Introduzione ad R

per il corso di

Probabilità e Statistica (Informatica)

Federica Giummolè
giummole@unive.it

Anno Accademico 2019-2020

Indice

Capitolo 1

Introduzione a R

1.1 Aprire e chiudere una sessione di R

Per iniziare una sessione R fare un doppio click di mouse sull'icona di R.
Per uscire da R usare `q()`. Per salvare i dati rispondere `y`, altrimenti rispondere `n`.
Per controllare cosa c'è disponibile nella directory di lavoro, chiamata anche *workspace*, si usa il comando:

```
> ls()
```

```
[1] "a"  "tf" "x"  "xx" "y"  "yy" "z"
```

Supponendo che sia presente un oggetto di nome `thing`, è possibile eliminarlo con il comando `rm()`

```
> rm(thing)
```

A questo punto, l'oggetto di nome `thing` non sarà più presente nel *workspace*

```
> thing
```

```
Error:  Object thing not found
```

Quando si inizia una nuova sessione di lavoro, è opportuno rimuovere tutti i vecchi oggetti che non servono. Un comando utile è:

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

o, in alternativa, `rm(list=objects())`.

1.2 Lavorare con RStudio

RStudio è un'interfaccia che rende più facile l'uso di R.
Per aprire RStudio fare un doppio click con il mouse sull'icona di RStudio.
Ci sono 4 finestre in RStudio.

- La finestra in alto a sinistra si chiama *Source*. Per creare un nuovo documento, selezionare dal menù *File*, poi *New File* e infine scegliere il tipo di documento che si intende creare. Si aprirà un *template* che vi aiuterà ad iniziare. Nel nostro corso utilizzeremo in particolare documenti che contengono codice R (estensione .R) e documenti che mescolano insieme parti di codice R e parti di testo (estensione .Rmd).

- La finestra in basso a sinistra è la *Console*. E' il cervello di R. Ogni cosa inviata alla *Console* viene eseguita da R.

- La finestra in alto a destra ha due importanti etichette: *Environment* e *History*. Ogni comando scritto nella *Console* viene salvato in *History* in modo da poter agevolmente essere richiamato. *Environment* mostra l'area di lavoro, *workspace*, e gli oggetti, le funzioni e i dataset creati e salvati durante la sessione di lavoro.

- La finestra in basso a destra ha quattro importanti etichette:
  - *Files* consente di caricare documenti in RStudio;
  - *Plots* mostra i grafici creati dalla *Console*;
  - *Packages* permette di arricchire la versione base di R installando nuovi *packages*;
  - *Help* mostra le pagine dell'*help*.

1.3 Documenti *RScript* e *RMarkdown*

E' spesso utile salvare i comandi di una sessione di lavoro in un file di testo, detto *RScript* con estensione .R. Sarà comodo per richiamare i comandi già eseguiti e per riprendere il lavoro in una prossima sessione.
RMarkdown è un *package* che può essere installato in R. Serve per generare report di alta qualità mescolando insieme parti di testo e parti di codice che vengono eseguite da R. Un documento *RMarkdown* ha estensione .Rmd e viene compilato in RStudio usando il tasto *Knit* della finestra in alto a sinistra. Il risultato è un documento HTML, Word o PDF, in cui sono presenti le parti di testo e i risultati delle parti di codice compilate in R.

Esercizio: Utilizzando un *template* di RStudio, salvate nel vostro computer un documento *RMarkdown* e provate a compilarlo ottenendo il corrispondente documento HTML .

1.4 Semplice aritmetica

In R, qualunque cosa venga scritta al prompt della *Console* viene valutata:

```
> 1+2+3
```

```
[1] 6

> 2+3*4

[1] 14

> 3/2+1

[1] 2.5

> 2+(3*4)

[1] 14

> (2 + 3) * 4

[1] 20

> 4*3**3 #Usa ** o ^ per calcolare un elevamento a potenza.

[1] 108
```

R fornisce anche tutte le funzioni che si trovano su un calcolatore tascabile:

```
> sqrt(2)

[1] 1.414214

> sin(3.14159)  #sin(Pi greco) e' zero

[1] 2.65359e-06
```

Il seno di π è zero e questo è vicino. R fornisce anche il valore di π:

```
> sin(pi)

[1] 1.224647e-16
```

che è molto più vicino a zero.
Le funzioni possono essere annidate:

```
> sqrt(sin(45*pi/180))

[1] 0.8408964
```

In realtà R possiede un gran numero di funzioni, anzi proprio la ricchezza di funzioni e la possibilità di incrementarne il numero costituiscono uno dei punti di forza del linguaggio. Per chiedere aiuto su di una funzione si può digitare

```
> ?sqrt
```

ma se non si sa se esiste una funzione particolare è possibile ricorrere ad un aiuto più generale

```
> help.start()
```

1.5 Assegnazioni di valori

Si può salvare un valore assegnandolo ad un oggetto mediante il simbolo `<-` oppure il simbolo `=`

```
> x <- sqrt(2)  #salva in x la radice quadrata di 2
> x

[1] 1.414214

> x**3

[1] 2.828427

> log(x)->y
> y

[1] 0.3465736

> z=x+y
> z

[1] 1.760787
```

1.6 Valori logici

R permette di gestire operazioni e variabili logiche:

```
> x <- 10         #fissa x uguale a 10
> x > 10        #  x e' piu' grande di 10?

[1] FALSE

> x <= 10

[1] TRUE

> tf <- x > 10
> tf

[1] FALSE

> FALSE

[1] FALSE
```

Gli operatori logici sono `<`, `<=`, `>`, `>=`, `==` per l'uguale e `!=` per il diverso. Inoltre, se `a1` e `a2` sono espressioni logiche, allora `a1 & a2` rappresenta l'intersezione, `a1 | a2` rappresenta l'unione e `!a1` è la negazione di `a1`. Per i dettagli, vedere ad esempio `help(&)`.

1.7 Vettori

1.7.1 Creazione di vettori

Per creare un vettore, si usa la funzione `c()`:

```
> x <- c(2,3,5,7,11)
> x

[1]  2  3  5  7 11
```

1.7.2 Successioni

Si può usare la notazione `a:b` per creare vettori che sono sequenze di numeri interi:

```
> xx <- 1:10
> xx

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

> xx <- 100:1
> xx

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

La stessa operazione può essere fatta con:

```
> xx<-seq(from=100, to=1)
> xx

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

Per creare sequenze di numeri equispaziati si può utilizzare l'opzione `by`:

```
> seq(0,1, by=0.1)

 [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

Possono anche essere creati dei vettori che contengono elementi ripetuti

```
> rep(2,times=3)

[1] 2 2 2

> rep(2,3)

[1] 2 2 2

> a<-c(rep(2,3),4,5,rep(1,5))
> a

 [1] 2 2 2 4 5 1 1 1 1 1
```

Ai vettori può essere applicata la stessa aritmetica di base che è stata applicata ai valori scalari:

```
> x <- 1:10
> x*2

 [1]  2  4  6  8 10 12 14 16 18 20

> x * x

 [1]   1   4   9  16  25  36  49  64  81 100
```

Possono essere eseguite operazioni logiche anche sui vettori

```
> x > 5

 [1] FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE
 [8]  TRUE  TRUE  TRUE
```

1.7.3 Estrazione di elementi da un vettore

Gli elementi di un vettore possono essere estratti usando le parentesi quadre [ ]:

```
> xx[7]

[1] 94
```

Si possono estrarre anche sottoinsiemi di elementi:

```
> xx[c(2,3,5,7,11)]

[1] 99 98 96 94 90

> xx[85:91]

[1] 16 15 14 13 12 11 10

> xx[91:85]

[1] 10 11 12 13 14 15 16

> xx[c(1:5,8:10)]

[1] 100  99  98  97  96  93  92  91

> xx[c(1,1,1,1,2,2,2,2)]

[1] 100 100 100 100  99  99  99  99
```

Ovviamente, sottoinsiemi di elementi possono essere salvati in nuovi vettori:

```
> yy <- xx[c(1,2,4,8,16,32,64)]
> yy

[1] 100  99  97  93  85  69  37
```

Se le parentesi quadre racchiudono un numero negativo, l'elemento corrispondente viene omesso dal vettore risultante:

```
> x <- c(1,2,4,8,16,32)
> x

[1]  1  2  4  8 16 32

> x[-4]

[1]  1  2  4 16 32
```

All'interno delle parentesi quadre possiamo anche inserire il risultato di un'operazione logica. Saranno estratte solo le componenti del vettore che soddisfano la condizione:

```
> x

[1]  1  2  4  8 16 32

> x[x>4]

[1]  8 16 32
```

Alcune funzioni utili per la manipolazione di vettori sono le seguenti:

```
> x <- 26:3
> x

 [1] 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12
[16] 11 10  9  8  7  6  5  4  3

> length(x)

[1] 24

> max(x)

[1] 26

> min(x)

[1] 3

> sum(x)

[1] 348

> prod(x)

[1] 2.016457e+26
```

Esercizio: Porre n = 5 + il proprio giorno di nascita. Creare un vettore dei multipli di n compresi fra 1253 e 2037.

1.8 Matrici

R consente anche di usare le matrici:

```
> x <- matrix(c(2,3,5,7,11,13),ncol=2)
> x

     [,1] [,2]
[1,]    2    7
[2,]    3   11
[3,]    5   13
```

NB: Bisogna specificare `nrow` o `ncol` per comunicare a R la dimensione della matrice.
Per estrarre un elemento da una matrice, bisogna specificarne le due coordinate:

```
> x[2,1]
[1] 3
> x[2,2]
[1] 11
```

Se non si mette una delle coordinate, si ottiene una intera riga/colonna:

```
> x[,1]

[1] 2 3 5

> x[3,]

[1]  5 13
```

Possono essere estratti sottoinsiemi di righe e/o colonne:

```
> x <- matrix(1:16,ncol=4)
> x

     [,1] [,2] [,3] [,4]
[1,]    1    5    9   13
[2,]    2    6   10   14
[3,]    3    7   11   15
[4,]    4    8   12   16

> x[c(1,4),c(3,4)]

     [,1] [,2]
[1,]    9   13
[2,]   12   16
```

La funzione `dim()` indica la dimensione (numero di righe e numero di colonne) della matrice:

```
> dim(x)

[1] 4 4
```

1.9 Packages

Un sistema di *packages* estende le funzionalità di base di R. Alcuni di questi sono automaticamente caricati in ogni sessione di lavoro e sono pronti per l'uso. Altri devono essere prima installati (una sola volta) e poi caricati (ad ogni sessione di lavoro).
Per installare un *package* si utilizza il tasto *Install* dell'etichetta *Packages* della finestra in basso a destra di RStudio, e si seguono le istruzioni.
Per caricare un *package* per la sessione di lavoro, si usa il comando *library()* o *require()*:

```
> library(tigerstats)
```

Una volta caricato, un *package* rimane disponibile per tutta la sessione di lavoro.

Esercizio: Installare il *package tigerstats* e tutti i *packages* collegati. Controllare che siano stati caricati anche *mosaic*, *mosaicData* e *abd*.

Capitolo 2

Probabilità

Da qui in avanti ci occuperemo di Probabilità. In particolare in questo capitolo tratteremo le distribuzioni di probabilità più comuni, il loro utilizzo e la simulazione di variabili (pseudo) casuali.
Per iniziare, è utile illustrare alcune funzioni di R per effettuare conteggi di calcolo combinatorio.

2.1 Calcolo combinatorio

Il fattoriale di un numero n, indicato con $n!$ cioè il prodotto di un intero positivo n per tutti gli interi positivi più piccoli di questo fino ad arrivare all'1, ovvero

$$n \times (n-1) \times (n-2) \times (n-3) \times \cdots \times 1,$$

si ottiene utilizzando la funzione `prod(1:n)` oppure la funzione `factorial(n)`. Ad esempio, 5!=120 si calcola con

```
> prod(1:5)

[1] 120

> factorial(5)

[1] 120
```

In modo simile si può calcolare il numero di disposizioni semplici di n oggetti a gruppi di k, $D_{n,k}$, ovvero il prodotto di un intero positivo n per i primi $(k-1)$ interi positivi più piccoli di questo. Come sappiamo $D_{n,k}$ fornisce tutti i gruppi che si possono formare prendendo k tra n oggetti distinti, in modo che ogni gruppo differisca dai restanti o per almeno un elemento o per l'ordine con cui gli oggetti sono disposti; è sufficiente calcolare `prod(n:(n-k+1))`. Ad esempio $D_{6,3}$ è

```
> prod(6:(6-3+1))

[1] 120
```

In R esistono due funzioni per calcolare il coefficiente binomiale $\binom{n}{k}$ e il suo logaritmo che si chiamano rispettivamente `choose()` e `lchoose()`. Il coefficiente binomiale $\binom{4}{2}$ è quindi

```
> choose(4,2)

[1] 6
```

e il suo logaritmo

```
> lchoose(4,2)

[1] 1.791759
```

2.2 Campionamento

Un aspetto molto interessante dell'uso del computer consiste nella possibilità di ricreare artificialmente alcune situazioni che nella realtà risultano difficili da replicare, costose o dispendiose in termini di tempo. Ecco qui alcuni primi semplici esempi di simulazione.
Il comando `sample()` permette di estrarre (con o senza reinserimento) un certo numero di valori da un insieme prefissato.

```
> ?sample
```

Una permutazione dei primi 12 numeri naturali si ottiene da

```
> x <- 1:12
> sample(x)

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

E con ripetizioni:

```
> sample(x, replace=TRUE)

 [1]  4  7  6  4  4 12  1 12  5  9  6  6
```

Per generare i possibili risultati di 10 lanci di un dado equilibrato, possiamo fare così:

```
> sample(1:6, 10, replace=T)

 [1] 3 6 6 6 6 5 6 6 1 5
```

ancora

```
> sample(1:6, 10, replace=T)
```

```
 [1] 3 4 6 6 6 3 3 5 4 2
```

e ancora

```
> sample(1:6, 10, replace=T)
```

```
 [1] 1 4 5 2 4 4 6 2 4 6
```

I risultati sono diversi ogni volta: un generatore casuale sta lavorando dentro R !
Lanciamo ora il dado tantissime volte e registriamo i risultati in un vettore:

```
> y <- sample(1:6, 10000, replace=T)
> table(y)/10000
```

```
y
     1      2      3      4      5      6
0.1650 0.1697 0.1697 0.1615 0.1659 0.1682
```

Le frequenze relative dei diversi risultati sono molto vicine alle probabilità teoriche dei risultati possibili del lancio di un dado. In effetti, se il numero di simulazioni è abbastanza elevato, le frequenze relative si possono usare come approssimazioni delle vere probabilità. Si tratta di una legge importante del calcolo delle probabilità, su cui torneremo più avanti.
Possiamo anche lanciare un dado truccato, con probabilità 0.5 che esca il numero 1:

```
> z <- sample(1:6, 10000, replace=T, prob=c(0.5,rep(0.1,5)))
> table(z)/10000
```

```
z
     1      2      3      4      5      6
0.5058 0.1035 0.0994 0.0944 0.0949 0.1020
```

Ancora le frequenze relative approssimano le rispettive probabilità.
Adesso giochiamo a carte! Immaginiamo di avere un classico mazzo da 52 carte, con 4 semi (C, Q, F, P) e 13 carte per ogni seme (A, 2, ...,10, J, Q, K). Il nostro gioco consiste nell'estrazione di 10 carte dal mazzo con reinserimento e nel conteggio del numero di carte rosse. Dato che ci interessa solo il colore, possiamo facilmente costruire il nostro mazzo,

```
> mazzo <- c(rep("R", times=26), rep("N", times=26))
```

estrarre dal mazzo le 10 carte e contare il numero di carte rosse estratte:

```
> k <- sample(mazzo, 10, replace=T)
> sum(k=="R")
```

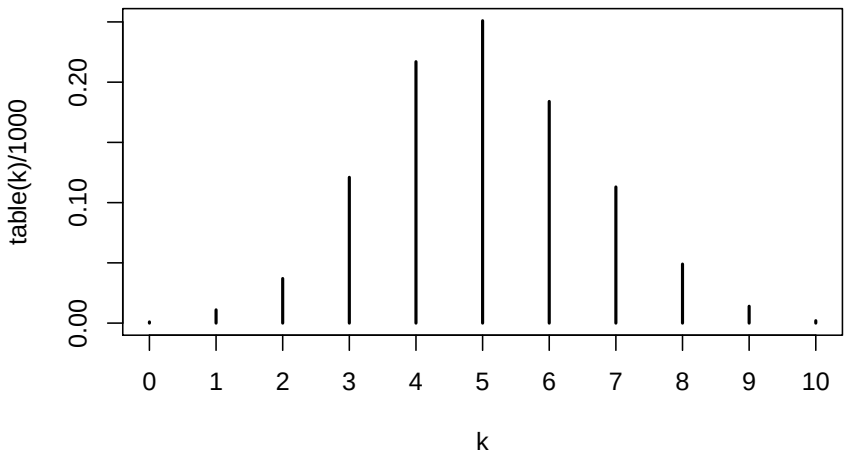
```
[1] 6
```

Quante carte rosse abbiamo estratto? Quante ci aspettavamo di estrarne? Se ripetiamo la procedura ci accorgiamo che il risultato ha una certa variabilità. Possiamo usare la simulazione per approssimare la probabilità di ottenere i diversi valori possibili.
La funzione `do()` del pacchetto `mosaic` ci aiuta con le replicazioni:

```
> library(mosaic)
> k <- do(1000)*sum(sample(mazzo, 10, replace=T)=="R")
> table(k)/1000

k
    0     1     2     3     4     5     6     7     8     9
0.001 0.011 0.037 0.121 0.217 0.251 0.184 0.113 0.049 0.014
   10
0.002
```

Sapete calcolare usando le regole della probabilità elementare le vere probabilità? Confrontatele con le frequenze trovate.
E adesso una rappresentazione grafica:

```
> plot( table(k)/1000 )
```

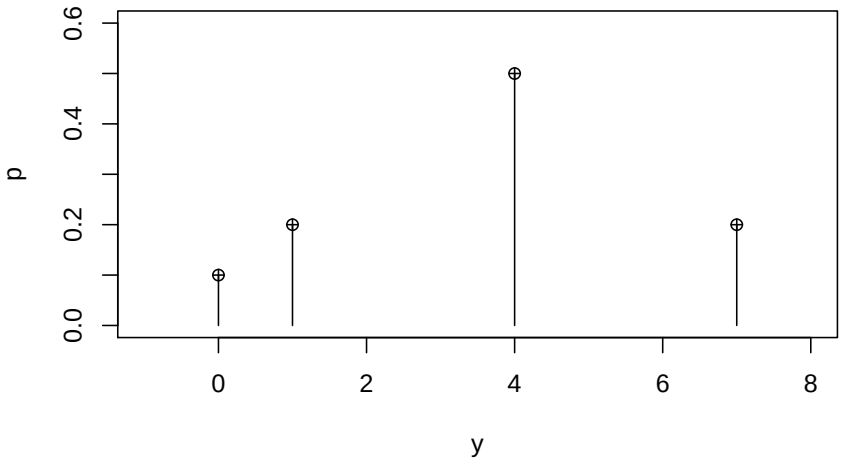


Esercizio: Create un'urna contenente quattro palline bianche numerate da 1 a 4 e tre nere numerate da 1 a 3, ed estraete due palline senza reinserimento e poi dieci con reinserimento. Approssimate, usando una frequenza relativa, la probabilità di ottenere due palline nere estraendole senza reinserimento e confrontate il valore che avete ottenuto con quello calcolato usando le regole della probabilità elementare.

2.3 Distribuzioni di probabilità

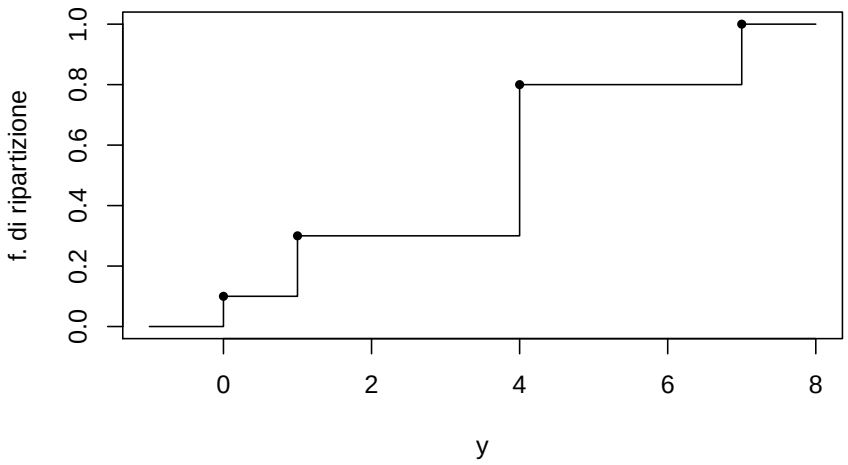
Definiamo ora una variabile aleatoria discreta Y a valori nell'insieme $\{0, 1, 4, 7\}$ con funzione di probabilità $p(0) = 0.1$, $p(1) = 0.2$, $p(4) = 0.5$ e $p(7) = 0.2$, e rappresentiamola graficamente tramite diagramma a bastoncini:

```
> y<-c(0,1,4,7)
> p<-c(0.1,0.2,0.5,0.2)
> plot(y, p, type="h", ylim=c(0,0.6), xlim=c(-1,8))
> points(y, p, pch=10)
```



Possiamo anche disegnare il grafico della funzione di ripartizione

```
> Fr<-cumsum(p)
> Fr0<-c(0, Fr, 1)
> y0<-c(-1, y, 8)
> plot(y0, Fr0, type="s", ylab="f. di ripartizione", xlab="y")
> points(y, Fr, pch=20)
```



e calcolare valore atteso e varianza di Y

```
> mu<-sum(y*p)
> mu

[1] 3.6

> s2<-sum(y^2*p)-mu^2
> s2

[1] 5.04
```

Infine, possiamo simulare valori casuali dalla distribuzione di Y, semplicemente utilizzando il comando `sample()`:

```
> sample(y, 10, prob=p, replace=T)

 [1] 4 7 0 0 4 4 7 4 7 0
```

R consente di gestire tutte le distribuzioni di probabilità più comuni di cui fornisce automaticamente densità, funzione di ripartizione, quantili e generazione di numeri casuali. Alcune delle distribuzioni disponibili sono:

R	Distribuzione	Parametri	Defaults
unif	Uniforme	min, max	0, 1
binom	Binomiale	n, p	-, -
pois	Poisson	mean	-
exp	Esponenziale	rate	1
gamma	Gamma	shape, rate	-, 1
norm	normale	mean, sd	0, 1
t	t di Student	df	-
chisq	chi quadrato	df	-
f	F di Fisher	df1, df2	-, -
...			

2.3.1 La distribuzione binomiale

Consideriamo ad esempio la distribuzione Binomiale: se $X \sim \text{Bin}(n = 10, p = 0.2)$ la probabilità che X assuma il valore $x = 2$ è data dalla funzione

```
> dbinom(2,10,0.2)

[1] 0.3019899
```

Come si vede per ottenere la densità della variabile casuale in $x = 2$ è stato semplicemente aggiunto il prefisso `d` al nome della distribuzione `binom`; la sintassi è quindi `dbinom(x,n,p)`. Per calcolare la funzione di ripartizione ossia $P(X \leq x)$ il prefisso da utilizzare è `p`, con sintassi del tutto simile; per ottenere invece la probabilità dell'evento complementare $P(X > x)$ si deve aggiungere l'opzione `lower.tail=F`.

```
> pbinom(2,10,0.2)

[1] 0.6777995

> pbinom(2,10,0.2,lower.tail=F)

[1] 0.3222005
```

Allo stesso modo, per ottenere i quantili delle diverse distribuzioni il prefisso da utilizzare è `q`. Sempre per fare un esempio nell'ambito della Binomiale la sintassi da usare è `qbinom`$(\alpha, \texttt{n}, \texttt{p})$, dove con α si è indicato il livello del quantile.

```
> qbinom(0.45,3,1/3)

[1] 1
```

che è corretto. Infatti, il quantile in corrispondenza di 0.45 è il valore 1 poiché cumulando fino a 1 la densità di probabilità di una binomiale $n = 3$ e $p = 1/3$ si ha

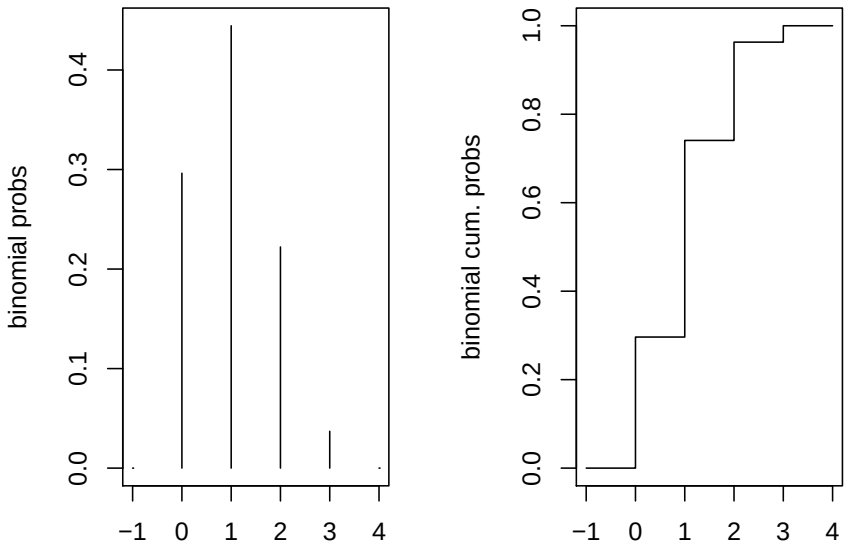
```
> pbinom(1,3,1/3)

[1] 0.7407407
```

mentre

```
> pbinom(0,3,1/3)

[1] 0.2962963
```

E' facile ottenere grafici della funzione di probabilità e della funzione di ripartizione della binomiale considerata:

```
> par(mfrow=c(1,2))
> plot(-1:4,dbinom(-1:4,3,1/3),type="h",xlab="",ylab="binomial probs")
> plot(-1:4,pbinom(-1:4,3,1/3),type="s",xlab="",ylab="binomial cum. probs")
> par(mfrow=c(1,1))
```



2.3.2 La distribuzione normale

Prendiamo ora la distribuzione Normale: $X \sim \mathrm{N}(\mu = 5, \sigma = 0.8)$. La densità di X in $x = 4.5$ è data da

```
> dnorm(4.5, 5, 0.8)

[1] 0.4102012
```

La probabilità $P(4.6 < X < 5.2)$ si calcola utilizzando la funzione di ripartizione,

```
> pnorm(5.2, 5, 0.8)-pnorm(4.6, 5, 0.8)

[1] 0.2901688
```

Si osservi che, utilizzando la normale standard (default), potremmo anche fare

```
> pnorm( (5.2-5)/0.8 )-pnorm( (4.6-5)/0.8 )

[1] 0.2901688
```

Per ottenere i quantili della distribuzione normale la sintassi da usare è `qnorm`(α, μ, σ), dove con α si è indicato il livello del quantile, ad esempio

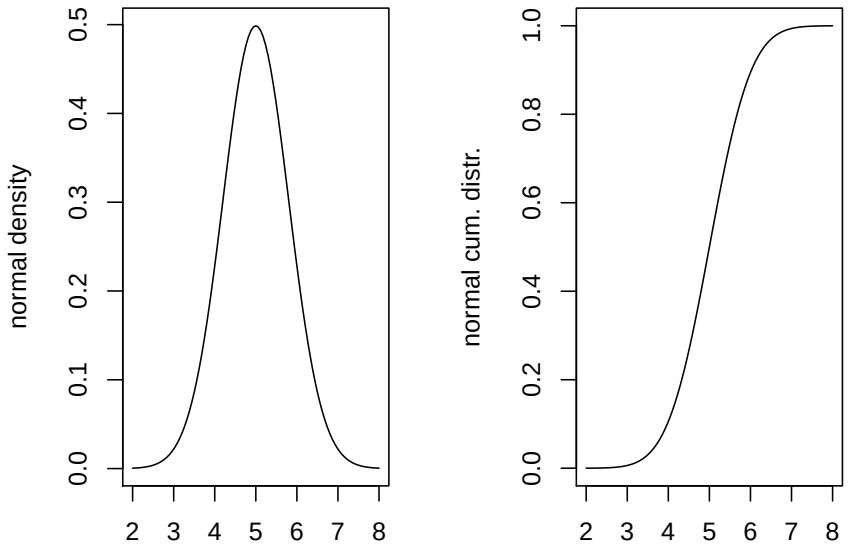
```
> qnorm(0.45, 5, 0.8)

[1] 4.899471
```

che è lo stesso di

```
> qnorm(0.45)*0.8+5

[1] 4.899471
```

E' facile ottenere grafici della funzione di densità e della funzione di ripartizione della normale considerata:

```
> par(mfrow=c(1,2))
> curve(dnorm(x, 5, 0.8),xlim=c(2,8),xlab="",ylab="normal density")
> curve(pnorm(x, 5, 0.8),xlim=c(2,8),xlab="",ylab="normal cum. distr.")
> par(mfrow=c(1,1))
```



Esercizio 1: Quale comando si utilizza in R per calcolare $P(X > 3.3)$ nei seguenti casi:

1. $X \sim N(\mu = 2, \sigma^2 = 2)$
2. $X \sim Exp(\lambda = 3)$
3. $X \sim U(2, 4)$
4. $X \sim Bi(n = 10, p = 0.3)$
5. $X \sim P(\lambda = 7)$

Esercizio 2: Quale comando si utilizza in R per calcolare il valore x tale che $P(X > x) = 0.65$, quando:

1. $X \sim N(\mu = 2, \sigma^2 = 2)$

2. $X \sim Exp(\lambda = 3)$
3. $X \sim U(2, 4)$
4. $X \sim Bi(n = 10, p = 0.3)$
5. $X \sim P(\lambda = 7)$

2.4 Simulazione di variabili casuali

2.4.1 Generazione di numeri pseudo-casuali

Il primo passo per generare una successione di numeri casuali da una certa distribuzione, consiste nell'ottenere una successione di valori da una v.c. uniforme in (0,1). Per fare questo molti software tra cui R impiegano un cosidetto generatore di numeri *pseudo*-casuali. Si tratta di una sequenza di n numeri ottenuta deterministicamente ma virtualmente indistinguibile da un campione casuale semplice di dimensione n estratto da una v.c. $U(0, 1)$.
I generatori di tipo congruenziale hanno la forma

$$y_{n+1} = (ay_n + b) \bmod m \longrightarrow \begin{cases} \text{prendo il resto della divisione per} \\ m \end{cases}$$

a, b e m sono valori positivi interi e y_0, valore iniziale, è detto seme. Se $b = 0$ si ottiene un generatore di tipo moltiplicativo.
Un tale algoritmo dà luogo a una sequenza di interi compresi nell'intervallo $[0, m-1]$. I valori $u_i = \frac{y_i}{m}$ vengono assimilati alle determinazioni di una $U(0, 1)$ (cioè a numeri frazionari compresi fra 0 e 1 e tali che non vi sia preferenza per alcun valore).
Ecco un esempio:

$$\begin{array}{llll} X_0 = 89 & a = 1573 & b = 19 & m = 10^3 \\ X_1 = 140016 & (\text{mod } 1000) & = 16 & \\ X_2 = 25187 & (\text{mod } 1000) & = 187 & \\ \vdots & & & \end{array}$$

e il corrispondente codice in R

```
> n<-5
> y<-numeric(n+1)# il seme
> y[1]<-89
> a<-1573
> m<-10^3
> b<-19
> for (i in 2:(n+1))
+   y[i]<- (a*y[i-1]+b)%%m
> y<-y/m
> y

[1] 0.089 0.016 0.187 0.170 0.429 0.836
```

a, b ed m vanno scelti opportunamente al fine di rendere facili i calcoli, ottenere cicli lunghi (dell'ordine di 10^8) ed evitare che vi sia correlazione fra numeri successivi della serie. Una scelta comune per m è $2^{31} - 1$, che è il più grande intero che può essere tenuto in memoria da molti computer. Associato a questo, in letteratura per a è stato proposto il valore 7^5.
La funzione di default in R per generare valori da una v.c. $U(0, 1)$ è

```
> set.seed(203)
> runif(5)
```

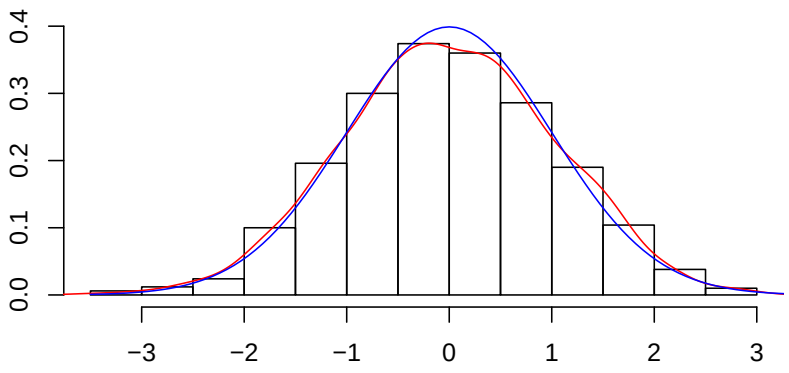
```
[1] 0.1705013 0.4675975 0.4635794 0.1868930 0.4206904
```

A partire da questi valori si possono poi ottenere dati casuali estratti da qualsiasi modello probabilistico discreto e continuo (dalla normale, dalla binomiale, etc.). Le tecniche per fare questo sono molte (per inversione, per accettazione/rifiuto, per composizione, etc). Troviamo comunque già pronti all'uso i generatori di numeri casuali per i principali tipi di modelli probabilistici in R o in altri pacchetti statistici. Per ottenere la generazione di una serie di numeri casuali provenienti dalle variabili casuali disponibili in R si deve anteporre il prefisso r al nome che identifica la distribuzione. Se vogliamo dunque simulare 1000 valori da una distribuzione normale standard, la sintassi è

```
> x <- rnorm(1000)
```

A questo punto possiamo pensare di produrre una stima della densità del vettore x, attraverso la funzione `density()` (sappiamo già come rappresentare graficamente la distribuzione di un insieme di dati attraverso i comandi `boxplot()` e `hist()`) e di metterla a confronto con la densità vera di una variabile casuale normale standard, utilizzando il comando `curve()` che serve a tracciare il grafico di una qualunque funzione:

```
> hist(x,prob=T, main="1000 valori da N(0,1)",xlab="",ylab="",ylim=c(0,0.45))
> lines(density(x), col="red")
> curve(dnorm(x), xlim=c(-3.5,3.5), add=T, col="blue")
```

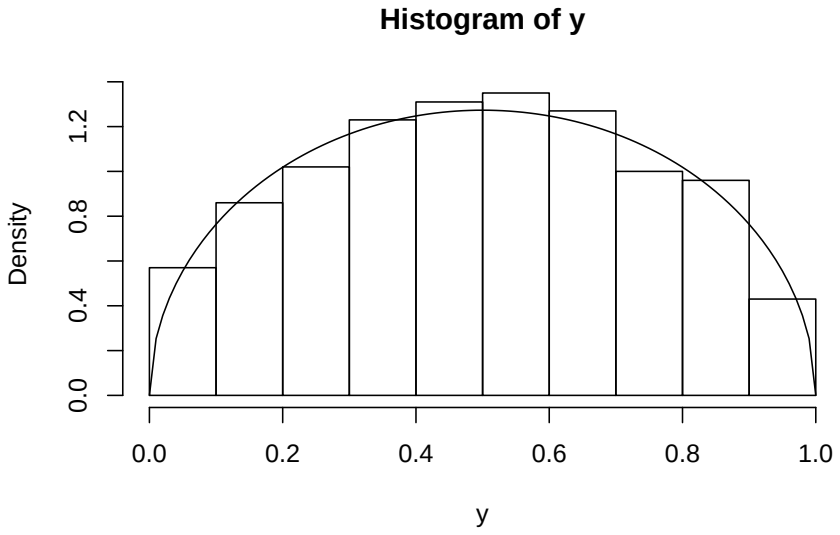


Se vogliamo simulare 1000 valori indipendenti da una v.c. Beta con densità

$$f(y) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} y^{a-1}(1-y)^{b-1}, \qquad 0 < y < 1$$

con $a = b = 1.5$, avremo

```
> a<-1.5
> b<-1.5
> n<-1000
> set.seed(567)
> y<-rbeta(n,a,b)
> hist(y,prob=TRUE)
> curve(dbeta(x, a,b),from=0, to=1, ylab='f(y)', xlab='y', add=TRUE)
```



dove abbiamo confrontato l'istogramma dei valori simulati con la funzione di densità corrispondente.

2.4.2 Metodo di inversione

Il più semplice metodo di simulazione è quello detto di inversione e si basa sul seguente risultato:
Se Y è un v.c. con f.r. F_Y, allora $U = F_Y(Y) \sim U(0,1)$.
Cosicché se $U \sim U(0,1)$ allora

$$Y = F^{-1}(U)$$

ha funzione di ripartizione F.
Ad esempio se volessimo produrre un campione dalla distribuzione $Exp(\lambda)$, la funzione di ripartizione è

$$F(y) = 1 - e^{-\lambda y}, \quad y > 0,$$

dalla quale di ricava

$$F^{-1}(u) = -\log(1-u)/\lambda, \qquad 0 < u < 1.$$

```
> expsim <- function(n,lambda)
+ {
+  y <- -log(1-runif(n))/lambda
+  return(y)
+ }
>  y<-expsim(5,2)
> y

[1] 0.2617059 0.3984065 0.1331501 0.3785203 0.3963135
```

2.4.3 Metodo dell'accettazione/rifiuto

Viene generalmente utilizzato per ottenere valori da distribuzioni continue. L'idea di base consiste nel generare un valore da una distribuzione con certe proprietà e poi accettare il valore simulato, con una prefissata probabilità p. Se p è scelta correttamente il valore generato risulta essere una realizzazione della distribuzione di interesse.
Sia X una v.a. continua con densità $f(x)$ e sia $g(x)$ un'altra funzione di densità tale che $f(x) \leq Kg(x)\ \forall\ x$. Inoltre si supponga di poter facilmente generare valori casuali dalla distribuzione con densità $g(x)$.

Algoritmo:
1. si genera $\hat{x}$ da $g(\cdot)$
2. si genera u da $U(0, Kg(\hat{x}))$
3. se $u \leq f(\hat{x})$ allora si accetta $\hat{x}$.

In alternativa i punti 2.-3. possono essere sostituiti dai seguenti
2b. si genera u da $U(0, 1)$
3b. se $u \leq \frac{f(\hat{x})}{Kg(\hat{x})}$ allora si accetta $\hat{x}$.

L'efficienza dell'algoritmo dipende dal valore di K. Se è necessario un valore di K molto elevato per poter soddisfare la disuguaglianza $f(x) \leq Kg(x)\ \forall\ x$ allora la probabilità di accettare i valori simulati da $g(\cdot)$ è bassa. Quindi sarà necessario un numero elevato di generazioni da $g(\cdot)$ per ottenere pochi valori accettabili.
Proviamo ora a simulare dei valori da una distribuzione $Beta(2, 2)$, senza usare le funzioni predefinite in R . Utilizzando il metodo dell'accettazione/rifiuto, possiamo selezionare dei valori simulati da un'uniforme in $(0, 1)$. Si vede facilmente che la densità Beta considerata assume un valore massimo pari a $k = 3/2$.

```
> sim.f<-function(n=1,k=3/2){
+ x<-NULL
+ j<-0
+ for ( i in 1:n )
+ {
+ fx<-0
+ gx<-1
+ u<-1
```

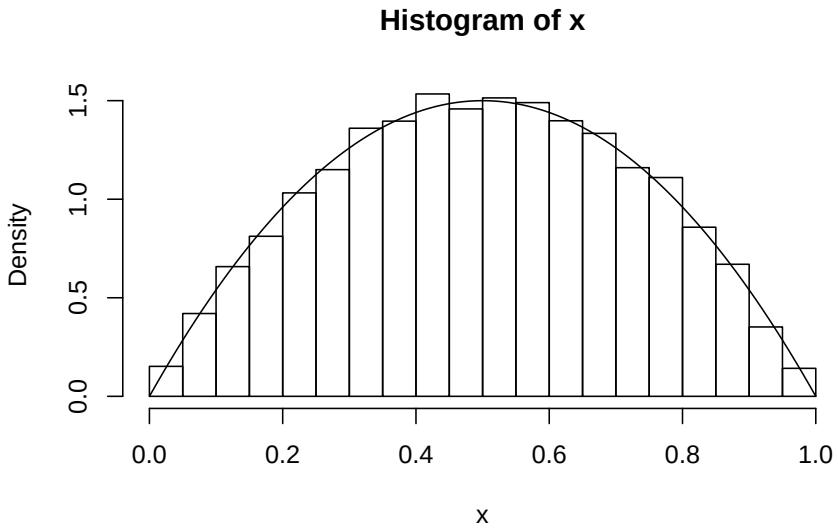
```
+ while(fx/(k*gx)<u)
+ {
+ u<-runif(1)
+ xsim<-runif(1)
+ fx<-6*xsim*(1-xsim)
+ gx<-1
+ j<-j+1
+ }
+ x<-c(x,xsim)
+ }
+ return(list(dati=x,simulazioni=j))
+ }
```

Proviamo la nostra funzione variando il valore di k:

```
> sim.f(100)
> sim.f(100,2)
> sim.f(100,5)
```

Cosa possiamo dire?
Adesso verifichiamo che il nostro algoritmo funzioni bene:

```
> x<-sim.f(10000)
> x$sim

[1] 15028

> x<-x$dati
> hist(x,freq=F)
> curve(dbeta(x,2,2),add=T)
```



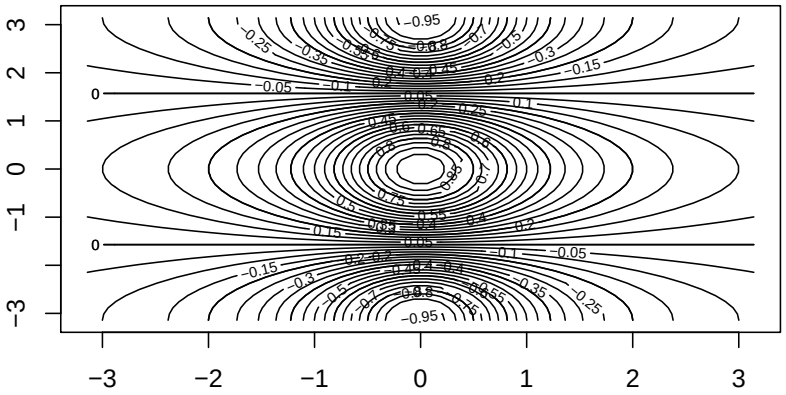
2.5 Grafici tridimensionali

In questo capitolo impariamo a costruire grafici tridimensionali.
La funzione `contour()` produce grafici simili alle mappe topografiche. Ha bisogno di 3 argomenti: un vettore di valori per l'asse x, un vettore per l'asse y e una matrice contenente i valori z corrispondenti ad ogni coppia di coordinate (x, y). Per saperne di più,

```
> ?contour
```

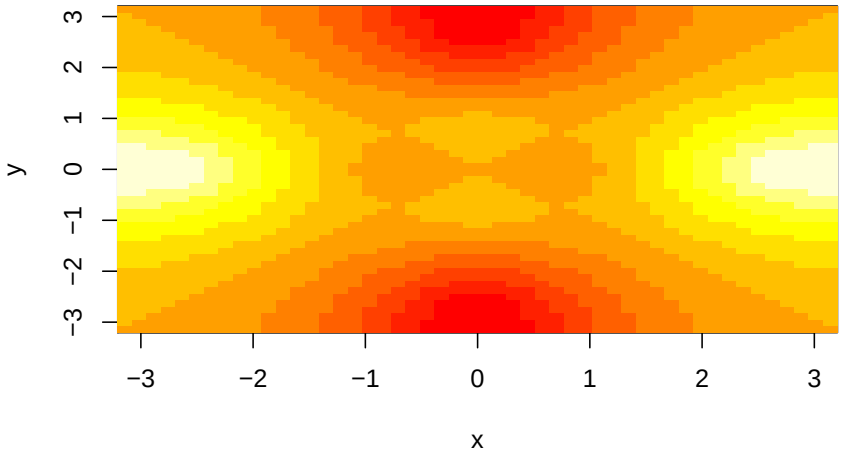
Proviamo a rappresentare la funzione $f(x, y) = \cos(y)/(1 + x^2)$.

```
> x=seq(-pi,pi,length=50)
> y=x
> f=outer(x,y,function(x,y)cos(y)/(1+x^2))
> contour(x,y,f)
> contour(x,y,f,nlevels=45,add=T)
```



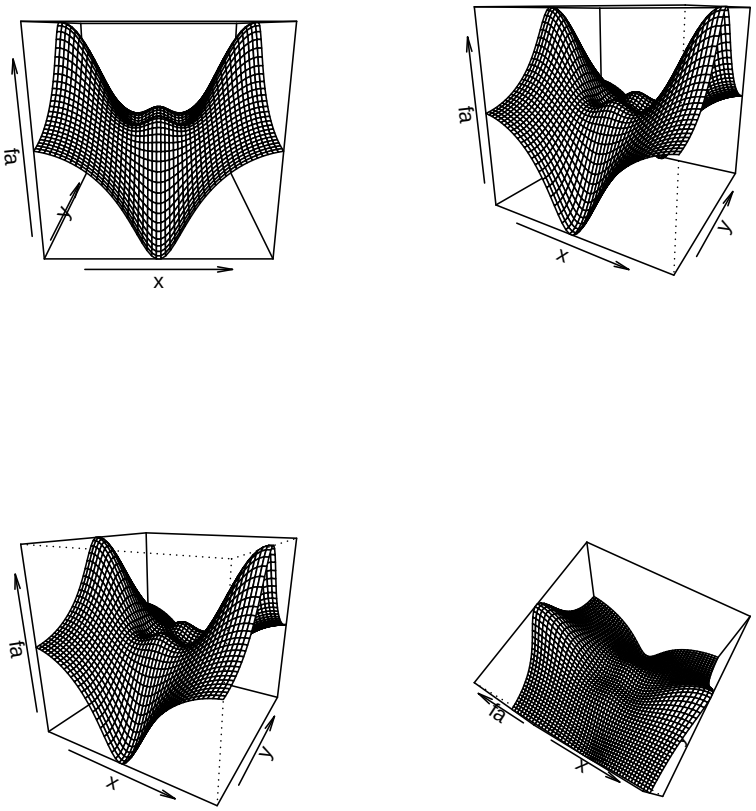
La funzione `image()` lavora in modo simile a `contour()`, ma usa colori diversi a seconda dei valori di z. Produce mappe come quelle usate per rappresentare temperature diverse nelle previsioni del tempo.

```
> fa=(f-t(f))/2
> image(x,y,fa)
```



Infine possiamo produrre grafici tridimensionali usando il comando `persp()`. Le opzioni `theta` e `phi` controllano gli angoli da cui si vede il grafico.

```
> par(mfrow=c(2,2))
> persp(x,y,fa)
> persp(x,y,fa,theta=30)
> persp(x,y,fa,theta=30,phi=20)
> persp(x,y,fa,theta=30,phi=70)
> par(mfrow=c(1,1))
```

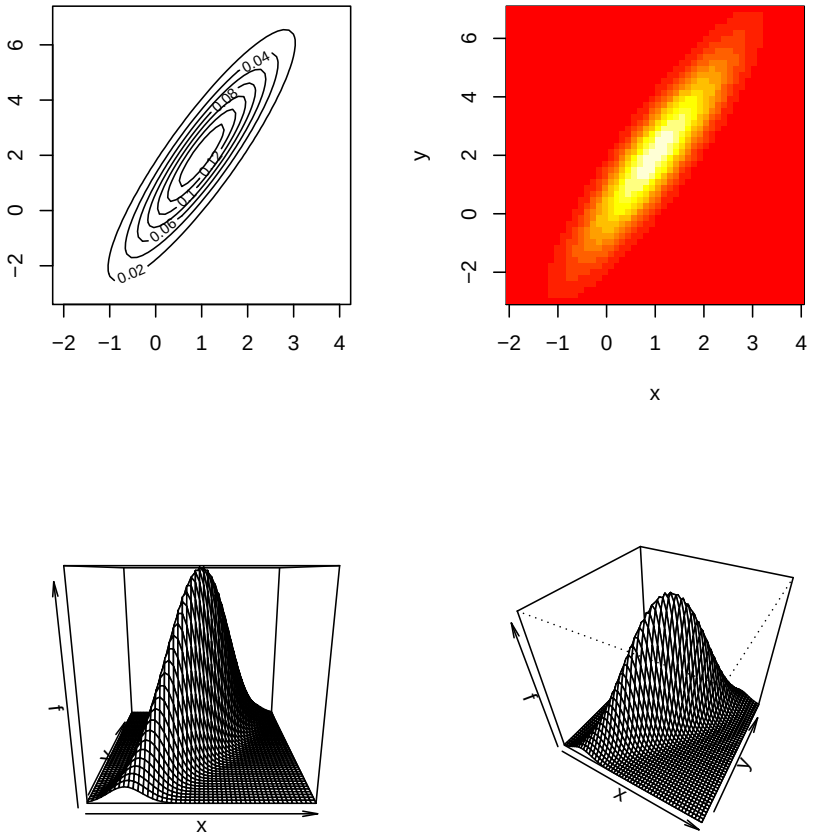


2.5.1 La normale bivariata

Ora, vogliamo usare i comandi appena introdotti per rappresentare la distribuzione normale bivariata.
La libreria `mnormt` fornisce già le funzioni necessarie per calcolare la densità e la funzione di ripartizione della normale bivariata. Il loro uso è simile a quello delle funzioni `dnorm()` e `pnorm()` che conosciamo. Bisogna solo specificare in modo opportuno un vettore di dimensione 2 per la media e una matrice 2×2 di varianza-covarianza.

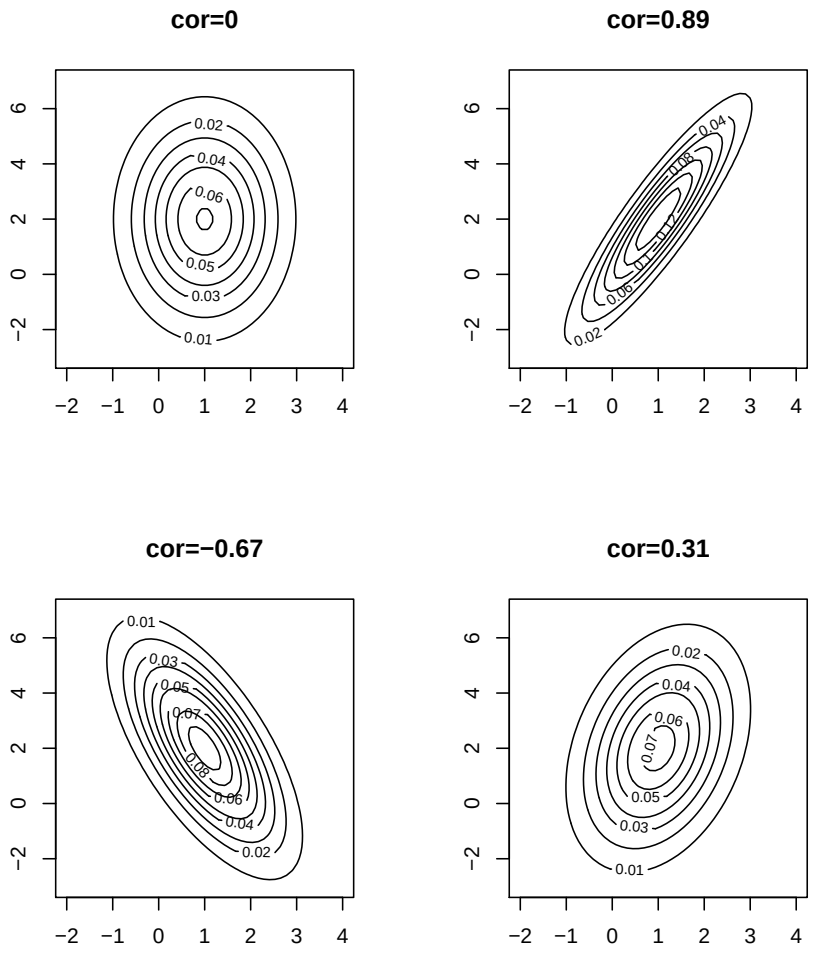
```
> ?dmnorm

> par(mfrow=c(2,2))
> x=seq(-2,4,length=50)
> y=seq(-3,7,length=50)
> mu <- c(1,2)
> Sigma <- matrix(c(1,2,2,5), 2, 2)
> f <- outer(x,y, function(x,y)dmnorm(cbind(x,y), mu, Sigma))
> contour(x,y,f,nlevels=10)
> image(x,y,f)
```

```
> persp(x,y,f)
> persp(x,y,f,theta=30,phi=40)
> par(mfrow=c(1,1))
```



Se facciamo variare la matrice di varianza-covarianza, cosa cambia?

```
> par(mfrow=c(2,2))
> Sigma1 <- matrix(c(1,0,0,5), 2, 2)
> Sigma2 <- matrix(c(1,2,2,5), 2, 2)
> Sigma3 <- matrix(c(1,-1.5,-1.5,5), 2, 2)
> Sigma4 <- matrix(c(1,0.7,0.7,5), 2, 2)
> f1 <- outer(x,y, function(x,y)dmnorm(cbind(x,y), mu, Sigma1))
> f2 <- outer(x,y, function(x,y)dmnorm(cbind(x,y), mu, Sigma2))
> f3 <- outer(x,y, function(x,y)dmnorm(cbind(x,y), mu, Sigma3))
> f4 <- outer(x,y, function(x,y)dmnorm(cbind(x,y), mu, Sigma4))
> contour(x,y,f1, nlevels=10, main="cor=0")
> contour(x,y,f2, nlevels=10, main="cor=0.89")
> contour(x,y,f3, nlevels=10, main="cor=-0.67")
> contour(x,y,f4, nlevels=10, main="cor=0.31")
> par(mfrow=c(1,1))
```



Nel *package* `MASS` la funzione `mvrnorm()` permette di generare dei valori casuali da una distribuzione normale multivariata. Proviamola per il caso bivariato:

```
> ?mvrnorm
```

```
> Sigma <- matrix(c(10,-3,-3,2),2,2)
> Sigma
```

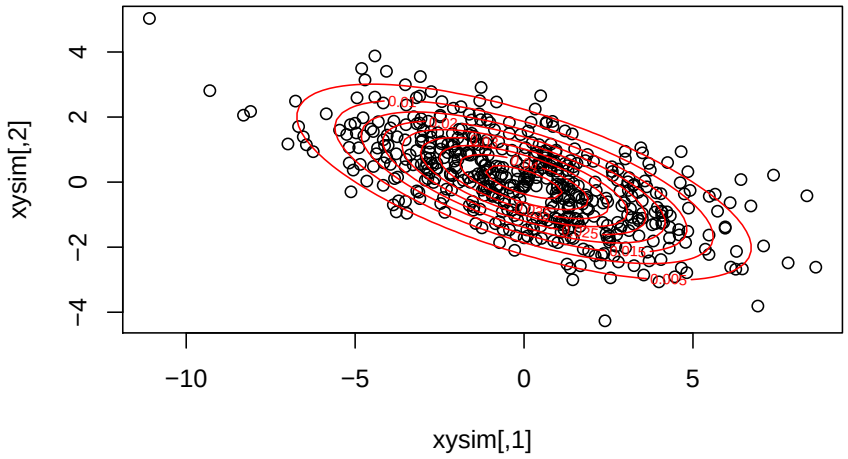
```
     [,1] [,2]
[1,]   10   -3
[2,]   -3    2
```

```
> xysim <- mvrnorm(n=500, rep(0, 2), Sigma)
> var(xysim)
```

```
          [,1]      [,2]
[1,]  9.000793 -2.610282
[2,] -2.610282  1.809579
```

Adesso sovrapponiamo al diagramma di dispersione dei dati le linee di livello della normale bivariata da cui abbiamo generato i dati stessi:

```
> plot(xysim)
> x=seq(-10,10,length=100)
> y=x
> f <- outer(x,y, function(x,y)dmnorm(cbind(x,y), rep(0, 2), Sigma))
> contour(x,y,f, nlevels=10, col="red", add=T)
```



2.6 Esercizi

1. Controllare graficamente la distribuzione di 1000 valori generati per inversione da una distribuzione esponenziale, sovrapponendo all'istogramma la densità corrispondente. Come si possono simulare per inversione 1000 valori da una v.c. Gamma(10,2)? E da una binomiale $Bin(10, 0.3)$?

2. Generare tramite il metodo di accettazione/rifiuto 10000 valori da una distribuzione normale standard. Confrontare graficamente l'istogramma dei valori simulati con la densità della normale standard.

3. Costruire l'istogramma relativo alla variabile `eruptions` nell'insieme di dati `faithful` di R e sovrapporne una stima della densità ottenuta con il comando `density()`.

4. Disegnare per i dati `rivers` di R un istogramma e sovrapporre la densità di una variabile aleatoria esponenziale di parametro `1/mean(rivers)`. Confrontare quindi frequenze e quartili dei dati con probabilità e quartili della variabile esponenziale considerata.

5. Simulare 1000 valori da una distribuzione normale bivariata (X, Y) tale che $X \sim \mathrm{N}(\mu = 5, \sigma^2 = 3)$ e $Y \sim \mathrm{N}(\mu = -2, \sigma^2 = 6)$ con $cor(X, Y) = 0.77$. Rappresentare i valori simulati in un diagramma di dispersione e sovrapporre le linee di livello della normale bivariata generatrice dei dati stessi.

6. Simulare 1000 valori da una distribuzione normale bivariata di media $\mu = (5, -2)$ e matrice di varianza-covarianza $\Sigma = (\sigma_{ij})$ tale che $\sigma_{11} = 2$, $\sigma_{22} = 3$ e $\sigma_{12} = -1$. Usando un'opportuna rappresentazione grafica, mostrare che i vettori simulati marginali hanno ciascuno distribuzione normale.

Capitolo 3

Teoremi limite e applicazioni

3.1 Il teorema del limite centrale

Se X_i $i = 1, 2, \ldots$ è una successione di variabili casuali i.i.d. di media μ e varianza σ^2 finita, allora

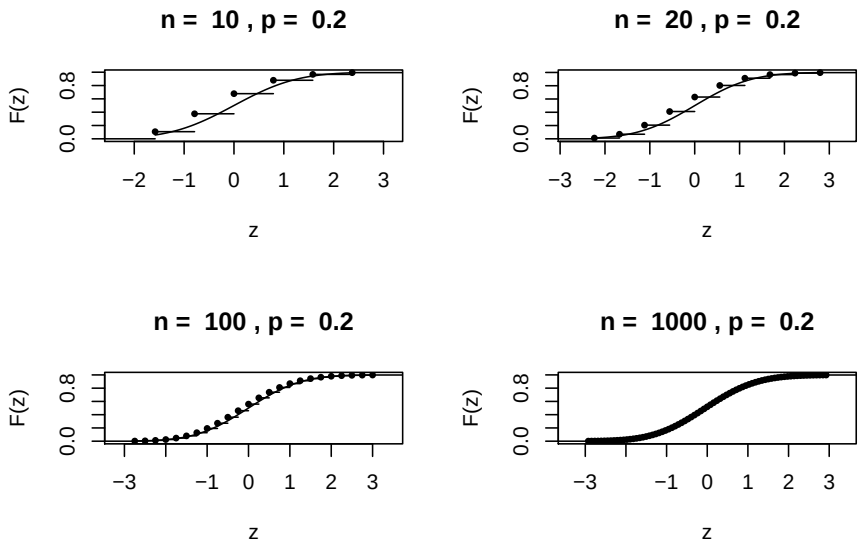
$$\bar{Z}_n = \frac{\bar{X}_n - \mu}{\sigma/\sqrt{n}}$$

converge in distribuzione ad una v.c. $\mathcal{N}(0, 1)$. La convergenza in distribuzione è la convergenza puntuale delle rispettive funzioni di ripartizione.
Consideriamo il caso in cui $X_i \sim \mathcal{B}\backslash(1, 0.2)$ e quindi $\sigma^2 = var(X_i) = p(1-p) = 0.16$. Facciamo vedere come all'aumentare di n la funzione di ripartizione di

$$\bar{Z}_n = \frac{\bar{X}_n - \mu}{\sigma/\sqrt{n}}$$

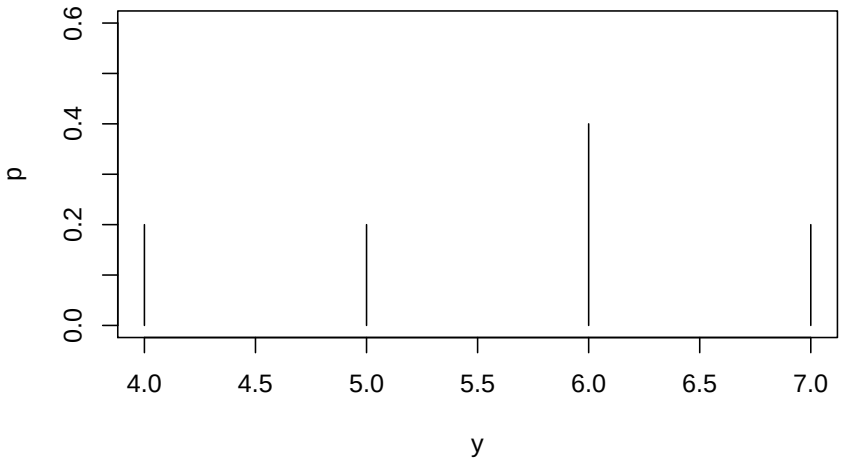
si avvicini sempre più a quella di una v.c. $\mathcal{N}(0, 1)$.

```
> par(mfrow=c(2,2))
> p<-0.2
> nobs<-c(10,20,100,1000)
> for (n in nobs)
+ {
+ y<-0:n
+ prob<-pbinom(y,n,p)
+ z<-(y/n-p)*sqrt(n)/sqrt(p*(1-p))
+ ind<-(z>-3)&( z<3)
+ z<-z[ind]
+ prob<-c(0,prob[ind])
+ plot(stepfun(z, prob, f = 0),verticals=FALSE,pch=20,
+   main=paste("n = ",n ,", p = ", p),ylab="F(z)",xlab="z")
+ curve(pnorm(x),from=min(z),to=max(z),add=TRUE)
+ }
> par(mfrow=c(1,1))
```



Consideriamo ora un altro esempio. Sia Y una v.c. che assume come possibili valori gli interi da 4 a 7 con probabilità rispettivamente 1/5, 1/5, 2/5, 1/5. Definiamola in R , rappresentiamola mediante diagramma a bastoncini e calcoliamone valore atteso e varianza:

```
> y<-4:7
> p<-c(1,1,2,1)/5
> plot(y,p,type='h',ylim=c(0,0.6))
> sum(p*y)

[1] 5.6

> sum(p*y^2)-sum(p*y)^2

[1] 1.04
```

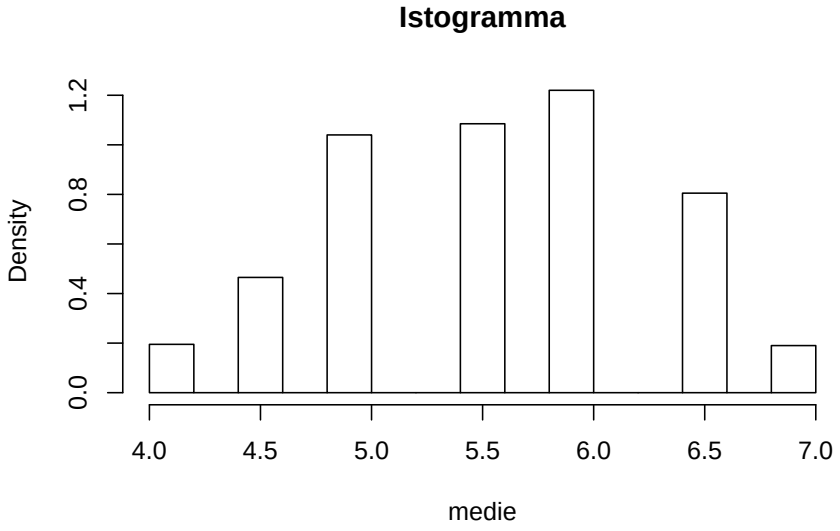


Consideriamo ora un campione di dimensione 2 da Y e la sua media:

```
> set.seed(12)
> ys<-sample(y,size=2,prob=p,replace=TRUE)
> ys

[1] 6 4

> mean(ys)

[1] 5
```

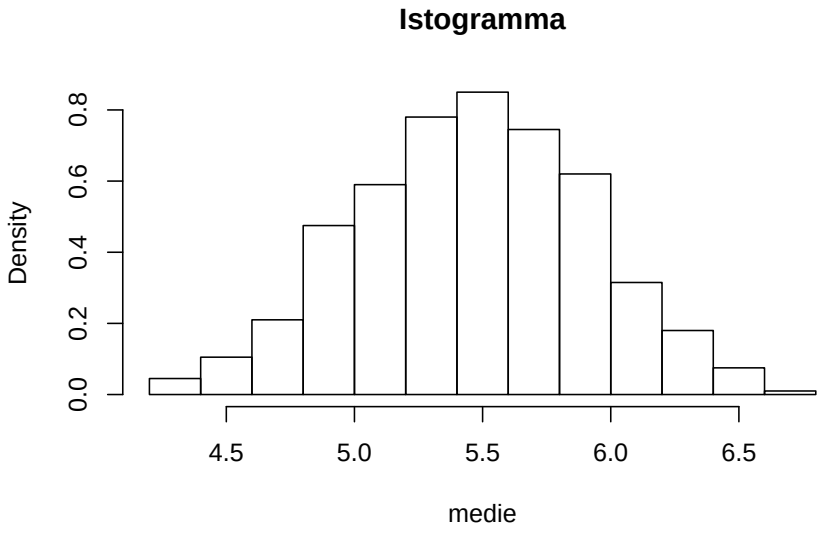
Ora, per immaginarsi come è fatta la distribuzione della media campionaria basata su un campione di dimensione 2 da Y, simuliamo 1000 campioni di dimensione 2 da Y, calcoliamo le loro medie e rappresentiamole graficamente tramite un istogramma:

```
> set.seed(1)
> nsim<-1000
> n<-2
> ys<-matrix(sample(y,n*nsim, replace=TRUE, prob=p), nsim, n)
> ym<-apply(ys,1,mean)
> hist(ym,main='Istogramma', xlab='medie', prob=T)
```



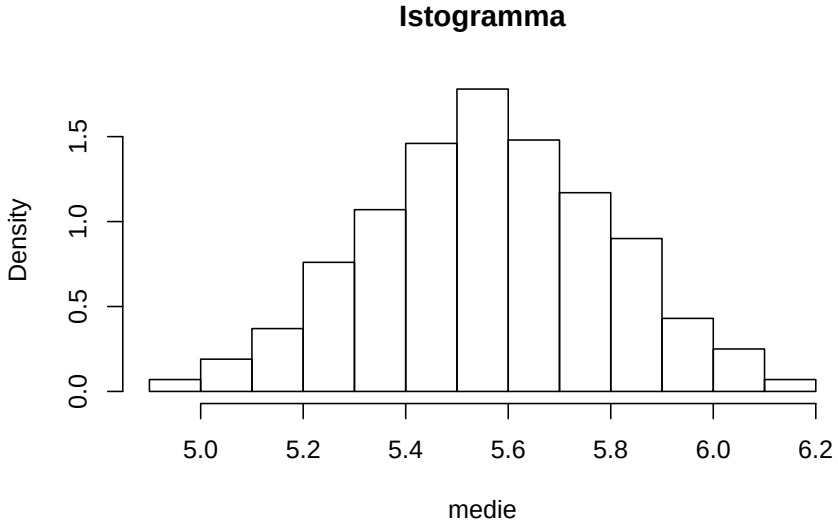
Ripetiamo l'operazione con 1000 campioni di dimensione 5 da Y:

```
> set.seed(1)
> nsim<-1000
> n<-5
> ys<-matrix(sample(y,n*nsim, replace=TRUE, prob=p), nsim, n)
> ym<-apply(ys,1,mean)
> hist(ym,main='Istogramma', xlab='medie', prob=T)
```



e poi di dimensione 20

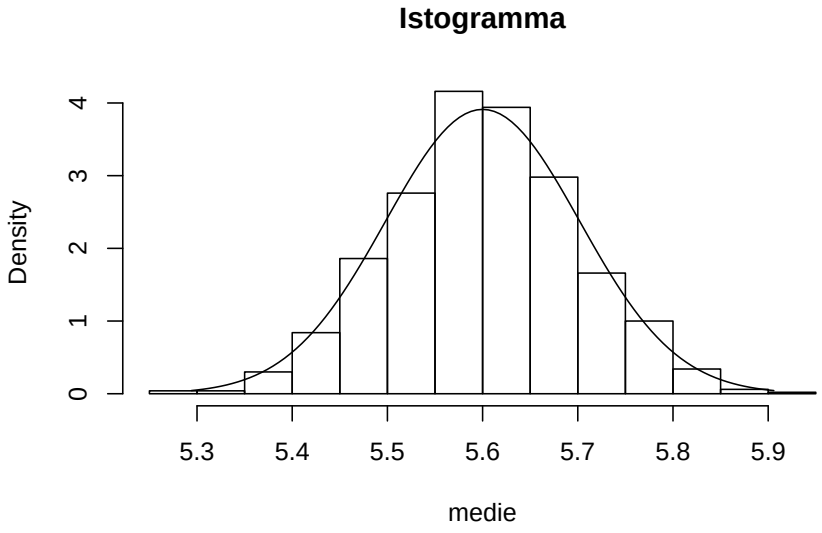
```
> set.seed(1)
> nsim<-1000
> n<-20
> ys<-matrix(sample(y,n*nsim, replace=TRUE, prob=p), nsim, n)
> ym<-apply(ys,1,mean)
> hist(ym,main='Istogramma', xlab='medie', prob=T)
```



e infine 100

```
> nsim<-1000
> n<-100
> ys<-matrix(sample(y, n*nsim, replace=TRUE, prob=p), nsim, n)
> ym<-apply(ys,1,mean)
> hist(ym,main='Istogramma', xlab='medie', prob=T)
```

```
> curve(dnorm(x,5.6,sqrt(1.04/n)),
+ from=5.6-3*sqrt(1.04/n),
+ to=5.6+3*sqrt(1.04/n), add=TRUE)
```



In quest'ultimo grafico abbiamo sovrapposto all'istogramma dei valori simulati della media campionaria la densità della distribuzione approssimante, $\mathcal{N}(5.6, 1.04/n)$.

Esercizi:

1. Simulate 10000 campioni da una v.c. $Y = X_1 + \ldots + X_{12}$ dove le X_i sono i.i.d. $U(-1/2, 1/2)$; disegnate l'istogramma dei 10000 campioni e confrontatelo con quello di una v.c. $\mathcal{N}(0, 1)$ (perché?).

2. Considerate la variabile Y =media dei risultati di 10 lanci di un dado equilibrato. Simulate 1000 valori dalla distribuzione di Y e rappresentateli graficamente tramite un istogramma. Sovrapponete il grafico della densità della v.c. normale approssimante (qual è?). Ritenete che $n = 10$ sia sufficientemente grande per ottenere una buona approssimazione tramite il teorema limite centrale?

3.2 La legge dei grandi numeri

Se X_i $i = 1, 2, \ldots$ è una successione di variabile casuali i.i.d. con valore atteso $E(X_i) = \mu$ allora la media campionaria

$$\bar{X}_n = \frac{\sum_{i=1}^{n} X_i}{n}$$

converge quasi certamente (e dunque anche in probabilità) al valore μ.
Per convergenza quasi certa di una successione di v.c. X_i $i = 1, \ldots$ ad una costante c si intende che

$$\Pr(\lim_{n\to\infty} X_n = c) = 1.$$

Questo teorema può essere verificato empiricamente utilizzando R : fissata la variabile casuale di riferimento, si possono generare n valori casuali, calcolarne la media e iterare il procedimento aumentando n di volta in volta. Supponiamo perciò di iniziare con $n = 10$ replicazioni da $Y \sim \mathcal{P}(5)$ e calcoliamone la media aritmetica:

```
> set.seed(30)
> y<-rpois(10,5)
> mean(y)

[1] 4.5
```

Raddoppiamo ora le replicazioni

```
>  y<-c(y,rpois(10,5))
>  mean(y)

[1] 4.7
```

e raddoppiamo ancora

```
>  y<-c(y,rpois(20,5))
>  mean(y)

[1] 4.325
```

Come si può vedere, la media campionaria oscilla intorno al vero valore della media. Proviamo ad aumentare ulteriormente la sequenza di mille replicazioni

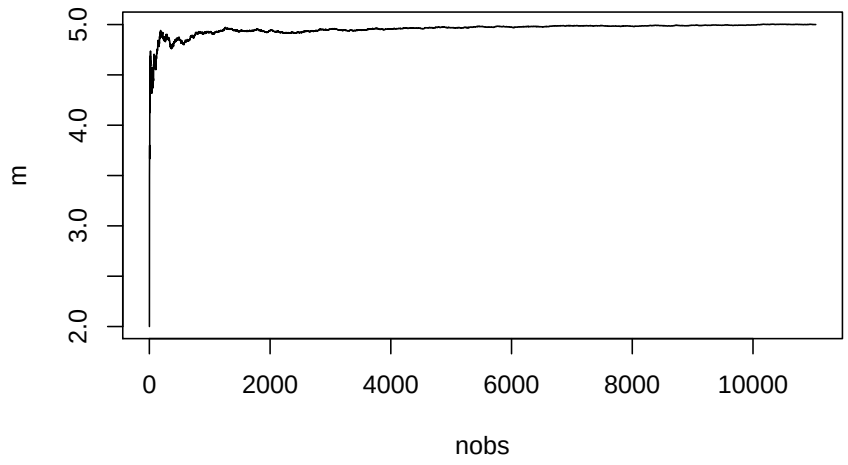
```
>  y<-c(y,rpois(1000,5))
>  mean(y)

[1] 4.907692
```

e di altre 10000 replicazioni

```
>  y<-c(y,rpois(10000,5))
>  mean(y)

[1] 5.000181
```

I risultati ottenuti confermano il fatto che la media campionaria si avvicini al vero valore della media della distribuzione di riferimento, al crescere del numero di replicazioni. È comunque chiaro che vi possono essere delle oscillazioni e velocità di convergenza diverse a seconda del caso.
Rappresentiamo in un grafico l'andamento della successione $\bar{X}_n$, $n = 1, \ldots$.

```
>  nobs<-(1:length(y))
>  m<-cumsum(y)/nobs
>  plot(nobs,m,type='l')
```

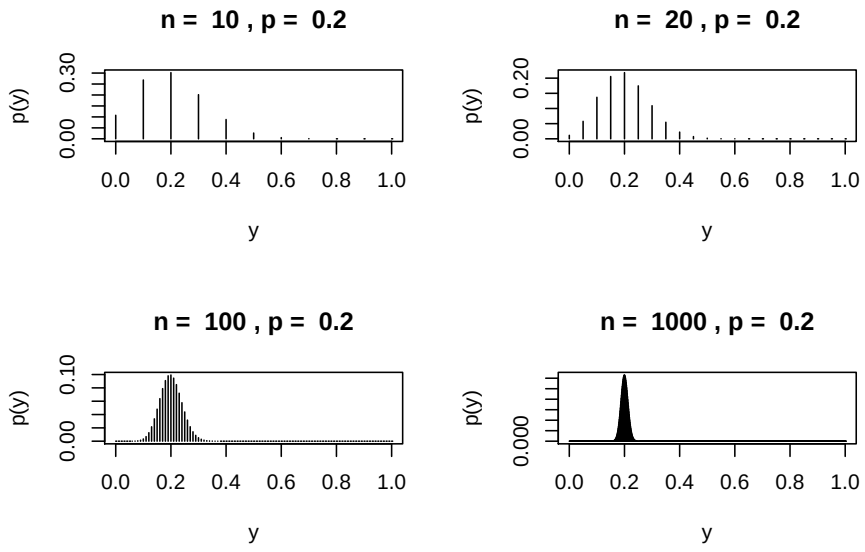


Per convergenza in probabilità di una successione di v.c. X_i $i = 1, 2, \ldots$ ad una costante c si intende che

$$\lim_{n\to\infty} \Pr(|X_n - c| > \varepsilon) = 0$$

per ogni $\varepsilon > 0$.
Illustriamo ora la convergenza in probabilità di $\bar{X}_n = \frac{\sum_{i=1}^{n} X_i}{n}$, dove $X_i \sim \mathcal{B}\backslash(1, 0.2)$ i.i.d. e quindi $\sum_{i=1}^{n} X_i \sim \mathcal{B}\backslash(n, 0.2)$.

```
> p<-0.2
> nobs<-c(10,20,100,1000)
> par(mfrow=c(2,2))
> for (n in nobs)
+ {
+ y<-0:n
+ d<-dbinom(y,n,p)
+ y<-(y/n)
+ plot(y,d,type='h',main=paste("n = ",n,", p = ",p),ylab="p(y)",xlab='y')
+ }
```



3.2.1 I metodi Monte Carlo

I metodi Monte Carlo utilizzano la legge dei grandi numeri per calcolare in modo approssimato certe quantità.
Supponiamo ad esempio di voler calcolare l'integrale

$$I(f) = \int_0^1 f(y)dy.$$

Si osservi che questo integrale può essere interpretato come il valore atteso della variabile $f(Y)$, dove $Y \sim U(0,1)$. Possiamo allora generare un gran numero di v.c. $Y_1, Y_2, \ldots, Y_n$ indipendenti e uniformemente distribuite in (0,1) e calcolare

$$\widehat{I}(f) = \frac{1}{n}\sum_{i=1} f(Y_i).$$

La legge dei grandi numeri ci garantisce che $\widehat{I}(f)$ converge in probabilità a

$$E(f(Y)) = \int_0^1 f(y)dy = I(f).$$

Ad esempio sia

$$I(f) = \int_0^1 \frac{1}{\sqrt{2\pi}} e^{-y^2/2} dy.$$

Usando la procedura appena illustrata,

```
> set.seed(1)
> n<-1000
> y<-runif(n)
> a<-mean(exp(-y^2/2))/sqrt(2*pi)
> a
```

```
[1] 0.341497
```

Possiamo controllare la bontà dell'approssimazione ottenuta, usando la funzione di ripartizione della normale:

```
> pnorm(1)-pnorm(0)
```

```
[1] 0.3413447
```

Un'altra possibile soluzione al problema precedente si basa sull'idea che, in virtù della legge dei grandi numeri, ogni probabilità p può essere approssimata da una frequenza relativa. Basta infatti considerare $X_1, X_2, \ldots, X_n$ indipendenti con distribuzione di Bernoulli con parametro p. Allora $\bar{X}_n \to E(X_1) = p$.
Utilizzando questo approccio, $P(0 \leq \mathcal{N}(0,1) \leq 1)$ si approssima generando molti valori da una normale standard e calcolando la frequenza relativa dei valori ottenuti nell'intervallo $(0,1)$.

Esercizi:

1. Approssimate, utilizzando la simulazione di valori casuali, le seguenti quantità:
   (a) `pexp(5,6)`
   (b) `qexp(0.5,6)`
   (c) `pnorm(0,1,2)`
   (d) `qnorm(0.7,1,5)`
   (e) `dbinom(2,3,0.5)`
   (f) `pbinom(2,3,0.5)`
   (g) `dpois(5,5)`
   (h) `ppois(5,4)`

2. Ricalcolate l'integrale
$$I(f) = \int_0^1 \frac{1}{\sqrt{2\pi}} e^{-y^2/2} dy$$
simulando direttamente dalla $N(0,1)$, anziché dall'uniforme. Calcolate la varianza stimata dei due stimatori. Qual è il più efficiente?

3. Stimate il valore di π usando la frequenza relativa di coppie di valori (x, y) uniformi in $(0,1)$, tali che $x^2 + y^2 \leq 1$.

3.2.2 Le catene di Markov

Una successione di variabili aleatorie $X_0, X_1, X_2, \ldots$, a valori in un insieme finito (o numerabile) $S = \{1, 2, \ldots, M\}$ detto **spazio degli stati**, è una catena di Markov (omogenea) se

$$P(X_{n+1} = j \quad | \quad X_n = i, X_{n-1} = i_{n-1}, \ldots, X_0 = i_0) = \\ = P(X_{n+1} = j | X_n = i) = p_{ij}.$$

$P = (p_{ij})_{ij}$ è la matrice di transizione e i suoi elementi p_{ij} sono le probabilità di transizione. Conoscere la matrice di transizione e la funzione di probabilità dello stato iniziale, $p^{(0)}$, permette di calcolare probabilità condizionate, congiunte e marginali della catena.
Consideriamo l'esempio del tempo nel mondo di Oz, dove ci sono tre possibili situazioni meteorologiche: 1=pioggia, 2=sole, 3=neve. La matrice di transizione che regola il tempo e la distribuzione iniziale, sapendo che oggi c'è il sole, sono:

```
> P<-matrix(c(0.5,0.5,0.25,0.25,0,0.25,0.25,0.5,0.5),3,3)
> p0<-c(0,1,0)
> P

     [,1] [,2] [,3]
[1,] 0.50 0.25 0.25
[2,] 0.50 0.00 0.50
[3,] 0.25 0.25 0.50

> p0

[1] 0 1 0
```

Le matrici di transizione a 2 e 3 passi si calcolano facilmente:

```
> P %*% P

       [,1]   [,2]   [,3]
[1,] 0.4375 0.1875 0.3750
[2,] 0.3750 0.2500 0.3750
[3,] 0.3750 0.1875 0.4375

> P %*% P %*% P

         [,1]     [,2]     [,3]
[1,] 0.406250 0.203125 0.390625
[2,] 0.406250 0.187500 0.406250
[3,] 0.390625 0.203125 0.406250
```

e così pure le distribuzioni marginali al tempo 1, 2 e 3:

```
> p0 %*% P
```

```
     [,1] [,2] [,3]
[1,]  0.5    0  0.5
```

```
> p0 %*% (P %*% P)
```

```
      [,1] [,2]  [,3]
[1,] 0.375 0.25 0.375
```

```
> p0 %*% (P %*% P %*% P)
```

```
        [,1]   [,2]    [,3]
[1,] 0.40625 0.1875 0.40625
```

Utilizzando la funzione `Markov()` della libreria di R *labstat*, possiamo simulare una possibile situazione meteorologica nel mondo di Oz per i prossimi 15 giorni:

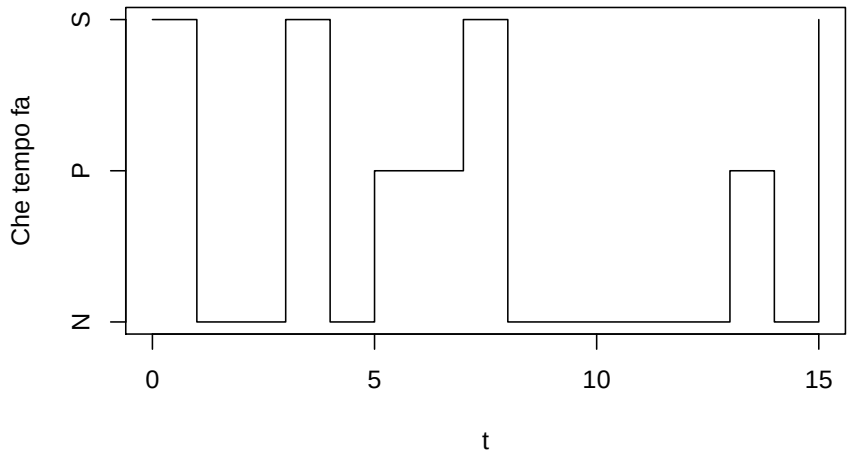
```
> library(labstatR)
> ?Markov
```

```
> x<-c("P","S","N")
> traj<-Markov("S",15,x,P)
> traj
```

```
$X
 [1] "S" "N" "N" "S" "N" "P" "P" "S" "N" "N" "N" "N" "N" "P"
[15] "N" "S"

$t
 [1]  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
```

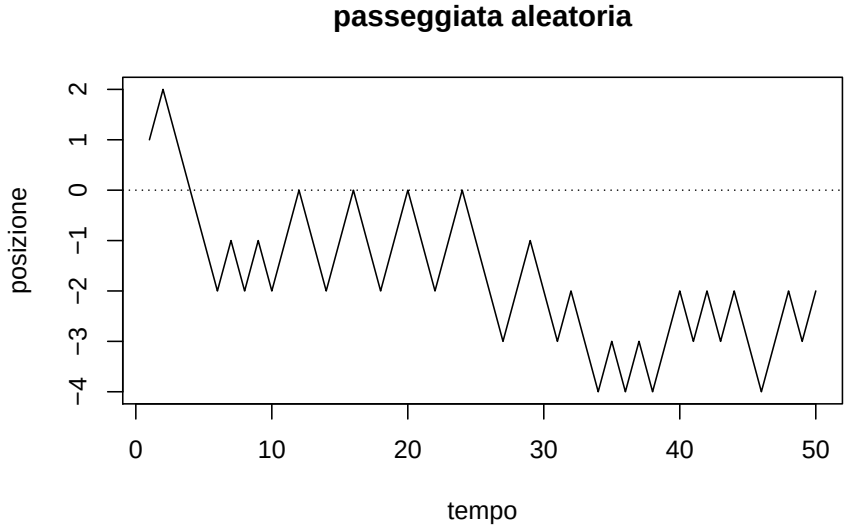
Una volta simulata la traiettoria, possiamo rappresentarla graficamente:

```
> plot(traj$t,unclass(factor(traj$X)),type="s",axes=FALSE,
+   xlab="t",ylab="Che tempo fa")
> axis(1)
> axis(2,c(1,2,3),levels(factor(traj$X)))
> box()
```



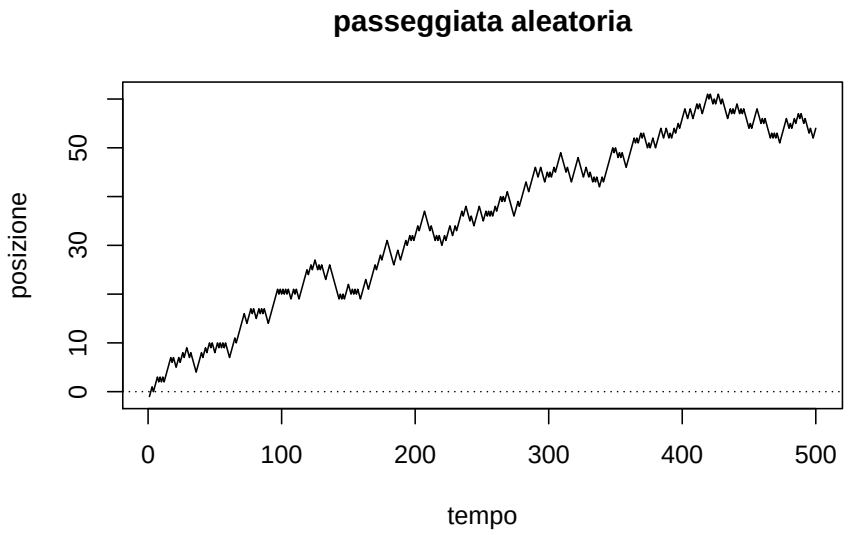
Consideriamo ora una passeggiata aleatoria, ovvero una catena di Markov con spazio degli stati $S = \mathbb{Z}$ che ad ogni istante si muove di un passo a destra o a sinistra con probabilità rispettivamente p e $1 - p$. Fissiamo $p = 0.5$ per iniziare:

```
> n<-50
> x<-rbinom(n,1,0.5)
> x[x==0]<--1
> x

 [1]  1  1 -1 -1 -1 -1  1 -1  1 -1  1  1 -1 -1  1  1 -1 -1
[19]  1  1 -1 -1  1  1 -1 -1 -1  1  1 -1 -1  1 -1 -1  1 -1
[37]  1 -1  1  1 -1  1 -1  1 -1 -1  1  1 -1  1

> y<-cumsum(x)
> y

 [1]  1  2  1  0 -1 -2 -1 -2 -1 -2 -1  0 -1 -2 -1  0 -1 -2
[19] -1  0 -1 -2 -1  0 -1 -2 -3 -2 -1 -2 -3 -2 -3 -4 -3 -4
[37] -3 -4 -3 -2 -3 -2 -3 -2 -3 -4 -3 -2 -3 -2

> plot(1:n,y,type="l",main="passeggiata aleatoria",xlab="tempo",ylab="posizione")
> abline(h=0,lty=3)
```



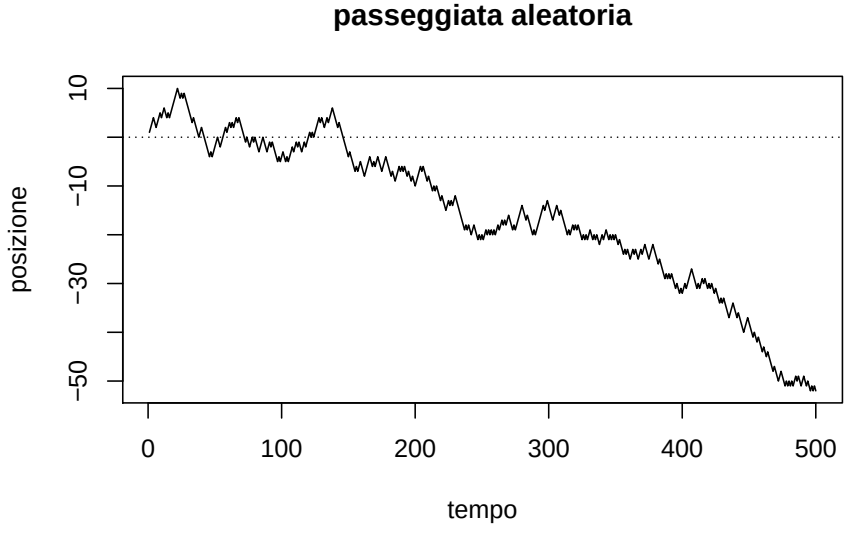
Cambiamo ora leggermente la probabilità ponendo $p = 0.51$:

```
> n<-500
> x<-rbinom(n,1,0.51)
> x[x==0]<--1
> y<-cumsum(x)
> plot(1:n,y,type="l",main="passeggiata aleatoria",xlab="tempo",ylab="posizione")
> abline(h=0,lty=3)
```



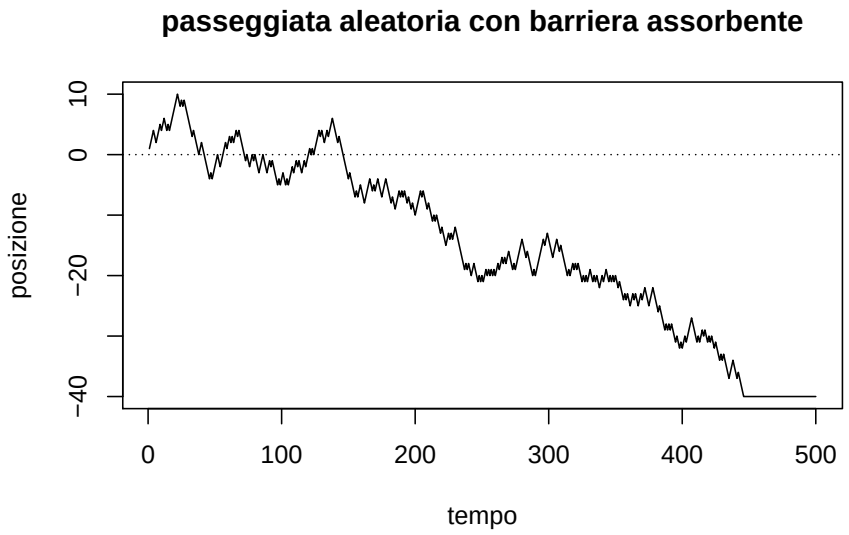
Se $p = 0.45$:

```
> n<-500
> x<-rbinom(n,1,0.45)
> x[x==0]<--1
> y<-cumsum(x)
> plot(1:n,y,type="l",main="passeggiata aleatoria",xlab="tempo",ylab="posizione")
> abline(h=0,lty=3)
```



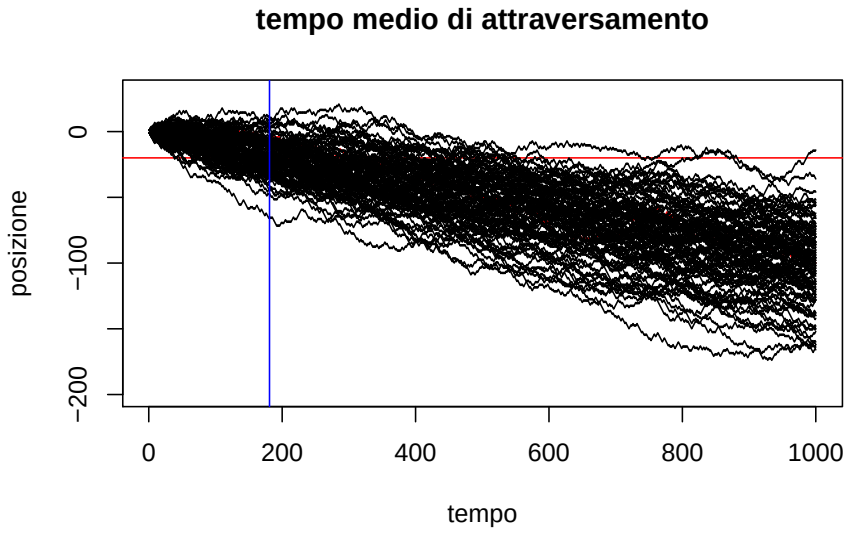
E' anche possibile fissare delle barriere assorbenti:

```
> L<--40
> t1<-min(which(y==L))
> y[t1:500]<-L
> plot(1:n,y,type="l",main="passeggiata aleatoria con barriera assorbente",xlab="tem
> abline(h=0,lty=3)
```



Utilizzando la simulazione e i metodi Monte Carlo, possiamo stimare il tempo medio di primo attraversamento della barriera. Lo facciamo calcolando il tempo di primo attraversamento della barriera per moltissime traiettorie e calcolando poi la media dei tempi ottenuti in ogni simulazione.

```
> set.seed(1700)
> M<-100
> n<-1000
> L<--20
> t<-numeric(M)
> x<-2*rbinom(n,1,0.45)-1
> y<-cumsum(x)
> t[1]<-min(which(y==L))
> plot(1:n,y,type="l",main="tempo medio di attraversamento",xlab="tempo",
+          ylab="posizione",col="red",ylim=c(-200,30))
> abline(h=L,col="red")
> for(i in 2:M)
+ {
+         x<-2*rbinom(n,1,0.45)-1
+         y<-cumsum(x)
+         t[i]<-min(which(y==L))
+         #controllo per Inf
+         lines(1:n,y)
+ }
> mean(t)

[1] 181.04

> abline(v=mean(t),col="blue")
```



Esercizi:

1. Si scriva una funzione di R che simuli una traiettoria di lunghezza 500 della passeggiata aleatoria con probabilità di salire (+1) pari a 0.4 ad ogni passo.
   (a) Qual è l'andamento della traiettoria simulata?
   (b) Si calcoli il primo tempo in cui la traiettoria simulata tocca la barriera -30.
   (c) Si ripetano i passi precedenti per 1000 volte e si stimi usando il metodo Monte Carlo il tempo medio di raggiungimento della barriera -30.
2. Si scriva una funzione di R che simuli una traiettoria di lunghezza 500 della passeggiata aleatoria con probabilità di salire (+1) pari a 0.7 ad ogni passo.
   (a) Qual è l'andamento della traiettoria simulata?
   (b) Si calcoli il primo tempo in cui la traiettoria simulata tocca la barriera 30.
   (c) Si ripetano i passi precedenti per 1000 volte e si stimi, usando un metodo Monte Carlo, il tempo medio di raggiungimento della barriera 30.
3. In una città ogni giorno il tempo è soleggiato (0) o piovoso (1). Un giorno di sole è seguito da un altro giorno di sole con probabilità 0.8, mentre un giorno di pioggia è seguito da uno di sole con probabilità 0.3.
   (a) Lunedì piove. Si simuli una possibile previsione per l'intera settimana.
   (b) Lunedì piove. Si stimi, usando un metodo Monte Carlo, quanti giorni si dovranno attendere in media per avere il primo giorno di sole.
   (c) Si calcoli una distribuzione stazionaria per la catena di Markov.

4. In una città ogni giorno il tempo è soleggiato (0) o piovoso (1). Un giorno di sole è seguito da un altro giorno di sole con probabilità 0.2, mentre un giorno di pioggia è seguito da uno di sole con probabilità 0.6.

   (a) Lunedì c'è il sole. Si simuli una possibile previsione per l'intera settimana.

   (b) Lunedì c'è il sole con probabilità pari a 0.8. Si stimi, usando un metodo Monte Carlo, la probabilità che giovedì ci sia il sole.

   (c) Si calcoli una distribuzione stazionaria per la catena di Markov.