,1] # prima riga, seconda colonna, prima "fetta"
```

```
[1] 3
```

Facciamo un po' di pratica!

Aprirete e tenete aperto questo link:

<https://etherpad.wikimedia.org/p/arca-corsoR>

