Laboratorio di ST1 - Lezione 1

Claudia Abundo

Dipartimento di Matematica
Università degli Studi Roma Tre

Introduzione

In questa prima esercitazione si faranno i primi passi con **R**.
In particolare ci focalizzeremo sull'installazione e sul linguaggio.

Perchè R

- Il programma è distribuito gratuitamente attraverso internet sotto la forma di GNU General Public License (programma *open source*)
- La grandezza della sua comunità scientifica
- Si sta diffondendo anche in ambito aziendale come strumento operativo
- Il programma gira sotto i principali sistemi operativi (Windows, Mac OS X, Unix e nelle varie distribuzioni di Linux)

Come installare R

1. Tramite il vostro browser visitate il sito `http://www.r-project.org` e cliccate sul link CRAN situato nella sezione Download.
2. Scegliete un mirror vicino a voi, ossia un qualsiasi sito italiano
3. Cliccate su Windows 95 and later
4. Cliccate su Base.
5. Cliccate su R-2.10.1-win32.exe
6. Salvate il file eseguibile sul vostro computer.

Se volete evitare tutta questa procedura potete direttamente digitare nel vostro browser l'indirizzo

`http://rm.mirror.garr.it/mirrors/CRAN/bin/windows/base/release.htm`

Un aspetto importante da notare è che con la precedente procedura si è installato:

- il motore di calcolo
- il motore grafico del programma
- un insieme di routines che costituiscono il pacchetto base

Oltre a questo pacchetto base esistono altri pacchetti aggiuntivi. Questi pacchetti aggiuntivi sono stati “scritti” dalla comunità scientifica, ed in questo senso è importante avere una grande numero di users. Questo in altre parole significa che per la maggior parte dei modelli statistici utilizzabili esistono già le specifiche routines da lanciare in **R**.

Installazione dei pacchetti aggiuntivi

Per installare i pacchetti aggiuntivi si può seguire la seguente procedura dal menu a tendina:

1. cliccate su `Pacchetti`
2. scegliete un Mirror (consiglio: Padova)
3. cliccate su `Installa pacchetto...`
4. fate il download del pacchetto scelto

Ora apriamo **R**!

Il messaggio iniziale della *console* di **R** si conclude con il simbolo $>$ che, in gergo, viene definito *prompt* dei comandi. Questo simbolo segnala all'utente che **R** è pronto ad eseguire il comando digitato dopo di esso. Un comando di **R** è sempre concluso dal tasto Invio [Return].
Iniziamo a giocare con la sintassi dei comandi di **R** attraverso semplici esempi.

Primi passi con R

```
> 2+2
[1] 4
> 2*3
[1] 6
> 3/5
[1] 0.6
> 3^3
[1] 27
> a
Errore: oggetto "a" non trovato
> a<-4
> a
[1] 4
```

Primi passi con R - 2

```
> b
Errore: oggetto "b" non trovato
> b=5
> b
[1] 5
> c=a+b
[1] 9
```

Si puo notare come il risultato del comando sia preceduto da [1] che sta ad indicare che l'output è un vettore di lunghezza 1.
La spiegazione di questa situazione è che **R** lavora per costruzione con i vettori, quindi interpreta un numero come un vettore di dimensione 1.
Nell'ultimo esempio si vede come sia possibile creare una relazione matematica tra oggetti definiti dall'utente, in questo caso la somma tra gli oggetti a e b.

Gli operatori di assegnazione

Nel precedente esempio abbiamo visto i due modi per creare una assegnazione:

- l'operatore $<-$ ovvero il tasto minore seguito dal tasto meno senza uno spazio tra i due simboli
- l'operatore =

Tipi e oggetti

In **R** un oggetto, definito attraverso una operazione di assegnazione, può essere di tre tipi:

- un elemento numerico
- un elemento composto da una stringa di caratteri
- un elemento logico

Oggetti Numerici

Gli elementi numerici a loro volta si possono distinguere in interi, reali e complessi.
Un facile esempio è dato da:

```
a<-10
b<-66.33
c<-3+2.3i
```

Precisazione. L'insieme dei numeri rappresentabili è in realtà un sottoinsieme dei numeri naturali e reali. Un altra puntualizzazione è che i numeri reali sono approssimati attraverso un insieme finito di razionali. In **R**, comunque, non ci sono particolari problemi di approssimazione nelle applicazioni pratiche in quanto il più piccolo numero rappresentabile è dell'ordine di 10^{-300}

Gli altri tipi

Per definire un oggetto come stringa di caratteri è necessario virgolettare la stringa, ad esempio:

```
# Oggetto composto da una stringa di caratteri#

d<-"claudia"
```

Il terzo tipo è formato dagli operatori logici `TRUE` e `FALSE` o, più semplicemente, `T` e `F`. Questi operatori logici si utilizzano, in genere, all'interno di funzioni come operatori di controllo.

I Vettori

Fino ad ora attraverso l'operazione di assegnazione si sono definiti degli oggetti composti da un unico elemento. Per definire un oggetto composto da più elementi è necessario utilizzare il comando di concatenamento `c(...)`.
In pratica per definire un vettore numerico bastera inserire come argomento di tale funzione i singoli elementi numerici separati da una virgola, mentre per definire un vettore di stringhe occorre ricordarsi, inoltre, di virgolettare gli elementi, come ad esempio...

```
> a<-1
> a
[1] 1
> b<-3
> c<-c(a,b)
> c
[1] 1 3
>#Attenzione c*c fa il quadrato degli elementi#
> c*c
[1] 1 9
>#Comando per conoscere la lunghezza di un vettore#
> length(c)
[1] 2
> d<-c(c,a,b)
[1] 1 3 1 3
> e<-c("claudia","matematica","statistica")
> e
[1] "claudia"  "matematica" "statistica"
```

Un po' di accortezza va posta sul fatto che i vettori devono essere composti da elementi omogenei, ovvero appartenenti tutti allo stesso tipo.
Talvolta è necessario modificare un elemento del vettore lasciando invariato tutto il resto. In **R** l'i-esimo elemento di un vettore generico `a` è dato da: `a[i]`. Quindi per modificare questo elemento è sufficiente effettuare una nuova operazione di assegnazione, come ad esempio...

```
> a<-c(1,2,3)
> a
[1] 1 2 3
> a[2]<-77
> a
[1] 1 77 3
```

Spesso in statistica è necessario dover definire particolari vettori composti o da sequenze numeriche o solamente da uno stesso elemento. Per evitare di imputare manualmente questi vettori si possono utilizzare le funzioni `rep(...)` e `seq(...)`.

Il comando `rep(x,n)` crea un vettore ripetendo un oggetto `x` per n volte, ad esempio

```
> b<-c(1,2)
> d<-rep(b,3)
> d
[1] 1 2 1 2 1 2
> e<-rep(1,6)
> e
[1] 1 1 1 1 1 1
```

Il comando `seq(start,end, by=)` crea in un vettore una sequenza di numeri. I parametri `start` ed `end` sono rispettivamente il punto di inizio ed il punto di fine della sequenza, mentre `by` rappresenta il passo della sequenza. Giusto per dare un esempio del comando abbiamo che

```
> seq(1,15,3)
[1]  1  4  7 10 13
> 1:100
  [1]   1   2   3   4   5   6   7   8   9  10  11
 [16]  16  17  18  19  20  21  22  23  24  25  26
 [31]  31  32  33  34  35  36  37  38  39  40  41
 [46]  46  47  48  49  50  51  52  53  54  55  56
 [61]  61  62  63  64  65  66  67  68  69  70  71
 [76]  76  77  78  79  80  81  82  83  84  85  86
 [91]  91  92  93  94  95  96  97  98  99 100
```

L'ultimo comando digitato rappresenta una scorciatoia nel caso in cui il passo della sequenza sia uguale ad uno.

Le Matrici

Fino ad ora abbiamo definito dei vettori mentre ora introdurremo le matrici. Il comando è `matrix(ogg , nrow = , ncol =)`, `ogg` è un generico oggetto mentre `nrow` e `ncol` sono i parametri che permettono di fissare il numero di righe ed il numero di colonne della matrice da creare. Proviamo a descrivero il comando `matrix` da qualche esempio.

```
> m<-matrix(ncol=2,nrow=2)
> m
     [,1] [,2]
[1,]   NA   NA
[2,]   NA   NA
> m2<-matrix(0,ncol=2,nrow=2)
> m2
     [,1] [,2]
[1,]    0    0
[2,]    0    0
```

```
> m3<-matrix(1:9,ncol=3,nrow=3)
> m3
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9

m4<-matrix(1:9,ncol=3,nrow=3,byrow=T)
> m4
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
> m4[2,3]
[1] 6
```

Come si vede dal primo esempio, non avendo specificato l'oggetto con cui "riempire" la matrice, gli elementi risultano missing ovvero `NA`.
Per accedere ad un elemento di una matrice è sufficiente specificare tra parentesi quadrate la riga e la colonna dell'elemento separandole con una virgola. Se si vuole accedere alla i-esima riga di una matrice A si deve digitare `A[i,]`, analogamente per la j-esima colonna di A si deve digitare `A[,j]`.

```
> A<-matrix(1:4,ncol=2)
> A
     [,1] [,2]
[1,]    1    3
[2,]    2    4
> A[1,]
[1] 1 3
> A[1,2]
[1] 3
> A[,2]
[1] 3 4
```

Attenzione: potrebbe accadere una situazione del tipo:

```
> A<-matrix(1:5,ncol=2,nrow=2) Warning message: la lunghezza [5]
dei dati non è un sottomultiplo o un multiplo del numero di righe
[2] in matrix

> A
     [,1] [,2]
[1,]    1    3
[2,]    2    4
```

Abbiamo fatto un errore (mancata coerenza nelle dimensioni dell'oggetto rispetto a quelle della matrice) nella definizione della matrice ma questa è stata comunque creata.

Talvolta è necessario fare alcune operazioni con le matrici come:

- fusioni di due matrici
- trasposte
- inverse
- prodotti righe per colonne
- ...

I comandi `rbind(...)` e `cbind(...)`.
Il primo unisce due matrici o vettori (attenzioni alle dimensioni che potete controllare con il comando `dim`) per le righe, in pratica mette una matrice sopra l'altra.
Il secondo comando opera la fusione, invece, affiancando orizzontalmente le matrici. Due esempi chiarificheranno il tutto.

```
> a<-matrix(1:6,ncol=2)
> a
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
> b<-7:9
> b
[1] 7 8 9
> prova<-cbind(a,b)
> prova
[1,] 1 4 7
[2,] 2 5 8
[3,] 3 6 9
> c<-c(1,2)
> prova2<-rbind(a,c)
> prova2
[1,] 1 4
[2,] 2 5
[3,] 3 6
[4,] 1 2
```

Passiamo ora alle altre operazioni con le matrici.

- `t(...)` che implementa la trasposta di una matrice
- `solve(...)` codifica l'inversa
- `%*%` è l'operatore che produce il prodotto righe per colonne. Un consiglio utile è quello di effettuare il prodotto righe per colonne solamente tra matrici, cioè definire anche i vettori con il comando `matrix` per evitare ambiguità tra le dimensioni.

Help

Cominciamo dai comandi di help in linea. Con questo strano nome vengono definiti i comandi di aiuto che possono lanciati essere dal prompt. In particolare in **R** sono a disposizione dell'utente tre diversi comandi:

- help(...)
- help.search("...")
- help.start()

Con il primo comando **R** apre una nuova finestra in cui spiega la sintassi del comando, descrive gli argomenti e fornisce degli esempi su come utilizzare tale comando, ad esempio provate a digitare help(quit) o, in maniera più rapida, `?quit` e studiate la finestra che esce in pop-up. Questo comando di help viene utilizzato quando l'utente conosce il nome di un comando che intende utilizzare.

Quando non si conosce il nome di un comando da utilizzare, si può effettuare una ricerca per parole chiavi attraverso l'uso del `help.search(``...'')`. Questo comando apre una finestra (eventualmente vuota) in cui sono presenti tutti i riferimenti dove le vostra stringa di parole chiavi è presente.

Supponiamo, ad esempio, che non si conosca il comando del coseno, allora, il primo passo è quello di digitare `help.search(``cos'')`.

```
Help files with alias or concept or title matching 'cos' using
regular expression matching:
cosh(base)            Hyperbolic Functions
Trig(base)          Trigonometric Functions
Airline(Ecdat)     Cost for U.S. Airlines
Electricity(Ecdat) Cost Function for Electricity Producers, 1970
. . .
```

Nella finestra che si apre al secondo punto della lista compare un comando `Trig` (ATTENZIONE: **R** è case sensitive ovvero distingue fra maiuscole e minuscole) che si trova nel pacchetto base che contiene le funzioni trigonometriche.
A questo punto il passo successivo della ricerca è digitare `help(Trig)` scoprendo che il coseno di x è semplicemente `cos(x)`.

Il più flessibile ed intuivo ma anche il più dispendioso in termini di tempo è il comando `help.start()`. Digitando questo comando si apre una finestra di help interattiva tramite il vostro browser di default (IE, Firefox, Opera, ...). Cliccando sui vari link potete sfogliare il manuale che accompagna **R**. Questo manuale si trova sull hard disk del vostro pc (C:/Programmi/R/R-2.10.1/doc/manual/), quindi è consultabile anche quando non siete on-line.

Tutti e tre i comandi sono utilizzabili anche tramite i menu a tendina di **R** sotto la voce `Aiuto`. L'ultimo comando di help richiama, in realtà, l'accesso elettronico ai manuali che si dispongono nelle sottocartelle della cartella dove **R** è stato installato.

Documentazione On-Line

Maggiori riferimenti bibliografici e link a documentazione di varia natura possono essere trovati sul sito ufficiale, all'indirizzo `http://www.r-project.org/other-docs.html`.

Salvataggio e Memoria

Un aspetto importante da conoscere è come uscire da **R** salvando il proprio lavoro. Innanzitutto il comando per uscire da una sessione è `quit()` o, più semplicemente, `q()`.
Una volta mandato in esecuzione questo comando, il sistema apre in pop-up una finestra che chiede il salvataggio dell'area di lavoro.

- Rispondendo no si ottiene la chiusura del programma senza alcun salvataggio
- Rispondendo si, invece, il sistema salva tutti gli oggetti ed i comandi presenti in memoria (*workspace*) e termina la propria sessione. Se si riapre **R** dopo un salvataggio, il sistema ricarica automaticamente in memoria l'ultimo *workspace*

Salvataggio e Memoria - 2

Il salvataggio dell'area di lavoro consiste nella creazione di due files nella directory di **R**: .Rhistory e .Rdata.
Il primo file è un file di tipo ASCII che può essere aperto tramite un qualunque editor di testo (es. Blocco Note) e contiene tutti i comandi eseguiti durante la sessione salvata.
Il secondo è, invece, un file di tipo binario (codificato in linguaggio macchina e, quindi, dal contenuto illeggibile).

Salvataggio e Memoria - 3

Un comando che ci permette di visionare l'insieme di oggetti contenuti nella memoria del programma è dato dal comando `ls()`. Tale comando restituisce in uscita un vettore composto dai nome di tutti gli oggetti in memoria. Se non si è ancora definito alcun oggetto restituisce l'espressione `character(0)`, ovvero un vettore composto da 0 elementi. Se si desidera rimuovere un oggetto dalla memoria, ad esempio perchè ne occupa troppa, si può utilizzare il comando `rm(nomeoggetto)`. Se si desidera rimuovere più di un oggetto allora bisogna inserire come argomento della funzione una sequenza di nomi separati da una virgola oppure un vettore di tipo `character`. Per rimuovere tutti gli oggetti dal dataset è necessario eseguire una variante del precedente comando data da: `rm(list=ls(all=TRUE))`.

Salvataggio e Memoria - 4

E' utile scegliere la directory di lavoro (*working directory* o WD) in cui si andrà a salvare il workspace ed, eventualmente modificarla.
I comandi che assolvono a queste necessità sono: `getwd()` e `setwd(...)`.
Il primo fornisce in output il percorso (*path*) dell'attuale directory di lavoro.
Il secondo comando permette, invece, di cambiare WD specificando all'interno delle parentesi il *path* della directory scelta.

Caricare un dataset

Le applicazioni statistiche coinvolgono l'utilizzo di determinate metodologie nell'analisi di una specifica situazione. La situazione è rappresentata dall'informazione riassunta nella collezione di dati (dataset) che un ricercatore ha a disposizione. Generalmente i dataset sono memorizzati in numerosi formati. Ne vedremo degli esempi.

Una cosa molto comune è quella di avere i dati salvati in un foglio elettronico (spreadsheet), spesso in excel (.xls). Per importare questo tipo di dati è conveniente fare un passo intermedio, ovvero esportare i dati da excel in formato .csv (menu a tendina $\rightarrow$ salva con nome $\rightarrow$ formato .csv) e poi caricare il tutto su **R** tramite il comando `read.csv(``...'')`.