R per l'analisi statistica multivariata

Unità A: calcolo scientifico ed algebra lineare

Tommaso Rigon

Università Milano-Bicocca

Unità A

Argomenti affrontati

- A-B-C: il software **R** come calcolatrice scientifica
- Operazioni di routine: pulizia del *workspace*, simboli speciali,
- Funzioni matematiche, grafico di una funzione, operazioni logiche
- Operazioni con vettori e matrici: prodotto e decomposizioni
- Cenni alle liste

- Esercizi **R** associati: `https://tommasorigon.github.io/introR/exe/es_1.html`

Operazioni di base

- **R** può essere usato come se fosse una **calcolatrice scientifica**.

```
2 + 2
# [1] 4
4 * (3 + 5) # La somma entro parentesi viene eseguita per prima
# [1] 32
pi / 4 # Pi greco quarti
# [1] 0.7853982
```

- Per calcolare la potenza a^b si usa la stessa sintassi di una calcolatrice scientifica

```
2^5 # Sintassi alternativa: 2**5
# [1] 32
```

- Per calcolare ad esempio $\sqrt{2}$ e $\sin(\pi/4)$ si possono usare le funzioni seguenti:

```
sqrt(2)
# [1] 1.414214
sin(pi / 4)
# [1] 0.7071068
```

- **Nota**. Tutto ciò che viene scritto dopo un cancelletto (#) è considerato un **commento**.

Assegnazione di un valore

- È possibile **salvare un valore** assegnandolo ad un oggetto tramite il simbolo `<-`.

```
x <- sqrt(5) # Sintassi alternativa (sconsigliata): x = sqrt(5)
```

- **Nota**. **R** è **case sensitive**, pertanto l'oggetto `x` è diverso dall'oggetto `X`.

- Il valore contenuto in `x` può essere successivamente richiamato, modificato e salvato in un nuovo oggetto chiamato `y`.

```
y <- x + pi # ovvero pi greco + radice quadrata di 5
y
# [1] 5.377661
```

- Per **rimuovere un oggetto** dalla memoria, si usa il comando `rm`.

```
rm(x) # x non è più presente nel "workspace"
```

Pulizia del "workspace"

- È buona norma mantenere pulito il **workspace**, ovvero l'ambiente di lavoro.
- Se un oggetto non è più necessario, è possibile eliminarlo tramite il comando `rm`.
- È possibile visualizzare la lista di oggetti salvati in memoria tramite il comando seguente:

```
ls() # Nel workspace è presente l'oggetto y
# [1] "y"
```

- Pertanto, per **eliminare** tutti gli **oggetti** salvati, si può usare

```
rm(list = ls())
```

Alcune funzioni matematiche

- Supponiamo che `x` sia un numero reale, ad esempio ponendo `x <- 1/2`.

```r
x <- 1/2

exp(x) # Esponenziale e logaritmo naturale
log(x)

abs(x) # Valore assoluto
sign(x) # Funzione segno

sin(x) # Funzioni trigonometriche (seno, coseno, tangente)
cos(x)
tan(x)

asin(x) # Funzioni trigonometriche inverse
acos(x)
atan(x)
```

- **Nota**. Le funzioni di **R** si possono combinare tra loro, ad esempio `log(abs(x))`.

Ulteriori funzioni matematiche

- Supponiamo che `x` e `y` siano due **numeri reali**. Inoltre, siano `n` e `k` due **numeri naturali**.

- Si noti l'uso del "`;`" che può essere usato per separare due comandi nella stessa riga.

```
x <- 1 / 2; y <- 1 / 3 # Numeri reali
n <- 5; k <- 2 # Numeri naturali

factorial(n) # n!
choose(n, k) # Coefficiente binomiale

round(x, digits = 2) # Arrotonda x usando 2 cifre decimali
floor(x) # Arrotonda x all'intero più vicino, per difetto
ceiling(x) # Arrotonda x all'intero più vicino, per eccesso
```

- La funzione gamma $\Gamma(x) = \int_0^\infty s^{x-1} e^{-s} ds$ si calcola in **R** come segue:

```
gamma(x) # Funzione gamma
```

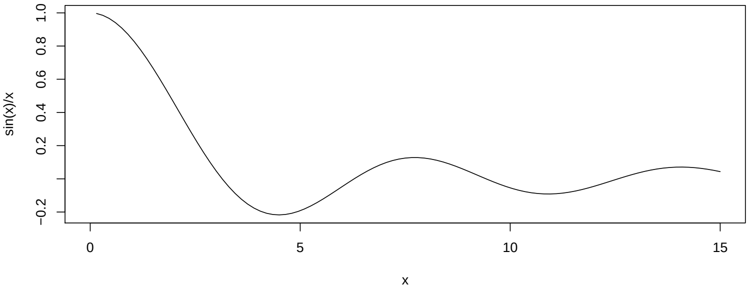
- La funzione beta $\mathcal{B}(x, y) = \int_0^1 s^{x-1}(1-s)^{y-1} ds$ si calcola in **R** come segue:

```
beta(x, y) # Funzione beta
```

Grafico di una funzione

- In **R** è possibile **disegnare** una qualsiasi **funzione** tramite il comando `curve`.
- Se ad esempio si considera la funzione
$$f(x) = \frac{\sin(x)}{x},$$
allora possiamo disegnare $f(x)$ nell'intervallo $(0, 15)$ come segue:

```
curve(sin(x) / x, from = 0, to = 15)
```



La documentazione ufficiale

- La documentazione di **R** è la principale fonte di informazioni.

- A cosa serve una funzione? Qual è la definizione dei suoi argomenti? La risposta va sempre cercata nella **documentazione ufficiale** e non in queste slide.

- Il comando "`? funzione`" apre una finestra in cui vengono descritta nel dettaglio una funzione. Esempio:

```
? log # Documentazione della funzione log
```

- **Nota riguardante l'esame**. Durante la prova d'esame è legittimo (anzi, è caldamente consigliato) consultare la documentazione.

Simboli speciali

- Numeri molto grandi (e molti piccoli) in **R** vengono rappresentati come segue

```
10^15
# [1] 1e+15
10^(-15)
# [1] 1e-15
```

- Per questioni di approssimazione numerica, quando un numero è troppo grande **R** riporta `Inf`, ovvero **infinito**. Per esempio:

```
10^1000 # Numero molto grande, anche se finito
# [1] Inf
```

- Il simbolo `NaN` significa invece "**Not a Number**" e si ottiene per esempio nel seguente caso:

```
log(-1) # Questo comando genera inoltre un avviso
# [1] NaN
```

Errori di approssimazione numerica

- È ben noto che $\sin(\pi) = 0$. Tuttavia, in **R** si ottiene un numero molto vicino a 0, ma strettamente positivo. Infatti:

```r
sin(pi)
# [1] 1.224647e-16
```

- **R** è uno strumento di **calcolo numerico** e pertanto sono sempre presenti **errori** di **approssimazione numerica**.

- Fortunatamente, nella maggior parte dei casi pratici la differenza tra 0 e 10^{-16} è irrilevante.

- In altre situazioni, errori di approssimazione numerica possono portare a conclusioni fuorvianti. Occorre quindi fare attenzione e valutare caso per caso.

- Ad ogni modo, l'approssimazione numerica potrebbe anche migliorare. Ad esempio:

```r
cos(pi)
# [1] -1
```

Operazioni logiche

- In **R** è spesso necessario verificare se una o più condizioni sono verificate o meno.

```
x <- 5
x < 0 # Il valore di x è minore di 0?
# [1] FALSE

a <- (x == -3) # Il valore di x è uguale a -3?
a
# [1] FALSE
```

- Il valore di a è un indicatore **binario** o **booleano**, ovvero può essere vero (`TRUE`) oppure falso (`FALSE`).

- Altre **funzioni logiche** disponibili (assumendo che `y` sia un numero e `b` un booleano) sono:

```
x >= y # x è maggiore o uguale a y? (Si usa "<=" per minore uguale)
x != y # x è diverso da y?
a & b # a AND b. I valori booleani a e b sono entrambi veri?
a | b # a OR b. Almeno uno tra a ed b è vero?
```

Vettori

- Un **vettore** in **R** viene definito tramite la funzione `c()`, come nel seguente esempio

```
x <- c(4, 2, 2, 8, 10)
x
# [1]  4  2  2  8 10
```

- **Nota**. Con il termine generico *vettore* in **R** **non** si fa riferimento alla nozione dell'algebra lineare ma semplicemente ad una stringa di valori consecutivi.

- Infatti il seguente oggetto è un **vettore**, in **R** nonostante l'oggetto `x` sia composto sia numeri che da lettere

```
x <- c("A", "B", 2, 8, 10)
x
# [1] "A"  "B"  "2"  "8"  "10"
```

Creazione di vettori

- Talvolta è comodo creare dei vettori i cui elementi sono dei **numeri consecutivi**

```
x <- 5:10 # Equivalente a: x <- c(5, 6, 7, 8, 9, 10)
x
# [1]  5  6  7  8  9 10
```

- Usando la stessa sintassi, è possibile creare successioni in **ordine decrescente**

```
x <- 10:-10
x
# [1]  10   9   8   7   6   5   4   3   2   1   0  -1  -2  -3  -4  -5  -6  -7  -8  -9 -10
```

- Per creare una successione di **numeri reali** si usa il comando `seq`:

```
x <- seq(from = 0, to = 1, by = 0.1)
x
# [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

- Per creare un vettore di **valori ripetuti** si usa il comando `rep`:

```
x <- rep(10, 7) # Vettore in cui il numero 10 è ripetuto 7 volte
x
# [1] 10 10 10 10 10 10 10
```

Operazioni sui vettori I

- La maggior parte delle funzioni matematiche di **R** sono **vettorizzate**. In altri termini, le funzioni agiscono su tutti gli elementi di un vettore.

```
exp(1:6) + (1:6) / 2 + 1 # Esempio 1
# [1]   4.218282   9.389056  22.585537  57.598150 151.913159 407.428793

x <- c(10, 10^2, 10^3, 10^4, 10^5, 10^6) # Esempio 2
log(x, base = 10)
# [1] 1 2 3 4 5 6

1:8 > 4 # Esempio 3
# [1] FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE
```

- Altre funzioni invece sono utili proprio nel caso in cui l'**argomento** sia un **vettore**:

```
x <- c(2, 3, 1, 3, 10, 5)
length(x) # Lunghezza del vettore
# [1] 6
sum(x) # Somma degli elementi del vettore
# [1] 24
cumsum(x) # Somme cumulate
# [1]  2  5  6  9 19 24
```

Operazioni sui vettori II

- Ulteriori semplici operazioni in cui l'argomento è un vettore sono elencate nel seguito

```
x <- c(2, 3, 1, 3, 10, 5)
prod(x) # Prodotto degli elementi del vettore
# [1] 900
cumprod(x) # Prodotti cumulati
# [1]   2   6   6  18 180 900
sort(x, decreasing = FALSE) # Vettore ordinato in ordine crescente
# [1]  1  2  3  3  5 10
min(x) # Valore minimo
# [1] 1
which.min(x) # Posizione del valore corrispondente al minimo
# [1] 3
```

- Infine, il funzionamento delle seguenti funzioni dovrebbero essere intuibile da quanto visto finora

```
max(x) # Valore massimo
which.max(x) # Posizione del valore corrispondente al massimo
range(x) # Equivalente a: c(min(x), max(x))
```

Manipolazione dei vettori

- È possibile **selezionare gli elementi** di un vettore usando le parentesi quadrate, come nei seguenti esempi

```r
# Concatenazione di vettori
x <- c(rep(pi, 2), sqrt(2), c(10, 7))

x[3] # Estrae il terzo elemento dal vettore x, ovvero sqrt(2)
# [1] 1.414214

x[c(1, 3, 5)] # Estrae il primo, il terzo ed il quinto elemento
# [1] 3.141593 1.414214 7.000000

x[-c(1, 3, 5)] # Elimina il primo, il terzo ed il quinto elemento
# [1]  3.141593 10.000000

x[x > 3.5] # Estrae gli elementi maggiori di 3.5
# [1] 10  7
```

- L'ultimo comando suggerisce che gli elementi di un vettore possono essere selezionati tramite una **condizione** relativa al vettore stesso.

Avvertenze sul calcolo vettoriale in R

- Cosa succede quando vengono sommati due vettori di dimensioni diverse?

- **Danger zone**. La maggior parte dei linguaggi di programmazione restituisce un errore. **R** invece esegue ugualmente l'operazione "allungando" il vettore più corto.

- In questo primo esempio, quantomeno **R** restituisce un **warning**.

```r
# Concatenazione di vettori
x <- 1:5
y <- 1:6
x + y # Equivalente a: c(x, x[1]) + y
# [1]  2  4  6  8 10  7
# Warning message:
# In x + y : longer object length is not a multiple of shorter object length
```

- In questo secondo esempio, invece, **R** non restituisce alcun avviso, rendendo la cosa particolarmente pericolosa

```r
x <- 1:3
y <- 1:6
x + y # Equivalente a: c(x, x) + y
# [1] 2 4 6 5 7 9
```

Matrici

- Una **matrice** **A** è una collezione di elementi $(a)_{ij}$ per $i = 1, \dots, n$ e $j = 1, \dots, m$.

- Per esempio, la matrice quadrata di dimensione 2×2

$$\mathbf{A} = \begin{pmatrix} 5 & 2 \\ 1 & 4 \end{pmatrix},$$

si può definire in **R** tramite il comando `matrix` come segue:

```
A <- matrix(c(5, 1, 2, 4), nrow = 2, ncol = 2)
A
#      [,1] [,2]
# [1,]    5    2
# [2,]    1    4
```

- È inoltre possibile elencare gli elementi **riga per riga**

```
# Definizione equivalente
A <- matrix(c(5, 2, 1, 4), nrow = 2, ncol = 2, byrow = TRUE)
```

Manipolazione di matrici

- È possibile **selezionare gli elementi** di una matrice in maniera analoga a quanto fatto con i vettori.

```
A[1, 2] # Estrazione di elemento in posizione (1,2)
A[, 2] # Estrazione seconda colonna
A[1, ] # Estrazione prima riga
```

- Alcuni **comandi di base** per manipolare le matrici sono i seguenti

```
dim(A) # Restituisce la dimensione della matrice
# [1] 2 2
a <- c(A) # Converte la matrice in un vettore
a
# [1] 5 1 2 4
```

```
diag(A) # Restituisce la diagonale della matrice
# [1] 5 4
t(A) # Calcola la matrice trasposta A'
#      [,1] [,2]
# [1,]    5    1
# [2,]    2    4
sum(A) # Somma di tutti gli elementi di A
# [1] 12
```

Operazioni sulle matrici

- Come per i vettori, le operazioni elementari (somma, prodotto, `log`, `exp`, etc.) vengono eseguite **elemento per elemento**.

```
exp(A)
#             [,1]      [,2]
# [1,] 148.413159  7.389056
# [2,]   2.718282 54.598150
```

- Siano $\mathbf{A}$ e $\mathbf{B}$ due matrici aventi lo stesso numero di colonne e definiamo

$$\mathbf{C} = \begin{pmatrix} \mathbf{A} \\ \mathbf{B} \end{pmatrix}.$$

```
B <- A # Creo una matrice B identica ad A, per semplicità
C <- rbind(A, B)
```

- In maniera analoga, siano $\mathbf{A}$ e $\mathbf{B}$ due matrici aventi lo stesso numero di righe e definiamo $\mathbf{C} = \begin{pmatrix} \mathbf{A} & \mathbf{B} \end{pmatrix}$.

```
C <- cbind(A, B)
```

Vettori riga e vettori colonna

- Una matrice con una sola colonna è un **vettore colonna**:

```
x_col <- matrix(c(1, 10, 3, 5), ncol = 1)
x_col
#      [,1]
# [1,]    1
# [2,]   10
# [3,]    3
# [4,]    5
```

- Una matrice con una sola colonna è un **vettore riga**:

```
x_row <- matrix(c(1, 10, 3, 5), nrow = 1)
x_row
#      [,1] [,2] [,3] [,4]
# [1,]    1   10    3    5
```

Vettori riga e vettori di **R**

- Nella maggior parte dei casi, il **vettore riga** `x_row` è intercambiabile col vettore `x`.

```
x_row <- matrix(c(1, 10, 3, 5), nrow = 1)
x <- c(1, 10, 3, 5) # Simile, ma non identico, a x_row
```

- Ad esempio, le funzioni `sum(x_row)` e `sum(x)` forniscono lo stesso risultato.

- Ci sono tuttavia alcune lievi distinzioni. Ad esempio:

```
dim(x_row)
# [1] 1 4
dim(x)
# NULL
```

- Il simbolo `NULL` significa **non definito**, perché non esiste la nozione di dimensione per un generico vettore **R**.

Il prodotto incrociato

- Siano $\mathbf{x}$ e $\mathbf{y}$ due vettori colonna in $\mathbb{R}^p$. Allora, il loro **prodotto incrociato** è pari a

$$\mathbf{x}^\intercal \mathbf{y} = \sum_{i=1}^{p} x_i y_i.$$

- in **R** possiamo usare il comando `crossprod`

```
x <- matrix(c(-4, 2, 6, 10, 22), ncol = 1)
y <- matrix(c(3, 2, 2, 7, 9), ncol = 1)
crossprod(x, y) # Equivalente a: sum(x * y)
#      [,1]
# [1,]  272
```

- Il comando `crossprod` funziona correttamente anche con "vettori" **R**.

- Il comando `crossprod` può essere usato anche per calcolare il seguente **prodotto tra matrici**

$$\mathbf{A}^\intercal \mathbf{B},$$

dove $\mathbf{A}$ e $\mathbf{B}$ sono due matrici di dimensioni compatibili.

Il prodotto matriciale

- In algebra lineare il **prodotto tra matrici** compatibili $\mathbf{AB}$ è chiamato prodotto righe per colonne. In **R** si usa il comando seguente

```
A <- rbind(c(1, 2, 3), c(4, 9, 2), c(2, 2, 2))
B <- rbind(c(5, 2, 5), c(3, 3, 7), c(-2, -8, 10))

A %*% B # Prodotto righe per colonne AB
#      [,1] [,2] [,3]
# [1,]    5  -16   49
# [2,]   43   19  103
# [3,]   12   -6   44
```

- **Nota**. Il comando `A * B` indica il prodotto elemento per elemento e non il prodotto righe per colonne.

- Se le matrici non sono compatibili **R** produce un errore (provateci per esercizio!)

Calcolo della matrice inversa

- Sia $\mathbf{A}$ una matrice quadrata $n \times n$ a valori reali. La sua **matrice inversa** $\mathbf{A}^{-1}$, quando esiste, è l'unica matrice tale per cui

$$\mathbf{A}\mathbf{A}^{-1} = \mathbf{A}^{-1}\mathbf{A} = I_n.$$

- Per ottenere $\mathbf{A}^{-1}$ si usa il comando `solve`.

```
A <- rbind(c(1, 2, 3), c(4, 9, 2), c(2, 2, 2))
A1 <- solve(A) # Matrice inversa di A
A1
#            [,1]        [,2]        [,3]
# [1,] -0.5833333 -0.08333333  0.95833333
# [2,]  0.1666667  0.16666667 -0.41666667
# [3,]  0.4166667 -0.08333333 -0.04166667

round(A %*% A1, digits = 5) # Operazione di controllo
#      [,1] [,2] [,3]
# [1,]    1    0    0
# [2,]    0    1    0
# [3,]    0    0    1
```

```
det(A) # Calcola il determinante della matrice A
# [1] -24
```

Ulteriori operazioni con le matrici

- Nel caso una matrice non sia invertibile, il **determinante** è pari a 0. Il comando `solve` in quel caso produce un errore.

```
# Esempio di matrice NON invertibile
A <- rbind(c(1, 2, 3), c(2, 4, 6), c(2, 2, 2))
det(A) # Deteminante pari a 0, solve(A) produce un errore
# [1] 0
```

- Ci sono numerose funzioni per la **decomposizione di matrici**, il cui output è a volte una **lista**.

- Il risultato delle seguenti funzioni è omesso.

```
A <- matrix(c(4, 1, 1, 8), ncol = 2)
chol(A) # Decomposizione di Cholesky
qr(A) # Decomposizione QR
eigen(A) # Decomposizione spettrale
```

Liste

- Una **lista** è una collezione di oggetti (numeri, vettori, matrici, altro).
- Per **salvare** o per **estrarre** un oggetto da una lista si usa il simbolo $.

```
# Creazione di una lista
new_list <- list(
  A = matrix(c(4, 1, 1, 8), ncol = 2),
  x = c(1, 2, 6, 6, 9)
)

new_list
# $A
#      [,1] [,2]
# [1,]    4    1
# [2,]    1    8
#
# $x
# [1] 1 2 6 6 9
```

Decomposizione spettrale

- Gli autovalori ed autovettori di una matrice $\boldsymbol{A}$ si ottengono tramite il comando `eigen`.
- Il risultato è una **lista**, contenente gli autovettori (`vectors`) e autovalori (`values`).

```
Spec_A <- eigen(A) # Decomposizione spettrale della matrice A
Spec_A
# eigen() decomposition
# $values
# [1] 8.236068 3.763932
#
# $vectors
#           [,1]       [,2]
# [1,] 0.2297529 -0.9732490
# [2,] 0.9732490  0.2297529
```

```
Spec_A$values # Estrazione degli autovalori
# [1] 8.236068 3.763932
```