Introduzione al linguaggio R

Prof. Crescenzio Gallo
Università di Foggia
Dipartimenti di Area Medica

Il linguaggio R (che è open source e gratuito sotto la licenza GNU General Public License della Free Software Foundation) deriva da S-PLUS (la versione commerciale). È uno strumento molto potente, ma manca di una vera e propria interfaccia grafica e, di conseguenza, l'interazione con l'utente avviene a livello di linea di comando (il prompt è il carattere **>**).

Inoltre R, come tutti i linguaggi di derivazione UNIX, è *case sensitive*, cioè distingue tra lettere maiuscole e lettere minuscole.

Costanti e vettori

R lavora con valori, stringhe di caratteri, vettori e matrici, che vengono assegnati alle variabili con opportuni comandi. Ad esempio, il comando:

```
> y <- 3
```

assegna il valore 3 alla variabile y. Invece il comando:

```
> x <- c(1, 2, 3)
```

crea un vettore x contenente i numeri 1, 2 e 3, dove per *vettore* intendiamo una serie di numeri (o stringhe) consecutivi.

Matrici

Oltre ai vettori, in R possiamo definire le matrici. Ad esempio il comando:

```
> z <- matrix(c(1, 2, 3, 4, 5, 6, 7, 8), 2, 4, byrow=TRUE)
```

crea una matrice z di 2 righe e 4 colonne, contenente i numeri da 1 a 8. La matrice viene riempita per riga.

Per visualizzare il contenuto di una variabile basta digitarne il nome. Ad esempio:

```
> z
[,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
```

Gli elementi di una matrice possono essere richiamati con un opportuno utilizzo delle parentesi quadre:

```
> z[1,3]
[1] 3
```

Operazioni ed operatori

Le variabili possono essere create anche con opportune operazioni algebriche, che si eseguono utilizzando i normali operatori (+, -, *, /). Ad esempio:

```
> f<-2*y
> f
[1] 6
```

Funzioni ed argomenti

Per eseguire operazioni particolari si utilizzano, in genere, le funzioni. Una funzione è richiamata con un nome ed uno o più argomenti. Ad esempio, la funzione:

```
> log(5)
```

calcola il logaritmo naturale di 5 e richiede un solo argomento, cioè il numero di cui calcolare il logaritmo.

Al contrario, la funzione:

```
> log(100, 2)
```

calcola il logaritmo in base 2 di 100 e richiede due argomenti, il numero di cui calcolare il logaritmo e la base del logaritmo stesso.

Quando sono necessari due o più argomenti, essi debbono essere messi nell'ordine esatto (in questo caso prima il numero poi la base) oppure debbono essere utilizzati i riferimenti corretti. Ad esempio, le due seguenti funzioni restituiscono lo stesso risultato:

```
> log(100, base=2)
> log(base=2, 100)
```

Dataframe

Oltre a vettori e matrici, in R esiste un altro importante oggetto, il **dataframe**, costituito da una tabella di dati con una o più colonne (le *variabili*) e una o più righe (le *istanze*). A differenza della matrice, il dataframe può essere utilizzato per memorizzare variabili di diverso tipo (numeri e caratteri).

Un dataframe può essere creato unendo più vettori, come nell'esempio seguente:

```
> campioni <- c(1,2,3,4,5,6)
> provenienza <- factor(c('A','A','B','B','C','C'))
> dati <- c(12,15,16,13,11,19)
> tabella <- data.frame('Camp' = campioni, 'Prov' = provenienza, 'Rilev'
= dati)
```

```
> tabella
  Camp Prov Rilev
1    1    A    12
2    2    A    15
3    3    B    16
4    4    B    13
5    5    C    11
6    6    C    19
```

Per utilizzare i dati in un DATAFRAME, bisognerà accedere alle sue colonne. Per far questo possiamo utilizzare l'estrattore $:

```
> tabella$Camp
[1] 1 2 3 4 5 6
```

oppure possiamo utilizzare gli indici:

```
> tabella[,1]
[1] 1 2 3 4 5 6
```

I dataframe possono essere editati velocemente utilizzando il comando **`fix`**, che fa apparire una finestra di editing tipo 'foglio elettronico'.

Per avere informazioni sulla natura di un oggetto creato in R, posso usare la funzione **`str()`**, come nell'esempio seguente:

```
> str(tabella)
'data.frame':  6 obs. of  3 variables:
 $ Camp : num  1 2 3 4 5 6
 $ Prov : Factor w/ 3 levels "A","B","C": 1 1 2 2 3 3
 $ Rilev: num  12 15 16 13 11 19
```

R ci informa che l'oggetto 'tabella' è un dataframe composto da tre colonne, di cui la prima e la terza sono numeriche; la seconda è invece una variabile qualitativa (factor).

Immissione di numeri progressivi (sequenze)

Per creare una serie progressiva, si può utilizzare il comando `seq(n,m,by=step)` che genera una sequenza da `n` a `m` con passo pari a `step`.

```
> campioni <- seq(1,20,1)
> campioni
[1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

A volte le provenienze dei campioni dagli esperimenti sono sequenze ripetute di stringhe. Ad esempio, le prime quattro rilevazioni potrebbero essere riferite all'esperimento ESP1, le seconde quattro all'esperimento ESP2, le successive quattro all'esperimento ESP3.

Per creare velocemente questo vettore, possiamo utilizzare la funzione `rep()` in questo modo:

```
> provenienza <- rep(provenienza, each=4)
> provenienza
[1] ESP1 ESP1 ESP1 ESP1 ESP2 ESP2 ESP2 ESP2 ESP3 ESP3 ESP3 ESP3
Levels: ESP1 ESP2 ESP3
```

Notare l'uso della funzione `factor()` per creare un vettore di fattori (dati qualitativi).

Come leggere e salvare dati esterni

I dati possono anche essere immessi da tastiera con il comando `scan()`, ma soprattutto possono essere importati in R da file esterni.

Partiamo dal presupposto di aver creato con Excel il dataset illustrato di lato, che si riferisce a 20 piante di mais, e di averlo salvato in formato CSV con separatore ";" nel file "dati.csv".

Per importarlo in R nel DATAFRAME `dati` diamo il comando:

```
> dati <- read.csv2("dati.csv", header=TRUE)
```

Un oggetto R può essere salvato in un file esterno (in formato binario):

```
> save(file="dati1.rda", dati)
```

ed eventualmente ricaricato:

```
> load("dati1.rda")
```

Per scrivere in un file di testo CSV (in questo caso con separatore la virgola) si utilizza il seguente comando:

```
> write.table(dati, "dati2.csv",
row.names=FALSE, col.names=TRUE, sep=",")
```

Pianta	Varietà	Altezza
1	N	172
2	S	154
3	V	150
4	V	188
5	C	162
6	N	145
7	C	157
8	C	178
9	V	175
10	N	158
11	N	153
12	N	191
13	S	174
14	C	141
15	N	165
16	C	163
17	V	148
18	S	152
19	C	169
20	C	185

Selezionare un subset di dati

È possibile estrarre da un dataframe un subset di dati utilizzando la funzione: **`subset`**(dataframe, condizione). Ad esempio, se consideriamo il dataframe precedente, è possibile selezionare tutte le righe relative alle varietà N e C come segue:

```
> dati2 <- subset(dati, Varietà=="N" | Varietà=="C")
> dati2
   Pianta Varietà Altezza
1       1       N     172
5       5       C     162
6       6       N     145
7       7       C     157
8       8       C     178
10     10       N     158
11     11       N     153
12     12       N     191
14     14       C     141
15     15       N     165
16     16       C     163
19     19       C     169
20     20       C     185
```

Si noti il carattere | che esprime la condizione logica OR.

Ordinare un vettore o un dataframe

Un vettore (numerico o carattere) può essere ordinato con il comando `sort()`:

```
> y <- c(12, 15, 11, 17, 12, 8, 7, 15)
> sort(y, decreasing = FALSE)
[1] 7 8 11 12 12 15 15 17
> z <- c("A", "C", "D", "B", "F", "L", "M", "E")
> sort(z, decreasing = TRUE)
[1] "M" "L" "F" "E" "D" "C" "B" "A"
```

Un dataframe può essere invece ordinato con il comando `order()`:

```
> dati[order(dati$Varietà, dati$Altezza), ]
   Pianta Varietà Altezza
14     14       C     141
7       7       C     157
5       5       C     162
16     16       C     163
19     19       C     169
8       8       C     178
20     20       C     185
6       6       N     145
11     11       N     153
10     10       N     158
15     15       N     165
1       1       N     172
12     12       N     191
18     18       S     152
2       2       S     154
13     13       S     174
17     17       V     148
3       3       V     150
9       9       V     175
4       4       V     188
```

Pianta	Varietà	Altezza
1	N	172
2	S	154
3	V	150
4	V	188
5	C	162
6	N	145
7	C	157
8	C	178
9	V	175
10	N	158
11	N	153
12	N	191
13	S	174
14	C	141
15	N	165
16	C	163
17	V	148
18	S	152
19	C	169
20	C	185

Ordinare un vettore o un dataframe

Per l'ordinamento decrescente si utilizza il segno meno:

```
> dati[order(dati$Varietà, -dati$Altezza), ]
   Pianta Varietà Altezza
20     20       C     185
8       8       C     178
19     19       C     169
16     16       C     163
5       5       C     162
7       7       C     157
14     14       C     141
12     12       N     191
1       1       N     172
15     15       N     165
10     10       N     158
11     11       N     153
6       6       N     145
13     13       S     174
2       2       S     154
18     18       S     152
4       4       V     188
9       9       V     175
3       3       V     150
17     17       V     148
```

Pianta	Varietà	Altezza
1	N	172
2	S	154
3	V	150
4	V	188
5	C	162
6	N	145
7	C	157
8	C	178
9	V	175
10	N	158
11	N	153
12	N	191
13	S	174
14	C	141
15	N	165
16	C	163
17	V	148
18	S	152
19	C	169
20	C	185

Workspace

Gli oggetti creati durante una sessione di lavoro vengono memorizzati nel cosiddetto *workspace*.

Il contenuto di quest'ultimo può essere visualizzato:

```
> ls()
```

cancellato:

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

salvato nella directory corrente:

```
> save.image('nomefile.RData')
```

e richiamato, per proseguire il lavoro dal punto in cui lo si è interrotto:

```
> load('nomefile.RData')
```

Script o programmi

in R è possibile scrivere programmi (funzioni) in un semplice editor di testo (o con quello incorporato in R) da memorizzare in un file con estensione '.r' e richiamare in seguito. Lavorare con gli script è molto comodo e consigliabile perché sono sequenze di comandi riutilizzabili.

Oltre agli script, è possibile creare funzioni personalizzate fino ad arrivare a veri e propri programmi (packages).

Interrogazione di oggetti

R memorizza i risultati delle analisi negli oggetti, mostrando un output video piuttosto minimale. Per ottenere informazioni è necessario interrogare opportunamente gli oggetti che al loro interno possono contenere altri oggetti da cui recuperare le informazioni interessanti. Gli oggetti che contengono altri oggetti sono detti *liste*.

Ad esempio, se vogliamo calcolare autovettori ed autovalori di una matrice, utilizziamo la funzione `eigen()`. Questa funzione restituisce una lista di oggetti, che al suo interno contiene i due oggetti `values` (autovalori) e `vectors` (autovettori). Per recuperare l'uno o l'altro dei due risultati si usa l'operatore di concatenamento (detto anche *estrattore*) $:

```
> matrice <- matrix(c(2,1,3,4),2,2)
> ev <- eigen(matrice)
> ev
eigen() decomposition
$values
[1] 5 1
$vectors
           [,1]       [,2]
[1,] -0.7071068 -0.9486833
[2,] -0.7071068  0.3162278
> ev$values
[1] 5 1
```

Altre funzioni matriciali

Oltre che autovettori ed autovalori di una matrice, R ci permette di gestire altre funzioni matriciali. Se ad esempio abbiamo le matrici:

$$Z = \begin{pmatrix} 1 & 2 \\ 2 & 3 \end{pmatrix} \quad Y = \begin{pmatrix} 3 & 2 \end{pmatrix}$$

possiamo caricarle in R con i seguenti comandi:

```
> Z <- matrix(c(1,2,2,3),2,2)
> Y <- matrix(c(3,2),1,2)
```

Possiamo poi ottenere la trasposta di Z con il comando:

```
> t(Z)
     [,1] [,2]
[1,]    1    2
[2,]    2    3
```

Possiamo moltiplicare Y e Z utilizzando l'operatore `%*%`:

```
> Y%*%Z
     [,1] [,2]
[1,]    7   12
```

Possiamo calcolare l'inversa di Z con:

```
> solve(Z)
     [,1] [,2]
[1,]   -3    2
[2,]    2   -1
```

Cenni sulle funzionalità grafiche in R

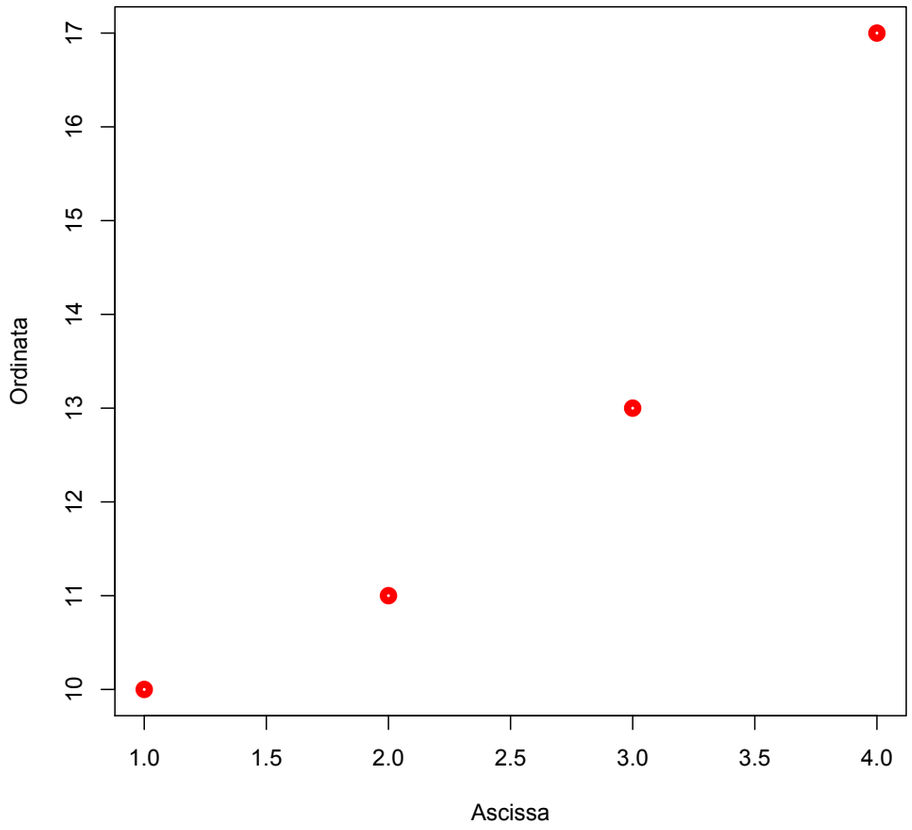
R è un linguaggio abbastanza potente e permette di creare grafici interessanti. La funzione più utilizzata per produrre grafici è:

```
plot(x, y, type, xlab, ylab, col, lwd, lty...)
```

dove `x` ed `y` sono i vettori con le coordinate dei punti da disegnare; `type` rappresenta il tipo di grafico ("p" produce un grafico a punti, "l" un grafico a linee, "b" disegna punti uniti da linee, "h" disegna istogrammi); `Title` disegna il titolo del grafico, `sub` il sottotitolo, `xlab` e `ylab` le etichette degli assi, `col` è il colore dell'oggetto, `lwd` il suo spessore, `lty` il tipo di linea e così via.

Per una descrizione più dettagliata si consiglia di consultare la documentazione on line. A titolo di esempio mostriamo che i comandi seguenti producono il grafico qui a lato:

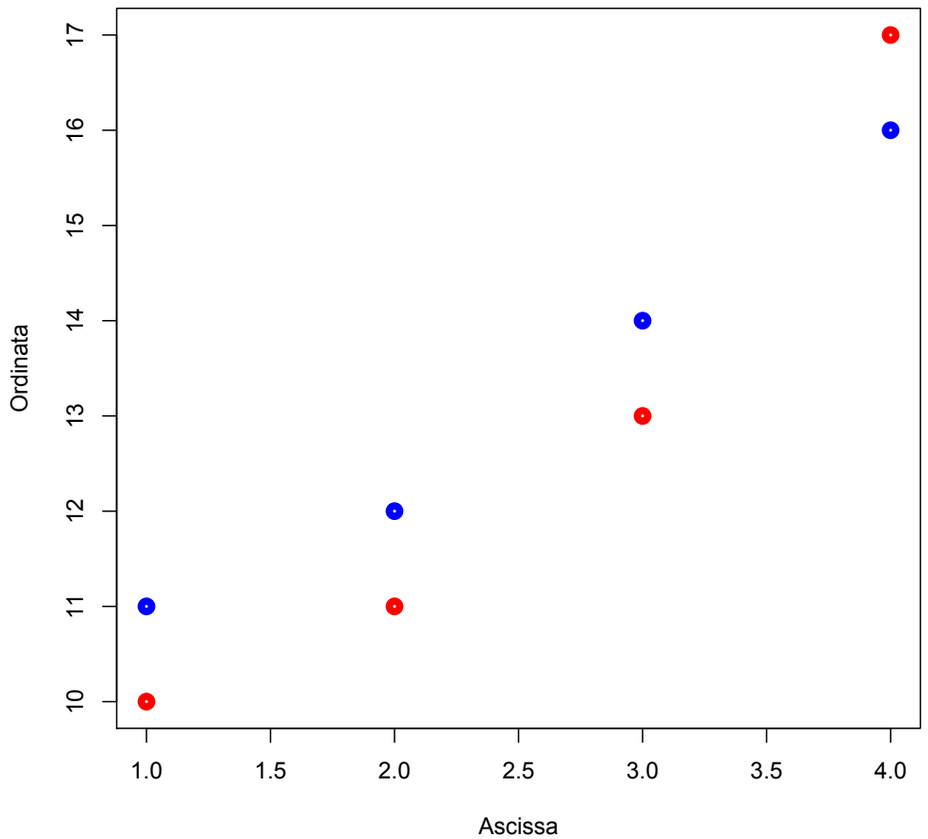
```
> x <- c(1,2,3,4)
> y <- c(10,11,13,17)
> plot(x, y, "p", col="red", lwd=5,
xlab="Ascissa", ylab="Ordinata")
```



Per sovrapporre un'altra serie di punti alla precedente possiamo utilizzare il comando:

```
> y2 <- c(11,12,14,16)
> points(x, y2, col="blue", lwd=5)
```

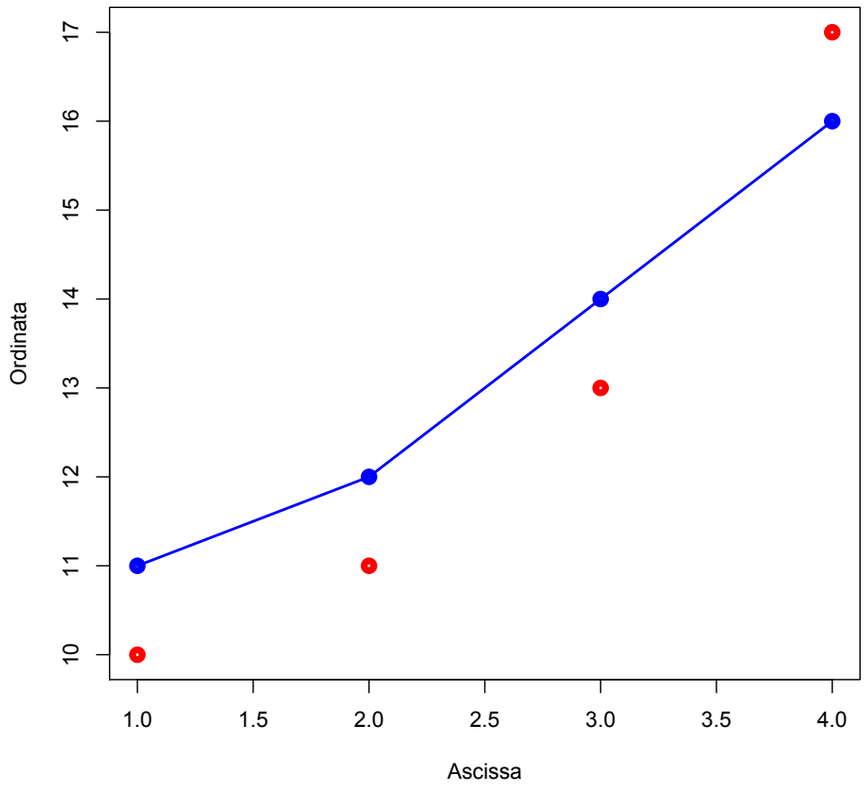
che produce come output la figura qui a destra.



Se avessimo voluto sovrapporre una grafico a linee avremmo invece utilizzato la funzione:

```
> lines(x, y2, col="blue", lwd=2)
```

che avrebbe prodotto il grafico qui sotto:



Per disegnare una curva si può utilizzare la funzione:

```
> curve(funzione, Xiniziale, Xfinale,
add=FALSE/TRUE)
```

dove il metodo `add` serve per specificare se la funzione deve essere aggiunta ad un grafico preesistente.

Per aggiungere un titolo ad un grafico possiamo utilizzare la funzione:

```
> title(main="Titolo")
```

Per aggiungere una legenda utilizziamo la funzione:

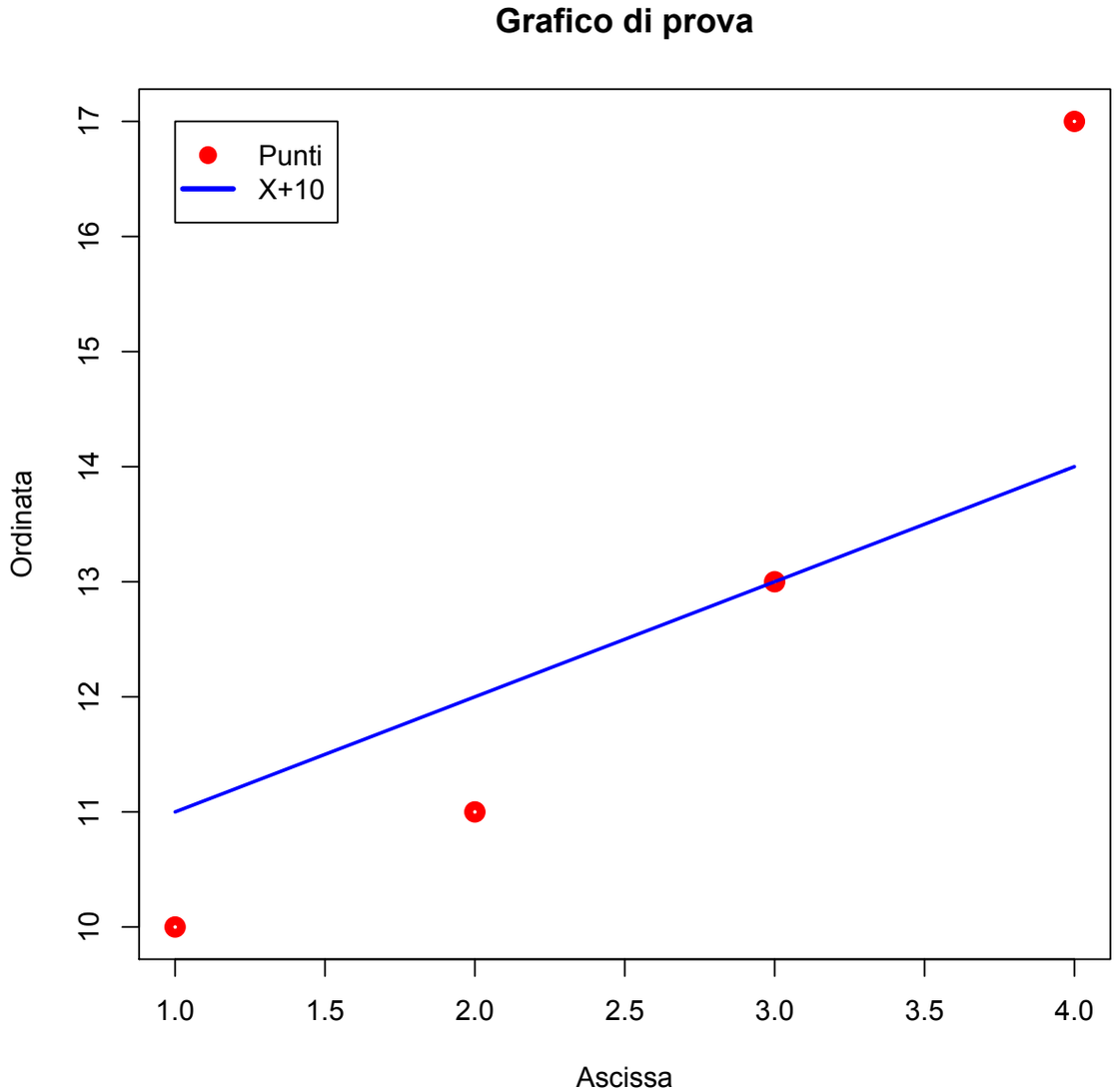
```
> legend(Xcoord, YCoord , legend=c("Punti","X+10"),
pch=c(19,-1), col=c("Red","Blue"), lwd=c(3,3), lty=c(0,3))
```

ove i vettori indicano, per ogni elemento della legenda, il testo che deve essere riportato (legend), il tipo di simbolo (pch, con -1 che indica nessun simbolo), il colore (col), la larghezza (lwd) e il tipo di linea (lty, con 0 che indica nessuna linea).

Ad esempio:

```
> plot(x, y, "p", col="red",
lwd=5, xlab="Ascissa",
ylab="Ordinata")
> curve(10+x, add=TRUE, lty=1,
lwd=2, col="blue")
> title(main="Grafico di prova")
> legend(1, 17,
legend=c("Punti","X+10"),
pch=c(19,-1),
col=c("Red","Blue"), lwd=c(3,3),
lty=c(0,1))
```

produce il grafico qui a lato:



Un'ultima cosa utile da menzionare è la possibilità di disegnare grafici a torta, utilizzando il comando:

```
> pie(vettoreNumeri, vettoreEtichette, vettoreColori)
```

Ad esempio il comando:

```
> pie(c(20,30,50), label=c("B","C"), col=c("blue","green","red"))
```

produce l'output riportato qui sotto: