Introduzione ad R

Roberto Boggiani
Versione 4.0

21 gennaio 2003

1 Introduzione

1.1 Introduzione ad R

R è un ambiente statistico per la manipolazione, l'analisi e la rappresentazione grafica dei dati. E' un ambiente interattivo, ossia i comandi producono una risposta immediata, e prevedono una programmazione orientata agli oggetti. Avviato il programma R, apparirà una nuova finestra con ilsimbolo
>
che indica che l'ambiente è pronto per ricevere delle istruzioni.

1.2 Il salvataggio dei dati

In ogni sessione di R vengono continuamente inseriti nuovi dati e nuovi comandi vengono digitati. Per salvare le variabili create sarà sufficiente utilizzare il comando:
`>save workspace`
disponibile dal menù file, mentre per salvare i comandi digitati utilizzeremo il comando:
`>save history`
sempre disponibile dal menù file. Si noti che tali file possono essere memorizzati in qualunque directory si voglia e quindi si potranno creare directory diverse a seconda delle indagini statistiche che si stanno realizzando.

1.3 Help

L'ambiente R dispone di un help in linea molto efficiente che consente di ottenere delle informazioni sulle funzioni in esso implementate. Vi sono tre diversi sitemi di help:

- help in formato windows si ottiene con la seguente sequenza di comandi
  > options(winhelp=T)
  > help()
- help in formato html compilato si ottiene con
  > options(chmhelp=T)
  > help()
- help in formato html si ottiene con
  > help.start()

In ogni caso se si vogliono delle informazioni su un particolare comando basterà utilizzare la sintassi
> ?comando
comparirà la descrizione del comando e la sua sintassi d'uso. Provare per esempio a digitare
> ?mean
Nel caso in cui si voglia la ricerca di un help relativo ad una parola chiave possiamo utilizzare il comando:
`>help.search`("parolachiave")
in questo modo compariranno tutti i comandi in cui la parola chiave è contenuta.

1.4 Personalizzazione

Il programma R permette anche di effettuare una personalizzazione di certe opzioni come il browser in cui sarà visualizzato l'help, le cifre decimali da visualizzare, come deve apparire come prompt dei comandi. Una lista completa di tali personalizzazioni si può vedere dando in R il comando:
`>?options`
Se vogliamo utilizzare nel corso della sessione avviata i numeri con 4 cifre decimali dobbiamo dare il comando:
`>options( digits=4)`
se desideriamo invece che in ogni avvio di R i numeri siano sempre visualizzati con 4 cifre decimali dobbiamo modificare la variabile di avvio .First inserendo in essa il comando appena digitato. Ciò e fatto con:
`>.First<-fix(.First)`
ed inserendo tra le parentesi graffe la riga:
`>options( digits=4)`
Tale metodo resta valido per ogni opzione che puo' essere introdotta con il comando options, così se vogliamo modificare cifre ed editor basterà date il comando:
`>options( digits=4,editor=`"myeditor")
o inserirlo all'interno della varibile .First a seconda dei casi.

1.5 Fissare il numero delle cifre da visualizzare

Per fissare il numero di cifre da visualizzare si usa il seguente comando:
`>option(digits=n)`
in questo modo tutte le cifre successivamente ottenute saranno visualizzate con n cifre decimali. Se si vuole mantenere fisso questo numero in ogni sessione di avvio del programma si veda quanto scritto nel paragrafo 1.4.

2 Il caricamento dei pacchetti

2.1 I pacchetti in R

Il programma presenta una serie di pacchetti aggiuntivi che ne aumentano notevolmente la potenzialità. I pacchetti che possono essere utilizzati possono essere di tre tipologie diverse:

- pacchetti automaticamente installati e automaticamente avviati
- pacchetti automaticamente installati ma non avviati
- pacchetti reperibili in rete

mentre i pacchetti del primo tipo non presentano problemi, in quanto disponibili e utilizzabili sempre all'avvio del programma, i pacchetti degli altri due tipi richiedono alcune considerazioni circa il loro uso.

2.2 Pacchetti automaticamente installati ma non avviati

Quando il programma è installato oltre alla sua configurazione base sono automaticamente installati una serie di pacchetti aggiuntivi che sono molto utili nelle analisi statistiche. Tali pacchetti

non sono utilizzabili all'avvio del programma ma devono essere richiamati tramite la voce del menu con la seguente sequenza di comandi:
`>Packages - load packages`
e scegliere il pacchetto da installare dalla lista presentata. Noi utilizzeremo spesso i pacchetti **grid, lattice, stepfun** non precaricari e per questo motivo converrà che tali pacchetti siamo precaricati all'avvio del programma. Per far si che tali pacchetti vengano caricati automaticamente all'avvio di R sarà necessario modificare la funzione .First nel seguente modo:
`>.First<-fix(.First)`
in modo che il suo contenuto divenga il seguente:

```
function() {
        require("ctest", quietly = TRUE)
        require("grid", quietly = TRUE)
        require("lattice", quietly = TRUE)
        require("stepfun", quietly = TRUE)
                }
```

2.3 Pacchetti reperibili in rete

Esistono anche dei pacchetti reperibili in rete, molto interessanti e che presentano funzioni molto importanti per compiere sofisticate analisi statistiche di dati. Per caricare tali pacchetti in R dobbiamo utilizzare la seguente procedura:

- prima di tutto aggiungere il pacchetto ad R e ciò viene fatto con la sequenza di comandi Packages - install package from local zip file - selezionare la directory dove è memorizzato il file - doppio click
- caricare il pacchetto in R è ciò avviene con la sequenza di comandi Packages - load packages e scegliere il pacchetto da installare dalla lista presentata

Vedremo in un capitolo successivo come creare propri pacchetti aggiungibili ad R.

2.4 Pacchetti in codice sorgente

Un metodo molto pratico per aggiungere pacchetti appositamente creati è quello che consiste nella creazione di appositi file detti source nei quali vengono inserite più funzioni scritte in linguaggio R come vedremo nel paragrafo 19. Per caricare tali funzioni basterà digitare la sequenza file - source e indicare il percorso e il nome del file dove il pacchetto codice è memorizzato. In questo modo possiamo facilmente caricare funzioni personali create ad hoc per l'analisi di dati.

3 Gli operatori

3.1 Introduzione

Prima di proseguire nell'analisi dei principali comandi del programma R, sarà opportuno conoscere i principali operatori che possono essere utilizzati in tale programma. Tale analisi sarà svolta nei paragrafi successivi.

3.2 Operatori aritmetici

Gli operatori aritmetici sono gli operatori fondamentali che servono per svolgere le principali operazioni matematiche. Essi sono:

- + addizione
- − sottrazione
- ∗ moltiplicazione
- / divisione
- `^` elevamento a potenza

Il loro uso è analogo a quello che comunemente si fa su una calcolatrice ad esempio:

- `> (7+2)*3`
- `> 7^2`

3.3 Gli operatori relazionali

Gli operatori relazionali sono gli operatori fondamentali per effettuare confronti tra numeri o lettere. Essi sono:

- `<` minore
- `>` maggiore
- `<=` minore o uguale
- `>=` maggiore o uguale
- `==` uguale
- `!=` diverso

Un esempio del loro uso può essere il seguente:
`> 4>2`

3.4 Gli operatori logici

Gli operatori logici sono gli operatori fondamentali che servono per collegare tra di loro espressioni contenenti operatori relazionali. Essi sono:

- `&` che sta per l'operatore and
- `|` che sta per l'operatore or

Un esempio del loro uso può essere il seguente:
`> 1>2 & 2<3`

4 Le funzioni matematiche elementari e i numeri complessi

4.1 Le funzioni elementari matematiche

Il programma mette a disposizione molte funzioni matematiche tra cui segnaliamo:

- sqrt() radice quadrata
- abs() valore assoluto
- sin() cos() tan() funzioni trigonometriche
- asin(), acos(), atan() funzioni trigonometriche inverse
- sinh(), cosh() tanh() funzioni iperboliche
- asinh(), acosh(), atanh() funzioni iperboliche inverse
- exp() la funzione e
- log() logaritmo base e
- log10() logaritmo in base 10
- ceiling() arrotonda all'intero più alto
- floor() arrotonda all'intero più basso
- trunc() tronca la parte decimale
- round(x,n) arrotonda al numero di cifre specificato con n
- signif(x,n) arrotonda al numero di cifre significative specificate con n
- pi restituisce il valore di pi greco

4.2 I numeri complessi

I numeri complessi sono utilizzati nel seguente modo:

- `>a+ib` restituisce un numero complesso
- `>a+1i` restituisce un numero complesso con parte immaginaria avente coefficiente reale 1
- `>Re(a+ib)` restituisce la parte reale del numero complesso
- `>Im(a+ib)` restituisce la parte immaginaria del numero complesso
- `>Mod(a+ib)` restituisce il modulo della forma polare del numero complesso
- `>Arg(a+ib)` restituisce l'argomento della forma polare del numero complesso
- `>Conj(a+ib)` restituisce il coniugato di un numero complesso

I calcoli con i numeri complessi posso essere eseguiti normalmente con i comuni operatori aritmetici visti sopra.

5 Le variabili

5.1 Assegnazione di un valore ad una variabile

Per assegnare un valore numerico alla variabile x si usa il comando:
`> x<-2`
mentre per assegnare un carattere alla variabile y si usa il comando:
`> x<-`“casa“
Assegnando un nuovo valore ad una variabile, verrà automaticamente cancellato il valore precedentemente assunto dalla stessa.

Un altro modo per assegnare un valore ad una variabile è quello evidenziato dai seguenti esempi:

- `>assign`(“x“,10)
- `>assign`(“x“,“casa“)

5.2 Visualizzare il contenuto di una variabile

Per visualizzare il contenuto di una variabile basterà digitare il nome della variabile stessa: `> x` `>y`

5.3 Visualizzazione di tutte le variabili esistenti

Per visualizzare il nome di tutte le variabili esistenti nell'area di lavoro si usa uno dei seguenti comandi:
`>objects()`
`>ls()`

5.4 Cancellazione di variabili

Per cancellare una variabile di nome `x` basterà digitare il comando:
`> rm(x)` Per calcellare tutte le variabili esistenti nell'area di lavoro comprese le funzioni create o aggiunte si usa invece il comando:
`>rm(list=ls())`
attenzione in questo caso ad essere sicuri di quello che si sta facendo.

6 Oggetti base di R

6.1 Gli oggetti base

Gli oggetti base usati dal programma R sono i seguenti:

- vettori
- matrici
- array
- liste
- fattori

- serie storiche
- data frame

I prossimi paragrafi saranno dedicati allo studio di tali oggetti.

6.2 Gli attributi degli oggetti base

Tutti gli oggetti base di R possiedono degli attributi che permettono loro di essere più flessibili ed utilizzabili in modo più veloce. Nella singola trattazione degli oggetti verrano indicati in dettaglio oggetto per oggetto gli attributi posseduti.

7 I vettori

7.1 Introduzione

I vettori sono dati dello stesso tipo che sono raggruppati in una unica variabile. Essi sono molto importanti nell'analisi statistica dei dati in quanto dati raggruppati in vettori sono generalmente facili da essere studiati e manipolati.

7.2 La funzione *c*

Con il comando `c(elemento1,elemento2,.....)` viene creato un vettore tramite la concatenazione degli elementi specificati tra parentesi e separati da virgole. Gli elementi non possono essere contemporaneamente caratteri e numeri. Ad esempio con:

```
>x<-c(1.5,2,2.5)
```

viene creato un vettore numerico che contiene i valori specificati.

Sempre tramite in comando `c()` si possono aggiungere nuovi elementi ad un vettore precedentemente creato. Ad esempio con:

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

viene aggiunto alla fine del vettore `x` il numero 3.

Per ottenere un vettore di stringhe sarà sufficiente inserire ogni stringa tra apici, come nel seguente esempio:

```
>x<-c(“questo“,“è“,“un esempio“)
```

7.3 Tipologie di vettori

I vettori in R possono essere di varie tipologie:

- numerico
- logico
- complesso
- caratteri

a secondo della tipologia di elementi che li compongono.

7.4 Come creare un vettore

Oltre alla funzione `c` per creare un vettore possiamo utilizzare uno dei seguenti metodi:

- `>x<-scan()` crea un vettore numerico
- `>x<-scan(what=`"character") crea un vettore di caratteri
- `>x<-scan(what=`"complex") crea un vettore di numeri complessi
- `x<-n1:n2` crea un vettore composto da numeri che vanno da `n1` a `n2` con passo pari ad `uno`

il vantaggio di tale metodo è quello di poter inserire gli elementi uno ad uno direttamente da tastiera, il semplice invio senza aver digitato nulla equivale alal conclusione dell'immissione dei dati nel vettore.

7.5 Inizializzazione di un vettore

In molti casi invece di creare un vettore abbiamo solamente bisogno di inizializzarlo. Questa operazione viene fatta con uno dei seguenti comandi:

- `>x<-vector(`"numeric",5)
- `>x<-logical(5)`
- `>x<-numeric(5)`
- `>x<-complex(5)`
- `>x<-character(5)`

Si noti che il primo modo di inizializzare un vettore potrà essere usato anche per vettori di tipo logico, complessi o di caratteri.

7.6 Creazione di un vettore con la funzione *fix*

Tramite l'utilizzo della funzione `fix` è possibile procedere alla creazione di un vettore utilizzando la seguente sintassi:
`>x<-0`
`>x<-fix(x)`
e scrivendo i numeri utilizzando la sintassi:
`c(n1,n2,...)`
in cui `n1,n2` sono i numeri che compongono il vettore stesso.

7.7 Attributi di un vettore

Se `x` è un vettore qualsiasi, gli attributi di cui esso gode saranno i seguenti:

- `>length(x)` lunghezza del vettore
- `>mode(x)` modo del vettore
- `>names(x)` nomi del vettore

7.8 Nomi degli elementi di un vettore

Una volta creato un vettore che indicheremo con `x` sarà possibile assegnare a ciascun elemento del vettore un nome o una etichetta. Per fare ciò dobbiamo pero' avere a disposizione un vettore che indicheremo con `nomi` della stessa lunghezza di `x` che potrà essere sia numerico che composto di caratteri. L'attribuzione del nome avviene nel seguente modo:
`>names(x)<-c(y)`

7.9 Richiamare i singoli elementi di un vettore

Per richiamare i singoli elementi di un vettore `x` possiamo usare uno dei seguenti modi:

- `>x` richiama l'intero vettore
- `>x[n]` richiama l'elemento di posto `n` del vettore
- `>x[c(n1,n2,n3)]` richiama gli elementi di posto `n1,n2,n3` del vettore
- `>x[n1:n2]` richiama gli elementi di posto da `n1` a `n2` del vettore
- `>x[-(n1:n2)]` richiama tutti gli elementi del vettore tranne quelli da `n1` a `n2`
- `>x[-c(n1,n2,n3)]` richiama tutti gli elementi del vettore tranne quelli di posto `n1,n2,n3`
- `>x[x>n1]` richiama gli elementi del vettore maggiori di `n1`
- `>x[x>n1 | x<n2]` richiama gli elementi del vettore maggiori di `n1` o minori di `n2`
- `>x[x>n1 & x<n2]` richiama gli elementi del vettore maggiori di `n1` e minori di `n2`
- `>x[`"a"] restituisce l'elemento del vettore con etichetta a"

7.10 Creare un vettore con seq

La funzione `seq` viene usata per creare delle sequenze di numeri. La sintassi è la seguente
`seq(primo,ultimo,incremento)`
in cui:

- `primo` è il primo elemento della sequenza
- `ultimo` è l'ultimo elemento della sequenza
- `incremento` è il passo con cui si va da `primo` ad `ultimo`

Si noti anche che:

- `incremento` può anche essere un numero negativo, in tale caso la sequeza al posto di essere crescente sarà decrescente, in questo caso si faccia attenzione ai valori dati a `primo` e `ultimo`
- se `incremento` non viene specificato viene utilizzato come valore `uno`

7.11 Creare un vettore con rep

La funzione `rep` serve per creare un vettore con dati ripetuti. La sua sintassi è la seguente:

- `>x<-rep(2,5)` crea 2, 2, 2, 2, 2
- `>x<-rep(c(1,2,3),2)` crea 1, 2, 3, 1, 2, 3
- `>x<-rep(c(1,2,3), each=2)` crea 1, 1, 2, 2, 3, 3
- `>x<-rep(c(1,2,3),c(2,3,4))` crea 1, 1, 2, 2, 2, 3, 3, 3, 3

7.12 Creare un vettore con cut

La funzione `cut` serve per assegnare i valori di un vettore, di solito derivante da dati continui, a classi di ampiezza prefissata o no. Si tratta in generale di una operazione di dicretizzazione del vettore. In certi casi, infatti, dato un vettore derivante da una distribuzione continua è necessario assegnare le sue componenti a classi appositamente prefissate. Supponendo di avere a disposizione un vettore `x` per compiere tale operazione possiamo usare i seguenti comandi:

- `>x<-cut(y,3)` suddivide y in tre gruppi
- `>x<-cut(y,breaks=c(0,2,4,6))` suddivide i dati nei gruppi 0-2, 2-4, 4,6
- `>x<-cut(y,breaks=c(0,2,4,6),labels=c(`“0-2“,“2-4“,“4-6“)) li suddivide nei gruppi e attribuisce loro i nomi scritti in labels
- `>x<-cut(y,breaks=seq(2,6,2),labels=c(`“0-2“,“2-4“,“4-6“))come sopra ma con uso di seq

Le calssi così ottenute risulteranno essere chiuse a destra, ma usando il comando `cut` con l'opzione `right=F` la classe che viene creata risulta essere chiusa a sinistra.

Si noti che possiamo anche limitare le classi ad esempio se x è un vettore con i primi 100 numeri naturali con in comando
`<cut(x,c(0,5,10,15,100),labels=c(`“0-5“,“5-10“,“10-15“,“15+))
otteniamo una suddivisione di x in 4 classi.

7.13 Creare un vettore logico

In certi casi è necessario stabilre quanti e quali elementi che compongono un vettore soddisfino ad una determinata condizione. Ciò può essere fatto con la creazione di un vettore logico, ossia un vettore che contiene solamente `TRUE o FALSE`. Se `x` è un vettore composto da numeri naturali con:
`>x<=5`
otteniamo un vettore logico composto da `T o F` a seconda che l'elemento del vettore soddisfi o meno la condizione scritta.

7.14 Ordinare un vettore

Per ordinare elementi di un vettore `x` utilizziamo il seguente comando:
`>sort(x)`
con il quale gli elementi di `x` sono ordinati in modo crescente, se invece vogliamo ottenere l'ordinamento decrescente dobbiamo utilizzare il comando:
`>sort(x,decreasing=T)`

7.15 Le funzioni elementari statistiche

Dato un vettore di tipo numerico `x`, le principali funzioni elementari statistiche applicabili a tale vettore sono le seguenti:

- `>min(x)` restituisce il minimo di x
- `>max(x)` restituisce il massimo di x
- `>range(x)` restituisce il range di x
- `>mean(x)` restituisce la media aritmetica semplice di x
- `>median(x)` restituisce la mediana di x
- `>quantile(x,y)` con `y` vettore di numero compresi tra zero ed uno, restituisce i quantili di x in base ai valori contenuti in `y`
- `>var(x)` restituisce la varianza di x
- `>sd(x)` restituisce la deviazione standard di x
- `>mad(x)` calcola la median absolute deviation
- `cor(x,y)` con `y` vettore numerico restuisce la correlazione tra x ed y
- `>cumsum(x)` restituisce la somma progressiva di x
- `>cumprod(x)` restituisce il prodotto progressivo di x
- `>sum(x)` restituisce la somma di x
- `>prod(x)` restituisce il prodotto di x

7.16 Operazioni che coinvolgono i vettori

Quando applichiamo una funzione aritmetica o di confronto ad un vettore, tale funzione sarà applicata a ciascun elemento che compone il vettore stesso. Quindi se `x` e `y` sono due vettori qualunque con:

- `>a+b` otteniamo un vettore che ha come elementi la somma degli elementi corrispondenti dei vettori `x` e `y`
- `>a*b` otteniamo un vettore che ha come elementi il prodotto degli elementi corrispondenti dei vettori `x` e `y`
- `>log(x)` otteniamo un vettore i cui elementi sono i logaritmi degli elementi di `x`

8 Le matrici

8.1 Introduzione

Le matrici sono costotuite da dati dello stesso tipo che sono graggruppati in tabelle a doppia entrata. Anche le matrici come i vettori sono fondamentali nell'analisi statistica dei dati in quanto permettono un'analisi congiunta di due aspetti di una stessa unità statistica.

8.2 Come creare una matrice

Le matrici possono essere di tipo numerico, di stringhe di caratteri e logiche. Se il vettore `x` contiene i dati da inserire nella matrice ordinati secondo le colonne, per creare una matrice possiamo utilizzare la funzione `matrix` nel seguente modo:
`matrix(x,nrow=numerorighe,ncol=numerocolonne,byrow=F)`
Si noti che la lunghezza del vettore deve essere pari al valore di `nrow*ncol` in caso contrario si ottiene un messaggio di errore. Precisiamo inoltre quanto segue:

- l'opzione `byrow=F` è stabilita di default e fa sì che la creazione della matrice avvenga procedendo seguendo l'ordine delle colonne. Essa può essere quindi omessa se si desidera che i dati del vettore `x` siano ordinati secondo le colonne
- specificando l'opzione `byrow=T` vengono assegnati alla matrice i dati del vettore `x` seguendo l'ordine delle righe
- è sufficiente fissare uno solo dei parametri `nrow` e `ncol`in quanto il mancante sarà automaticamente determinato dalla lunghezza del vettore `x`

Altri modi in cui può essere creata una matrice sono i seguenti:

- `>x<-matrix(1:10,5,2)`
- `>x<-matrix(c(1,2,3,4),nr=2,dimnames=list(AAA=c(`“a“,“b“),BBB=c(“A“,“B“)))
- `>x<-matrix(c(1,0,0,1),2)`
- `>x<-1:10`
  `>dim(x)<-c(5,2)`
- `>x<-matrix(scan(datafile),ncol=10)`
- x>`x<-matrix(scan(datafile,what=`“character“),ncol=10)
- `>x<-matrix(scan(),ncol=5)`

Se si vuole creare una matrice avente tutti gli elementi uguali ad un numero prefissato `a` possiamo dare il seguente comando:
`>matrix(a,2,10)`
crea una matrice 2 x 10 composta di numeri `a`

8.3 Come creare una matrice diagonale

Per creare una matrice diagonale da un vettore `x` di dati noti possiamo dare la seguente sequenza di comandi:
`diag(x)`
Si ottiene una matrice diagonale di ordine pari alla lunghezza del vettore `x`.

8.4 Creazione di una matrice la funzione fix

Tramite l'utilizzo della funzione `fix` è possibile procedere alla creazione di una matrice utilizzando la seguente sintassi:
`>x<-matrix(0)`
`>fix(x)`
e scrivendo i numeri all'interno delle singole celle.

8.5 Attributi di una matrice

Se `x` è una data matrice, applicando i seguenti comandi possiamo ottenere alcune proprietà della matrice stessa:

- `>length(x)` restituisce la lunghezza di x
- `>mode(x)` restiduisce il modo di x
- `>dimnames(x)` restituisce i nomi di x
- `>dim(x)` restituisce le dimensioni di x

8.6 Dare un nome alle righe e colonne di una

Se `x` è una matrice di dimensioni mxn è possibile dare un nome alle sue righe ed alle sue colonne con il seguente comando:

```
dimnames(x)<-list(c("nomerig1","nomerig2",...,"nomerigm"),
c("nomecol1","nomecol2",...,"nomecoln"))\\
```

Se vogliamo dare un nome solamente alle sue righe usiamo il seguente comando:

```
dimnames(x)[[1]]<-list(c("nomerig1","nomerig2",...,"nomerigm"))
```

Se invece vogliamo dare un noime solamente alle sue colonne usiamo il seguente comando:

```
dimnames(x)[[2]]<-list(c("nomecol11","nomecol2",...,"nomecoln"))
```

8.7 Estrarre dati da una matrice

Data una matrice mxn `x` di cui sappiamo che:

- le righe sono state chiamate nomerig1, nomerig2,...,nomerigm
- le colonne sono state chiamate nomecol1, nomecol2, ..., nomecoln

per richiamare i suoi elementi possiamo utilizzare i seguenti comandi:

- `>x[,`"nomecol1"]=x[,1] richiama la prima colonna di `x`
- `>x[`"nomerig1",]=x[1,] richiama la prima riga di `x`
- `>x[,c(`"nomecol1","nomecol2")]=x[,c(1:2)] richiama le prime due colonne di `x`
- `>x[,c(1,3)]` richiama la prima e la terza colonna di `x`
- `>x[c(1,3),c(1,3)]` richiama la prima e la terza riga e colonna di `x`
- `>x[-1,c(1,3)]` richiama la prima e la terza colonne di `x` tranne la riga 1
- `>x[1:2,3]` richiama i primi 2 elementi della colonna 3
- `>x[x[,1]>=2,1:2]` richiama le colonne da 1 a 2 di `x` i cui elementi della prima colonna sono maggiori o uguali a 2
- `>x[x[,1]>=2,]` richiama tutte le colonne di `x` i cui elementi della prima colonna sono maggiori o uguali a 2
- x[1,1] richiama l'elemento di posto $(1; 1)$

Da questi esempi si possono facilmente ricavare altri casi di estrazione non contemplati.

8.8 Richiamare i nomi delle righe e delle colonne una matrice

Se `x` è una matrice allora con il comando:
`>dimnames(x)`
richiamiamo i nomi delle righe e delle colonne di `x`.

Per richiamare solo i nomi delle righe della matrice dobbiamo usare il comando:
`>dimnames(x)[[1]]`
mentre per richiamare solo i nomi delle colonne dobbiamo eseguire:
`>dimnames(x)[[2]]`

8.9 Le funzioni cbind e rbind

Per aggiungere righe o colonne ad una matrice possiamo utilizzare le funzioni:

- `cbind` con il quale si aggiungono una o più colonne ad una matrice esistente
- `rbind` con il quale si aggiungono una o più righe ad una matrice esistente.

Se `x` è una matrice ed `y,z` sono due vettori che hanno lo stesso numero di righe di `x` con il comando:
`>w<-cbind(x,yyy=y,zzz=z)` si crea la matrice `w` che contiene la matrice `xxx` aumentata di due colonne denominate `xxx,yyy`

Se `x` è una matrice ed `y,z` sono due vettori che hanno come lunghezza lo stesso numero di colonne di di `x` con il comando:
`>w<-rbind(x,yyy=y,zzz=z)` si crea la matrice `w` che contiene la matrice `x` aumentato di due righe denominate `xxx,yyy`

8.10 Trasformare una matrice in un vettore

Può essere utile a volte trasformare una matrice in un vettore. Se `x` è una matrice con il comando:
`>y <- as.vector(x)`
si trasforma la matrice `x` nel vettore `y` seguendo l'ordine delle colonne.

8.11 Operazioni sulle matrici

Come accade per i vettori, anche per le matrici le operazioni aritmetiche e di confronto vengono eseguite elemento per elemento. Se `a e b` sono due matrici che permettono di eseguire le operazioni elementari si avrà che:

- per eseguire la somma
  `>a+b`
- per eseguire il prodotto
  `>a%*%b`
- per calcolare l'inversa
  `>solve(a)`
- per trovare la trasposta `>t(a)`

Se `x` è una matrice mxn ed `y` è un vettore $mx1$ allora con il comando
`x-y`
da ogni colonna di `x` il corrispondente valore del vettore `y`.

8.12 Trovare il determinante di una matrice

Data una matrice `x` il determinante di tale matrice si trova con il comando:
`>det(x)`

8.13 Autovalori ed autovettori di una matrice

Data una matrice `x` gli autovalori e gli autovettori di tale matrice si trova con il comando:
`>eigen(A,symmetric)`
gli autovettori ottenuti in questo modo non sono normalizzati. Per ottenere invece autovettori normalizzati è preferibile usare il comando:
`>La.eigen(A,symmetric)`
Si noti che `symmetric` può assumere valore `T` o `F`. é sempre meglio specificare se siamo in presenza di una matrice simmetrica oppure no nel calcolo degli autovalori ed autovettori di una matrice.

8.14 Aggiungere righe e colonne ad una matrice

Per creare una matrice o aggiungere righe o colonne ad una matrice possiamo utilizzare i comandi

- `cbind` il quale crea una matrice per colonne o aggiunge una o più colonne ad una matrice esistente
- `rbind` il quale crea una matrice per righe o aggiunge una o più righe ad una matrice esistente

Se `x` ed `y` sono vettori di uguale lunghezza con:

- `>w<-cbind(x,y)` si crea una matrice avente come colonne gli elementi dei vettori `x` e `y` e nomi delle colonne `x` ed `y`
- `>w<-rbind(x,y)` si crea una matrice avente come righe gli elementi dei vettori `x` e `y` e nomi delle righe `x` ed `y`
- `>w<-cbind(A,x)` se `A` è una matrice compatibile con il vettore `x` si ottiene una matrice data dalla matrice `A` alla quale è stata aggiunta una nuova colonna composta dagli elementi di `x` e chiamata proprio `x`
- `>w<-rbind(A,x)` se `A` è una matrice compatibile con il vettore `x` si ottiene una matrice data dalla matrice `A` alla quale è stata aggiunta una nuova riga composta dagli elementi di `x` e chiamata proprio `x`

9 Gli array

9.1 Introduzione

Gli array sono costituiti da dati dello stesso tipo che sono graggruppati in tabelle ad entrata multipla ordiante in relazione alle variabili che lo compongono. Gli array come i vettori e le matrici sono fondamentali nell'analisi statistica dei dati in quanto permettono un'analisi congiunta di più di due aspetti di una stessa unità statistica.

9.2 Come creare un'array

Se il vettore `x` contiene i dati da inserire nell'array ordinati secondo le colonne e secondo l'ordine delle matrici da ottenere, per creare un'array possiamo utilizzare la funzione `array` nel seguente modo:
`matrix(x,dim=c(n1, n2, ... nk)`
in cui `n1, n2, ... nk` sono numeri naturali che rappresentano la numerosità delle modalità di ogni singola variabile che compone l'array. Si noti che la lunghezza del vettore deve essere pari al valore di `n1*n2*...*nk` in caso contrario si ottiene un messaggio di errore. Un alto modo per creare un array è il seguente:
`>dim(x)<-c(n1, n2, ..., nk)`
in cui `n1, n2, ... nk` come appena detto, sono numeri naturali che rappresentano la numerosità delle modalità di ogni singola variabile che compone l'array.

9.3 Attributi di un'array

Se `x` è un array applicando i seguenti comandi possiamo ottenere alcune proprietà dell'array stesso:

- `>length(x)` restituisce la lunghezza di x
- `>mode(x)` restituisce il modo di x
- `>dimnames(x)` restituisce i nomi di x
- `>dim(x)` restituisce le dimensioni di x

9.4 Dare un nome agli elementi dell'array

Se `x` è un'array formato dalle variabili `n1 x n2 x ... x nk` è possibile dare un nome alle modalità con cui si presenta ogni singola variabile con il seguente comando:

```
dimnames(x)<-list(c("n11","n12",...,"n1m"),
c("n21","n22",...,"n2n"),
c("....................."),
c("nk1","nk2",...,"nkq"))\\
```

in cui gli elementi fra virgolette sono i nomi che vengono dati alle singole modalità di ogni variabile dell'array.

9.5 Richiamare un array in base ad una modalità

Dato un array `x` formato dalle variabili `n1 x n2 x n3` per richiamare l'array in base allo stato che assume una singola modalità di una variabile dell'array possiamo utilizzare i seguenti comandi:

- `>x[,1,]` stato primo assunto dalal variabile seconda
- `>x[,`"n2i",] stato i assunto dalla variabile seconda

Si può facilmente estendere questi esempi a casi non contemplati.

9.6 Richiamare i nomi delle modalità delle varibili di un'array

Se `x` è un'array allora con il comando:
`>dimnames(x)`
richiamiamo i nomi delle modalità delle singole variabili di `x`.

Per richiamare i nomi delle modalità della i-ma variabile dell'array dobbiamo usare il comando:
`>dimnames(x)[[i]]`

9.7 Trasformare un array in un vettore

Può essere utile a volte trasformare un array in un vettore. Se `x` è un array con il comando:
`>y <- as.vector(x)`
si trasforma l'array `x` nel vettore `y` seguendo l'ordine delle colonne.

9.8 Lavorare con gli array

Vogliamo ne l seguente paragrafo mostrare come con l'uso degli array sia possibile semplificare notevolmente l'analisi statistica dei dati di una certa analisi. Supponiamo di avere a disposizione il seguente array:

```
Titanic
   , , Age = Child, Survived = No

       Sex
   Class  Male Female
   1st     0      0
   2nd     0      0
   3rd    35     17
   Crew    0      0

   , , Age = Adult, Survived = No

       Sex
   Class  Male Female
   1st   118      4
   2nd   154     13
   3rd   387     89
   Crew  670      3

   , , Age = Child, Survived = Yes

       Sex
   Class  Male Female
   1st     5      1
   2nd    11     13
   3rd    13     14
   Crew    0      0
```

```
, , Age = Adult, Survived = Yes

    Sex
Class  Male Female
1st    57    140
2nd    14     80
3rd    75     76
Crew  192     20
```

Notiamo che l'array Titanic presenta le variabili

- class
- sex
- age
- survived

aventi le seguenti modalità:

- class = 1 2 3 4
- sex = m f
- age = child adult
- survived = no Yes

Se vogliamo ottenere un array in cui perdiamo la variabile `age` dobbiamo dare i seguenti comandi:
`>xxx<-Titanic[,,1,]+Titanic[,,2,]`
così facendo otteniamo un array denominato `xxx` avente variabili e modalità date da:

- class = 1 2 3 4
- sex = m f
- survived = no Yes

Digitando ora il seguente comando:
`yyy<-xxx[,1,]+xxx[,2,]`
otteniamo l'array denominato `yyy` in cui perdiamo la variabile sex e che sarà composto dalle seguenti variabili aventi modalità:

- class = 1 2 3 4
- survived = no Yes

Da questo semplice esempio è possibile rendersi immediatamente conto dell'importanza dell'uso degli array nell'analisi dei dati.

10 List

10.1 Introduzione

Le liste sono degli oggetti destinati a contenere più dati anche di diversa natura. Esse sono di notevole importanza quando un comando deve produrre dati sia di tipo numerico che di tipo carattere.

10.2 Come creare una lista

Una lista può essere creata nel seguente modo:
`>x<-rnorm(100) >y<-c`(“stringa1“,“stringa2“,“stringa3“)
`>z<-matrix(rnorm(100),ncol=10)`
`>mylist<-list(nome1=x,nome2=y,nome3=z)`
si ottiene in questo modo una lista di nome mylist e contenente oggetti diversi tra di loro il primo del quale si chiama nome1, il secondo nome2 e il terzo nome3.

10.3 Attributi di una lista

Gli attributi di una lista sono i seguenti:

- `>length(x)` restituisce la lunghezza di x
- `>mode(x)` restituisce il modo di x
- `>names(x)` restituisce i nomi di x

10.4 Dare un nome agli elementi di una lista

Se `x` è una lista composta da tre oggetti per dare dei nomi a tali oggetti o per cambiare i nomi eventualmente già asseganti possiamo usare il seguente comando:
`>names(x)<-c`(“nome1“,“nome2“,“nome3“)

10.5 Richiamare gli elementi di una lista

Se `x` è una lista composta da tre oggetti aventi rispettivamente `nome1, nome2, nome3` per richiamarne i singoli elementi possono usare operare nel seguente modo:

- `>x[[1]]` restituisce il primo elemento della lista
- `x[[1]][n1:n2]` restituisce gli elementi dal n1 al n2 del primo elemento della lista
- `>x$a` richiama il primo elemento della lista
- `x$a[9:12]` richiama gli elementi da n1 ad n2 del primo elemento della lista

Analogamente si opera per gli altri elementi della lista.

11 Factor

11.1 Introduzione

In una indagine statistica può accadere alle volte che i dati di una certa variabile siano raccolti interamente in un unico vettore e solamente tramite un altro vettore della stessa lunghezza del primo è possibile stabilire da quale realtà essi provengono. Ad esempio se volessimo fare un'analisi dei voti riportati in italiano dagli alunni di una scuola sarebbe più agevole raccogliere in una variabile di nome `italiano` i voti riportati dai singoli alunni e creare una variabile `classe` nella quale inseriremo in corrispondenza del voto la classe dell'alunno che ha riportato quel voto. Le variabili che permetto di fare cio' sono dette variabili di tipo `factor`. Gli elementi distinti che compongono l'oggetto factor sono detti `livelli` del fattore. Le variabili di tipo factor permettono quindi di attribuire un dato ad un deteminato livello di un fattore considerato.

11.2 Come creare una variabile factor

Supponiamo di avere a disposizione due variabili `x` e `y` così formate:
`>x<-c`(“a,“b“,“c“,“a“,“a“)
`>y<-c(1,2,3,4,3,2,1,2,3,3,3,4,4,3,2,1,1,1,2,3,4,5,5,5,4,3,2,3)`
per trasformarle in oggetti di tipo factor possiamo operare nel seguente modo:

- `>x<-factor(x)`
  si crea un factor da caratteri che saranno ordinati secondo ordine alfabetico
- `>y<-factor(y)`
  si crea un factor da numeri che saranno ordinati secondo l'ordine naturale
- `>x<-ordered(x),levels=c`(“c“,“b“,“a“))
  si crea un factor da caratteri che saranno ordinati secondo l'ordine dato dall'opzione `levels`
- `>y<-ordered(y),levels=c(5,4,3,2,1))`
  si crea un factor da numeri che saranno ordinati secondo l'ordine dato dall'opzione `levels`
- `>x<-factor(x,levels=c`(a“,“b“))
  gli elementi del vettore con la lettera c evidenzieranno un NA
- `>x<-factor(x,levels=c`(a“,“b“,“c“),labels=c(“CorsoA“,“CorsoB“,“Corso C“))
  si crea un factor con i livelli nominati come scritto nell'opzione `labels`
- `>x<-factor(x,levels=c`(a“,“b“),labels=c(“CorsoA“,“CorsoB“))
  si crea un factor con i livelli nominati come scritto il `labels` tranne quelli con c che appariranno con NA
- `>x<-factor(x,exclude=c`(c“))
  in questo modo escudiamo dalla creazione dei fattori il corso c e dove ci sarà comparirà un NA

11.3 Creare factor da dati continui

E' possibile creare degli oggetti factor da dati continui. Se creiamo un vettore x contenente 100 numeri casuali provenienti da una variabile aleatoria normale di media 10 e varianza 1:

```
>x<-rnorm(100,10,1)
```
con il comando
```
>y<-cut(x,breaks=c(0,2,4,6,10))
>y<-factor(y)
```
E' anche possibile ordinare i dati in modo diverso o escluderne qualcuno utilizzando i comandi visti precedentemente.

11.4 Come creare un factor dicotomico

Se `x` è un vettore che contiene i numeri naturali da 1 a 5 con il comando:
`>y<-factor(x==3)` si crea un factor avente tutti F tranne un T al posto 3. Utilizzando questo procedimento è sempre possibile creare da ogni qualsivoglia vettore dei factor di tipo dicotomico.

11.5 Attributi di un factor

Gli attributi di un factor `x` sono i seguenti:

- `>length(x)` restituisce la lunghezza di x
- `>mode(x)` restituisce il modo di x
- `>names(x)` restituisce i nomi di x
- `>levels(x)` restituisce i livelli di x
- `>class(x)` restituisce la classe di x

11.6 La funzione gl

La funzione `gl` è una importante funzione che permette di creare oggetti factor in modo molto semplice. La sua sintassi è la seguente:
```
>gl(n, k, length = n*k, labels = 1:n, ordered = FALSE)
```
in cui:

- n è un intero che rappresenta il numero dei livelli
- k è un intero che indica il numero delle replicazioni
- length è un intero che restituisce la lunghezza del risultato
- labels è un vettore opzionale di lunghezza n che da i nomi dei livelli del fattore
- ordered è un oggetto logico per stabilire se i fattori devono essere ordinati o no

Utilizzando questa funzione sarà possibile creare replicazioni di livelli nell'ordine che vogliamo per poter risparmiare tempo nell'introduzione di tali oggetti soprattutto in problemi di analisi della varianza.

12 Data Frame

12.1 Introduzione

I `data frame` sono oggetti simili alle matrici ma con colonne che possono contenere dati di diverso tipo. Nel `data frame` sarà allora possibile combinare colonne contenenti numeri con colonne di stringhe o di oggetti factor. E' molto comodo rappresentare i dati provenienti da una indagine statistica in `data fra me` in cui le righe rappresentano l'unità statistica di rilevazione mentre le colonne rappresentano le variabili rilevate nell'indagine stessa.

12.2 Come creare un data frame

Supponiamo di avere a disposizione tre variabili con lo stesso numero di elementi date da:
`>x<-rnorm(6)`
`>y<-c`(“a“,“a“,“b“,“c“,“a“,“c“)
`>z<-1:6`
il data frame `xxx` sarà creato in uno dei seguenti modi:

1. `>xxx<-data.frame(x,y,z)` crea un data frame con tre varibili aventi per nomi `x,y,z`
2. `>xxx<-data.frame(nome1=x,nome2=y,nome3=z)` crea un data frame con tre variabili aventi per nomi `nome1,nome2,nome3`

Se una delle varibile del data frame è di tipo `character` viene automaticamente trasformata in variabile di tipo factor.

12.3 Creazione di un data.frame con la funzione fix

Tramite l'utilizzo della funzione `fix` è possibile procedere alla creazione di un data frame utilizzando la seguente sintassi:
`>xxx<-data.frame() >fix(x)`

12.4 Attributi di un data frame

Se `xxx` è un data frame i suoi attributi sono ottenuti con:

- `>length(xxx)` restituisce la lunghezza di xxx
- `>mode(xxx)` restituisce il modo di xxx
- `>row.names(xxx)` restituisce i nomi delle righe di xxx
- `>dimnames(xxx)` restituisce i nomi sia delle righe che delle colonne di di xxx
- `>dim(xxx)` restituisce le dimensioni di xxx
- `>nrow(xxx)` restituisce il numero delle righe di xxx
- `>ncol(xxx)` restituisce il numero delle colonne di xxx
- `>names(xxx)` restituisce i nomi delle colonne di xxx
- `>class(xxx)` restituisce la classe di xxx

12.5 Dare un nome alle righe e colonne di un data frame

Se `xxx` è una data frame di dimensioni mxn è possibile dare un nome alle sue righe ed alle sue colonne con il seguente comando:

```
dimnames(xxx)<-list(c("nomerig1","nomerig2",...,"nomerigm"),
c("nomecol1","nomecol2",...,"nomecoln"))\\
```

Se vogliamo dare un nome solamente alle sue righe usiamo il seguente comando:

```
dimnames(xxx)[[1]]<-list(c("nomerig1","nomerig2",...,"nomerigm"))
```

Se invece vogliamo dare un noime solamente alle sue colonne usiamo il seguente comando:

```
dimnames(xxx)[[2]]<-list(c("nomecol11","nomecol2",...,"nomecoln"))
```

12.6 Estrarre dati da un data frame

Se `xxx` è un data frame di dimensioni mxn di cui sappiamo che:

- le righe sono state chiamate nomerig1, nomerig2,...,nomerigm
- le colonne sono state chiamate nomecol1, nomecol2, ..., nomecoln

per richiamare i suoi elementi possiamo utilizzare i seguenti comandi:

- `>xxx[,`"nomecol1"]=xxx[,1] richiama la prima colonna di `xxx`
- `>xxx[`"nomerig1",]=xxx[1,] richiama la prima riga di `xxx`
- `>xxx[,c(`"nomecol1","nomecol2")]=xxx[,c(1:2)] richiama le prime due colonne di `xxx`
- `>xxx[,c(1,3)]` richiama la prima e la terza colonna di `xxx`
- `>xxx[c(1,3),c(1,3)]` richiama la prima e la terza riga e colonna di `xxx`
- `>xxx[-1,c(1,3)]` richiama la prima e la terza colonne di `xxx` tranne la riga 1
- `>xxx[1:2,3]` richiama i primi 2 elementi della colonna 3
- `>xxx[x[,1]>=2,1:2]` richiama le colonne da 1 a 2 di `xxx` i cui elementi della prima colonna sono maggiori o uguali a 2
- `>xxx[x[,1]>=2,]` richiama tutte le colonne di `xxx` i cui elementi della prima colonna sono maggiori o uguali a 2
- xxx[1,1] richiama l'elemento di posto $(1; 1)$

Da questi esempi si possono facilmente ricavare altri casi di estrazione non contemplati. Si noterà certamente che questo modo di procedere è identico a quello usato nelle matrici. In un data frame è possibile anche usare una sintassi differente per richimare le singole colonne del data frame stesso. Infatti con:

```
>xxx$nomecol1 si richiama la prima colonna di xxx
>xxx$nomecol5 richiama la quinta colonna di xxx
```

12.7 Estrarre dati da un data frame: casi di notevole interesse

Vogliamo in questo paragrafo esaminare alcuni casi interessanti di estrazione di dati da un data frame. Se `xxx` è un data frame con variabili `var1,var2,var3,var4`, in cui `var1,var2` sono di tipo numerico, mentre `var3` e di tipo factor con livelli rappresentati da caratteri e denominati `var3liv1,var3liv2,var3liv3` e `var4` è ancora di tipo factor ma con livelli rappresentati da numeri `var4liv1,var4liv2,var4liv3` con i seguenti comandi:

```
>xxx[xxx$var4==var4liv1,]      estraiamo dal data frame solo gli elementi
                               che presentano nella variabile 4il livello 1
>xxx[xxx$var3=="var3liv1",]    estraiamo dal data frame solo gli elementi
                               che presentano nella variabile 3 il livello 1
>xxx[xxx$var3=="var3liv2",]    estraiamo dal data frame solo gli elementi
                               che presentano nella variabile 3 il livello 2
>xxx[xxx$var1>numero,]         estraiamo dal data frame solo gli elementi
                               che presentano nella variabile 1 numeri maggiori
                               di numero
>xxx[xxx$var2>=numero,"var1"]  estraiamo dal data frame solo gli elementi
      che presentano nella variabile 2 numeri maggiori
      di numero ma estraiamo solamente i valori della
                               variabile 2
>xxx[xxx$var2>=numero,1]       idem come sopra
>xxx[xxx$var2<numero,c("var1","var3")] estraiamo dal data frame solo
      gli elementi che presentano nella variabile 2
                               numeri maggiori o uguali di numero  ma
      estraiamo solamente i valori della variabile
      1 e variabile 3
>xxx[xxx$var2<,c(1,3)]         idem come sopra
>xxx[,"var1"]                  si ottiene un vettore che riporta solamente
      i valori della var1
>xxx[,1]                       idem come sopra
```

Naturalmente sarà anche possibile applicare gli operatori logici `&` ed `|` per combinare tra di loro più condizioni.

12.8 Richiamare i nomi delle righe e delle colonne un data frame

Se `xxx` è un data frame allora con il comando:
`>dimnames(xxxx)`
richiamiamo i nomi delle righe e delle colonne di `xxx`.

Per richiamare solo i nomi delle righe del data frame dobbiamo usare il comando:
`>dimnames(xxx)[[1]]`
mentre per richiamare solo i nomi delle colonne dobbiamo eseguire:
`>dimnames(xxx)[[2]]`

12.9 Le funzioni cbind e rbind

Per aggiungere righe o colonne ad un data frame possiamo utilizzare le funzioni:

- `cbind` con il quale si aggiungono una o più colonne ad un data frame esistente

- `rbind` con il quale si aggiungono una o più righe ad un data frame esistente.

Se `xxx` è un data frame ed `y,z` sono due vettori che hanno la stessa lunghezza di `xxx` con il comando:
`>www<-cbind(xxx,yyy=y,zzz=z)` si crea il data frame `www` che contiene in data frame `xxx` aumentato di due variabili denominate `xxx,yyy`

Se `xxx` è un data frame ed `y,z` sono due vettori che hanno come lunghezza il numero delle variabili di `xxx` con il comando:
`>www<-rbind(xxx,yyy=y,zzz=z)` si crea il data frame `www` che contiene in data frame `xxx` aumentato di due righe denominate `xxx,yyy`

Si noti in ogni caso che che per aggiungere una colonna ad un data frame `xxx` vi è anche la possibilità di usare la sintassi:
`w<-data.frame(xxx,yyy=y)`
con la quele si aggiunge la varibile `y` con nome `yyy` al data frame `xxx`

12.10 Le funzioni attach e detach

Queste due funzioni sono molto importanti in un data frame in quanto permetto di accedere alle singole variabili che compongono il data frame con una sintassi più semplice e più immediata. Se `xxx` è un data frame con colonne denominate `var1, var2` dopo aver eseguito il comando:
`>attach(xxx)`
potremmo accedere alle variabili `var1` e `var2` senza dover utilizzare la sintassi

```
xxx$var1
xxx$var2
```

ma semplicemente digitando

```
var1
var2
```

Con il comando `>detach(xxxx)` si ripristina la condizione originaria.

12.11 Ordinamento di un data frame

Molte volte cui troviamo nella necessità di dover ordinare un intero data frame facendo riferimento all'ordinamento di una sua variabile. Se `xxx` è un data frame con colonne `var1,var2,var3,var4` dopo aver eseguito il comando:
`>attach(xxx)`
se desideriamo ordinare l'intero data frame rispetto alla variabile `var1` dobbiamo eseguire il comando:
`>xxx[order(xxx[,1]),]`
oppure l'equivalente
`>xxx[order(xxx[,`var1]),]
analogamente si ragiona se l'ordinamento deve essere fatto utilizzando le altre variabili.

12.12 Dati provenienti da una tabella semplice

Può accadere, soprattuto per analisi statistiche avanzate, di dover inserire in un data frame i dati relativi ad una tabella semplice in modo da riportare per ogni dato sia la riga che la colonna da cui questi dati provengono. Supponiamo di disporre di una tabella del tipo:

Blocco \ Trattamento	A	B	C	D
1	89	88	97	94
2	84	77	92	79
3	81	87	87	85
4	87	92	89	84
5	79	81	80	88

Per inserire tali dati nel modo precisato in premessa in un data frame denominato `xxx` possiamo operare nel seguente modo:

- prima di tutto inseriamo i dati numerici in ordine di colonna nella variabile dati
- quindi creiamo la variabile trattamento con
  `>trattamento<-rep(paste(`“T“,LETTERS[1:4]),c(5,5,5,5))
  oppure con
  `>trattamento<-rep(paste(`“T“,LETTERS[1:4]),rep(5,4))
- quindi creiamo la variabile blocco con
  `>blocco<-rep(paste(`“blocco “,1:5),4)
- quindi creiamo il data.frame con
  `>xxx<-data.frame(dati=dati,trattamenti=trattamento,blocchi=blocco)`

12.13 Dati provenieti da una tabella con più entrate di valori

Può accadere che i dati provenienti da una stessa riga e una stessa colonna siano più di uno. Ci troviamo allora nella necessità di dover inserire in un data frame che chiameremo `xxx` più dati per ogni riga e ogni colonna. Supponiamo di disporre di una tabella del tipo:

Blocco \ Trattamento	A	B	C	D
1	0.31	0.82	0.43	0.45
	0.45	1.10	0.45	0.71
	0.46	0.88	0.63	0.66
	0.43	0.72	0.76	0.62
2	0.32	0.72	0.53	0.15
	0.15	1.16	0.15	0.11
	0.26	0.08	0.03	1.66
	0.93	0.72	0.16	2.62
3	0.21	0.42	0.03	0.85
	0.05	1.30	0.15	0.41
	0.06	0.28	0.93	0.16
	0.03	0.12	0.16	0.02

Per inserire tali dati in un data frame chiamato `xxx` nel modo precisato in premessa possiamo operare nel seguente modo:

- prima di tutto inseriamo i dati numerici in ordine di colonna nella variabile dati
- quindi creiamo la variabile trattamento con
  `>trattamento<-rep(paste(`“T“,LETTERS[1:4]),rep(12,4))

- quindi creiamo la variabile blocco con
  `>xxx<-rep(paste(`“blocco “,1:3),rep(3,3))
  e quindi
  `>blocco<-c(xxx,xxx,xxx,xxx)`
  oppure con
  `>blocco<-rep(xxx,4)`
- quindi creiamo il data.frame con
  `>xxx<-data.frame(dati=dati,trattamenti=trattamento,blocchi=blocco)`

13 Testing e coercing data

13.1 Introduzione

Molte volte ci si trova nella necessità di dover verificare se un determinato oggetto appartiene ad un tipo specificato oppure di convertire il tipo di oggetto in un'altro. Per compiere queste operazioni possiamo utilizzare i comandi descritti nei paragrafi successivi.

13.2 Testare e convertire oggetti

Per testare e convertire gli oggetti utilizzati dal programma possiamo utilizzare i seguenti comandi:

- is.array(x)
- as.array(x)
- is.complex(x)
- as.complex(x)
- is.data.frame(x)
- as.data.frame(x)
- is.double(x)
- as.double(x)
- is.factor(x)
- as.factor(x)
- is.integer(x)
- as.integer(x)
- is.list(x)
- as.list(x)
- is.logical(x)
- as.logical(x)

- is.matrix(x)
- as.matrix(x)
- is.na(x)
- is.null(x)
- as.null(x)
- is.numeric(x)
- as.numeric(x)
- is.ts(x)
- as.ts(x)
- is.vector(x)
- as.vector(x)

Si noti in particolare che con l'uso di:

- `is.` vogliamo verificare se un certo oggetto sia di un certo tipo
- `as.` forziamo un oggetto ad essere di un tipo specificato

14 Uso di alcune funzioni notevoli

14.1 Introduzione

Vogliamo in questi paragrafi analizzare alcune funzioni di notevole importanza soprattutto nell'analisi dei dati contenuti in un data frame.

14.2 Richiesta di una funzione in un comando

In certi casi sarà necessario costruire una funzione per applicarla ai dati che si hanno a disposizione. Si possono presentare due casi:

- la funzione che vogliamo applicare ai dati è già presente in R o perchè predefinita o perchè già creata in precedenza: in questo caso basterà digitare il nome della funzione stessa quando necessario
- la funzione non è presente in R e quindi dovrà essere inserita tramite, ad esempio, con la sintassi:
  `function(x) x^2`

Inoltre ricordiamo che tramite il comando:
`na.rm=T`
facciamo in modo che la funzione ignori i valori `NA`, in caso contrario in presenza di tali valori errebe generato un errore.

14.3 La funzione aggregate

Con `aggregate` possiamo applicare una funzione contemporaneamente a più varibili inserite in un un data frame e di applicare contemporaneamente tale funzione anche a sottoinsiemi in cui eventualmente tali varibili sono suddivise. Se `xxx` è un data frame con colonne ordinate `var1,var2,var3,var4` con il seguente comando:

```
>aggregate(xxx[,c("var1","var2")],by=list(x$var3,x$var4),FUN=mean)
```

otteniamo la media delle variabili `var1, var2` suddivise in base ai valori assunti dalle variabili `var3, var4`. Di solito le variabili `var3, var4` sono di tipo factor.

14.4 La funzione apply

Con `apply` possiamo applicare una funzione alle righe o alle colonne di un data frame o di una matrice che indicheremo con `xxx`. Il suo uso è il seguente:

- `>apply(xxx,1,mean)` calcola la media di tutte le righe
- `>apply(xxx,2,mean)` calcola la media di tutte le colonne
- `>apply(xxx,2,mean,na.rm=T)` escude dalla media i valori NA

Nell'uso di tale funzione bisognerà fare attenzione, nel caso di un data frame, della natura delle singole varibili che lo compongono. Non è infatti possibile sommare o fare medie tra numeri e caratteri.

14.5 La funzione tapply

Con `tapply` possiamo applicare una funzione ad una variabile di un data frame facendo riferimento a eventuali sottoinsiemi in cui tale varibile e suddivisa e rappresentati da variabili di tipo `factor`. La sua sintassi è la seguente:

- `>tapply(x,factor,mean)`
- `>tapply(x,list(factor1,factor2),mean)`
- `>tapply(x,list(factor1,factor2),var)`

Si noti che factor può anche essere un oggetto numeric ed in questo caso sarà automaticamente convertito in oggetto factor. Si può usare dove necessario l'opzione `na.rm` come visto precedentemente. Tale funzione è analoga alla funzione aggregate solamente che è applicata ad una sola varibile a differenza di aggregate che può essere applicato a più variabili contemporaneamente.

14.6 La funzione lapply

Con `lapply` possiamo applicare una funzione ad un vettore, lista o data frame di dati. Il risultato sarà `una lista` che conterrà il valore della funzione applicato ad ogni elemento dell'oggetto considerato. Ad esempio con:

```
>x<-seq(-3,3,0.1)
>lapply(x, function(x) pnorm(x))
```

si ottiene una lista composta da 61 elementi.

14.7 La funzione sapply

Con `sapply` possiamo applicare una funzione ad un vettore, lista o data frame di dati. Il risultato sarà `un vettore` o `un array` che conterrà il valore della funzione applicato ad ogni elemento dell'oggetto considerato. Ad esempio con:

```
>x<-seq(-3,3,0.1)
>lapply(x, function(x) pnorm(x))
```

si ottiene `un vettore` composta da 61 elementi.

14.8 La funzione paste

Con `paste` possaimo unire stringhe con numeri o altro. Ad esempio con :

```
>paste(letters[1:5],1:5,sep=" ")
```

otteniamo come risultato:

```
>[1] a 1  b 2  c 3  d 4  e 5
```

Se non si vuole la separazione tra le lettere e i numeri basterà semplicemente nell'opzione `sep` introdurre le virgolette senza lo spazio.

14.9 La funzione split

Con `split` possiamo ottenere dati suddivisi per gruppi a seconda di uno o più oggetti factor specificati. Se `xxx` è un data frame con variabili `var1,var2,var3,var4` e le variabili `var3, var4` sono di tipo factor, con il comando:

```
>split(xxx$var1,list(xxx$var3))
```

otteniamo i vettori della `var1` suddivisi in base ai valori della `var3`. Se invece usiamo:

```
>split(xxx,list(xxx$var3,xxx$var4))
```

otteniamo più data frame suddivisi sia per `var3` che per `var4`. I risultati sono ottenuti sotto forma di oggetti di tipo list.

14.10 La funzione stem

Con `stem` si ottiene un grafico che permette di fare delle considerazioni preliminari sui dati di una analisi contenuti in un vettore. Se `x` è un vettore che contiene i dati da analissare possiamo utilizzare la seguetne sintassi:

```
>stem(x)
>stem(x,scale=2)
```

14.11 La funzione summary

Con `summary` otteniamo dei dati di riepilogo su molti oggetti utilizzati dal programma stesso. La sua sintassi è la seguente:

```
>summary(x)
```

14.12 La funzione tabulate

Con `tabulate` possiamo ottenere le frequenze assolute con cui una certa modalità di una certa variabile compare nel vettore in cui abbiamo raccolto i dati della nostra analisi. Se abbiamo a disposizione un vettore `x` di dimensione `n` con la funzione:

```
tabulate(x)
```

otteniamo un vettore che raccoglie le frequenze assolute delle modalità con cui la variabile `x` si presenta.

14.13 La funzione fivenum

Con `fivenum` otteniamo i cinque numeri fondamentali relativi ad una singola variabile di tipo quantitativo. Se abbiamo a disposizione un vettore `x` di dimensione `n` con la funzione:
`fivenum(x)`
otteniamo come risultato un vettore formato dal minimo, primo quartile, mediana, terzo quartile e massimo calcolati sui valori del vettore argomento della funzione stessa.

14.14 La funzione which

Con `which` possiamo conoscere gli indici degli elementi di un vettore che soddisfano ad una certa condizione. Se `x` è un vettore numerico, con il comando:
`which(x>7)`
vengono individuati gli indice degli elementi del vettore che soddisfano la condizione data.

14.15 La funzione unique

Con `unique` vengono individuate le modalità con cui una certa varibile si presenta. Se `x` è un vettore la funzione verrà eseguita con il seguente comando:
`unique(x)`

14.16 Uso della funzione subset

Dato un data frame `xxx` se vogliamo ottenere il data frame condizionato al valore assunto da una sua variabile dobbiamo utilizzare il comando
`subset(nomedataframe,variabile==valore)` In cui nomedataframe è il nome del data frame da considerare, variabile è il nome della variabile e valore è il valore che deve assumenre la variabile. Si noti che l'operatore relazionale `==` può assumenre qualunque valore relazionale desiderato.

15 Importare i dati in R

15.1 Introduzione

Per poter compiere una analisi statistica dei dati è necessario caricare questi dati in R. Se i dati sono in numero ridotto ciò può essere fatto utilizzando le funzioni:

- `scan` per i vettori
- `fix` per i data frame

Molte volte i dati da analizzare sono già disponibili in file generati da programmi esterni ad R e quindi disponibili in formati di diverso tipo. L'importazione dei dati, in questo caso, può avvenire in diversi modi come documentato nell'help delle funzioni dedicate allo scopo ossia:
`>read.table`
`>read.csv`
`>read.csv2`

```
>read.delim
>read.delim2
>scan
```

Si ricorda in ogni caso che i file sono importabili in R se sono in formato testo, esistono però innumerevoli tipi di formato testo a seconda del modo in cui i record e i campi sono tra di loro separati. Nei prossimi paragrafi analizzeremo solamente l'importazione di alcune tipologie tra le più diffuse di file testo. Se i dati a disposizione sono in formato diverso da quelli trattati, sarà possibile importarli in R consultando l'help delle funzioni sopra descritte.

15.2 La preparazione dei dati

Se i file provengono da programmi della famiglia dei prodotti office come office, staroffice o openoffice, prima di procedere alla loro importazione sarà necessario operare una preparazione preliminare consistente in:

- eliminazione di ogni tipo di formattazione dei dati in particolare alla suddivisione in migliaia delle cifre dei numeri da importare, questa operazione è della massima importanza infatti se lasciamo i punti di separazione i dati non saranno importati in R correttamente
- una volta eliminata la formattazione per evitare problemi legati ad una diversa interpretazione dell'area dei dati da importare sarà consigliabile copiare in un nuovo foglio di lavoro i dati puliti ottenuti eliminando la formattazione e lavorare con questo nuovo foglio di lavoro
- nella prima riga di ogni colonna dovrà essere indicato, preferibilmente in minuscolo, il nome della variabile
- per i dati di tipo numerico procedere alla loro riformattazione come numeri

15.3 La funzione read.table

La funzione read table consente di importare i dati specificando tra l'altro sia il tipo di separatore di campo che il carattere utilizzato per il separatore decimale. Per utilizzare questa funzione sarà quindi necessario conoscere il separatore di campo utilizzato e il formato per la separazione dei numeri decimali utilizzato. Se i dati sono disponibili in file generati dai prodotti della famiglia office, effettuata l'operazione preliminare sui dati, per portare gli stessi un formato testo, leggibile da read table possiamo:

- se si usa il programma office i dati dovranno essere salvati nel formato `testo delimitato` da tabulazione.
- se si usano i programmi staroffice o openoffice i dati dovranno essere salvati nel formato `csv` usando come separatore di campo `tab`.

In questo modo viene generato un file formato testo delimitato da tabulazione e in cui il separatore decimale è la virgola. Si noti che i programmi office permettono di salvare dati in formato testo anche con modalità di tipo diverso come vedremo successivamente.

La funzione `read.table`, come detto, permette di importare un file salvato nel formato `testo delimitato da tabulazione` in un `data.frame`. Supponendo che il file da importare già convertito in tale formato e sia denominato `xxx.txt` per importarlo si usa la seguente sintassi:

```
\verb">xxx<-read.table("c:\\mydir\\xxx.txt",header=T,sep="\t",dec=",")
```

in cui:

- `header=T` significa che la prima riga del file sarà destinata ai nomi delle colonne del data frame.
- `dec=“,“` significa che i dati numerici da importare hanno come separatore decimale la virgola.

15.4 Uso della funzione read.csv

Effettuata l'operazione preliminare sui dati, per portare gli stessi in formato testo, possiamo se si utilizza office salvarli nel formato `csv`. La funzione `read.csv` permette di importare un file salvato nel formato `csv` delimitato dal separatore di elenco in un `data.frame`. Supponendo che il file da importare già convertito in csv delimiatato dal separatore di elenco e sia denominato `xxx.csv` per importare i dati si usa la seguente sintassi:

```
>xxx<-read.csv("c:\\mydir\\xxx.csv")
```

15.5 Uso della funzione read.csv2

Effettuata l'operazione preliminare sui dati, per portare gli stessi in formato testo, possiamo se si utilizza office salvarli nel formato `csv msdos`. La funzione `read.csv2` permette di importare un file salvato nel formato `csv msdos` in un `data.frame`. Supponendo che il file da importare già convertito in csv msdos e sia denominato `xxx.csv` per importare i dati si usa la seguente sintassi:

```
>xxx<-read.csv2("c:\\mydir\\xxx.csv")
```

15.6 Conclusioni

L'importazione dei dati è una operazione alquanto complessa. Quando si ha a disposizione un file in formato testo dobbiamo conoscere e sattamente in che modo sono stati separati i record e i campi e il formato utilizzato per la virgola decimale. Conoscendo tali informazioni e con lievi modifiche a quanto scritto ei paragrafi precedenti sarà sempre possibile importare un file in formato testo contenente dati da analizzare in R.

16 Esportare i dati da R

16.1 Introduzione

Molte volte dopo aver eseguito delle analisi statistiche sarà necessario esportare i dati così ottenuti da R in un altro programma di solito un prodotto della famiglia office. Per fare questo abbiamo bisogno di una funzione che ci permetta di esportare i dati in un formato leggibile da tali prodotti. La funzione che possiamo utilizzare è `write.table`.

16.2 La funzione write.table

La funzione write table converte un qualunque oggetto di R in un file di testo importabile in programmi esterni o in qualche editor di dati. La sintassi di tale comando è la seguente:

```
write.table(x, file = "c:\\miofile.txt", row.names = TRUE, col.names = TRUE,
sep="",quote=T,dec=",")
```

in cui:

- con tale comando sono esportati anche i nomi delle righe e delle colonne, per non esportarli combiare in F nelle opzioni row.names e col.names
- tutti i dati sono esportati tra virgolette, per evitare questo fatto basterà cambiare il F l'opzione relativa a quote
- se vogliamo che i dati siano salvati nel formato testo delimitato da tabulazione bisognerà utilizzare

  `"sep="\t"`

- se vogliamo che i dati presentino come separatore decimale la virgola bisognerà inserire l'opzione dec=,

17 Le tavole in R

17.1 Introduzione

Molto spesso se i dati sono disponibili in vettori sarà necessario creare delle tabelle che ci permettano di ottenere le frequenze assolute o relative delle modalità con cui i dati si presentano nel singolo vettore o in vettori congiuntamente considerati. Per fare ciò si usa la funzione `table` o la funzione `ftable`.

17.2 Uso delle funzioni table e ftable

La funzione table è una funzione molto potente di R e permette di calcolare:

- se applicata ad un vettore le frequenze assolute delle modalità presenti nel vettore
- se applicata a due vettori una tabella di contingenza con le frequenze assolute delle modalità congiunte dei due vettori
- se applicata a più vettori un array con le frequenze assolute congiunte di tutti i vettori

La funzione si usa con il seguente comando:

```
>table(x)
```

in cui `x` è un vettore di dati.

Si noti che la funzione table si presta bene ad essere utilizzate con la funzione cut. Supponiamo di creare con il comando:

```
>x<-runif(1000,1,100)
```

mille numeri casuali tra 1 e 1000 e di voler vederne una tabella degli stessi ma raggruppati di 5 in 5. Ciò puo' essere effettuato con:
`>table(cut(x,breaks=(0+5*(1:20))))` in questo modo otteniamo una tabella delle frequenze di 5 in 5 dei numeri inseriti nel vettore x.

Se `x,y` sono due vettori di dati, per ottenere una tavola di contingenza dobbiamo operare nel seguente modo:
`>table(x,y)`

Per utilizzare la funzione `table` con più vettori possiamo usare la seguente sintassi:
`>table(x,y,z,....)`
in cui `x,y,z,....` ottenendo come già detto un array.

Molto usata in R è anche la funzione `ftable` la quale ha la stessa sintassi di `table` ma fornisce le tabelle in modo diverso. La sua sintassi è la seguente:
`>ftable(x)`
`>ftable(x,y)`
`>ftable(x~y)`
`>ftable(x,y,z,....)` in cui `x,y,z,...` sono dei vettori.

17.3 Uso della funzione prop.table

La funzione `prop.table` prende come argomento o una matrice o un oggetto creato con table e ne restituisce le frequenze relative. La sua sintassi è la seguente:
`>prop.table(xxx,n)`
in cui

- `xxx` è una matrice o un oggetto table
- n=1 viene calcolata la frequenza relativa sul totale generale
- n=2 viene calcolata la frequenza relativa per riga
- n=3 viene calcolata la frequenza relativa per colonna

17.4 Uso della funzione margin.table

La funzione `margin.table` prende come argomento o una matrice o un oggetto creato con table e ne restituisce le frequenze assoluta. La sua sintassi è la seguente:
`>margin.table(xxx,n)`
in cui

- `xxx` è una matrice o un oggetto table
- n non è specificato viene calcolata la somma di tutti gli elementi della tabella
- n=1 viene calcolata la frequenza assoluta per riga
- n=2 viene calcolata la frequenza assoluta per colonna

17.5 La funzione plot con un oggetto table

La funzione `plot` applicata ad un oggetto table o una matrice consente di ottenere un grafico del tipo `mosaicplot`. La sua sintassi è la seguente:
`>plot(xxx)`
in cui `xxx` è un oggetto table o una matrice

17.6 La funzione summary con un oggetto table

Se applichiamo la funzione summary su un oggetto table
`>summary(xxx)`
con `xxx` oggetto table:
si ottiene il test `chi quadrato` di contingenza sulla tabella data.

17.7 La funzione barplot con un oggetto table

Se `xxx` è un oggetto table, possiamo appliocare ad esso il comando grafico `barplot` in uno dei seguenti comandi:
```
>barplot(x)
>barplot(x,beside=T)
>barplot(x,legend=T)
```
ottenendo un grafico di tipo `barplot`.

18 Elementi di programmazione in R

18.1 Introduzione

Pur presentando innumerevoli funzioni, il programma R ci consente sia di creare funzioni personalizzate che degli script ossia delle successioni di istruzioniche permettono di eseguire in modo rapido una serie di comandi senza bisogno di digitarli dalla tastiera. Per fare ciò dobbiamo conoscere alcuni elementi di programmazione in R. Si noti che ogni istruzione o ogni comando visto precedentemente può essere inserito in un listato di programmazione nello stesso modo in cui noi lo abbiamo trattato. Per questo motivo vedremo successivametne solo alcuni aspetti particolari della programmazione rimandando alla documentazione in linea offerta da R gli ulteriori approfondimenti.

18.2 Come scrivere una funzione

Per scrivere e quindi memorizzare unafunzione conviene utilizzare la seguente successione di comandi:
```
>nomefunzione<-function(var1=val1,var2=val2,var3=val3,...,varn=valn){}
>nomefunzione<-edit(nomefunzione)
```
in cui:

- `nomefunzione` è il nome assegnato alla funzione
- `(var1=val1,var2=val2,var3=val3,...,varn=valn)` sono le variabili indipendenti con il relativo valore die default che la funzione dovrà utilizzare. Non è necessario impostare il valore predefinito anche se in certi casi ciò è molto utile

- `{ }` è il corpo della funzione ossia le istruzioni che devono essere eseguite al richiamo del nome delal funzione.

Un modo alternativo per creare una funzione prevede l'utilizzo della funzione `fix(x)` nel seguente modo:
`>nomefunzione<-fix(nomefunzione)`
in questo nodo si aprirà automaticamente l'editor di testo e sarà possibile inserire sia le variabili che il corpo della funzione.

18.3 Come scrivere gli script

Gli sript sono scritti in modo analogo alle funzioni solamente che non prevedono l'inserimentodi alcuna variabile. Per scriverli utilizzeremo allora la seguente successione di comandi:
```
>nomescript<-function(){ }
>nomescript<-edit(nomescript)
```

18.4 Come eseguire una funzione

Una volta creata la funzione per eseguirla dobbiamo semplicemente richiamarla con il nome seguito dal nome e dal valore delle variabili nel seguente modo:
`>nomefunzione(var1=val1,var2=val2,var3=val3,...,varn=valn)`
Se non inseriamo nome e valore di una variabile ad essa automaticamente sarà passato il valore di default.

18.5 Come eseguire uno script

Essendo gli script delel funzioni senza varibili, la loro esequzione avviene nel seguente modo:
`>nomescript()`

18.6 Le strutture di controllo

Qualunque linguaggio di programmazione prevede l'utilizzo delle strutture di controllo. Tali strutture sono le seguenti:

- if
- repeat
- while
- for
- switch

Vedremo nei paragrafi successivi il loro uso.

18.7 La struttura di confrollo if

Il suo uso è il seguente:

```
if(condizione){ }
else{ }
```

18.8 La struttura di confrollo repeat

Il suo uso è il seguente:

```
repeat{ }
```

18.9 La struttura di confrollo while

Il suo uso è il seguente:

```
while(condizione){ }
```

18.10 La struttura di confrollo for

Il suo uso è il seguente:

```
for(i in 1:n){ }
```

18.11 La struttura di confrollo switch

Il suo uso è il seguente:

```
switch(variabilecontrollo,istr1,istr2,istr3,...istrn)
```

18.12 Esempio di programmazione: equazione di secondo grado

Una semplice funzione per ottenere le soluzioni di una equazione di secondo grado è la seguente:

```
secgrad<-function(a,b,c)
           {
              xx<-0
              delta<-b^2-4*a*c
              if(delta<0)
               return("Equazione impossibile")
              else{
xx[1]<--b+sqrt(delta))/(2*a)
               xx[2]<--b-sqrt(delta))/(2*a)}
           }
```

Una versione alternativa potrebbe essere la seguente:

```
secgrad<-function(a,b,c)
          {
              delta<-b^2-4*a*c
              if(delta<0)
               return("Equazione impossibile")
              else{
x1<--b+sqrt(delta))/(2*a)
                x2<--b-sqrt(delta))/(2*a)
                list(x1=x1,x2=x2)}
          }
```

18.13 Esempio di programmazione: indici di connessione

Una semplice funzione per ottenere gli indici di conessione dato un vettore di dati è la seguente:

```
connessione<-function (x)
{
      options(warn=-1)
      dimensione<-dim(x)
      chis<-chisq.test(x)$statistic
      names(chis)<-c()
      n<-sum(x)
      chiquadrato<-1/(n*min(dimensione-1))*chis
      y<-chisq.test(x)$expected
      mortara<-1/(2*n)*sum(abs(x-y))
      options(warn=0)
     list(chiquadrato=chiquadrato,mortara=mortara)
         }
```

18.14 Esempio di programmazione: indici di mutabilità

Una semplice funzione per ottenere gli indici di mutabilità dato un vettore di dati è la seguente:

```
mutability<-function (x)
{
        h<-length(x)
        hh<-sum(x)
        p<-x/hh
        som<-sum(p*(1-p))
        gini<-h/(h-1)*som
        shannon<--sum(p*log(p))/log(h)
        list(gini=gini,shannon=shannon)
}
```

18.15 Esempio di programmazione: le medie

Una semplice funzione per ottenere le medie di un dato vettore di dati è la seguente:

```
medie<-function (x, r)
{
       if (r == 0) {
                                prod(x)^(1/length(x))
                               }
       else {
                         pd<-r-trunc(r/2)
                         if(pd==0){
                                    mean(x^r)^(1/r)
                                  }
                         else{
```

```
                                segno<-sign(sum(x^r))
                                valore<-abs(mean(x^r))^(1/r)
                                valore*segno
                              }
          }
}
```

19 Come creare propri file di funzioni in R

19.1 Introduzione

Una volta create funzioni e script, avremmo la necessità di creare anche dei file per memorizzarle e che siano leggibili da R in modo tale da poterle anche importare ed utilizzare successivamente o di permettere anche ad altre persone di utilizzarle. Vefremo nei paragrafi successivi come creare questi file.

19.2 I file source

I file necessari per compiere queste operazioni sono i file `source`. Tali file raccolgono semplicemente una dietro l'altra le funzioni e gli script che abbiamo creato. Ad esempio il file `personal.r` contiene le funzioni e gli script:

- connessione per l'analisi della connessione
- mutability per l'analisi della mutabilità
- limite.centrale per l'analisi grafica del limite centrale
- intervalli per l'analisi grafica degli intervalli di confidenza
- medie per il calcolo delle medie potenziate
- lotto per l'estrazione casuale di numeri al lotto
- plotline per la visualizzazione di un grafico particolare

e sarà così composto:

```
  connessione<-
function (x)
{
    options(warn=-1)
    dimensione<-dim(x)
    chis<-chisq.test(x)$statistic
    names(chis)<-c()
    n<-sum(x)
    chiquadrato<-1/(n*min(dimensione-1))*chis
    y<-chisq.test(x)$expected
    mortara<-1/(2*n)*sum(abs(x-y))
    options(warn=0)
```

```
    list(chiquadrato=chiquadrato,mortara=mortara)
    }

mutability<-
function (x)
{
      h<-length(x)
      hh<-sum(x)
      p<-x/hh
      som<-sum(p*(1-p))
      gini<-h/(h-1)*som
      shannon<--sum(p*log(p))/log(h)
      list(gini=gini,shannon=shannon)
}

medie<-
function (x, r)
{
     if (r == 0) {
         prod(x)^(1/length(x))
     }
     else {
            pd<-r-trunc(r/2)
            if(pd==0){
                mean(x^r)^(1/r)
               }
            else{
               segno<-sign(sum(x^r))
               valore<-abs(mean(x^r))^(1/r)
               valore*segno
                }
          }
}


intervalli<-
function(media=4,sqm=1,alfa=0.05,nN=100,nn=20)
{
    m.sim<-0
    for(i in 1:nN)
       {
        sim<-rnorm(nn,media,sqm)
        m.sim[i]<-mean(sim)
       }
    zeta<-qnorm(1-alfa/2,0,1)
    lim<-matrix(0,2,nN)
    for(i in 1:nN)
```

```
      {
        lim[1,i]<-m.sim[i]-zeta*sqm/sqrt(nn)
        lim[2,i]<-m.sim[i]+zeta*sqm/sqrt(nn)
      }
    y<-seq(1:nN)
    y<-matrix(y,nN,2)
    lim2<-t(lim)
    plot(y,lim2)
    for(i in 1:nN)
      lines(y[i,],lim2[i,])
    lines(c(0,100),c(media,media))
    vero<-matrix(0,nN,2)
    vero<-c(lim2[,1]>media,lim2[,2]<media)
    sum(vero)

}

limite.centrale<-
function(x,n=100,min=0,max=1)
{
dd<-0
for(i in 1:x)
dd[i]<-mean(runif(n,min,max))
dd<-(dd-(min+max)/2)/((max-min)/sqrt(12*n))
aaa<-hist(dd,plot=F)$breaks
par(mfrow=c(2,2))
hist(dd,main="Istogramma delle frequenze assolute")
hist(dd,prob=T,main="Istogramma delle frequenza relative")
irq<-summary(dd)[5]-summary(dd)[1]
lines(density(dd,width=irq))
hist(dd,prob=T,main="Distribuzione media campionaria")
bbb<-seq(min(aaa),max(aaa),0.01)
lines(bbb,dnorm(bbb,0,1))
plot(ecdf(dd),verticals=T,do.p=F,main="Fz distr. emp. e rip. N(0,1)")
lines(sort(dd),pnorm(sort(dd),mean=0,sd=1))
par(mfrow=c(1,1))
}

lotto<-
function (n)
{
     x<-trunc(runif(1,0,10))+1
     y<-trunc(runif(n,0,90))+1
     z<-list(ruota=x,numeri=y)
     z
}
```

```
plotline<-
function(x,y)
{
     yy<-unique(y)
     l<-length(yy)
     media<-tapply(x,y,mean)
     sqm<-sqrt(tapply(x,y,var))
     z<-sort(media)
     z<-sort(sqm)
     plot(yy,media,xlab="Fattori",ylab="Valori",ylim=c(media[1]-2*sqm[l],
     media[l]+2*sqm[l]),
     main="variabilita' delle medie")
     rug(y)
     for (i in 1:l)
{
           segments(yy[i],media[i]-sqm[i],yy[i],media[i]+sqm[i])
        }

}
```

19.3 Importare i file source

Per importare i file source basterà digitare il seguente comando:
`>source(`directory/filesource.r)
Le funzioni in esso inserite saranno richiamabili nel modo descritto in precedenza. In windows sarà possibile richiamare tali file direttamente dal menu.

20 I grafici tradizionali

20.1 Introduzione all'uso dei grafici in R

In R abbiamo varie tipologie di grafici che verranno trattati nei paragrafi seguenti.

20.2 Il comando plot

Il comando `plot` è il comando base per l'analisi grafica dei dati. Se x ed y sono due vettori di uguale dimensione, con

- `>plot(x)` otteniamo un grafico avente per ordinate gli elementi del vettore x e per ascissa un vettore numerato da 1 a length(x)
- `>plot(x,y)` otteniamo un grafico formato dai punti avente come ascissa gli elementi di x e ordinata gli elementi di y

20.3 Opzioni del comando plot

Una volta digitato il comando plot, possiamo aggiungere allo stesso usando la sintassi:
`>plot(x,y,....)`
al posto dei puntini una serie di opzioni, separate da una virgola, che servono per personalizzare il grafico ottenuto. Esse sono le seguenti:

- `main=“ “` crea un titolo nel grafico
- `sub=“ “` crea un sottotitolo nel grafico
- `xlab=“ “` crea un nome dell'asse delle x
- `ylab=“ “` crea nome dell'asse delle y
- `xlim=c(valore1, valore2)` traccia l'asse del x tra valore1 e valore2
- `ylim=c(valore1, valore2)` traccia l'asse dell y tra valore1 e valore2
- `log=“x“,log=“y“,log=“xy“` traccia i grafici con scala logaritmica come da opzioni
- `type=“p“` traccia il grafico per punti
- `type=“l“` traccia il grafico per linee
- `type=“b“` traccia il grafico per linee e punti
- `type=“o“` traccia il grafico per linnee e punti
- `type=“h“` traccia il grafico con un High-density plot
- `type=“s“` traccia il grafico con un stairstep
- `type=“n“` non traccia nulla

Un'altra serie di opzioni che si possono inserire sempre nello stesso modo riguarda invece la formattazione del grafico ossia il tipo, il colore e la grandezza delle linee o dei punti usati nel grafico stesso. Tali opzioni si ricordi però che agiscono sia sul grafico che sugli assi dello stesso e sono:

- `lty=1` traccia il grafico con linea continua
- `lty=2` traccia il grafico con linea con punti
- `lty=3` traccia il grafico con linea con punti e tratteggiata
- `lty=4` traccia il grafico con linea con tratteggio
- `lty=5` traccia il grafico con linea con tratteggio e tre punti ogni tanto
- `lty=6` traccia il grafico con linea con due tipi di tratteggio
- `lty=7` traccia il grafico con linea con tratteggio a punti
- `lty=8` traccia il grafico con linea con tratteggio piccolo
- `pch=“ “` stampa il tipo di carattere specificato tra virgolette, deve essere usato solo con l'opzione `type=“p“` che peraltro è la predefinita se usiamo pch. Si noti che è anche possibile l'uso con caratteri predefiniti, puntiti, triangoli, ecc utilizzando la sintassi `pch=“ numero“` in cui numero è un naturale
- `col=1....` colora il grafico
- `axes T o F` stampa o no gli assi coordinati
- `lwd=1,....` gestisce la grossezza delle linee e degli assi.

Se vogliamo ottenere solamente la formattazione del grafico e non quella degli assi dovremmo usare tali opzioni con i comandi lines e points

Si noti che in ogni caso si potrà utilizzare il comando plot senza le opzioni precedentemente scritte e successivamente con il comando:
`>title(main=“...“,sub=“...“,xlab=“...“,ylab=“...“)`
possiamo aggiungere successivamente il titolo, il sottotitolo e le etichette degli assi x ed y.

20.4 Aggiungere una legenda ad un grafico generato con plot

In alcuni casi è necessario dopo aver usato il comando plot inserire una legenda sul grafico stesso. Questo può essere fatto in molti modi alternativi e legati al modo in cui si è costruito il grafico con plot. Vediamo in dettaglio vari esempi:

- grafico creato con linee usando l'opzione lty. In questo caso potremmo usare la seguente successione di comandi:
  `>plot(x,y,type=“l“,lty=1)`
  `>legend(locator(1),legend=“.....“,lty=1)`
  cliccando su qualunque punto del grafico otterremo come legenda ciò che è racchiuso tra virgolette
- grafico creato con punti usando l'opzione pch. In questo caso potremmo usare la seguente successione di comandi:
  `>plot(x,y,type=“p“,pch=“...“)`
  `>legend(locator(1),legend=“.....“,pch=“...“)`
  cliccando su qualunque punto del grafico otterremo come legenda ciò che è racchiuso tra virgolette

- grafico creato con linee usando l'opzione lwd. In questo caso potremmo usare la seguente successione di comandi:
  `>plot(x,y,type=`"l",lwd=1)
  `>legend(locator(1),legend=`".....",lwd=1)
  cliccando su qualunque punto del grafico otterremo come legenda ciò che è racchiuso tra virgolette

- grafico creato con linee usando le opzioni col e lwd. In questo caso potremmo usare la seguente successione di comandi:
  `>plot(x,y,type=`"l",col=2,lwd=1)
  `>legend(locator(1),legend=`".....",col=1,lwd=1)
  cliccando su qualunque punto del grafico otterremo come legenda ciò che è racchiuso tra virgolette

- grafico creato con punti usando l'opzione col e pch . In questo caso potremmo usare la seguente successione di comandi:
  `>plot(x,y,type=`"p",col=3,pch="...")
  `>legend(locator(1),legend=`".....",col=3,pch="...")
  cliccando su qualunque punto del grafico otterremo come legenda ciò che è racchiuso tra virgolette

- grafico creato con linee usando le opzioni col e lty. In questo caso potremmo usare la seguente successione di comandi:
  `>plot(x,y,type=`"l",col=3,lty=1)
  `>legend(locator(1),legend=`".....",col=3,lty=1)
  cliccando su qualunque punto del grafico otterremo come legenda ciò che è racchiuso tra virgolette

20.5 Identificare il numero di posizione della coppia

Per identificare il numero di posizione della coppia si usa il la seguente successione di comandi:
`>identify(x,y,n=n)`
con la quale viene identificata la posizione di n punti tracciati nel grafico. Se l'opzione n non è inserita si continuerà a identificare numeri fino a che non si premerà il tasto esc.

Potrà anche essere inserita l'opzione: `labels=names(...)`
in cui al posto dei puntini va il nome di una variabile che potrà essere o x o y. In questo caso quando si clicca su un punto apparirà il nome della variabile scelta.

20.6 Plot e vettori con fattori evidenziati separatamente

Se si hanno a disposizione due vettori ciascuno di lunghezza n, x numerico e y di tipo fattore, può essere necessario a volte rappresentare in uno scatter plot il vettore x in modo però da evidenziare il fattore di provenienza. Ciò può essere fatto con la sequenza di comandi:
`>plot(1:n,x,type=`"n")
`>text(1:n,x,as.character(y))`
eventualmente per rendere meglio il grafico possiamo anche utilizzare l'opzione `col=.....`

Si noti che tale metodo può essere utilizzato anche nel caso si abbia a disposizione tre vettori aventi la stessa numerosità x,y di tipo numerico e il terzo z di tipo fattore per rappresentare uno scatter plot x contro y ma evidenziando i caratteri del fattore z. Si opera nel seguente modo:

```
>plot(x,y,type="n")
>text(x,y,as.character(z))
```

20.7 Plot e scatterplot con fattori evidenziati separatamente

Supponiamo di avere a disposizione un data frame che chiameremo `aaa` che contiene tre variabili che indicheremo con `x`, `y` e `z`. Le variabili `x` ed `y` sono numeriche mentre la variabile `z` è una variabili di tipo fattore con tre livelli. Possiamo tracciare il boxplot della variabile `x` contro la variabile `y` ma colorato diversamente a seconda dell'appartenenza ad un fattore rispetto ad un altro con la seguente sequenza di comandi:

```
>plot(aaa$x,aaa$y,pch=21,bg=c("red","grey","green")[codes(aaa$c)])
```

Un modo alternativo e semiautomatico può essere il seguente:

```
>plot(aaa$x,aaa$y,pch=21,bg=codes(aaa$c))
```

In ogni caso per aggiungere una legenda, ossia per evidenziare da quali gruppi di fattori provengono i singoli punti, dovremmo aggiungere una legenda e ciò sarà fatto con il comando:
`>legend(locator(1),legend=as.character(1:5),pch=21,col=1:5)`
nell'ipotesi che i codici di `aaa$c` siano in numero di cinque. Nel caso avessimo usato la prima sintassi, avremmo dovuto specificare la stessa sequenza di colori scritta dopo `bg`.

20.8 Plot e boxplot

Si noti anche che con il comando:
`>plot(x ~ y )`
in cui:

- x e un vettore di dati
- y è la corrispondente variabile di fattori

si ottiene un grafico formato da tanti boxplot quanti sono i fattori di y e naturalmente fatti con idati dei fattori corrispondenti.

20.9 I comandi points e lines

I comandi points e lines consentono di riscrivere rispettivamente punti e linee su un grafico già precedentemente tracciato. Tutte le opzioni che valgono per plot valgono anche per queste due funzioni.

Tali funzioni sono usate per formattare solamente il grafico e non gli assi dello stesso. Vi sono quindi vari casi da analizzare:

- per colorare solamente i punti di un grafico ottenuto per punti e non gli assi si userà la seguente successione di comandi:
  ```
  >plot(x,y,type="n")
  >points(x,y,col=3)
  ```

- per ottenere solamente i punti più grandi di un grafico ottenuto per punti e non gli assi si userà la seguente successione di comandi:
  `>plot(x,y,type=“n“)`
  `>points(x,y,lwd=6)`

- per colorare solamente le linee di un grafico ottenuto per linee e non gli assi si userà la seguente successione di comandi:
  `>plot(x,y,type=“n“)`
  `>lines(x,y,col=3)`

- per ottenere solamente le linee più grandi di un grafico ottenuto per linee e non gli assi si userà la seguente successione di comandi:
  `>plot(x,y,type=“n“)`
  `>lines(x,y,lwd=6)`

- per ottenere solamente le linee diverse di un grafico ottenuto per linee e non gli assi si userà la seguente successione di comandi:
  `>plot(x,y,type=“n“)`
  `>lines(x,y,lty=6)`

In questo caso per l'uso della legenda bisognerà fare riferimento a quanto specificato nelle funzioni points e lines.

20.10 Uso del comando plot due grafici

Quando devo tracciare due grafici contemporaneamente sarà necessario per tracciare il primo grafico utilizzare il comando plot e successivamente per tracciare i grafi successivi utilizzare i comandi points e lines. Il loro uso è abbastanza semplice, in ogni caso si vedano i seguenti esempi in cui x, y e z sono vettori aventi analoga lunghezza:

- per tracciare contemporaneamente due grafici con colori diversi possiamo utilizzare i comandi:
  `>plot(x,y,type=“l“,col=1)`
  `>lines(x,z,type=“l“,col=2)`
  `>legend(locator(1),legend=c(“y“,“z“),col=c(1,2),lwd=c(1,1)`

- per tracciare contemporaneamente due grafici con linee di diversa forma possiamo utilizzare i comandi:
  `>plot(x,y,type=“l“,lty=1)`
  `>lines(x,z,type=“l“,lty=2)`
  `>legend(locator(1),legend=c(“y“,“z“),lty=c(1,2)`

- per tracciare contemporaneamente due grafici con linee di diversa grandezza possiamo utilizzare i comandi:
  `>plot(x,y,type=“l“,lwd=1)`
  `>lines(x,z,type=“l“,lwd=2)`
  `>legend(locator(1),legend=c(“y“,“z“),lwd=c(1,2)`

- per tracciare contemporaneamente due grafici con caratteri diversi possiamo utilizzare i comandi:

```
>plot(x,y,type="p",pch="a")
>lines(x,z,type="p",pch="b")
>legend(locator(1),legend=c("y","z"),pch=c("ab")
```

- per tracciare contemporaneamente due grafici con caratteri e linee possiamo utilizzare i seguenti comandi:
  ```
  >plot(x,y,type="p",pch="a")
  >lines(x,z,type="l",lty=1)
  >legend(locator(1),legend=c("y","z"),pch="a",lty=c(0,1)
  ```

20.11 Grafico della funzione ad una variabile

Per tracciare il grafico delle funzioni ad una variabile dobbiamo operare nel seguente modo:

```
>curve(sin(x),a,b)
>axis(1,pos=0)
>axis(2,pos=0))
```

naturalmente la funzione può essere definita come visto in programmazione o scritta semplicemente in termini di x.

Si noti che se in curve aggiungiamo l'opzione add=Tla curva sarà aggiunta a lgrafico corrente.

20.12 Aggiungere una linea ai minimi quadrati

Tracciato uno scatter plot tra due vettori x ed y, per aggiungere una linea ai minimi quadrati si usa il comando:

```
>abline(lm(y~x),lty=2)
```

Si noti che la linea ai minimi quadrati può essere anche usata con l'opzione:

```
lty=...
```

molto utile per tracciare linee ai minimi quadrati provenienti da modelli differenti sullo stesso grafico.

20.13 Il grafico assocplot

Se `x` è una tabella di contingenza bidimensionale è possibile tracciare un grafico del tipo assocplot con il comando:

```
>assocplot(x)
```

20.14 Il grafico barplot

Dato un vettore numerico x e un vettore di caratteri y contenente i nomi degli elementi di x e avente la stessa lunghezza di x per per tracciare un diagramma a barre possiamo usare uno dei seguenti modi:

- `>barplot(x)` crea solamente il digramma a barre
- `>barplot(x,names=y)` crea il diagramma a barre e assegna ad ogni barra un nome
- `>barplot(x,col=1:5,legend=y)` crea il diagramma ed una legenda con il nome di ogni barra colorata

- `>barplot(x,col=1:5)`
  `>legend(locator(1),legend=y,fill=c(1:5))`
  lo stesso del punto precedente.

si noti che il vettore y al posto di essere generato prima può essere creato contestualmente alla creazione del diagramma.

Se invece x è una matrice di 2 righe e tre colonne,y un vettore contenente i nomi delle colonne e z un vettore contenente i nomi delle righe sempre di x, possiamo creare il diagramma a barra in uno dei seguenti modi:

- `>barplot(x,beside=T)` crea solamente il diagramma a barre
- `>barplot(x,names=y,legend=z,beside=T)` crea il diagramma a barre, assegna i nomi alle barre e crea una legenda
- `>barplot(x,col=c(1,2),beside=T)`
  `>legend(locator(1),legend=z,fill=c(1:2))` lo stesso del punto precedente

E' interessante usare questa opzione popo aver usato il comando tapply con più fattori, il grafico che si ottiene è molto efficace come si evince dal seguente esempio:

```
>xxx<-tapply(ragionieri$italiano,list(ragionieri$classe,ragionieri$corso)
,mean,na.rm=T)
>barplot(xxx,beside=T,names=dimnames(xxx)[[2]],legend=dimnames(xxx)[[1]],
ylim=c(0,10))
```

Il comando barplot si usa anche per tracciare dei grafici a barre, facenti funzioni di istogramma quando i dati sono raggruppati in classi non aventi la stessa dimensione. Supponiamo di dover rappresentare graficamente la seguente situazione:

classi	frequenze
120 -| 135	10
135 -| 145	20
145 -| 150	60
150 -| 165	10

Si puo' operare nel seguente modo:

```
>lunghezza<-c(15,10,5,15)
>frequenze<-c(10,20,60,10)
```

possiamo allora utilizzare il comando:

```
>barplot(frequenza,lunghezza)
```

così facendo però abbiamo una visione sfalsata della realtà in quanto sono le aree dei rettangoli e non le altezze che devono essere proporzionali. Per ovviare al problema conviene allora tracciare il barplot con il seguente comando:

```
>fc<-frequenza/(sum(frequenza)*lunghezza)
>barplot(fc,lunghezza,names=c(“120-|135“,“135-|145“,“145-|150“,“150-|165|“)“
```

Un modo alternativo di procedere è quello che consiste nell'operare nel seguente modo:

```
>dati<-c(rep(“120-|135“,10),rep(“135-|145“,20),rep(“145-|150“,60),rep(“150-|165|“,10))“
>lunghezza<-c(15,10,5,15)
>frequenze<-table(lunghezza)
>fc<-frequenza/(sum(frequenza)*lunghezza)
>barplot(fc,lunghezza,names=c(“120-|135“,“135-|145“,“145-|150“,“150-|165|“)“
```

Nell'uso del comando barplot vi sono alcune opzioni di particolare importanza. Esse sono:

- `>barplot(x,names=y,horiz=T)` le barre sono tracciate in modo orrizontale
- `>barplot(x,names=y,beside=T)` le barre non sono tracciate una sopra l'altra ma avvicinate, si usa con x matrice

20.15 Il grafico dotchart

Dato un vettore numerico x e un vettore di caratteri y contenente i nomi degli elementi di x e avente la stessa lunghezza di x per per tracciare un diagramma dotchart possiamo utilizzare il seguente comando:
`>dotchart(x,labels=y)`
Se invece z e un oggetto factor della stessa lunghezza di x possiamo utilizzare il comando:
`>dotchart(x,groups=z)`
Si noti che in questo caso `x` può essere anche matrice.

20.16 Il grafico piechart

Dato un vettore numerico x e un vettore di caratteri y contenente i nomi degli elementi di x e avente la stessa lunghezza di x per per tracciare un diagramma piechart possiamo utilizzare il seguente comando:
`>piechart(x,labels=y,col=1:n)`
in cui n sta a significare la lunghezza di x.

20.17 Il grafico boxplot

Dati i vettori numerici x,y e z e un vettore di caratteri w contenente i nomi dei vettori x, y e z per tracciare un diagramma boxplot possiamo utilizzare uno dei seguenti comandi:

- `>boxplot(x,names=`nomedix)
- `>boxplot(x,y,z,names=w)`

Si noti che se x è un vettore numerico ed y un vettore factor della stessa lunghezza con il comando:
`>boxplot(split(x,y))`
otteniamo tanti grafici boxplot quanti sono i livelli di y.

Un modo alternativo per usare questa funzione è quello di usarla con una formula nel seguente modo:
`>boxplot(x~y)`
in cui x è un vettore numerico ed y il corrispondente vettore di fattori.

E' anche possibile far stampare il grafico in modo orrizzontale utilizzando il seguente comando:
`>boxplot(x,horizontal=T)`

20.18 IL grafico coplot

Tramite il comando coplot è possibile ottenere uno scatter plot di un vettore numeri x contro y ciondizionato dal valore che assume una variabile di fattori a. L'uso del comando è il seguente:
`coplot(y~x|a,data=....)`
in cui .... è il nome del data frame che contiene le tra variabili precedentemente usate.

20.19 Il grafico fourfoldplot

Se x è un array del tipo 2x2x2x....x2xk tabelle di contingenza è possibile usare il comando:
`>fourfoldplot(x)`

20.20 Il grafico hist

Dato un vettore numerico x per tracciare un diagramma hist possiamo utilizzare uno dei seguenti comandi:

- `>hist(x)` traccia l'istogramma in modo automatico
- `>hist(x,breaks=seq(a,b,c))` traccia l'istogramma seguendo le classi indicate dal comando seq
- `>hist(x,breaks=c(a,b,c,d,e)` traccia l'istogramma considerando le classi indicate dal comando breaks
- `>hist(x,breaks=n)` traccia l'istogramma considerando n classi indicate nel comando breaks
- `>hist(x,nclass=n)` traccia l'istogramma considerando `n` classi indicate con il comando nclass

Una opzione particolarmente importante del comando hist è l'opzione probability, Se infatti lanciamo il comando: `>hist(x,probability=T)`
otteniamo automaticamente le frequenze relative del vettore x.

20.21 Il grafico density

Molto utile nell'analisi grafica delle distribuzioni dei dati è il diagramma density. Esso si ottiene con la seguente successione di comandi: `>idq<-summary(x)[5]-summary(x)[2]`
`>lines(density(x,width=2*idq))`
Tale comando inserisce in un grafico già esistente una linea che interpola graficamente una densità. Si usa molto spesso dopo aver creato il grafico hist con l'opzione probability=T.

20.22 Il grafico qqPlot

Per verificare se la la distribuzione da cui si pensa possano provenire i dati raccolti nel vettore x è una normale possiamo usare le seguenti funzioni:

- `>qqnorm(x)`
- `>qqline(x)`

Se pensiamo invece che la distribuzione da cui provengono i dati raccolti nel vettore x sia una uniforme continua possiamo usare la seguente sequenza di comandi:
`>plot(qunif(ppoints(x)),sort(x))`
Naturalmente la funzione quinif potrà essere sostituita da qualunque funzione tra le seguenti:

- qbeta con argomenti obbligatori shape1 e shape2
- qcauchy con argomenti facoltativi location scale

- qchisq con argomenti obbligatori df
- qexp con argomenti facoltativi rate
- qf con argomenti obbligatori df1 e df2
- qgamma con argomenti obbligatori shape
- qlnorm con argomenti facoltativi mean sd
- qnorm con argomenti facoltativi mean sd
- qt con argomenti obbligatori df
- qunif con argomenti facoltativi min max

Per verificare invece se due vettori x ed y provengono dalla medesima distribuzione di probabilità possiamo utilizzare il comando:
`>qqplot(x,y)`
e naturalmente x ed y devono avere lo stesso numero di dati. Se hanno numeri di dati differenti il grafico è tracciato ugualmente su dati interpolati.

20.23 Il grafico pairs

Se x è un data frame o una matrice possiamo ottenere il grafico dello scatter plot di ogni variabile verso le altre con il comando:
`>pairs(x)` Possiamo usare anche con pairs e con la stessa sintassi quanto detto nel paragrafo 20.7.

20.24 Il grafico mosaiplot

Se x è una tabella di contingenza possiamo ottenere il grafico di tipo mosaiplot con il seguente comando:
`>mosaicplot(x)`

20.25 Il grafico matplot

Se x ed y sono due matrici delle stesse dimensioni, supponiamo aventi m righe ed n colonne, con il comando maplot otteniamo n scatter plot tutti sullo stesso grafico ottenuti contrapponendo ordinatamente le colonne della prima matrice con le colonne della seconda matrice. Il comando da utilizzare sarà allora il seguente:
`>matplot(x, y, pch=“c(.....)“)`
in cui nell'opzione pch sono inseriti gli n caratteri che saranno stampati nel grafico. Se tale opzione non viene inserita la numerazione inizierà da 1 fino a raggiungere il numero delle colonne delle due matrici.

Un altro uso molto interessante del comando matplot è il seguente, se abbiamo una matrice x di m righe ed n colonne con il comando:
`>matplot(1:m,x)`
`>matplot(1:m,x,pch=“c(.....)“)`
otteniamo n scatter plot tutti sullo stesso grafico aventi tutti per per ascisse la successione 1:m e per ordinata ogni colonna di x.

Notiamo che esistono anche altri comandi della classe matplot e sono dati da:

- `>matpoints(x, y, pch=“c(.....)“)`
- `>matlines(x, y,  pch=“c(.....)“)`

i quali aggiungono punti o linee ad un grafico già esistente.

Tali tipi di grafici sono usati prevalentmente per analisi statistiche multivariate per vedere i legami tra le varie distribuzioni delle variabili

20.26 Il grafico stars

E' anche possibile generare degli star plot. Se abbiamo una matrice 20x5 con il comando:
`>stars(x)`
otteniamo 20 stelle a 5 punte che permettono di vedere i legami tra le 5 varibili colonna all'interno di ciascuna riga. E' usato prevalentemente per analisi multivariate in cui le righe della matrici sono le osservazioni mentre le colonne sono le variabili

20.27 Il grafico stripchart

Possiamo ottenere un grafico del tipo stripchart in uno dei seguenti modi:
`>stripchart(list(x,y))`
`>stripchart(x~a)`
in cui x ed y sono vettori numerici, mentre a è un vettore factor.

20.28 I grafici ldahist

Dopo aver caricato il pacchetto aggiuntivo MASS è possibile ottenere degli istrogrammi per gruppi di fattori con il seguente comando:
`ldahist(x, g)`
in cui `x` è il vettore dei dati e g il corrispondente vettore dei fattori.

20.29 I grafici tridimensionali

Un modo molto utile per utilizzare le funzioni tridimensionali è il procedimento che consente di rappresentare graficamente una tabella di contingenza. Possiamo allora utilizzare il seguente procediemnto:

```
>w<-table(xxx,yyy)
>contour(w)
>filled.contourw)
>image(w)
>persp(w)
```

Se x ed y sono due vettori numerici che danno il range di x e di y per una funzione a due variabili, e myfunction è una funzione generata come visto nel paragrafo relativo alla programmazione per generare una grafico tridimensionale possiamo allora utilizzare la seguente sintassi:

```
>z<-outer(x,y,function)
>contour(x,y,z)
>filled.contour(x,y,z)
>image(x,y,z)
>persp(x,y,z)
```

20.30 Parti comuni

Si noti che in ogni caso dopo aver generato un grafico possiamo usare i comandi

- `>title(main, sub, xlab, ylab, axes=F)`
- `>axes(main, sub, xlab, ylab, axes=T)`

per inserire titolo, sottotitolo, nome degli assi. Si noti che se usiamo
`>title(.....)`
senza alcuna opzione è predefinita l'opzione `main`. Naturalmente dopo il nome dell'opzione ci sarà il segno di uguale e quindi sarà necessario inserire i nomi desiderati racchiusa tra virgolette.

Si noti inoltre che se non si vuole che compaiano gli assi in un grafico in qualunque modo generato si può usare l'opzione:
`>plot(1:4, rnorm(4), axes=FALSE)`
Per aggiungere invece tick in intervalli voluti possiamo operare nel seguente modo:
`>plot(seq(20,75,5),cumsum(fi))`
`>axis(1, seq(20,75,5), as.character(seq(20,75,5)))`
in questo modo le etichette volute compariranno nell'asse delle `x`. Per farle comparire nell'asse delle `y` basterà sostituire con il `axis` l'opzione 2. Tale comando può essere usato anche per inserire etichette non numeriche.

20.31 Il comando rug

Dopo aver tracciato un grafico, molto utile è anche lanciare il seguente comando:
`>rug(x)`
il quale mette delle tacchette sull'asse delle x del grafico in corrispondenza dei valori presenti nel vettore x. Per inserire il corrispondente anche nell'asse delle y si dovrà usare il comando:
`>rug(x,side=2)`

20.32 Il comando locator

Una volta creato un grafico possiamo agire in modo interattivo su di esso con il comando:
`>locator(n=n)` con il quale è possibile cliccando n volte in qualunque punto del grafico ottenere una lista di due elementi contenente le coordinate degli n punti selezionati.

Il comando locator può anche essere utilizzato con la seguente sintassi:
`>locator(n=n,type=“ “)`
in cui

- se tra le virgolette vi è `l` viene aggiunta nel grafico una linea congiungente i punti in cui si è cliccato con il mouse
- se tra le virgolette è inserito `p` vengono aggiunti nel grafico n punti in corrispondenza a dove si è cliccato con il mouse
- se tra le virgolette è inserito `o` vengono inseriti nel grafico sia gli n punti dove si clicca con il mouse che una linea che li congiunge
- se tra le virgolette è inserito `b` si ottiene com il parametro o ma la linea non tocca esattamente il punto creato

si noti che questo comando è utilissimo per evidenziare i punti già tracciati nel grafico e si usa spesso dopo il comando `plot`.

20.33 Aggiungere un testo in un grafico

In alcuni casi è necessario dopo aver costruito un grafico, aggiungere allo stesso un testo in punti particolari. Questo può essere effettuato con il comando:
`>text(locator(1),“....“)`
con il quale è possibile aggiungere il testo “....“ nel punto dove si fa click con il mouse.

E' possibile anche aggiungere più testi in punti specificati, si veda in particolare l'esempio seguente:
`>plot(1:n,x,type=“n“)`
`>text(1:55,x,as.character(y))` con x vettore numerico ed y vettore di caratteri della stessa lunghezza.

20.34 I grafici multipli nella stessa pagina

E' anche possibile far comparire più grafici su di una stessa pagina. Per fare ciò possiamo utilizzare due distinti procedimenti. Il primo consiste nell'utilizzare il comando:
verb»par(mfrow=c(m,n))
il quale permette di realizzare un numero d imxn grafici su di una pagina sola. Una volta dato questo comando ogni volta che si da un comando grafico questo sarà automaticamente aggiunto alla pagina secondo la scansione proposta. Il secondo comando fa riferimento alla funzione screen la quale prevede il seguente gruppo di comandi:
`>split.screen(c(m,n))`
in quale permette di realizzare una pagina con un numero di mxn grafici.
`>screen(n)`
con il quale si decide in quale parte dello schermo deve essere disegnato il grafico prescelto.

21 Grafici Trellis

21.1 Uso dei grafici trellis

I grafici trellis sono grafici utilissimi per tracciare diagrammi a multipannello. Inserendo nel comando per generare il grafico trellis una delle seguente opzioni

- `|a`
- `|a*b*...`

in cui a, b, ecc possono essere sia fattori che vettori numerici, si otterranno dei grafici multi-pannello con le condizioni specificate. In questo paragrafo prenderà il nome di condizione un vettore che potrà essere o numerico o factor.

Se a,b,c, sono numeri discreti ossia valori che si ripetono sempre uguali il condizionamento è fatto per ogni singolo valore considerando come dati il numeri di volte in cui esso si è ripetuto.

Se a,b,c sono numeri reali ma non discreti nel senso che non hanno valori che si ripetono uguali tra di loro ma sono costruiti da valori diversi dovremmo considerare il condizionamento rispetto ad un intervallo in quanto in caso contrario il grafico che otterremo sarebbe privo di significato. In questo caso prima di effettuare il condizionamento dobbiamo creare gli intervalli con:
`k <-equal.count(x,number=n,overlap=m)`
in cui x è la variabile condizionante, n è il numero delle classi ed m è il numero dei punti

condivisi con intervalli successivi. Si noti che con tale comando il programma tenta di ripartire in n classi lo stesso numero di osservazioni. Fatto ciò sarà la variabile k che dovremmo usare nel condizionamento. Molto importanti sono le seguenti istruzioni:

- `>range(k)`
- `>plot(k)`
- `>levels(k)`

In tale tipo di grafici le variabili possono essere chiamate solamente con il nome senza inserire davanti il nome del data frame o matrice in cui esse sono inserite. Sarà però necessario usare l'opzione:
`>data=......`
all'interno del comando trellis utilizzato per creare il grafico. Naturalmente `......` è il nome del data frame o matrice che contiene le variabili usate per generare il grafico.

Usando i grafici Trellis, viene generato come detto un notevole numero di grafici in base al condizionamento proposto. Per evitare di ottenere grafici illeggibili possiamo con l'opzione:
`>layout=c(n,m,k)`
verranno inserite in k pagine $n * m$ grafici. Naturalmente questa opzione deve essere inserita all'interno del comando usato per generare il grafico trellis desiderato.

21.2 I pacchetti necessari

Per poter utilizzare i grafici trellis in R sarà necessario installare i pacchetti:
`grid`
`lattice`
solo in questo modo essi potranno essere tracciati

21.3 Il grafico xyplot

Per usare il comando xyplot possiamo usare uno dei seguenti modi:

- `>xyplot(numeric1 ~ numeric2|condizione)`
- `>xyplot( ~ numeric|condizione)`

Si noti anche che è possibile ottenere il grafico di una sola parte dei dati numerici considerati nel seguente modo:

- `>xyplot(y[x<1] ~ x[x<1]|condizione,data=z)`
- `>xyplot(y ~ x|condizione ,data=z,subset=x<1)`

21.4 Il grafico bwplot

Per usare il comando bwplot possiamo usare uno dei seguenti modi:

- `>bwplot(factor ~ numeric|condizione)`
- `>bwplot( ~ numeric|condizione)` ma in questo caso la condizione non deve essere un factor

Se la variabile factor usata non è un fattore viene automaticamente convertita in fattore dal programma.

21.5 Il grafico stripplot

Per usare il comando stripplot possiamo usare uno dei seguenti modi:

- `>stripplot(factor ~ numeric|condizione)`
- `>stripplot( ~ numeric|condizione)` ma in questo caso la condizione non può essere un factor

Se la variabile factor usata non è un fattore viene automaticamente convertita in fattore dal programma.

21.6 Il grafico qq

Per usare il comando qq possiamo usare il seguente modo:
`>qq(factor ~ numeric|condizione)`
E si noti che la variabile factor deve essere fattore o numerica con esattamente due livelli. Nel caso in cui la variabile factor avesse più livelli per usare qq dovremmo indicarne solo due operando allora nel seguente modo:
`>qq(factor ~ numeric|condizione  ,subset(factor==`“a“)|factor==“b“))

21.7 Il grafico dotplot

Per usare il comando dotplot possiamo usare uno dei seguenti modi:

- `>dotplot(factor ~ numeric|condizione)`
- `>dotplot( ~ numeric|condizione)` ma in questo caso la condizione non può essere un factor

Se la variabile factor usata non è un fattore viene automaticamente convertita in fattore dal programma.

21.8 Il grafico qqmath

Per usare il comando qqmath possiamo usare uno dei seguenti modi:
`>qqmath( ~ numeric|condizione,distribution= function(p)qt(p,df=7)))`
`>qqmath(~numeric|condizione,subset=(y==`“a“),distribution=function(p)qt(p,df=7)))
in cui y è una variabile factor esistente nel data frame originario.

Si noti che il valore di distribution che finora è stato supposto qt, è quello visto parlando dei grafici tradizionali ossia:

- qbeta con argomenti obbligatori shape1 e shape2
- qcauchy con argomenti facoltativi location scale
- qchisq con argomenti obbligatori df
- qexp con argomenti facoltativi rate
- qf con argomenti obbligatori df1 e df2
- qgamma con argomenti obbligatori shape

- qlnorm con argomenti facoltativi mean sd
- qnorm con argomenti facoltativi mean sd
- qt con argomenti obbligatori df
- qunif con argomenti facoltativi min max

21.9 Il grafico barchart

Per usare il comando barchart possiamo usare uno dei seguenti modi:

- `>barchart(factor ~ numeric|condizione)`
- verb»barchart(numeric|condizione) ma in questo caso la condizione non può essere un factor

Se la variabile factor usata non è un fattore viene automaticamente convertita in fattore dal programma.

21.10 Il grafico histogram

Per usare il comando histogram possiamo usare il seguente modo:
h`>histogram( ~ numeric|condizione)`
Possiamo naturalemnte inserire anche le seguenti opzioni

- `breaks` come nel caso nei grafici non trellis
- `nint` per specificare il numero delle classi
- `type=`"percent" o "count" se si vuole il grafico percentualizzato o di frequenza assolute.

21.11 Il grafico densityplot

Per usare il comando densityplot possiamo usare il seguente modo:
`>densityplot( ~ numeric|condizione)`

21.12 Il grafico splom

Per usare il comando splom possiamo usare il seguente modo:
`>splom( ~ dataframe|condizione)`

21.13 Il grafico parallel

Per usare il comando parallel possiamo usare il seguente modo:
`>parallel( ~ dataframe|condizione)`

21.14 Il grafico rfs

Il grafico rfs consente di ottenere due grafici per visualizzare i valori residui e quelli stimati con il modello. La sintassi da usare è la seguente:
`>rfs(model)`
in cui model è un oggetto ottenuto con la regressione o l'analisi della varianza.

21.15 Grafici a più dimensioni

Avendo a disposizione tre vettori con la stessa numerosità x, y e z possiamo rappresentare una relazione tra queste tre variabili tramite grafici di tipo tridimensionali di tipo trellis. Un primo modo di rappresentare questi dati è quello di usare la seguente sintassi:

```
>cloud(z ~ x*y|condizione)
>levelplot(z~x*y|condizione)
```

si ricorda inoltre sempre la possibilità se x, y e z sono variabili di un data frame di utilizzare i nomi delle stesse variabiabili con l'opzione `data=.....`.

Anche i trellis grafici permettono di ottenere grafici di funzioni a due variabili. Dobbiamo prima di tutto inizializzare le varibili nel seguente modo:

```
>x<-rep(seq(-1.5,1.5,length=50),50)
>y<-rep(seq(-1.5,1.5,length=50),each=50)
>z<-exp(-(x^2+y^2+x*y))
>w<-data.frame(x,y,z)
```

Possiamo allora usare i grafici trellis tridimensionali nel seguente modo:

```
>cloud(z ~ x*y|condizione,data=w)
```

in cui condizione come al solito è un vettore numeric o factor.

22 Aggiungere del testo ad un grafico

22.1 Introduzione

In molti casi è necessario inserire in un grafico un testo qualsiasi per evidenziare un punto, una linea o per altri motivi. Questo può essere fatto in modo molto semplice in R attraverso il comando *text*.

22.2 Il comando text

Creato un grafico con il comando *text* possiamo aggiungere un testo qualsiasi in un punto qualsiasi del grafico con la seguente sintassi:

```
>text(x, y, labels ="..... " )
>text(locator(n),labels ="..... " )
```

in cui x ed y sono vettori o scalari ,nel caso di inserimento di un solo testo, che mi danno le coordinate dei punti dove inserire il testo.

23 Aggiungere del testo matematico ad un grafico

23.1 Introduzione

In molti casi è necessario aggiungere legende e testi matematici ad un grafico precedentemente creato o aggiungerne ad uno che stimo creando. Tutte le volte che ciò è richiesto da comandi tipo *main*, *xlab*, *ylab* e altri possiamo con un comando apposito introdurre formule matemtiche a ltesto così creato.

23.2 L'opzione expression

Il comando expression si usa ad esempio ne lseguente modo:

```
>text(locator(1),expression(pi))
```

`>title(main=expression(pi^2))`
se i grafici sono già stati precedentemente creati o con ad esempio:
`>plot(x,y,main=expression(pi^2)`
in sede di creazione del grafico.

23.3 Quali sono i simboli matematici inseribili

Un elenco dei simboli matematici inseribili si ottiene lanciando il demo di R sulla parte grafica con il comando:
`>demo`(“graphics“)

24 Le Variabili aleatorie fondamentali dell'inferenza statistica

24.1 Elenco delle variabili aleatorie fondamentali

L'inferenza statisitca fa continuamente uso di variabili aleatorie, fondamentali in tale tipo di analisi. Le varibili che possono essere utilizzate sono le seguenti:

- normale indicata con `norm`
- beta indicata con `beta`
- cauchy indicata con `cauchy`
- di chiquadrato indicata con `chisq`
- esponenziale indicata con `exp`
- f indicata con `f`
- gamma indicata con `gamma`
- lognormale indicata con `lnorm`
- logistica indicata con `logis`
- t di studentindicata con `t`
- uniforme continua indicata con `unif`
- weibull indicata con `weibull`
- binomiale indicata con `binom`
- geometrica indicata con `geom`
- ipergeometrica indicata con `hyper`
- binomiale negativa indicata con `nbinom`
- poisson indicata con `pois`
- wilxon indicata con `wilcox`

Esistono altri pacchetti aggiuntivi come il pacchetto `SuppDists` i quali forniscono altre varibili aleatorie utilizzabili per analisi statistiche particolari.

24.2 Uso delle variabili aleatorie fondamentali

Le variabili aleatori fondmanetali non possono essere usate semplicemente digitando i loro nome ma tale nome deve essere fatto precedere da un prefisso che potrà essere:

- `p` per ottenere la funzione di distribuzione della variabile
- `d` per ottenere la funzione di probabilità della variabile
- `q` per ottenere i quantili della variabile

- `r` per ottenere numeri casuali derivanti dalla variabile

Naturalmente a seconda del prefisso inserito, per ottenere il valore desiderato, dovremmo indicare alcuni parametri dati da:

- se il prefisso inserito è `p` dovremmo indicare il quantile o il vettore dei quantili dei quali desideriamo ottenere la funzione di ripartizione
- se il prefisso inserito è `d` dovremmo indicare un numero o un vettore di numeri del quale ottenere la funzione di densità
- se il prefisso inserito è `q` indicare la coda inderiore di probabilità o il vettore delle code superiori di probabilità dei quali desideriamo ottnere i quantili
- se il prefisso inserito è `r` dovremmo indicare un numero intero indicante la numerosità dei numeri casuali che vogliamo ottnerei

24.3 Argomenti addizionali

Oltre al prefisso e ai parametri indicati nei paragrafi precedenti, per ottenere un valore dalle variabili aleatorie dovremmo indicare alcuni argomenti addizionali diversi da variabile aleatoria a variabile aleatoria. L'elenco completo di tali argomenti addizionali è il seguente:

- `norm` richiede mean=, sd=
- `beta` richiede shape1=, shape2=, ncp=
- `cauchy` richiede location=, scale=
- `chisq` richiede df=, ncp=
- `exp` richiede rate=
- `f` richiede df1=, df2=, ncp=
- `gamma` richiede shape=, scale=
- `lnorm` richiede meanlog=, sdlog=
- `logis` richiede location=, scale=
- `t` richiede df=, ncp=
- `unif` richiede min=, max=
- `weibull` richiede shape=, scale=
- `binom` richiede size=, prob=
- `geom` richiede prob=
- `hyper` richiede m=, n=, k=
- `nbinom` richiede size=, prob=

- `pois` richiede lambda=
- `wilcox` richiede m=,n=

indipendentemente dal prefisso utilizzato. Per ulteriori informazioni si consiglia di utilizzare per le varibili viste l'help in linea.

25 Utilizzo dei grafici nell'inferenza statistica

25.1 Grafici per la verifica della normalità dei campioni

Molti dei test inferenziali che vedremo scessivamente si basano sull'assunzione della normalità della popolazione da cui proviene il campione. Per la verifica di tale ipotesi, oltre a quanto visto nel capitolo riguardante l'adattamento dei dati ad una particolare distribuzione possiamo utilizzare una serie di grafici dati da:

- `>hist(x)`
- `>boxplot(x)`
- `>idq<-summary(x)[5]-summary(x)[3]`
  `>plot(density(x,width=2*idq),xlab=`“x“,ylab=““,type=“l“)
- `>qqnorm(x)`
- `>qqline(x)`

Tali grafici possono facilmente essere ottenuti in modo automatico con la creazione di una opportuna funzione.

25.2 Grafici per la verifica di correlazione nel campione

I test inferenzaili che utilizzeremo presuppongono sempre che il campione sia un campione di tipo bernulliano e che quindi i dati campionari non siano tra di loro correlati. Per la verifica di tale fatto dopo aver caricato il pacchetto aggiuntivo:
`ts`
possiamo utilizzare i seguenti grafici:

- `>ts.plot(x)`
- `>acf(x)`

Tali grafici possono facilmente essere ottenuti in modo automatico con la creazione di una opportuna funzione.

25.3 Grafici per la verifica della correlazione tra due campioni

I test inferenzali che implicano due campioni presuppongono sempre che tra i due campioni non vi sia traccia di correlazione. Per la verifica di tale fatto dopo aver caricato il pacchetto aggiuntivo:
`ts`
possiamo utilizzare i seguenti grafici:

- `>plot(x,y)`
- `>identify(x,y,n=...)` per identificare i valori anomali
- `>acf(x)`
- `>acf(y)`

Tali grafici possono facilmente essere ottenuti in modo automatico con la creazione di una opportuna funzione.

26 Test inferenziali

26.1 I pacchetti necessari

Per poter utilizzare i test inferenziali descritti in questo paragrafo sarà necessario avere installato il pacchetto aggiuntivo:
`ctest`
che di solito è già presente e operante in qualunque distribuzione del programma.

27 Verifica di ipotesi su una variabile casuale normale

27.1 Introduzione

In tutto il presente paragrafo faremo esclusivamente riferimento a campioni bernulliani estratti da popolazioni normalmente distribuite. Ci troviamo quindi nell'ambito della statistica parametrica.

27.2 Verifica di ipotesi sulla media

La verifica di ipotesi sulla media della popolazione da cui proviene un campione nel caso in cui la varianza di tale popolazione sia sconosciuta è effettuato tramite il `test t di student`. Tale test consente di verificare le seguenti ipotesi:

- se la media della popolazione da cui proviene il campione è pari ad un valore prefissato
- se la media delle popolazioni da cui sono estratti due campioni sono uguali tra di loro
- se un certo trattamento effettuato sulle unità campionarie dopo l'estrazione ne ha modificato la media

Tale test ha la seguente sintassi:
>t.test(x,y=null,alt="two.sided",mu=0,paired=F, var.equal=F,conf.level=0.95)
Notiamo allora che:

- `x` ed `y` sono vettori numerici e rappresentano i dati raccolti con l'analisi campionaria. Se è introdotto un unico vettore il test riguarderà la verifica della media della popolazione da cui il campione è estratto, se invece sono introdotti due vettori, il test potrà riguardare la verifica dell'ipotesi dell'uguaglinza delle medie delle due popolazioni da cui sono estratti i campioni. In questo caso possiamo anche considerare l'analisi per dati appaiati modificando il valore di `paired` e portandolo a `TRUE`

- nel caso in cui si voglia cambiare il valore dell'ipotesi nulla da testare basterà cambiare il valore di `mu` al valore desiderato
- nel caso si voglia cambiare il livello di significatività del test sarà necessario cambiare il valore di `conf.level` portandolo al valore desiderato
- l'opzione `alt` può essere cambiata anche in `greater` e `less` a seconda che interessi la coda superiore o quella inferiore
- nel caso di due campioni il test ipotizza sempre che la varianza da cui provengono i campioni sia diversa, e quindi per il corretto uso del test sarà necessario modificare l'opzione in `var.equal=T`. Eè buona norma in questo caso far precedere tale test dal test `var.test` che testa se le varianze delle popolazioni da cui provengono i campioni sono uguali oppure diverse.

27.3 Verifica dell'ipotesi di uguaglianza della media: caso dei confronti multipli

Avendo a disposizione `k` campioni provenienti da `k` popolazioni diverse, a volte è necessario effettuare un test per verificare l'uguaglianza delle medie mediante confronti multipli effettuati tra i `k` campioni prendendone sempre due a due in modo da considerare tutte i possibili accoppiamenti. Si tratta allora di effettuare un test `t di student` ripetuto più volte. Il comando per eseguire tali confronti nultipli è il seguente:
`>pairwise.t.test(x, g, p.adjust.method=p.adjust.methods, pool.sd=TRUE,...)`
in cui abbiamo che:

- `x` è il vettore dei dati
- `g` rappresenta il vettore dei fattori ossia un oggetto factor che indicherà con numeri o lettere l'appartenenza dei dati di x ad un determinato campione
- `p.adjust` indica il metodo da utilizzare che può essere dato da
  - holm"
  - hochberg"
  - bonferroni"
  - none"

Nel paragrafo 35.6 vedremo un altro modo per effettuare tali confronti multipli.

27.4 Verifica dell'ipotesi di uguaglianza della media: caso del confronto globale

Avendo a disposizione `k` campioni provenienti da `k` popolazioni diverse, a volte è necessario effettuare un test per verificare l'uguaglianza delle medie mediante un confronto globale effettuato simultaneamente sui `k` campioni. Si tratta di effettuare un test noto come analisi della varianza che verrà studiato nel paragrafo 35. Senza entrare in dettagli vediamo che l'analisi della varianza viene eseguita con il seguente comando:
`>anavar<-aov(x~g)` oppure
`>anavar<-aov(x~g,data=...)`

`>summary(anavar)`
in cui si avrà che:

- `x` il vettore dei dati
- `g` rappresenta il vettore dei fattori ossia un oggetto factor che indicherà con numeri o lettere l'appartenenza dei dati di x ad un campione

27.5 Il test sulla uguaglianza delle varianze di due popolazioni

Avendo a disposizione 2 campioni provenienti da 2 popolazioni diverse, ci si pone il problema di verificare se le varianze delle popolazioni da cui tali campioni provengono sono tra di loro uguali. Tale verifica, come detto, è preliminare all'applicazione del test `t di student` nel caso di due campioni. Il test per eseguire questa verifica viene eseguito con il seguente comando:
`>var.test(x,y,alt=`"two.sided",conf.level=0.95)
in cui

- `x` ed `y` sono vettori numerici e rappresentano i dati raccolti con l'analisi campionaria nei due campioni
- nel caso si voglia cambiare il livello di significatività del test sarà necessario cambiare il valore di `conf.level` portandolo al valore desiderato
- l'opzione `alt` può essere cambiata anche in `greater` e `less` a seconda che interessi la coda superiore o quella inferiore

Si ricordi che se tale test non permette di accettare l'ipotesi nulla, non potremmo eseguire il test t di student almeno in presenza di piccoli campioni.

27.6 Il test sulla uguaglianza delle varianza di più di due popolazioni

Avendo a disposizione `k` campioni provenienti da `k` popolazioni diverse ci si pone il problema di verificare se le varianze delle popolazioni da cui provengono tali campioni siano tra di loro uguali. Ciò può esserre effettuato con il `test di barlett`. Tale test viene eseguito con il seguente comando:
`>bartlett.test(x, g, ...)`
oppure con il seguente comando:
`>bartlett.test(formula, data, subset,na.action,...)`
in cui:

- `x` è il vettore dei dati
- `g` rappresenta il vettore dei fattori ossia un oggetto factor che indicherà con numeri o lettere l'appartenenza dei dati di x ad un determinato campione
- `formula` rappresenta una formula intesa nel seguente modo `y~g`

Si ricordi che se tale test non permette di accettare l'ipotesi nulla, non potremmo eseguire l'analisi della varianza sui campioni in precedenza raccolti.

27.7 Il test sulla correlazione

Avendo a disposizione 2 campioni provenienti da 2 popolazioni diverse, ci si pone il problema di verificare se le popolazioni da cui provengono tali campioni sono tra di loro correlate. Ciò può esserre effettuato con il `test di correlazione`. Tale test viene effettuato con il seguente comando:
`>cor.test(x,y,alt=`“two.sided“,method=“pearson“,conf.level = 0.95)
Notiamo allora che:

- `x` ed `y` sono vettori numerici e rappresentano i dati raccolti con l'analisi campionaria nei due campioni
- nel caso si voglia cambiare il livello di significatività del test sarà necessario cambiare il valore di `conf.level` portandolo al valore desiderato
- l'opzione `alt` può essere cambiata anche in `greater` e `less` a seconda che
- il valore di `method` utilizzato è quello di `Pearson` e quindi per effettuare il test di correlazione non deve essere modificato

Con lo stesso comando ma cambiando l'opzione `method` possiamo eseguire i test non parametrici o con distribuzione libera come vedremo nel paragrafo 28.6.

27.8 Funzione potenza per il test t

La funzione potenza indica la probabilità di rifiutare l'ipotesi nulla al variare del parametro ignoto oggetto del test. La funzione potenza per il test `t di student` può essere eseguita con il comando:

```
power.t.test(n=NULL, delta=NULL, sd=1, sig.level=0.05, power=NULL, type=
c("two.sample","one.sample", "paired"),alternative=c("two.sided", "one.sided"))
```

Si avrà che:

- `n` indica la numerosità del campione o dei campioni nel caso di test con due campioni
- delta indica la differenza tra il valore di `mu` utilizzato nel test t di student e quello variabile in base al quale si vuole calcolare il valore della funzione potenza
- `sd` indica lo scarto quadratico medio del campione o dei due campioni
- `sig.level` indica il livello di significatività ossia l'errore di prima specie
- `power` indica il valore della funzione potenza
- `type` indica se la funzione potenza è calcolata su test t di student calcolato su unico campione, su due campioni o su due campioni con dati appaiati
- `alternative` indica se la funzione potenza è calcolata su un test t di student in cui l'ipotesi alternativa era two.sided o one.sided

si noterà certamente che vei valori `n,delta,sd,sig.level,power` ne dovrà mancare solamente uno e il test restituirà proprio il valore del paramentro non introdotto in funzione degli altri.

Per tracciare il grafico della funzione potenza dobbiamo distinguere i seguenti casi:

- se `alternative=`"one" e se il sistema di ipotesi è il seguente:

$$\begin{cases} H_0 : \mu = \mu_0 \\ H_1 : \mu > \mu_0 \end{cases}$$

  per ottenere il grafico dobbiamo operare nel seguente modo:

```
xxx<-power.t.test(delta=seq(-1,1,0.1),sd=1, n=25,type="one",
alternative="one")$power
plot(seq(-1,1,0.1),xxx)
```

- se `alternative=`"one" e se il sistema di ipotesi è il seguente:

$$\begin{cases} H_0 : \mu = \mu_0 \\ H_1 : \mu < \mu_0 \end{cases}$$

  per ottenere il grafico dobbiamo operare nel seguente modo:

```
xxx<-power.t.test(delta=seq(-1,1,0.1),sd=1, n=25,type="one",
alternative="one")$power
plot(seq(-1,1,0.1),sort(xxx,decreasing=T)
```

- se `alternative=`"two" per ottenere il grafico dobbiamo operare nel seguente modo:

```
xxxpos<-power.t.test(delta=seq(0,1,0.1),sd=1, n=25,type="one",
alternative="""two""")$power
xxxneg<-power.t.test(delta=seq(0,1,0.1),sd=1, n=25,type="one",
alternative="two")$power
plot(c(seq(-1,0,0.1),seq(0,1,0.1)),c(sort(xxxneg,decreasing=T),xxxpos))
```

Analogamente si ragione nel caso in cui `type=`"two".

28 Verifica di ipotesi su variabili con distribuzione libera

28.1 Introduzione

In tutto questo capitolo si presuppone di non conoscere o di non avere alcuna informazione circa la popolazione da cui i campioni sono estratti. Ci troviamo quindi nell'abito della statistica non parametrica.

28.2 Verifica di ipotesi sulla mediana

La verifica di ipotesi sulla mediana della popolazione da cui proviene un campione in ambito non parametrico è effettuata tramite il `test di wilcoxon`. Tale test consente di verificare le seguenti ipotesi:

- se la mediana della popolazione da cui proviene il campione è pari ad un valore prefissato
- se la mediana delle popolazioni da cui sono estratti due campioni sono uguali tra di loro

- se un certo trattamento effettuato sulle unità campionarie dopo l'estrazione ne ha modificato la mediana

Tale test ha la seguente sintassi:
>wilcox.test(x,y=null,alt="two.sided",mu=0,paired=F, var.equal=F,conf.level=0.95)
Notiamo allora che:

- `x` ed `y` sono vettori numerici e rappresentano i dati raccolti con l'analisi campionaria. Se è introdotto un unico vettore il test riguarderà la verifica della mediana della popolazione da cui il campione è estratto, se invece sono introdotti due vettori, il test potrà riguardare la verifica dell'ipotesi dell'uguaglianza delle mediane delle due popolazioni da cui sono estratti i campioni. In questo caso possiamo anche considerare l'analisi per dati appaiati modificando il valore di `paired` e portandolo a `TRUE`
- nel caso in cui si voglia cambiare il valore dell'ipotesi nulla da testare basterà cambiare il valore di `mu` al valore desiderato
- nel caso si voglia cambiare il livello di significatività del test sarà necessario cambiare il valore di `conf.level` portandolo al valore desiderato
- l'opzione `alt` può essere cambiata anche in `greater` e `less` a seconda che interessi la coda superiore o quella inferiore
- nel caso di due campioni il test ipotizza sempre che la varianza da cui provengono i campioni sia diversa, e quindi per il corretto uso del test sarà necessario modificare l'opzione in `var.equal=T`. Eè buona norma in questo caso far precedere tale test dal test `var.test` che testa se le varianze delle popolazioni da cui provengono i campioni sono uguali oppure diverse.

28.3 Verifica dell'ipotesi di uguaglianza della mediana: caso dei confronti multipli

Avendo a disposizione `k` campioni provenienti da `k` popolazioni diverse, a volte è necessario effettuare un test per verificare l'uguaglianza delle mediane mediante confronti multipli effettuati tra i `k` campioni prendendone sempre due a due in modo da considerare tutte i possibili accoppiamenti. Si tratta allora di effettuare un `test di wilcoxon` ripetuto più volte. Il comando per eseguire tali confronti multipli è il seguente:
`>pairwise.wilcox.test(x, g, p.adjust.method=p.adjust.methods, pool.sd=TRUE,...)`
in cui abbiamo che:

- `x` è il vettore dei dati
- `g` rappresenta il vettore dei fattori ossia un oggetto factor che indicherà con numeri o lettere l'appartenenza dei dati di x ad un determinato campione
- `p.adjust` indica il metodo da utilizzare che può essere dato da
  - holm"
  - hochberg"
  - bonferroni"
  - none"

28.4 Verifica dell'ipotesi di uguaglianza della mediana: caso del confronto globale

Avendo a disposizione `k` campioni provenienti da `k` popolazioni diverse, a volte è necessario effettuare un test per verificare l'uguaglianza delle mediane mediante un confronto globale effettuato simultaneamente sui `k` campioni. Si tratta di effettuare un test noto come analisi della varianza non parametrica che verrà studiato nel paragrafo 35. Senza entrare in dettagli vediamo che l'analisi della varianza viene eseguita con il seguente comando:
`>kruskal.test(x, g, ...)`
oppure `>kruskal.test(formula, data, subset,na.action,...)`
in cui si avrà che:

- `x` il vettore dei dati
- `g` rappresenta il vettore dei fattori ossia un oggetto factor che indicherà con numeri o lettere l'appartenenza dei dati di x ad un campione
- il valore di `formula` rappresenta la formula intesa nel seguente modo `y~g`

28.5 Il test sulla uguaglianza delle varianze di più popolazioni

Avendo a disposizione `k` campioni provenienti da `k` popolazioni diverse ci si pone il problema di verificare se le varianze delle popolazioni da cui provengono tali campioni siano tra di loro uguali. Ciò può esserre effettuato con il `test di fligner`. Tale test viene eseguito con il seguente comando:
`>fligner.test(x, g, ...)`
oppure il seguente comando:
`>fligner.test(formula, data, subset,na.action,...)`
in cui:

- `x` è il vettore dei dati
- `g` rappresenta il vettore dei fattori ossia un oggetto factor che indicherà con numeri o lettere l'appartenenza dei dati di x ad un determinato campione
- `formula` rappresenta una formula intesa nel seguente modo `x~g`

28.6 Il test sulla correlazione

Avendo a disposizione 2 campioni provenienti da 2 popolazioni diverse, ci si pone il problema di verificare se le popolazioni da cui provengono tali campioni sono tra di loro correlate. Ciò può esserre effettuato con il `test di correlazione`. Tale test viene effettuato con il seguente comando:
`>cor.test(x,y,alt=`"two.sided",method=" ",conf.level = 0.95)
Notiamo allora che:

- `x` ed `y` sono vettori numerici e rappresentano i dati raccolti con l'analisi campionaria nei due campioni
- nel caso si voglia cambiare il livello di significatività del test sarà necessario cambiare il valore di `conf.level` portandolo al valore desiderato

- l'opzione `alt` può essere cambiata anche in `greater` e `less` a seconda che
- il valore di `method` utilizzato per l'analisi non parametrica della correlazione è dato da:
  - kendall
  - sperman

29 I test per le proporzioni

29.1 Introduzione

I test per proporzioni sono quei test che riguardano inferenze su una certa modalità di un carattere quantitativo presente in una certa popolazione. Essi si articolano in una serie di test che studieremo nei paragrafi successivi.

29.2 Il test binomiale

Si supponga che una certa modalità di un carattere quantitativo sia presente in una certa popolazione in una percentuale che indicheremo con `p`. Sulla base dei risultati derivanti da un campione bernulliano estratto da tale popolazione vogliamo sottoporre a test l'ipotesi che la vera proporzione con cui il carattere è ripartito nella popolazione sia pari ad un certo valore. Tale inferenza viene effettuata con il `test binomiale esatto`. Tale test si esegue con il comando:
`>binom.test(x, n, p = 0.5,alt = c`(“two.sided“, “less“, “greater“),conf.level=0.95)
in cui si avrà che:

- `x` indica la numerosità dei successi che si sono presentati nel campione
- `n` indica la numerosità campionaria
- `p` indica la vera percentuale che supponiamo essere presente nella popolazione

Possiamo anche notare che:

- nel caso in cui si desidera modificare il livello di significatività del test sarà necessario cambiare il valore di `conf.level` sostituendolo con il valore desiderato
- il valore di `alt` può essere cambiato in `greater` e `less` a seconda che interessi la coda superiore o quella inferiore
- è possibile inserire in `x` un vettore con due componenti i cui elementi sono dati dal numero di successi e da quello degli insuccessi, in questo caso non si deve inserire alcun valore per `n`

Il test binomiale è importante in quanto fornisce anche gli intervalli di confidenza della vera proporzione con cui il carattere quantitativo è presente nella popolazione. Per ottenere tale intervallo sarà sufficiente introdurre come valore di `p` la percentuale di successi presente nel campione. Ad esempio se vogliamo ottenere l'intervallo di confidenza della vera proporzione dei votanti un determinato condidato in una certa popolazione avendo rilevato su un campione di 100 persone che 55 hanno votato per esso basterà utilizzare il seguente comando:
`>binom.test(55, 100, p = 0.55)`

29.3 Il test per due o più le proporzioni

Si supponga che una certa modalità di un carattere quantitativo sia presente in `k` popolazioni in una certa percentuale che indicheremo con `p1,p2,...pk`. Sulla base dei risultati derivanti da `k` campioni bernulliani estratti dalle `k` popolazioni vogliamo sottoporre a test l'ipotesi che la proporzione con cui il carattere è presente nelle `k` popolazioni sia uguale ai valori `p1,p2,...pk`. Tale inferenza viene effettuata con il `test prop.test`. Tale test si esegue con il comando:
`>prop.test(x, n, p = NULL, alt = c`("two.sided", "less", "greater"),conf.level = 0.95,
`correct=TRUE)`
in cui si avrà che:

- `x` indica un vettore contenente i successi ottenuti in ognuno dei `k` campioni o una matrice con due colonne contenente nella prima i successi ottenuti in ogni campione e nella seconda colonna gli insuccessi ottenuti in ogni campione
- `n` indica un vettore contenente la numerosità dei `k` campioni e dovrà avere lo stesso numero di righe di x. Tale parametro è ignorato se `x` è una matrice
- `p` indica un vettore che contiene le vere percentali che supponiamo essere presenti nelle popolazioni da cui i campioni sono estratti . Tale vettore deve avere la stessa lunghezza di `x` e se `p` non è inserito si considera che la proporzioni di successi siano uguali in ogni popolazione.
- `correct` indica se usare o meno la correzione di continuità

Si noti inoltre che il `prop test` nel caso di due proporzioni fornisce anche gli intervalli fiduciari per le differenze delle vere proporzioni fra le due popolazioni.

Il prop.test può anche essere usato valori scalari e non con vettori ossia in questo caso

- x scalare di successi
- n scalare di tentativi
- p scalare indicante una probabilità se p non è inserito viene considerata pari a 0.5
- correct indica se usare o meno la correzione di continuità

Si noti che in questo caso si ottiene anche un intervallo di confidenza per la vera proporzione dei successi nella popolazione.

Si noti che il `prop test` è l'equivalente del test di chi quadrato per l'indipendenza di una tabella di contingenza solamente che in questo caso si ragiona solamente su due risposte si o no e quindi si desume che le popolazioni di origine siano delle binomiali.

29.4 Funzione potenza nel caso di due proporzioni

Il test per calcolare la funzione potenza nel caso di due proporzioni è il seguente:

```
power.prop.test(n=NULL, p1=NULL, p2=NULL, sig.level=0.05,
power=NULL,alternative=c("two.sided", "one.sided"))
```

in cui abbiamo che:

- n numero delle osservazioni per gruppo

- p1 probabilità in un gruppo
- p2 probabilità nell'altro gruppo
- sig.level livello di significatività o errore di prima specie
- power potenze del test o 1 meno probabilità dell'errore di seconda specie
- alternative alternativa a due code o ad una sola cosa

Si noti che dei valori predefiniti con `NULL` tre devono essere dati ed in funzione di essi il comando calcola il quarto mancante.

30 I test per tabelle di contingenza

30.1 I pacchetti necessari

Per utilizzare i test previsti in questo capitolo sarà necessario aver installato il pacchetto:
`ctest`

30.2 Le tabelle di contingenza

La tabella di contingenza è una tabella a doppia entrata che raccoglie le osservazioni effettuate su un campione analizzato attraverso l'impiego di due variabili statistiche congiuntamente considerate. Il fatto più importante che interessa in una tabella di contingenza è quello di stabilire, facendo delle inferenze sul campione estratto, se nella popolazione d'origine le due variabili sono tra di loro indipendenti. Vi sono sono molti test che a tal fine possono essere utilizzati e che verranno specificati nei paragrafi successivi.

30.3 Il test di Chi quadrato

Il test di Chi quadrato per l'indipendenza viene effettuato con uno dei seguenti comandi:
`>chisq.test(z)`
`>chisq.test(x,y)`
in cui si avrà che:

- `z` è una tabella di contingenza
- `x` è la variabile risposta mentre `y` è una variabile di tipo factor

è possibile usare l'opzione `simulate.p.value=T` mediante la quale si passa dalla distribuzione esatta a quella asintotica

30.4 Il test di Fisher

Il test di Fisher per l'indipendenza viene utilizzato soprattutto per analizzare dati discontinui, sia nominali che ordinali, quando i due campioni indipendenti sono molto piccoli. Esso viene effettuato con uno dei seguenti comandi:
`>fisher.test(z)`
`>fisher.test(x,y)`
in cui si avrà che:

- `z` è una tabella di contingenza
- `x` e la variabile risposta mentre `y` è una variabile di tipo factor

30.5 Il test di Mantelhaen

Il test di Mantelhaen per l'indipendenza viene effettuato con il seguente comando:
`>mantelhaen.test(z,correct=T)`
in cui si avrà che `z` è un array di n matrici di dimensione 2 x 2.

30.6 Il test di McNemar

Il test di McNemar per l'indipendenza viene utilizzato prevalentemente per verificare l'esistenza di differenze prima e dopo un certo trattamento qualora siano disponibili dati sotto forma di frequenze. Esso viene effettuato con il seguente comando:
`>mcnemar.test(z,correct=T)`
in cui si avrà che `z` è una tavola di contingenza di dimensione 2 x 2

31 Adattamento dei dati ad una distribuzione

31.1 I pacchetti necessari

Per utilizzare i grafici previsti in questa sezione sarà necessario aver installato il pacchetto:
`stepfun`

31.2 Una prima analisi dei dati

Quando si hanno a disposizione dei dati provenienti da una indagine statistica e si tenta di adattare questi dati ad una distribuzione teorica, la prima analisi da effettuare e quella di tentare di verificare se i dati provengono da una distribuzione discreta o da una distribuzione continua. Fatto ciò dobbiamo tentare di ipotizzare tramite una analisi grafica il tipo di popolazione da cui i dati provengono. Mentre per quanto riguarda il primo punto, in generale non si presentano problemi rilevanti, l'identificazione di una distribuzione teorica tramite l'analisi grafica si presenta molte volte difficile e ardua. Tale tipo di analisi può essere effettuata utilizzando i seguenti comandi:

- `plot(table(x))`
- `hist(x)`
- `qqnorm(x)  e qqline(x)`
- `qqplot(x)`

31.3 La funzione cumulata di distribuzione

Un altro utile strumento grafico di analisi è quello legato alla funzione cumulata di distribuzione. Tale funzione ci aiuta a verificare:

- in un campione l'adattamento ad una distribuzione specifica

- in due campioni l'uguaglianza della distribuzione di provenienza degli stessi

Nel caso avessimo a disposizione un solo campione i cui dati sono memorizzati nella variabile `x`, per verificare se tale campione proviene da una distribuzione specificata si può utilizzare la seguente sintassi:

```
>plot(ecdf(x),verticals=T,do.p=F)
>y<-sort(x)
>lines(y,pfunc(y,....))
```

in cui si avrà che:

- il parametro `pfunc` sarà sostituito con i nomi di una delle distribuzioni base viste in precedenza ossia: pnorm, pbeta, pcauchy, pchisq, pexp, pf, pgamma, plnorm, plogis, pt, punif, pweibull, pbinom, pgeom, phyper, pnbinom, ppois, pwilcox
- al posto dei puntini andranno inseriti i parametri richiesti dalla funzione `pfunc` di solito stimati con i dati derivanti dal campione `x` stesso.

Ad esempio se supponiamo che `x` possa provenire da una distribuzione normale potremmo operare nel seguente modo:

```
>plot(ecdf(x),verticals=T,do.p=F)
>y<-sort(x)
>lines(y,pnorm(y,mean(y),sd(y)))
```

Nel caso avessimo a disposizione due campioni, per verificare se essi provengono dalla stessa distribuzione si può utilizzare la seguente sintassi:

```
>plot(ecdf(x),verticals=T,do.p=F)
>plot(ecdf(y),verticals=T,do.p=F,add=T)
```

in cui `x ed y` sono vettori che possono anche assumere diversa lunghezza.

31.4 Le funzioni qqnorm, qqline e qqplot

Un metodo alternativo per verificare se un vettore proviene da una distribuzione di probabilità o se due vettori provengono dalla stessa distribuzione di probabilità è quello che fa riferimento all'uso delle funzioni `qq`. Si possono allora distinguere vari casi.

Se abbiamo a disposizione un vettore di osservazioni `x` è vogliamo verificare se esso proviene da una distribuzione normale possiamo utilizzare i comandi:

```
>qqnorm(x)
>qqline(x)
```

Per verificare invece se i dati provengono da altre distribuzioni possiamo usare la seguente sintassi:

```
>plot(qfunc(ppoints(x),...),sort(x))
>aaa<-quantile(x,c(0.25,0.75))
>bbb<-qfunc(c(0.25,0.75),...)
>coeff<-(aaa[2]-aaa[1])/(bbb[2]-bbb[1])
>inter<-aaa[1]-coeff*bbb[1]
>abline(inter,coeff)
```

in cui si avrà che:

- il parametro `qfunc` sarà sostituito con i nomi di una delle distribuzioni base viste in precedenza ossia: pnorm, pbeta, pcauchy, pchisq, pexp, pf, pgamma, plnorm, plogis, pt, punif, pweibull, pbinom, pgeom, phyper, pnbinom, ppois, pwilcox

- al posto dei puntini andranno inseriti i parametri richiesti dalla funzione `pfunc` di solito stimati con i dati derivanti dal campione `x` stesso.

Per verificare invece se i dati di due vettori `x` ed `y` provengono dalla stessa distribuzione possiamo usare la seguente sintassi:

```
>qqplot(x,y)
>aaa<-quantile(x,c(0.25,0.75))
>bbb<-quantile(y,c(0.25,0.75))
>coeff<-(aaa[2]-aaa[1])/(bbb[2]-bbb[1])
>inter<-aaa[1]-coeff*bbb[1]
>abline(inter,coeff)
```

31.5 Il test chi quadrato la bontà dell'adattamento nel caso di distribuzioni discrete

Il test chi quadrato per valutare la bontà dell'adattamento viene effettuato in modo diverso a seconda della distribuzione oggetto di adattamento. In ogni caso si deve ricordare che per usare i test che vedremmo successivamente il confronto è fatto sempre tra due vettori, il primo che raccoglie le frequenze osservate e che indicheremo con `fi` mentre il secondo raccoglie le probabilità stimate e che abbiamo inserito nella variabile `p`. Si avrà allora che:

- se la distribuzione a cui si vogliono adattare i dati è una `distribuzione binomiale` possiamo utilizzare la seguente sintassi:
  ```
  >fi<-c(15,45,76,51,13)
  >pro<-dbinom(0:4,4,0.5)
  >chisq.test(fi,p=pro)
  ```
  in cui il valore di 0.5 è stato scelto come esempio ma è proprio il parametro più importante da prendere in considerazione. Il valore di `p` di solito viene stimato dai dati campionari a disposizione.

- se la distribuzione a cui si vogliono adattare i dati è una `distribuzione uniforme` possiamo utilizzare la seguente sintassi:
  ```
  >fi<-c(15,45,76,51,13)
  >chisq.test(fi)
  ```

- se la distribuzione a cui si vogliono adattare i dati è una `distribuzione multinomiale` possiamo utilizzare la seguente sintassi:
  ```
  >fi<-c(15,45,76,51,13)
  >pro<-c(a,b,c,d,e)
  >chisq.test(fi,p=pro)
  ```
  in cui i valore di `a,b,c,d,e` sono le ipotizzate probabilità per ogni frequenza ottenuta di solito stimate attraverso i dati campionari otenuti. Si noti che se si vuole provare che le probabilità effettive sono uguali basterà usare il comando: `>chisq.test(fi)`
  in quanto sarà automaticamente ipotizzato che la probabilità delle frequenze teoriche siano per ogni frequenza assoluta pari ad `1/5`.

31.6 Il test chi quadrato la bontà dell'adattamento nel caso di distribuzioni continue

Se la distribuzione a cui si vogliono adattare i dati è una distribuzione continua, ad esempio una normale la procedura da seguire è più articolata. Dobbiamo prima di tutto ragruppare le frequenze assolute ottenute dall'analisi campionaria in classi ed ottenere quindi una tabella tipo la seguente:

<55	55-60	60-65	65-70	70-75	75-80	80-85	>85
8	68	272	680	617	283	57	15

Se vogliamo verificare se il campione ottenuto proviene da una `N(70,25)`. La sintassi da usare è la seguente:

```
>fi<-c(8,68,272,680,617,283,57,15)
>y1<-seq(55,85,5)
y2<-seq(50,80,5)
>y1<-(y1-70)/5
y2<-(y2-70)/5
>xxx<-pnorm(y1)-pnorm(y2)
>xxx[8]<-1-pnorm(3)
>chisq.test(fi,p=xxx)
```

31.7 Il test di Kolmogorov-Smirnov per la bontà dell'adattamento

Il test di Kolmogorov-Smirnov viene effettuato con il seguente comando:
`>ks.test(x,y=null,`"pfunc",...,alt="two.sided")
in cui si ha che:

- il parametro `pfunc` sarà sostituito con i nomi di una delle distribuzioni base viste in precedenza ossia: pnorm, pbeta, pcauchy, pchisq, pexp, pf, pgamma, plnorm, plogis, pt, punif, pweibull, pbinom, pgeom, phyper, pnbinom, ppois, pwilcox
- al posto dei puntini andranno inseriti i parametri richiesti dalla funzione `pfunc` di solito stimati con i dati derivanti dal campione `x` stesso.
- alt può essere greater e less

31.8 Il test di Shapiro

Per verificare la normalità dei dati su cui vogliamo fare un'analisi è anche possibile usare il test di Shapiro che ha la seguente sintassi:

```
>shapiro.test(x)
```

in cui x è il vettore dei dati da analizzare.

32 Ricerca degli zeri ed ottimizzazione di una funzione

32.1 Introduzione

In R sono presenti alcune funzioni di particolare importanza che sono adatte per la ricerca degli zeri e per l'ottimizzazione di funzioni ad una o più variabili. Tali funzioni saranno descritte dettagliatamente nei paragrafi che seguiranno. In tutti i casi in cui è richiesto l'uso di una funzione questa potrà essere assegnata in due modi distinti:

- definendola direttamente come visto nel paragrafo relativo alla programmazione ossia nel seguente modo:
  `>ff<-function(x){ }`
  e quindi nel comando utilizzeremo per richiamare la funzione il suo nome ossia ff
- definendola direttamente nel comando con la seguente sintassi:
  `>comando(function(x) { }.....)`

Si noti che i due modi di procedere sono alternativi ma equivalenti.

32.2 La funzione uniroot

La funzione uniroot serve per trovare gli zeri di una funzione ad una variabile. Il suo uso è il seguente:

```
>uniroot(f, interval)
```

Si noti il seguente esempio:

```
>f <- function (x,a) x - a
>uniroot(f, c(0, 1), tol = 0.0001, a = 1/3)
```

Con tale comando abbiamo ottenuto gli zeri di f nell'intervallo [0,1] e si noti che è possibile stabilire anche la tolleranza per il calcolo delle radici stesse.

32.3 La funzione optimize

La funzione optimize serve per trovare il massimo o il minimo di una funzione in un intervallo specificato. Il suo uso è il seguente:

```
>optimize(f, interval , maximum = FALSE)
```

Si noti il seguente esempio:

```
>f <- function (x,a) (x-a)^2
>optimize(f, c(0, 1), tol = 0.0001, a = 1/3)
```

Si noti che in tale modo abbiamo ottenuto il minimo della funzione f nell'intervallo [0,1], per ottenere il massimo avremmo dovuto specificare l'opzione maxim=T.

32.4 La funzione optim

La funzione optim è usata prevalentemente per trovare il minimo di un sistema di equazioni non lineari. Tale ricerca è effettuata utilizzando vari metodi di ottimizzazione ossia Nelder-Mead, quasi-Newton e conjugate-gradient algorithms. Il suo uso è il seguente:

```
>optim(par, fn, gr = NULL, method = c("Nelder-Mead", "BFGS", "CG", "L-BFGS-B",
 SANN"), lower = -Inf, upper = Inf, control = list(), hessian = FALSE,...)
```

Si noti il seguente esempio:

```
fr <- function(x) {   ## Rosenbrock Banana function
                    x1 <- x[1]
                    x2 <- x[2]
                    100 * (x2 - x1 * x1)^2 + (1 - x1)^2
                    }
grr <- function(x) { ## Gradient of 'fr'
                    x1 <- x[1]
                    x2 <- x[2]
                    c(-400 * x1 * (x2 - x1 * x1) - 2 * (1 - x1),
                    200 *      (x2 - x1 * x1))
                    }
optim(c(-1.2,1), fr)
optim(c(-1.2,1), fr, grr, method = "BFGS")
optim(c(-1.2,1), fr, NULL, method = "BFGS", hessian = TRUE)
optim(c(-1.2,1), fr, grr, method = "CG")
optim(c(-1.2,1), fr, grr, method = "CG", control=list(type=2))
optim(c(-1.2,1), fr, grr, method = "L-BFGS-B")
```

Tale funzione può anche essere usata per ottimizzare una funzione ad una variabile.

32.5 La funzione simplex

La funzione simplex serve per ottimizzare una funzione lineare soggetta a vincoli lineari utilizzando il metodo del simplesso. Tale funzione richiede l'uso del pacchetto boot. Si noti che i vincoli sono espressi nel seguente modo:

- a rappresenta il vettore dei coefficienti della funzione obiettivo
- $A1 * x <= b1$
- $A2 * x >= b2$
- $A3 * x = b3$

in cui ovviamente vale il vincolo di segno che $x >= 0$. La sintassi dell'uso di questa funzione è quindi la seguente:

```
>simplex(a, A1=NULL, b1=NULL, A2=NULL, b2=NULL,A3=NULL,
b3=NULL, maxi=FALSE, n.iter=n + 2 *m,eps=1e-10)
```

Un esempio di uso di tale funzione è il seguente:

```
>enj <- c(200, 6000, 3000, -200)
>fat <- c(800, 6000, 1000, 400)
>vitx <- c(50, 3, 150, 100)
>vity <- c(10, 10, 75, 100)
>vitz <-c(150, 35, 75, 5)
>simplex(a=enj, A1=fat, b1=13800, A2=rbind(vitx,vity,vitz),
b2=c(600, 300, 550), maxi=TRUE).
```

33 Formule di quadratura di una funzione

34 La regressione lineare

34.1 Introduzione

La regressione lineare consiste in una serie di tecniche volte principalmente alla ricerca di un modello con cui una variabile dipende linearmente da un'altra serie di variabili. I dati per lo studio della regressione lineare sono composti principalmente da un data frame che nel corso del paragrafo chiameremo sempre `anar` composto da più variabili, la prima che chiameremo sempre `y` ,sempre di tipo numerico, rappresenta la variabile dipendente mentre le altre che indicheremo con `x1,x2,...`, sempre di tipo numerico o factor, rappresentano le varibili indipendenti che compongono il modello stesso.

34.2 Analisi grafica

Un primo approccio relativo all'analisi della regressione che serve per valutare quali variabili sono tra loro legate da un legame di tipo lineare è l'approccio grafico. Tale analisi viene compiuta con il comando:
`>pairs(anar)`
In tale modo siamo in grado di poter visualizzare i possibili legami di tipo linearetra le variabili.

34.3 La regressione lineare ai minimi quadrati

La regressione lineare ai minimi quadrati viene effettuata con il comando:
`>anar.lm<-lm(y~x1+x2+...+xn,data=anar)`
se siamo interessati ad ottenere la regressione senza il termine noto il comando diviene:
`>anar.lm<-lm(y~-1+x1+x2+...+xn,data=anar)`
Se anar è un data frame contenente come prima variabile la varibile dipendente e come successive variabili quelle indipendenti, per effettuare la regressione può essere usata la seguente sintassi semplificata:
`>anar.lm<-lm(anar)`

Se l'analisi non deve essere compiuta su tutti i dati a disposizione ma solamente su un sottoinsieme di dati che può essere ottenuto anche da un'altra variabile presente nel data frame `anav`, possiamo effettuare l'analisi con il comando:
`>anar.lm<-lm(y~x1+x2+...xn,subset=(variabile=='valore')`
in cui

- `variabile` indica in nome della variabile attraverso la quale estraiamo il sottoinsieme di dati che ci interessano
- l'operatore relazionale `==` può essere sostituito a seconda dei casi da un qualunque altro operatore relazione
- `valore` indica il criterio di estrazione dei dati

L'opzione subset può anche essere usata in questo modo, supponiamo che da una analisi grafica o altro abbiamo deciso che alcuni valori del data frame non debbano essere usati nel calcolo. Stabiliti tali valori ad esempio quelli della riga1,riga5 e riga10 del data frame `anar` dobbiamo costruire un vettore:
`outliers<-c(1,5,10)`
e dare il comando:

`>ana<-lm(y~x1,subset=-outliers`
in tale modo vengono utilizzate solamente le righe del data frame `anar` escluse duelle indicate in `outliers`.

Una volta effettuata tale regressione per visualizzare vedere le informazioni ottenute possiamo usare i seguenti comandi:
`>anar.lm` per vedere i dati salienti della regressione
`>summary(anar.lm)` per vedere informazioni più dettagliate sulla regressione
`>coefficients(anar.lm)` per vedere aolamente i coefficienti stimati tramite la regressionei
`>residuals(anar.lm)` per vedere i residui della regressione
`>fitted.values(xxx.lm)` per vedere i valori stimati della regressione

Da tali comandi individuiamo una serie di informazioni utili per una corretta stima del modello ai minimi quadrati, si noti che tra l'altro che il `residual standard error` fornito da tali comandi è la stima di σ e non di σ^2 come tra l'altro era facile intuire dal nome utilizzato.

34.4 Anova nella regressione

Con il comando `>anova(anar.lm)`
si ottiene una tavola della analisi della varianza relativa alla regressione in oggetto da cui è facile calcolare il valore di R^2 dato da:

$$R^2 = 1 - \frac{devianzaresidua}{devianzatotale}$$

mentre il test F sarà dato da:

$$F = \frac{devianzaspiegata/(k-1)}{devianzatotale/(n-k)}$$

in cui k sono i parametri stimati della regressione mentre n rappresenta la numerosità del campione.

34.5 I grafici della regressione

Supponendo di aver costruito un modello di regressione lineare e averlo memorizzato nella variabile `xxx-lm` possiamo studiare tale modello anche considerando i seguenti grafici:

- grafico dei residui ottenuto con il comando:
  `>plot(1:n,residuals(xxx.lm))`
  che è utile soprattutto per evidenziare presenze di correlazioni tra le osservazioni

- grafico dei residui rispetto alle variabili esplicative ottenuto con il comando:
  `>plot(xi,residuals(xxx.lm))`
  fatto per ogni variabile esplicativa x1, x2, xn, che sono utili per individuare scorrette specificazioni della dipendenza dalle variabili esplicative, come ad esempio dipendenze non lineari.

- grafico dei residui rispetto ai valori stimati dal modello ottenuto con il comando:
  `>plot(fitted(xxx.lm),residuals(xxx.lm))`
  utile per verificare se le ipotesi base del modello ossia media zero dei residui, omoschedasticità, e correlazione nulla dei residui siano verificate.

Si noti che alcuni di questi grafici possono essere ottenuti con il comando:
`>plot(xxx.lm)`
e che tra i grafici ottenuti con tale comando figura anche quello basato sulla distanza di Cook, utile per individuare la presenza di valori anormali.

34.6 Normalità dei residui

Uno degli aspetti fondamentali della regressione lineare consiste nel fatto che i residui della regressione siano distribuiti normalmente. Per verificare tale fatto possiamo utilizzare anche congiuntamente due strumenti:

- l'analisi grafica
- i test di adattamento e quelli di normalità

34.7 Regressione su un insieme limitato di dati

Per effettuare la regressione su un insieme limitato di dati dovremmo utilizzare il comando:
`>xxx.ln<-lm(y~x1+x2+...+xn,subset=.....)`
in cui i puntini vanno sostituiti con un vettore soddisfante una condizione particolare. Se ad esempio disponiamo di un data frame con variabili vento, temperatura e mesi e vogliamo effettuare una regressione della temperatura sul vento ma considernado solamente i dati del mese 5 dobbiamo utilizzare la sintassi:
`>xxx.ln<-lm(temperatura~vento,subset=mese==5)`

34.8 Le variabili da considerare nella regressione

Quando si hanno a disposizione più varibili indipendenti c'è il problema di stabilire quali considerare nella regressione e quali invece escludere dalla regressione stessa. Il primo approccio da utilizzare è quello grafico che si ottiene con il comando
`>pairs(xxx)`
in cui xxx è il data frame che contiene sia la variabile dipendente che quella indipendente. Dall'analisi grafica è possibile quindi evidenziare le relazioni di tipo lineare che legano ciascuna variabile indipendnete a quella dipendnete e stabilire così in prima approssimazione le variabili indipendneti da utilizzare.

Se abbiamo già costruito un modello con il comando:
`>xxx.lm<-lm(y~x1+x2+...+xn)`
è possibile decidere quali varibili eliminare dalla regressione tramite l'utilizzo del comando `drop1` il quale mi indica la variabile da togliere al modello in base al metodo dell'AIC. In pratica tramite il comando:
`>drop1(xxx.lm)`
viene evidenzata per ogni variabile presente nella regressione il suo AIC. Dovrà allora uscire dalla regressione la variabile che presenta l'AIC più basso. Decisa quindi la varibile da far uscire dalla regressione che supponiamo essere x2 l'aggiornamento del modello avviene con il comando:
`>update(xxx.lm,.~.-x2)`

Se stiamo partendo dall'inizio e non abbiamo nessuna idea di quali varibili aggiungere al modello possiamo utilizzare il comando `add1` il quale indica la variabile da aggiungere alla regressione in base al metodo dell'AIC. Il suo utilizzo è il seguente, prima di tutto viene costruito il modello comprendente solo l'intercetta con

`>xxx.ln<-lm(y~1)`
e tramite il comando
`>add1(xxx.lm,~x1+x2+x3+...+xn)`
viene evidenzata per ogni variabile il sui AIC. Dovrà allora entrare nella regressione la variabile che presenta l'AIC più basso. Decisa quindi la variabile da far entrare nella regressione che supponiamo essere x3 l'aggiornamento del modello avviene con il comando:
`>update(xxx.lm,.~.+x3)`
Si noti chè è possibile utilizzare il comando `add1` anche con un modello già precedentmente stimato in questo caso il comando dovrà essere utilizzato nel seguente modo:
`>add1(xxx.lm,~x1+x2+x3+...+xn)`
in cui però x1, x2 x3 ,....,xn sono sia le variabili della regressione già stimata sia quelle che vogliamo aggiungere.

Per ottenere immediatamente il miglior modello sempre in base all'AIC si può utilizzare il comando `step` il quale fornisce il miglior modello stimabile in base al metodo dell'AIC. Il suo uso è il seguente, prima di tutto si procede alla costruzione del modello avente solo l'intercetta con :
`>xxx.ln<-lm(y~1)`
successivamente con il comando: `>step(xxx.lm,~x1+x2+x3+...+xn)`
viene automaticamente creato il modello mostrando tutti i passaggi che sono stati effettuati. Se non si desidera vedere i passaggi ma ottenere il miglio modello stimabile si deve utilizzare la seguente sintassi:
`>step(xxx.lm,~x1+x2+x3+...+xn,trace=F)`

34.9 Regressione con una funzione predefinita

Nel caso in cui dovessimo effettuare una regressione ai minimi quadrati utilizzando una funzione predefinita ad esempio la funzione

$$f(x) = a + bx + cx^2$$

oppure con la funzione

$$f(x) = \frac{a}{x} + bx^2$$

il comando da utilizzare nel primo caso sarà
`lm(y~I(x)+I(x^2))'`
mentre per il secondo caso sarà
`lm(y~I(1/x)+I(x^2)`

34.10 La previsione

Una volta stimato il modello di regressione e memorizzato nella variabile `anar.lm` è possibile calcolare i valori previsti con il seguente comando:
`>predict(anar.lm)`
se vogliamo calcolare i valore previsti in punti diversi dal data frame utilizzato per stimare i coefficiente possiamo utilizzare il seguente comando:
`>predict(anar.lm,nuovodataframe)`
in cui `nuovodataframe` sarà un data frame con i nomi delle variabili utilizzate per la stima dei coefficienti ma aventi valori e numerosità campionaria diversa.

Se vogliamo calcolare anche lo `standar error` di tale previsione possiamo utilizzare il seguente comando:
`>predict(anar.lm,se.fit=T)`

Per ottenere invece gli intervalli di confidenza dei valori predetti si usa la sintassi:
`>predict(anar.lm,interval=`"none","confidence,prediction")
in cui l'opzione `interval` potrà essere `confidence o prediction` a seconda della scelta effettuata.

34.11 Lo stimatore ai minimi quadrati generalizzati

Se dall'analisi grafica effettuata con lo scatter plot dei valori stimati contro i valori residui otteniamo che tale grafico al posto di essere contenuto in una strisica regolare di piano ha un andamento diverso, l'ipotesi di omoschedasticità non può più essere ritenuta valida e quindi il modello dovrà essere stimato considerando lo stimatore ai minimi quadrati generalizzato. Senza entrare in ulteriori questioni di tipo teorico, basti osservare che è possibile ottenere tale stimatore, ma prima di tutto si dovranno caricare i seguenti pacchetti aggiuntivi:

- `nls`
- `nlme`

caricati tali pacchetti, la stima potrà essere e ffettuata con il comando:
`>gls(y~x1+x2+....,weights=varFunc(formula))`
in cui formula rappresenta la struttura della funzione di varianza. Ad esempio se l'eteroschedasticità è dovuta al fatto che la matrice Ω presenta sulla diagonale principale i reciproci dei numeri`12,6,11,10,11` possiamo dare il seguente comando:
`>n<-c(12,6,11,10,11)`
`>gls(y~x1+x2+....,weights=varFunc(~1/n))`
naturalmente si suppone che l'analisi è fatta su cinque osservazioni. Si noti che è possibile ottenere più formulazioni della funzione di varianza.

Un altro problema che si riscontra nel caso di violazioni di ipotesi che stanno alla base dello stimatore ai minimi quadrati ordinario è quello relativo al caso in cui chi errori non siano tra di loro indipendenti ma siano correlati. Un metodo per visualizzare graficamente questo caso è quello di effettuare uno scatter plot dei residui al tempo t e di quelli al tempo $t-k$ con $k = 1..n$ e vedere se vi sono andamenti regolari. Tra i vari tipo di correlazione dei residui quella più studiata è la correlazione di tipo `Ar1`. Nell'ipoesi che $\phi = 0.10''$ lo stimatore ai minimi quadrati generalizzato potrà essere ottenuto con il seguente comando:
`>gls(y~x1+x2+....,correlation=corAR1(0.10,form=~1,T))`
Si noti che è possibile ottenere più formulazioni della funzione di correlazione.

35 L'analisi della varianza

35.1 Introduzione

L'analisi della varianza consiste in una serie di tecniche volte principalmente alla verifica dell'uguaglianza delle medie tra popolazioni sottoposte a trattamenti differenti. I dati per lo studio dell'analisi della varianza sono composti principalmente da un data frame che nel corso del paragrafo chiameremo sempre `anav` composto da più variabili, la prima che chiameremo sempre `y` ,sempre di tipo numerico, rappresenta il vettore delle osservazioni mentre le altre che indicheremo con `x1,x2,...`, sempre di tipo factor, rappresentano i fattori a cui sono state sottoposte le singole osservazioni. Ogni fattore sarà composto da più modalità. Si avrà allora che:

- nel caso di un solo fattore per trattamenti si intendono le modalità con cui si manifesta il fattore medesimo
- nel caso di due fattori i trattamenti per trattamenti si intendono tutte le coppie di modalità con cui si manifestano i due fattori in gioco
- nel caso di tre o più fattori l'estenzione è naturale

E' molto importante che i dati delle variabili `x1,x2,...` siano di tipo factor, in caso contrario i risultati potrebbero essere errati. Si noti in ogni caso che se i dati sono inseriti in un file ed importati con il comando `read.table` essi vengono, in caso non siano numerici, convertiti automaticamente in un oggetto di tipo factor. Bisognerà quindi prestare attenzione al caso in cui i dati delle varibili suddette siano numerici e quindi una volta importati dovranno essere convertiti manualmente in oggetti di tipo factor.

35.2 Analisi con un solo fattore

Ci occuperemo in questo paragrafo del caso più semplice dell'analisi della varianza ossia quella relativa al fatto di avere a disposizione il data frame `anav` composto dalla variabile risposta `y` e da un'unica varuiabile fattore `x1` avente come modalità `a,b,c,d`.

Una prima analisi dei dati del data frame `anav` finalizzata ad avere un primo tipo di risposta relativa alla uguaglianza delle medie relative ai diversi trattamenti a cui è stata sottoposta la variabile fattore `x1` consiste nella costruzione di un grafico che evidenzia le medie e le mediane delle variabili dei singoli trattamenti. Ciò può essere effettuato tramite la seguente sequenza di comandi:

```
>attach(anav)
>par(mfrow=c(1,2))
>plot(c(1,1,1,1),tapply(y,x1,mean),type="n") il numero di 1 dipende dal numero di modalità di x1
>text(c(1,1,1,1),tapply(y,x1,mean),c("a","b","c","d"))
>plot(c(1,1,1,1),tapply(y,x1,median),type="n") il numero di 1 dipende dal numero di modalità di x1
>text(c(1,1,1,1),tapply(y,x1,median),c("a","b","c","d"))
```

Un altro grafico molto interessante da analizzare è il boxplot relativo alla della variabile `y` suddiviso per le singole modalità di `x1`. Tale grafico si ottiene con il comando:

```
>plot(y~x1)
```

E possibile costruire una funzione la quale automatizza la procedura per la creazione dei grafici.

L'analisi della varianza ad un fattore viene effettuata con il seguente comando:
`>ana<-aov(y~x1)`
`>summary(ana)`
Molto interessante è anche il caso in cui l'analisi non deve essere compiuta su tutti i dati a disposizione ma solamente su un sottoinsieme che può essere ricavato anche da an'altra variabile presente nel data frame `anav`. In questo caso dovremo dare il seguente comando:
`>ana<-aov(y~x1,subset=(variabile==valore)`
`>summary(ana)`
in cui variabile indica in nome della variabile attraverso la quele estraiamo i dati desiderati, il simbolo == può essere sostituito a seconda dei casi da un qualunque operatore relazione, in nome valore indica il criterio di estrazione dei dati.

L'opzione subset può anche essere usata in questo modo, supponiamo che da una analisi grafica o altro abbiamo deciso che alcuni valori del data frame non debbano essere usati nel calcolo. Stabiliti tali valori ad esempio quelli della riga1,riga5 e riga10 dobbiamo costruire un vettore:
`outliers<-c(1,5,10)`
e dare il comando:
`>ana<-aov(y~x1,subset=-outliers`
`>summary(ana)`

Si noti che l'analisi della varianza ad un fattore equivale ad una analisi di regressione con i minimi quadrati in cui le variabili indipendenti sono tante variabili di tipo variabili dummy che valgono 1 se la risposta proviene dalla modalità del fattore e zero in caso contrario quante sono le modalità con cui si presenta il fattore `x1`. A questo punto per compiere una più approfondita analisi possiamo utilizzare i seguenti comandi: `>plot(ana)` per visualizzare i grafici relativi all'analisi della varianza
`>coefficients(ana)` per vedere i coefficienti della regressione ai minimi quadrati
`>residuals(ana)` per vedere i residui della regressione ai minimi quadrati
`>fitted.values(ana)` per vedere i valori stimati della regressione
`>model.tables(ana)`
`>model.tables(ana,type=`"means")

Compiuta l'analisi della varianza, possiamo passare all'analisi dei residui volta ad identificare se i residui derivanti dalla analisi fatta abbiano una distribuzione di tipo normale oppure no. Infatti affinchè l'analisi della varianza sia soddisfacente i residui dovrebbero disporsi in forma di una normale. Tale analisi può essere effettuata con le tecniche viste nel capitolo relativo all'adattamento dei dati ad una distribuzione prefissata, in ogni caso è anche molto utile un'analisi grafica dei residui che si ottiene attraverso i seguenti comandi:
`>hist(resid(ana))`
`>qqnorm(resid(ana))`.

Ricordiamo inoltre che per effettuare l'analisi della varianza è essenziale che le varianze delle popolazioni da cui i campioni provengono siano uguali. Ciò può essere verificato utilizzando il test di bartlett con il seguente comando:
`>bartlett.test(y~x1)`
dal quale si evince se si deve accettare o respingere l'uguaglianza delle varianze tra le varie popolazioni.

Nel caso in cui l'analisi della varianza porti a rifiutare l'ipotesi della uguaglianza delle medie una prima indicazione realtiva a quali possono essere le coppie di medie tra loro diverse potrà essere effettuata utilizzando i test legati alla t di student ed in particolare si potrà utilizzare il

seguente comando
`>pairwise.t.test(y,x1)`

35.3 Analisi con due fattori non replicati

L'analisi della varianza con due fattori non replicati si ha quando si prendono in considerazione due fattori e per ogni coppia di modalità relativa ai due fattori si effettua una sola osservazione. Sia il data frame `anav` composto dalla variabile risposta `y` , da una variabile fattore `x1` avente come modalità `a,b,c,d` e da una variabile fattore `x2` avente come modalità `A,B,C,D`. In questo caso per ogni combinazione di modalità dei fattori `x1` e `x2` supponiamo di disporre di una sola osservazione `y`.

Una prima analisi dei dati del data frame `anav` finalizzata ad avere un primo tipo di risposta relativa alla uguaglianza delle medie relative ai diversi trattamenti a cui è stata sottoposta la variabile fattore `x1` consiste nella costruzione di un grafico che evidenzia le medie e le mediane delle variabili dei singoli trattamenti. Ciò può essere effettuato tramite la seguente sequenza di comandi:
`>attach(anav)`
`>par(mfrow=c(1,2))`
`>plot(c(1,1,1,1),tapply(y,x1,mean),type=`"n") il numero di 1 dipende dal numero di modalità di `x1`
`>text(c(1,1,1,1),tapply(y,x1,mean),c(`"a","b","c","d"))
`>plot(c(1,1,1,1),tapply(y,x1,median),type=`"n") il numero di 1 dipende dal numero di modalità di `x1`
`>text(c(1,1,1,1),tapply(y,x1,median),c(`"a","b","c","d"))
Un altro grafico molto interessante da analizzare è il boxplot relativo alla della variabile `y` suddiviso per le singole modalità di `x1`. Tale grafico si ottiene con il comando:
`>plot(y~x1)`
I comandi per il fattore `x2` sono analoghi.

Nel caso di due fattori è anche necessario analizzare le interazioni tra le modalità dei due fattori considerati e tale analisi può essere compiuta con il comando:
`>interaction.plot(y,x1,x2)`
`>interaction.plot(y,x1,x2,median)`

L'analisi della varianza a due fattori viene effettuata con il seguente comando:
`>ana<-aov(y~x1+x2)`
`>summary(ana)`

A questo punto per compiere una più approfondita analisi possiamo utilizzare i seguenti comandi: `>plot(ana)` per visualizzare i grafici relativi all'analisi della varianza
`>coefficients(ana)` per vedere i coefficienti della regressione ai minimi quadrati
`>residuals(ana)` per vedere i residui della regressione ai minimi quadrati
`>fitted.values(ana)` per vedere i valori stimati della regressione
`>model.tables(ana)`
`>model.tables(ana,type=`"means")

Compiuta l'analisi della varianza, possiamo passare all'analisi dei residui volta ad identificare se i residui derivanti dalla analisi fatta abbiano una distribuzione di tipo normale oppure no. Infatti affinchè l'analisi della varianza sia soddisfacente i residui dovrebbero disporsi in forma di una normale. Tale analisi può essere effettuata con le tecniche viste nel capitolo relativo all'adattamento dei dati ad una distribuzione prefissata, in ogni caso è anche molto utile un'analisi

grafica dei residui che si ottiene attraverso i seguenti comandi:
`>hist(resid(ana))`
`>qqnorm(resid(ana))`.

Ricordiamo inoltre che per effettuare l'analisi della varianza è essenziale che le varianze delle popolazioni da cui i campioni provengono siano uguali. Ciò può essere verificato utilizzando il test di bartlett con il seguente comando:
`>bartlett.test(y~x1)`
`>bartlett.test(y~x2)`
dal quale si evince se si deve accettare o respingere l'uguaglianza delle varianze tra le varie popolazioni.

Nel caso in cui l'analisi della varianza porti a rifiutare l'ipotesi della uguaglianza delle medie una prima indicazione realtiva a quali possono essere le coppie di medie tra loro diverse potrà essere effettuata utilizzando i test legati alla t di student ed in particolare si potrà utilizzare il seguente comando
`>pairwise.t.test(y,x1)` `>pairwise.t.test(y,x2)`

35.4 Analisi con due fattori replicati

L'analisi della varianza con due fattori non replicati si ha quando si prendono in considerazione due fattori e per ogni coppia di modalità relativa ai due fattori si effettuano più osservazioni. Sia il data frame `anav` composto dalla variabile risposta `y` , da una variabile fattore `x1` avente come modalità `a,b,c,d` e da una variabile fattore `x2` avente come modalità `A,B,C,D`. In questo caso per ogni combinazione di modalità dei fattori `x1` e `x2` supponiamo di disporre di più osservazioni di `y`.

A differenza del caso precedente, in questa situazione possiamo anche effettuare la costruzione di un modello non solo additivo ma che tenga conto delle interrelazioni tra i due fattori. In questo caso dunque oltre il modello:
`>ana<-aov(y~x1+x2)`
potremmo considerare anche il modello:
`>ana<-aov(y~x1*x2)`

35.5 Analisi con più di due fattori

In questo caso per ottenere l'analisi della varianza basterà generalizzare quanto detto nei paragrafi precedenti.

35.6 Confronti multipli

Se l'analisi della varianza porta a rifiutare l'ipotesi dell'uguaglianza delle medie tra le varie modalità di un unico fattore potremmo essere interessati a rilevare quale coppia di modalità ha creato il rigetto di tale ipotesi. Questa analisi consiste nel calcolo dei confronti multipli. Se siamo interessati solamente al metodo dei confronti multipli di Tukey possiamo eseguire i seguenti comandi:

```
 > ana<-aov( y~x1)
> tHSD<-TukeyHSD(ana, "x1",ordered=FALSE)
> tHSD
> plot(tHSD)
```

Se l'analisi invece è a due fattori dobbiamo utilizzare i seguenti comandi:

```
> ana<-aov( y~x1+x2)
> tHSD<-TukeyHSD(ana, "x1 o x2",ordered=FALSE)
> tHSD
> plot(tHSD)
```

Se invece siamo interessati ad altri metodi di confronto dobbiamo installare i pacchetti:

- mvtnorm
- multcomp

ed eseguire i seguenti comandi:

```
> mca<-simint(y~x1, data=anav,method="Tukey")
> mca
```

Se l'analisi invece è a due fattori dobbiamo utilizzare i seguenti comandi:

```
> mca<-simint(y~x1+x2, data=anav,whichf="x1",method="Tukey")
> mca
```

I metodi che possono essere utilizzati sono i seguenti:

- Tukey
- Dunnett
- Sequen
- AVE
- Changepoint
- Williams
- Marcus
- McDermott
- Tetrade

Il comando `simint` presenta una serie di opzioni molto interessanti si veda l'help in linea per il loro utilizzo.

Si noti inoltre che con l'uso di `simint` è anche possibile ottenere altri valori molto utili per l'analisi statisitca dati da:

- il punto critico che si ottiene con `mca\$calpha`
- gli standar error che si ottengono con `\$sd`

35.7 Maggiori dettagli nell'analisi della varianza

Da un'analisi statistica effettuata abbiamo ricavato i dati che sono evidenziati nella tabella 1 L'analisi della varianza come visto si ottiene con il comando:

```
>survival.aov<-aov(dati~poison*treatment)
```

in quanto si tratta di una analisi della varianza a due fattori replicati. Per ottenere le informazioni sull'analisi della varianza possiamo eseguire il comando:

```
summary(survival.aov)
```

ma possiamo anche ottenere informazioni più dettagliate utilizzando il seguente comando:

```
summary(survival.aov,split=list(poison=list(1"=1,"2"=2,"3"=3,"4"=4),
treatment=list(a"=1,"b"=2,"c"=3))
```

dati	poison	treatment
0,31	1	a
0,45	1	a
0,46	1	a
0,43	1	a
0,36	2	a
0,29	2	a
0,4	2	a
0,23	2	a
0,22	3	a
0,21	3	a
0,18	3	a
0,23	3	a
0,82	1	b
1,1	1	b
0,88	1	b
0,72	1	b
0,92	2	b
0,61	2	b
0,49	2	b
1,24	2	b
0,3	3	b
0,37	3	b
0,38	3	b
0,29	3	b
0,43	1	c
0,45	1	c
0,63	1	c
0,76	1	c
0,44	2	c
0,35	2	c
0,31	2	c
0,4	2	c
0,23	3	c
0,25	3	c
0,24	3	c
0,22	3	c
0,45	1	d
0,71	1	d
0,66	1	d
0,62	1	d
0,56	2	d
1,02	2	d
0,71	2	d

Tabella 1: varianza

35.8 Analisi della varianza multivariata

L'analisi dellavarianza multivariata viene eseguita con il comando `manova` nel seguente modo. Supponiamo di disporre di un data frame chiamato `Y` ottenuto nel seguente modo:

```
tear <- c(6.5, 6.2, 5.8, 6.5, 6.5, 6.9, 7.2, 6.9, 6.1, 6.3,
                6.7, 6.6, 7.2, 7.1, 6.8, 7.1, 7.0, 7.2, 7.5, 7.6)
gloss <- c(9.5, 9.9, 9.6, 9.6, 9.2, 9.1, 10.0, 9.9, 9.5, 9.4,
                9.1, 9.3, 8.3, 8.4, 8.5, 9.2, 8.8, 9.7, 10.1, 9.2)
opacity <- c(4.4, 6.4, 3.0, 4.1, 0.8, 5.7, 2.0, 3.9, 1.9, 5.7,
                2.8, 4.1, 3.8, 1.6, 3.4, 8.4, 5.2, 6.9, 2.7, 1.9)
Y <- cbind(tear, gloss, opacity)
```

Creiamo successivamente due oggetti factor nel seguente modo:

```
rate <- factor(gl(2,10), labels=c("Low", "High"))
additive <- factor(gl(2, 5, len=20), labels=c("Low", "High"))
```

L'analisi della varianza multivariata viene eseguita nel seguente modo:

```
fit <- manova(Y ~ rate * additive)
summary.aov(fit)
```

Un altro modo per ottenere l'analisi della varianza multivariata consiste nel dare il seguente comando:
`>summary(fit, test=`"Wilks")
In cui il valore per la variabile test può essere uno dei seguenti:

- wilks
- Pillai
- Hotelling-Lawley
- Roy

36 Varianti in linux

36.1 Pacchetti automaticamente installati ma non avviati

Quando il programma è installato oltre alla sua configurazione base sono automaticamente installati una serie di pacchetti aggiuntivi che sono molto utili nelle analisi statistiche. Tali pacchetti non sono utilizzabili all'avvio del programma ma devono essere richiamati tramite la voce del menu con la seguente sequenza di comandi:
`>library(`"nomepacchetto"`)`
con nome pacchetto reperibile con il comando
`library()`
Circa l'avvio automatico dei pacchetti, se si utilizza mandrake 9.0 un buon contenuto del file
`.First`
può essere il seguente:

```
function() {
require("ctest", quietly = TRUE)
           require("grid", quietly = TRUE)
           require("lattice", quietly = TRUE)
           require("stepfun", quietly = TRUE)
           require("nls", quietly = TRUE)
           require("nlme", quietly = TRUE)
           require("mvtnorm", quietly = TRUE)
           require("multcomp", quietly = TRUE)
           require("scatterplot3d", quietly = TRUE)
           require("SuppDists", quietly = TRUE)
require("mva", quietly = TRUE)
           options(editor="kwrite",browser="mozilla")
                }
```

Si ricordi in ogni caso che i pacchetti `mvtnorm, multcomp,scatterplot3d e SuppDists` sono da installare preventivamente.

36.2 Pacchetti reperibili in rete

Esistono anche dei pacchetti reperibili in rete, molto interessanti e che presentano funzioni molto importanti per compiere sofisticate analisi statistiche di dati. Per caricare tali pacchetti in R dobbiamo utilizze da shell il seguente comando:
`R CMD INSTALL` "percorso/pacchetto"

36.3 Pacchetti in codice sorgente

Un metodo molto pratico per aggiungere pacchetti appositamente creati è quello che consiste nella creazione di appositi file detti source nei quali vengono inserite più funzioni scritte in linguaggio R come vedremo successivamente. Per caricare tali funzioni basterà dare il seguente comando:
`source(`"percorso/pacchetto"`)`

36.4 Ottenere un grafico in eps

Per ottenere un grafico in formato `eps` possiamo usare la seguente sequenza di comandi:
`>postscript`(“percorso/“nomefile.eps“)
`>hist(x)` ad esempio
`>dev.off()`

36.5 Importare dati in R

Per importare dati in R è consigliabile da open office o da star office salvare i dati in formato `csv` usando come separatore di campo `tab`. In R i dati sono importati con la sintassi:

```
>read.table("percorso/nomefile",header=T,sep="\t",dec=",")"
```

Indice