

Matematica e Statistica con R

Federico Comoglio e Maurizio Rinaldi

27 ottobre 2014

Indice

Prefazione	1
1 Introduzione ad R	3
1.1 Introduzione	3
2 Matematica con R	7
2.1 Numeri, operazioni aritmetiche e variabili	7
2.1.1 I numeri	7
2.1.2 Variabili	7
2.1.3 Costanti fondamentali	9
2.1.4 Costruzione di vettori	10
2.1.5 Generazione di sequenze: <code>rep</code> e <code>seq</code>	11
2.2 Funzioni e grafici	14
2.2.1 Definizione di funzioni	14
2.2.2 Funzioni predefinite	15
2.2.3 Grafici	17
2.3 Derivate di funzioni	28
2.3.1 Regola dei tre punti	32
2.3.2 Derivate “formali”	37
2.3.3 Metodo delle tangenti di Newton	40
2.4 Integrali definiti	42
2.4.1 Metodi numerici di integrazione di R	42
2.4.2 Metodi di integrazione: metodo dei rettangoli e metodo dei trapezi	43
2.4.3 Metodo Montecarlo	45
2.5 Numeri complessi	48
2.6 Variabili aleatorie discrete	50
2.7 Statistica descrittiva: singola variabile	53
2.7.1 Indicatori statistici	53

2.7.2	Raggruppamenti in classi	55
2.7.3	Areogrammi	58
2.7.4	Generazione di boxplot	60
2.7.5	Creazione di grafici a torta	63
2.8	Variabili doppie e rette di regressione	64
2.9	Distribuzioni in R	71
2.9.1	Distribuzione normale	71
2.9.2	La distribuzione t di Student	75
2.9.3	Intervalli di confidenza e test di Student (dati non appaiati)	75
2.9.4	Test di Student per dati appaiati	79
2.10	Test χ^2 di indipendenza	80
2.10.1	Test χ^2 di adeguamento	82
2.11	Distribuzione Binomiale	83
3	Strutture di dati	85
3.1	Visualizzazione di oggetti di R	85
3.2	Le stringhe	86
3.2.1	Stringhe <code>print</code> , <code>cat</code> e caratteri speciali	86
3.2.2	Operare con le stringhe	87
3.3	Matrici	90
3.3.1	Operazioni con le matrici	92
3.4	I <i>dataframe</i>	94
3.5	Gli <i>array</i>	100
3.6	Liste	101
3.6.1	Funzioni applicate a oggetti	102
4	Operatori logici e selezione di elementi	109
4.0.2	I valori non assegnati	110
4.1	Operatori condizionali e cicli	110
4.2	Funzioni di più variabili	111
4.3	R Output su file	112
5	Statistica II parte	113
5.1	Analisi della Varianza (ANOVA)	114
5.2	Test χ^2 di indipendenza	116
5.2.1	Test χ^2 di adeguamento	117
5.3	Distribuzione Binomiale	117
5.4	Distribuzione di Fisher	118
5.5	Tabelle di contingenza	118

5.6	Farmacocinetica e modelli non lineari	120
5.6.1	Regressione non lineare	121
5.6.2	I bambini di Kalama (Egitto). Ancora retta di regressione	124
5.7	QQPlot normality test	127
6	Grafica	129
6.1	Le <i>device</i> grafiche	129
6.2	Lo stato grafico	130
6.3	Elementi grafici di base	131
6.3.1	Costruzione di grafici multipli	131
6.3.2	Diagrammi a dispersione	133
6.3.3	Grafici multipli e/o sovrapposti	138
6.3.4	Aggiungere testo al grafico	140
6.3.5	Immagini di un vulcano	141
6.3.6	I rettangoli	143
6.4	Grafici professionali: legenda, annotazioni e formule	144
6.5	Pittura virtuale	148
6.6	Grafica 3D, il vulcano Maunga Whau	150
6.7	Web	153
6.7.1	Colori	154
6.7.2	Costruire una funzione per realizzare grafica su <i>device</i>	154
6.8	Grafica 3D, grafico di una funzione di due variabili	154
6.8.1	Il pacchetto rgl	154
6.8.2	Personalizzare la regione di plot	158
6.8.3	Il pacchetto lattice	160
6.8.4	Il parallel plot	163
6.8.5	Applicazione in campo geografico	165
6.8.6	Il parametro las	169
6.9	I rettangoli	169

Prefazione

Queste dispense devono la loro esistenza al corso di R che ormai da alcuni anni accompagna il corso di Matematica e Statistica. Esse seguono i principali temi del corso e fanno uso consistente di esempi. La maggior parte degli esempi sono stati pensati da noi, ma non rivendichiamo che essi siano esclusivi o abbiano una rilevanza particolare.

Inoltre, numerosi punti sono ancora carenti di adeguate referenze, che verranno aggiunte in maniera esaurente nelle prossime versioni.

Queste dispense sono state realizzate in L^AT_EX, con praticamente tutte le figure generate *on the fly* in R e incluse nel testo utilizzando **Sweave**.

Nonostante abbiamo speso tempo nel pensare ai concetti da includere ed organizzarne il contenuto, ci assumiamo le nostre responsabilità per errori che sono ancora presenti nel testo, figure o codice. A questo proposito ma non solo, ogni vostro suggerimento è ben accetto e potenzialmente molto prezioso.

Grazie,
Novara, 27 ottobre 2014
Maurizio e Federico

Capitolo 1

Introduzione ad R

R è un programma *freeware* (di distribuzione gratuita) e *open-source* (il cui codice sorgente può essere modificato e redistribuito) distribuito dalla “R foundation”, utilizzato per il calcolo statistico ed in grado di svolgere calcoli matematici ed algebrici (sicuramente di livello superiore ad una normale calcolatrice tascabile) ma che non rientrano nello scopo primario di R . La versione cui sono riferite queste dispense è la 2.15.0 Assumiamo che lo studioso abbia scaricato dal sito <http://www.r-project.org/> tale versione di R e la abbia installata sul suo computer.

1.1 Introduzione

Il programma si presenta all’utente con una interfaccia per l’inserimento dei dati, una finestra chiamata “R console”. Per esempio digitando a console (dopo il *prompt >*) il comando

```
> 1+1
```

e premendo il tasto di invio appare come risposta

```
[1] 2
```

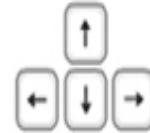
Oltre alla console R dispone di finestre di altra natura, per esempio finestre grafiche e finestre di *help*. Per affiancare verticalmente due o più finestre in ambiente Windows si seleziona l’opzione *finestre → affianca* dal menu a tendina oppure per installazione in lingua inglese: *windows → tile*.

L’insieme dei comandi eseguiti costituisce la storia o *cronologia* (*history*) di una sessione di lavoro. È possibile muoversi all’interno della cronologia, cioè richiamare i comandi precedenti e successivi utilizzando le frecce \uparrow e \downarrow

della tastiera.

Possiamo scrivere diversi comandi prima di eseguirli (tasto di invio): basta separarli con un “;”.

Si ricordi sempre che R è case sensitive ossia vi è differenza tra lettere minuscole e maiuscole. Questo è particolarmente importante in quanto ad esempio le variabili a ed A rappresentano entità distinte.



Help e guida on-line

Una qualunque richiesta di informazioni può essere effettuata digitando il comando: `?argomento` (o `help(argomento)`). Ad esempio `?sin` consente di ottenere informazioni sulla funzione seno e altre funzioni trigonometriche in quanto la guida è completamente indicizzata. È anche possibile utilizzare il menu a tendina (`help`), con modalità diverse a seconda del sistema operativo. Infine, il comando `help.start` consente di aprire l'`help` HTML di R da cui è poi possibile consultare la documentazione.

Inserire i commenti

I commenti sono frasi, annotazioni o codice da non eseguire momentaneamente molto utilizzati in programmazione. L'introduzione di un commento in R è possibile mediante il simbolo `#` (cancelletto singolo). Per evitare che un commento venga cancellato durante il normale *editing* del testo e che esso venga eseguito si consiglia di raddoppiare i simboli (`##`). Ad esempio:

```
> 1+1 # e' uguale a 2?  
[1] 2
```

non produrrà errore perché riconoscerà la stringa scritta dopo `#` come commento.

Quando un commento è utile e quando è superfluo?

A prescindere dalle preferenze personali, un commento dovrebbe essere inserito quando la porzione di codice che si sta considerando risulta a noi non ovvia, quando le dimensioni, tipo e ruolo delle variabili coinvolte non sono ovvie o semplicemente quando si ha bisogno di introdurre una annotazione che renda più facile la lettura del codice non solo a noi stessi ma anche ad altri. Quest'ultimo punto è di fondamentale importanza quando si vuole

condividere il codice con altri membri della comunità scientifica. Per capire se i commenti che inserite sono superflui o mancanti:

Ad un certo punto del corso salvate una porzione di codice (o una funzione) commentata in un file .R, lunga almeno 25 righe di codice. Aggiungete la data corrente al nome del file e in un periodo compreso tra 6 e 12 mesi semplicemente aprite il file e rileggetene il codice. Se siete in grado di capirne il contenuto, avete probabilmente fatto buon uso dei commenti.

Editori esterni

Talora è utile lavorare con editori esterni. Ne menzioniamo quattro di semplice utilizzo: Rstudio, Tinn-R, Komodo Edit e TextWrangler, i cui dettagli sono presenti in tabella 1.1.

A parte Tinn-R e RStudio, gli altri due editori necessitano di estensioni (*plug-*

Editor	Sistema Operativo	Website
RStudio	Mutlipiattaforma	http://www.Rstudio.org
Tinn-R	Windows	http://www.sciviews.org/Tinn-R
Komodo Edit	Mutlipiattaforma	http://www.activestate.com/komodo-edit
TextWrangler	Mac OSX	http://www.barebones.com/products/textwrangler/

Tabella 1.1: Alcune informazioni sugli editor citati nel testo

in) dedicate per riconoscere la sintassi di R, segnalare errori, fornire suggerimenti ed eseguire comandi in R direttamente dall'editor. Il *plug-in* per Komodo Edit si chiama SciViews-K (<http://www.sciviews.org>), mentre il *plug-in* per TextWrangler è scaricabile dal sito <http://www.smalltime.com/gene/R.plist>. Quest'ultimo deve essere poi inserito nella directory (librerie globali): `~/Library/Application Support/TextWrangler/Language Modules/`.

A livello di sviluppo sono invece disponibili editori dedicati, che forniscono numerose funzioni di diagnostica e *debug* del codice. Senza qui addentrarci nei dettagli, Emacs (<http://www.gnu.org/software/emacs/>) con il *plug-in* ESS (Emacs Speak Statistics, <http://ess.r-project.org/>) ha trovato negli ultimi anni largo impiego anche come editor per R.

Pacchetti e *directory* di lavoro

Oltre all'aggiornamento costante del programma, lo sviluppo di R consiste nello creazione e aggiornamento continua da parte della comunità mondiale di una serie di pacchetti addizionali. Un pacchetto contiene in particolare un insieme di comandi finalizzati alla realizzazione di un determinato compito: esistono per esempio pacchetti per la risoluzione di equazioni differenziali, pacchetti per l'analisi statistica di *microarray*, pacchetti per il *clustering* di dati. Per non appesantire R i pacchetti vengono caricati solo se richiesto dalle esigenze dell'utente. L'utente deve quindi inizialmente localizzare (per esempio via web) il pacchetto che lo interessa e scaricarlo localmente. Per fare questo deve selezionare il sito CRAN da cui scaricarlo e poi (direttamente dal menu a tendina di R) scaricarlo (con annesse eventuali *dependencies*). Una volta scaricato in locale il pacchetto può essere messo a disposizione dell'utente con il comando

```
> library("nome.pacchetto")
```

Il comando

```
> getwd()
```

indica la directory di lavoro mentre il comando

```
> setwd("indirizzo")
```

imposta la *working directory* nella posizione specificata da "indirizzo".

Capitolo 2

Matematica con R

2.1 Numeri, operazioni aritmetiche e variabili

2.1.1 I numeri

Si rappresentano usando il punto come separatore decimale, per numeri molto grandi o molto piccoli R ricorre alla notazione scientifica. Un numero “grande” viene scritto come $a \times 10^b$ (senza spazi) mentre un numero “piccolo” viene scritto come $a \times 10^{-b}$ (senza spazi) dove a varia tra 1 e 9 e b è l’esponente opportuno. Ad esempio:

```
> 0.0001  
[1] 1e-04  
> 5.4e3  
[1] 5400
```

Si abbia cura di utilizzare il punto ‘.’ invece della virgola ‘,’ come separatore decimale. Per quanto riguarda le operazioni aritmetiche fondamentali si usano i simboli: ‘+’, ‘-’, ‘*’, ‘/’. Per l’elevamento a potenza il simbolo è ‘^’.

2.1.2 Variabili

Una variabile è una porzione di memoria caratterizzata da un nome che contiene dati che possono essere modificati durante l’esecuzione del programma. È possibile definire una qualunque variabile mediante l’assegnazione di un valore ad essa.

Sono valide entrambe le notazioni seguenti: $\text{variabile} \leftarrow \text{valore}$ e $\text{valore} \rightarrow \text{variabile}$; ad esempio:

```
> a<-2
```

associa il valore 2 alla variabile `a`. Per comporre le frecce si usano i segni “meno” “maggiore” \rightarrow (freccia a destra) o “minore” “meno” \leftarrow (freccia a sinistra). Si abbia cura di evitare di inserire spazi tra il nome della variabile e la freccia, così come tra la freccia ed il valore assegnato. Il programma per assegnazioni semplici non invia alcun messaggio di errore, ma si possono verificare problemi quando vi siano più membri costituenti il valore assegnato. Rivolgere **sempre** la freccia dal valore alla variabile, infatti essa è una scatola in grado di contenere il valore voluto. Ad esempio non è valida la stringa: *variabile* \rightarrow *valore* o *valore* \leftarrow *variabile*. **Non** lasciare spazi vuoti nel definire il nome della variabile, è possibile invece utilizzare il sottotratto (*underscore*) _ ad esempio definendo: `variabile_uno`, **non** `variabile uno`. Si può anche usare il punto e scrivere `variabile.uno`, L’assegnamento di uno stesso valore a più variabili (o costanti) in un unico comando è possibile con una sintassi del tipo:

```
> a<-b<-c<-2
```

L’assegnazione può avvenire anche usando il segno ‘=’, ad esempio

```
> a=2
```

o il comando

```
> assign("a",2)
```

Ogni variabile è richiamabile scrivendo il suo nome. Ad esempio: `a` richiama la variabile `a` appena definita e mostra il valore di `a` in output, ovvero 2¹, Per rimuovere una variabile o un generico oggetto si utilizza il comando:

`rm(name),`

ossia *remove memory*, `name` è il nome della variabile da cancellare. Ad esempio:

```
> rm(a)
> a=2;a
[1] 2
```

¹Spesso dopo aver scritto un comando ad esempio `a<-2` ci si interroga sull’assenza di output. In realtà il comando è stato eseguito. Per verificarlo basta scrivere `a` ed inviare. In modo alternativo il comando (`a<-2`) oltre a definire l’associazione di 2 ad `a` ne stampa il valore.

svuota il contenuto della memoria **a**. Per conoscere gli oggetti presenti nella sessione corrente basta il comando

```
> ls()  
[1] "a" "b" "c"
```

o in modo equivalente

```
> objects()  
[1] "a" "b" "c"
```

2.1.3 Costanti fondamentali

Per costante ci si riferisce ad un valore non modificabile. Ogni linguaggio permette di utilizzare diversi tipi di costanti: numeri, caratteri e stringhe. Tra le costanti definite in R di interesse generale abbiamo

- π definito come **pi**.
- **e**: numero di Nepero definito come il valore in $x = 1$ della funzione esponenziale **exp(1)**. La notazione è equivalente ad e^1 . Non è possibile definire **e** solo con la stringa **exp** senza assegnare l'esponente, in quanto esso è definito come una funzione di x . Ovviamente le costanti sono utilizzabili in qualsiasi calcolo:

```
> duepi<-2*pi  
> twopi  
[1] 6.283185
```

Abbiamo assegnato alla variabile **twopi** valore 2π e l'abbiamo richiamata per visualizzare il risultato.

Nota bene: se si intende visualizzare il valore e non ad utilizzarlo in calcoli successivi è più comodo ricorrere alla visualizzazione a *console*, come nell'esempio:

```
> 2*pi  
[1] 6.283185
```

in output si ottiene direttamente il risultato del calcolo.

2.1.4 Costruzione di vettori

Un vettore (o *array* monodimensionale) può essere visto come un casellario, le cui caselle sono dette celle del vettore stesso. Tutte le celle sono variabili di uno stesso tipo, detto tipo base del vettore. Si possono così definire vettori di numeri e di caratteri. In R l'assegnamento della dimensione di un vettore non è obbligatorio. Un vettore può essere definito utilizzando il comando **c** di sintassi generale:

$$c(valore_1, valore_2, \dots, valore_n)$$

Ad esempio

```
> c(1,5,9,11,17,34)
[1] 1 5 9 11 17 34
```

Si può associare il vettore ad una variabile con la notazione:

$$c(valore_1, valore_2, valore_n) \rightarrow variabile$$

Per esempio:

```
> vettore <- c(1,5,9,11,17,34)
```

Due operazioni fondamentali sono l'estrazione di una cella

```
> i=4; vettore[i]
[1] 11
```

e la sostituzione

```
> vettore[i] <- 5
```

Si noti anche che se scriviamo

```
> vettore[i] <- "testo"
> vettore
[1] "1"      "5"      "9"      "testo"   "17"      "34"
```

abbiamo anche cambiato il tipo di base del vettore mentre, se scriviamo

```
> vettore[i] <- 11
> vettore
[1] "1"   "5"   "9"   "11"  "17"  "34"
```

il vettore non viene restaurato. Per farlo occorre in questo caso scrivere

```
> as.numeric(vettore) -> vettore
```

Possiamo anche definire celle con indice superiore alla lunghezza

```
> vettore[10] <- 3
> vettore
[1] 1 5 9 11 17 34 NA NA NA 3
```

il vettore viene prolungato fino a comprendere 10 celle. Le celle non assegnate vengono etichettate con `NA`. Se invece si scrive

```
> vettore[-3]
[1] 1 5 11 17 34 NA NA NA 3
```

la cella di posizione 3 viene eliminata. Possiamo determinare la lunghezza di un vettore con la funzione `length`:

```
> length(vettore)
[1] 10
```

Possiamo poi ovviamente eseguire operazioni sui vettori

```
> 3*vettore
[1] 3 15 27 33 51 102 NA NA NA 9
```

Il vettore di nome `vettore`, viene moltiplicato per 3, ovvero ciascuna cella viene moltiplicata per 3.

2.1.5 Generazione di sequenze: `rep` e `seq`

È possibile definire sequenze di numeri consecutivi come `n:m`. Ad esempio:

```
> 1:11
[1] 1 2 3 4 5 6 7 8 9 10 11
```

consente di ottenere la sequenza dei numeri interi da 1 a 11. È anche possibile aggiungere un numero alla sequenza con il comando: `c(n:m, numero aggiunto)`. Per esempio se vogliamo aggiungere il numero 100 alla lista precedente scriveremo:

```
> c(1:11, 100)
```

```
[1] 1 2 3 4 5 6 7 8 9 10 11 100
```

Nota: l'aggiunta è posizionale, cioè rispetta l'ordine con cui compaiono gli elementi.

È inoltre possibile generare liste costanti di valore k con il comando: **rep(k,n)** Se vogliamo una lista i cui elementi sono tutti 3 per un totale di 10 elementi scriveremo:

```
> rep(3,10)
[1] 3 3 3 3 3 3 3 3 3 3
```

Infine il parametro **each= n** consente di ripetere ciascun elemento della sequenza n volte. Ad esempio:

```
> rep(c(3,2,1),each=3)
[1] 3 3 3 2 2 2 1 1 1
```

Ripete 3 volte il numero 3, 3 volte il numero 2 e 3 volte il numero 1. Diverso è il risultato con il comando

```
> rep(c(3,2,1),times=5)
[1] 3 2 1 3 2 1 3 2 1 3 2 1 3 2 1
```

dove si ha la ripetizione **times** volte dell'intero vettore. Cosa succede se 2 vettori hanno lunghezza diversa? In matematica in questo caso la somma non è definita. In R invece vale la regola del riciclaggio (*recycling rule*: ad esempio

```
> 1:10+1:3
[1] 2 4 6 5 7 9 8 10 12 11
> 1:10+1:2
[1] 2 4 4 6 6 8 8 10 10 12
```

In pratica il vettore più corto viene riciclato tante volte quanto serve per raggiungere la lunghezza del vettore più lungo. Se la ripetizione non occorre un numero intero di volte allora viene emesso un messaggio di *Warning*.

Il comando **seq** consente di generare una sequenza (vettore) i cui elementi sono relativi da una proprietà caratteristica: sono tutti numeri pari, dispari, di passo n , ecc.

Si definisce una sequenza come: **seq(n,m)**

Ad esempio per generare la sequenza dei numeri da 0 a 100 a passo 1 (default) si scriverà:

```
> seq(0,100,1) ##analogo di seq(0,100)
[1]  0  1  2  3  4  5  6  7  8  9 10 11 12
[14] 13 14 15 16 17 18 19 20 21 22 23 24 25
[27] 26 27 28 29 30 31 32 33 34 35 36 37 38
[40] 39 40 41 42 43 44 45 46 47 48 49 50 51
[53] 52 53 54 55 56 57 58 59 60 61 62 63 64
[66] 65 66 67 68 69 70 71 72 73 74 75 76 77
[79] 78 79 80 81 82 83 84 85 86 87 88 89 90
[92] 91 92 93 94 95 96 97 98 99 100
```

Possiamo ottenere la sequenza di tutti i numeri dispari da n a m utilizzando il parametro `by`, che consente di impostare il passo nella sequenza. Il comando:

```
> seq(1,100,by=2)
[1]  1  3  5  7  9 11 13 15 17 19 21 23 25 27 29 31 33 35
[19] 37 39 41 43 45 47 49 51 53 55 57 59 61 63 65 67 69 71
[37] 73 75 77 79 81 83 85 87 89 91 93 95 97 99
```

restituisce tutti i numeri tra 1 e 100 a passo 2, ovvero tutti i numeri dispari compresi nell'intervallo. Per ottenere i numeri pari compresi nell'intervallo possiamo reiterare il comando precedente ma partendo da 0 (e volendo possiamo sottintendere la specifica `by=`)

```
> seq(0,100,2)
[1]  0  2  4  6  8 10 12 14 16 18 20 22 24 26
[15] 28 30 32 34 36 38 40 42 44 46 48 50 52 54
[29] 56 58 60 62 64 66 68 70 72 74 76 78 80 82
[43] 84 86 88 90 92 94 96 98 100
```

il parametro `length` consente di definire il numero totale di elementi della sequenza, ovvero divide l'intervallo da m a n utilizzando un numero di punti pari alla lunghezza assegnata. Ad esempio se nella sequenza di numeri da 1 a 100 volessimo avere in totale 10 numeri scriveremo:

```
> seq(1,100,length=10)
[1]  1 12 23 34 45 56 67 78 89 100
```

In effetti l'intervallelo da 1 a 100 di lunghezza 99 viene diviso da 10 punti in 9 parti di ampiezza 11. Useremo tale parametro per il calcolo degli integrali con il metodo dei rettangoli.

1. Costruire un vettore contenente i numeri da 1 a 15. Identificare almeno due modi per invertire l'ordine degli elementi. Scambiare poi il 5^o e il 6^o ingresso del vettore e stampare a video il risultato.
2. Calcolare $23 + 37/10 - 2^3$
3. Scrivere il vettore $x = (1, 1, 2, 3, 5, 8)$. Stampare il 3^o elemento di x . Determinare la lunghezza di x . Costruire un vettore i cui elementi siano la radice quadrata degli elementi di x .
4. Si consideri la funzione $f(x) = x * \sin(x)$. Si determinino i valori di f negli estremi della suddivisione dell'intervallo $[2, 5]$ in 20 sotto-intervalli di ugual ampiezza.

2.2 Funzioni e grafici

2.2.1 Definizione di funzioni

Per definire una funzione f di una variabile *variabile* si utilizza il comando:

```
f <- function(variabile) espressione
```

Ad esempio per definire la funzione

$$f(x) = x^2$$

scriviamo:

```
> f<-function(x) x^2
```

Possiamo calcolare il valore della funzione in un qualunque valore di x con il comando: $f(valore)$ ad esempio in $x = 6$ si avrà

```
> f(6)
[1] 36
```

Attenzione: nell'esempio che segue viene richiesto il valore di $f(x)$ in $x = 0$:

```
> f<-function(x) 1/x
> f(0)
[1] Inf
```

La dicitura **Inf** corrisponde ad ∞ , infatti la funzione esplode in $x = 0$.

2.2.2 Funzioni predefinite

R consente l'utilizzo di funzioni matematiche già definite, come seno, coseno, tangente, le funzioni trigonometriche inverse, esponenziali, logaritmi, radici, ecc. Ecco di seguito le principali, la cui sintassi può essere visualizzata utilizzando l'*help* di R inserendo la parola chiave corretta (ad esempio `?sin`) per ottenere informazioni sul formalismo corretto:

- Funzioni trigonometriche

<code>cos</code>	coseno
<code>sin</code>	seno
<code>tan</code>	tangente
<code>acos</code>	arcocoseno
<code>asin</code>	arcoseno
<code>atan</code>	arcotangente

- Logaritmi.

`log` è il logaritmo naturale. Ad esempio:

```
> log(3)
[1] 1.098612
```

`log2` è il logaritmo di base 2

```
> log2(3)
[1] 1.584963
```

Controlliamo:

```
> log2(3)->a
> 2^a
[1] 3
```

Sullo stesso principio il logaritmo in base 10 sarà `log10`. Ad esempio:

```
> log10(100)
```

```
[1] 2
```

Per basi diverse si usa il parametro **base** per specificare la base (si noti che il nome del parametro può anche essere omesso).

```
> log(64,base=4)
[1] 3
> log(64,4)
[1] 3
```

- Esponenziali.

La funzione esponenziale (come già visto) è denotata da **exp**, si abbia cura di indicare **exp** tutto minuscolo. Ad esempio per ottenere il valore numerico dell'espressione e^4 si scrive:

```
> exp(4)
[1] 54.59815
```

- Radice quadrata.

Si indica come **sqrt** (*square root*), ad esempio per ottenere la radice quadrata di 225 scriveremo:

```
> sqrt(225)
[1] 15
```

Ovviamente la radice quadrata equivale ad elevare a potenza 1/2 per cui la radice precedente può essere espressa anche come²:

```
> 225^(1/2)
[1] 15
```

Tale procedura può essere utilizzata per radici di indice qualunque, ad esempio:

```
> 16384^(1/7)
[1] 4
```

²Nel comando che segue non si dimentichino le parentesi.

2.2.3 Grafici

Il comando `curve`, consente di tracciare il grafico di una funzione; `curve(funzione)`, automaticamente assegna estremi sull'asse x da 0 ad 1 (a meno che non siano già stati definiti). Volendo possiamo specificare gli estremi con il comando:

```
curve(espressione/nome funzione,  
estremo inferiore , estremo superiore)
```

Ad esempio volendo tracciare il grafico della funzione $1/x$ nell'intervallo $(0, 2)$ possiamo definire la funzione oppure scriverne direttamente l'espressione. La differenza tra le due scelte dipende dall'uso futuro della funzione, se essa sarà applicata in seguito conviene assegnarle un nome. I comandi:

```
> curve(1/x,0,2)
```

oppure:

```
> f<-function(x) 1/x  
> curve(f,0,2)
```

generano lo stesso risultato, illustrato in Figura 2.1.

Può essere utile sovrapporre ad un grafico già tracciato un ulteriore grafico, tracciando ad esempio l'asse x , l'asse y , o una ulteriore funzione; la finestra grafica è quella preesistente e non deve essere ridefinita ogni volta (si suggerisce di realizzare per primo il grafico con finestra grafica più ampia). Il parametro da utilizzare in tale caso è `add` (variabile booleana, può assumere solo valore `True` o `False`, indicabili come `T` o `F` o per esteso come `TRUE` o `FALSE`, consigliato), ovviamente nel caso `F` l'assegnazione del parametro non è richiesta). Ad esempio per aggiungere la retta $y = 20x$ alla funzione precedente si scriverà:

```
> g<-function(x) 20*x  
> curve (g,add=T)
```

ottenendo la Figura 2.2. Vediamo qui di seguito alcune opzioni grafiche:

- Definizione della finestra grafica.

Possiamo specificare completamente la finestra grafica. Per l'asse delle x si ricorre al parametro `xlim`:

```
xlim=c(estremo inferiore, estremo superiore)
```

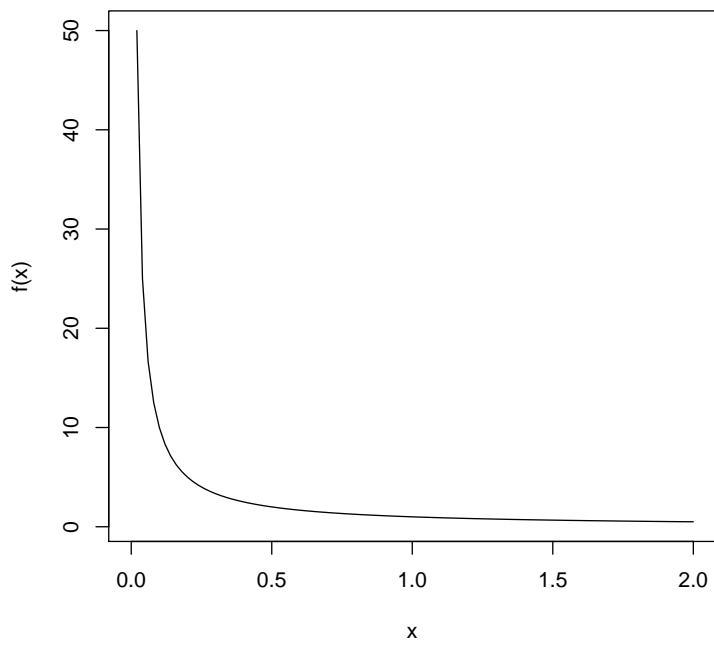


Figura 2.1: Grafico di $1/x$ nell'intervallo $(0, 2)$.

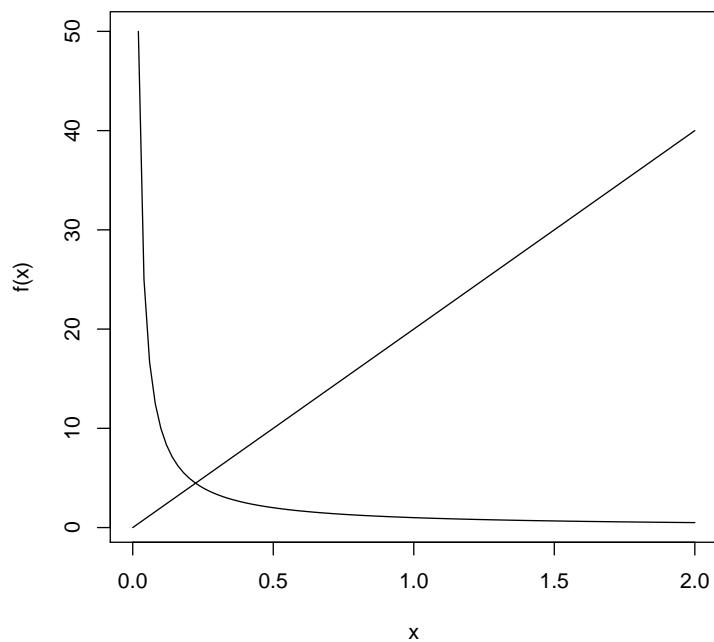


Figura 2.2: Sovrapposizione di 2 grafici

e in modo simile per l'asse delle y il parametro è `ylim`:

```
ylim=c(estremo inferiore, estremo superiore)
```

Se ad esempio volessimo tracciare il grafico della funzione precedente tra -3 e 3 sull'asse x e tra -4 e 4 sull'asse y , scriveremo:

```
> curve(1/x,xlim=c( -3,3), ylim=c(-4,4))
```

ed otterremo il grafico della figura 2.3.

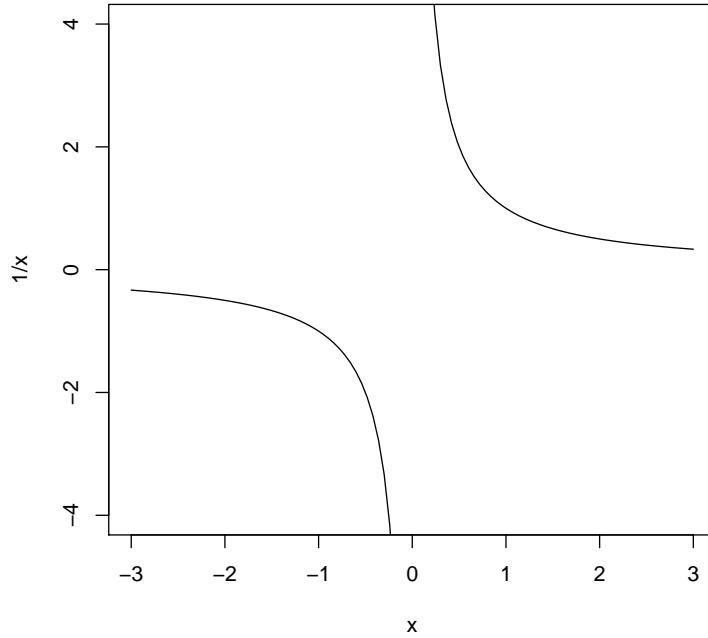


Figura 2.3: Scelta della finestra grafica

- Selezione del colore della curva.

Per cambiare il colore predefinito del grafico si deve utilizzare il parametro `col`, scrivendo `col("nome colore")`. Il nome del colore deve

essere definito in lingua inglese. Per visualizzare l'elenco completo dei colori disponibili si usa il comando: `colors()`. Tra i principali colori troviamo

rosso	red
blu	blue
bianco	white
rosa	pink
verde	green
giallo	yellow
viola	purple
nero	black

Sono utilizzabili anche i parametri `light` e `dark`, esse devono precedere il nome del colore. Possiamo pensare di colorare la curva precedente di blu scuro e scrivere: Otteniamo la figura 2.4. Si noti che la dicitura `dark blue` e `darkblue` senza spazi producono lo stesso risultato (sono implementati ambo i colori)

- Selezione dello spessore delle linee

Per cambiare lo spessore delle linee è possibile utilizzare il parametro `lwd`, esso può avere valori in una scala da 0 fino ad un numero qualunque (anche decimale) di *pixel*, se vogliamo visualizzare la curva precedente con spessore della linea pari a 5 pixel, scriveremo:

```
> curve(1/x, xlim=c( -3,3), ylim=c(-4,4),  
+ col="darkblue", lwd=5)
```

Otteniamo la figura 2.5

- Scelta del tipo di linea.

È possibile scegliere un numero di punti da visualizzare ed è anche possibile collegarli o meno. Per scegliere il numero di punti si usa il parametro `n` associandogli un numero intero. Per non collegare i punti si usa il parametro `type`, e si fa assumere valore "p", si scrive quindi: `type= "p"`. Ad esempio se vogliamo un numero di punti pari a 100, non collegati, scriveremo:

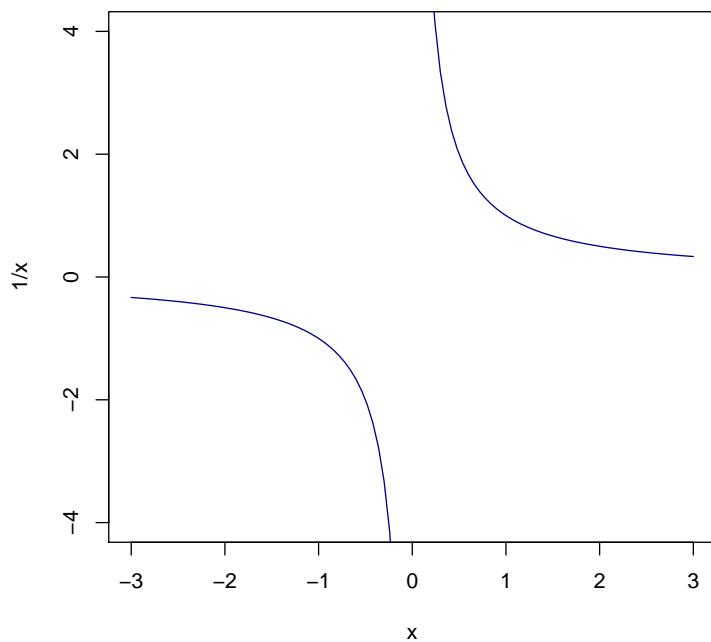


Figura 2.4: Scelta del colore

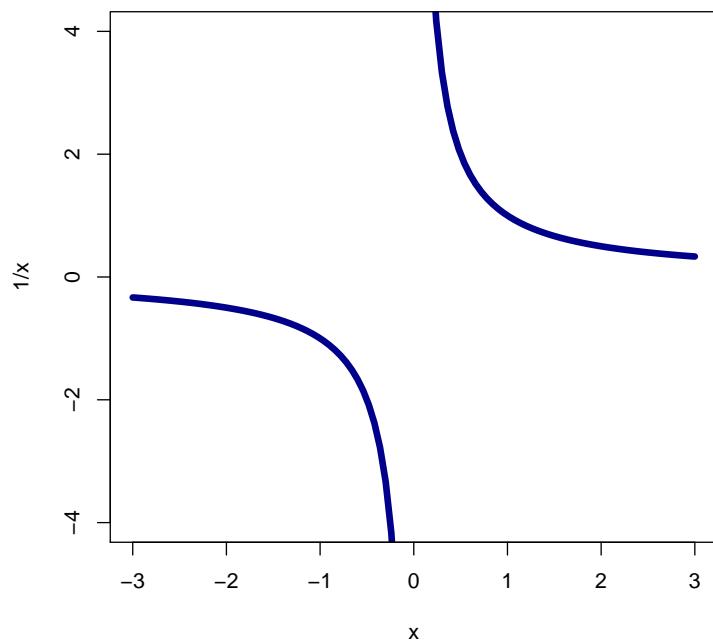


Figura 2.5: Spessore delle linee

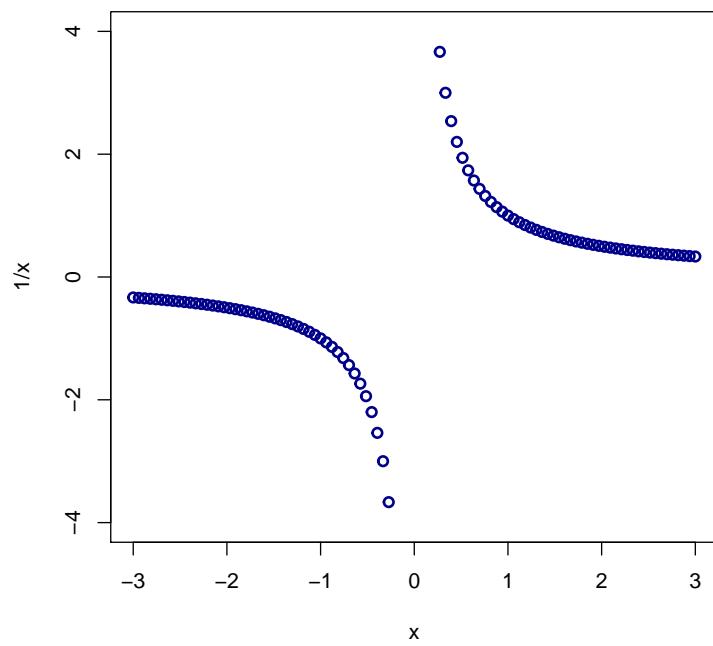


Figura 2.6: Grafico a punti

```
> curve(1/x,xlim=c(-3,3),ylim=c(-4,4),
+ col="dark blue", lwd=2,n=100,type="p")
```

il cui risultato è riportato in figura 2.6. Altre scelte possibili sono `type="n"` per il punto non vuoto ma anche `type="l"` per le linee (scelta di default) e `type="o"` per punti e linee.

- La funzione `abline`.

La funzione `abline`, di sintassi: `abline(b,a)` consente di tracciare una retta di pendenza `a` e di intercetta `b`. Se scriviamo `abline(h=valore)` otteniamo una linea orizzontale a quota `valore` mentre con: `abline(v=valore)` otteniamo una retta verticale ad ascissa pari al `valore`. Dopo avere tracciato un grafico si possono quindi aggiungere con i comandi: `abline(h=0)` e `abline(v=0)` cioè gli assi di riferimento del sistema cartesiano (utili nel caso ad esempio in cui si debbano individuare le intersezioni con gli assi della funzione). Ad esempio se scriviamo:

```
> curve(1/x,xlim=c( -3,3),
+ ylim=c(-4,4),col="darkblue",lwd=5);
> abline(h=0);abline(v=0);
```

Otteniamo la figura 2.7. Notiamo che cambiando di poco la finestra grafica

```
> curve(1/x,xlim=c( -3,3.1),
+ ylim=c(-4,4),col="darkblue",lwd=5);
> abline(h=0);abline(v=0);
```

Otteniamo la figura 2.8.

In questa figura una porzione del grafico è stata sovraimpressa all'asse verticale. In effetti uno è fittizio determinato dal fatto che la funzione non è definita in $x = 0$ e che R nel tracciare il grafico cerca di congiungere l'ultimo punto a sinistra dello zero ed il primo a destra. Per eliminare questo problema si può ricorrere ai comandi che seguono, che generano la figura 2.9:.

```
> curve(1/x,xlim=c( -3,3.1),
+ ylim=c(-4,4),type="n")
```

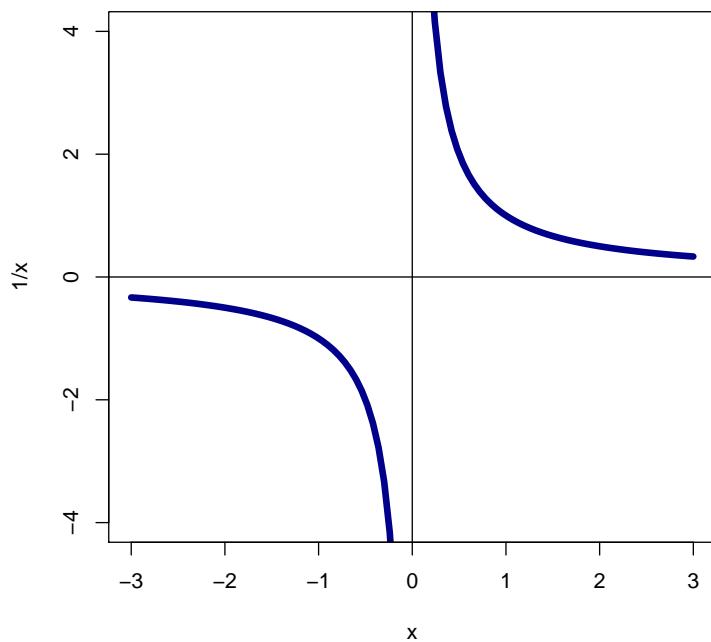


Figura 2.7: Iperbole con assi

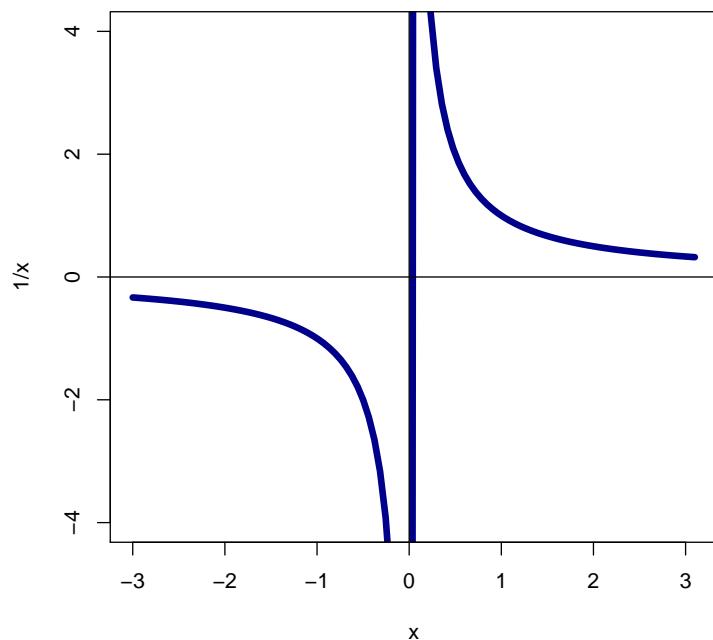


Figura 2.8: Problemi grafici

```
> curve(1/x,xlim=c(-3,0),
+ ylim=c(-4,4), col="darkblue",
+ lwd=5,add=T);
> curve(1/x,xlim=c(0,3.1),
+ ylim=c(-4,4),col="darkblue",
+ lwd=5,add=T); abline(h=0,
> col="red"); abline(v=0,col="red")
```

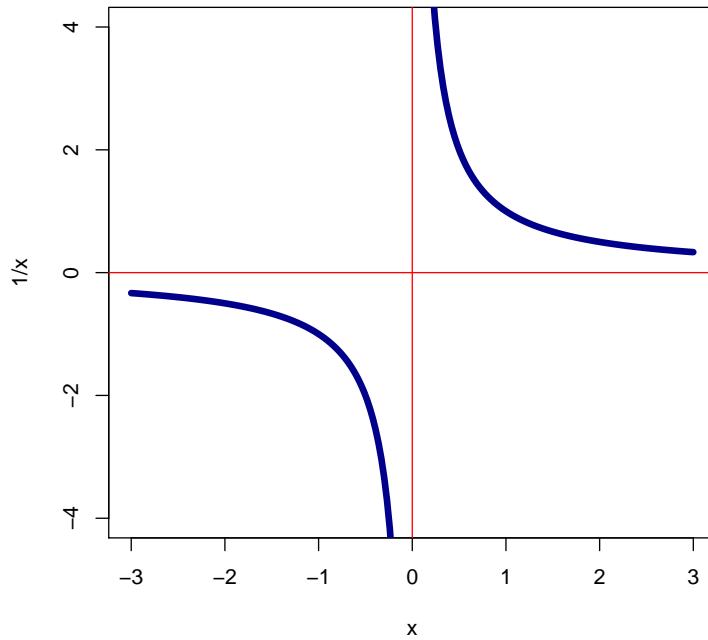
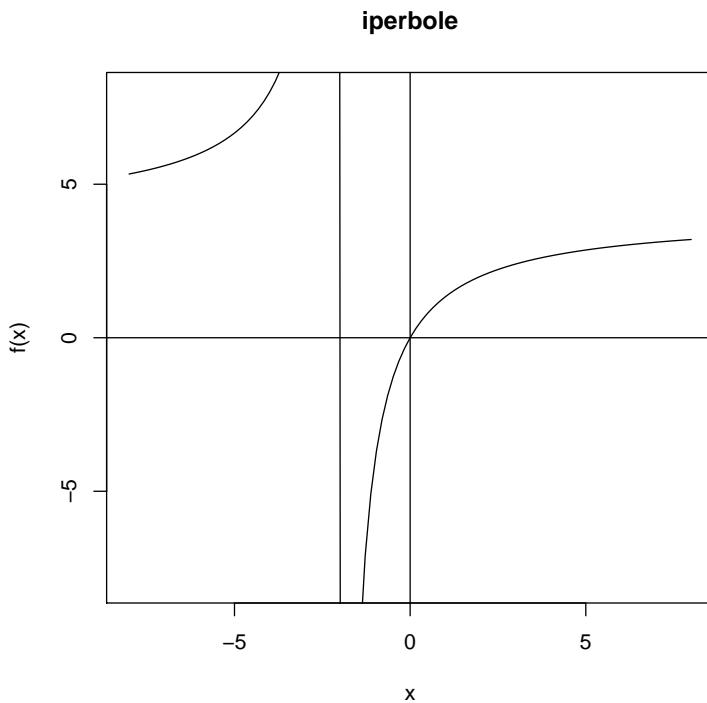


Figura 2.9: Grafico con assi e senza asintoto

2.3 Derivate di funzioni

Richiamiamo il significato geometrico della derivata: se una funzione f è derivabile in un punto x_0 allora il grafico si linearizza con un numero sufficiente di ingrandimenti attorno al punto $(x_0, f(x_0))$. Come illustrazione di questo processo consideriamo l'iperbole

```
> 8->d;curve(f,xlim=c(-d,d),ylim=c(-d,d),
> main="iperbole"); abline(h=0);abline(v=0)
```



```
> f<-function(x) 4*x/(2+x)
```

e disegniamo il suo grafico nella regione definita dalle condizioni $-d < x < d$ e $-d < y < d$, dove d è inizialmente pari a 8.

Consideriamo ora l'ingrandimento di tale grafico con fattore di zoom $xf=2$ attorno al punto $(0, 0)$: in altre parole dimezziamo il valore di d :

```
> d/2->d;curve(f,xlim=c(-d,d),ylim=c(-d,d),
> main="zoom 2"); abline(h=0);abline(v=0)
```

Iterando tale comando un certo numero di volte si assiste alla linearizzazione progressiva del grafico come in figura 2.10.

Consideriamo ora una funzione differente che combina un'espressione di quarto grado e una funzione circolare rapidamente oscillante

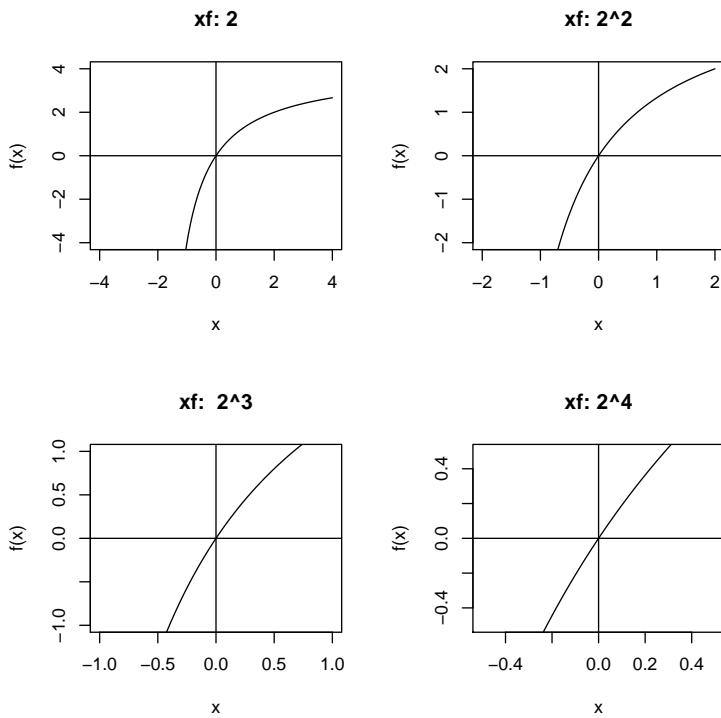


Figura 2.10: Linearizzazione di un grafico ottenuta attraverso una serie di ingrandimenti

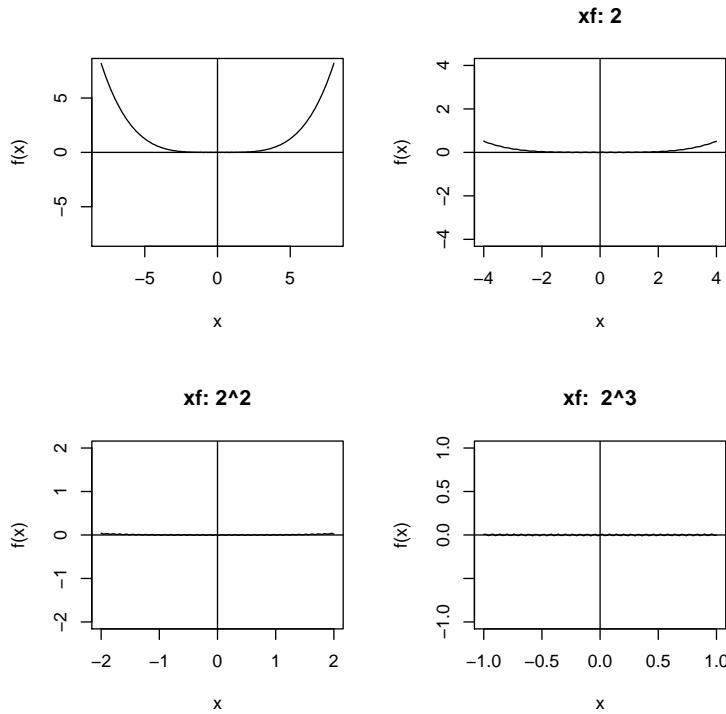


Figura 2.11: Linearizzazione progressiva di un grafico: prima fase

```
> f<-function(x) x^4/500 + sin(100*x)/100;
```

Il grafico per $-8 < x < 8$ come in precedenza suggerisce che la retta tangente in $x = 0$ sia orizzontale. Effettuando un certo numero di zoom con fattore di zoom 2 questa prima impressione sembra confermata, anche se possiamo iniziare a notare delle piccole oscillazioni nell'ultima figura del pannello 2.11.

Se procediamo con gli ingrandimenti questa sensazione viene confermata. Non ci saranno ulteriori sorprese da un certo punto in poi. Per quanto si prosegua la retta trovata in ultimo verrà confermata 2.12

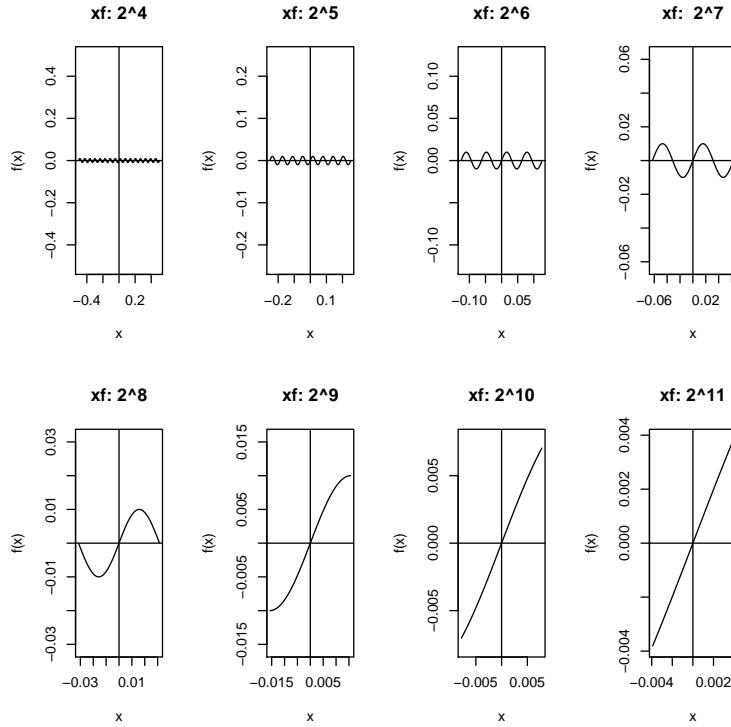


Figura 2.12: Linearizzazione progressiva di un grafico: seconda fase

2.3.1 Regola dei tre punti

Possiamo stimare la derivata di una funzione f in x con la regola dei tre punti nel punto x con incremento h

$$\text{trepunti}(f, x, h) = \frac{f(x + h) - f(x - h)}{2h}$$

Ora è sufficiente definire la funzione di cui vogliamo stimare la derivata, per poi richiamare la funzione `trepunti` fornendo il valore di x ed il valore dell'incremento h .

Approssimiamo ad esempio la derivata di $x \rightarrow \sin(x)$ Definiamo la funzione e la regola della differenza centrale

```
> f<-function(x) sin(x)
> trepunti<-function(f,x,h) (f(x+h)-f(x-h))/(2*h)
```

Per esempio assegnando

```
> h=0.001  
> x=1
```

Si ha

```
> trepunti(f,x,h)  
[1] 0.5403022
```

Otteniamo così la stima della derivata nel punto 1 con incremento 0.001. Possiamo anche rappresentare graficamente la funzione derivata (insieme alla funzione di **sin**) scrivendo:

```
> curve(trepunti(f,x,0.001),0,2*pi,col="blue")  
> curve(f,add=T,col="red")
```

e ottenendo il grafico 2.13 da cui si evince che $D\sin(x) = \cos(x)$. In modo simile si può mostrare che $D\cos(x) = -\sin(x)$. Analizziamo ancora la derivata delle funzioni esponenziali. Consideriamo inizialmente la funzione

$$f(x) = 2^x$$

In questo caso

```
> f<-function(x) 2^x  
> curve(trepunti(f,x,0.001),-2,2,  
+ col="blue")  
> curve(f,add=T,col="red")  
> curve(trepunti(f,x,0.001)/f(x),add=T,  
+ col="dark green")
```

sembra che la funzione (in rosso) e la sua derivata (in blu, approssimata con la regola dei 3 punti) siano proporzionali. Per verificarlo abbiamo tracciato anche il grafico di $f'(x)/f(x)$ che è una retta orizzontale a quota

```
> trepunti(f,0,0.001)/f(0)  
[1] 0.6931472
```

In modo simile possiamo considerare con la funzione 3^x :

```
> f<-function(x) 3^x  
> curve(trepunti(f,x,0.001),-2,2,col="blue")  
> curve(f,add=T,col="red")  
> curve(trepunti(f,x,0.001)/f(x),add=T,col="dark green")
```

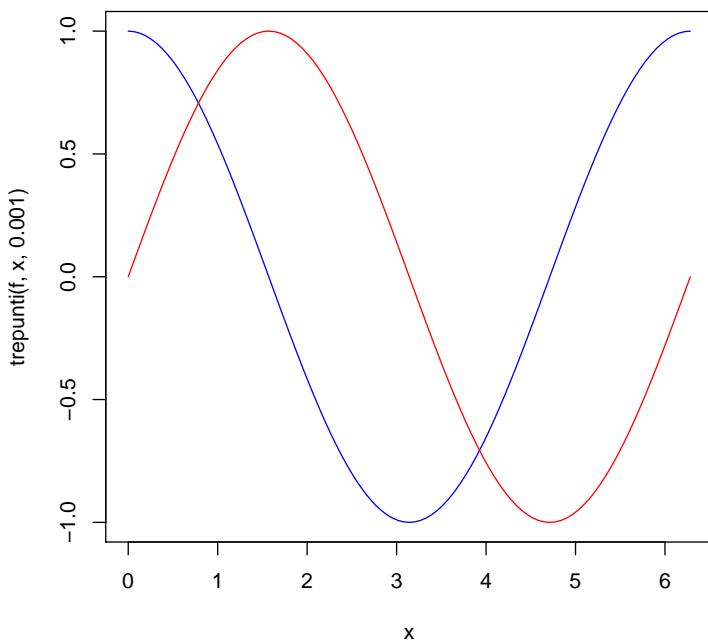


Figura 2.13: Grafico delle funzioni $\sin(x)$ e della sua derivata calcolata numericamente

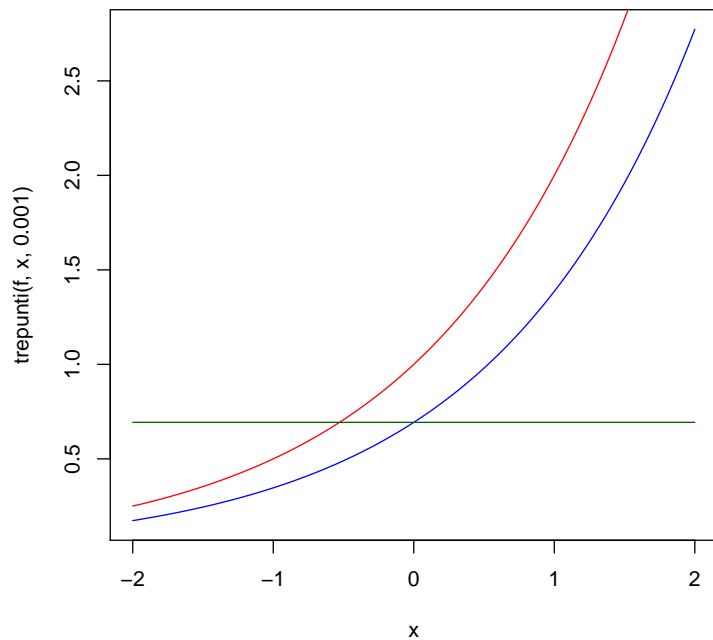


Figura 2.14: Grafico di 2^x e derivata

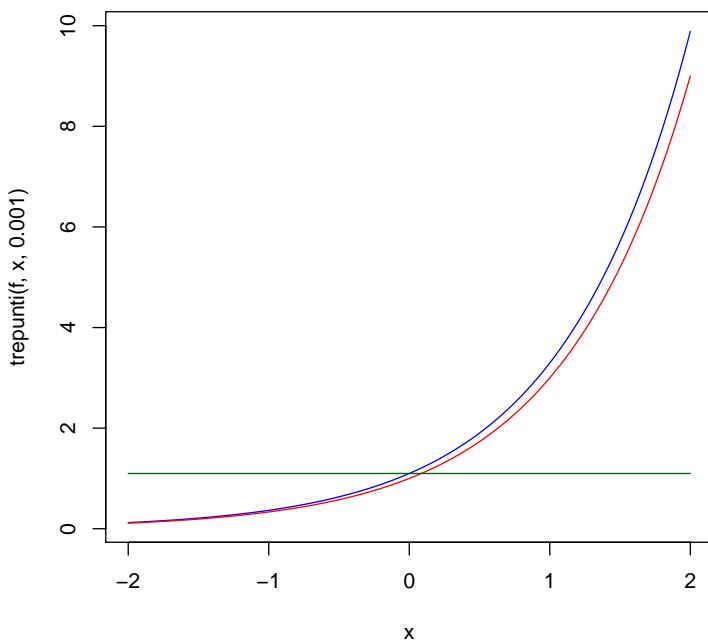


Figura 2.15: Grafico di 3^x e derivata

e anche qui le funzioni sembrano proporzionali. In questo caso però il rapporto derivata diviso funzione è maggiore di 1 e quindi la derivata è maggiore della funzione

Per verificarlo abbiamo tracciato anche il grafico di $f'(x)/f(x)$ che è una retta orizzontale a quota

```
> trepunti(f,0,0.001)/f(0)
[1] 1.098613
```

Ci si può chiedere se esista un valore intermedio tra 2 e 3 per cui derivata e funzione coincidano esattamente. La risposta è precisamente il numero di Nepero $\exp(1)$. Infatti abbiamo

```
> f<-function(x) exp(x)
> curve(trepunti(f,x,0.001),-2,2, col="blue")
> curve(f,add=T,col="red")
> curve(trepunti(f,x,0.001)/f(x),add=T, col="dark green")
> trepunti(f,0,0.001)/f(0)
```

Anche qui

```
> trepunti(f,0,0.001)/f(0)
[1] 1
```

mostra il valore del rapporto.

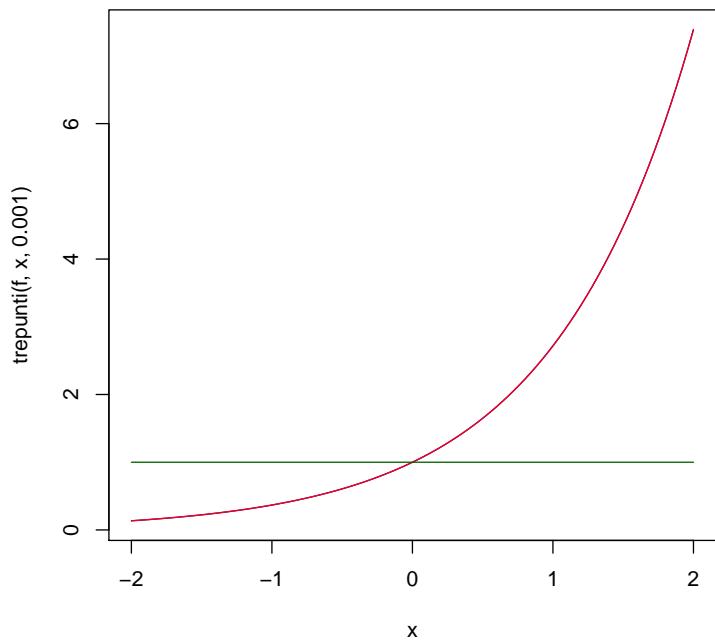
1. Ripeti le considerazioni precedenti usando il quoziente di Newton a destra.
2. Calcola il valore della approssimazione della derivata con la regola dei tre punti per la funzione $x^2 * \sin(x)$ in $x = \pi$ con $h = \pi/6$.

2.3.2 Derivate “formali”

Per calcolare la derivata di una qualunque espressione possiamo scrivere:

```
D(expression(funzione),variabile)
```

Ad esempio

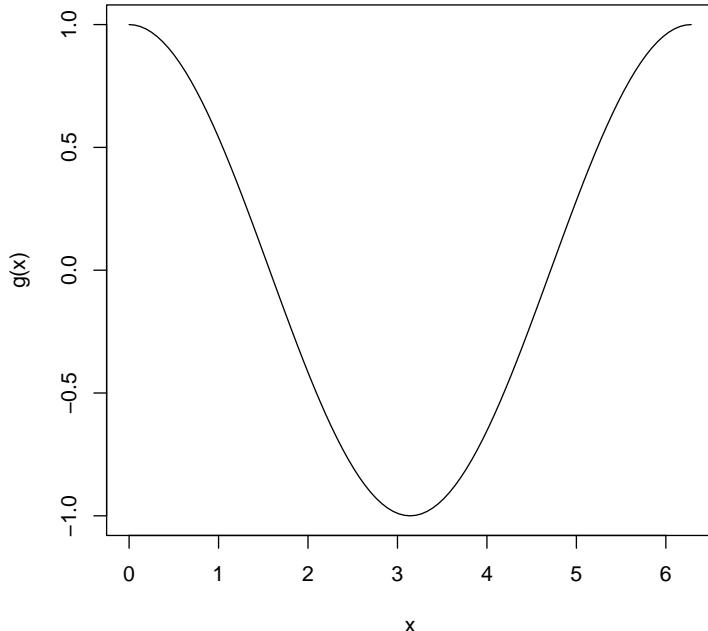


```
> D(expression(x^4), "x")
4 * x^3
```

Un utile comando per valutare una derivata in un punto o per ricavare una funzione da una derivata è il comando `eval`, ovvero *evaluate*. Scriveremo `eval(nome derivata)`.

```
> x<-4
> eval(D(expression(x^4), "x"))
[1] 256
```

```
> g<-function(x) eval(D(expression(sin(x)), "x"))
> curve(g, 0, 2*pi)
```



1. Calcola la derivata prima e seconda della funzione $x^2 * \sin(x)$ in $x = \pi$.

2. Determina l'equazione della retta tangente al grafico della funzione precedente e traccia grafico e retta in una finestra grafica opportuna.

2.3.3 Metodo delle tangenti di Newton

Utilizziamo ora il metodo delle tangenti di Newton per determinare gli zeri di una funzione f . Prima di iniziare vediamo come possiamo procedere a livello grafico. Consideriamo una semplice funzione

$$f(x) = x^5 - 3x + 1$$

Iniziamo a selezionare una finestra grafica abbastanza ampia. Restringendola man mano vediamo meglio cosa succede.

```
> f<-function(x) x^5-3*x+1
> curve(f,-10,10)
> abline(h=0)
> curve(f,-2,2)
> abline(h=0)
```

Dal grafico 2.19 è evidente che ci sono 3 intersezioni con l'asse delle x che ci proponiamo di determinare usando il metodo delle tangenti di Newton. Ricordiamo che nel metodo delle tangenti di Newton si deve scegliere un punto iniziale x_0 ed il numero di iterazioni (o un criterio di arresto). Il passo iterativo è descritto dalla funzione:

$$g(x) = x - \frac{f(x)}{f'(x)}$$

In assenza della derivata esatta possiamo anche usare la regola dei 3 punti. In quanto segue ipotizzeremo di avere scelto un 10 iterazioni a partire dal valore $x_0 = 2$.

```
> x<-2
```

Sceglio un incremento $h = 10^{-3}$ ed iteriamo il processo attraverso un ciclo **for**³:

³Il comando **for(k in set)** comando esegue il comando (eventualmente dipendente da **k**) per tutti i valori di **k** in **set**.

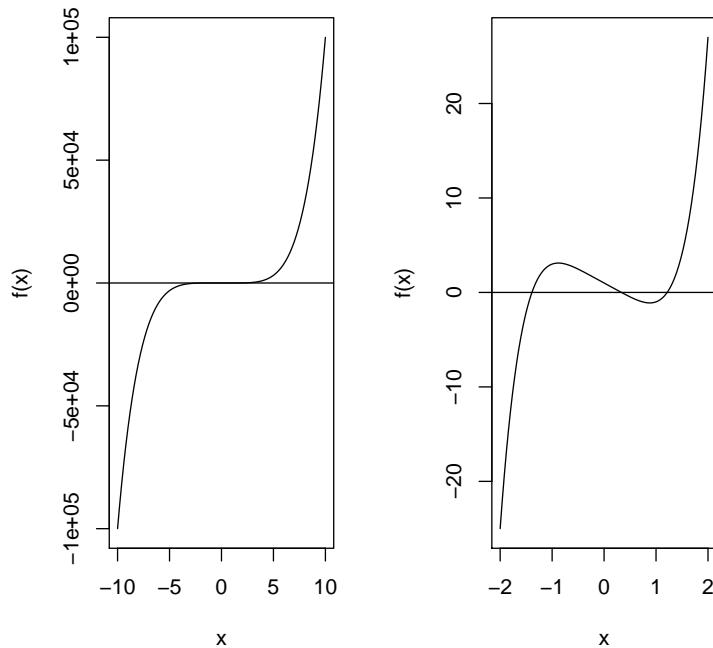


Figura 2.16: Intersezione con l'asse x

```
> h<-0.001
> for(k in 1:9) x[k+1]<-x[k]-f(x[k])/trepunti(f,x[k],h)
> x
[1] 2.000000 1.649351 1.406489 1.268587 1.220370
[6] 1.214721 1.214648 1.214648 1.214648 1.214648
```

Si genera la lista di 10 passi iterativi, il fatto che la lista si stabilizzi è indice del fatto che la lista converge all'intersezione cercata. Ricerchiamo anche un altro valore della *radice*:

```
> x<- -2; iterazioni=10;
> for(k in seq(length=iterazioni)) x[k+1]<-x[k]-f(x[k])/
+ trepunti(f,x[k],h)
> x
[1] -2.000000 -1.675325 -1.478238 -1.400446 -1.389020
[6] -1.388792 -1.388792 -1.388792 -1.388792 -1.388792
[11] -1.388792
```

Per determinare la terza ed ultima intersezione partiamo da $x_0 = 0.5$

```
> x<-0.5
> for(k in 1:10) x[k+1]<-x[k]-
+ f(x[k])/trepunti(f,x[k],h);x
[1] 0.5000000 0.3255812 0.3347240 0.3347341 0.3347341
[6] 0.3347341 0.3347341 0.3347341 0.3347341 0.3347341
[11] 0.3347341
```

Le radici sono 0.3347341, -1.388792, 1.214648.

2.4 Integrali definiti

Ci occupiamo ora del calcolo di integrali definiti

$$\int_a^b f(x)dx$$

2.4.1 Metodi numerici di integrazione di R

Per il calcolo di integrali definiti si scrive il comando

```
integrate(funzione, estremo inferiore, estremo superiore)
```

Ad esempio per x tra -1 e 2

```
> integrate(function(x) x^4,-1,2)
6.6 with absolute error < 7.3e-14
```

Il risultato è un numero e di tale numero viene fornito l'errore assoluto, molto basso (la tecnica utilizzata è buona).

2.4.2 Metodi di integrazione: metodo dei rettangoli e metodo dei trapezi

Ci proponiamo di calcolare

$$\int_a^b f(x)dx$$

suddividendo $[a, b]$ in n sottointervalli di eguale ampiezza. Richiamiamo la procedura: si suddivide l'intervallo $[a, b]$ in n sottointervalli di egual ampiezza determinando il passo del metodo $h = \frac{b-a}{n}$ e gli estremi degli intervalli $x_k = a + k h$ con $k = 0, \dots, n$. Si calcolano poi i corrispondenti valori $y_k = f(x_k)$ e si applicano poi le formule

$$\begin{aligned} L_n &= h \sum_{i=0}^{n-1} y_k \\ R_n &= h \sum_{i=1}^n y_k \\ T_n &= \frac{L_n + R_n}{2} \end{aligned}$$

che forniscono rispettivamente le stime a sinistra e a destra (metodo dei rettangoli) e la stima con il metodo dei trapezi. Integriamo per esempio una funzione $f(x) = x \cos(x)$ con $n = 10$, tra $a = 1$ e $b = 4$. Per prima cosa occorre determinare il valore del passo.

```
> 10->n
> 1->a
> 4->b
> (b-a)/n->h
```

Con il comando

```
> a+(0:n)*h->x;x
[1] 1.0 1.3 1.6 1.9 2.2 2.5 2.8 3.1 3.4 3.7 4.0
```

o anche con il comando

```
> x=seq(a,b,length=n+1)
```

si ottengono gli estremi degli intervalli. Si calcolano poi i valori corrispondenti della funzione e li si sostituiscono nelle formule di approssimazione

```
> f<-function(x) cos(x)*x
> f(x)
[1] 0.54030231 0.34774848 -0.04671924 -0.61425018
[5] -1.29470246 -2.00285904 -2.63822255 -3.09731897
[9] -3.28711385 -3.13797012 -2.61457448
> y<-f(x)
> ln<-h*sum(y[1:n] )
> rn<-h*sum(y[2:(n+1)])
> tn<-(ln+rn)/2
> tn
[1] -5.042563
```

Il confronto con il valore dell'integrale

```
> integrate(f,1,4)
-5.062627 with absolute error < 6e-14
```

è abbastanza buono. Aumentando il numero di punti della suddivisione la stima migliora.

```
> 100->n
> (b-a)/n->h
> seq(a,b,length=(n+1))->x
> y<-f(x)
> ln<-h*sum(y[1:n])
> rn<-h*sum(y[2:(n+1)])
> tn<-(ln+rn)/2
> tn
[1] -5.062426
> rn
[1] -5.109749
> ln
[1] -5.015103
```

2.4.3 Metodo Montecarlo

Supponiamo di voler applicare il metodo Montecarlo alla determinazione dell'integrale

$$\int_1^4 (x^3 + 3x) dx$$

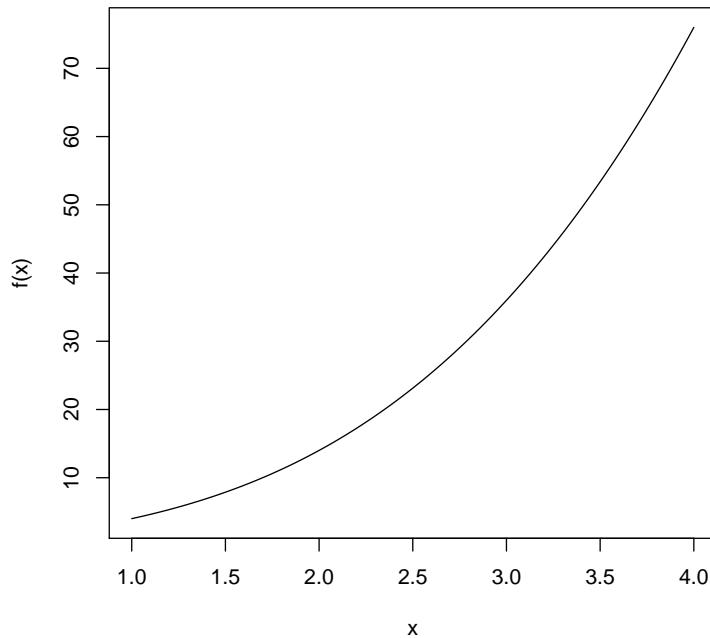


Figura 2.17: Determinazione del bersaglio

Occorre determinare la finestra grafica da utilizzare come bersaglio. Nell'esempio x varia tra 1 e 4. Per quanto riguarda la y tracciamo il grafico della funzione

```
> curve(x^3+3*x, 1, 4)
```

Questa è una funzione crescente (fatto che avrei potuto determinare anche dallo studio del segno della derivata) e nell'intervallo in esame ha un minimo

in $x = 1$ e un massimo in $x = 4$. I corrispondenti valori sono $m = 4$ e $M = 76$.

```
> f<-function(x) x^3+3*x
> f(4)
[1] 76
> f(1)
[1] 4
```

Possiamo quindi definire il bersaglio come

$$[1, 4] \times [0, 76]$$

Dobbiamo definire il numero di tiri. Il comando `runif(n, a, b)`, (*random-uniform*), permette di generare n numeri nell'intervallo (a, b) distribuiti uniformemente. Se l'intervallo non è specificato i numeri appartengono all'intervallo $(0, 1)$. Useremo il comando `runif` per effettuare i tiri.

```
> 1000->tiri;
> x<-runif(tiri,1,4)
> y<-runif(tiri,0,76)
> centri<-which(y<f(x))
> successi<-length(centri)
> successi/tiri*76*3
[1] 86.868
> integrate(f,1,4)
86.25 with absolute error < 9.6e-13
> plot(x[centri],y[centri],
+ col="red",cex=0.3)
> points(x[-centri],y[-centri],
+ col="blue",cex=0.3)
> curve(f,add=T)
```

In modo simile possiamo considerare la determinazione di π . Immaginiamo di dover determinare l'area del cerchio di raggio 1 che è racchiuso nel quadrato $[-1, 1] \times [-1, 1]$ del piano cartesiano. In tal caso i comandi sono

```
> 10000->tiri;
> x<-runif(tiri,-1,1)
> y<-runif(tiri,-1,1)
```

```
[1] 90.06  
86.25 with absolute error < 9.6e-13
```

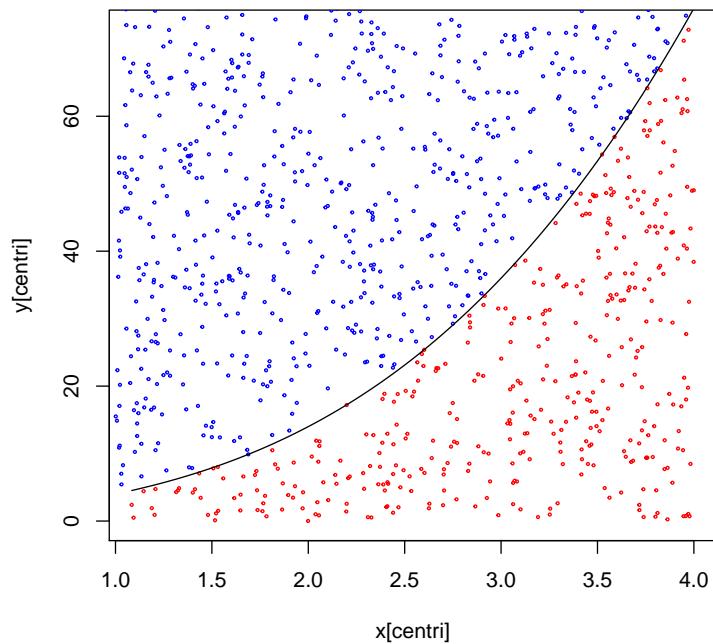


Figura 2.18: Metodo Montecarlo

```
> centri<-which(x^2+y^2<1)
> successi<-length(centri)
> successi/tiri*4
> plot(x[centri],y[centri],col="red",cex=0.4)
> points(x[-centri],y[-centri], col="blue",cex=0.4)
```

```
[1] 3.1336
```

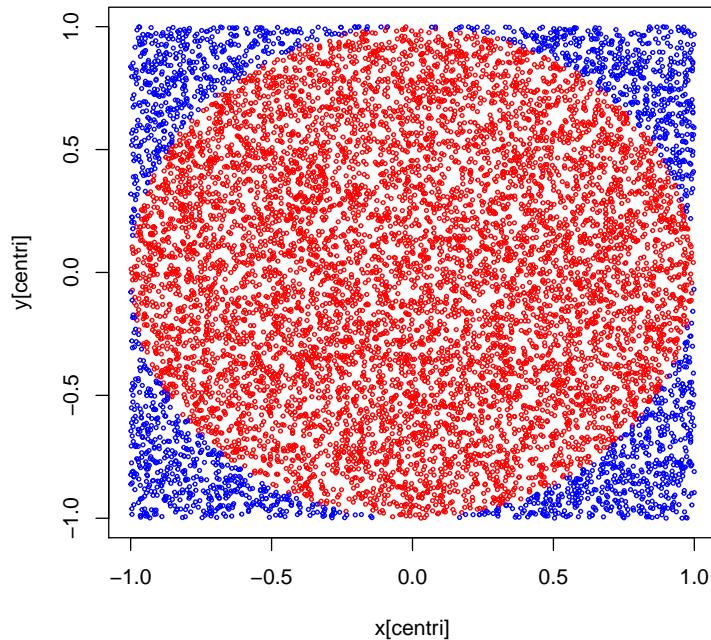


Figura 2.19: Metodo Montecarlo per il disco

SI noti che per ridurre la dimensione dei punti abbiamo usato l'opzione `cex`.

2.5 Numeri complessi

Un numero complesso viene indicato come

$$z = a + ib$$

dove a è la parte reale del numero complesso e b il coefficiente della parte immaginaria ib . In R un numero complesso viene scritto come

```
> z<-3 +4i
```

La componente immaginaria di un numero complesso contiene la lettera `i` minuscola. È possibile effettuare somme, sottrazioni, moltiplicazioni e divisioni usando i soliti simboli

```
> 2i+(4+5i)
[1] 4+7i
> (3+2i)*(3-1i)
[1] 11+3i
> (3+5i)/(4+2i)
[1] 1.1+0.7i
```

Alcune semplici operazioni con i numeri complessi

- La parte reale del numero complesso z si indica con `Re(z)`

```
> z<-3+4i
> Re(z)
[1] 3
```

- Il coefficiente della parte immaginaria di z si indica con `Im(z)`

```
> Im(z)
[1] 4
```

- Il coniugato del numero complesso $z = a + ib$ è $\bar{z} = a - ib$. In R si usa la funzione `Conj`:

```
> Conj(z)
[1] 3-4i
```

- L'argomento `arg` di un numero complesso $z = a + ib$ è definito come $\arg(z) = \arctan(b/a)$. In R si usa `Arg(z)` e per esempio

```
> Arg(z)
[1] 0.9272952
> atan(4/3)
[1] 0.9272952
```

- Il modulo di un numero complesso è semplicemente $\sqrt{z * \bar{z}}$ ed è indicato in R con Mod(z)

```
> Mod(z)
[1] 5
```

- L'esponenziale con esponente complesso si indica come al solito con exp(z)

```
> exp(z)
[1] -13.12878-15.20078i
> exp(z)*exp(Conj(z))
[1] 403.4288+0i
```

2.6 Variabili aleatorie discrete

Le variabili aleatorie discrete che assumono un numero limitato di valori si dicono anche *finite*. I valori di una variabile aleatoria discreta possono essere numerici o nominali. Supponiamo di avere una variabile aleatoria che possa assumere un insieme di valori in un *alfabeto* assegnato costituito da lettere, parole o numeri. Per esempio un alfabeto può essere del tipo che segue

- (Femmina, Maschio)
- (A,C,T,G)
- (0,1)
- (Ottimo, Buono, Discreto, Sufficiente, Insufficiente)
- (Testa, Croce).

- I numeri interi

Per caratterizzare completamente una variabile aleatoria discreta oltre ai valori che questa può assumere occorre conoscere la probabilità di questi valori.

Per semplicità considereremo variabili aleatorie finite.

Come possiamo simulare variabili aventi valore nell'alfabeto assegnato? In effetti qualunque comando di generazione su un computer non è perfettamente casuale; infatti la generazione avviene in effetti in modo pseudo-casuale e secondo un meccanismo che dipende dallo stato interno del computer codificato in una variabile indicata con `.Random.seed`. Se il *seme* iniziale è lo stesso i numeri generati saranno uguali. Spesso conviene che i calcoli (ad esempio a fine didattico) siano riproducibili. Ad esempio mettendo in una variabile `seme` il valore corrente di `.Random.seed` e richiamandolo o generandolo all'occorrenza. Scegliamo per la riproducibilità dei risultati

```
> seme=as.integer(c(0,1,2,3))
```

A questo punto possiamo simulare le variabili richieste usando la struttura

```
sample(alfabeto, n) (2.1)
```

Se l'alfabeto consiste di tutte le lettere minuscole dell'alfabeto ordinario e ne vogliamo selezionare $n = 8$ (in modo che ciascun uscita abbia la stessa probabilità) basta scrivere

```
> sample(letters,8)  
[1] "v" "m" "p" "q" "j" "l" "s" "t"
```

Se invece l'alfabeto consiste delle basi del DNA

```
> alfabeto=c("A","C","G","T")  
> sample(alfabeto,2)  
[1] "G" "C"  
>
```

Notiamo che

```
> sample(alfabeto)  
[1] "C" "T" "G" "A"
```

restituisce una permutazione dell'alfabeto, mentre chiedendo un campione di lunghezza superiore alla lunghezza dell'alfabeto otteniamo un messaggio di errore. Possiamo però immaginare di re-immettere la lettera estratta nell'urna dopo ogni estrazione. In questo caso non c'è limite alla sequenza generata. Per esempio

```
> alfabeto=c("testa","croce")
> sample(alfabeto,5,replace=T)
[1] "testa" "testa" "testa" "croce" "croce"
```

Il precursore del dado era chiamato astragalo ed era giocato nell'antica Grecia e nell'antica Roma [David].

Gli astragali sono dei piccoli ossicini di forma irregolare ed hanno 6 facce ma atterrano in modo stabile solo su 4 di esse numerate 1, 3, 4 e 6 con probabilità all'incirca 0.4 per il 3 e il 4 e di 0.1 per l'1 e il 6. Il tiro più gettonato all'epoca era l'uscita di 4 facce diverse nel lancio di 4 astragali e si chiamava *Venus*. Il lancio considerato peggiore sul singolo lancio era l'1 chiamato cane o avvoltoio. Torniamo ora ai classici dadi a 6 facce. Suppo-

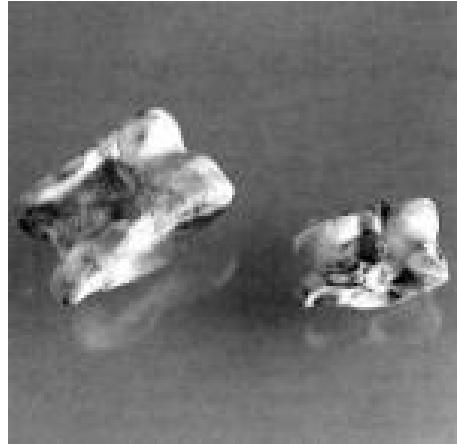


Figura 2.20: Astragalo.

niamo di lanciare 100 volte un dado equo a 6 facce e di registrare in **x** le uscite rilevate

```
> .Random.seed=seme
> dadi100<-sample(1:6,100,replace=T)
> dadi100
```

```
[1] 1 5 1 5 3 6 5 5 5 6 6 3 5 6 6 5 5 2 4 6 1 4 2 3 4 5 2 4  
[29] 5 6 3 3 5 5 6 1 6 4 2 6 6 3 2 2 3 5 1 3 3 2 5 6 3 4 2 4  
[57] 2 1 4 1 6 5 6 2 2 3 4 3 3 4 3 2 4 3 6 3 1 3 2 4 6 1 3 1  
[85] 1 6 6 6 5 5 3 3 3 4 1 4 4 2 5 3
```

Volendo invece simulare una combinazione da giocare al SuperEnalotto possiamo scrivere

```
> x<-sample(1:90,6,replace=T);x  
[1] 13 76 66 76 17 53
```

I numeri usciti sono stati salvati in una variabile `x`, per poter effettuare la ricerca di indicatori statistici. Il comando che consente di ordinare una lista o un vettore è `sort`, esso può essere usato in associazione al nome di una variabile o di una lista, ossia:

(2.2)

```
sort(variabile/lista)
```

Volendo ordinare i numeri precedentemente ricavati scriveremo

```
> sort(x)  
[1] 13 17 53 66 76 76
```

2.7 Statistica descrittiva: singola variabile

2.7.1 Indicatori statistici

- Media.

La media di una serie di numeri si ottiene con la funzione `mean` scrivendo: `mean(variabile)`. Ad esempio, lavorando con la lunghezza del sepalo di 150 piante di iris

```
> mean(x=iris[,1])  
[1] 5.843333
```

- Varianza campionaria

Si ottiene con la funzione predefinita di espressione: `var(variabile)`. Possiamo calcolare la varianza come

```
> var(x)  
[1] 814.9667
```

- Deviazione Standard campionaria.

Non è altro che la radice della varianza. Si ottiene con la funzione pre-definita di espressione: `sd(variable)`. Sempre basandosi sull'esempio precedente scriveremo

```
> sd(x)  
[1] 28.54762
```

- Quantili. La notazione standard è semplicemente: `quantile(variable)` che determina i quartili e ci fornisce in uscita la statistica dei 5 numeri

```
> quantile(x)  
0% 25% 50% 75% 100%  
13.0 26.0 59.5 73.5 76.0
```

Volendo ricavare i decili dovremo scrivere:

```
quantile(variable, seq(0,1,by=0.1))
```

in quanto vogliamo dividere l'intervallo $[0, 1]$ a passo 0.1

Nell'esempio:

```
> quantile(x,seq(0,1,by=0.1))  
0% 10% 20% 30% 40% 50% 60% 70% 80% 90% 100%  
13.0 15.0 17.0 35.0 53.0 59.5 66.0 71.0 76.0 76.0 76.0
```

Si noti che `quantile` ammette 9 varianti specificabili con l'opzione `type = n` dove n va da 1 a 9. Per esempio

```
> quantile(x,type=4)  
0% 25% 50% 75% 100%  
13 15 53 71 76
```

Sui dati in esame le 9 varianti coincidono.

Per quanto riguarda gli indicatori statistici nel caso di dati ripetuti basta notare che se la lista x contiene i valori e la lista f le frequenze assolute il comando

$\text{rep}(x, f)$

costruisce un'unica lista dei dati inclusiva delle ripetizioni. Per esempio

```
> x=1:6
> f=c(9,7,9,7,8,10)
> y=rep(x,f)
> y
[1] 1 1 1 1 1 1 1 1 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 4 4 4 4 4
[31] 4 4 5 5 5 5 5 5 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
```

Ovviamente senza bisogno di visualizzare y possiamo calcolarne tutti gli indicatori statistici. Il comando

```
> cumsum(f)
[1] 9 16 25 32 40 50
```

restituisce le frequenze cumulate, dalle quali si possono ricavare facilmente la mediana i quantili.

2.7.2 Raggruppamenti in classi

Consideriamo la rilevazione della temperatura media giornaliera di Milano nel mese di settembre 2010.

```
> sito="http://www.ilmeteo.it/portale/archivio-meteo/"
> indirizzo=paste(sito,"Milano/2010/Settembre?format=csv",sep="")
> meteo=read.table(indirizzo,sep=";")

> dim(meteo)
[1] 31 15
> meteo[-1,3]
[1] 18 19 21 22 22 20 17 19 18 20 19 20 20 19 20 22 22 18 18 18
[21] 18 19 19 17 14 15 15 15 15 15
Levels: 14 15 17 18 19 20 21 22 TMEDIA \260C
```

A questo punto eliminiamo i livelli di `meteo` con il comando `as.vector` e consideriamo il risultato come numerico con

```
> as.numeric(as.vector(meteo[-1,3]))->Milano;
> Milano
[1] 18 19 21 22 22 20 17 19 18 20 19 20 20 19 20 22 22 18 18 18
[21] 18 19 19 17 14 15 15 15 15 15 15
> quantile(Milano)
 0%   25%   50%   75% 100%
14.00 17.25 19.00 20.00 22.00
```

L'ultimo comando in particolare ci fornisce minimo e massimo dei dati. Possiamo esaminare la serie temporale dei dati con i comandi

```
> plot(Milano,type="l",xlab="settembre 2010 a milano",ylab="temperatura media")
ottenendo la figura 5.2
```

Raggruppiamo ora i dati in classi comprese tra due estremi che comprendono certamente tutti i dati, per esempio 13 e 23, decidendo di applicare un passo di 2 e vedere come si distribuiscono. Il comando `cut` associa a ciascun dato la classe di appartenenza selezionata in base ai punti di taglio.

```
> cut(Milano,breaks=seq(13.,23.,by=2))
[1] (17,19] (17,19] (19,21] (21,23] (21,23] (19,21] (15,17]
[8] (17,19] (17,19] (19,21] (17,19] (19,21] (19,21] (17,19]
[15] (19,21] (21,23] (21,23] (17,19] (17,19] (17,19] (17,19]
[22] (17,19] (17,19] (15,17] (13,15] (13,15] (13,15] (13,15]
[29] (13,15] (13,15]
Levels: (13,15] (15,17] (17,19] (19,21] (21,23]
```

Il comando `table` conta i dati di ciascuna classe

```
> table(cut(Milano,breaks=seq(13.,23.,by=2)))
(13,15] (15,17] (17,19] (19,21] (21,23]
       6        2       12        6        4
```

Si noti che la suddivisione in classi prevede intervalli aperti a sinistra e chiusi a destra. Per suddividere in modo che gli intervalli siano chiusi a sinistra e aperti a destra si specifica il parametro `right=FALSE`. Possiamo anche usare il comando `seq` per specificare i tagli.

```
table(cut(variabile,breaks=seq(estremo inf,
estremo sup,by = passo),right=FALSE))
```

27 ottobre 2014

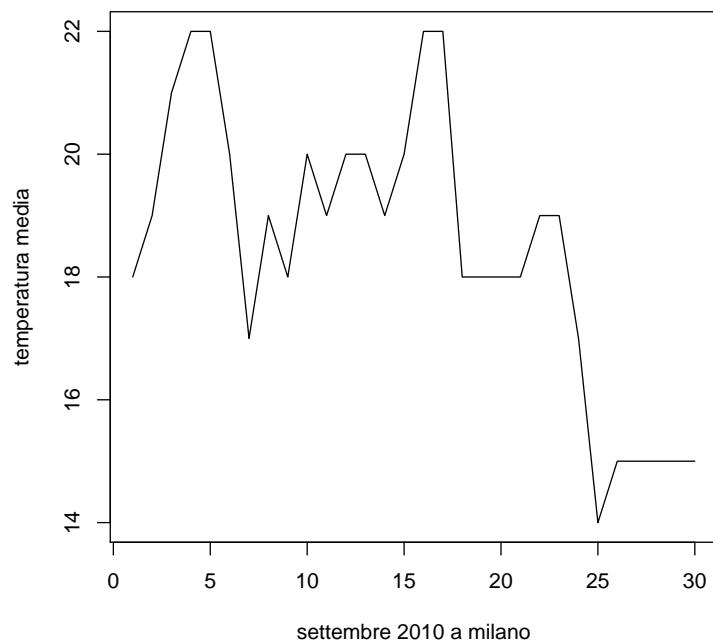


Figura 2.21: Andamento della temperatura a settembre 2010.

o in modo più generale

```
table(cut( variabile,  
breaks=c(estremo inferiore,...,estremo superiore))
```

estremamente utile in quanto consente di raggruppare i dati in classi non necessariamente di ugual ampiezza.

```
> table(cut(Milano,breaks=c(12.5,14,15,16, 18,20,22.5),  
+ right=F))  
[12.5,14) [14,15) [15,16) [16,18) [18,20) [20,22.5)  
0 1 5 2 12 10
```

Volendo raggruppare in classi i dati delle precedenti uscite del dado possiamo scrivere

```
> table(cut(dadi100,breaks=0:6))  
(0,1] (1,2] (2,3] (3,4] (4,5] (5,6]  
12 14 22 15 18 19
```

Se scegliamo di chiudere a sinistra gli intervalli

```
> table(cut(dadi100,breaks= 1:7,right=FALSE))  
[1,2) [2,3) [3,4) [4,5) [5,6) [6,7)  
12 14 22 15 18 19
```

In questo caso la occorre prestare attenzione alla chiusura agli estremi degli intervalli

2.7.3 Areogrammi

Il comando generico per generare un istogramma è:

```
hist(variabile)
```

che segue però la struttura del comando `cut`. L'ampiezza di ciascuna classe salvo diversamente indicato è costante e decisa da R. È possibile variare tale condizione definendo una lista con i punti di taglio (*cutoff*) delle classi volute:

```
hist(variabile,c(valore1,valore2,...)) (2.3)
```

Per esempio se `dadi100` rappresenta le solite 100 uscite del lancio del dado, il comando

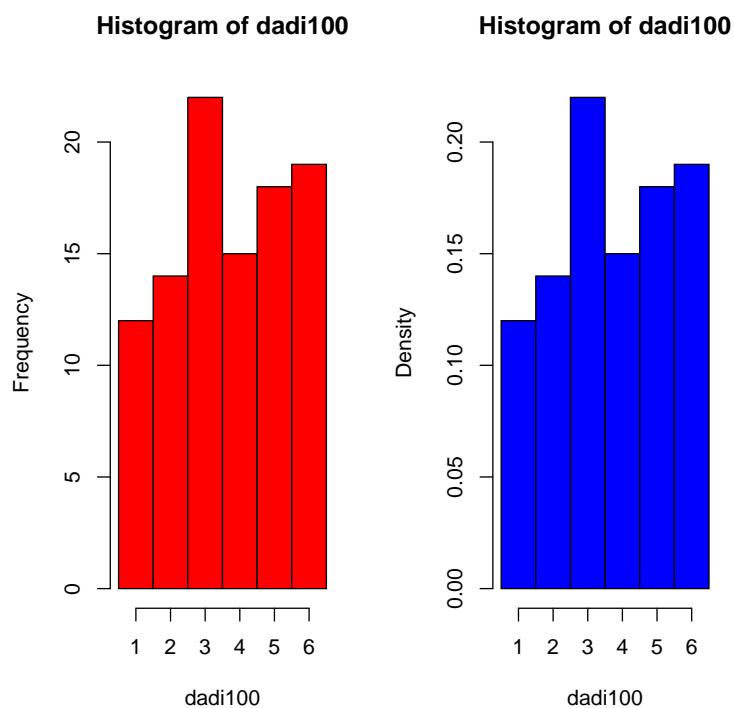


Figura 2.22: Diagramma a colonne e areogramma per il lancio di un dado.

```
> par(mfrow=c(1,2))
> hist(dadi100,breaks=seq(0.5,6.5,1),col="red")
> hist(dadi100,freq=FALSE,breaks=seq(0.5,6.5,1),col="blue")
```

genera l'istogramma (in rosso, a sinistra Figura 2.21) con le frequenze assolute delle classi in ordinata. La sequenza dei punti di taglio è stata scelta in modo che i numeri interi da 1 a 6 siano al centro delle classi corrispondenti. Se invece volessimo creare un areogramma (ossia avere un tracciato per cui le aree siano pari alle frequenze relative) a partire dalle stesse uscite dovremo impostare il parametro `freq=FALSE` ottenendo il pannello a destra (in blu) della figura (2.21). Avendo scelto classi di ampiezza costante i 2 grafici differiscono semplicemente per un cambio di scala sull'asse y .

In modo simile possiamo tracciare un areogramma dei dati nella variabile `milano`

```
> par(mfrow=c(1,2))
> hist(Milano, col="green", freq=FALSE, right=FALSE,
+ main="Cutoff automatici")
```

lasciando R libero di scegliere i punti di taglio (pannelli a sinistra della figura 2.23) o scegliendoli a nostra volta (pannelli a destra della stessa figura 2.23)

```
> hist(Milano, col="red", freq=FALSE,
+ breaks=c(12.5,15,17,18,20,22.5), right=FALSE,
+ main="Cutoff personalizzati")
```

Si noti la stabilità degli areogrammi rispetto ai cambi nella suddivisione.

2.7.4 Generazione di boxplot

Il `boxplot` è una rappresentazione grafica immediata della statistica dei 5 numeri e simultaneamente ci segnala eventuali punti discordanti o anomali, *outlier*. Il comando generico è:

$$\text{boxplot}(\text{variabile}) \quad (2.4)$$

prendendo il vettore x contenente i risultati di 100 lanci otteniamo la figura 2.24 da cui si evince che il valore massimo dei dati è 6, il minimo è 1 e non ci sono punti anomali, per cui non vi sono dati anomali, altrimenti evidenziati da un pallino. Si legge inoltre il valore di mediana (4) primo quartile (2) e terzo quartile (5).

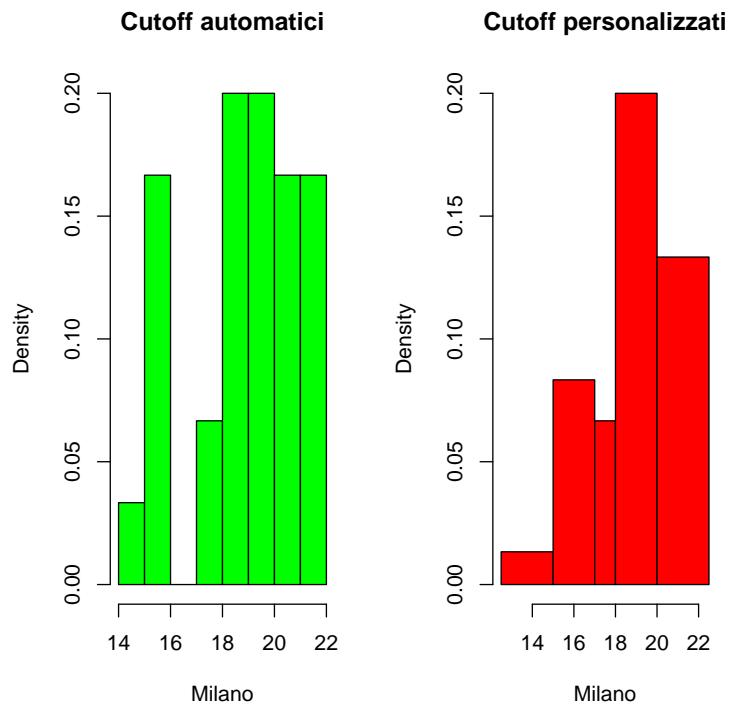


Figura 2.23: Areogramma dei dati della temperatura. Scelta automatica dei punti di taglio.

```
> boxplot(dadi100)
```

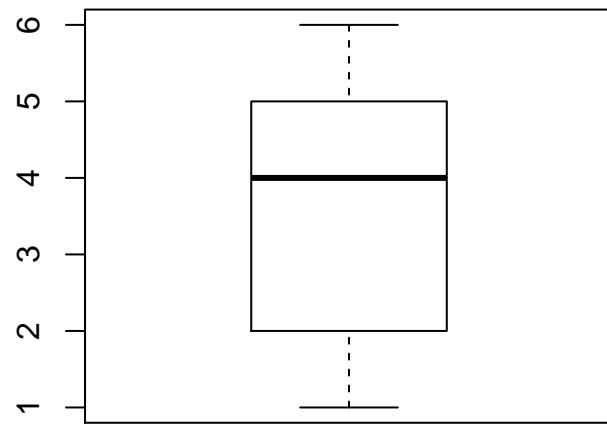


Figura 2.24: Boxplot dei risultati del lancio di un dado

2.7.5 Creazione di grafici a torta

Il comando `pie` consente, partendo da una tabella, di tracciare il diagramma a torta per una variabile nominale raggruppata in classi. Il comando è

```
pie(table(variabile))
```

ad esempio (facendo riferimento ai precedenti dati):

```
> pie(table(dadi100))
```

fornisce in uscita la Figura 2.25

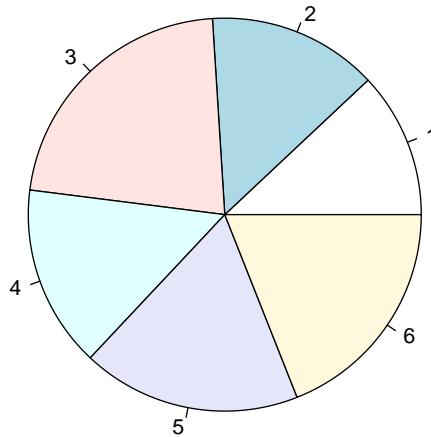


Figura 2.25: Diagramma a torta per il lancio di un dado equo.

Costruire una matrice contenente le coordinate di 50 punti nel rettangolo $[0, 4] \times [0, 2]$ in due dimensioni (generate utilizzando il generatore di numeri

pseudocasuali). Produrre un grafico con due pannelli, dove il primo pannello è uno scatter-plot

2.8 Variabili doppie e rette di regressione

Supponiamo di misurare la concentrazione di acido lattico muscolare durante uno sforzo di 10 minuti,

```
> x<-tempo<-c(1,2,3,4,5,6,7,8,9,10)
> y<-concentrazione<-c(0.3,0.65,0.7,0.8,0.95,1.05,1.3,1.7,1.9,
+ 2.5)
```

Per analizzare questi dati conviene preliminarmente tracciarne un diagramma a dispersione. Possiamo inoltre determinare il coefficiente di correlazione lineare

```
> cor(x,y)
[1] 0.9620456
```

Per definire un modello di relazione lineare occorre usare il comando `lm` (*linear model*). Nella sua generica forma il comando è espresso come⁴

$$\text{lm}(y \sim x)$$

Otteniamo i valori di pendenza e intercetta.

Possiamo tracciare la retta di regressione con il comando `abline`.

```
> plot(x,y,pch=19,col="red")
> abline(lm(y~x),col="blue")
```

Per determinare la retta di regressione sulle y dobbiamo invertire x e y .

```
> lm(x~y)
Call:
lm(formula = x ~ y)

Coefficients:
(Intercept)          y
0.3516        4.3446
```

⁴ Per digitare la tilde \sim su Mac premere ALT 5 su PC invece il tasto Alt Gr (attivazione del codice ASCII) e sul tastierino numerico digitare il numero 126. Lavorando su un portatile il tastierino numerico è spesso incorporato nella tastiera con colorazione blue dei tasti.

```
> plot(x,y)
```

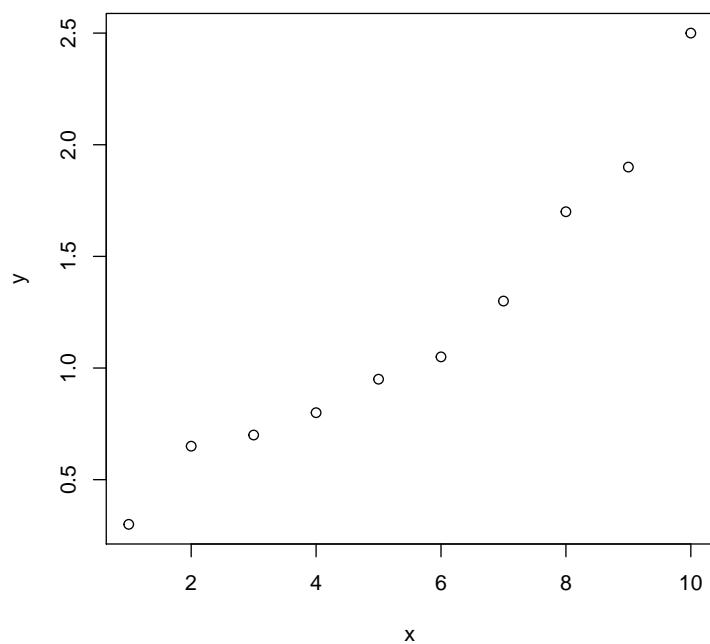


Figura 2.26: Diagramma a dispersione tempo/concentrazione.

```

> lm(x~y)$coefficients->coeff
> a=1/coeff[2];a;
      y
0.2301707
> b=-coeff[1]/coeff[2];
> abline(b,a,col="green")

```

In tal modo otteniamo il grafico 2.27. Consideriamo ora il seguente *dataset*

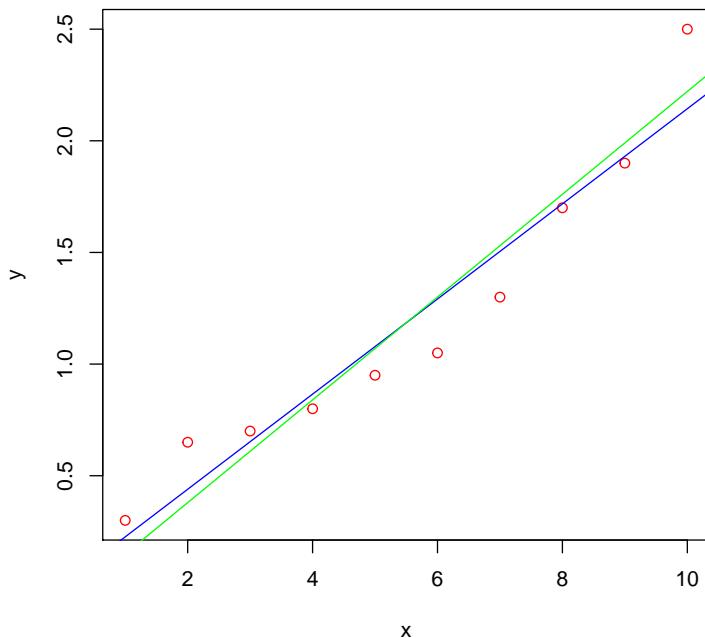


Figura 2.27: Rette di regressione. In blu R_x , in verde R_y .

di mammiferi in cui le 2 variabili rappresentano le dimensioni del corpo e del cervello.

```

> library(MASS)
> mammals
      body    brain
Arctic fox   3.385   44.50

```

Owl monkey	0.480	15.50
Mountain beaver	1.350	8.10
Cow	465.000	423.00
Grey wolf	36.330	119.50
Goat	27.660	115.00
Roe deer	14.830	98.20
Guinea pig	1.040	5.50
Verbet	4.190	58.00
Chinchilla	0.425	6.40
Ground squirrel	0.101	4.00
Arctic ground squirrel	0.920	5.70
African giant pouched rat	1.000	6.60
Lesser short-tailed shrew	0.005	0.14
Star-nosed mole	0.060	1.00
Nine-banded armadillo	3.500	10.80
Tree hyrax	2.000	12.30
N.A. opossum	1.700	6.30
Asian elephant	2547.000	4603.00
Big brown bat	0.023	0.30
Donkey	187.100	419.00
Horse	521.000	655.00
European hedgehog	0.785	3.50
Patas monkey	10.000	115.00
Cat	3.300	25.60
Galago	0.200	5.00
Genet	1.410	17.50
Giraffe	529.000	680.00
Gorilla	207.000	406.00
Grey seal	85.000	325.00
Rock hyrax-a	0.750	12.30
Human	62.000	1320.00
African elephant	6654.000	5712.00
Water opossum	3.500	3.90
Rhesus monkey	6.800	179.00
Kangaroo	35.000	56.00
Yellow-bellied marmot	4.050	17.00
Golden hamster	0.120	1.00
Mouse	0.023	0.40
Little brown bat	0.010	0.25
Slow loris	1.400	12.50

Okapi	250.000	490.00
Rabbit	2.500	12.10
Sheep	55.500	175.00
Jaguar	100.000	157.00
Chimpanzee	52.160	440.00
Baboon	10.550	179.50
Desert hedgehog	0.550	2.40
Giant armadillo	60.000	81.00
Rock hyrax-b	3.600	21.00
Raccoon	4.288	39.20
Rat	0.280	1.90
E. American mole	0.075	1.20
Mole rat	0.122	3.00
Musk shrew	0.048	0.33
Pig	192.000	180.00
Echidna	3.000	25.00
Brazilian tapir	160.000	169.00
Tenrec	0.900	2.60
Phalanger	1.620	11.40
Tree shrew	0.104	2.50
Red fox	4.235	50.40

Per prima cosa tracciamo il grafico dei punti in scala non trasformata e, visto la compresenza di dati molto prossimi all'origine e di dati molto distanti in scala logaritmica (sia le x che le y vengono trasformate prendendone i logaritmi)

```
> par(mfrow=c(1,2))
> plot(mammals)
> plot(mammals,log="xy")
```

come in Figura 2.28. Visti i risultati ottenuti usando la scala logaritmica tracciamo anche la corrispondente retta di regressione

```
> plot(log(mammals$brain)~log(mammals$body),col="BLUE",pch=19,type="p")
> abline(lm(log(mammals$brain)~ log(mammals$body)),col="red",lwd=3);
> uomo=which(rownames(mammals)=="Human")
> text(log(mammals[uomo ,1]),log(mammals[uomo ,2]),rownames(mammals)[uomo])
```

Si noti il comando `text(x,y, testo)` dove x e y e `testo` sono vettori di arbitraria lunghezza contenenti ascisse, ordinate e testo da inserire.

```
> par(mfrow=c(1,2))
> plot(mammals)
> plot(mammals,log="xy")
```

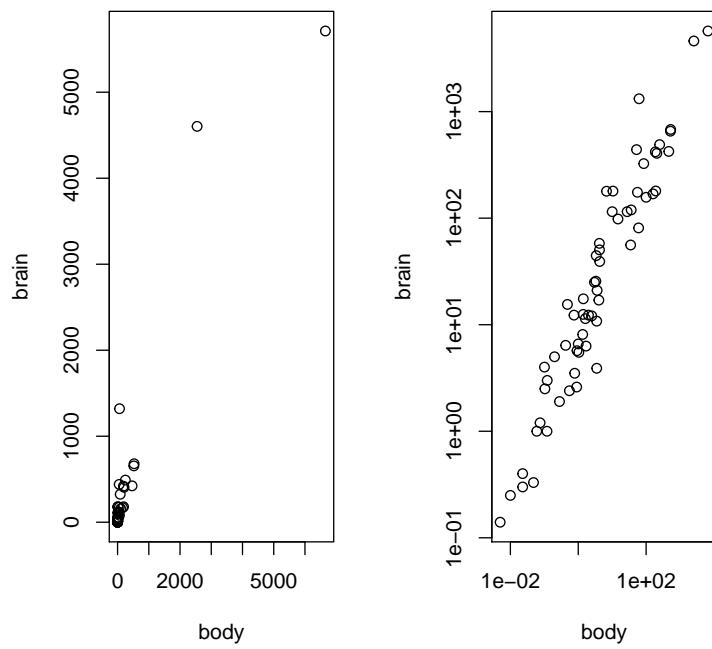


Figura 2.28: Diagramma a dispersione massa corporea/massa del cervello in scala normale ed in scala logaritmica.

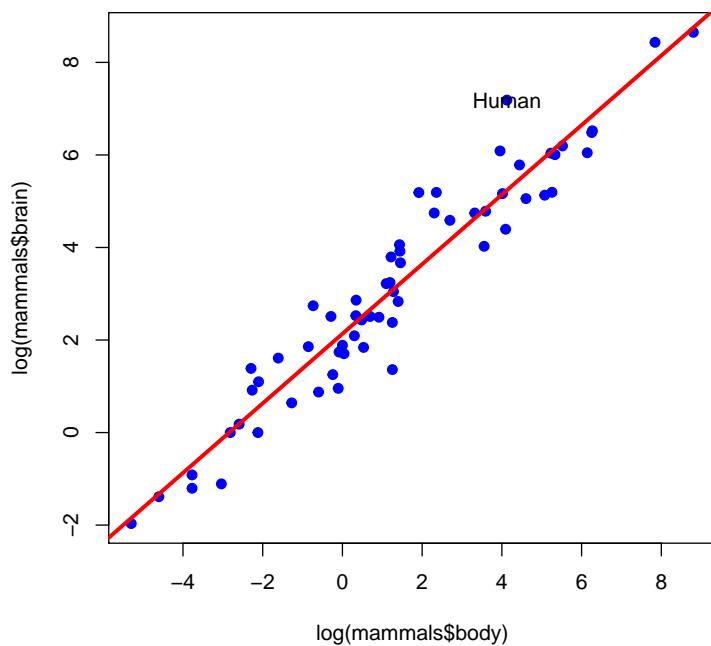


Figura 2.29: Retta di regressione. Dimensione del corpo e del cervello. Si noti la posizione dell'uomo.

2.9 Distribuzioni in R

I nomi delle principali distribuzioni in R sono

<code>norm</code>	normale
<code>t</code>	Student
<code>chisq</code>	chi quadro
<code>f</code>	Fisher
<code>binom</code>	binomiale

A questi nomi possiamo aggiungere diversi prefissi

<code>d</code>	densità
<code>p</code>	primitiva
<code>q</code>	quantile
<code>r</code>	random

per caratterizzare diversi aspetti.

2.9.1 Distribuzione normale

La funzione `dnorm`

Come appena visto R indica con il nome `dnorm`, la densità normale o gaussiana. Essa accetta come parametri sia la media μ che la deviazione standard σ come è possibile verificare con il comando `formals` che ci fornisce gli argomenti di una funzione e gli eventuali valori preassegnati.

```
> formals(dnorm)
```

```
$x
```

```
$mean  
[1] 0
```

```
$sd  
[1] 1
```

```
$log  
[1] FALSE
```

Se i parametri sono omessi `dnorm` rappresenta la densità normale standard con $\mu = 0$ e $\sigma = 1$. Il grafico (2.30) della gaussiana tra due estremi, ad esempio -2.5 e 2.5 si ottiene con il solito comando

```
> curve(dnorm, -2.5, 2.5)
```

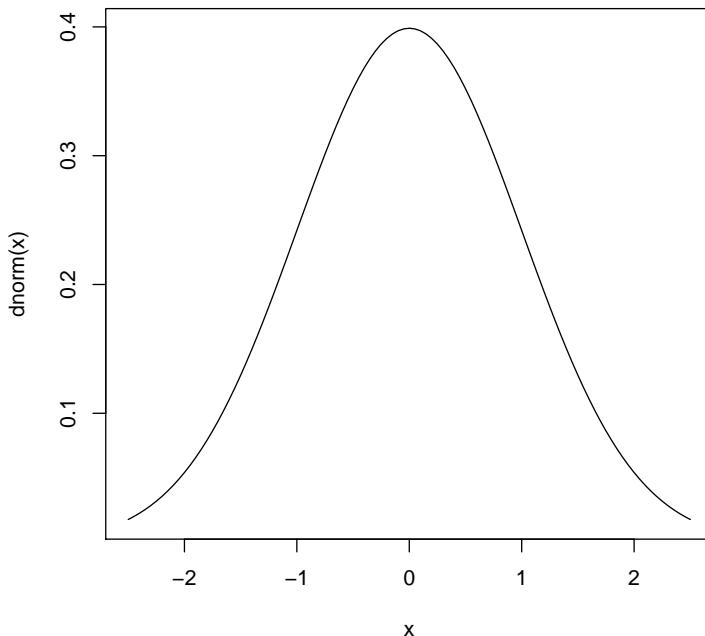


Figura 2.30: Grafico della normale standard nell'intervallo $[-2.5, 2.5]$.

Per visualizzare una gaussiana non standard, ad esempio con media $\mu = 1$, deviazione standard $\sigma = 1.5$, tra -3 e 3. scriveremo invece

```
> curve(dnorm(x, mean=1, sd=1.5), -3, 3)
```

La funzione `pnorm`

La funzione `pnorm(x)` è definita come

$$pnorm(x) = \int_{-\infty}^x dnorm(s)ds$$

Ovviamente

$$\int_a^b \text{dnorm}(x)dx = \text{pnorm}(b) - \text{pnorm}(a)$$

e per avere l'area sottesa tra 3 e 5 basta scrivere:

```
> pnorm(5)-pnorm(3)
[1] 0.001349611
```

Per ottenere il valore dell'area tra 0 e x bisogna allora sottrarre $\text{pnorm}(0)=0.5$ all'area fornita dalla funzione. Per cui possiamo scrivere:

```
> pnorm(1)-0.5
[1] 0.3413447
```

La funzione qnorm e la tabella della densità di Gauss

Notiamo preliminarmente che se

$$\int_{-x}^x \text{dnorm}(s)ds = y$$

allora

$$\text{pnorm}(x) = \int_{-\infty}^x \text{dnorm}(s)ds = 1 - \frac{1-y}{2} = \frac{1+y}{2}$$

La funzione **qnorm** rappresenta la funzione inversa di **pnorm** quindi

$$x = \text{qnorm}\left(\frac{1+y}{2}\right)$$

In termini pratici possiamo introdurre la funzione

```
> u<-function(area) qnorm((1+area)/2)
```

fornisce fissato il livello di fiducia l'ascissa x tale che l'intervallo simmetrico $[-x, x]$ racchiuda un'area pari al livello di fiducia. Per esempio

```
> u(0.95)
[1] 1.959964
```

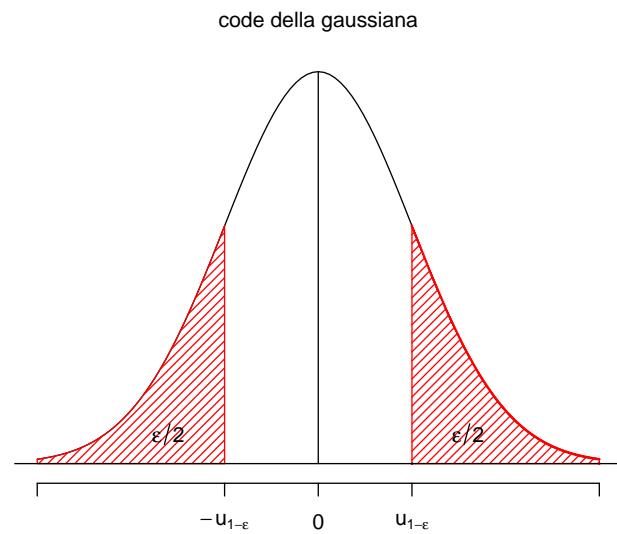


Figura 2.31: Code della distribuzione normale

La funzione `rnorm`

È possibile generare dei valori standardizzati casuali (media uguale a 0, deviazione standard pari a 1) che seguono la distribuzione normale standard. Basta semplicemente definire il numero di valori desiderati. Il comando nella sua espressione generale è:

$$\text{rnorm}(n, \text{mean} = \text{valore}_1, \text{sd} = \text{valore}_2) \quad (2.5)$$

Nel caso in cui volessimo una lista di 20 valori di una variabile normale con media assegnata 5 e deviazione standard 1 scriveremo :

```
> rnorm(20, mean=5, sd=1)
```

2.9.2 La distribuzione *t* di Student

In R la distribuzione di Student è indicata con la lettera `t`. Come per le altre densità si possono considerare le funzioni

dt	densità
pt	primitiva
qt	quantili
rt	generatore random

Il grafico della distribuzione di Student ad un certo numero `df` di gradi di libertà si ottiene con il comando

```
curve(dt(x, df), a, b)
```

Tracciamo ad esempio un grafico tra -2 e 2 per una distribuzione a 10 gradi di libertà (vedi figura (2.32)):

Ricordiamo che la distribuzione di Student si usa in particolare nei casi in cui la deviazione standard della popolazione σ non è conosciuta e viene rimpiazzata dalla deviazione standard campionaria S , calcolata con un numero N di dati e quindi con $N - 1$ gradi di libertà. Quando però il numero di dati si avvicina a 30 la curva di Student è praticamente sovrapposta a quella della distribuzione normale, come mostra il grafico (2.33):

2.9.3 Intervalli di confidenza e test di Student (dati non appaiati)

La funzione di R che esegue il test di Student nelle sue diverse forme è `t.test`
Nella sua forma più semplice

```
> curve(dt(x,10),-2,2)
```

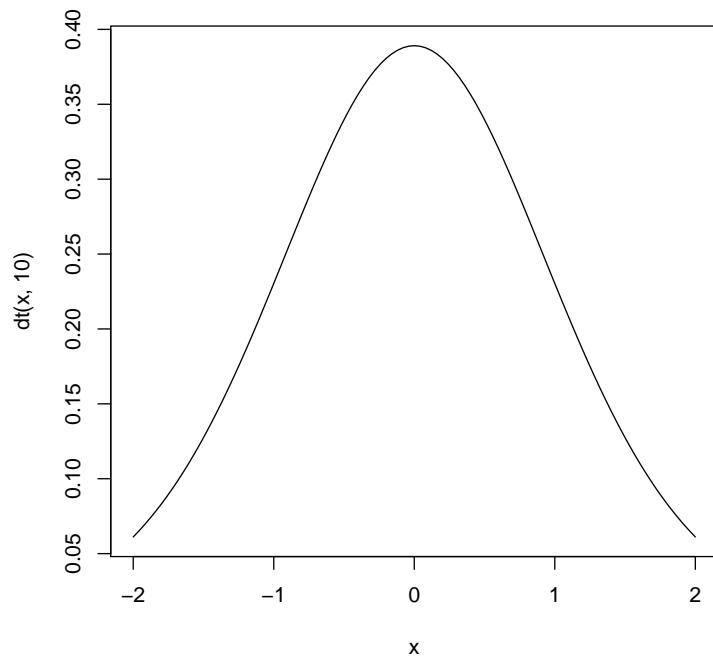


Figura 2.32: Grafico della distribuzione di Student a 10 gradi di libertà.

```
> curve(dnorm(x),-2,2,col=3)
> curve(dt(x,2),-2,2,col=1,add=T)
> curve(dt(x,25),-2,2,col=2,add=T)
> legend("topleft", c("df=2","df=25","normale"),pch=15,col=1:3);
```

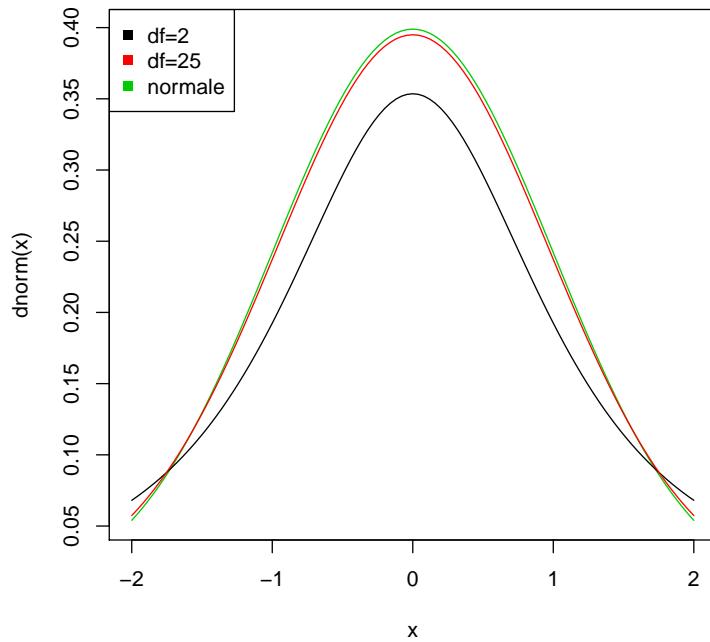


Figura 2.33: Grafico della distribuzione di Student a 10 gradi di libertà

```
> x=1:20; t.test(x)
One Sample t-test

data: x
t = 7.9373, df = 19, p-value = 1.884e-07
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 7.731189 13.268811
sample estimates:
mean of x
10.5
```

In assenza di ipotesi R calcola il consuntivo

$$t = \frac{M_N(X) - \mu}{S_X} \sqrt{N}$$

assumendo che sia $\mu = 0$. Possiamo anche eseguire specificare l'ipotesi sul valore di μ :

```
> t.test(x,mu=7)
One Sample t-test

data: x
t = 2.6458, df = 19, p-value = 0.01595
alternative hypothesis: true mean is not equal to 7
95 percent confidence interval:
 7.731189 13.268811
sample estimates:
mean of x
10.5
```

Possiamo infine specificare l'ipotesi alternativa. Per esempio se l'ipotesi alternativa è "less" il risultato del test cambia completamente.

```
> t.test(x,mu=7, alternative="less")
One Sample t-test

data: x
t = 2.6458, df = 19, p-value = 0.992
alternative hypothesis: true mean is less than 7
```

```
95 percent confidence interval:  
-Inf 12.78743  
sample estimates:  
mean of x  
10.5
```

In pratica ci viene fornito come *p*-value il valore dell'area sottesa dalla distribuzione di Student da $-\infty$ al valore di *t* se l'ipotesi alternativa è "less" e il valore dell'area sottesa dalla distribuzione di Student dal valore di *t* a $+\infty$ se l'ipotesi alternativa è "greater"

2.9.4 Test di Student per dati appaiati

Il test di Student per dati appaiati non è altro che un test di Student sulla differenza di 2 liste di dati di ugual lunghezza. Consideriamo ad esempio il confronto di 2 tecniche di misura applicate agli stessi campioni

```
> x<-c(1.46,2.22,2.84,1.97,1.13,2.35)  
> y<-c(1.42,2.38,2.67,1.8,1.09,2.25)
```

Possiamo calcolare la differenza *x*-*y* ed applicare il test di Student oppure ottenere lo stesso risultato specificando l'opzione paired=TRUE

```
> t.test(x,y,paired=TRUE)  
Paired t-test  
  
data: x and y  
t = 1.2, df = 5, p-value = 0.2839  
alternative hypothesis: true difference in means is not equal to 0  
95 percent confidence interval:  
-0.06852909 0.18852909  
sample estimates:  
mean of the differences  
0.06
```

Il consuntivo *t* cade entro la regione di accettazione del test. È possibile specificare il livello di fiducia da utilizzare per il test di Student come:

```
conf.level = numero
```

Il comando completo di tutti i parametri è quindi:

```
t.test(dati1, dati2,  
paired=TRUE, conf.level = valore)
```

Ad esempio eseguiamo un t -test per dati appaiati, tra $x = (1, 2, 3, 4)$ e $y = (3, 2, 4, 5)$ con *confidence level* di 0.85. Scriveremo

```
> t.test(1:4,5:2,paired=TRUE,conf.level=0.85)
```

Il consuntivo t cade fuori dalla regione di accettazione proposta.

2.10 Test χ^2 di indipendenza

Consideriamo il seguente *dataframe* che riporta le ambizioni di un gruppo di scolari americani

```
> read.table(  
+ "http://lib.stat.cmu.edu/DASL/Datafiles/PopularKids.html",  
+ skip=39,header=T,nrow=478,sep="\t")->kidinterest
```

	Gender	Grade	Age	Race	Urban.Rural	School	Goals
1	boy	5	11	White	Rural	Elm	Sports
2	boy	5	10	White	Rural	Elm	Popular
3	girl	5	11	White	Rural	Elm	Popular
4	girl	5	11	White	Rural	Elm	Popular
5	girl	5	10	White	Rural	Elm	Popular
6	girl	5	11	White	Rural	Elm	Popular
	Grades	Sports	Looks	Money			
1	1	2	4	3			
2	2	1	4	3			
3	4	3	1	2			
4	2	3	4	1			
5	4	2	1	3			
6	4	2	1	3			

Nella tabella le colonne che ci interessano al momento sono quelle che riguardano il sesso, gli obiettivi (scelti tra successo scolastico, capacità sportiva e popolarità) e la provenienza (colonne 1, 5 e 7). Nelle colonne dalla 8 alla 11 sono messi in ordine di importanza per il conseguimento della popolarità voti, sport, aspetto esteriore e denaro.

```
> colnames(kidinterest)  
[1] "Gender"      "Grade"       "Age"  
[4] "Race"        "Urban.Rural" "School"  
[7] "Goals"        "Grades"      "Sports"  
[10] "Looks"       "Money"
```

```
> interessi=kidinterest[,c(1,5,7)]
> head(intessi)
  Gender Urban.Rural  Goals
1   boy      Rural Sports
2   boy      Rural Popular
3  girl      Rural Popular
4  girl      Rural Popular
5  girl      Rural Popular
6  girl      Rural Popular
> table(intessi)
, , Goals = Grades

  Urban.Rural
Gender Rural Suburban Urban
  boy     21      51    45
  girl    36      36    58

, , Goals = Popular

  Urban.Rural
Gender Rural Suburban Urban
  boy     19      20    11
  girl    31      22    38

, , Goals = Sports

  Urban.Rural
Gender Rural Suburban Urban
  boy     26      18    16
  girl    16       4    10
```

Consideriamo per esempio le variabili provenienza e traguardi

```
> interessi2=kidinterest[,c(5,7)]
> tabella=table(intessi2)
> tabella

  Goals
Urban.Rural Grades Popular Sports
  Rural        57      50     42
```

Suburban	87	42	22
Urban	103	49	26

Il test χ^2 di indipendenza consente di verificare se due variabili sono indipendenti. Se consideriamo le due variabili precedenti sesso e interassi. R dispone del comando `chisq.test`, dalla sintassi generale:

```
chisq.test(tabella)
```

Nell'esempio

```
> studenti<-read.csv(  
+ "http://www.pharm.unipmn.it/rinaldi/file_corso/studenti.csv")  
> head(studenti)  
  X Sex  W   H Eyes Hair Sh hM hF  
1 1   M 86 1.90 castani neri 48 1.58 1.82  
2 2   M 53 1.76 azzurri castani 42 1.70 1.60  
3 3   M 64 1.74 verdi castani 41 1.63 1.80  
4 4   F 61 1.64 castani castani 39 1.65 1.78  
5 5   M 64 1.80 verdi castani 42 1.56 1.75  
6 6   F 51 1.68 castani castani 38 1.60 1.75  
> tabellaEH=table(studenti$Eyes,studenti$Hair)  
> chisq.test(tabellaEH)  
Pearson's Chi-squared test  
  
data: tabellaEH  
X-squared = 5.9614, df = 4, p-value = 0.202
```

L'intervallo di accettazione dell'ipotesi (che ricordiamo è l'indipendenza) al 95% di fiducia e 1 gradi di libertà è $[0, 3.841]$, il consuntivo cade dentro, per cui l'ipotesi è accettata. Per eliminare la correzione di Pearson si utilizza il parametro `correct=FALSE`. Ad esempio scriveremo:

```
> chisq.test(tabellaEH,correct=FALSE)
```

2.10.1 Test χ^2 di adeguamento

Consideriamo una variabile aleatoria discreta con frequenza assoluta delle uscite racchiuse in una lista `data`. Ci si pone il problema di stabilire se tali frequenze sono compatibili con le probabilità (riportate nella lista `p`).

```
> data<-c(2,3,4,5,6,7,8,9,10,11)
> prob<-c(5,20,5,10,5,15,5,10,10,15)
> sum(prob)
> chisq.test(data,p=prob,rescale.p=TRUE)
```

Si è usata qui la scelta `rescale.p=TRUE` in quanto la somma delle probabilità non era 1. L'uscita del test riporta il valore del consuntivo χ^2 i gradi di libertà ed il valore p .

2.11 Distribuzione Binomiale

Il coefficiente binomiale è definito come

$$\text{choose}(n, m) = \binom{n}{m} = \frac{n!}{m! \times (n - m)!}$$

Ad esempio

```
> choose(6,3)
[1] 20
```

La distribuzione binomiale in R ha la sintassi

`dbinom(successi, prove, probabilità successo)`

e fornisce la probabilità di ottenere nel corso di un certo numero di prove il numero di successi indicato. Ad esempio, nel lancio di un dado 10 volte, vogliamo determinare la probabilità che esca *esattamente* due volte il numero 4:

```
> dbinom(2,10,1/6)
[1] 0.29071
```

La probabilità è circa del 29%.

conta i caratteri del tuo nome

vettore che contenga il quadrato dei primi 8 numeri pari

vettore che contenga la radice cubica dei primi 10 numeri naturali.

inserire in una matrice le coordinate 2D dei punti di un pentagono. Plot dei punti, come punti, come linee e come linee tratteggiate. Più arduo: riempimento della superficie

derivata di $x^3 + x^2 + 2x + 1$, plot della funzione e plot della derivata come due pannelli distinti in un grafico unico. hint: par(mfrow)

Capitolo 3

Strutture di dati

3.1 Visualizzazione di oggetti di R

Per visualizzare un oggetto di R si può usare il comando `print` ma anche il comando `cat` che fornisce spesso un risultato migliore.

Per avere una stampa abbreviata

```
> head(iris)
   Sepal.Length Sepal.Width Petal.Length Petal.Width
1          5.1        3.5         1.4        0.2
2          4.9        3.0         1.4        0.2
3          4.7        3.2         1.3        0.2
4          4.6        3.1         1.5        0.2
5          5.0        3.6         1.4        0.2
6          5.4        3.9         1.7        0.4
   Species
1  setosa
2  setosa
3  setosa
4  setosa
5  setosa
6  setosa
> tail(iris)
   Sepal.Length Sepal.Width Petal.Length Petal.Width
145       6.7        3.3         5.7        2.5
146       6.7        3.0         5.2        2.3
147       6.3        2.5         5.0        1.9
```

```
148      6.5      3.0      5.2      2.0
149      6.2      3.4      5.4      2.3
150      5.9      3.0      5.1      1.8
      Species
145 virginica
146 virginica
147 virginica
148 virginica
149 virginica
150 virginica
```

Il comando **str** consente una visualizzazione parziale.

```
'data.frame':   150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",...: 1 1 1 1 1 1 1 1 1 1 ...
```

3.2 Le stringhe

3.2.1 Stringhe print, cat e caratteri speciali

Una stringa di testo è una collezione di caratteri; in genere, una stringa è resa riconoscibile dall'essere racchiusa tra virgolette. Oltre alle virgolette, vi sono numerosi altri caratteri speciali che possono apparire in una stringa. I più comuni sono “\t” per TAB, “\n” per una nuova linea e “\\” per un singolo *backslash*. Quest'ultimo carattere è un carattere di *escape* e consente una lettura diversa di quanto lo segue. Per esempio

```
> cat("\"sin\"")
"sin"
> nchar("\"sin\"")
[1] 5
> cat("\\")
\
> cat("ora a capo\nsono a capo?")
ora a capo
sono a capo?
> cat("ora spazio\\triprendo")
```

```
ora spazio      riprendo
```

La funzione `nchar`, che conta il numero di caratteri di una stringa, non includerà quindi il carattere di *escape* nel totale dei caratteri. Ad esempio:

```
> "Tab\t"  
[1] "Tab\t"  
> cat("Tab\t")  
Tab  
> nchar("Tab\t")  
[1] 4
```

3.2.2 Operare con le stringhe

Succede spesso di dover lavorare in modo automatico con stringhe di testo, anche nello scrivere indirizzi di rete o cartelle di lavoro. In R diversi comandi consentono la generazione, manipolazione e stampa di una o più stringhe di testo¹. Consideriamo inizialmente una singola frase.

```
> x="lavorare con le stringhe"
```

Possiamo verificarne la classe e determinare il numero di caratteri di `x`

```
> class(x)  
[1] "character"  
> nchar(x)  
[1] 24
```

e anche considerare sottostringhe

```
> substr(x,3,8)  
[1] "vorare"
```

o abbreviazioni ottenibili con il comando `abbreviate`

```
> abbreviate("Mario Rossi",4)  
Mario Rossi  
"MrRs"
```

¹Per un uso più specifico si può consultare il pacchetto `biostrings`.

Certi oggetti possono essere convertiti a stringhe: per esempio il numero 2 può essere visto come una stringa e riconvertito a numero.

```
> i=2;toString(i)
[1] "2"
> as.numeric(toString(i))
[1] 2
```

Alcune stringhe molto frequenti sono le lettere dell'alfabeto, maiuscole o minuscole

```
> letters[1:10]
[1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j"
> LETTERS[1:10]
[1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J"
```

o i mesi dell'anno (per esempio abbreviati in inglese)

```
> month.abb
[1] "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep"
[10] "Oct" "Nov" "Dec"
```

Le stringhe possono poi essere “incollate” con il comando

```
> paste("a","b",sep="")
```

dove `sep` indica il separatore usato. E tutto insieme

```
> for (i in 1:5) cat(paste("a",toString(i),"\t",sep=""))
a1      a2      a3      a4      a5
```

Il comando può anche essere utilizzato su vettori. Per esempio

```
> paste(letters[1:10],1:10,sep="")
[1] "a1"   "b2"   "c3"   "d4"   "e5"   "f6"   "g7"   "h8"   "i9"
[10] "j10"
```

La *recycling rule* continua a valere

```
> paste(letters[1:3],1:10,sep="")
```

```
[1] "a1"  "b2"  "c3"  "a4"  "b5"  "c6"  "a7"  "b8"  "c9"  
[10] "a10"  
> paste(letters[1:3], 1:12, sep = "")  
[1] "a1"  "b2"  "c3"  "a4"  "b5"  "c6"  "a7"  "b8"  "c9"  
[10] "a10" "b11" "c12"
```

e giocando con `rep` si possono ottenere diverse combinazioni.

```
> paste(rep(letters[1:3], each=5), 1:15, sep = "")  
[1] "a1"  "a2"  "a3"  "a4"  "a5"  "b6"  "b7"  "b8"  "b9"  
[10] "b10" "c11" "c12" "c13" "c14" "c15"  
> paste(rep(letters[1:3], ntimes=5), 1:15, sep = "")  
[1] "a1"  "b2"  "c3"  "a4"  "b5"  "c6"  "a7"  "b8"  "c9"  
[10] "a10" "b11" "c12" "a13" "b14" "c15"
```

Con l'opzione `collapse="x"` le stringhe vengono unite con separatore la stringa "x".

```
> paste(c("X", "Y"), 1:4, sep = "-", collapse = "--")  
[1] "X-1--Y-2--X-3--Y-4"
```

Si noti il separatore - dell'operazione `paste` e -- dell'operazione `collapse`.

1. Inserisci il tuo cognome in una variabile 'cognome' ed il tuo nome in una variabile 'nome'. Crea una terza variabile 'nomecognome' che contenga entrambi separati da un TAB. Stampa a console la scritta "Good job" seguita dal valore di nomecognome.
2. Creare un elenco che contenga mesi e anno dal 2001 al 2010 nel seguente formato "tre lettere iniziali del mese-anno".
3. Costruire una tabella che contenga tutte le parole di 2 lettere.
4. Si consideri

```
> paste(letters[1:7], 1:7, sep = "=")
```

Estendere la corrispondenza a tutto l'alfabeto.

-
5. Creare un elenco in cui a ciascun mese corrisponda il suo numero (a partire da gennaio).
 6. Creare un elenco con nomi i mesi e valori il numero di giorni di ciascun mese.
 7. Scrivere un elenco di 5 persone con le relative date di nascita nel formato anno-mese-giorno.

3.3 Matrici

Assegnati $n \times m$ ingressi possiamo costruire una matrice (ossia una tabella) con n righe e m colonne. Occorre solo riempire la matrice per righe o per colonne. Ad esempio:

```
> a<-matrix(letters[1:12],nrow=3,ncol=4)
> a
 [,1] [,2] [,3] [,4]
 [1,] "a"  "d"  "g"  "j"
 [2,] "b"  "e"  "h"  "k"
 [3,] "c"  "f"  "i"  "l"
> class(a)
[1] "matrix"
```

Se i parametri hanno natura diversa vengono resi uniformi

```
> a<-matrix(c(1:6,letters[1:6]),nrow=3,ncol=4);a
 [,1] [,2] [,3] [,4]
 [1,] "1"  "4"  "a"  "d"
 [2,] "2"  "5"  "b"  "e"
 [3,] "3"  "6"  "c"  "f"
```

Con il parametro `byrow=T` il riempimento avviene per righe, anzichè per colonne.

```
> a<-matrix(1:12,nrow=3,ncol=4,byrow=T)
> a
```

```
[,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12
```

Se i numeri sono insufficienti vengono *riciclati*

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

in modo pacifico se sono un sottomultiplo della dimensione della matrice o con qualche warning altrimenti.

```
> a<-matrix(2,nrow=3,ncol=4)
> a
```

Si possono anche definire gli ingressi attraverso opportune funzioni

```
> for(j in (1:4)) for(i in (1:3)) a[i,j]<-i^2+j
```

Per assegnare nomi alle righe e alle colonne:

```
colnames(matrice) = c(" nome1 ",
                      " nome2 ",..., " nomen ")
rownames(matrice) = c(" nome1 ",
                       " nome2 ",..., " nomen ")
```

```
> colnames(a)=c("c1","c2","c3")
> rownames(a)=c("r1","r2","r3")
> a
```

Aggiungere righe o colonne

Per aggiungere una o più righe (o colonne) ad una matrice si possono usare i comandi (**rbind** e **cbind**)

```
> dim(a)
[1] 3 4
> rbind(a,letters[1:ncol(a)])
 [,1] [,2] [,3] [,4]
[1,] "1"  "4"  "3"  "2"
[2,] "2"  "1"  "4"  "3"
[3,] "3"  "2"  "1"  "4"
[4,] "a"  "b"  "c"  "d"
> cbind(a,letters[1:nrow(a)])
 [,1] [,2] [,3] [,4] [,5]
[1,] "1"  "4"  "3"  "2"  "a"
[2,] "2"  "1"  "4"  "3"  "b"
[3,] "3"  "2"  "1"  "4"  "c"
```

Possiamo anche effettuare semplici operazioni, come somma degli elementi delle righe o delle colonne

```
> colSums(a)
[1] 6 7 8 9
> rowSums(a)
[1] 10 10 10
```

3.3.1 Operazioni con le matrici

Trasposizione

Per invertire righe e colonne di una matrice ossia per ottenere il trasposto di una matrice si usa il comando **t(matrice)**

```
> t(a)
 [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    1    2
[3,]    3    4    1
[4,]    2    3    4
```

Prodotto

Per la moltiplicazione di matrici (definita per ingressi numerici) si usa il simbolo `%*%`.

```
> b<-matrix(2,nrow=3,ncol=3)
> for(j in (1:3))
+ for(i in (1:3)) b[i,j]<-i+j+i^2
> b
[,1] [,2] [,3]
[1,]    3    4    5
[2,]    7    8    9
[3,]   13   14   15
```

Non è infatti possibile moltiplicare una matrice 3x4 con una 3x3. Possiamo però calcolare

```
> b%*%a
[,1] [,2] [,3] [,4]
[1,]   26   26   30   38
[2,]   50   54   62   74
[3,]   86   96  110  128
```

Determinante

Il determinante di una matrice quadrata si ottiene con il comando

$$\det(\text{matrice}) \quad (3.1)$$

```
> det(b)
[1] 1.887379e-14
```

Si noti che se eseguendo i calcoli a mano si trovano in alcuni casi risultati diversi da quelli di R. Per esempio la matrice in esame ha determinante 0 e 0 ne è anche un autovalore.

1. Creare una matrice 3×2 che abbia come ingressi i primi 6 numeri pari. Estendere la matrice aggiungendo due colonne contenenti i primi 6 numeri dispari. Calcolare e stampare la somma per riga e la somma per colonna. Modifica la matrice cambiando di segno la prima riga. Moltiplicare la matrice per 4.

3.4 I *dataframe*

I *dataframe* costituiscono in R la classe di oggetti fondamentali per la collezione di dati per una susseguente analisi statistica. Un *dataframe* è una collezione di vettori aventi egual lunghezza. È diverso da una matrice in quanto le colonne sono vettori eventualmente di tipi diversi. Il comando generale per costruire *dataframe* a partire da vettori o liste è `data.frame`. Esso richiede come parametri i nomi dei vettori (colonna) da affiancare nella tabella. In generale si scrive:

```
data.frame(vettore1, vettore2, ..., vettoren)
```

dove tutti i vettori hanno la stessa lunghezza. Si noti la asimmetria (rispetto ad una matrice) nel ruolo di righe e colonne. Le colonne sono omogenee, lo stesso non si può dire per le righe. Le colonne sono le variabili analizzate, le righe le unità statistiche. Anche i vari comandi che vedremo rispettano tale differenza.

Per esempio possiamo considerare il *dataframe* `d` definito come segue

```
> L3 <- LETTERS[1:3]
> d <- data.frame(cbind(x=1, y=1:10,
+ fac=sample(L3, 10, replace=TRUE)),
+ stringsAsFactors=TRUE)
> d
   x  y fac
1  1  1    C
2  1  2    B
3  1  3    B
4  1  4    B
5  1  5    C
6  1  6    B
7  1  7    A
```

```
8 1 8 A
9 1 9 C
10 1 10 A
```

Si noti che anche in questo caso si usa la *recycling rule*

```
> sito="http://www.pharm.unipmn.it/rinaldi/file_corso/"
> nomefile="australian-crabs.txt"
> indirizzo=paste(sito,indirizzo,sep="")
> read.table(indirizzo)->crabs
> dim(crabs)
```

```
[1] 200 8
```

In quanto segue lavoreremo con il seguente *dataframe* che rappresenta i risultati di un'indagine svolta sugli studenti che nell'Anno Accademico 2007/2008 frequentavano il primo anno del corso di Laurea di Farmacia della Facoltà di Farmacia del Piemonte Orientale. Potete scaricarlo dal sito e caricarlo in R con il comando

```
> sito="http://www.pharm.unipmn.it/rinaldi"
> nomefile="/file_corso/datifarmacia.txt"
> dati=read.table(paste(sito,nomefile,sep=""))
```

Il comando `colnames` consente di visualizzare o assegnare il nome alle colonne. Per esempio

```
> colnames(farmacia)
[1] "Sex"    "W"      "H"      "Eyes"   "Hair"   "Sh"     "hM"
[8] "hF"
```

Le varie colonne hanno dei nomi di facile interpretazione. Si noti anche che a fianco di variabili numeriche (W e H, peso-altezza ad esempio) sono presenti variabili *nominali* quali sesso (**Sex**) e colore degli occhi (**Eyes**). Per associare i nomi alle colonne alle varie colonne dobbiamo eseguire una operazione di collegamento con il comando `attach`,

```
> attach(farmacia)
```

A questo punto digitando il nome delle colonne appare il contenuto della colonna

```
> Sex
```

```
[1] F M M M F M F F M F F F F M F F F M M M F F F F  
[26] F M F M M F F M M F F F F M F M M M F F F F F F M  
[51] M M F F M  
Levels: F M
```

Le variabili nominali sono caratterizzate dal fatto che i loro valori (livelli, **levels**) non hanno significato numerico, anche se possono essere codificati con dei numeri. Ad esempio il sesso di una persona è una variabile nominale con due possibili valori, che sono stati indicati qui con la convenzione "F" per le femmine e "M" per i maschi. Se volessimo eliminare i livelli di una variabile nominale potremmo scrivere

```
> Sex=as.vector(Sex)  
> Sex  
[1] "F" "M" "M" "M" "F" "M" "F" "F" "M" "F" "F" "F" "F"  
[13] "F" "F" "M" "F" "F" "F" "M" "M" "M" "F" "F" "F" "F"  
[25] "F" "F" "M" "F" "M" "M" "F" "F" "M" "M" "M" "F" "F"  
[37] "F" "F" "M" "F" "M" "M" "F" "F" "F" "F" "F" "F" "F"  
[49] "F" "M" "M" "M" "F" "F" "M"  
  
> class(Sex)  
[1] "character"
```

Possiamo anche considerare il processo inverso e cambiare una variabile priva di livelli in una nominale

factor(*variable*) → *variable*

Per definire i suoi livelli (ad esempio n) scriveremo:

levels(*variable*) ← c(*nome₁*, *nome₂*, ..., *nome_n*)

Per rendere la variabile **Sex** nominale con nomi dei livelli F e M) scriveremo:

```
> Sex=factor(Sex)  
> Sex  
[1] F M M M F M F F F M F F F M M M F F F F  
[26] F M F M M F F M M F F F F M F M M M F F F F F M  
[51] M M F F M  
Levels: F M
```

Con il comando `detach` si può eliminare l'associazione creata tra colonne e nomi delle colonne. Consideriamo ora un *dataset* simile raccolto dagli studenti di Biotecnologie dello stesso anno

```
> sito="http://www.pharm.unipmn.it/rinaldi"
> nomefile="/file_corso/datibiotec.csv"
> biotec=read.table(paste(sito,nomefile,sep=""),sep=";",dec=",",header=T)
```

Scrivendo

```
> class(biotec)
[1] "data.frame"
```

vediamo che anche `biotec` è un *dataframe*. Inoltre confrontando i nomi delle colonne di `farmacia` e di `biotec` possiamo verificare che sono essenzialmente uguali a meno di traduzione e eventuale abbreviazione. Possiamo creare un *dataframe* unico che raggruppi `biotec` e `farmacia`. Per farlo vorremmo incollare un *dataframe* sopra all'altro. A tal fine occorre uniformare i nomi delle colonne scrivendo per esempio

```
> colnames(biotec)=colnames(farmacia)
> studenti=rbind(farmacia,biotec)
> head(studenti)

  Sex   W     H     Eyes    Hair Sh   hM   hF
1   F 62 1.75 CASTANI CASTANI 40 1.77 1.78
2   M 64 1.84 CASTANI CASTANI 43 1.72 1.80
3   M 80 1.70 CASTANI CASTANI 44 1.65 1.73
4   M 80 1.75 CASTANI      NERI 44 1.66 1.78
5   F 50 1.70 NOCCIOLA      NERI 38 1.65 1.79
6   M 63 1.88 CASTANI   BIONDI 44 1.58 1.75
```

Si noti che il comando `rbind` incolla per riga, mentre l'analogo comando `cbind` incolla le colonne. Le intestazioni di riga di dati sono

```
> rownames(studenti)
[1] "1"   "2"   "3"   "4"   "5"   "6"   "7"   "8"
[9] "9"   "10"  "11"  "12"  "13"  "14"  "15"  "16"
[17] "17"  "18"  "19"  "20"  "21"  "22"  "23"  "24"
[25] "25"  "26"  "27"  "28"  "29"  "30"  "31"  "32"
[33] "33"  "34"  "35"  "36"  "37"  "38"  "39"  "40"
[41] "41"  "42"  "43"  "44"  "45"  "46"  "47"  "48"
```

```
[49] "49"  "50"  "51"  "52"  "53"  "54"  "55"  "110"
[57] "210"  "310"  "410"  "56"  "61"  "71"  "81"  "91"
[65] "101"  "111"  "121"  "131"  "141"  "151"  "161"  "171"
[73] "181"  "191"  "201"  "211"  "221"  "231"  "241"  "251"
[81] "261"  "271"  "281"  "291"  "301"  "311"  "321"  "331"
[89] "341"  "351"  "361"  "371"  "381"  "391"  "401"  "411"
```

Per correggere la strana numerazione possiamo scrivere

```
> rownames(studenti)=seq(length=nrow(studenti))
```

Giunti a questo punto la tabella `dati` presenta ancora alcuni problemi; per esempio se scriviamo

```
> levels(studenti$Eyes)
[1] "AZZURRI"    "CASTANI"     "MARRONI"     "NERI"
[5] "NOCCIOLA"   "VERDI"       "azzurri"     "castani"
[9] "marroni"    "verdi"
> levels(studenti$Hair)
[1] "BIONDI"      "CASTANI"     "NERI"
[4] "biondi"      "castani"     "castano chiaro"
[7] "castano scuro" "neri"
```

Ci incuriosisce il dato con gli occhi neri. Verifichiamo:

```
> studenti[which(studenti$Eyes=="NERI"),]
  Sex  W  H Eyes Hair Sh  hM  hF
20  M 69 1.7 NERI NERI 41 1.55 1.75
```

Possiamo ritenere che sia un errore e che in realtà gli occhi siano marroni molto scuri. Risulta evidente che nel riportare i colori degli occhi si sono usate dizioni diverse per colori essenzialmente uguali, per esempio i livelli "CASTANI", "NOCCIOLA", "MARRONI" possono esser fatti confluire in un unico livello "castani" e possiamo rendere minuscoli i nomi degli altri livelli con il comando

```
> levels(studenti$Eyes)=c("azzurri","castani","castani", "castani", "castani")
```

A questo punto

```
> levels(studenti$Eyes)
[1] "azzurri" "castani" "verdi"
```

Facciamo lo stesso con i capelli

```
> levels(studenti$Hair)=c("biondi","castani","neri", "biondi", "castani","castani","ca  
> levels(studenti$Hair)  
[1] "biondi"  "castani" "neri"
```

Selezione in base a criteri

Supponiamo di voler selezionare gli studenti con gli occhi castani. Basta scrivere

```
> subset(studenti,studenti$Eyes=="verdi")  
   Sex   W     H Eyes    Hair Sh   hM   hF  
 7   F  63 1.70 verdi castani 38 1.72 1.82  
17  F  51 1.55 verdi castani 37 1.60 1.70  
25  F  91 1.81 verdi biondi 42 1.60 1.87  
33  M  75 1.82 verdi castani 43 1.60 1.75  
35  F  46 1.64 verdi castani 37 1.56 1.89  
37  F  56 1.70 verdi castani 39 1.68 1.90  
38  F  55 1.65 verdi castani 38 1.68 1.70  
41  M  56 1.70 verdi castani 39 1.65 1.80  
42  M  67 1.73 verdi castani 42 1.55 1.85  
47  F  52 1.75 verdi biondi 38 1.62 1.80  
51  M  75 1.76 verdi castani 42 1.60 1.68  
58  M  64 1.74 verdi castani 41 1.63 1.80  
60  M  64 1.80 verdi castani 42 1.56 1.75  
72  F  55 1.67 verdi castani 40 1.60 1.80  
76  M  85 1.84 verdi castani 43 1.69 1.69  
77  F  60 1.67 verdi castani 38 1.64 1.70  
81  F  62 1.61 verdi castani 39 1.60 1.66  
90  F  49 1.60 verdi castani 40 1.58 1.75  
91  F  62 1.76 verdi biondi 40 1.70 1.73  
93  F  53 1.65 verdi castani 38 1.55 1.85  
95  M  80 1.80 verdi castani 44 1.68 1.70
```

Se siamo invece interessati al colore dei capelli degli studenti con occhi castani

```
> subset(studenti,studenti$Eyes=="verdi",select="Hair")
```

```
      Hair
7  castani
17 castani
25 biondi
33 castani
35 castani
37 castani
38 castani
41 castani
42 castani
47 biondi
51 castani
58 castani
60 castani
72 castani
76 castani
77 castani
81 castani
90 castani
91 biondi
93 castani
95 castani
```

3.5 Gli *array*

Un *array* è una generalizzazione multidimensionale di una matrice. Gli *array* sono caratterizzati dal numero di dimensioni (se le dimensioni sono 2 un *array* si identifica con una matrice) e dal nome dei vari livelli

```
> array(LETTERS[1:24] ,
+ dim=c(2,3,4))
, , 1

 [,1] [,2] [,3]
[1,] "A"  "C"  "E"
[2,] "B"  "D"  "F"

, , 2
```

```
[,1] [,2] [,3]
[1,] "G"   "I"   "K"
[2,] "H"   "J"   "L"

, , 3

[,1] [,2] [,3]
[1,] "M"   "O"   "Q"
[2,] "N"   "P"   "R"

, , 4

[,1] [,2] [,3]
[1,] "S"   "U"   "W"
[2,] "T"   "V"   "X"
> array(sample(1:100,24), dim=c(3,4,2),dimnames=list(LETTERS[1:3],LETTERS[11:14],letter)
> x[,, "b"]
      K   L   M   N
A 26 90 83 23
B 97 67 79 95
C 93 84 49 54
```

3.6 Liste

Una lista (in R `list`) è un vettore di oggetti. Gli oggetti possono avere un nome ed avere natura diversa fra di loro. Per esempio

```
> x=list(a=month.abb , b=array(rep(0,20), dim=c(4,5)),c="your name")
> x
$a
[1] "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug"
[9] "Sep" "Oct" "Nov" "Dec"

$b
[,1] [,2] [,3] [,4] [,5]
[1,]    0    0    0    0    0
[2,]    0    0    0    0    0
[3,]    0    0    0    0    0
```

```
[4,]    0    0    0    0    0  
$c  
[1] "your name"
```

Possiamo annidare anche liste entro liste

```
> x=list(a=1:10,b=array(rep(0,20),dim=c(4,5)),  
+ c="testo",d=list(g="h",r=1:10) )  
> x  
$a  
[1] 1 2 3 4 5 6 7 8 9 10  
  
$b  
[,1] [,2] [,3] [,4] [,5]  
[1,] 0 0 0 0 0  
[2,] 0 0 0 0 0  
[3,] 0 0 0 0 0  
[4,] 0 0 0 0 0  
  
$c  
[1] "testo"  
  
$d  
$d$g  
[1] "h"  
  
$d$r  
[1] 1 2 3 4 5 6 7 8 9 10
```

3.6.1 Funzioni applicate a oggetti

La funzione apply

Quando abbiamo una struttura di dati a più dimensioni ad esempio un *array*

```
> array(1:30,dim=c(2,5,3))>x  
> x
```

```
, , 1  
[,1] [,2] [,3] [,4] [,5]  
[1,]    1     3     5     7     9  
[2,]    2     4     6     8    10  
  
, , 2  
[,1] [,2] [,3] [,4] [,5]  
[1,]   11    13    15    17    19  
[2,]   12    14    16    18    20  
  
, , 3  
[,1] [,2] [,3] [,4] [,5]  
[1,]   21    23    25    27    29  
[2,]   22    24    26    28    30
```

possiamo applicare una funzione selezionando il livello escluso:

```
> apply(x,2,mean)  
[1] 11.5 13.5 15.5 17.5 19.5  
> apply(x,c(1,3),mean)  
[,1] [,2] [,3]  
[1,]    5    15    25  
[2,]    6    16    26
```

In un *dataframe* possiamo considerare diverse operazioni che coinvolgano due o più colonne. Per esempio possiamo considerare la tabella **farmacia** e proporci di calcolare la media del peso dei maschi e la media del peso delle femmine. Il comando **tapply** consente di applicare una funzione ad una colonna di una tabella ripartita in gruppi in accordo ad un criterio specificato da un'altra colonna (nominale) della stessa tabella

tapply(vettore, criterio, funzione)

L'esempio sopra riportato si risolve con la scrittura:

```
> tapply(studenti$W, studenti$Sex,mean)->Wmean  
> Wmean
```

F	M
58.19298	69.34615

Possiamo anche chiedere di calcolare altri indicatori sempre basandoci su questa ripartizione maschi/femmine. Ad esempio possiamo ottenere la deviazione standard(**sd**) di maschi e femmine:

```
> tapply(studenti$W, studenti$Sex, sd) -> Wsd  
> Wsd
```

F	M
10.01149	10.61971

Il comando **table** applicato a diverse variabili nominali e le tabelle di contingenza

Consideriamo il seguente *dataframe* che riporta le ambizioni di un gruppo di scolari americani

```
> read.table("http://lib.stat.cmu.edu/DASL/Datafiles/PopularKids.html",  
+ skip=39, header=T, nrow=478, sep="\t") -> kidinterest
```

	Gender	Grade	Age	Race	Urban.Rural	School	Goals
1	boy	5	11	White	Rural	Elm	Sports
2	boy	5	10	White	Rural	Elm	Popular
3	girl	5	11	White	Rural	Elm	Popular
4	girl	5	11	White	Rural	Elm	Popular
5	girl	5	10	White	Rural	Elm	Popular
6	girl	5	11	White	Rural	Elm	Popular

	Grades	Sports	Looks	Money
1	1	2	4	3
2	2	1	4	3
3	4	3	1	2
4	2	3	4	1
5	4	2	1	3
6	4	2	1	3

Nella tabella le colonne che ci interessano al momento sono quelle che riguardano il sesso, gli obiettivi (scelti tra successo scolastico, capacità sportiva e popolarità) e la provenienza (colonne 1, 5 e 7). Nelle colonne dalla 8 alla 11 sono messi in ordine di importanza per il conseguimento della popolarità voti, sport, aspetto esteriore e denaro.

```
> colnames(kidinterest)
[1] "Gender"      "Grade"       "Age"
[4] "Race"        "Urban.Rural" "School"
[7] "Goals"        "Grades"      "Sports"
[10] "Looks"       "Money"

> interessi=kidinterest[,c(1,5,7)]
> head(intressi)

  Gender Urban.Rural   Goals
1    boy      Rural  Sports
2    boy      Rural Popular
3   girl      Rural Popular
4   girl      Rural Popular
5   girl      Rural Popular
6   girl      Rural Popular

> table(intressi)

, , Goals = Grades

  Urban.Rural
Gender Rural Suburban Urban
  boy     21      51     45
  girl    36      36     58

, , Goals = Popular

  Urban.Rural
Gender Rural Suburban Urban
  boy     19      20     11
  girl    31      22     38

, , Goals = Sports

  Urban.Rural
Gender Rural Suburban Urban
  boy     26      18     16
  girl    16       4     10
```

Consideriamo per esempio le variabili provenienza e traguardi

```
> interessi2=kidinterest[,c(5,7)]  
> tabella=table(interessei2)  
> tabella  
  
Goals  
Urban.Rural Grades Popular Sports  
Rural      57      50      42  
Suburban    87      42      22  
Urban       103     49      26  
  
> save.image(file="../Rdata/cap02.Rdata")
```

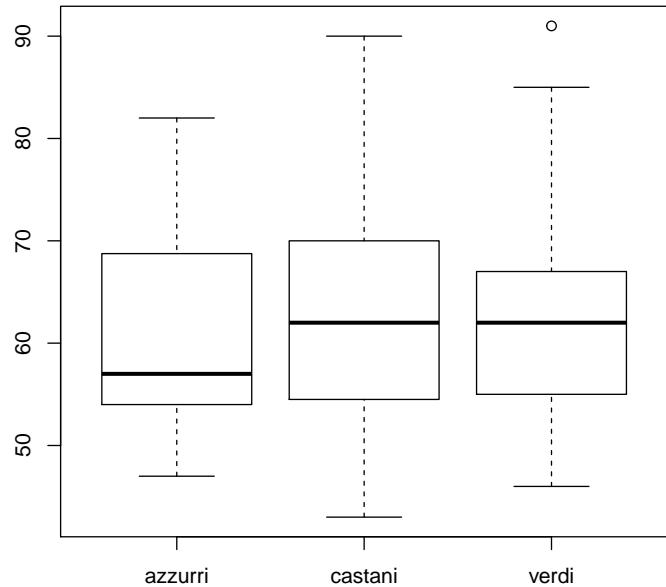
La tabella di contingenza delle due variabili (in questo caso a 3 livelli ciascuna) si realizza con il comando `table`. `table((lista))` applicato ad una singola variabile nominale conta le frequenze di ciascuna classe, se le variabili sono n esamina tutte le combinazioni possibili (2^n se le variabili sono dicotomiche) e ne conta tutte le occorrenze

Allo stesso modo possiamo creare una tabella dei colori degli occhi e dei capelli

```
> tabellaEH=table(studenti$Eyes,studenti$Hair)
```

O ripartirne il peso in base al colore degli occhi

```
> stEyeW=split(studenti$W,studenti$Eyes)  
> boxplot(stEyeW)
```



- Costruire una funzione `inserisci` tale che il comando `inserisci(x, dove, cosa)` inserisca nel vettore `x` l'elemento `cosa` nella posizione `dove`. Per esempio

```
> y <- letters[1:10]
> y
[1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j"

> inserisci(x, 5, 0)

[1] "a" "b" "c" "d" "0" "e" "f" "g" "h" "i" "j"
```

Generalizzare al caso di inserimento di righe in *dataframe*.

- Come si può verificare l'uguaglianza di 2 vettori includendo gli ingressi NA

> NA==NA

Aiuto: usare **is.na**

3. Carichiamo il *dataset* **juul** dal pacchetto **ISwR**. Ricavare il sottosinsieme di dati che corrispondono a ragazze tra i 7 e i 14 anni. Plottere **igf1** versus età per ragazzi e ragazze separatamente. Conclusioni?

Capitolo 4

Operatori logici e selezione di elementi

Consideriamo la selezione di elementi di un oggetto di R, per esempio `crabs`. Selezioniamo per esempio tra i primi 50 i granchi con carapaci di lunghezza (sesta colonna) maggiore di 42:

```
> crabs[1:50,6]>42
[1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[9] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[17] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[25] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[33] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[41] FALSE FALSE FALSE TRUE FALSE FALSE FALSE TRUE
[49] TRUE TRUE
```

Ciascun elemento del vettore viene confrontato con la soglia scelta (42) e quando supera la soglia la risposta è TRUE altrimenti FALSE.

operatore	significato
>	maggiori
>=	maggiori o uguali
<	minori
<=	minori o uguali
==	uguali
!=	diversi
&	and (e)
	or (o)

Gli ultimi due operatori combinano gli operatori precedenti. Per esempio

```
> crabs[crabs$sex == "F" | crabs$sp == "O",]  
> dim(crabs[crabs$sex == "F" & crabs$sp == "O",])
```

4.0.2 I valori non assegnati

Capita frequentemente che un oggetto contenga valori non assegnati. Per esempio

```
> x=1;x[10]=4;x  
[1] 1 NA NA NA NA NA NA NA NA 4  
  
> x[!is.na(x)]  
[1] 1 4  
> class(x)  
[1] "numeric"  
> class(unclass(x))  
[1] "numeric"  
> is.na(x)  
[1] FALSE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE  
[9] TRUE FALSE  
> DF <- data.frame(x = c(1, 2, 3), y = c(0, 10, NA))  
> na.omit(DF)  
   x  y  
1 1  0  
2 2 10  
> complete.cases(x)  
[1] TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
[9] FALSE  TRUE
```

4.1 Operatori condizionali e cicli

```
with(rainforest, table(complete.cases(root), species))
```

Gli operatori condizionali valutano una condizione e in base alle risultanze assegnano valore ad una espressione

```
> b=1
> if (b==1) {print ("b vale 1");b=b+1} else print(b);
[1] "b vale 1"
```

Abbiamo poi i cicli **for** con la struttura

for(*i*itsequenza)esegui*B*

```
> for (i in 1:10) print(i)#Somma di interi
> somma<-0
> for (i in 1:10) somma<-somma+i;somma
> #o anche come funzione
> somma<-function(n)
+ {somma<-0
+ for (i in 1:n) somma<-somma+i;somma
+ return(somma)}
```

4.2 Funzioni di più variabili

Finora abbiamo considerato funzioni di singola variabile. Una generalizzazione estremamente naturale consiste nel considerare funzioni di due o più variabili. Diamo alcuni esempi

1. Calcolare la lunghezza del segmento che congiunge due punti nel piano (distanza euclidea). Possiamo procedere con una funzione di 4 numeri (le 2 coordinate dei 2 punti)

```
> dist<-function(x1,y1,x2,y2)
+ {dx<-(x2-x1)^2;dy<-(y2-y1)^2;return (sqrt(dx+dy))}
> dist(0,0,3,4)
[1] 5
```

oppure possiamo generalizzare con una funzione che dipenda dai 2 vettori (e non direttamente dalle loro coordinate)

```
> dist<-function(x,y)
+ {return (sqrt(sum((x-y)^2)))}
```

2. : il massimo di due numeri

```
> massimo<-function(a,b)
+ if (a>b) return(a) else return(b)
```

Si noti che la condizione viene messa tra parentesi tonde.

3. : Il valore assoluto

```
> valoreassoluto<-function(a) if (a>0) return(a) else return(-a)
```

4. Somma sino a n

```
> n=0;while (somma(n)<10000)
> n=n+1
> sommasinoa<-function(b) {i<-1;somma<-i;while (somma<=b)
+ {i=i+1;somma=somma+i}; return(i)}
```

Generazione ricorsiva di successioni (successione di Fibonacci)

```
> fibonacci<-function(n)
+ {f=0;f[1]<-1;f[2]<-1;for (i in 3:n) {f[i]<-f[i-1]+f[i-2]};return(f[n])}
```

4.3 R Output su file

```
> cat("TITLE extra line", "2 4 5 7", "11 13 17", file="ex.data", sep="\n", append=TRUE)
> cat("TITLE extra line", "2 4 5 7", "11 13 17", sep="\n")

> print()
> read.delim
> aggregate
>
```

Capitolo 5

Statistica II parte

Uno degli usi rilevanti a fini statistici dei computer è la possibilità di stimare attraverso simulazioni probabilità difficili da calcolare teoricamente.

Estrazioni da un'urna. Supponiamo 2 palle siano estratte da un'urna contenente 6 palle rosse e 2 verdi. Simuliamo due estrazioni e consideriamo gli eventi:

- A = la prima palla è rossa
- B = la seconda palla è rossa

Determinare le probabilità $P(A)$, $P(B)$, $P(A \wedge B)$, $P(A | B)$, $P(B | A)$. Eseguire poi la simulazione 1000 volte e verificare che le stime frequentiste si avvicinano ai valori teorici. Si usi la funzione `replicates`. Gli ultimi due punti richiedono opportuni accorgimenti.

Simulazione del problema di Monty Hall. Il problema di Monty Hall ricalca una trasmissione televisiva americana. Immaginate di avere tre porte chiuse dietro alle quali si nascondono 2 capre e un premio, una Ferrari per esempio. L'ospite deve scegliere una porta; effettuata tale scelta il presentatore (che conosce cosa si nasconde dietro alle 3 porte) apre una porta mostrando una capra e dà all'ospite la scelta se attenersi alla scelta originale o cambiare porta. Cosa fareste se foste voi l'ospite?

Come si può simulare il problema di Monthly Hall? Iniziamo a disporre gli oggetti e a fare la nostra scelta

```
> set.seed(12)
> (sample(c("capra","capra","ferrari"))->posizione)
[1] "capra"   "capra"   "ferrari"
> (scelta<-sample(1:3,1))
[1] 1
```

A questo punto il presentatore deve aprire una porta,

```
> (ammessa<-(1:3)[-unique(c(scelta,which(posizione=="ferrari")))])
[1] 2
> if(length(ammessa)==1)
+ presentatore<-ammessa else presentatore=sample(ammessa,1)
```

Ripetiamolo ora 1000 volte

```
> replicate(1000,{
+ sample(c("capra","capra","ferrari"))->dietro;
+ scelta=sample(1:3,1);
+ ammessi=(1:3)[-unique(c(scelta,which(
+ dietro=="ferrari")))];
+ if(length(ammessi)==1) {presentatore=ammessi} else {presentatore=sample(ammessi,1)};
+ c(dietro[scelta],ditero[-c(scelta,presentatore)])})->risultato
> table(risultato[1,]);table(risultato[2,]);
>
```

5.1 Analisi della Varianza (ANOVA)

L'analisi della varianza diviene estremamente semplice da eseguire con R a patto di *preparare* i dati in modo opportuno. Consideriamo il seguente esempio tratto dal libro di Snedecor in cui si considera l'assorbimento di grasso durante la cottura di 24 *donuts*. Si esaminano 4 tipi di grasso di cottura 4 e la cottura di 6 *donuts* per ciascun tipo. Inseriamo nella lista *x* il gruppo di appartenenza e nella lista *y* le quantità di grasso assorbito.

```
> x<-rep(1:4,each=6)
> factor(x)->x
> x<-gl(4,6);x
```

```
[1] 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3 4 4 4 4 4 4  
Levels: 1 2 3 4  
> y<-c(15,20,22,23,25,26,80,90,76,56,5,  
+ 43,23,45,67,89,87,65,43,23,45,10,11,23)  
> tabelladati=data.frame(y,x)  
> tabelladati  
  
   y x  
1 15 1  
2 20 1  
3 22 1  
4 23 1  
5 25 1  
6 26 1  
7 80 2  
8 90 2  
9 76 2  
10 56 2  
11 5 2  
12 43 2  
13 23 3  
14 45 3  
15 67 3  
16 89 3  
17 87 3  
18 65 3  
19 43 4  
20 23 4  
21 45 4  
22 10 4  
23 11 4  
24 23 4
```

Il comando `tapply` consente di applicare una funzione (in questo caso la media) alla lista `y` in base alla ripartizione indicata da `x`.

`tapply(y,x,mean)`

In questo caso

```
> tapply(y,x,mean)
```

```
1          2          3          4  
21.83333 58.33333 62.66667 25.83333
```

In modo simile all'uso del simbolo \sim visto nei comandi per la regressione lineare il comando

```
boxplot( $y \sim x$ , data = tabella)
```

consente il tracciamento di un *boxplot* comparativo.

```
> boxplot(y~x,data=tabelladati)
```

Figura 5.1: Boxplot comparativo di 4 tipi di grasso di cottura.

In modo analogo `echo=TRUE` = `anova(lm(y ~ x))` fornisce il risultato dell'analisi della varianza.

5.2 Test χ^2 di indipendenza

```
> read.table("../filedati/PopularKids.html", skip=39, header=T, nrow=478, sep="\t")
```

Il test χ^2 di indipendenza consente di verificare se due variabili sono indipendenti. Se consideriamo le due variabili precedenti sesso e interessi. R dispone del comando `chisq.test`, dalla sintassi generale:

```
chisq.test(tabella)
```

Nell'esempio

```
> load("../Rdata/cap02.Rdata")  
> tabellaEH=table(studenti$Eyes,studenti$Hair)  
> chisq.test(tabellaEH)  
Pearson's Chi-squared test  
  
data: tabellaEH  
X-squared = 5.9614, df = 4, p-value = 0.202
```

L'intervallo di accettazione dell'ipotesi (che ricordiamo è l'indipendenza) al 95% di fiducia e 1 gradi di libertà è [0, 3.841], il consuntivo cade dentro, per cui l'ipotesi è accettata. Per eliminare la correzione di Pearson si utilizza il parametro `correct=FALSE`. Ad esempio scriveremo:

```
> chisq.test(tabellaEH,correct=FALSE)
```

5.2.1 Test χ^2 di adeguamento

Consideriamo una variabile aleatoria discreta con frequenza assoluta delle uscite racchiuse in una lista `data`. Ci si pone il problema di stabilire se tali frequenze sono compatibili con le probabilità (riportate nella lista `p`).

```
> data<-c(2,3,4,5,6,7,8,9,10,11)
> prob<-c(5,20,5,10,5,15,5,10,10,15)
> sum(prob)
> chisq.test(data,p=prob,rescale.p=TRUE)
```

Si è usata qui la scelta `rescale.p=TRUE` in quanto la somma delle probabilità non era 1. L'uscita del test riporta il valore del consuntivo χ^2 i gradi di libertà ed il valore p .

5.3 Distribuzione Binomiale

Il coefficiente binomiale è definito come

$$\text{choose}(n, m) = \binom{n}{m} = \frac{n!}{m! \times (n - m)!}$$

Ad esempio

```
> choose(6,3)
[1] 20
```

La distribuzione binomiale in R ha la sintassi

`dbinom(successi, prove, probabilità successo)`

e fornisce la probabilità di ottenere nel corso di un certo numero di prove il numero di successi indicato. Ad esempio, nel lancio di un dado 10 volte, vogliamo determinare la probabilità che esca *esattamente* due volte il numero 4:

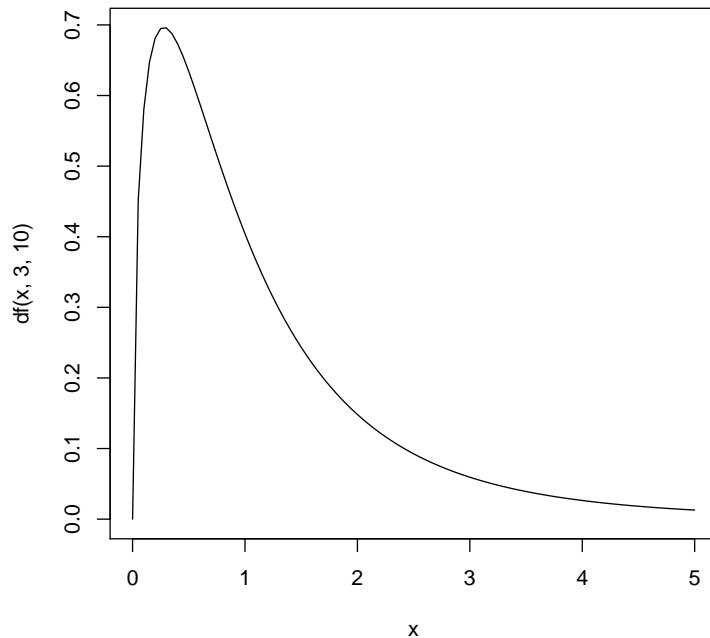
```
> dbinom(2,10,1/6)
[1] 0.29071
```

La probabilità è circa del 29%.

5.4 Distribuzione di Fisher

La distribuzione di Fisher è indicata in R con la lettera **f**. Per tracciare il grafico della densità con gradi di libertà ν_1 e ν_2 basta scrivere

```
> curve(df(x,3,10),0,5)
```



ottenendo il grafico .

Dobbiamo solo definire x , $df1$ e $df2$, gradi di libertà per poi applicare **df**, **qf**, **rf** come visto per le distribuzioni precedenti. Per esempio per ottenere il valore della funzione inversa per $x = 0.9$ e $df1 = 3$ e $df2 = 4$ scriveremo:

```
> qf(0.9,3,4)
[1] 4.19086
```

5.5 Tabelle di contingenza

Il comando **table** applicato ad un lista contenente valori di una singola variabile nominale conta le frequenze di ciascuna livello, se applicato ad un

```
> curve(df(x,3,10),0,5)
```

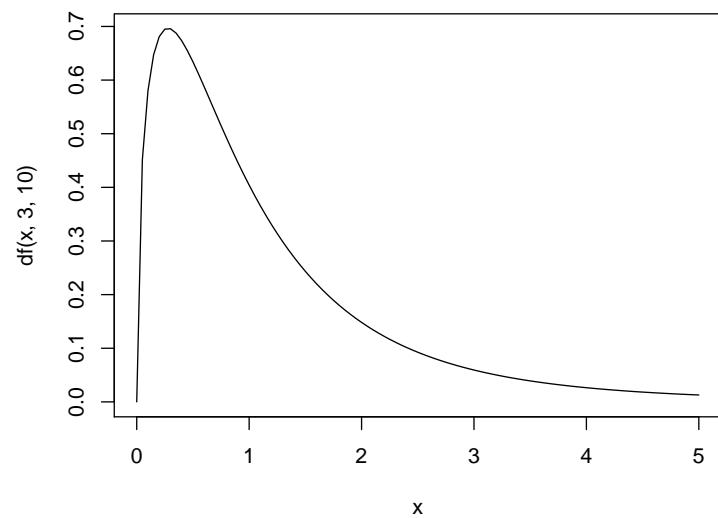


Figura 5.2: Distribuzione di Fisher.

dataframe di n variabili conta le occorrenze di ciascuna combinazione di livelli possibile (2^n se le variabili sono dicotomiche). Per esempio se consideriamo il *dataframe* `studenti` possiamo creare una tabella dei colori degli occhi e dei capelli

```
> tabellaEH=table(studenti$Eyes,studenti$Hair)
```

5.6 Farmacocinetica e modelli non lineari

Dopo la somministrazione, un farmaco viene distribuito alle regioni del corpo accessibili. Se consideriamo questo evento istantaneo, il corpo può qui essere considerato un contenitore omogeneo del farmaco, e la cinetica di diffusione può essere modellata con un modello a singolo compartimento aperto. Quest'ultimo termine significa che il contenuto del compartimento non è confinato nel compartimento. Se invece la diffusione non è istantanea e il contenuto arriva al siero secondo un modello di decrescita malthusiana, il sistema può essere modellato come un sistema a due compartimenti aperti.

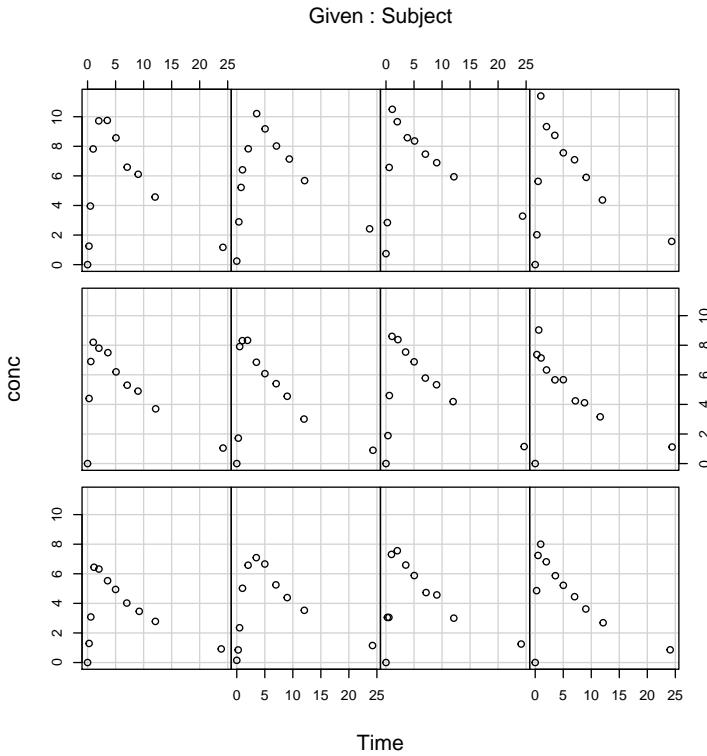
Il dataset `Theoph` contiene 11 misure di concentrazione (a diversa temporistica) per ciascuno dei 12 soggetti sottoposti a somministrazione orale di teofillina.

```
> head(Theoph)
```

	Subject	Wt	Dose	Time	conc
1	1	79.6	4.02	0.00	0.74
2	1	79.6	4.02	0.25	2.84
3	1	79.6	4.02	0.57	6.57
4	1	79.6	4.02	1.12	10.50
5	1	79.6	4.02	2.02	9.66
6	1	79.6	4.02	3.82	8.58

Diamo una prima occhiata ai dati con il comando `coplot`, ovvero *conditioning plot*. Il comando richiede un modello/formula, del tipo $(xy|a)$, che significa x versus y dato a (ecco perch? conditioning). Se le variabili rispetto alle quali riportare il grafico sono più di una, basta aggiungere tra esse un segno +.

```
> require(graphics)
> coplot(conc ~ Time | Subject, data = Theoph, show.given = FALSE)
```



Selezioniamo ora il paziente 4, lo possiamo fare note le righe che esso occupa nel dataframe oppure selezionando per un preciso valore della variabile Subject.

```
> Theoph.4 <- subset(Theoph, Subject == 4)
```

5.6.1 Regressione non lineare

In statistica, una regressione non lineare è una forma di regressione in cui i dati osservati sono modellati da una funzione che non è una combinazione lineare dei parametri e che dipende da una o più variabili indipendenti.

Il comando `nls` determina le stime dei parametri di un modello non lineare con il metodo dei minimi quadrati (ponderati). Se consideriamo un modello a 2 compartimenti aperti del primo ordine descritto da sistema di equazioni differenziali

$$\begin{cases} x'(t) = -k_a x(t) \\ y'(t) = k_a x(t) - k_e y(t) \end{cases} \quad (5.1)$$

troviamo la soluzione

$$\begin{cases} x(t) = x_0 e^{-k_a t} \\ y(t) = x_0 k_a \frac{e^{-k_e t} - e^{-k_a t}}{k_a - k_e} \end{cases} \quad (5.2)$$

In particolare la funzione `SSfol` in R consente di descrivere una cinetica del primo ordine per il modello a proposto. Essa infatti restituisce i logaritmi dei parametri di eliminazione, assorbimento e *clearance* (`lKe`, `lKa`, `lCl`). L'uso è il seguente

```
SSfol(Dose, input, lKe, lKa, lCl)
```

dove `Dose` rappresenta la dose iniziale, `input` è un vettore numerico con il quale valutiamo il modello. `lKe` è un parametro numerico che rappresenta il logaritmo naturale della costante velocità di eliminazione. `lKa` è un parametro numerico che rappresenta il logaritmo naturale della costante velocità di assorbimento mentre `lCl` è un parametro numerico che rappresenta il logaritmo naturale della *clearance*. Proviamo ad utilizzare il comando `SSfol`, e inseriamo il risultato della regressione nella variabile `fm1` (modello 1).

```
> fm1 <- nls(conc ~ SSfol(Dose, Time, lKe, lKa, lCl), data = Theoph.4)
```

Osserviamo `fm1`, con il comando `summary`

```
> summary(fm1)
```

```
Formula: conc ~ SSfol(Dose, Time, lKe, lKa, lCl)
```

Parameters:

	Estimate	Std. Error	t value	Pr(> t)
<code>lKe</code>	-2.4365	0.2257	-10.797	4.77e-06 ***
<code>lKa</code>	0.1583	0.2297	0.689	0.51
<code>lCl</code>	-3.2861	0.1448	-22.695	1.51e-08 ***

Signif. codes:

```
0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

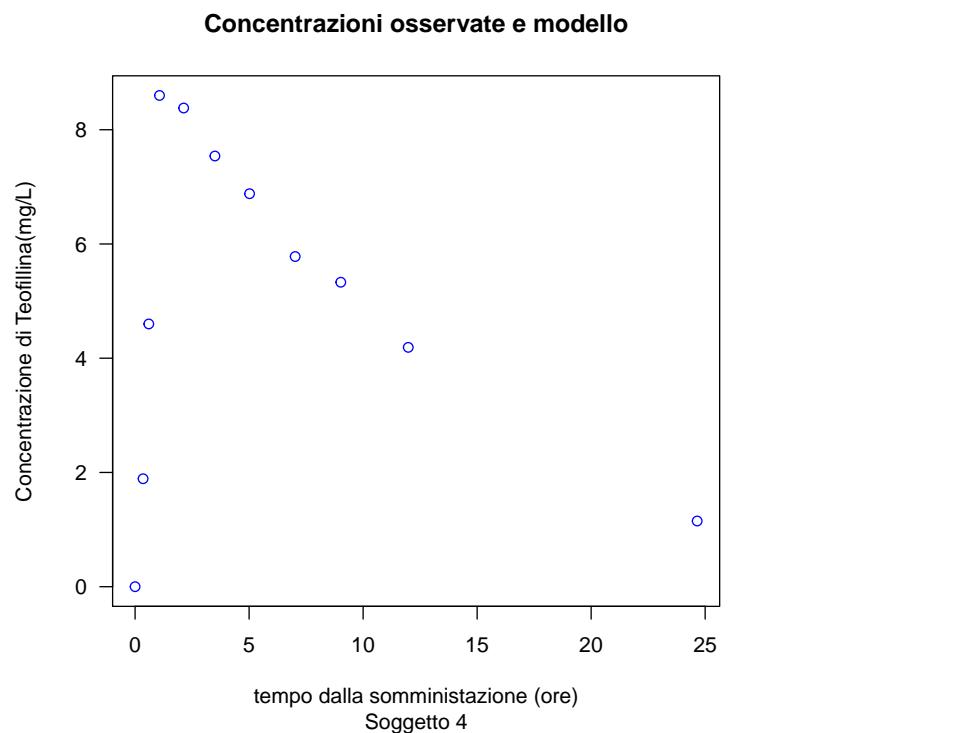
Residual standard error: 0.8465 on 8 degrees of freedom

Number of iterations to convergence: 7

Achieved convergence tolerance: 2.969e-06

Ora riportiamo il grafico delle osservazioni per il soggetto 4 su un grafico singolo:

```
> options(width=40)
> plot(conc ~ Time, data = Theoph.4,
+ xlab = "tempo dalla somministrazione (ore)",
+ ylab = "Concentrazione di Teofillina(mg/L)",
+ main = "Concentrazioni osservate e modello", sub = "Soggetto 4",
+ las = 1, col = 4)
```



Si può ora sovrapporre la curva ottenuta con il modello al grafico attuale.
Per prima cosa definiamo i possibili valori di x .

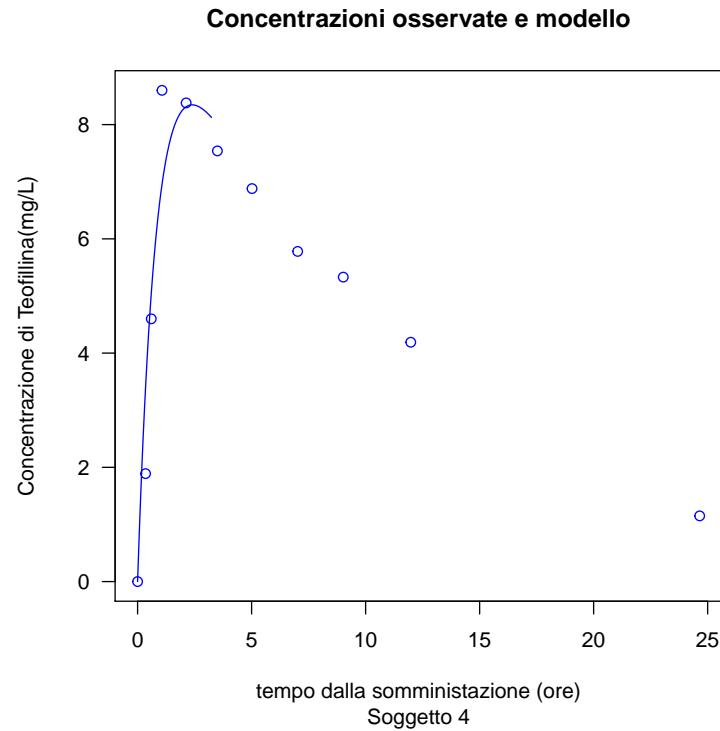
```
> xvals <- seq(0, par("usr")[2], length.out = 55)
```

dove `par(usr)` è un ottimo modo per conoscere l'estensione di una delle dimensioni della finestra grafica. A tali valori di x vengono associati i corrispondenti valori y della curva, ottenuti con il comando `predict`, che utilizza diversi metodi predittivi noti i valori delle costanti del modello ed i valori di x . In questo caso il tempo (la nostra x) è uguale ai valori `xvars` appena calcolati.

```

> plot(conc ~ Time, data = Theoph.4,
+ xlab = "tempo dalla somministrazione (ore)",
+ ylab = "Concentrazione di Teofillina(mg/L)",
+ main = "Concentrazioni osservate e modello", sub = "Soggetto 4",
+ las = 1, col = 4)
> lines(xvals, predict(fm1, newdata = list(Time = xvals)), col = 4)

```



5.6.2 I bambini di Kalama (Egitto). Ancora retta di regressione

Sempre da DASL possiamo scaricare un *dataset* in cui dei ricercatori hanno misurato le altezze (cm) dai 18 ai 29 mesi di vita, di 161 bambini di Kalama, un villaggio egiziano. Le altezze sono state mediate tra i bambini per fornire un singolo valore mese per mese.

```

> age=18:29
> height=c(76.1,77,78.1,78.2,78.8,79.7,79.9,81.1,81.2,81.8,82.8,83.5)

```

Possiamo quindi costruire il `data.frame`

```
> village=data.frame(age=age,height=height)
```

Ora diamo una prima occhiata ai dati: L'andamento è lineare. Determiniamo:

```
> plot(age,height)
```

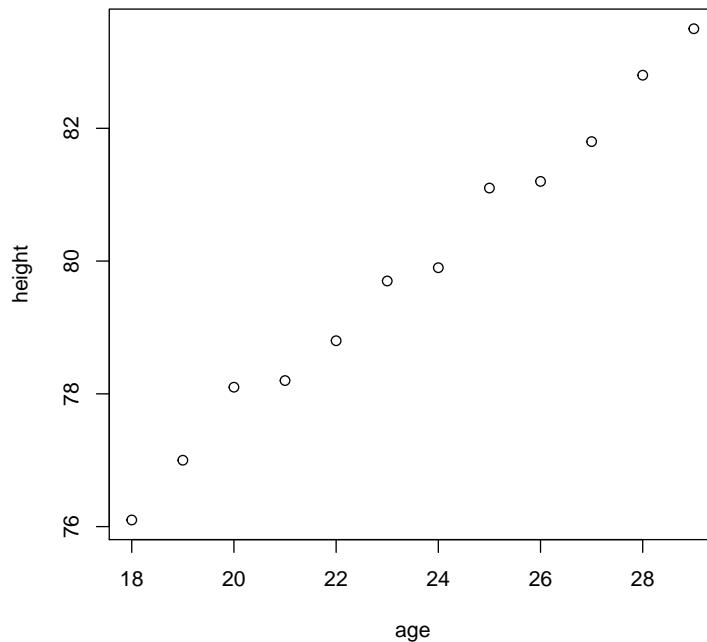


Figura 5.3: I bambini di Kalama

niamo la retta di regressione per predire l'altezza media nota l'età in mesi.

```
> ris=lm(height~age)
> ris
Call:
lm(formula = height ~ age)
```

Coefficients:

```
(Intercept)      age
       64.928     0.635
> plot(ris)
```

La retta di regressione cercata ha formula:

$$h(\text{age}) = 64.93 + 0.63 \text{ age}$$

Possiamo ora utilizzare R come semplice calcolatore per predire l'altezza a 27.5 mesi di età: oppure, è più efficiente utilizzare direttamente il *dataframe* e la funzione *predict*:

```
> predict(ris,data.frame(age=27.5))
   1
82.38986
```

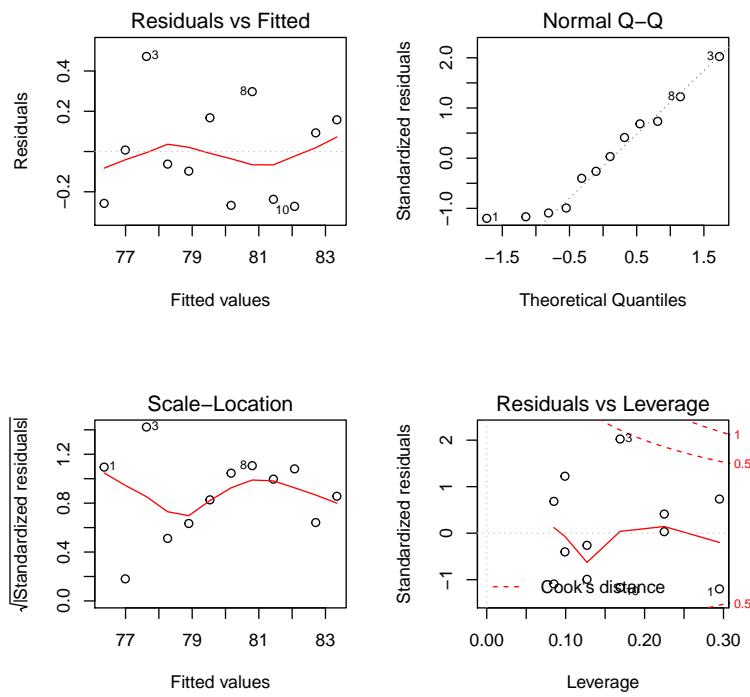
fornendo in input i parametri della retta ed un preciso valore della variabile indipendente (richiamata col proprio nome). Molti comandi di R sono in grado di manipolare dataframe lavorando direttamente sulla struttura. Per esempio, il comando *plot* di un *dataframe* in due colonne, esegue in automatico il grafico della seconda colonna (variabile dipendente) vs prima colonna (variabile indipendente). Possiamo ottenere il modello lineare visto nel caso precedente, passando *village* direttamente al comando:

```
> res=lm(height~age,data=village)
> res
Call:
lm(formula = height ~ age, data = village)

Coefficients:
(Intercept)      age
       64.928     0.635
```

con la formula $lm(y \sim x, data = dataset)$. Inoltre possiamo considerare il *plot* di un oggetto *lm* che fornisce una serie di rappresentazioni grafiche

```
> oldpar<-par(mfrow=c(2,2))
> par(ask=FALSE);plot(lm(height~age))
> oldpar
$mfrow
[1] 1 1
```



5.7 QQPlot normality test

27 ottobre 2014

Capitolo 6

Grafica

Il sistema grafico di R si compone di 3 parti: grafica di alto livello che produce grafici completi (per esempio `plot`, `curve`), grafica di basso livello che aggiunge elementi a grafici preesistenti (`text`, `abline`) e funzioni che lavorano interattivamente con grafici (ad esempio `locator()` e `identify()`). [Murrell]. A fianco della grafica tradizionale (gestita dal pacchetto `graphics`) esistono sistemi non tradizionali cui si può accedere attraverso il pacchetto `grid`. In questo capitolo ci dedicheremo essenzialmente ad inquadrare la grafica tradizionale e daremo alcuni cenni sulla grafica non tradizionale.

6.1 Le *device* grafiche

Le *device* grafiche consentono di realizzare grafici di R su file di diversi formati (per esempio `.pdf`, `.ps`, `.bmp`, `.jpeg`, `.tiff`). Il grafico anziché essere visualizzato a schermo, può quindi essere salvato direttamente in diversi formati. Per esempio il comando

```
> pdf(file = "miografico.pdf")
> plot(1:10)
> dev.off()
```

genera un file `.pdf` di nome `miografico` nella *working directory*. Se omettiamo il nome avremo il nome di defaults `Rplots`. Per particolari *device* (per esempio `pdf`) la stampa su *file* può occupare diverse pagine e quindi si possono inserire in un unico file diversi grafici. Occorre ricordarsi di chiudere la *device* al termine della produzione grafica con il comando `dev.off()`. Il grafico sarà direttamente prodotto nel file indicato, dunque non si vedranno

finestre grafiche e apparirà a schermo solo un messaggio di avvenuta realizzazione del file una volta che la *device* viene chiusa. La sintassi generica richiede dunque

1. Apertura della *device* (definendone il nome, la dimensione ed altri parametri non obbligatori, come la risoluzione)
2. Comandi grafici.
3. Chiusura della *device*.

Nota: le *device* possono presentare piccole variazioni nei parametri, in particolare nei loro nomi. Un esempio: il nome del file in `png` è definito dal parametro `filename=`, mentre nella device `pdf` è definito dal parametro `file=`. Consultare l'`help` prima di utilizzarle non è una cattiva idea. Tutti i parametri grafici visti valgono allo stesso modo per le *device*.

6.2 Lo stato grafico

Ogni *device* grafica mantiene uno stato grafico che viene consultato da R ogni volta che la *device* viene usata. Lo stato grafico consiste in una serie di parametri specificabili con il comando `par`, nella forma

`par(parametro1 = val1, ...)`

Tecnicamente `par()` è una funzione che restituisce una lista. I parametri grafici specificabili con `par` sono ottenibili con il comando

```
> options(width=55)
> class(par())
[1] "list"
> names(par())
[1] "xlog"      "ylog"      "adj"       "ann"
[5] "ask"       "bg"        "bty"       "cex"
[9] "cex.axis"   "cex.lab"    "cex.main"  "cex.sub"
[13] "cin"        "col"        "col.axis"  "col.lab"
[17] "col.main"   "col.sub"    "cra"       "crt"
[21] "csi"        "cxy"        "din"       "err"
[25] "family"     "fg"         "fig"       "fin"
[29] "font"       "font.axis"  "font.lab"  "font.main"
[33] "font.sub"   "lab"        "las"       "lend"
```

```
[37] "lheight"    "ljoin"      "lmitre"     "lty"
[41] "lwd"         "mai"        "mar"        "mex"
[45] "mfcol"       "mfg"        "mfrow"      "mgp"
[49] "mkh"         "new"        "oma"        "omd"
[53] "omi"         "page"       "pch"        "pin"
[57] "plt"         "ps"         "pty"        "smo"
[61] "srt"         "tck"        "tcl"        "usr"
[65] "xaxp"        "xaxs"       "xaxt"       "xpd"
[69] "yaxp"        "yaxs"       "yaxt"       "ylbias"
```

Per verificare lo stato grafico corrente basta scrivere `par()`. Per conoscere lo stato grafico di un particolare parametro basta scrivere `par("nome parametro")`. Ad esempio il parametro `bg` si riferisce al colore dello sfondo del grafico di default trasparente, mentre il parametro `lwd` si riferisce allo spessore delle linee. Per cambiare i default dei due parametri

```
> oldpar<-par(bg="seashell1","lwd=5")
> plot(sin)
```

Qui contestualmente alla ridefinizione abbiamo salvato (con il nome `oldpar`) la versione corrente dei parametri per poterli eventualmente ripristinare in seguito. ottenendo la figura 6.1. Si noti che si possono riesumare i valori precedenti

```
> par("bg")
[1] "seashell"
> par(oldpar)
> par("bg")
[1] "transparent"
```

6.3 Elementi grafici di base

6.3.1 Costruzione di grafici multipli

In questo capitolo disegneremo diversi grafici; consideriamo preliminarmente la possibilità di affiancarli in riga e in colonna, sia per ragioni di spazio che di chiarezza. Per costruire tabelle di grafici in una finestra unica si usa il comando

```
par(mfrow=c(nrigh, ncolonne))
```

Per esempio con una riga e due colonne

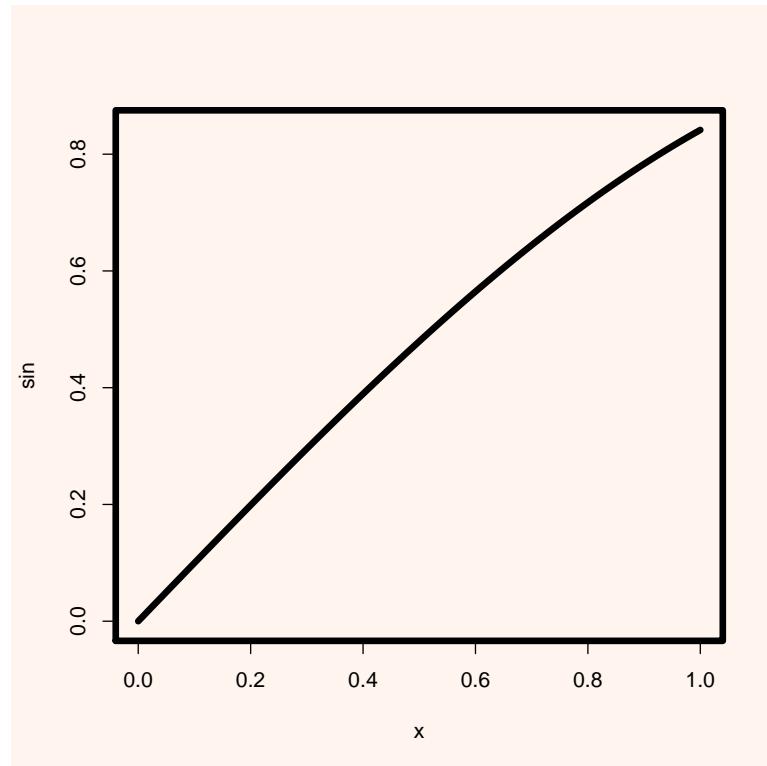
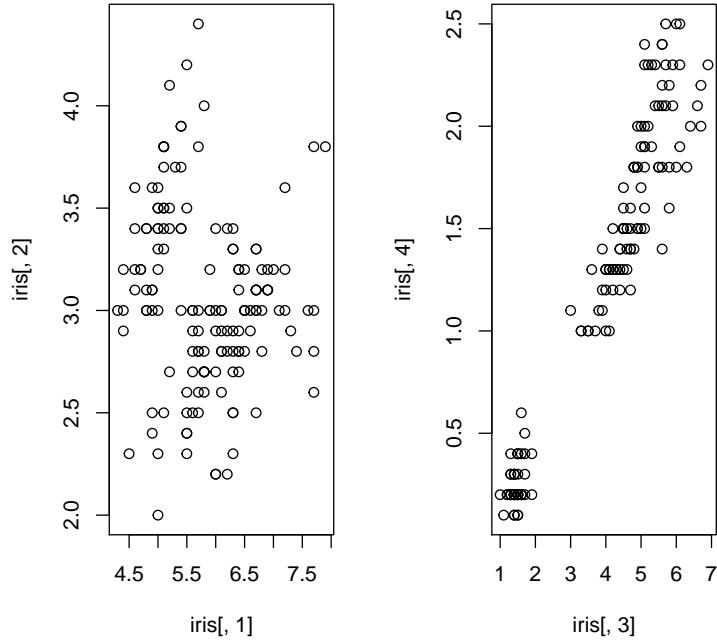


Figura 6.1: esempio di utilizzo di `par()`

Figura 6.2: Grafici affiancati con `mfrow=c(1,2)`

```
> par(mfrow=c(1,2))
> plot(iris[,1],iris[,2])
> plot(iris[,3],iris[,4])
```

otteniamo i grafici affiancati della figura 6.2.

6.3.2 Diagrammi a dispersione

La sintassi di riferimento della funzione `plot` è

$$\text{plot}(\text{dati}, \text{opzione}_1 = \text{val}_1, \text{opzione}_2 = \text{val}_2, \dots, \text{opzione}_n = \text{val}_n)$$

Scarichiamo da [il dataset](#) che chiameremo **Gompertz** in cui si studia la mortalità di una popolazione di moscerini al variare dell'età

```
> url="http://lib.stat.cmu.edu/DASL/Datafiles/Medflies.html"
> gompertz<-read.table(url,skip=25,nrow=173,header=T)

> class(gompertz)
[1] "data.frame"
> head(gompertz)
  day living mort.rate
1   0 1203646         0
2   1 1203646     0.0014
3   2 1201913     0.0040
4   3 1197098     0.0051
5   4 1191020     0.0064
6   5 1183419     0.0075

> par(mfrow=c(1,2))
> plot(gompertz[,1],gompertz[,2])
> plot(gompertz[,1],gompertz[,3])
```

Il tasso di mortalità (`mort.rate`) all'istante t_1 è definito come rapporto tra numero di individui deceduti nell'intervallo di tempo (t_1, t_1+1) e numero di individui all'istante t_1 . Il comando `diff` applicato ad una lista ne calcola le differenze termine a termine. Per esempio

```
> diff(2^(1:10))
[1] 2 4 8 16 32 64 128 256 512
```

Il comando che segue consente quindi di verificare la terza colonna di Gompertz

```
> head(-diff(gompertz[,2])/gompertz[-1,2])
[1] 0.000000000 0.001441868 0.004022227 0.005103189
[5] 0.006422915 0.007592154
```

Alcune opzioni che possono perfezionare un grafico sono

- `main`=: assegna il titolo al grafico;
- `xlab`=: assegna l'etichetta all'asse delle x ;
- `ylab`=: assegna l'etichetta all'asse delle y ;
- `type`=:

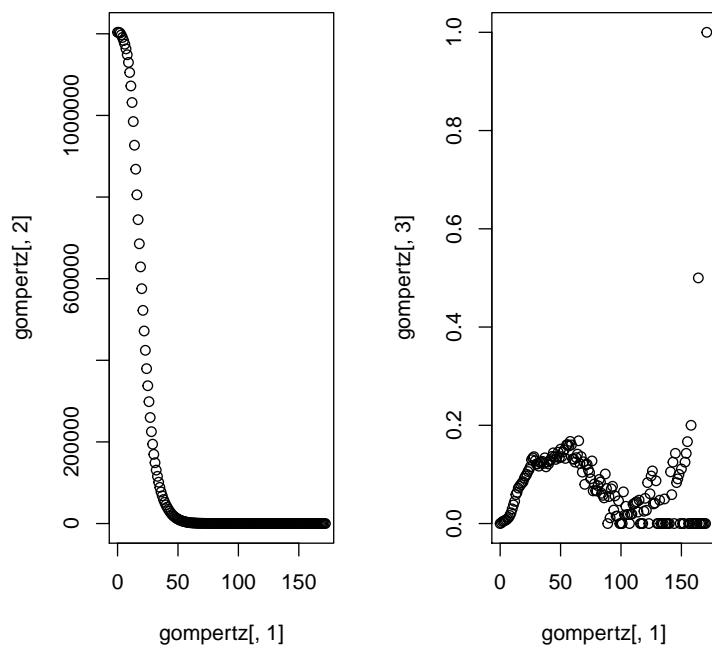


Figura 6.3: Mortalità dei moscerini. A sinistra la decrescita della popolazione, a destra la mortalità.

- **p** Disegna solo i punti.
 - **l** Disegna linee.
 - **b** Disegna entrambi ma senza sovrapporre le linee ai punti.
 - **o** Disegna linee e punti ma sovrapposti.
 - **n** Non disegna né linee né punti. Serve talora a impostare la finestra grafica.
- **col**=: assegna il colore il cui elenco si ottiene con il comando: `colors()`;
 - **cex**=: un valore numerico, da 0 (sopprime i punti) a valori numerici consigliati minori di 2, definisce la dimensione dei punti;
 - **lty**=: definisce il tipo di linea (*line type*), se 1 linea continua, se 2 tratteggiata, se 3 tratteggiata con tratto ridotto, se 4 tratto e punto;
 - **lwd**=: definisce lo spessore della linea (*line width*), valori suggeriti tra 0.5 e 2;
 - **pch**=: definisce il *marker* dei punti

1

Per esempio

```
> par(mfrow=c(2,2))
> gompertz[,c(1,3)]->dati
> plot(dati,main="La mortalit\u00e0 dei moscerini", xlab="t (giorni)",
+ ylab="Tasso di mortalit\u00e0")
> plot(dati,main="La mortalit\u00e0 dei moscerini", xlab="t (giorni)",
+ ylab="Tasso di mortalit\u00e0",type="l")
> plot(dati,main="La mortalit\u00e0 dei moscerini", xlab="t (giorni)",
+ ylab="Tasso di mortalit\u00e0", col="dark green",cex=0.5)
> plot(dati,main="La mortalit\u00e0 dei moscerini", xlab="t (giorni)",
+ ylab="Tasso di mortalit\u00e0", col="dark green",cex=0.5,
+ lty=4,lwd=2)
```

Il risultato è riportato nella figura 6.4

¹Si noti che tali opzioni coincidono in alcuni casi con opzioni specificabili con `par`. In questo caso sono però solo locali.

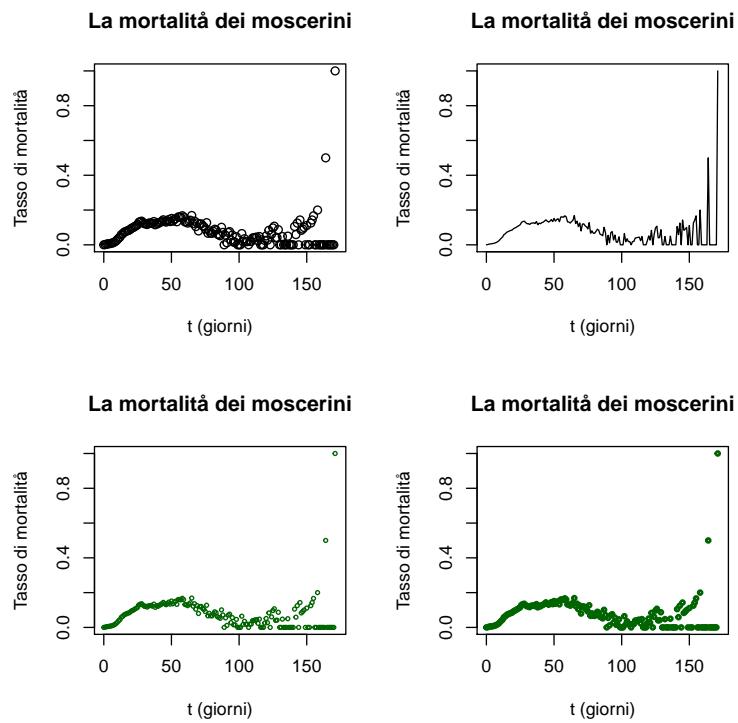


Figura 6.4: Utilizzo di diversi parametri grafici

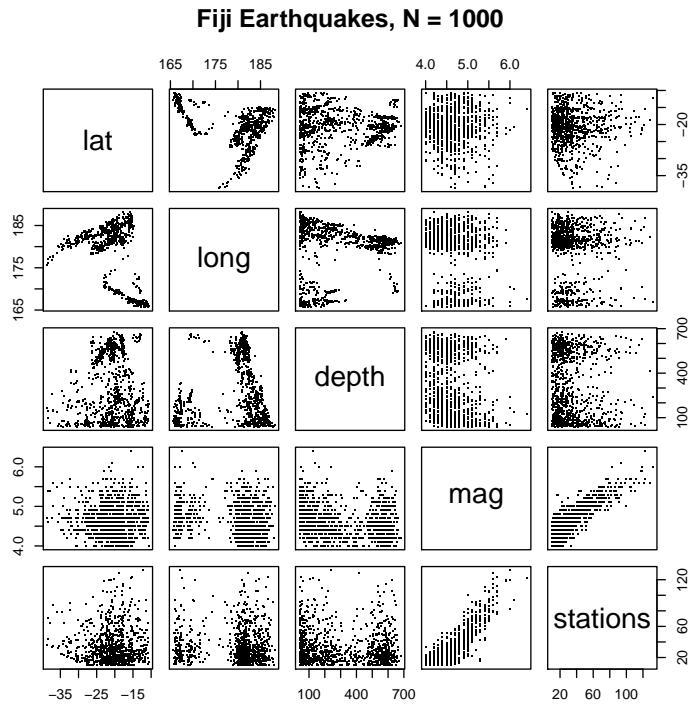


Figura 6.5: Plot per variabili multiple

6.3.3 Grafici multipli e/o sovrapposti

Analizziamo il *dataframe* `quakes` che riporta numerosi dati sui terremoti che colpiscono la zona delle isole Fiji. Carichiamo i dati e utilizziamo la funzione `pairs` per costruire grafici multipli in un'unica finestra:

```
> quakes
> pairs(quakes, main = "Fiji Earthquakes, N = 1000", cex.main=1.2, pch=".")
```

In effetti il comando `plot` funziona anche su *dataframe* come è verificabile con

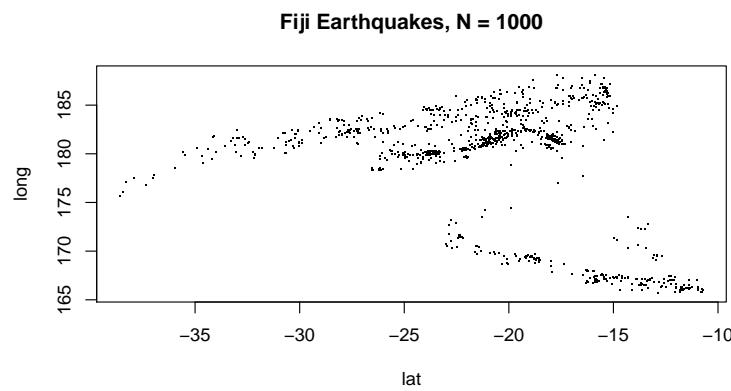
```
> methods(plot)
```

```
[1] plot.HoltWinters*   plot.TukeyHSD*
[3] plot.acf*          plot.correspondence*
[5] plot.data.frame*   plot.decomposed.ts*
[7] plot.default        plot.dendrogram*
[9] plot.density*       plot.ecdf
[11] plot.factor*        plot.formula*
[13] plot.function       plot.hclust*
[15] plot.histogram*    plot.isoreg*
[17] plot.lda*           plot.lm*
[19] plot.mca*           plot.medpolish*
[21] plot.mlm*           plot.ppr*
[23] plot.prcomp*        plot.princomp*
[25] plot.profile*       plot.profile.nls*
[27] plot.ridgelm*       plot.spec*
[29] plot.stepfun         plot.stl*
[31] plot.table*          plot.ts
[33] plot.tskernel*
```

Non-visible functions are asterisked

e produce lo stesso risultato. Il grafico risulta complesso da comprendere anche per via delle ridotte dimensioni. Selezioniamo solo i dati relativi alla longitudine ed alla latitudine (prime 2 colonne):

```
> plot(quakes[,c(1,2)], main = "Fiji Earthquakes, N = 1000", cex.main=1.2, pch=".")
```

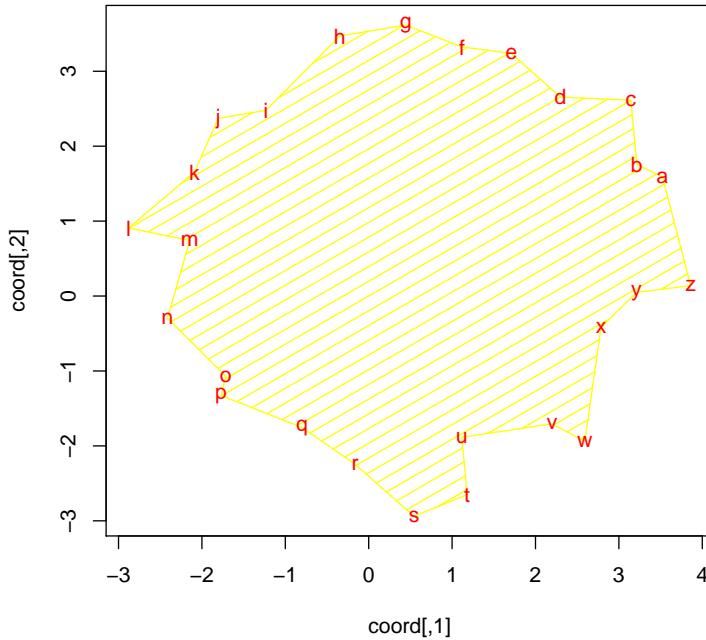


Migliorare questi grafici secondo il gusto personale inserendo, titoli, colorazioni, sfondo, tipo di rappresentazione, tipo di punti, etc.

6.3.4 Aggiungere testo al grafico

Si può aggiungere testo in qualunque posizione del grafico, in particolare possiamo posizionare commenti, annotazioni, formule, ecc. Si devono solo specificare coordinate e testo, oltre ad altri parametri visualizzabili con `?text`. Nell'esempio seguente generiamo i vertici di un poligono regolare con numero di lati uguali al numero di lettere dell'alfabeto.

```
> set.seed(20)#inizializza il generatore
> nl=length(letters)#numero di lettere
> coord.pol=matrix(c(cos(2*pi*(1:nl)/nl), sin(2*pi*(1:nl)/nl)),
+ nc=2,nrow=nl);#vertici del poligono
> coord.random<-matrix(runif(2*nl),nc=2,nrow=nl);#perturbazione
> coord=coord.random+3*coord.pol;#coordinate perturbate
> #plot(coord,col="blue",lwd=4,lty=1,type="l")
> #text(coord,letters,col="red",cex=1.6,adj=2);#inserimento del testo
> plot(coord,col="blue",lwd=4,lty=1,type="n")
> polygon(coord,col="yellow",density=10,angle=30)
> text(coord,letters,col="red")
```



Il parametro `adj` sposta il testo rispetto alle coordinate

6.3.5 Immagini di un vulcano

Vediamo alcune rappresentazioni grafiche 3D.

```
> #require(grDevices);
> #require(graphics)
> filled.contour(volcano, color.palette = terrain.colors, asp = 1)
> title(main = "volcano data: filled contour map")
```

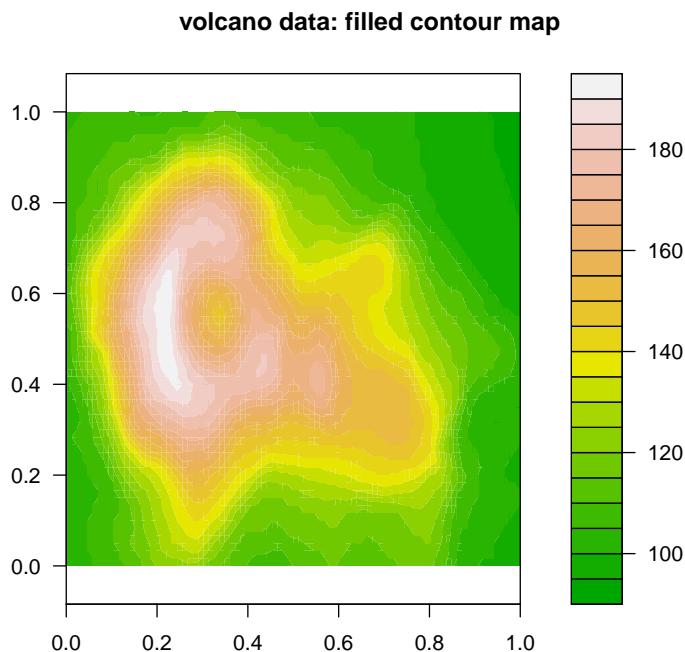
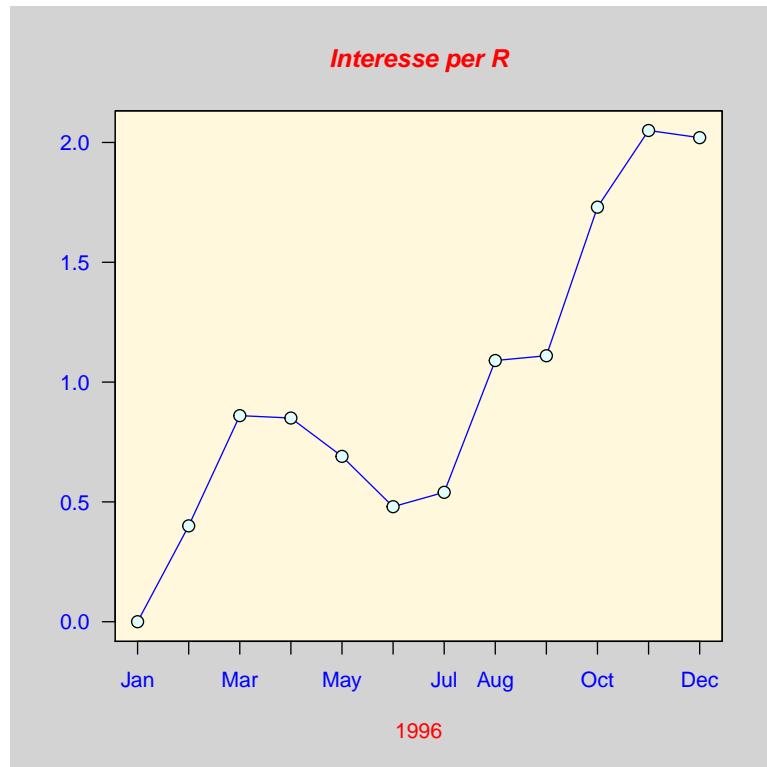


Figura 6.6: Linee di livello

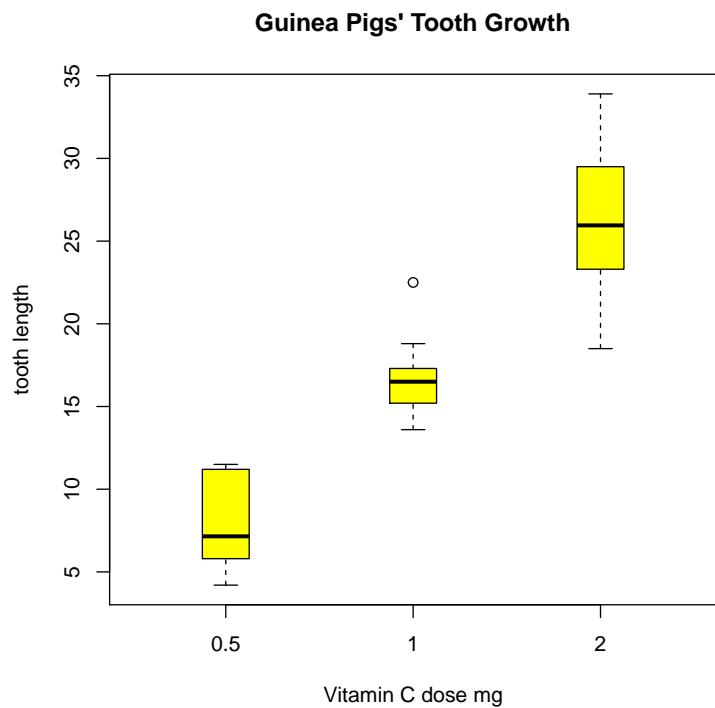
```
> x<- c(0.00, 0.40, 0.86, 0.85, 0.69, 0.48, 0.54, 1.09, 1.11, 1.73, 2.05, 2.02)
> par(bg="lightgray")
> plot(x, type="n", axes=FALSE, ann=FALSE)
```

```
> usr <- par("usr")
> rect(usr[1], usr[3], usr[2], usr[4], col="cornsilk", border="black")
> lines(x, col="blue")
> points(x, pch=21, bg="lightcyan", cex=1.25)
> axis(2, col.axis="blue", las=1)
> axis(1, at=1:12, lab=month.abb, col.axis="blue")
> box()
> title(main="Interesse per R", font.main=4, col.main="red")
> title(xlab="1996", col.lab="red")
```



Un esempio:

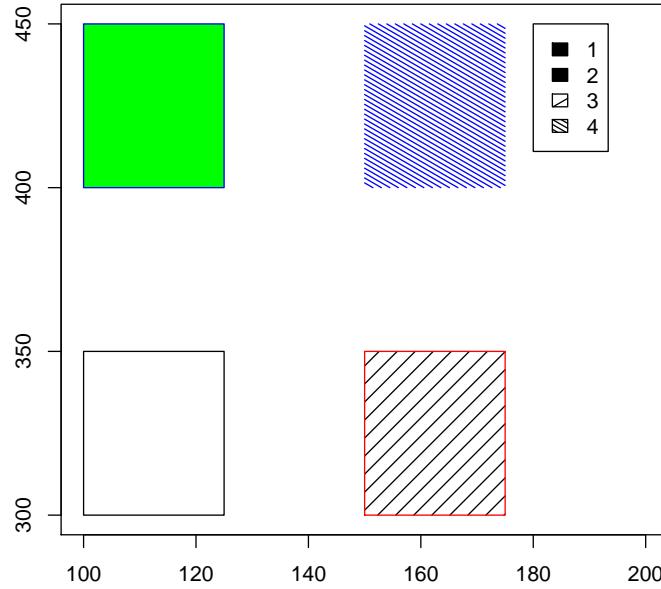
```
> boxplot(len~dose, data = ToothGrowth,
+ boxwex = 0.25, at = 1:3 - 0.2,
+ subset = supp == "VC", col = "yellow",
+ main = "Guinea Pigs' Tooth Growth",
+ xlab = "Vitamin C dose mg",
+ ylab = "tooth length")
```



6.3.6 I rettangoli

R consente di disegnare forme (dette primitive) come rettangoli o poligoni. I rettangoli in particolare sono molto utili e versatili, consentendo di realizzare riquadri, evidenziare regioni grafiche, ecc.

```
> plot(c(100, 200), c(300, 450), type= "n", xlab= "", ylab= "")  
> rect(100, 300, 125, 350) # transparent  
> rect(100, 400, 125, 450, col="green", border="blue") # coloured  
> rect(115, 375, 150, 425, col=par("bg"), border="transparent")  
> rect(150, 300, 175, 350, density=10, border="red")  
> rect(150, 400, 175, 450, density=30, col="blue", angle=-30, border="transparent")  
> legend(180, 450, legend=1:4, fill=c(NA, "green", par("fg"), "blue"),  
+ density=c(NA, NA, 10, 30), angle=c(NA, NA, 30, -30))
```

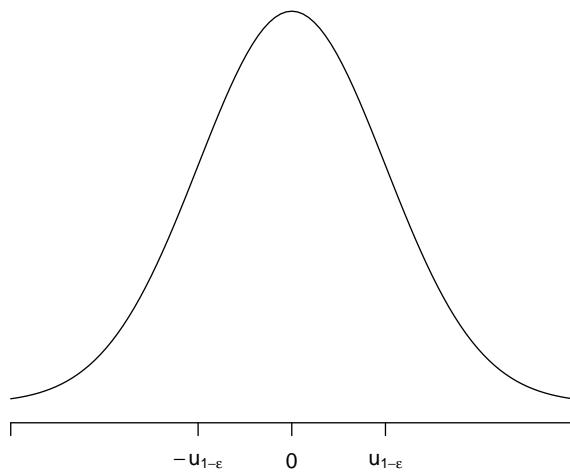


6.4 Grafici professionali: legenda, annotazioni e formule

Ricordiamo che una volta che sia assegnato il livello di fiducia $1 - \epsilon$, il numero $u_{1-\epsilon}$ che leggiamo nelle tabelle distribuzione normale è il valore dell'ascissa tale che nell'intervallo $[-u_{1-\epsilon}, u_{1-\epsilon}]$ cada un'area pari al livello di fiducia; in altre parole le 2 code della distribuzione vengono rimosse. Vogliamo ora creare un grafico illustrativo di questa definizione.

Traceremo inizialmente un grafico della normale tra $[-3, 3]$ scegliendo come valore dell'ascissa $u_{1-\epsilon}$ il numero 1. Vogliamo quindi scrivere l'espressione $u_{1-\epsilon}$ nel punto di ascissa 1 e il suo opposto nel punto di ascissa -1 dell'asse x . Per disegnare delle tacche sugli assi e assegnare valori alfanumerici alle stesse si può utilizzare Il comando

```
axis(num, c(tacca1, tacca2, ..., taccan),
      c("nome1, "nome2", ..., "nomen"))
```

Figura 6.7: Selezione delle etichette sull'asse x .

Qui *tacca* rappresenta il valore della coordinata mentre *num* è un indice che rappresenta l'asse in esame come segue 1=sotto, 2=sinistra, 3=sopra and 4=destra.

Possiamo procedere con

```
> curve(dnorm(x), -3, 3, axes=FALSE, ylab="", xlab="", ylim=c(0,0.5))
> axis(1,c(-3,-1,0,1,3),c("",expression(-u[1-epsilon]),0,expression(u[1-epsilon]),""))
```

Il comando **expression** consente di scrivere simboli e anche lettere greche. Come uscita di questo primo comando troviamo il grafico (6.7) Per evidenziare le 2 code tratteggiamo l'area sottesa dalla normale tra 1 e 3 (e in seguito da -3 a -1). Per operare questo tratteggio si approssima la regione da tratteggiare con un poligono con un numero molto elevato (200) di vertici definiti come segue

```
> xmin<-1;xmax<-3;
> npunti<-200
> f=dnorm#selezione degli estremi
> #e del numero di punti base del poligono
> vals<-seq(xmin,xmax,length=npunti)
> x<-c(xmin,vals,xmax,xmin)
> y<-c(0,f(vals),0,0);
```

La sintassi di polygon?è

```
polygon(x,y,density = val1,angle = val2,
        col="col")
```

Il comando

```
> curve(dnorm(x),-3,3,axes=FALSE,ylab="",xlab="",ylim=c(0,0.5))
> axis(1,c(-3,-1,0,1,3),c("",expression(-u[1-epsilon])),0,expression(u[1-epsilon]))
```

consente la visualizzazione dei punti ottenuti (come in figura (6.8)).

Questi punti possono poi essere uniti con una curva poligonale

```
> curve(dnorm(x),-3,3,axes=FALSE,ylab="",xlab="",ylim=c(0,0.5))
> axis(1,c(-3,-1,0,1,3),c("",expression(-u[1-epsilon])),0,expression(u[1-epsilon]))
> points(x,y,pch=20,col="red",cex=0.2)
> polygon(x,y,density=20,angle=45,col="RED")
> x<--x
> y<-c(0,dnorm(-vals),0,0)
> polygon(x,y,density=20,angle=45,col="RED")
```

Infine possiamo riportare i valori delle aree delle 2 code, evidenziare l'asse di simmetria e dare un titolo al grafico

```
> curve(dnorm(x),-3,3,axes=FALSE,ylab="",xlab="",ylim=c(0,0.5))
> axis(1,c(-3,-1,0,1,3),c("",expression(-u[1-epsilon])),0,expression(u[1-epsilon]))
> points(x,y,pch=20,col="red",cex=0.2)
> polygon(x,y,density=20,angle=45,col="RED")
> x<--x
> y<-c(0,dnorm(-vals),0,0)
> polygon(x,y,density=20,angle=45,col="RED")
> polygon(x,y,density=20,angle=45,col="RED")
> abline(h=min(y))
```

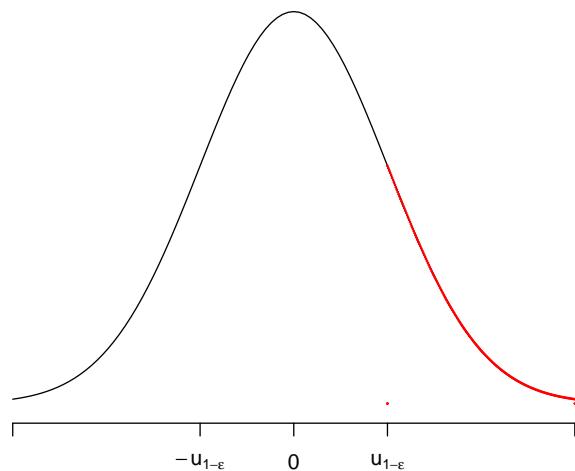
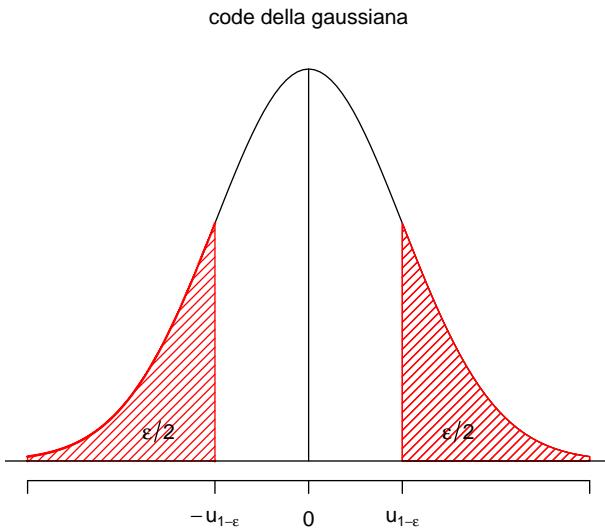


Figura 6.8: Selezione dei punti per l'ombreggiamento di una regione poligonale

```

> text(0,0.45,expression("code della gaussiana"))
> text(-1.6,0.03,expression(epsilon/2))
> text(1.6,0.03,expression(epsilon/2))
> lines(c(0,0),c(0,dnorm(0)))

```



6.5 Pittura virtuale

Iniziamo a disegnare dei rettangoli, di coordinate casuali ma entro un dato range, di colore casuale, con spessore del bordo casuale.

```

> par(bg="skyblue");
> plot(0,xlim=c(0.3,1),ylim=c(0.3,1),axes=F, xlab="",ylab="")
> for(i in 1:100)
+ { Sys.sleep(0.01);
+ runif(1)->sotto;
+ runif(1)->sinistra;
+ runif(1)+sotto->sopra;
+ runif(1)+sinistra->destra;

```

```
> curve(dnorm(x),-3,3,axes=FALSE,ylab="",xlab="",ylim=c(0,0.5))
> polygon(x,y,density=20,angle=45,col="RED")
> abline(h=min(y))
> text(0,0.45,expression("code della gaussiana"))
> text(-1.6,0.03,expression(epsilon/2))
> text(1.6,0.03,expression(epsilon/2))
> lines(c(0,0),c(0,dnorm(0)))
```

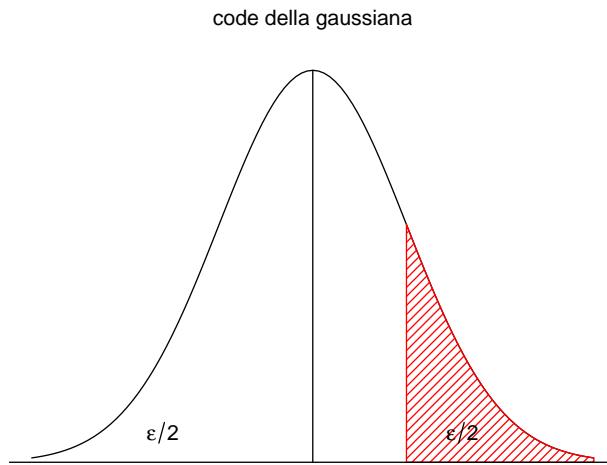
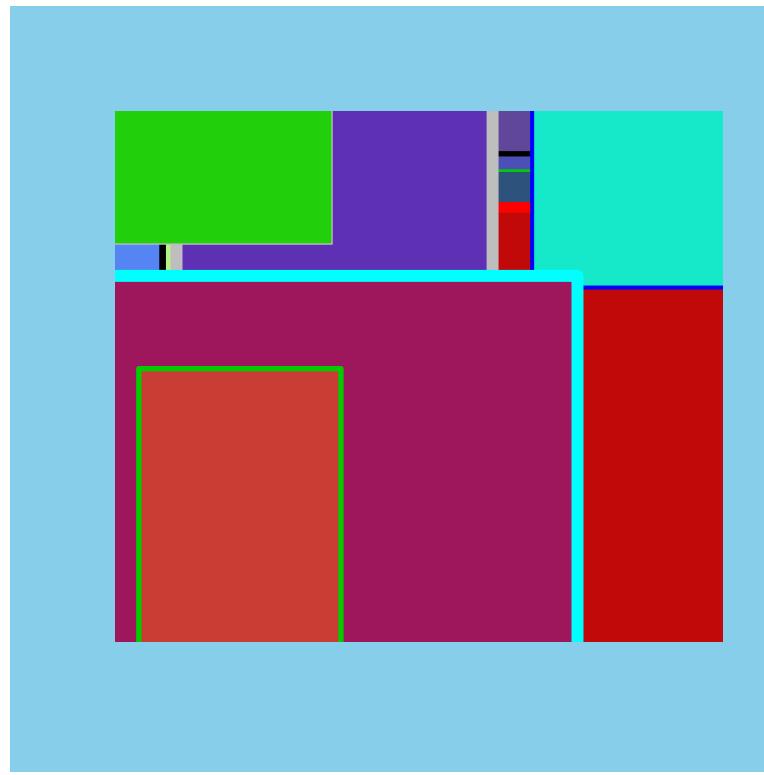


Figura 6.9: Area delle code della distribuzione normale. Il valore di $u_{1-\epsilon}$ è quello riportato nella tabella della distribuzione normale.

```
+ bordi<-0;for(i in 1:100) bordi[i]<-floor(runif(1,1,10))
+ spess<-0;for(i in 1:100) spess[i]<-floor(runif(1,1,10))
+ rect(sinistra,sotto,destra,sopra,col=rgb(runif(1),runif(1),runif(1)),border=1)
```



6.6 Grafica 3D, il vulcano Maunga Whau

Con il codice che segue

```
> z<- 2 * volcano
> x<- 10 * (1:nrow(z))
> y<- 10 * (1:ncol(z))
> z0<- min(z) - 20
> z<- rbind(z0, cbind(z0, z, z0), z0)
> x<- c(min(x) - 1e-10, x, max(x) + 1e-10)
> y<- c(min(y) - 1e-10, y, max(y) + 1e-10)
> fill <- matrix("green3", nr = nrow(z)-1, nc = ncol(z)-1)
> fill[ , i2 <- c(1,ncol(fill))] <- "gray"
> fill[i1 <- c(1,nrow(fill)) , ] <- "gray"
```

```
> fcol <- fill
> zi <- volcano[ -1,-1] + volcano[ -1,-61] +
+ volcano[-87,-1] + volcano[-87,-61] ## / 4
> fcol[-i1,-i2] <- terrain.colors(20)[cut(zi, quantile(zi, seq(0,1, len = 21)),
+ include.lowest = TRUE)]
> par(mar=rep(.5,4))
> persp(x, y, 2*z, theta = 110, phi = 40, col = fcol, scale = FALSE,
+ ltheta = -120, shade = 0.4, border = NA, box = FALSE)
```

otteniamo la figura 6.10. Il comando di base è `persp` che richiede una griglia di valori di (x, y) e il corrispondente valore di quota.

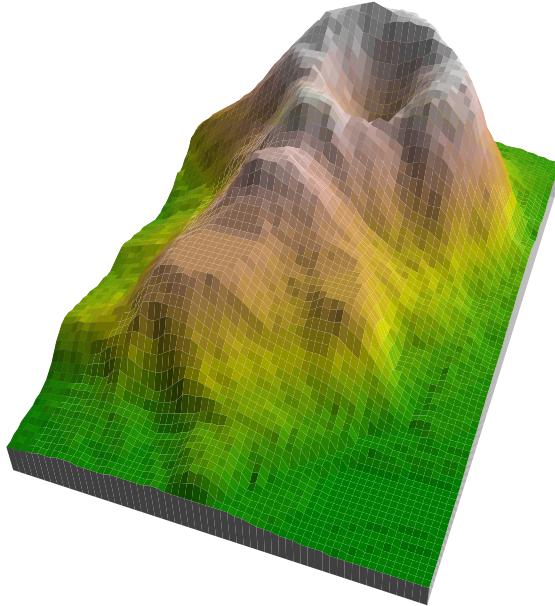
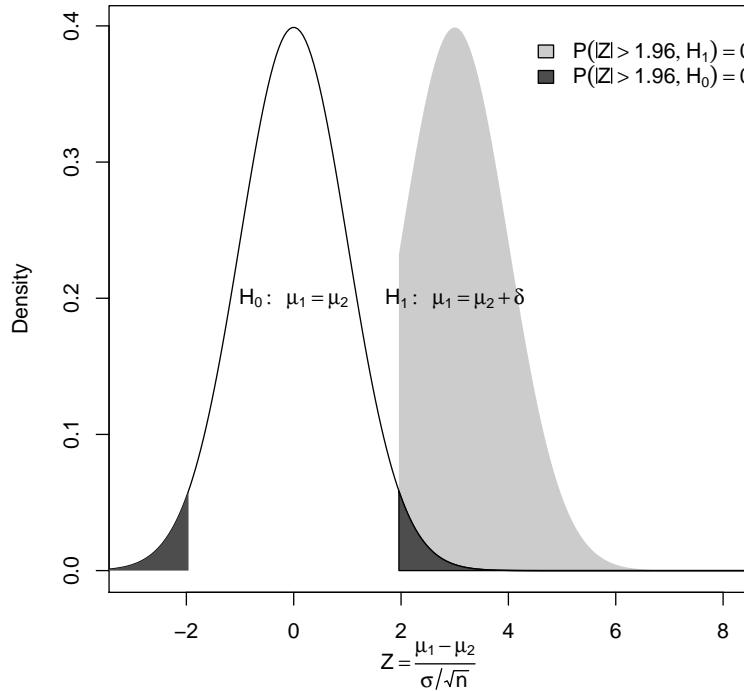


Figura 6.10: Grafica 3 dimensionale, ottenuta con il comando `persp`

Costruiamo due curve che rappresentano due distribuzioni normali, in particolare al posto della funzione `dnorm` andremo a calcolare i singoli valori di y associati ad un set di valori di x , essendo poi liberi di utilizzare la funzione `plot`

```
> options(width=60)
> x<-seq(-10,10,length=400)
> y1<-dnorm(x)
> y2<-dnorm(x,m=3)
> par(mar=c(5,4,2,1))
> plot(x, y2, xlim=c(-3,8), type="n", xlab=quote(Z==frac(mu[1]-mu[2],
+ sigma/sqrt(n))), ylab="Density")
> polygon(c(1.96,1.96,x[240:400],10), c(0,dnorm(1.96,m=3),y2[240:400],0),
+ col="grey80", lty=0)
> lines(x, y1)
> polygon(c(-1.96,-1.96,x[161:1],-10), c(0,dnorm(-1.96,m=0), y1[161:1],0),
+ col="grey30", lty=0)
> polygon(c(1.96, 1.96, x[240:400], 10), c(0,dnorm(1.96,m=0),
+ y1[240:400],0), col="grey30")
> legend(4.2, .4, fill=c("grey80","grey30"),
+ legend=expression(P(abs(Z)>1.96, H[1])==0.85,
+ P(abs(Z)>1.96,H[0])==0.05), bty="n")
> text(0, .2, quote(H[0]:~~mu[1]==mu[2]))
> text(3, .2, quote(H[1]:~~mu[1]==mu[2]+delta))
```



6.7 Web

In questa sezione vedremo come collegarsi al *web*, come leggere il contenuto di file `.html`, come aprire indirizzi web nel browser. La funzione `readLines` consente di leggere il codice sorgente (`.html`) di un determinato sito web. Il numero che segue l'indirizzo definisce il numero di righe che si vogliono leggere. Se invece vogliamo aprire una pagina web direttamente nel browser utilizziamo `browseURL`, definendo il browser (il default è dipendente dalle preferenze dell'utente e soprattutto dal sistema operativo).

```
> readLines("http://www.r-project.org/",4)

> channel="http://www.youtube.com/watch?v=W2GZFeYGu3s&feature=related"

> browseURL(channel,browser=getOption("browser"))
```

6.7.1 Colori

6.7.2 Costruire una funzione per realizzare grafica su *device*

Lo scopo di questa procedura è quello di realizzare una funzione, che chiameremo `mondrian`, che ci consentirà di contenere ed eseguire il codice necessario per disegnare la riproduzione del quadro che alcuni di voi hanno manualmente digitato la scorsa lezione. Gli step necessari per realizzare ciò sono i seguenti:

- 1. Copiare in locale il codice per realizzare la figura su *device*;
- 2. Inserire questo codice nella funzione `mondrian`, che realizziamo con il comando `fix(mondrian)`;
- 3. Richiamare la funzione con il comando: `mondrian()`
- 4. Verificare che sia stato prodotto il file `mondrian.png` e sua apertura. Se non specifichiamo l'indirizzo del file esso si troverà nella cartella di lavoro di R, ottenibile con il comando `getwd()`.

6.8 Grafica 3D, grafico di una funzione di due variabili

La funzione di base in R per realizzare grafica 3D statica è `persp`. Gli input richiesti sono due vettori (coordinate sulle *x* e sulle *y*) ed una matrice (di ingressi ingressi *x* e *y*, contenente all'incrocio *z*).

```
> f<-function(x,y)  y^2+ x^2
> xset<-seq(-1,1,length=101)
> yset<-seq(-1,1,length=101)
> zmatr<-matrix(0,length(xset),length(yset))
> for(i in 1:length(xset)) for(j in 1:length(yset)) zmatr[i,j]<-f(xset[i],xse
> persp(xset,yset,zmatr)##persp possiede parametri di rotazione, colorazione,
> persp(xset,yset,zmatr,theta = 30, phi = 60, r = 10, d = 10)
```

6.8.1 Il pacchetto `rgl`

Il pacchetto `rgl` si compone di funzioni per la grafica 3D dinamica e consente di realizzare grafici ruotabili, eventualmente animati e sicuramente di grande impatto e potenzialità. Il comando base per aprire una *device* grafica di `rgl`

è `open3d()`. Cominciamo con il disegnare un solido, in particolare un cubo, mediante il comando `shade3d`, Prestiamo attenzione alla generazione della lista dei colori (`rep` è un comando molto potente):

```
> library(rgl)
> rgl.postscript(file="cubo.eps")
> shade3d(cube3d(color=rep(rainbow(6),rep(4,6))))
```

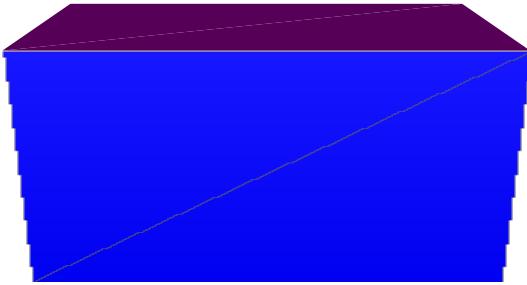


Figura 6.11: **default**

Il comando `plot3d` è il più semplice ed immediato comando per realizzare un grafico tridimensionale. Esso richiede come input le coordinate (x, y, z) dei punti da rappresentare. Nell'esempio generiamo 1000 punti, notate la colorazione ed il grafico rotabile (mentre con il tasto destro del mouse si esegue lo zoom).

```
> rgl.postscript(file="1000punti.eps")
> open3d()
> x <- sort(rnorm(1000))
> y <- rnorm(1000)
> z <- rnorm(1000)
> plot3d(x, y, z, col=rainbow(1000), size=2)
> dev.off()
```

otteniamo cos' la figura

Ora riprendiamo le coordinate del Maunga Whau e rappresentiamolo con il comando , il quale possiede numerosi parametri di ombreggiatura, tonalità e sfumature di colorazioni, tutte ottenibili dal comando `par3d`. Il vulcano:

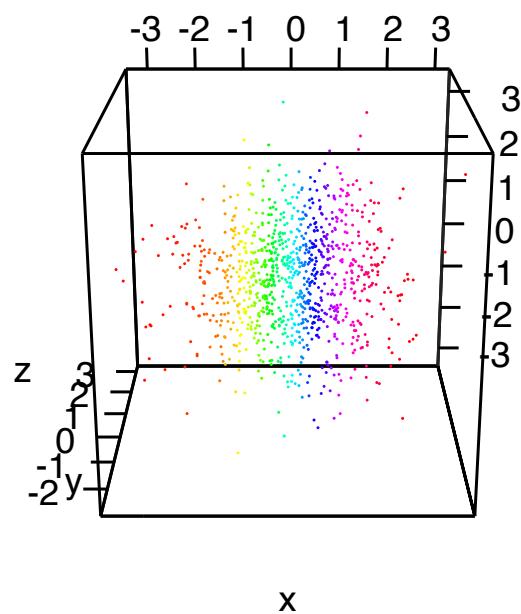


Figura 6.12: Risultato `plot3D`

```
> library(rgl)
> rgl.postscript(file="../grafici/vulcano.eps")
> data(volcano)
> z <- 2 * volcano
> x <- 10 * (1:nrow(z))
> y <- 10 * (1:ncol(z))
> zlim <- range(y)
> zlen <- zlim[2] - zlim[1] + 1
> colorlut <- terrain.colors(zlen)
> col <- colorlut[ z-zlim[1]+1 ]
> open3d()
> surface3d(x, y, z, color=col, back="lines")
```

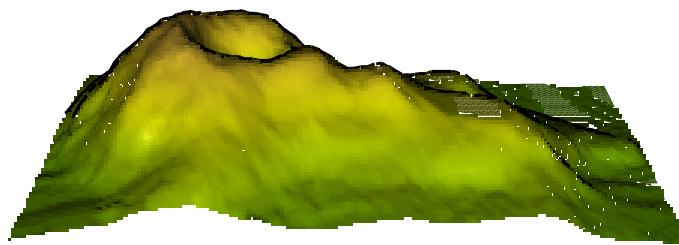


Figura 6.13: Il vulcano realizzato con `surface3D`

Il grafico realizzato con il comando precedente in modo automatico, usando il comando `rgl.viewpoint`:

```
> example(rgl.surface)
> for(i in 1:360) {rgl.viewpoint(i, i*(60/360), interactive=F)}
```

Creare piccole animazione grafiche: play3d

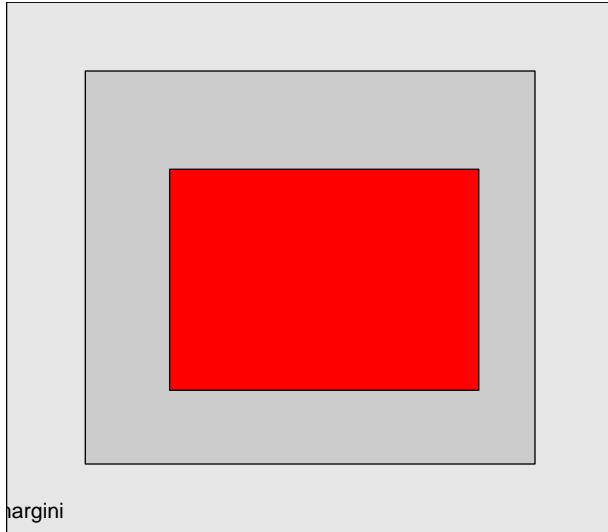
```
> open3d()
> plot3d( cube3d(col="green") )
> M <- par3d("userMatrix")
> play3d( par3dinterp( userMatrix=list(M,rotate3d(M, pi/2, 1, 0, 0),
+ rotate3d(M, pi/2, 0, 1, 0) ), duration=1 )
```

6.8.2 Personalizzare la regione di plot

Il comando `plot.new()` crea una finestra che contiene la regione di plot vera e propria, fiancheggiata dalle regioni di margine che sono predisposte per contenere le annotazioni. La finestra complessiva di default è mostrata in figura

A volte è necessario personalizzare i margini, ad esempio se le annotazioni dei due assi sono molto ingombranti oppure nel caso di plot multipli in un'unica finestra (in tal caso, i grafici diventano molto piccoli ed è possibile guadagnare spazio eliminando i margini inutilizzati).

```
> plot.new()
> rect(par("usr")[1],par("usr")[3],par("usr")[2],par("usr")[4],col=gray(0.9))
> rect(0.1,0.1,0.9,0.9,col=gray(0.8))
> rect(0.25,0.25,0.8,0.7,col="red")
> text(0,0,"margini")
```



Tale modifica si opera con il comando `mar`.

$$\text{par}(\text{mar}=\text{c}(l_1, l_2, l_3, l_4)) \quad (6.1)$$

dove l_1, l_2, l_3, l_4 sono le linee di testo da lasciare libere per annotazioni su ciascun lato come in figura. Un comando analogo è `mai`, che consente di definire gli spazi in *inch* (pollici):

$$\text{par}(\text{mai}=\text{c}(i_1, i_2, i_3, i_4)) \quad (6.2)$$

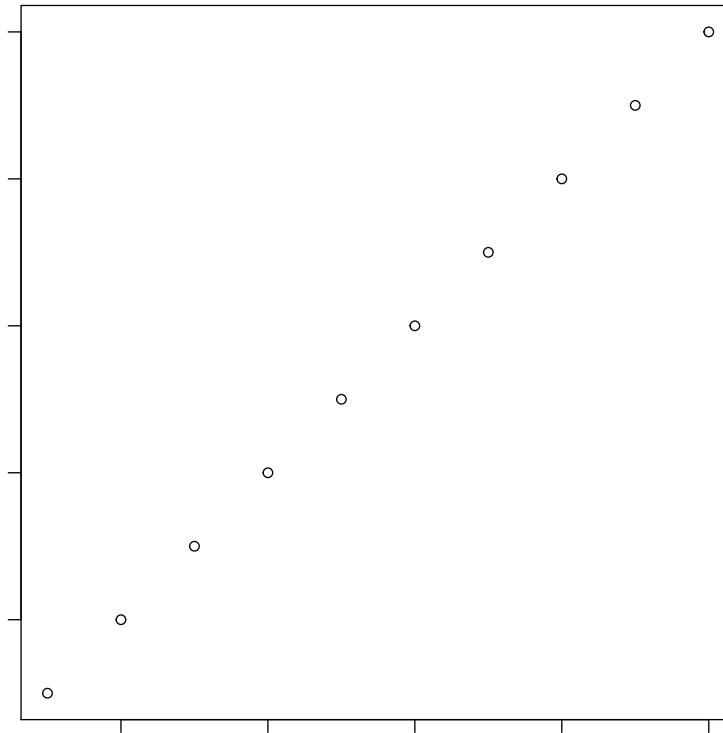
Infine, è possibile definire direttamente la regione di plot utilizzando `pin`, di sintassi:

$$\text{par}(\text{pin}=\text{c}(w, h))$$

dove w è la larghezza e h l'altezza della regione di plot. Proviamo i comandi:

```
> par(mar=c(1,1,1,1))
> plot(1:10)
> par(mai=c(1,1,1,1))
> plot(1:10)
```

```
> par(fin=c(5,3)) ##notare come questa sia centrata  
> plot(1:10)
```



6.8.3 Il pacchetto lattice

Il pacchetto `lattice` consente di utilizzare funzioni avanzate di grafica, tra cui alcune che sono in grado di lavorare con `data.frame` anche molto complessi.

Grafica 3D, il vulcano con la grafica non tradizionale del pacchetto `lattice`

Una prima funzione in particolare è la funzione `wireframe`, parente stretta di `persp3D` e delle funzioni 3D di `rgl`. Possiamo utilizzarla per realizzare il plot del solito vulcano, in particolare abilitando la colorazione (`shade=F` evita il riempimento), dandone un aspetto pieno (rapporto larghezza/altezza, dipendente da `ratio`) e definendo una sorgente di luce particolare:

```
> library(datasets)
> library(lattice)
> pdf(file="provawire.pdf")
> wireframe(volcano, shade = TRUE,
+ aspect = c(61/87, 0.4),
+ light.source = c(10,0,10))
> dev.off()
```

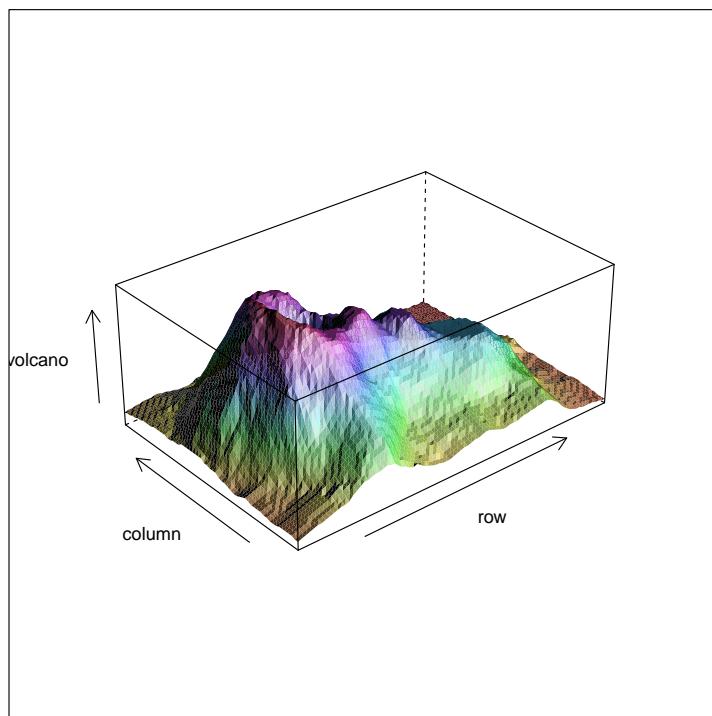


Figura 6.14: Grafico wireframe.

Una seconda funzione importante, che consente di realizzare dei plot in tre dimensioni è `cloud`. Se ad esempio disponiamo di una tabella come `Titanic` (riporta il numero di passeggeri sopravvissuti o meno al naufragio ripartiti per età, classe di viaggio e sesso) e voletessimo ottenere invece una tabella che ne riporti la proporzione sul totale possiamo usare la funzione `prop.table()`. Disegniamo ora con `cloud` la situazione dopo il naufragio, con `strip` definiamo le due righe di titolo (in cui sono ripartiti i dati), e

con la quadra [,1:2] finale disegniamo entrambe le classi (sopravvissuti e non sopravvissuti).

```
> pdf(file="../grafici/Titanic.pdf")
> library(lattice)
> cloud(prop.table(Titanic),
+ type = c("p", "h"), strip = strip.custom(strip.names = TRUE),
+ scales = list(arrows = FALSE, distance = 2), panel.aspect = 0.7,
+ zlab = "Proportion")[,1:2]
> dev.off()
```

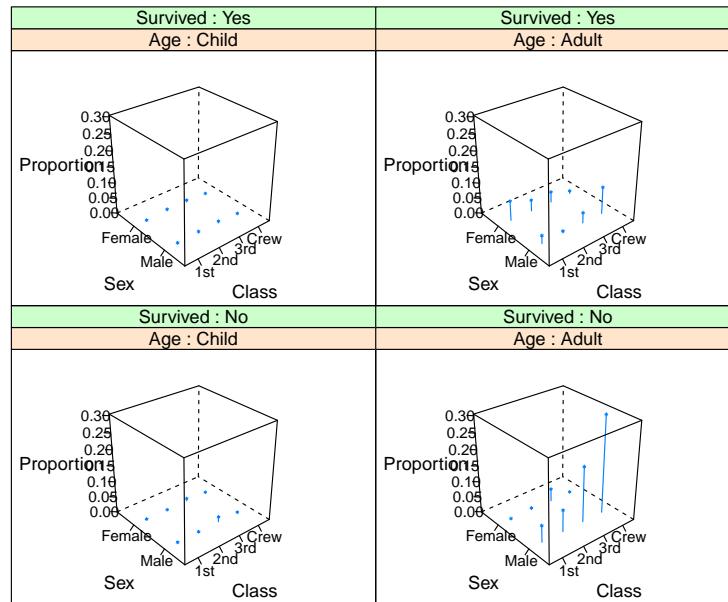


Figura 6.15: default

Oppure rappresentiamo il *dataset* colore occhi/capelli degli studenti ripartito in base al sesso

```
> pdf(file="../grafici/cloudgraph.pdf")
> library(lattice)
```

```
> cloud(prop.table(HairEyeColor),
+ type = c("p", "h"),
+ strip = strip.custom(strip.names = TRUE))
> dev.off()

pdf
2
```

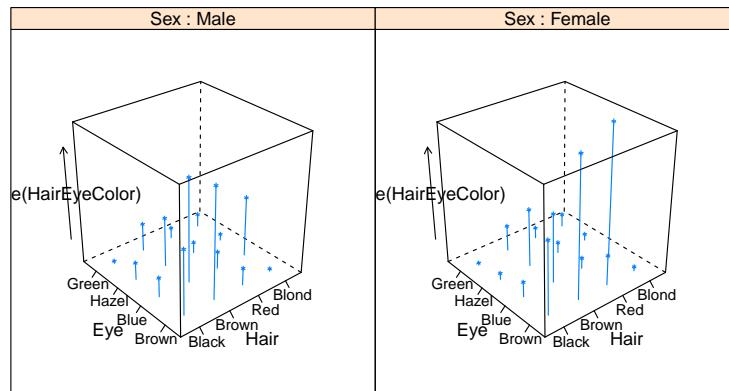


Figura 6.16: Cloud graph

6.8.4 Il parallel plot

Un plot di questo tipo parte da una matrice o da un *dataframe*, rappresentando graficamente ogni riga dello stesso (ingresso) con le sue coordinate (il cui numero dipende dal numero dei parametri). I punti ottenuti vengono così congiunti da segmenti (attribuendo un colore automatico ad ogni

campione, ridefinibile). L'idea di fondo è quella di valutare se campioni ad esempio di specie o fenotipi differenti presentano profilo simile (e se diverso lo scopo è quello di identificare le variabili/colonne discriminanti). Nell'esempio vogliamo rappresentare larghezza e lunghezza dei petali e del sepalo dei fiori Iris, volendone poi valutare i profili tra le tre differenti specie. I dati che ci interessano sono cos? le prime 4 colonne (diremo iris[1:4] perché il modello della funzione lo richiede) mentre le intestazioni saranno le specie di iris, definite da Species, iris.

```
> pdf(file="../grafici/parallelplot.pdf")
> parallelplot(~iris[1:4] | Species, iris)
> dev.off()

pdf
2
```

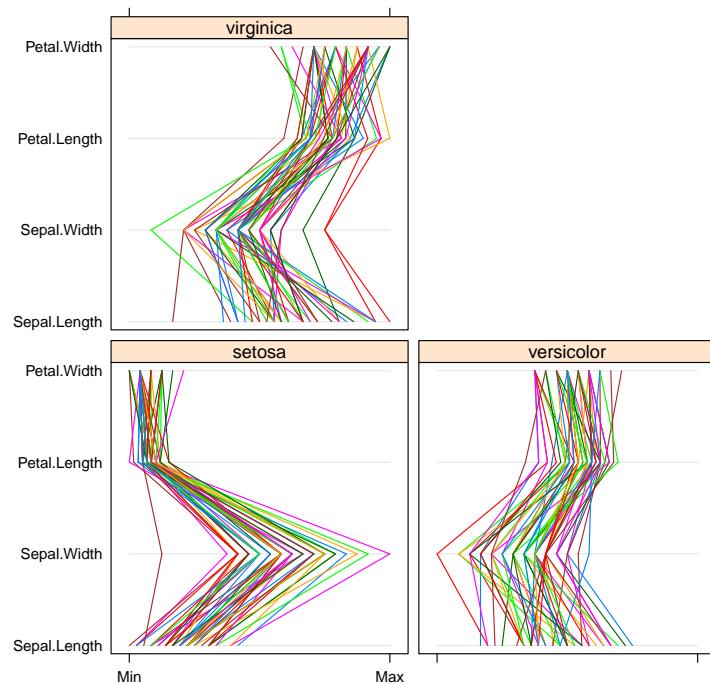


Figura 6.17: Cloud graph

Oppure valutiamo due diversi trattamenti con vitamina C per la crescita dei denti suini, ricordate?

```
> library(lattice)
> pdf(file="../grafici/parplot2.pdf")
> parallelplot(~ToothGrowth[c(1,3)] | supp, ToothGrowth)
> dev.off()

pdf
2
```

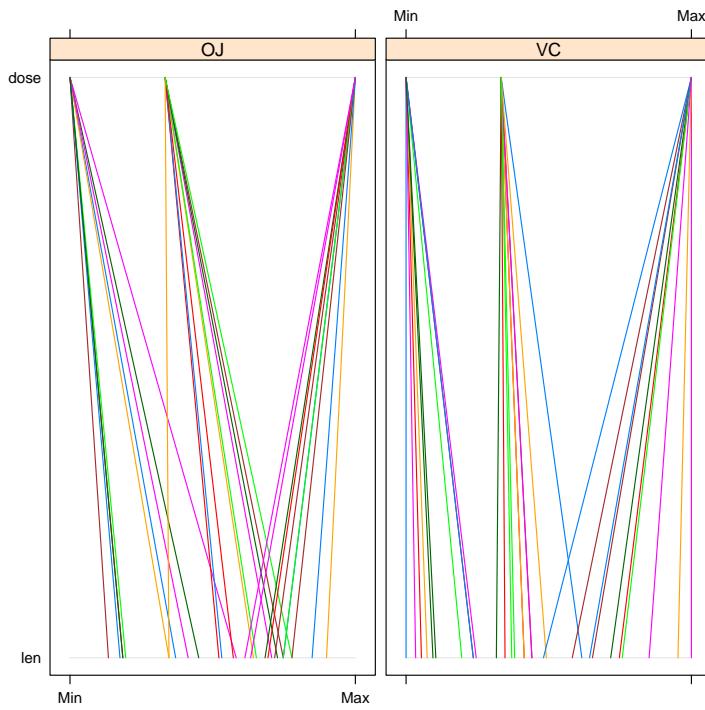


Figura 6.18: Cloud graph

6.8.5 Applicazione in campo geografico

Proveremo ora a rappresentare gli Stati Uniti d’America sfruttando un *dataset* predisposto e due nuovi pacchetti che sono spesso utilizzati per studi

geografici e rappresentazione di cartine topologiche. Esse sono in particolare **sp** (*spatial graphs*, con numerosi comandi di grafica) e **maptools**, che si compone sostanzialmente di funzioni in grado di lavorare con dei poligoni. Scaricate (dal menu installazione pacchetti) e caricate i due pacchetti:

```
> library(sp)
> library(maptools)
> #gpclibPermit()
```

Fatto ciò, la prima cosa da fare è leggere il contenuto del file *sids.shp*, il quale contiene le informazioni sui poligoni (i vari stati componenti), i confini e le annotazioni letterarie (ad es. file capitali). Il file risiede nelle *library* di sistema di R (indirizzo visibile col comando: system.file(-shapes/sids.shp)). Costruiamo l'oggetto base (**nc**) e trattiamo le coordinate con CRS:

```
> options(width=55)
> nc <- readShapePoly(system.file("shapes/sids.shp", package="maptools")[1],
+ proj4string=CRS("+proj=longlat +datum=NAD27"))
> names(nc)
```

Poi creiamo i due pannelli **pannello1** e **pannello2** con il comando **sample** (sceglie un campione di 100 numeri a caso tra 1 a 5), con etichette le prime 5 lettere dell'alfabeto.

```
> set.seed(31) #fissiamo il random seed per il sample
> nc$pannello1 = factor(sample(1:5,100,replace=T),labels=letters[1:5])
> nc$pannello2 = factor(sample(1:5,100,replace=T),labels=letters[1:5])
```

Ora carichiamo la grafica di **RColorBrewer**:

```
> library(RColorBrewer)
```

e utilizziamo **spplot**, la funzione costruita appositamente per lavorare con dati spaziali aventi attributi. Essa prendere come ingressi **nc**, gli attributi **f**, **g** e colorerà le regioni con la paletta **brewer.pal(5,Set3)**. Potrete poi provare anche Set1 e Set2.

```
> pdf("../grafici/SPPLOT.pdf")
> spplot(nc, c("pannello1","pannello2"),
+ col.regions=brewer.pal(5, "Set3"), scales=list(draw=TRUE))
> dev.off()
```

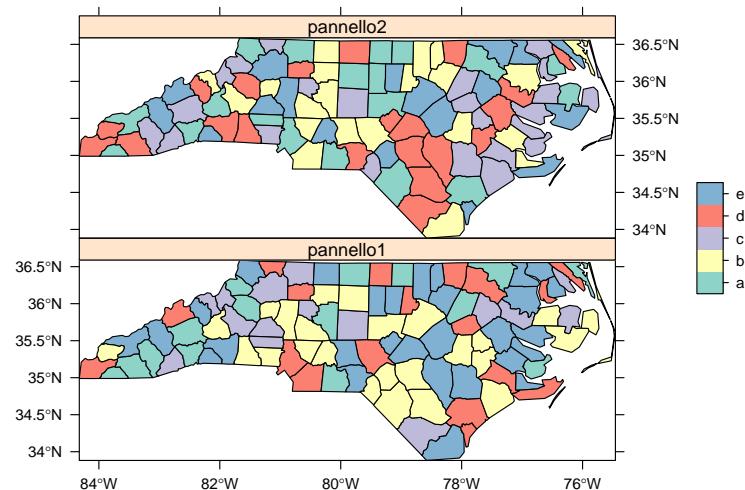
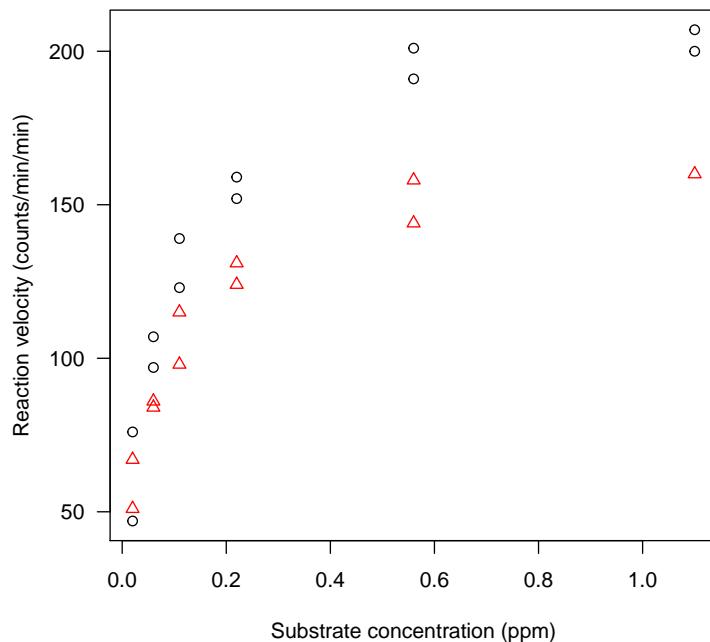


Figura 6.19: maptool

```
> library(datasets) #eurodist
> require(stats); require(graphics)
> plot(rate ~ conc, data = Puromycin, las = 1,
+ xlab = "Substrate concentration (ppm)",
+ ylab = "Reaction velocity (counts/min/min)",
+ pch = as.integer(Puromycin$state),
+ col = as.integer(Puromycin$state),
+ main = "Puromycin data and fitted Michaelis-Menten curves")
```

Puromycin data and fitted Michaelis-Menten curves

Sarebbe poi meglio sostituire con il reale valore degli anni le etichette presenti sull'asse x originale. Questo si può ottenere in diversi modi (vedi `axis`), ma perché non sfruttare l'attributo `names` della struttura dati?

```
■echo=TRUE,eval=FALSE■= names(dati)<-as.character(floor(seq(1860,1959,length=length(dati)))
plot(as.vector(names(dati)),dati,main="Andamento delle scoperte scientifiche 1860-1959", xlab="Anni, dal 1860 al 1959",ylab="Numero di scoperte",type="o", col="dark green",cex=0.5,lty=4)
```

Possiamo poi anche orientare diversamente il grafico, scambiando l'asse x con l'asse y , la rappresentazione dei dati specie nell'insieme può apparire più appropriata, oppure peggiore:

```
> plot(dati,as.vector(names(dati)),main="Andamento delle scoperte scientifiche 1860-1959",
+ xlab="Anni, dal 1860 al 1959",ylab="Numero di scoperte",type="o",
+ col="dark green",cex=0.5,lty=4,lwd=2)
```

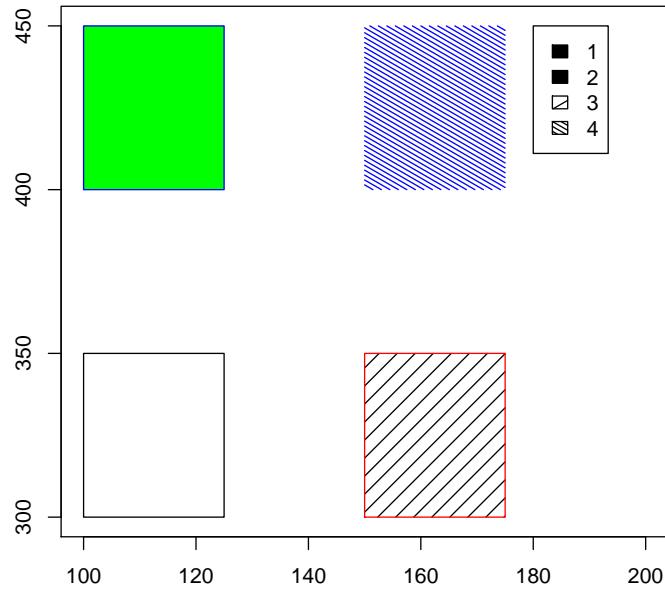
6.8.6 Il parametro las

L'orientamento delle etichette degli assi è molto importante, il parametro `las`, che assume valori interi tra 0 e 3 può essere usato a questo scopo. Provate a capire cosa succede al comando precedente aggiungendo nei parametri `las=2` e altri possibili valori.

6.9 I rettangoli

R consente di disegnare forme (dette primitive) come rettangoli o poligoni. I rettangoli in particolare sono molto utili e versatili, consentendo di realizzare riquadri, evidenziare regioni grafiche, ecc.

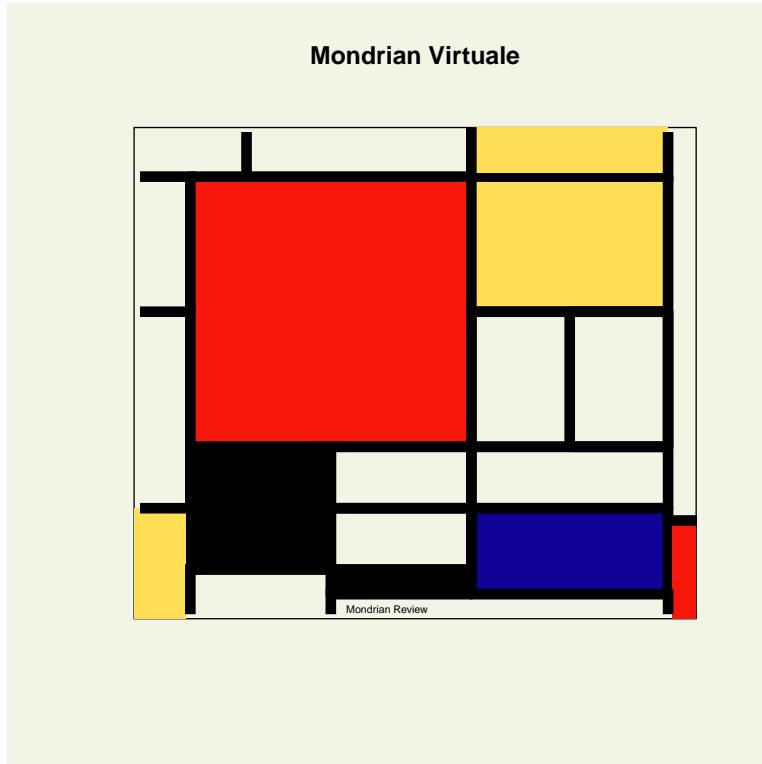
```
> plot(c(100, 200), c(300, 450), type= "n", xlab= "", ylab= "")
> rect(100, 300, 125, 350)
> rect(100, 400, 125, 450, col="green", border="blue")
> rect(115, 375, 150, 425, col=par("bg"), border="transparent")
> rect(150, 300, 175, 350, density=10, border="red")
> rect(150, 400, 175, 450, density=30, col="blue", angle=-30, border="transparent")
> legend(180, 450, legend=1:4, fill=c(NA, "green", par("fg"), "blue"), density=c(NA,
```



Ed ora, riproduciamo un Mondrian:

```
> par(bg=rgb(0.95,0.96,0.9))
> plot(NA,xlim=c(0,1),ylim=c(0,1),axes=F,xlab="",ylab="",main="Mondrian Virtuale")
> rect(0,0,1,1,col=rgb(0.95,0.96,0.9),lwd=1)
> rect(0.1,0.35,0.6,0.9,col=rgb(0.973,0.09,0.047),lwd=8)
> rect(0.6,0.625,0.95,0.9,col=rgb(1,0.87,0.34),lwd=8)
> rect(0.6,0.35,0.95,0.625,lwd=8) ##bianco
> rect(0.775,0.35,0.95,0.625,lwd=8) ##bianco 2
> rect(0.6,0.225,0.95,0.35,lwd=8) ##bianco 3
> rect(0.6,0.05,0.95,0.225,lwd=8,col=rgb(0.063,0.008,0.58))
> rect(0.35,0.225,0.6,0.35,lwd=8) #bianco sx 1
> rect(0.35,0.1,0.6,0.225,lwd=8) #bianco sx basso
> rect(0.35,0.05,0.6,0.1,lwd=8,col="black") ##nero sx basso
> rect(0.1,0.1,0.35,0.35,lwd=8,col="black") ##nero sx grande
> rect(0.6,0.908,0.95,1,lwd=NA,col=rgb(1,0.87,0.34)) #giallo alto dx
> rect(0.9575,0,1,0.2,lwd=NA,col=rgb(0.973,0.09,0.047)) #rosso basso dx
> rect(0,0,0.092,0.225,lwd=NA,col=rgb(1,0.87,0.34)) #giallo basso sx
> lines(c(0.02,0.1),c(0.225,0.225),lwd=8,lend="square")
```

```
> lines(c(0.02,0.1),c(0.625,0.625),lwd=8,lend="square")
> lines(c(0.02,0.1),c(0.9,0.9),lwd=8,lend="square")
> lines(c(0.2,0.2),c(0.9,0.98),lwd=8,lend="square")
> lines(c(0.95,0.95),c(0.9,0.98),lwd=8,lend="square")
> lines(c(0.6,0.6),c(0.9,0.99),lwd=8,lend="square")
> lines(c(0.95,0.992),c(0.2,0.2),lwd=8,lend="square")
> lines(c(0.95,0.95),c(0.05,0.02),lwd=8,lend="square")
> lines(c(0.35,0.35),c(0.05,0.02),lwd=8,lend="square")
> lines(c(0.1,0.1),c(0.1,0.02),lwd=8,lend="square")
> text(0.45,0.02,"Mondrian Review",cex=0.5)
```



Provate a confrontarlo con l'originale, cercatelo tra le immagini di Google (urlhttp://images.google.com)

Il comando `locator(n)` consente di selezionare con il puntatore n punti in un grafico. Selezioniamo 1 punto cliccando sul punto che riteniamo il massimo del grafico

```
> locator(1)->coord
```

ora clicchiamo sul massimo del grafico

```
> library(datasets)
> plot(as.vector(discoveries))
> coord= c(12,26)
> points(coord[2],coord[1], cex=3, col="red")
```

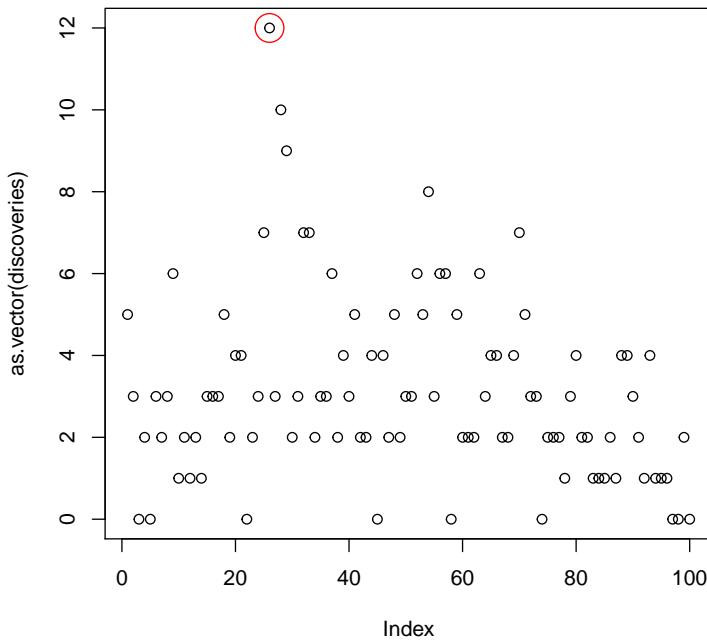


Tabelle delle distribuzioni statistiche

Per generare la tabella delle aree sottese dalla distribuzione normale da 0 ad x si deve per prima cosa tenere conto del fatto che `pnorm` ??????? la cumulativa ad una sola coda. Si sceglie l'intervallo di tabulazione, il numero di colonne ed il numero di cifre.

```
> start=0.01
> stop=3.00
> step=0.01;
> nc=6;
> cifre=5;
> correzione<-function(x) round(10^cifre* (pnorm(x)-0.5))/10^cifre
```

```
> tabnormale<-cbind(matrix(correzione(seq(start,stop,by=step)),nc=nc),
+ matrix(seq(start,stop,by=step),nc=nc))
> as.vector(t(matrix(c(nc+1:nc,1:nc),nc=2)))>->ordinecol
> colnames(tabnormale)=rep(c("P=A","x"),each=6)
> rownames(tabnormale)=rep("",nrow(tabnormale))
```

In modo simile per generare la tabella della distribuzione t di Student si selezionano i livelli di fiducia di interesse e i gradi di libertà.

```
> gradi=c(1:40,50,60,70,80,90,100,150,200,Inf)
> fiducia=c(0.8,0.85,0.9,0.95,0.98,0.99,0.999);
> ncol=length(fiducia);
> nrow=length(gradi);
> cifre=5;
> tstud<-function(x,gradi,cifre)
+ round(10^cifre*qt((1+x)/2,gradi))/10^cifre
> tabstudent=matrix(0,ncol=ncol,nrow=nrow)
> for (i in 1:length(fiducia))
+ tabstudent[,i]=tstud(fiducia[i],gradi,5)
> rownames(tabstudent)=gradi
> colnames(tabstudent)=fiducia
```

Per la distribuzione χ^2 si procede esattamente come sopra

```
> gradi=c(1:40,50,60,70,80,90,100,150,200)
> fiducia=c(0.8,0.85,0.9,0.95,0.98,0.99,0.999);
> ncol=length(fiducia);
> nrow=length(gradi);
> cifre=4;
> chiqua<-function(x,gradi,cifre)
+ round(10^cifre*qchisq(x,gradi))/10^cifre
> tabchi=matrix(0,ncol=ncol,nrow=nrow)
> for (i in 1:length(fiducia))
+ tabchi[,i]=chiqua(fiducia[i],gradi,5)
> rownames(tabchi)=gradi
> colnames(tabchi)=fiducia
```

Per la distribuzione di Fisher occorre specificare il numero di gradi di libertà del numeratore e del denominatore e fissare i valori di significatività. (0.05 e 0.01)

```
> gradinum=1:10
> gradiden=c(1:40,50,60,70,80,90,100,150,200)
> fiducia=c(0.95,0.99);
> ncol=length(gradinum);
> nrow=length(gradiden);
> cifre=3;
> fisher95<-function(gradinum,gradiden,cifre)
+ round(10^cifre*qf(fiducia[1],gradinum,gradiden))/10^cifre
> fisher095=matrix(0,ncol=ncol,nrow=nrow);
> for (i in 1:length(gradinum))
+ fisher095[,i]= fisher95(gradinum[i],gradiden,cifre)
> rownames(fisher095)=gradiden
> colnames(fisher095)=gradinum
> fisher099=fisher095;
> fisher99<-function(gradinum,gradiden,cifre)
+ round(10^cifre*qf(fiducia
+ [2],gradinum,gradiden))/10^cifre;for (i in 1:length(gradinum))
> fisher099[,i]= fisher99(gradinum[i],gradiden,cifre)
```

Aree A della distribuzione normale da 0 ad x

x	$P=A$										
0.01	0.00399	0.51	0.19497	1.01	0.34375	1.51	0.43448	2.01	0.47778	2.51	0.49396
0.02	0.00798	0.52	0.19847	1.02	0.34614	1.52	0.43574	2.02	0.47831	2.52	0.49413
0.03	0.01197	0.53	0.20194	1.03	0.34849	1.53	0.43699	2.03	0.47882	2.53	0.49430
0.04	0.01595	0.54	0.20540	1.04	0.35083	1.54	0.43822	2.04	0.47932	2.54	0.49446
0.05	0.01994	0.55	0.20884	1.05	0.35314	1.55	0.43943	2.05	0.47982	2.55	0.49461
0.06	0.02392	0.56	0.21226	1.06	0.35543	1.56	0.44062	2.06	0.48030	2.56	0.49477
0.07	0.02790	0.57	0.21566	1.07	0.35769	1.57	0.44179	2.07	0.48077	2.57	0.49492
0.08	0.03188	0.58	0.21904	1.08	0.35993	1.58	0.44295	2.08	0.48124	2.58	0.49506
0.09	0.03586	0.59	0.22240	1.09	0.36214	1.59	0.44408	2.09	0.48169	2.59	0.49520
0.10	0.03983	0.60	0.22575	1.10	0.36433	1.60	0.44520	2.10	0.48214	2.60	0.49534
0.11	0.04380	0.61	0.22907	1.11	0.36650	1.61	0.44630	2.11	0.48257	2.61	0.49547
0.12	0.04776	0.62	0.23237	1.12	0.36864	1.62	0.44738	2.12	0.48300	2.62	0.49560
0.13	0.05172	0.63	0.23565	1.13	0.37076	1.63	0.44845	2.13	0.48341	2.63	0.49573
0.14	0.05567	0.64	0.23891	1.14	0.37286	1.64	0.44950	2.14	0.48382	2.64	0.49585
0.15	0.05962	0.65	0.24215	1.15	0.37493	1.65	0.45053	2.15	0.48422	2.65	0.49598
0.16	0.06356	0.66	0.24537	1.16	0.37698	1.66	0.45154	2.16	0.48461	2.66	0.49609
0.17	0.06749	0.67	0.24857	1.17	0.37900	1.67	0.45254	2.17	0.48500	2.67	0.49621
0.18	0.07142	0.68	0.25175	1.18	0.38100	1.68	0.45352	2.18	0.48537	2.68	0.49632
0.19	0.07535	0.69	0.25490	1.19	0.38298	1.69	0.45449	2.19	0.48574	2.69	0.49643
0.20	0.07926	0.70	0.25804	1.20	0.38493	1.70	0.45543	2.20	0.48610	2.70	0.49653
0.21	0.08317	0.71	0.26115	1.21	0.38686	1.71	0.45637	2.21	0.48645	2.71	0.49664
0.22	0.08706	0.72	0.26424	1.22	0.38877	1.72	0.45728	2.22	0.48679	2.72	0.49674
0.23	0.09095	0.73	0.26730	1.23	0.39065	1.73	0.45818	2.23	0.48713	2.73	0.49683
0.24	0.09483	0.74	0.27035	1.24	0.39251	1.74	0.45907	2.24	0.48745	2.74	0.49693
0.25	0.09871	0.75	0.27337	1.25	0.39435	1.75	0.45994	2.25	0.48778	2.75	0.49702
0.26	0.10257	0.76	0.27637	1.26	0.39617	1.76	0.46080	2.26	0.48809	2.76	0.49711
0.27	0.10642	0.77	0.27935	1.27	0.39796	1.77	0.46164	2.27	0.48840	2.77	0.49720
0.28	0.11026	0.78	0.28230	1.28	0.39973	1.78	0.46246	2.28	0.48870	2.78	0.49728
0.29	0.11409	0.79	0.28524	1.29	0.40147	1.79	0.46327	2.29	0.48899	2.79	0.49736
0.30	0.11791	0.80	0.28814	1.30	0.40320	1.80	0.46407	2.30	0.48928	2.80	0.49744
0.31	0.12172	0.81	0.29103	1.31	0.40490	1.81	0.46485	2.31	0.48956	2.81	0.49752
0.32	0.12552	0.82	0.29389	1.32	0.40658	1.82	0.46562	2.32	0.48983	2.82	0.49760
0.33	0.12930	0.83	0.29673	1.33	0.40824	1.83	0.46638	2.33	0.49010	2.83	0.49767
0.34	0.13307	0.84	0.29955	1.34	0.40988	1.84	0.46712	2.34	0.49036	2.84	0.49774
0.35	0.13683	0.85	0.30234	1.35	0.41149	1.85	0.46784	2.35	0.49061	2.85	0.49781
0.36	0.14058	0.86	0.30511	1.36	0.41309	1.86	0.46856	2.36	0.49086	2.86	0.49788
0.37	0.14431	0.87	0.30785	1.37	0.41466	1.87	0.46926	2.37	0.49111	2.87	0.49795
0.38	0.14803	0.88	0.31057	1.38	0.41621	1.88	0.46995	2.38	0.49134	2.88	0.49801
0.39	0.15173	0.89	0.31327	1.39	0.41774	1.89	0.47062	2.39	0.49158	2.89	0.49807
0.40	0.15542	0.90	0.31594	1.40	0.41924	1.90	0.47128	2.40	0.49180	2.90	0.49813
0.41	0.15910	0.91	0.31859	1.41	0.42073	1.91	0.47193	2.41	0.49202	2.91	0.49819
0.42	0.16276	0.92	0.32121	1.42	0.42220	1.92	0.47257	2.42	0.49224	2.92	0.49825
0.43	0.16640	0.93	0.32381	1.43	0.42364	1.93	0.47320	2.43	0.49245	2.93	0.49831
0.44	0.17003	0.94	0.32639	1.44	0.42507	1.94	0.47381	2.44	0.49266	2.94	0.49836
0.45	0.17364	0.95	0.32894	1.45	0.42647	1.95	0.47441	2.45	0.49286	2.95	0.49841
0.46	0.17724	0.96	0.33147	1.46	0.42785	1.96	0.47500	2.46	0.49305	2.96	0.49846
0.47	0.18082	0.97	0.33398	1.47	0.42922	1.97	0.47558	2.47	0.49324	2.97	0.49851
0.48	0.18439	0.98	0.33646	1.48	0.43056	1.98	0.47615	2.48	0.49343	2.98	0.49856
0.49	0.18793	0.99	0.33891	1.49	0.43189	1.99	0.47670	2.49	0.49361	2.99	0.49861
0.50	0.19146	1.00	0.34134	1.50	0.43319	2.00	0.47725	2.50	0.49379	3.00	0.49865

Distribuzione di Student

	0.8	0.85	0.9	0.95	0.98	0.99	0.999
1	3.07768	4.16530	6.31375	12.70620	31.82052	63.65674	636.61925
2	1.88562	2.28193	2.91999	4.30265	6.96456	9.92484	31.59905
3	1.63774	1.92432	2.35336	3.18245	4.54070	5.84091	12.92398
4	1.53321	1.77819	2.13185	2.77645	3.74695	4.60409	8.61030
5	1.47588	1.69936	2.01505	2.57058	3.36493	4.03214	6.86883
6	1.43976	1.65017	1.94318	2.44691	3.14267	3.70743	5.95882
7	1.41492	1.61659	1.89458	2.36462	2.99795	3.49948	5.40788
8	1.39682	1.59222	1.85955	2.30600	2.89646	3.35539	5.04131
9	1.38303	1.57374	1.83311	2.26216	2.82144	3.24984	4.78091
10	1.37218	1.55924	1.81246	2.22814	2.76377	3.16927	4.58689
11	1.36343	1.54756	1.79588	2.20099	2.71808	3.10581	4.43698
12	1.35622	1.53796	1.78229	2.17881	2.68100	3.05454	4.31779
13	1.35017	1.52992	1.77093	2.16037	2.65031	3.01228	4.22083
14	1.34503	1.52310	1.76131	2.14479	2.62449	2.97684	4.14045
15	1.34061	1.51723	1.75305	2.13145	2.60248	2.94671	4.07277
16	1.33676	1.51213	1.74588	2.11991	2.58349	2.92078	4.01500
17	1.33338	1.50766	1.73961	2.10982	2.56693	2.89823	3.96513
18	1.33039	1.50371	1.73406	2.10092	2.55238	2.87844	3.92165
19	1.32773	1.50019	1.72913	2.09302	2.53948	2.86093	3.88341
20	1.32534	1.49704	1.72472	2.08596	2.52798	2.84534	3.84952
21	1.32319	1.49419	1.72074	2.07961	2.51765	2.83136	3.81928
22	1.32124	1.49162	1.71714	2.07387	2.50832	2.81876	3.79213
23	1.31946	1.48928	1.71387	2.06866	2.49987	2.80734	3.76763
24	1.31784	1.48714	1.71088	2.06390	2.49216	2.79694	3.74540
25	1.31635	1.48517	1.70814	2.05954	2.48511	2.78744	3.72514
26	1.31497	1.48336	1.70562	2.05553	2.47863	2.77871	3.70661
27	1.31370	1.48169	1.70329	2.05183	2.47266	2.77068	3.68959
28	1.31253	1.48014	1.70113	2.04841	2.46714	2.76326	3.67391
29	1.31143	1.47870	1.69913	2.04523	2.46202	2.75639	3.65941
30	1.31042	1.47736	1.69726	2.04227	2.45726	2.75000	3.64596
31	1.30946	1.47611	1.69552	2.03951	2.45282	2.74404	3.63346
32	1.30857	1.47494	1.69389	2.03693	2.44868	2.73848	3.62180
33	1.30774	1.47384	1.69236	2.03452	2.44479	2.73328	3.61091
34	1.30695	1.47281	1.69092	2.03224	2.44115	2.72839	3.60072
35	1.30621	1.47184	1.68957	2.03011	2.43772	2.72381	3.59115
36	1.30551	1.47092	1.68830	2.02809	2.43449	2.71948	3.58215
37	1.30485	1.47005	1.68709	2.02619	2.43145	2.71541	3.57367
38	1.30423	1.46923	1.68595	2.02439	2.42857	2.71156	3.56568
39	1.30364	1.46846	1.68488	2.02269	2.42584	2.70791	3.55812
40	1.30308	1.46772	1.68385	2.02108	2.42326	2.70446	3.55097
50	1.29871	1.46199	1.67591	2.00856	2.40327	2.67779	3.49601
60	1.29582	1.45820	1.67065	2.00030	2.39012	2.66028	3.46020
70	1.29376	1.45550	1.66691	1.99444	2.38081	2.64790	3.43501
80	1.29222	1.45349	1.66412	1.99006	2.37387	2.63869	3.41634
90	1.29103	1.45192	1.66196	1.98667	2.36850	2.63157	3.40194
100	1.29007	1.45067	1.66023	1.98397	2.36422	2.62589	3.39049
150	1.28722	1.44694	1.65508	1.97591	2.35146	2.60900	3.35657
200	1.28580	1.44508	1.65251	1.97190	2.34514	2.60063	3.33984
Inf	1.28155	1.43953	1.64485	1.95996	2.32635	2.57583	3.29053

Distribuzione χ^2

> tabchi

	0.8	0.85	0.9	0.95	0.98	0.99	0.999
1	1.64237	2.07225	2.70554	3.84146	5.41189	6.63490	10.82757
2	3.21888	3.79424	4.60517	5.99146	7.82405	9.21034	13.81551
3	4.64163	5.31705	6.25139	7.81473	9.83741	11.34487	16.26624
4	5.98862	6.74488	7.77944	9.48773	11.66784	13.27670	18.46683
5	7.28928	8.11520	9.23636	11.07050	13.38822	15.08627	20.51501
6	8.55806	9.44610	10.64464	12.59159	15.03321	16.81189	22.45774
7	9.80325	10.74790	12.01704	14.06714	16.62242	18.47531	24.32189
8	11.03009	12.02707	13.36157	15.50731	18.16823	20.09024	26.12448
9	12.24215	13.28804	14.68366	16.91898	19.67902	21.66599	27.87716
10	13.44196	14.53394	15.98718	18.30704	21.16077	23.20925	29.58830
11	14.63142	15.76710	17.27501	19.67514	22.61794	24.72497	31.26413
12	15.81199	16.98931	18.54935	21.02607	24.05396	26.21697	32.90949
13	16.98480	18.20198	19.81193	22.36203	25.47151	27.68825	34.52818
14	18.15077	19.40624	21.06414	23.68479	26.87276	29.14124	36.12327
15	19.31066	20.60301	22.30713	24.99579	28.25950	30.57791	37.69730
16	20.46508	21.79306	23.54183	26.29623	29.63318	31.99993	39.25235
17	21.61456	22.97703	24.76904	27.58711	30.99505	33.40866	40.79022
18	22.75955	24.15547	25.98942	28.86930	32.34616	34.80531	42.31240
19	23.90042	25.32885	27.20357	30.14353	33.68743	36.19087	43.82020
20	25.03751	26.49758	28.41198	31.41043	35.01963	37.56623	45.31475
21	26.17110	27.66201	29.61509	32.67057	36.34345	38.93217	46.79704
22	27.30145	28.82245	30.81328	33.92444	37.65950	40.28936	48.26794
23	28.42879	29.97919	32.00690	35.17246	38.96831	41.63840	49.72823
24	29.55332	31.13246	33.19624	36.41503	40.27036	42.97982	51.17860
25	30.67520	32.28249	34.38159	37.65248	41.56607	44.31410	52.61966
26	31.79461	33.42947	35.56317	38.88514	42.85583	45.64168	54.05196
27	32.91169	34.57358	36.74122	40.11327	44.13999	46.96294	55.47602
28	34.02657	35.71499	37.91592	41.33714	45.41885	48.27824	56.89229
29	35.13936	36.85383	39.08747	42.55697	46.69270	49.58788	58.30117
30	36.25019	37.99025	40.25602	43.77297	47.96180	50.89218	59.70306
31	37.35914	39.12437	41.42174	44.98534	49.22640	52.19139	61.09831
32	38.46631	40.25630	42.58475	46.19426	50.48670	53.48577	62.48722
33	39.57179	41.38614	43.74518	47.39988	51.74292	54.77554	63.87010
34	40.67565	42.51399	44.90316	48.60237	52.99524	56.06091	65.24722
35	41.77796	43.63994	46.05879	49.80185	54.24383	57.34207	66.61883
36	42.87880	44.76407	47.21217	50.99846	55.48886	58.61921	67.98517
37	43.97822	45.88645	48.36341	52.19232	56.73047	59.89250	69.34645
38	45.07628	47.00717	49.51258	53.38354	57.96880	61.16209	70.70289
39	46.17303	48.12628	50.65977	54.57223	59.20398	62.42812	72.05466
40	47.26854	49.24385	51.80506	55.75848	60.43613	63.69074	73.40196
50	58.16380	60.34599	63.16712	67.50481	72.61325	76.15389	86.66082
60	68.97207	71.34110	74.39701	79.08194	84.57995	88.37942	99.60723
70	79.71465	82.25535	85.52704	90.53123	96.38754	100.42518	112.31693
80	90.40535	93.10575	96.57820	101.87947	108.06934	112.32879	124.83922
90	101.05372	103.90406	107.56501	113.14527	119.64846	124.11632	137.20835
100	111.66671	114.65882	118.49800	124.34211	131.14168	135.80672	149.44925
150	164.34919	167.96177	172.58121	179.58063	187.67850	193.20769	209.26460
200	216.60878	220.74413	226.02105	233.99427	243.18692	249.44512	267.54053

Tabella della distribuzione di Fisher 95%

> fisher095

	1	2	3	4	5	6	7	8	9
1	161.448	199.500	215.707	224.583	230.162	233.986	236.768	238.883	240.543
2	18.513	19.000	19.164	19.247	19.296	19.330	19.353	19.371	19.385
3	10.128	9.552	9.277	9.117	9.013	8.941	8.887	8.845	8.812
4	7.709	6.944	6.591	6.388	6.256	6.163	6.094	6.041	5.999
5	6.608	5.786	5.409	5.192	5.050	4.950	4.876	4.818	4.772
6	5.987	5.143	4.757	4.534	4.387	4.284	4.207	4.147	4.099
7	5.591	4.737	4.347	4.120	3.972	3.866	3.787	3.726	3.677
8	5.318	4.459	4.066	3.838	3.687	3.581	3.500	3.438	3.388
9	5.117	4.256	3.863	3.633	3.482	3.374	3.293	3.230	3.179
10	4.965	4.103	3.708	3.478	3.326	3.217	3.135	3.072	3.020
11	4.844	3.982	3.587	3.357	3.204	3.095	3.012	2.948	2.896
12	4.747	3.885	3.490	3.259	3.106	2.996	2.913	2.849	2.796
13	4.667	3.806	3.411	3.179	3.025	2.915	2.832	2.767	2.714
14	4.600	3.739	3.344	3.112	2.958	2.848	2.764	2.699	2.646
15	4.543	3.682	3.287	3.056	2.901	2.790	2.707	2.641	2.588
16	4.494	3.634	3.239	3.007	2.852	2.741	2.657	2.591	2.538
17	4.451	3.592	3.197	2.965	2.810	2.699	2.614	2.548	2.494
18	4.414	3.555	3.160	2.928	2.773	2.661	2.577	2.510	2.456
19	4.381	3.522	3.127	2.895	2.740	2.628	2.544	2.477	2.423
20	4.351	3.493	3.098	2.866	2.711	2.599	2.514	2.447	2.393
21	4.325	3.467	3.072	2.840	2.685	2.573	2.488	2.420	2.366
22	4.301	3.443	3.049	2.817	2.661	2.549	2.464	2.397	2.342
23	4.279	3.422	3.028	2.796	2.640	2.528	2.442	2.375	2.320
24	4.260	3.403	3.009	2.776	2.621	2.508	2.423	2.355	2.300
25	4.242	3.385	2.991	2.759	2.603	2.490	2.405	2.337	2.282
26	4.225	3.369	2.975	2.743	2.587	2.474	2.388	2.321	2.265
27	4.210	3.354	2.960	2.728	2.572	2.459	2.373	2.305	2.250
28	4.196	3.340	2.947	2.714	2.558	2.445	2.359	2.291	2.236
29	4.183	3.328	2.934	2.701	2.545	2.432	2.346	2.278	2.223
30	4.171	3.316	2.922	2.690	2.534	2.421	2.334	2.266	2.211
31	4.160	3.305	2.911	2.679	2.523	2.409	2.323	2.255	2.199
32	4.149	3.295	2.901	2.668	2.512	2.399	2.313	2.244	2.189
33	4.139	3.285	2.892	2.659	2.503	2.389	2.303	2.235	2.179
34	4.130	3.276	2.883	2.650	2.494	2.380	2.294	2.225	2.170
35	4.121	3.267	2.874	2.641	2.485	2.372	2.285	2.217	2.161
36	4.113	3.259	2.866	2.634	2.477	2.364	2.277	2.209	2.153
37	4.105	3.252	2.859	2.626	2.470	2.356	2.270	2.201	2.145
38	4.098	3.245	2.852	2.619	2.463	2.349	2.262	2.194	2.138
39	4.091	3.238	2.845	2.612	2.456	2.342	2.255	2.187	2.131
40	4.085	3.232	2.839	2.606	2.449	2.336	2.249	2.180	2.124
50	4.034	3.183	2.790	2.557	2.400	2.286	2.199	2.130	2.073
60	4.001	3.150	2.758	2.525	2.368	2.254	2.167	2.097	2.040
70	3.978	3.128	2.736	2.503	2.346	2.231	2.143	2.074	2.017
80	3.960	3.111	2.719	2.486	2.329	2.214	2.126	2.056	1.999
90	3.947	3.098	2.706	2.473	2.316	2.201	2.113	2.043	1.986
100	3.936	3.087	2.696	2.463	2.305	2.191	2.103	2.032	1.975
150	3.904	3.056	2.665	2.432	2.274	2.160	2.071	2.001	1.943
200	3.888	3.041	2.650	2.417	2.259	2.144	2.056	1.985	1.927

1	241.882
2	19.396
3	8.786
4	5.964
5	4.735
6	4.060
7	3.637
8	3.347
9	3.137
10	2.978
11	2.854
12	2.753
13	2.671
14	2.602
15	2.544
16	2.494
17	2.450
18	2.412
19	2.378
20	2.348
21	2.321
22	2.297
23	2.275
24	2.255
25	2.236
26	2.220
27	2.204
28	2.190
29	2.177
30	2.165
31	2.153
32	2.142
33	2.133
34	2.123
35	2.114
36	2.106
37	2.098
38	2.091
39	2.084
40	2.077
50	2.026
60	1.993
70	1.969
80	1.951
90	1.938
100	1.927
150	1.894
200	1.878

>

Tabella della distribuzione di Fisher 99%

> fisher099

	1	2	3	4	5	6	7	8
1	4052.181	4999.500	5403.352	5624.583	5763.650	5858.986	5928.356	5981.070
2	98.503	99.000	99.166	99.249	99.299	99.333	99.356	99.374
3	34.116	30.817	29.457	28.710	28.237	27.911	27.672	27.489
4	21.198	18.000	16.694	15.977	15.522	15.207	14.976	14.799
5	16.258	13.274	12.060	11.392	10.967	10.672	10.456	10.289
6	13.745	10.925	9.780	9.148	8.746	8.466	8.260	8.102
7	12.246	9.547	8.451	7.847	7.460	7.191	6.993	6.840
8	11.259	8.649	7.591	7.006	6.632	6.371	6.178	6.029
9	10.561	8.022	6.992	6.422	6.057	5.802	5.613	5.467
10	10.044	7.559	6.552	5.994	5.636	5.386	5.200	5.057
11	9.646	7.206	6.217	5.668	5.316	5.069	4.886	4.744
12	9.330	6.927	5.953	5.412	5.064	4.821	4.640	4.499
13	9.074	6.701	5.739	5.205	4.862	4.620	4.441	4.302
14	8.862	6.515	5.564	5.035	4.695	4.456	4.278	4.140
15	8.683	6.359	5.417	4.893	4.556	4.318	4.142	4.004
16	8.531	6.226	5.292	4.773	4.437	4.202	4.026	3.890
17	8.400	6.112	5.185	4.669	4.336	4.102	3.927	3.791
18	8.285	6.013	5.092	4.579	4.248	4.015	3.841	3.705
19	8.185	5.926	5.010	4.500	4.171	3.939	3.765	3.631
20	8.096	5.849	4.938	4.431	4.103	3.871	3.699	3.564
21	8.017	5.780	4.874	4.369	4.042	3.812	3.640	3.506
22	7.945	5.719	4.817	4.313	3.988	3.758	3.587	3.453
23	7.881	5.664	4.765	4.264	3.939	3.710	3.539	3.406
24	7.823	5.614	4.718	4.218	3.895	3.667	3.496	3.363
25	7.770	5.568	4.675	4.177	3.855	3.627	3.457	3.324
26	7.721	5.526	4.637	4.140	3.818	3.591	3.421	3.288
27	7.677	5.488	4.601	4.106	3.785	3.558	3.388	3.256
28	7.636	5.453	4.568	4.074	3.754	3.528	3.358	3.226
29	7.598	5.420	4.538	4.045	3.725	3.499	3.330	3.198
30	7.562	5.390	4.510	4.018	3.699	3.473	3.304	3.173
31	7.530	5.362	4.484	3.993	3.675	3.449	3.281	3.149
32	7.499	5.336	4.459	3.969	3.652	3.427	3.258	3.127
33	7.471	5.312	4.437	3.948	3.630	3.406	3.238	3.106
34	7.444	5.289	4.416	3.927	3.611	3.386	3.218	3.087
35	7.419	5.268	4.396	3.908	3.592	3.368	3.200	3.069
36	7.396	5.248	4.377	3.890	3.574	3.351	3.183	3.052
37	7.373	5.229	4.360	3.873	3.558	3.334	3.167	3.036
38	7.353	5.211	4.343	3.858	3.542	3.319	3.152	3.021
39	7.333	5.194	4.327	3.843	3.528	3.305	3.137	3.006
40	7.314	5.179	4.313	3.828	3.514	3.291	3.124	2.993
50	7.171	5.057	4.199	3.720	3.408	3.186	3.020	2.890
60	7.077	4.977	4.126	3.649	3.339	3.119	2.953	2.823
70	7.011	4.922	4.074	3.600	3.291	3.071	2.906	2.777
80	6.963	4.881	4.036	3.563	3.255	3.036	2.871	2.742
90	6.925	4.849	4.007	3.535	3.228	3.009	2.845	2.715
100	6.895	4.824	3.984	3.513	3.206	2.988	2.823	2.694
150	6.807	4.749	3.915	3.447	3.142	2.924	2.761	2.632
200	6.763	4.713	3.881	3.414	3.110	2.893	2.730	2.601

1	6022.473	6055.847
2	99.388	99.399
3	27.345	27.229
4	14.659	14.546
5	10.158	10.051
6	7.976	7.874
7	6.719	6.620
8	5.911	5.814
9	5.351	5.257
10	4.942	4.849
11	4.632	4.539
12	4.388	4.296
13	4.191	4.100
14	4.030	3.939
15	3.895	3.805
16	3.780	3.691
17	3.682	3.593
18	3.597	3.508
19	3.523	3.434
20	3.457	3.368
21	3.398	3.310
22	3.346	3.258
23	3.299	3.211
24	3.256	3.168
25	3.217	3.129
26	3.182	3.094
27	3.149	3.062
28	3.120	3.032
29	3.092	3.005
30	3.067	2.979
31	3.043	2.955
32	3.021	2.934
33	3.000	2.913
34	2.981	2.894
35	2.963	2.876
36	2.946	2.859
37	2.930	2.843
38	2.915	2.828
39	2.901	2.814
40	2.888	2.801
50	2.785	2.698
60	2.718	2.632
70	2.672	2.585
80	2.637	2.551
90	2.611	2.524
100	2.590	2.503
150	2.528	2.441
200	2.497	2.411

Bibliografia

- [DASL] The data and Stories Library). <http://lib.stat.cmu.edu/DASL>
- [Dalgaard] P.Dalgaard). *Introductory Statistics with R* , Springer,
- [Snedecor-Cochran] George W. Snedecor , William G. Cochran,Statistical Methods
Iowa State University Press; 8 edition (January 15, 1989)
- [David] David F. M., (1955), Studies in the History of Probability and Statistics I.
Dicing and Gaming (A Note on the History of Probability). *Biometrika Trust*,
42, 1–15.
- [Murrell] Paul Murrell, *R graphics*, Chapman& Hall/CRC

Indice analitico

escape, 87
LETTERS, 88
#, 4
%*%, prodotto di matrici, 93
abbreviate, 87
cbind, 97
chisq.tst,test χ^2 , 82, 116
cloud, 161
colnames, 91
console, 3
c, 10
dbinom, 83, 117
diff, 134
getwd(), 6
grid, 129
help, 4
history, 3
las, 169
length, 11
letters, 88
level, 96
list, 101
locator, 171
month.abb, 88
nchar, 87
par3D, 155
paste, 88
pi, 9
plot, 133
rbind, 97
replicates, 113
rm(), 8
rownames, 91
seq, 12
setwd(), 6
strip, 161
str, 86
surface3d, 155
tapply, 103
t, trasposto, 92
autovalore, 93
browseURL, 153
detach, 97
pairs, 138
stringa, 86
trasposto, 92