Applicazioni **PARTE I** Modelli Statistici II Corso di Laurea Magistrale in Biostatistica

Prof.ssa Fulvia Pennoni fulvia.pennoni@unimib.it

Ottobre 2022

Contents

Queste dispense integrano quelle della parte di Teoria del file Pennoni F. (2020). *Dispense dell'insegnamento Modelli Statistici II*, Corso di Laurea Magistrale in Biostatistica, Dipartimento di Statistica e Metodi Quantitativi, Università degli Studi di Milano-Bicocca.

Introduzione a `R`

La statistica attualmente si sviluppa sempre di più nel suo aspetto computazionale che è stato inaugurato negli anni '50 con i primi linguaggi di programmazione (nel 1952 Grace Hopper propone il primo linguaggio di programmazione simbolico, che si sviluppa poi nel linguaggio di programmazione noto come FORTRAN). Sotto tale aspetto viene definita *computational statistics o statistical computing*.

L'idea di fondo dei sistemi statistici attuali come SAS e `R` è quella di permettere le elaborazioni dei dati con uno specifico linguaggio di utilizzazione e di programmazione.

In particolare il linguaggio di programmazione di `R` rientra tra i linguaggi definiti *programming languages* o linguaggi matriciali che si avvalgono di strumenti algebrici per definire oggetti di due o più dimensioni (array, costituiti da vettori e matrici).
Alcuni vantaggi nell'utilizzo di `R` sono i seguenti: è open-source, è efficiente per strutture di programmazione sofisticate, consente di integrare anche altri linguaggi di programmazione, dispone di una buona interfaccia grafica. Tra gli ambienti sviluppati per `R` (Integrated Development Environments) c'è `RStudio` https://www.rstudio.com/ che ha un editor di testo; editor dei dati; etc…

Attualmente `RStudio` (Rstudio, 2020) ha la possibilità di utilizzare un marcatore di linguaggio denominato Markdown. Il Markdown permette di rendere le analisi dei dati *riproducibili*: nello stesso file è presente il codice (script), i risultati dell'elaborazione ed il relativo commento. La libreria denominata `knitr` (Yihui Xie, 2013) consente di produrre documenti in HTML in cui codici e descrizioni possono essere integrati. Si parte da un file sorgente che ha estensione `.Rmd` che viene valutato dalla funzione `knit` e restituisce come output un file integrato di testo e grafici.

`R` può essere definito come un sistema di analisi statistica, che è contemporaneamente un linguaggio ed un software o meglio un ambiente. E' un linguaggio orientato agli oggetti. Infatti ogni oggetto (es. vettore, grafico, ecc.) viene trattato dalle funzioni di `R` con uno specifico metodo (ovvero un codice di programma per gestire e manipolare le strutture dati) e nuovi metodi possono essere implementati per ampliare le possibilità delle stesse funzioni.

Una caratteristica di `R` è quella di essere *Open Source* dal 1995. Tale caratteristica implica tra le altre che si ha la possibilità di accedere al codice sorgente e di modificarlo; dispone di una vasta manualistica (in lingua inglese) consultabile e scaricabile da Internet; è possibile accedere tramite Internet ad una vasta gamma di librerie molto dettagliate per le analisi statistiche, create e messe a disposizione da altri utenti.

E' possibile contare sul supporto e sull'assistenza dell'`R` Development Core Team e di tutti gli utenti di `R` (una vera community a livello mondiale) grazie al sito Internet e alle liste di discussione, tramite le quali ci si può avvalere dell'aiuto di esperti in statistica e informatica.

E' distribuito gratuitamente sotto i vincoli della GPL (General Public License) che è una licenza per il software libero che permette all'utente libertà di utilizzo, copia, modifica e distribuzione.

E' basato sul linguaggio `S` sviluppato da John Chambers e altri del Bell Laboratories nel 1976 ed è disponibile per diverse architetture hardware e sistemi operativi: Unix, Linux, Windows, MacOS. Può essere scaricato dal sito `http://cran.r-project.org` dove `CRAN` è l'acronimo di "Comprehensive `R` Archive Network". La versione da installare per l'uso più comune è la binary version (vesione binaria).

Si consideri che se si incontrano difficoltà nella programmazione è possibile porre domande o cercare risposte presso http://stackoverflow.com/questions.

Si veda anche http://rseek.org/ per trovare informazioni ed esempi.

https://pragprog.com/about per indicazioni sulle modalità di programmazione e http://www.r-bloggers.com/

Sintassi di base per `R`

La directory di lavoro può essere scelta dall'utente. La directory corrente di lavoro si verifica con il comando `getwd` mentre con `setwd` ci si sposta tra le directory. Con `dir()` si verifica il contenuto della directory in uso.

```
getwd()
setwd("C:/testo")
dir
```

Le espressioni aritmetiche possono essere sviluppate quando sulla *console* di `R` compare compare il simbolo `>`.

Per chiudere la sessione si usa `q()`. Se si digita invio viene chiesto se salvare o meno il workspace. Tale workspace contiene un tracciato dei comandi che sono stati digitati durante la sessione e può contenere i risultati salvati. Generalmente questo non viene salvato, ciò che si salva è lo script finale ovvero il file di testo contenente tutte le istruzioni da eseguire. L'editor per creare uno script appare dal menu File. I comandi dello script vengono eseguiti in `R` con il comando Ctrl-R.

Alcuni calcoli con `R`:

```
3^4
#> [1] 81
7 * 4 + 2
#> [1] 30
```

```
31/7
#> [1] 4.428571
31%/%7
#> [1] 4
```

R ha un workspace noto come *global enviroment* (ambiente globale) che viene usato per contenere i risultati dei calcoli che si svolgono e gli oggetti che si generano. Ad esempio se si vuole salvare il risultato del calcolo 1.0025^{30} occorre assegnare il valore ad un oggetto a cui si assegna un nome ad esempio `int.30`

```
 1.0025^30
#> [1] 1.077783
int.30<-1.0025^30
```

l'assegnazione avviene con una freccia orientata a sinistra ad esempio utilizzando il simbolo minore seguito dalla lineetta. Anche il segno uguale viene supportato da R ma è meglio usare la freccia per precisare che si richiede un azione invece di stabilire una relazione. Il risultato dell'assegnazione non è visibile se non digitando il nome dell'oggetto, ricordando che R è case sensitive ovvero sensibile alle maiuscole: il seguente codice fornisce un errore

```
X <- 1:10
mean(x)
```

Quando non risulta definito nel workspace l'oggetto per cui è richiesto il calcolo viene fornito il seguente messaggio di errore: `Errore in mean(x) : oggetto "x" non trovato`.

La funzione `mean()` implementata in R permette il calcolo della media aritmetica.

```
X <- 1:10
mean(X)
#> [1] 5.5
```

Il comando `objects()` oppure `ls()` consente di visualizzare tutti gli oggetti di R creati che sono presenti nel workspace. Il comando `c()` serve per concatenare gli elementi che vengono forniti come argomento

```
c(0, 7, 8)
#> [1] 0 7 8
x <- c(0, 7, 8)
```

Il comando serve anche per concatenare dei vettori nel seguente modo

```
num <- 5:20
a.gruppo <- c(num, x)
a.gruppo
#>  [1]  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20  0  7  8
```

Per visualizzare alcuni elementi del vettore si usano le parentesi quadrate

```
a.gruppo[20]
#> [1] NA
```

L'uso dei numeri negativi serve per evitare di considerare alcuni elementi, ad esempio il secondo elemento dell'oggetto `a.gruppo`

```
 a.gruppo[-2]
#>  [1]  5  7  8  9 10 11 12 13 14 15 16 17 18 19 20  0  7  8
```

Le operazioni tra vettori sono fatte elemento per elemento

```
 x * 3
#> [1]  0 21 24
y <- x - 5
```

L'elevamento a potenza del tipo $(y_1^{x_1}, y_2^{x_2}, ...)$ si ha con

```
 x^3
#> [1]   0 343 512
```

```
 y^x
#> [1]    1  128 6561
```

Esistono alcuni operatori che permettono di generare delle sequenze di numeri interi: `seq` e `rep`. Il primo comando consente di specificare il passo della successione

```
seq(1, 21, by = 2)
#>  [1]  1  3  5  7  9 11 13 15 17 19 21
```

```
rep(3, 12)
#>  [1] 3 3 3 3 3 3 3 3 3 3 3 3
```

Ad esempio si noti l'utilizzo delle funzioni `rep` e `seq` per creare il vettore seguente

1 2 3 4 5 2 3 4 5 6 3 4 5 6 7 4 5 6 7 8 5 6 7 8 9

```
rep(seq(1, 5), times = 5) + rep(0:4, each = 5)
#>  [1] 1 2 3 4 5 2 3 4 5 6 3 4 5 6 7 4 5 6 7 8 5 6 7 8 9
```

Per sistemare dei valori in una matrice si usa la funzione `matrix()`

```
m <- matrix(1:6, nrow = 2, ncol = 3)
m[1, 2]
#> [1] 3
m[4]
#> [1] 4
m[1, ]
#> [1] 1 3 5
```

Diversa è invece la funzione `array()`

```
array(1:24, c(3, 4, 2))
#> , , 1
#>
#>      [,1] [,2] [,3] [,4]
#> [1,]    1    4    7   10
#> [2,]    2    5    8   11
#> [3,]    3    6    9   12
#>
#> , , 2
#>
#>      [,1] [,2] [,3] [,4]
#> [1,]   13   16   19   22
#> [2,]   14   17   20   23
#> [3,]   15   18   21   24
```

La funzione `dump` serve per salvare un file di dati in formato .`R` e `source` per richiamarli.

```
a <- c(2, 3, 4)
dump("a", "uso.R")
source("uso.R")
```

Un data frame corrisponde a quello che viene definito come una matrice di dati o un data set. Ovvero si tratta di una lista di vettori o di fattori della stessa lunghezza che sono in relazione dato che corrispondono alla stessa unità sperimentale ed è pertanto presente un unico insieme di nomi per la riga.

E' possibile estrarre degli oggetti dal data frame nel seguente modo

```
 x <- c(1, 2, 3)
 y <- c(1, 2, 3)
 mio <- data.frame(x, y)
mio$x
#> [1] 1 2 3
```

Quando i dati sono strutturati come una matrice in cui in ogni riga vi sono le osservazioni relative ad ogni individuo e le colonne rappresentano le variabili osservate assomigliano alla struttura dei data frame di `R`. La prima riga può contenere i nomi delle variabili e la prima colonna le etichette assegnate alle osservazioni. Se ad esempio il data set è scritto in formato testo e si trova nel file prova.dat nella directory C allora viene letto da `R` come

```
prova.df <- read.table("c:/prova.dat", header = T)
```

Se file contiene l'intestazione questo viene specificato con `header=TRUE` se invece non la contiene `header=FALSE`. Per lavorare direttamente con le colonne dei dati caricati si usa comando `attach`. I dati caricati possono anche essere visualizzati con l'editor che si trova nella finestra Modifica.

Cenni su operatori logici e funzioni

I simboli TRUE e FALSE sono i risultati della condizione logica che viene posta. Ad esempio operatori relazionali sono i seguenti

```
a <- c(3, 6, 9)
a > 4
#> [1] FALSE  TRUE  TRUE
```

Si noti che la condizione logica restituisce un vettore vero o falso che passato alla funzione `which` restituisce quanto segue

```
x <- c(3, 6, 9, 10, 20, 30, 21, 2, 1, 4)
x > 10
#>  [1] FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE FALSE FALSE FALSE
which(x > 10)
#> [1] 5 6 7
```

Ovvero `which` restituisce gli indici di un vettore i cui elementi contengono TRUE o FALSE.

Altri usi di `which` sono i seguenti

```
x <- c(3, 6, 9, 10, 20, 30, 21, 2, 1, 4)
which((x > 2) & (x < 20))
#> [1]  1  2  3  4 10
z <- which((x < 2) | (x > 7))
z
#> [1] 3 4 5 6 7 9
x[z]
#> [1]  9 10 20 30 21  1
```

Poi vengono visualizzati gli elementi del vettore che soddisfano la condizione.

Si notino anche gli altri opratori

```
a == 4
#> [1] FALSE FALSE FALSE
a >= 4
#> [1] FALSE  TRUE  TRUE
a != 4
#> [1] TRUE TRUE TRUE
a[a > 4]
#> [1] 6 9
```

```
b <- c(4, 6, 8)
a < b
#> [1]  TRUE FALSE FALSE
a[a < b]
#> [1] 3
```

significa {non a}

```
a <- c(TRUE, FALSE, FALSE, TRUE)
!a
#> [1] FALSE  TRUE  TRUE FALSE
```

Help

Per accedere alla guida di `R` ci sono molte opzioni. Quando si conosce il nome della funzione per cui si vuole consultare la guida basta digitare

```
`?`(mean)
 help(mean)
```

viene aperta una nuova finestra che contiene la descrizione completa del comando ed in fondo vengono riportati una serie di utili esempi di utilizzo della funzione.

Un'altra possibilità è la funzione `example()`:

```
 example(mean)
#>
#> mean> x <- c(0:10, 50)
#>
#> mean> xm <- mean(x)
#>
#> mean> c(xm, mean(x, trim = 0.10))
#> [1] 8.75 5.50
```

vengono mostrati tutti i possibili utilizzi della funzione `mean`.

L'opzione `trim` consente di calcolare la media troncata ovvero calcolata non tenendo conto dei valori estremi della distribuzione. Il valore di `trim` è compreso tra 0 e 0.5 e consiste nella proporzione di osservazioni che `R` scarta prima di effettuare la media in ciascuna coda della distribuzione.

Quando non si conosce il nome della funzione è conveniente utilizzare `help.start()`. Si apre una pagina del browser in cui c'è un menu con alcune opzioni. Un'altra funzione è `help.search()` ad esempio `help.search(mean)`

Programmazione

Programmare consiste nello scrivere delle istruzioni più o meno complesse con una sintassi ben definita. I due principali modi di programmare sono quello imperativo, usato ad esempio da `R`, che consiste nel legare insieme delle istruzioni che specificano le cose che il calcolatore deve fare; quello dichiarativo usato ad esempio per le pagine WEB come l'HTML che consiste nello scrivere una descrizione del risultato finale senza specificare come questo deve essere raggiunto. Il metodo imperativo in `R` viene definito procedurale ovvero comporta la descrizione di ogni passo necessario per raggiunre un certo obbiettivo. Tra i costrutti di base di un programma si trovano i seguenti test che permettono la verifica di alcune condizioni nel flusso degli algoritmi.

- Alcune regole generali di programmazione sono le seguenti:
- [1] capire bene l'argomento;
- [2] esplicitare l'idea la risoluzione del problema;
- [3] tradurre in implementazione l'idea;

- [4] controllare: se funziona, se è efficiente, se non è tale tornare al secondo punto.
- Il ciclo `if()` è un costrutto condizionale che consente di controllare quali operazioni vengono eseguite. La sintassi è la seguente

`if (condizione) {comandi quando la condizione è vera}` oppure `if (condizione) comandi quando la condizione è vera else comandi quando è falsa`.

Ad esempio

```
x <- 3
if (x > 2) y <- 2 * x else y <- 3 * x
 y
#> [1] 6
```

I valori logici sono convertiti in valori numerici secondo la regola che FALSE diventa 0 e TRUE diventa 1.

- Il ciclo iterativo `for()` consente di speficare che una certa operazione deve essere ripetuta un certo numero di volte predefinito. La sintassi è la seguente \

`for(variabile a valori in un insieme) {esegui i comandi}`.

Ad esempio

```
for (i in 1:9) print(1:i)
#> [1] 1
#> [1] 1 2
#> [1] 1 2 3
#> [1] 1 2 3 4
#> [1] 1 2 3 4 5
#> [1] 1 2 3 4 5 6
#> [1] 1 2 3 4 5 6 7
#> [1] 1 2 3 4 5 6 7 8
#> [1] 1 2 3 4 5 6 7 8 9
```

Nell'esempio è stato stampato di cosa si occupa il ciclo for: di considerare i numeri da 1 a i con i che varia nell'insieme $[1, 9]$.

Esempio

si consideri la successione di Fibonacci che è una nota sequenza di numeri interi naturali in cui i primi due elementi sono definiti come $[1, 1]$ ed i successivi sono la somma dei due elementi precedenti. Ad esempio il terzo elemento è 2 (= 1+1), il quarto elemento è 3 (= 1+2). In `R` per ottenere i primi 12 numeri di Fibonacci si usa la seguente sintassi

```
Fibonacci <- numeric(12)
 Fibonacci[1] <- Fibonacci[2] <- 1
 for (i in 3:12) Fibonacci[i] <- Fibonacci[i - 2]  Fibonacci[i -1]
```

Si crea il vettore nullo `Fibonacci`. Si inseriscono i numeri 1 nelle prime due posizioni del vettore. Con il ciclo si determinano gli elementi dal terzo in poi. Come si nota è necessario inizializzare il vettore contenente i risultati dell'operazione.

- Il ciclo `while()` consente di ripetere certe operazioni fintantoché vale la condizione stabilita. Quando la condizione non è più valida si esce dal ciclo: se la condizione viene valutata come falsa non viene compiuta nessuna azione.
  La sintassi è la seguente

`while (condizione) { esegui i comandi }`.
Se viene valutata come vera i comandi vengono eseguiti ed il processo viene ripetuto rivalutando la condizione ogni volta.

Volendo scrivere la sequenza di Fibonacci che arrivi fino a 300 numeri si può usare il ciclo `while` per impostare tale condizione. Ad esempio

```
Fib1 <- 1
Fib2 <- 1
 Fibonacci <- 1
 while (Fib2 < 300) {
     Fibonacci <- c(Fibonacci, Fib2)
     oldFib2 <- Fib2
     Fib2 <- Fib1 + Fib2
     Fib1 <- oldFib2
 }
 Fibonacci
#>  [1]   1   1   2   3   5   8  13  21  34  55  89 144 233
```

Fino a che `Fib2` è minore di 300 il vettore `Fibonacci` viene incrementato. `Fib2` viene aggiornato aggiungendo `Fib1` che a sua volta è aggiornato con il valore precedente di `Fib2`.

Le funzioni

Le funzioni sono delle unità costruite a partire dai comandi elementari. In generale sono oggetti che ricevono degli input, svolgono dei calcoli e producono un risultato.

Le funzioni sono degli oggetti che possono essere modificati. In `R` la funzione ha la seguente struttura:

`function (argomenti) { esegui i comandi }`.

La parola function è seguita dalle parentesi tonde che racchiudono la lista degli argomenti e poi dalle parentesi graffe che contengono un comando o una sequenza di comandi. Ogni funzione ha un nome proprio con cui viene richiamata.

La funzione dell'esempio seguente ha come nome `Eratostene`. Tale funzione prende come argomento `n` che specifica il limite superiore della sequenza dei numeri del vettore `crivello`. Gli argomenti della funzione possono essere molteplici. Si possono anche assegnare dei valori di default ad alcuni argomenti tali che se nell'uso della funzione non si specifica il loro valore viene considerato il valore di default.

Esempio

Il crivello di Eratostene è un antico procedimento per il calcolo delle tabelle di numeri primi fino ad un certo numero n prefissato. È a tutt'oggi utilizzato come algoritmo di calcolo dei numeri primi da molti programmi per computer; pur non essendo un algoritmo straordinariamente efficiente, infatti, è in compenso piuttosto semplice da tradurre in un qualsiasi linguaggio di programmazione. Si scrivono tutti i naturali a partire da 2 fino n in un elenco. Poi si cancellano (setacciano) tutti i multipli del primo numero del setaccio (escluso lui stesso). Ad esempio iniziando da 2 si cancellano i multipli di 2. Ci si sposta all'elemento successivo 3 e si cancellano i suoi multipli. Si prosegue così fino ad arrivare in fondo. I numeri che restano sono i numeri primi minori od uguali a n.

```
Eratostene <- function(n) {
     if (n >= 2) {
        crivello <- seq(2, n)
         primi <- c()
         for (i in seq(2, n)) {
             if (any(crivello == i)) {
                 primi <- c(primi, i)
                 crivello <- c(crivello[(crivello%%i) != 0], i)
             }
         }
         return(primi)
     }
     else {
```

```
        stop("il valore iniziale deve essere almeno 2")
    }
}
Eratostene(50)
#>  [1]  2  3  5  7 11 13 17 19 23 29 31 37 41 43 47
```

Il vettore crivello contiene la lista di numeri che devono essere valutati ovvero i numeri interi da 2 a n. L'oggetto `primi` è un oggetto vuoto creato per contenre il vettore dei numeri primi che verranno trovati.

Ogni numero intero `i` nella sequenza viene controllato per vedere se è di nuovo nel vettore attraverso la condizione `if`. La funzione `any()` fornisce un risultato VERO (TRUE) se almeno 1 degli elementi logici risulta vero. Se l'elemento è vero significa che tale elemento è ancora nel vettore `crivello`.

Tutti i multipli di `i` vengono eliminati dato che sono necessariamnete composti e `i` viene attaccato a `primi`. L'espressione

```
(crivello%%i) == 0
```

risulta VERA per gli elementi di `crivello` che sono multipli di `i`. In pratica si calcola il resto della divisione intera tra ogni elemento di `crivello` e `i` con %% e si valuta se questo è uguale a zero. Dato che gli elementi pari a zero devono essere eliminati ed i restanti devono essere salvati si utilizza la condizione contraria salvando salvando tutti gli elementi diversi da zero != ovvero si usa

```
crivello[(crivello%%i)!=0]
```

ma in questo modo anche `i` viene eliminato il quale tuttavia è stato precedentemente salvato in primi. Pertanto con l'ultima espressione si eliminano tutti i multipli di `i` dal vettore `crivello`.

Nel corpo della fuzione potrebbe trovarsi un comando `return(valore)` che specifica e stampa il risultato della funzione. In `R` ogni funzione produce un solo output. Se il return non è specificato allora viene stampato il valore dell'ultimo comando presente nel corpo della funzione.

Nell'esempio sopra sono stati creati degli oggetti locali interni alla funzione utilizzati come: `n`, `sieve` e `primes` che sono oggetti locali interni alla funzione e poi sono stati richiamate le funzioni: `seq`, `c`, `any`, `return` e `stop` che fanno parte dell'ambiente di `R`.

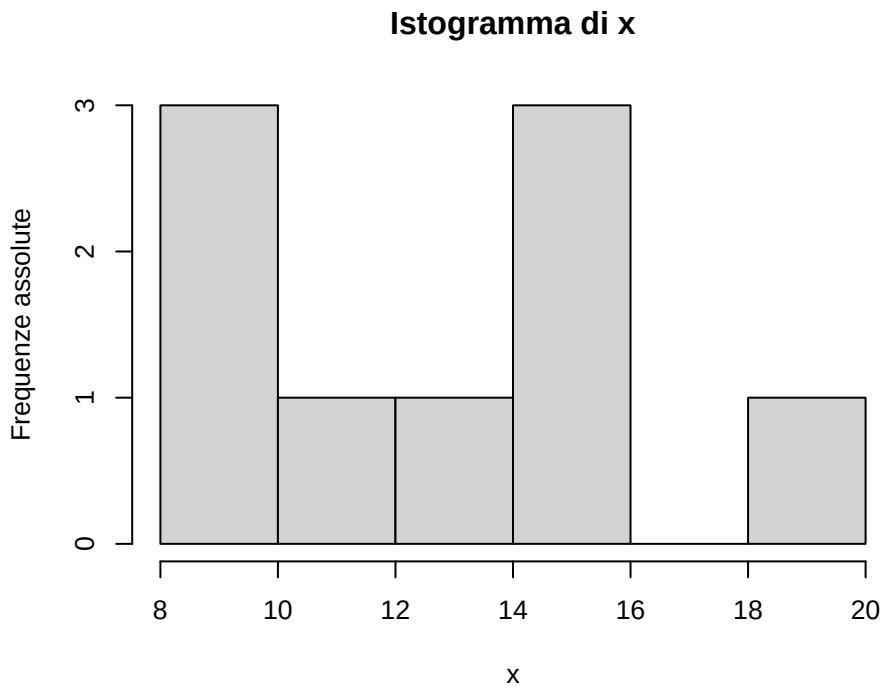
Per correggere gli errori che possono presentarsi quando si scrive la funzione si usa il comando `fix()`. Tale comando apre in un file di testo il testo della funzione.

Per inserire i commenti e descrizioni nella funzione si usa `#` in modo che `R` ignora i tutto il testo che segue (quando si utilizza il file di sintassi .R) cancelletto fino alla fine della riga.

Rappresentazioni grafiche

L'istogramma è uno strumento per rappresentare graficamente i dati di fenomeni quantitativi continui.

```
 x <- c(12, 15, 20, 14, 16, 10, 10, 8, 15)

a <- hist(x,
     main = "Istogramma di x",
     ylab = "Frequenze assolute")
```




```
a
#> $breaks
#> [1]  8 10 12 14 16 18 20
#>
#> $counts
#> [1] 3 1 1 3 0 1
#>
#> $density
```

```
#> [1] 0.16666667 0.05555556 0.05555556 0.16666667 0.00000000 0.05555556
#>
#> $mids
#> [1]  9 11 13 15 17 19
#>
#> $xname
#> [1] "x"
#>
#> $equidist
#> [1] TRUE
#>
#> attr(,"class")
#> [1] "histogram"
```

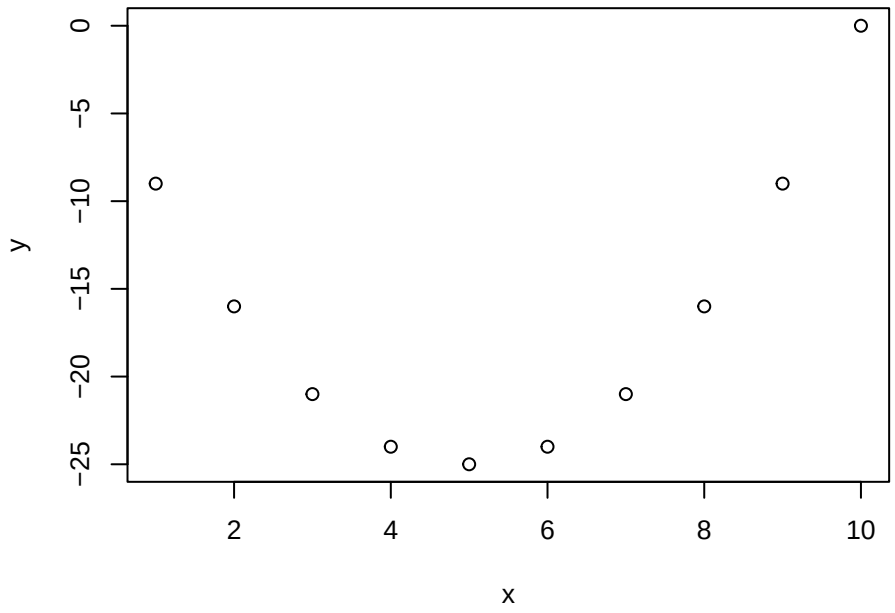
Gli istogrammi hanno base proporzionale agli estremi delle classi e altezza dei rettangoli proporzionali alla densità di frequenza (altezza/base). Con il comando `hist()` `R` disegna in modo automatico l'istogramma. L'oggetto di nome `a` contiene gli oggetti `breaks` che rappresentano gli estremi degli intervalli, `counts` sono le frequenze assolute, `mids` sono i punti centrali degli intervalli, `xname` è il nome della variabile, `equidist` è una variabile logica che vale TRUE solo nel caso in cui tutti gli intervalli hanno stessa ampiezza. Le `intensities` rappresentano le densità di frequenza associata a ciascun punto dell'intervallo corrispondente. Si possono modificare le opzioni di default ad esempio si può stabilire il numero delle classi con le sintassi seguente

`hist(W, break=11, main= "11 classi")`.

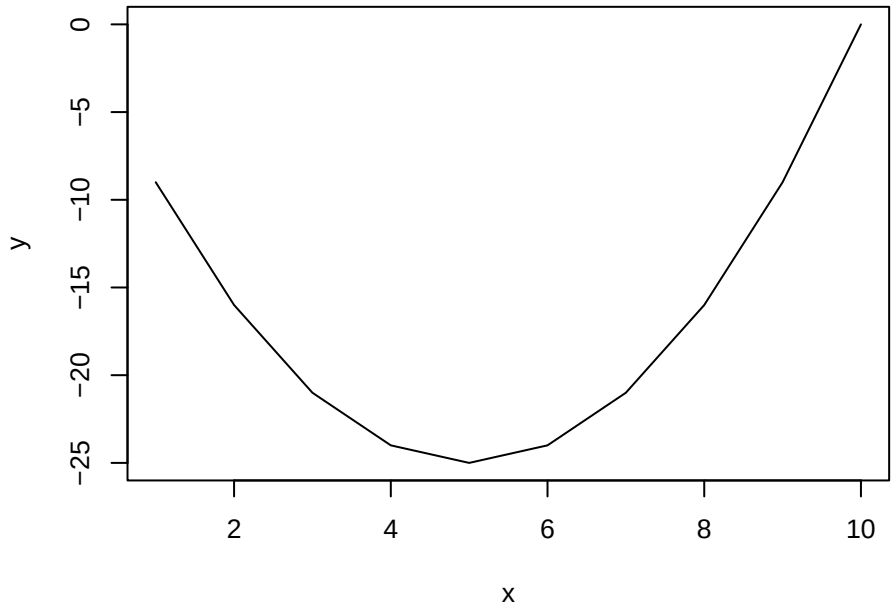
Esistono varie regole empiriche per determinare il numero degli intervalli che prendono il nome dagli autori che le hanno proposte. `R` implementa la regola di Sturges che avendo n valori di x divide il campo di variazione dei dati per $\log_2(n)+1$ intervalli, in pratica si ha $\frac{x_{(n)}-x_{(1)}}{\log_2(n)+1}$.
Se n è elevato il metodo comporta un numero molto elevato di classi. Altre opzioni sono `breaks = "Scott"` e `breaks = Freedman-Diaconis`.

Lo scatter plot o diagramma a dispersione si genera come segue.

```
x <- seq(1, 10)
y <- x^2 - 10 * x
plot(x, y)
```

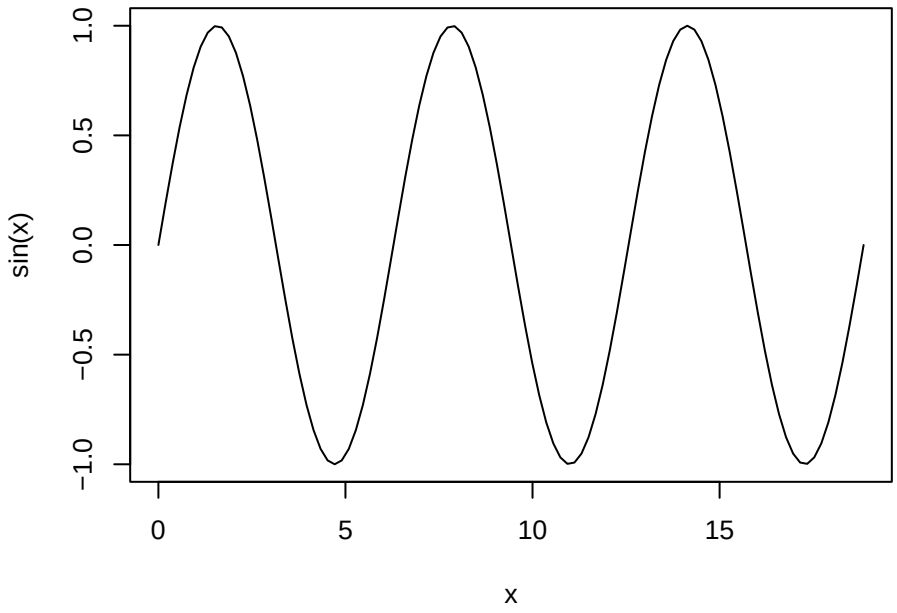



```
plot(x, y, type = "l")
```



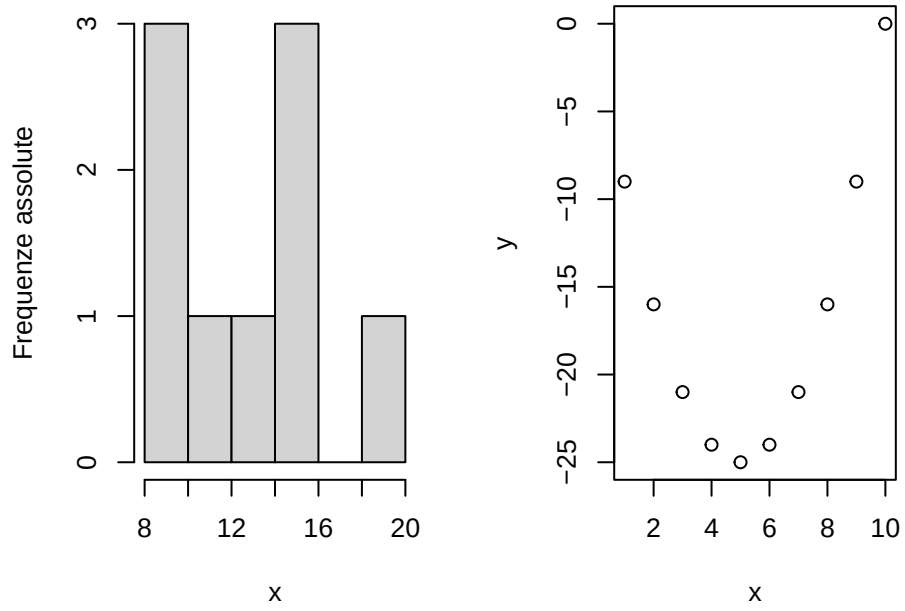
Un'altra funzione importante è `curve` che serve per disegnare il grafico di una funzione univariata su un certo intervallo. I punti iniziali e finali dell'intervallo sono specificati dagli argomenti `from` e `to`. Ad esempio se si vuole disegnare la funzione seno di x tra $[0, 6\pi]$

```
curve(sin, from = 0, to = 6 * pi)
```



Per disegnare più grafici insieme nella stessa finestra grafica si utilizza la seguente opzione che consente di scegliere il numero di grafici da inserire

```
 par(mfrow = c(1, 2))
x <- c(12, 15, 20, 14, 16, 10, 10, 8, 15)
hist(x,
     main = "",
     ylab = "Frequenze assolute")
x <- seq(1, 10)
y = x^2 - 10*x
plot(x, y)
```



Pulizia ambiente di lavoro

Il seguente permette di creare un'ambiente vuoto

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

Le librerie

Le liberie di R consistono in una serie di funzioni già implementate in R. Consentono di eseguire le operazioni comuni nell'ambito dell'applicazione per la quale sono state create. Dal prompt di R si può richiedere l'installazione di librerie aggiuntive. Accendendo ad internet si verifica il contenuto del CRAN e si aggiornano i packages disponibili. Una volta installata è sufficiente dal prompt utilizzare il comando seguente per renderla disponibile in R

```
library(boot)
library(LMest)
```

Per ottenere istruzioni sul contenuto della libreria si usa

```
library(help = boot)
library(help = LMest)
```

La funzione apropos permette di cercare informazini su una oggetto di R o su una funzione utilizzando alcune lettere della parola.

```
data(Orange)
apropos("Oran")
#> [1] "Orange" "Orange"
head(Orange)
```

Tree	age	circumference
1	118	30
1	484	58
1	664	87
1	1004	115
1	1231	120
1	1372	142

Per citare R si può scrivere

```
citation()
#>
#> To cite R in publications use:
#>
#>   R Core Team (2022). R: A language and environment for statistical
#>   computing. R Foundation for Statistical Computing, Vienna, Austria.
#>   URL https://www.R-project.org/.
#>
#> Una voce BibTeX per gli utenti LaTeX è
#>
#>   @Manual{,
#>     title = {R: A Language and Environment for Statistical Computing},
#>     author = {{R Core Team}},
#>     organization = {R Foundation for Statistical Computing},
#>     address = {Vienna, Austria},
#>     year = {2022},
#>     url = {https://www.R-project.org/},
#>   }
#>
#> We have invested a lot of time and effort in creating R, please cite it
#> when using it for data analysis. See also 'citation("pkgname")' for
#> citing R packages.
```

Le librerie possono essere visualizzate da internet al seguente sito http://rm.mirror.garr.it/mirrors/CRAN/ seguendo Packages, Table of available packages.

Per ulteriori approfondimenti sulla sintassi di R si veda *An Introduction to R* https://cran.r-project.org/doc/manuals/r-release/R-intro.pdf.

Introduction to R Markdown

Workflow R Markdown è un formato sviluppato dal team responsabile dell'IDE RStudio per poter generare documenti riproducibili (ad esempio report) direttamente nell'ambiente R.

A packege in R is just a collection of functions, data sets and help pages. The following libraries are required

```
install.packages(c("knitr", "rmarkdown"))
library("knitr")
library("rmarkdown")
```

Com la libreria knitr si creano documenti **riproducibili**.

La citazione di una libreria si ottiene con la funzione citation seguita dal nome della libreria tra virgolette. Ad esempio

```
citation("knitr")
#>
#> To cite the 'knitr' package in publications use:
#>
#>   Yihui Xie (2022). knitr: A General-Purpose Package for Dynamic Report
#>   Generation in R. R package version 1.39.
#>
#>   Yihui Xie (2015) Dynamic Documents with R and knitr. 2nd edition.
#>   Chapman and Hall/CRC. ISBN 978-1498716963
#>
#>   Yihui Xie (2014) knitr: A Comprehensive Tool for Reproducible
#>   Research in R. In Victoria Stodden, Friedrich Leisch and Roger D.
#>   Peng, editors, Implementing Reproducible Computational Research.
#>   Chapman and Hall/CRC. ISBN 978-1466561595
#>
#> To see these entries in BibTeX format, use 'print(<citation>,
#> bibtex=TRUE)', 'toBibtex(.)', or set
#> 'options(citation.bibtex.max=999)'.
```

Buone pratiche

Si elencano le seguenti raccomandazioni:

- I files sorgenti devono essere allocati nella stessa directory

- Occorre scelgliere ed utilizzare una **working directory** predefinita dove si trova il **documento sorgente** all'inizio del lavoro e mantenere questa per tutta la durata del lavoro.
- I file dei dati da impiegare devono trovarsi nella stessa cartella del file sorgente e la working directory deve essere posizionata in questa cartella.
- Iniziare sempre riavviando una nuova sessione di `Rstudio` e compilare il documento sorgente dopo aver aperto la nuova sessione di `R`
- Anche se il codice o il documento non è perfetto meglio condividerlo con il docente che non condividerlo affatto!

Utilizzo

Occorre verificare che il file sorgente abbia estenzione `.Rmd`

- Dalla finestra del menù selezionare File->New File->R Markdown
- Scegliere il formato pdf, word o html e ok

File

Il file creato è un file di solo testo (perfettamente leggibile da altri programmi) dove alcuni elementi basilari di formattazione del testo sono indicati da caratteri speciali. Ad esempio scrivendo:

```
# Titolo

## Sottotitolo

### Sotto-sottotitolo

**testo in grassetto**

*testo in corsivo*

- Elenco puntato
- Elenco puntato
  + con altri
  + elementi annidati
```

Codice

Il codice `R` e solo questo codice deve essere incluso in **blocchi speciali** chiamati 'chunks'.

Mentre il **testo** che comprende i **commenti** al codice e la descrizione dei dati e dei risultati deve essere esclusivamente inserito fuori dal chunk.

Il chunk

Il chunk è delemitato da `{ }`.

La parte tra parentesi graffe, `{r echo = TRUE}`

indica il codice in linguaggio R che si intende mostrare (`echo`).

Per mostrare il codice senza eseguirlo si utlizza `eval = FALSE`.

Le possibili opzioni da inserire nel code chuncks sono le seguenti:

- **eval TRUE** se far eseguire (valutare il codice e includere i risultati)
- **echo TRUE** se mostrare il codice ed i suoi risultati
- **warning TRUE** se mostrare i warnings nel testo
- **error TRUE** se mostrare gli errori
- **include TRUE** se mostrare l'output del chunk, se FALSE tutto l'output non viene inserito nel documento compilato ma viene mostrato il codice

Per maggiori dettagli si rimanda ai files R *MarkdownCheatsheet.pdf* e *rmarkdown-reference.pdf.*

Echo e Eval

Una delle cose che rendono il Markdown particolarmente conveniente e' la capacita' di includere grafici generati da codice R. Questi vengono riportati a seguito del codice stesso.

Esempio

Nel seguente esempio il chunk contiene il codice con quale si estrae un numero casule uniforme ed in base al costrutto condizionale si chiede di stampare il risultato del lancio di una monetina

```
set.seed(234)
uniforme <- runif(1); uniforme
#> [1] 0.74562
if(uniforme > 0.5){
  print("Il risultato del lancio della moneta e' testa")
  } else {
  print("Il risultato del lancio della moneta e' croce")
}
#> [1] "Il risultato del lancio della moneta e' testa"
```

Per poter generare il report in diversi formati occorre cliccare sull'icona del gomitolo Knit e scegliere il formato Word (o html).

Automaticamente `R` esegue il codice nel chunk e inserisce nel report i risultati, salva il file nella directory di lavoro. Alcune volte il file di preview viene aperto automaticamente.

Estrazione del codice

E' possible ottenere un file con l'estensione `.R` con solo il codice di `R` presente nel file Markdown con la seguente funzione `purl`

```
knitr::purl("ASM_1_Mark.Rmd",
            "ASM_1_Mark.R",
            documentation = 1)
```

Ulteriore documentazione è reperibile presso https://rmarkdown.rstudio.com/ e nel `blog.rstudio.com`.

Si rimanda anche al seguente tutorial https://www.rstudio.com/resources/webinars/programming-part-3-package-writing-in-rstudio/.

Generatore linerare di numeri pseudo-causali

Nel seguito si mostra un esempio che implementa un algoritmo lineare di tipo congruenziale **misto** per la generazione di numeri pseudo-casuali. Si riporta anche la funzione di `R` nel seguito chiamata `runif`.

Metodo congruenziale lineare

Si utilizza il metodo di generazione congruenziale misto in cui sono presenti le seguenti quantità. Si genera una successione di valori in modo deterministico che appare causuale.

Il generatore ha la seguente struttura:

$$x_{i+1} = (ax_i + c) \quad mod \quad m.$$

Lo applichiamo specificando i seguenti valori

$$x_{i+1} = [(2^{16} + 1) * 47838 + 5] \quad mod \quad 34359738368$$

Ovvero:

- il moltiplicatore è $a = 2^{16} + 1 = 65537$
- il valore iniziale o seme è $x_0 = 47838$
- l'incremento è $c = 5$
- il modulo è $m = 34359738368$ è $2^{\beta} = 2^{35}$

Risultato

- **Primo numero** generato:

```
a <-  65537
m <-34359738368
xini <- 47838
c <- 5
(a*xini + c)%%m
#> [1] 3135159011
```

Da cui il primo numero **pseudo-casuale** si ottiene dividendo per il modulo

```
x1 <- (a*xini+c)%%m
x1/m
#> [1] 0.09124514
```

- **Secondo numero**: il valore successivo si ottiene da quello ottenuto in precedenza

```
(a*x1 + c)%%m
#> [1] 32040401640
x2 <- (a*x1+ c)%%m
x2/m
#> [1] 0.9324984
```

- Il seguente **costrutto iterativo** permette di generare una sequenza di 1857 numeri pseudo-casuali le cui determinazioni vengono salvate nel vettore `random.n`

```
n <- 1857
random.n <- numeric(n)

for(j in 1:n){
  x1  <- (a*x1+c)%%m
  random.n[j] <- x1/m
}

head(random.n)
#> [1] 0.9324984 0.1487612 0.3650336 0.2063155 0.2976069 0.2639079
tail(random.n)
#> [1] 0.6885469 0.2974622 0.7813871 0.7653215 0.8742654 0.7332189
```

Si rimanda all'help `Random` per informazioni sul generatore di `R`. Il il metodo di default è "Mersenne-Twister".

La funzione `set.seed` richiede un intero per specificare il seme in `R`.

Valutazione della pseudo-casualità della serie

Statistiche descrittive

```
require(skimr)
skim_without_charts(random.n)
```

Table 2: Data summary

Name	random.n
Number of rows	1857
Number of columns	1
_____	
Column type frequency:	
numeric	1
_____	
Group variables	None

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100
data	0	1	0.5	0.29	0	0.25	0.5	0.75	1

```
var(random.n)
#> [1] 0.08328912
```

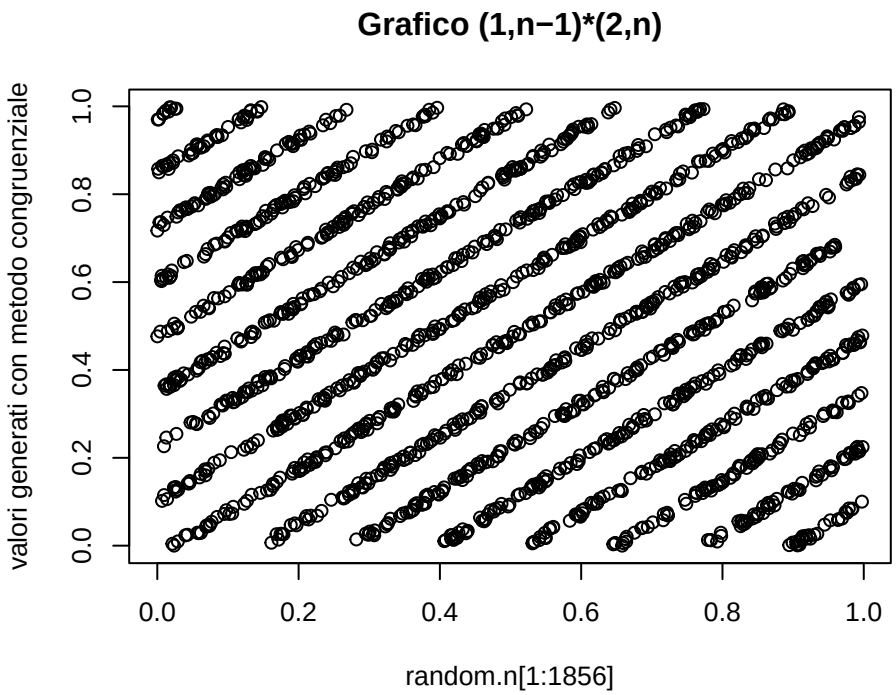
Gli indici di posizione sono in linea con quelli attesi rispetto alla distribuzione uniforme la cui variabile casuale ha un valore atteso è pari a $1/2$. La varianza è conforme con quella teorica $1/12 = 0.08333$.

Test grafici

a) Diagramma a dispersione

Il primo test grafico è il **diagramma a dispersione** sul piano $(1, n-1) \cdot (2 : n)$

```
plot(random.n[1:1856],
     random.n[2:1857],
     main="Grafico (1,n-1)*(2,n)",
     ylab = "valori generati con metodo congruenziale")
```



Si nota che i punti sono in sequenza e non sono sparsi nel piano in modo casuale come dovrebbero essere se fossero osservazioni indipendeti. Se ci fosse indipendenza ogni coppia di punti (u_i, u_{i-1}) dovrebbe essere disposta con uguale probabilità sul piano. Il grafico evidenzia che il generatore non è adeguato.

Si verifichi che basta semplicemente aumentare il valore assegnato al termine c (incremento), ad esempio ponendo c=235 perchè la serie non presenti più questo andamento nel grafico a dipesesione.

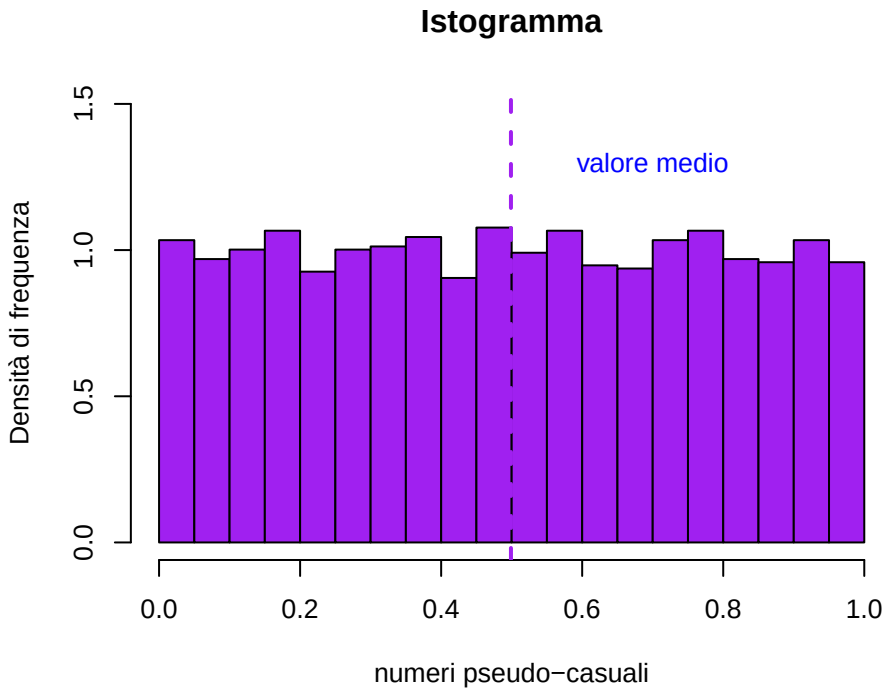
c) Istogramma

L'**istogramma** permette di valutare la densità di frequenza per ogni classe di valori.

Nel seguito si disegna l'istogramma fissando 20 classi e si aggiunge il valore medio osservato oltre al testo nel grafico.

```
hist(random.n, col = 'purple',
     breaks = 20,
```

```
     freq=FALSE,
     ylim =c(0,1.5),
     main='Istogramma',
     xlab='numeri pseudo-casuali',
     ylab='Densità di frequenza')
abline(v=mean(random.n),
       col='purple',
       lwd=2,lty=2)
text(0.7,1.3,
     c("valore medio"),
     col="blue")
```



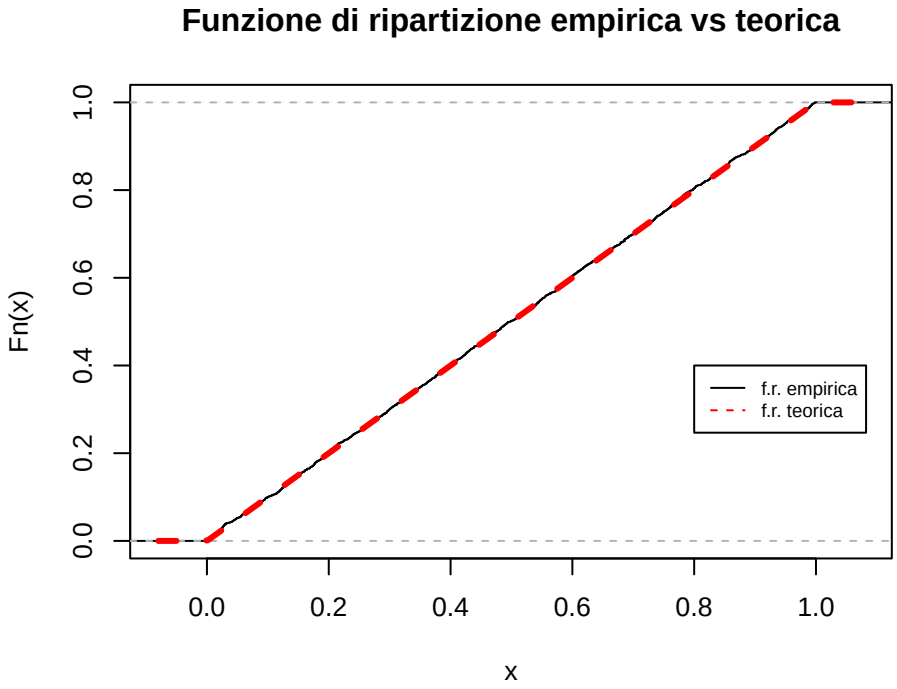
Si nota che le densità di frequenza sono approssimativamente simili ed intorno al valore 1. Il grafico è simile a quello atteso sotto l'ipotesi di uniforme distribuzione $[0, 1]$.

c) Funzione di ripartizione empirica

Il test grafico seguente permette di confrontare la **funzione di ripartizione empirica** e quella teorica ottenuta con la funzione nativa di `R`.

Nel seguente chunk prima si disegna la distribuzione empirica e poi si aggiunge quella teorica ed infine si aggiunge la legenda al grafico.

```
plot(ecdf(random.n),
     do.points=FALSE,
     main ='Funzione di ripartizione empirica vs teorica')
curve(punif(x,0,1),
      lty='dashed',
      col='red',
      lwd='3',
      add=TRUE)
legend(0.8,0.4,
       col=c("black","red"),
       c("f.r. empirica","f.r. teorica"),
       lty=c(1,2),
       cex=0.7)
```



Non si notano particolari differenze tra le due funzioni di ripartizione poste a confronto.

Test di Kolmogorov-Smirnov

Il test non parametrico che permette di confrontare la funzione di ripartizione empirica con quella teorica della distribuzione uniforme si effettua con la funzione `ks.test`.

Dalle dispense della teoria ricordiamo che la statistica test si basa sulla massima differenza in modulo tra le due funzioni di ripartizione.

Sotto l'ipotesi nulla che le due funzioni di ripartizione siano uguali si utilizza la distribuzione asintotica (n grande) della statistica test (Birnbaum & Tingey, 1951).

```
ks.test(random.n, "punif")
#>
#>  Asymptotic one-sample Kolmogorov-Smirnov test
#>
#> data:  random.n
#> D = 0.0095202, p-value = 0.996
#> alternative hypothesis: two-sided
```

Il valore osservato della statistica test è 0.009 e l'area definita è prossima a 1. Il livello di significatività osservato **p-value** è superiore a 0.1.,0.05, 0.01 pertanto il test induce ad accettare l'ipotesi nulla che i dati siano realizzazioni dalla variabile casuale uniforme in $[0, 1]$.

Test Chi Quadrato

Per eseguire questo test si calcola la frequenza relativa dei valori osservati per ogniuno dei sottointervalli

```
n <- length(random.n)
int<-seq(0,1,by=0.1); int
#>  [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
foss<-table(cut(random.n, int))/n; foss
#>
#>    (0,0.1]   (0.1,0.2]   (0.2,0.3]   (0.3,0.4]   (0.4,0.5]   (0.5,0.6]   (0.6,0.7]
#> 0.10016155 0.10339257 0.09639203 0.10285407 0.09908454 0.10285407 0.09423802
#>   (0.7,0.8]   (0.8,0.9]     (0.9,1]
#> 0.10500808 0.09639203 0.09962305
```

Le frequenze osservate vengono confrontate con quelle attese in base all'ipotesi che le determinazioni siano generate dalla distribuzione uniforme

```
p<- rep(0.1,10); p
#>  [1] 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1
```

La funzione `chisq.test` permette di effettuare il test nel modo seguente

```
chisq.test(foss,p=p)
#>
#>  Chi-squared test for given probabilities
#>
#> data:  foss
#> X-squared = 0.0011312, df = 9, p-value = 1
```

Il $p - value$ è prossimo a 1.

Il valore critico della statistica test se l'ipotesi nulla è vera si determina fissando un valore per l'errore di primo tipo ad esempio $\alpha = 0.05$ nel modo seguente

```
qchisq(0.95,df=9)
#> [1] 16.91898
```

Essendo il valore critico del test pari a 17 questo risulta molto maggiore del valore ossevato (0.0011) ed il test non permette di rifiutare l'ipotesi nulla per ogni livello di significatività.

Pertanto anche in base a questo test le determinazioni sono assimilabili a quelle di una distribuzione uniforme in $[0, 1]$.

Occorre eseguire il test per altre (molte) serie di numeri generate con il generatore congruenziale precedente (cambiando il seme) per valutare accuratemente la qualità del generatore.

Si noti che nelle situazioni reali in cui si considerano i dati campionari il test deve essere svolto utilizzando le frequenze assolute e non le frequenze relative perchè altrimenti le conclusioni possono essere errate dipendendo dalla numerosità campionaria.

Nel presente contesto occorre simulare un'ampia serie di numeri pseudo casuali per cui è comunque possibile utilizzare le frequenze relative.

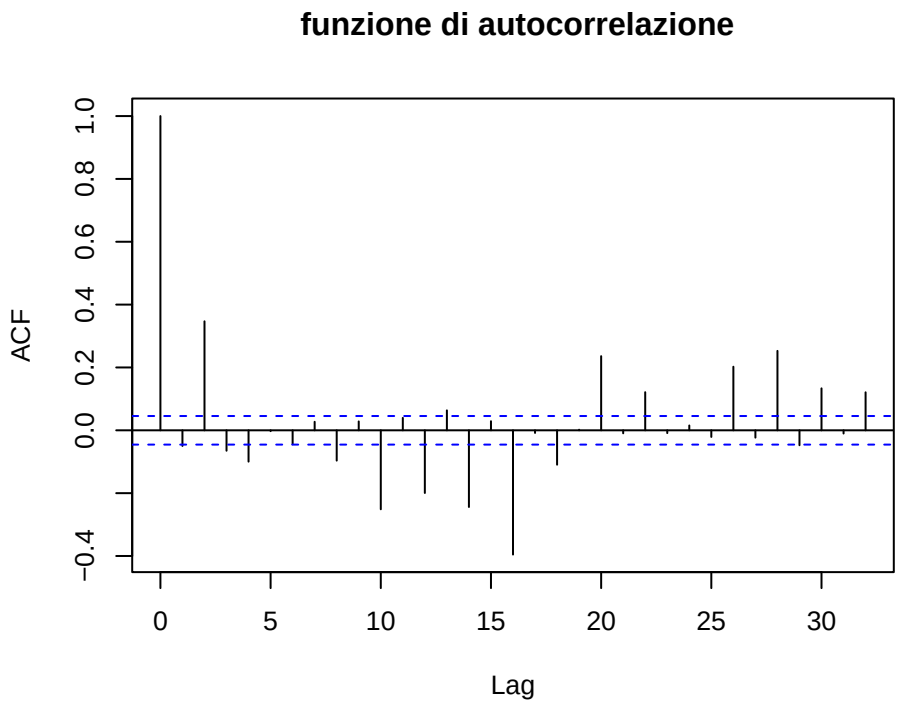
Con le frequenze assolute il test è il seguente

```
n <- length(random.n)
int<-seq(0,1,by=0.1); int
#>  [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
fossN<-table(cut(random.n, int)); fossN
#>
#>   (0,0.1] (0.1,0.2] (0.2,0.3] (0.3,0.4] (0.4,0.5] (0.5,0.6] (0.6,0.7] (0.7,0.8]
#>       186       192       179       191       184       191       175       195
#> (0.8,0.9]   (0.9,1]
#>       179       185
chisq.test(fossN)
#>
#>  Chi-squared test for given probabilities
#>
#> data:  fossN
#> X-squared = 2.1007, df = 9, p-value = 0.9898
```

Funzione di autocorrelazione empirica

La rappresentazione grafica della **funzione di autocorrelazione empirica** viene chiamata **correlogramma**. In `R` si disegna con la funzione `acf` dove il ritardo di default è pari a $10 \cdot \log(n)$

```
acf(random.n,
    main = " funzione di autocorrelazione")
```

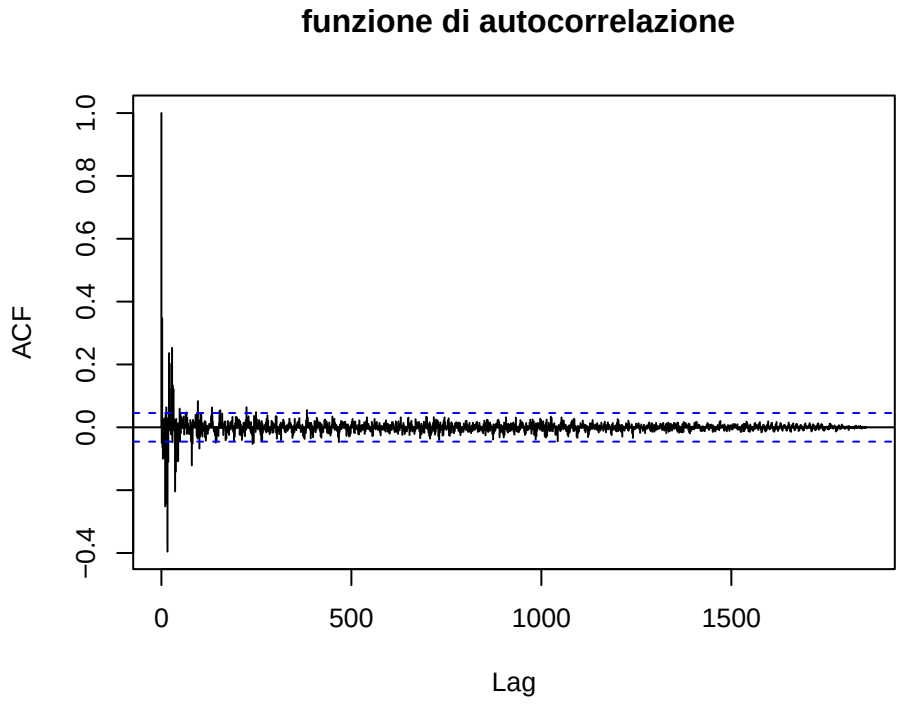


Il grafico si utilizza per le serie storiche in quanto i dati sono generati a istanti successivi.

Il grafico mostra il valore della correlazione determinato tra due valori della serie con $k = lag$: i valori sono correlati: serial dependence. Al crescere al crescere della lunghezza dell'intervallo ci si aspetta che la correlazione tra X_t e X_{t+h} diminuisca. Il modo in cui la funzione di autocorrelazione tende a zero viene considerato come una misura di memoria del processo

Nel primo grafico si notano dei valori che si discostano da quelli attesi.

```
n<-length(random.n)
acf(random.n, main = " funzione di autocorrelazione", lag.max =n)
```



Anche nel grafico con ritardo pari a 1 le osservazioni sono dipendenti. Infatti a livello antitotico vale la seguente distribuzione

$$\hat{\rho}(k) \sim N(0, 1/n).$$

Si notano alcuni valori superiori ai valori soglia (linee tratteggiate) in base all'ipotesi di processo stazionario sottostante.

Questo grafico suggerisce che le determinazioni non sono indipendenti come già notato nel grafico dei punti posti sul piano cartesiano.

Funzione runif

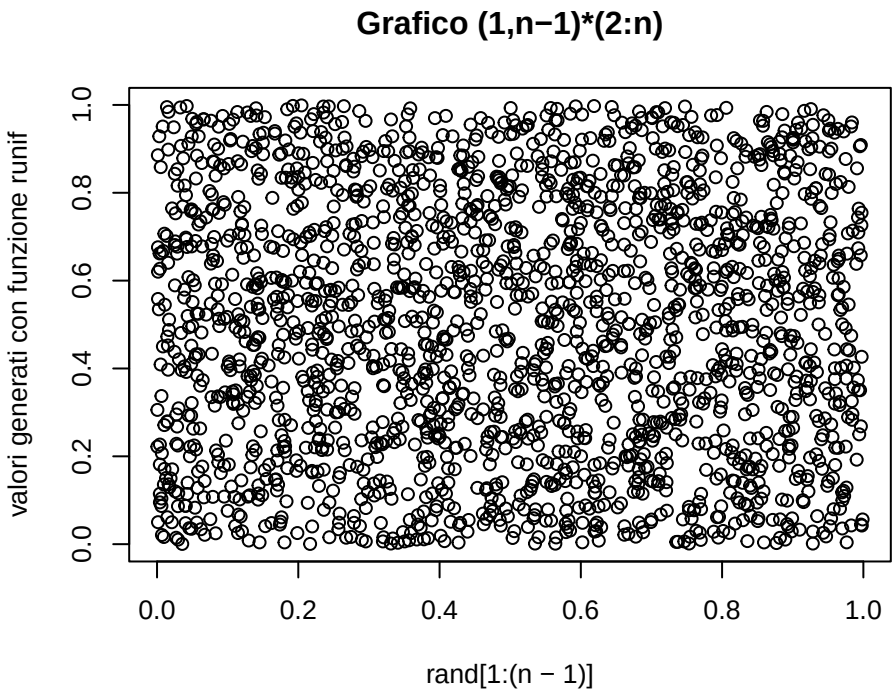
Nel seguito si utilizza il generatore di `R` che si richiama con la funzione `runif` e si ripetono i test effettuati in precedenza su questa nuova serie di realizzazioni di numeri pseudo-casuali.

Si noti che la funzione `set.seed` definisce il valore iniziale e permette di poter generare sempre gli stessi valori.

```
set.seed(3882)
n <- 2000
rand <- runif(n, min=0, max=1)
head(rand)
#> [1] 0.5768151 0.4648768 0.5794360 0.3892577 0.4496224 0.5984687
```

a) diagramma a dispersione

```
plot(rand[1:(n-1)],
     rand[2:n],
     main="Grafico (1,n-1)*(2:n)",
     ylab = "valori generati con funzione runif")
```

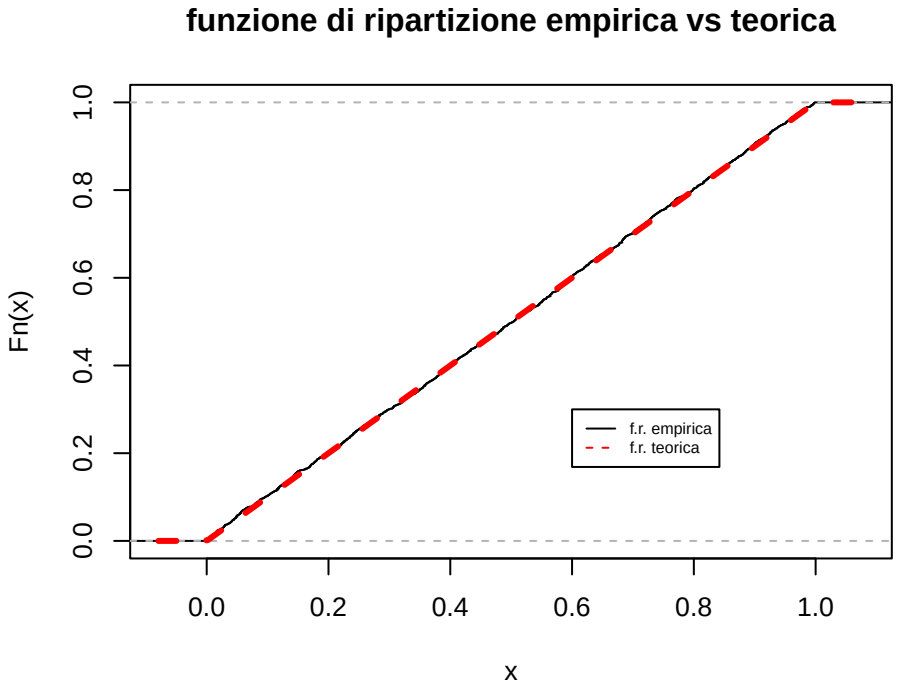


Si nota che i punti sono sparsi nel piano in modo casuale.

b) Funzione di ripartizione empirica

```
plot(ecdf(rand),
     do.points=FALSE,
     main ='funzione di ripartizione empirica vs teorica')
curve(punif(x,0,1),
      lty='dashed',
      col='red',
      lwd='3',
      add=TRUE)
legend(0.6,0.3,
```

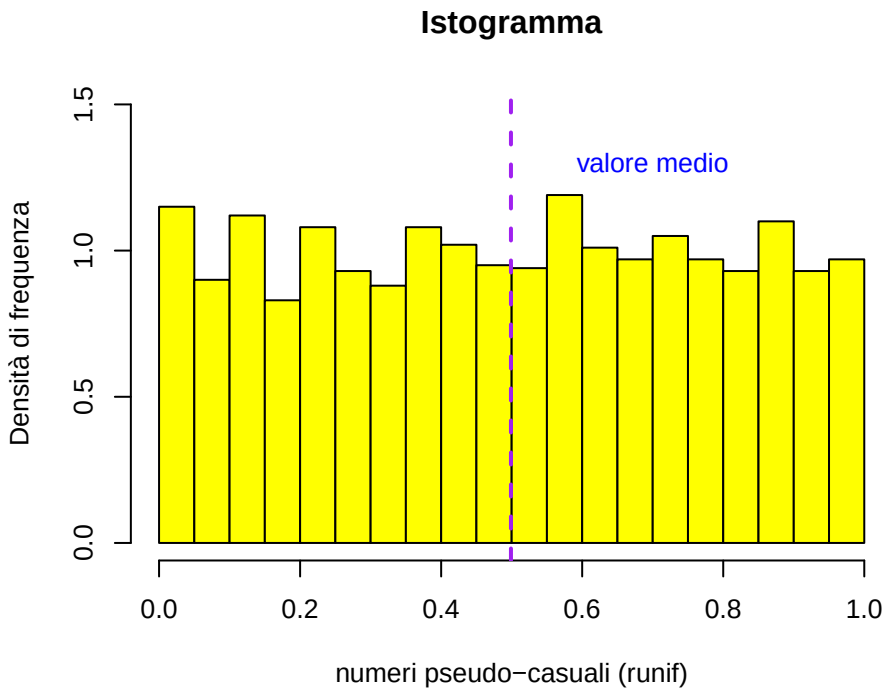
```
       col=c("black","red"),
       c("f.r. empirica","f.r. teorica"),
       lty=c(1,2),
       cex=0.6)
```



Non si notano particolari differenze tra le due funzioni di ripartizione.

c) Istogramma

```
hist(rand, col = 'yellow',
     breaks = 20,
     freq=FALSE,
     ylim =c(0,1.5),
     main='Istogramma',
     xlab='numeri pseudo-casuali (runif)',
     ylab='Densità di frequenza')
abline(v=mean(random.n),
       col='purple',
       lwd=2,lty=2)
text(0.7,1.3,
     c("valore medio"),
     col="blue")
```



La densità appare quasi uniforme.

Rispetto ai grafici riferiti alla serie di numeri ottenuti con il metodo congruenziale questi ultimi conducono ad accettare l'ipotesi che le determinazioni provengano da una distribuzione uniforme.

Anche i test statistici confermano quest'affermazione.

Test statistici

Kolmogorov-Smirnov

```
ks.test(rand, "punif")
#>
#>  Asymptotic one-sample Kolmogorov-Smirnov test
#>
#> data:  rand
#> D = 0.011039, p-value = 0.9678
#> alternative hypothesis: two-sided
```

Chi Quadrato

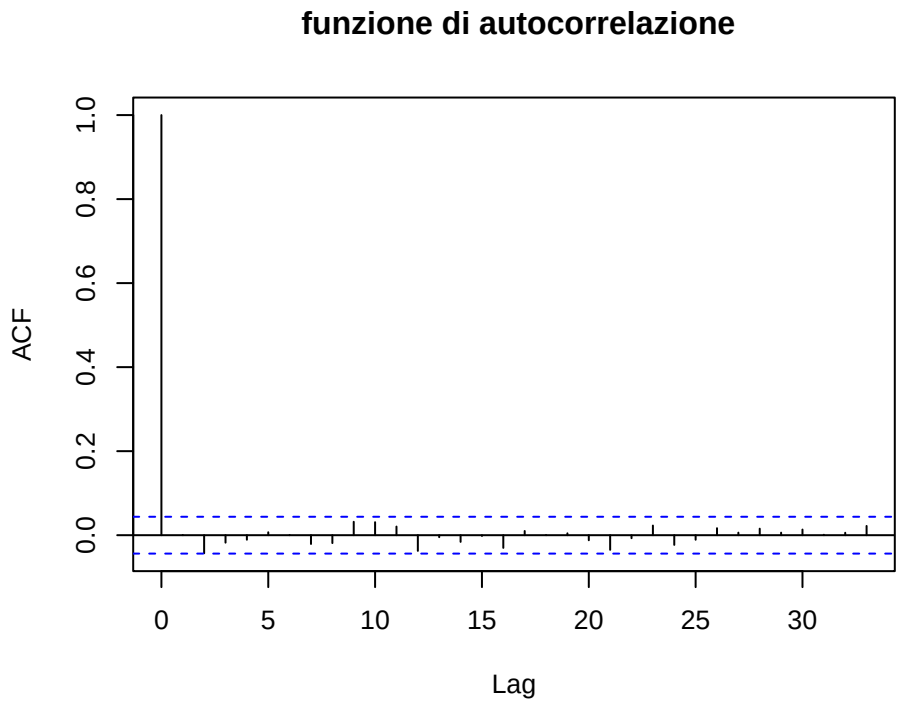
```
foss<-table(cut(rand, int))/n; foss
#>
```

```
#>   (0,0.1] (0.1,0.2] (0.2,0.3] (0.3,0.4] (0.4,0.5] (0.5,0.6] (0.6,0.7] (0.7,0.8]
#>    0.1025    0.0975    0.1005    0.0980    0.0985    0.1065    0.0990    0.1010
#> (0.8,0.9]   (0.9,1]
#>    0.1015    0.0950
```

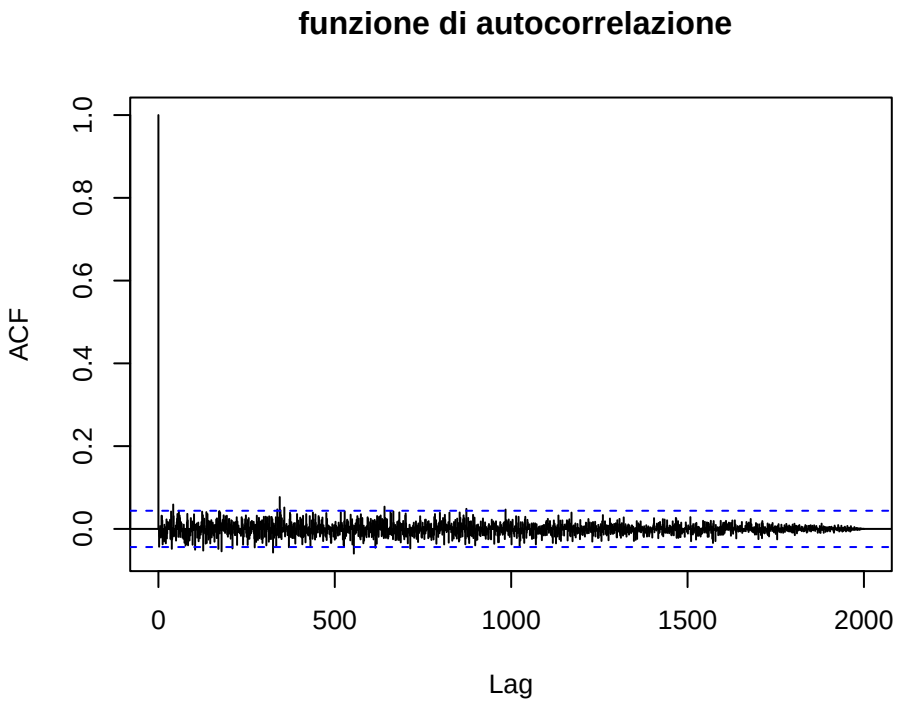
```
chisq.test(foss,p=p)
#>
#>  Chi-squared test for given probabilities
#>
#> data:  foss
#> X-squared = 0.000905, df = 9, p-value = 1
```

Autocorrelazione

```
acf(rand,
    main = " funzione di autocorrelazione")
```




```
acf(rand, lag.max = n,
    main = " funzione di autocorrelazione")
```


funzione di autocorrelazione
ACF
0.0
0.2
0.4
0.6
0.8
1.0
0
500
1000
1500
2000
Lag

Generazione di determinazioni da variabili casuali

Le funzioni implementate in `R` utilizzano dei suffissi per definire le seguenti quantità:

`d` per calcolare la densità in un punto;

`p` per calcolare la funzione di ripartizione in un punto

`q` per calcolare un quantile

`r` per generare pseudo-determinazioni dalla distribuzione

Le funzioni di :

- Exponential: nome `exp`;
- Gamma: nome `gamma`;
- Student t: nome `t`
- Normale: nome `norm`

Per la distribuzione Normale standard nella libreria `stats` si ha:

- `dnorm`
- `pnorm`
- `qnorm`
- `rnorm`

Generazione di pseudo-derminazioni dalla v.c. di Gauss

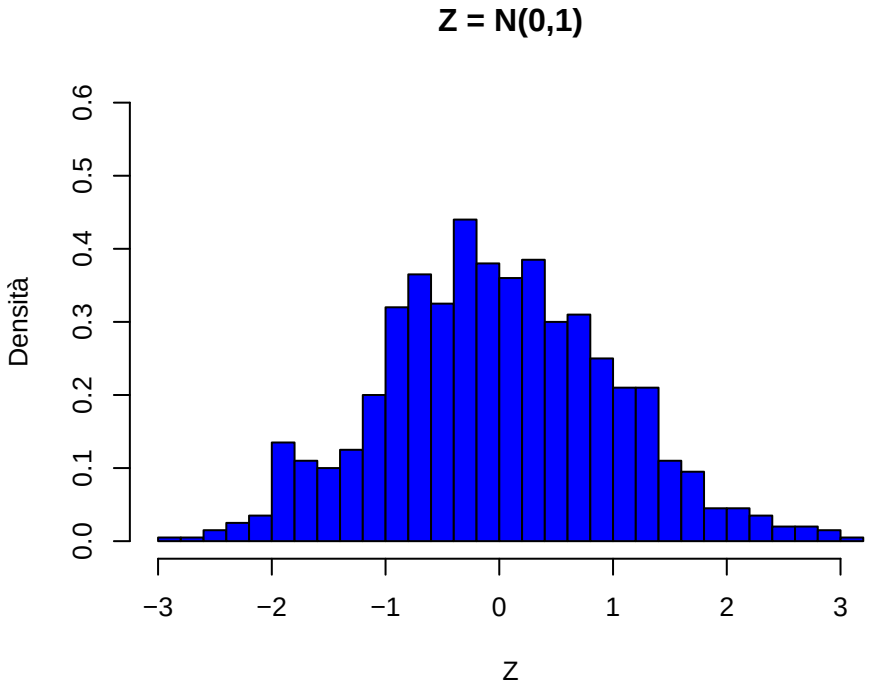
Si generano 10 realizzazioni dalla variabile casuale di Gauss con media zero e varianza 1

```
rnorm(10,0,1)
#>  [1]  0.3311984  0.6804855 -1.6002592 -3.2580751  0.2087347 -0.1635518
#>  [7]  0.8724223 -0.4829397 -0.6494778 -0.4161440
```

Si generano 1000 realizzazioni e si disegna distribuzione empirica

```
n <- 1000
set.seed(27732)
Z <- rnorm(1000)
mean(Z); sd(Z)
#> [1] -0.01054604
#> [1] 1.007595
#
```

```
hist(Z,
     breaks = 30,
     freq=FALSE,
     main="Z = N(0,1)",
     col ="blue",
     ylab="Densità",
     ylim =c(0,0.6))
```



Per generare 10 realizzazioni da una distribuzione di Gauss tale che $X \sim N(4, 16)$ si può applicare la trasformazione $(Z \cdot \sigma + \mu)$ dove Z rappresentano i generati dalla Normale standard oppure utilizzare la funzione specificando media e **deviazione standard**

```
rnorm(10,mean = 4, sd = 4 )
#>  [1] -3.321001  1.193820 12.282747  5.945237 -3.245020 -6.574799  1.855293
#>  [8]  9.809065  1.054175 -3.303407
```

Generazione dalla variabile casuale Esponenziale

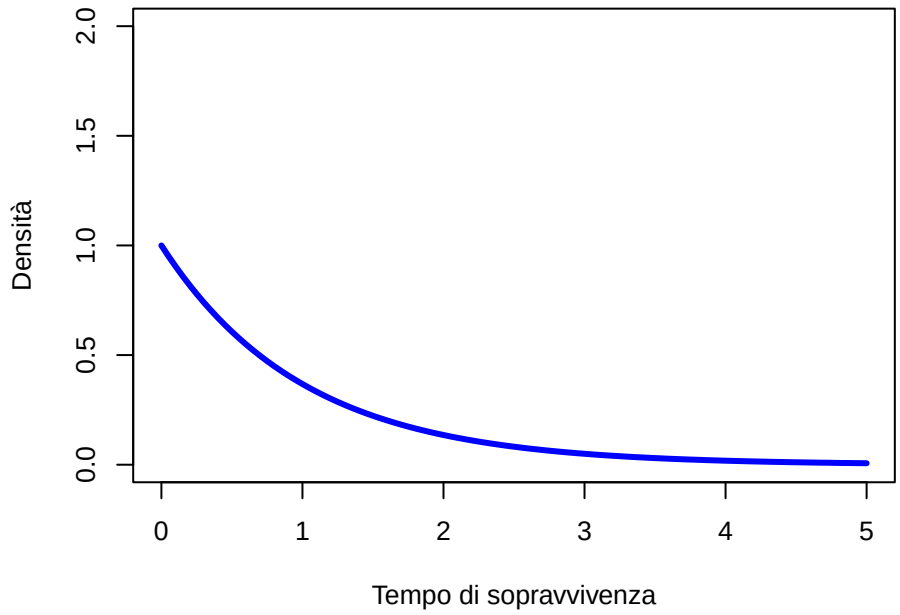
Nel seguente chuck si crea una griglia di 101 valori e si calcola la densità della v.c. esponenziale con $\lambda = 1$ nei punti generati

```r
x<-seq(0,5,length=101)
head(x)
#> [1] 0.00 0.05 0.10 0.15 0.20 0.25
h<-dexp(x,rate = 1)
head(h)
#> [1] 1.0000000 0.9512294 0.9048374 0.8607080 0.8187308 0.7788008
```

La media della v.c. è 1 e la varianza è 1.

Nel seguente grafico si rappresenta la funzione di densità

```r
plot(x,h,
     type="l", col = "blue",
     lwd = 3, ylim = c(0,2),
     xlab = "Tempo di sopravvivenza", ylab = "Densità")
```



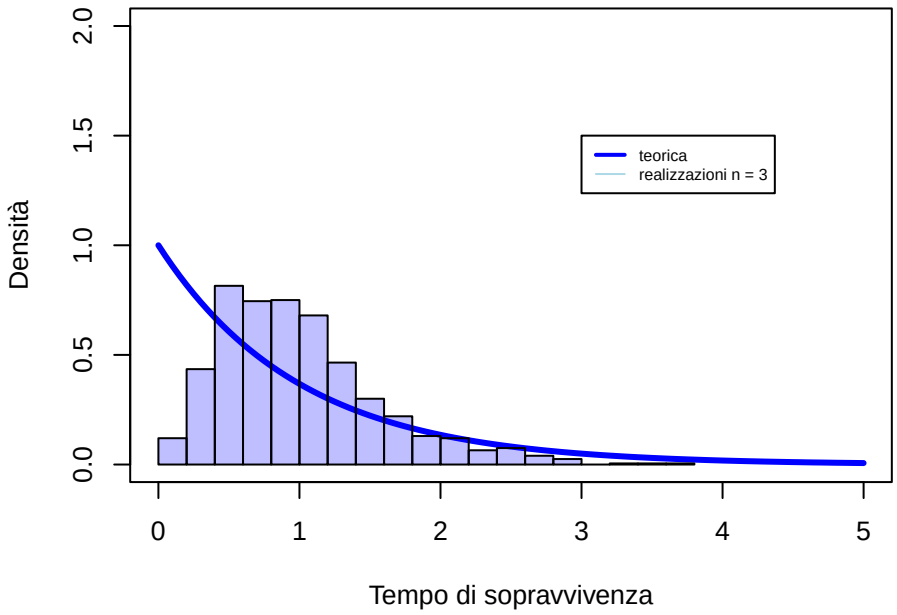
Si generano 1000 volte delle realizzazioni da v.c. esponenziali indipendenti con $\lambda = 1$.

Nel seguente si fissa una numerosità campionaria, si generano 3 realizzazioni dalla v.c. esponenziale di riferimento, si calcola la media aritmetica delle tre realizzazioni e si salvano nell'oggetto `m`

```r
n<-3
m<-rep(0,1000)
for(i in 1:1000){
  m[i]<-mean(rexp(n, rate=1))
}
head(m)
#> [1] 0.4708260 0.3383811 0.3472054 0.7999226 0.5198195 0.4173393
```

Si sovrappone l'istogramma dei valori generati alla curva disegnata in precedenza

```
plot(x,h,
     type="l", col = "blue",
     lwd = 3, ylim = c(0,2),
     xlab = "Tempo di sopravvivenza", ylab = "Densità")
hist(m, prob= T, add=T, col = rgb(0,0,1,1/4), breaks = 25)
legend(3,1.5, c("teorica", "realizzazioni n = 3"),
       col = c("blue", "lightblue"),
       lty  = c(1,1),
       lwd = c(2,1),
       cex = 0.6)
```



Per il **teorema del limite centrale** sappiamo che c'è la covergenza alla distribuzione Normale.

Generazione di pseudo-derminazioni dalla v.c. Beta

La distribuzione Beta(α, β) è definita per valori di $0 < x < 1$ e per $\alpha, \beta > 0$.

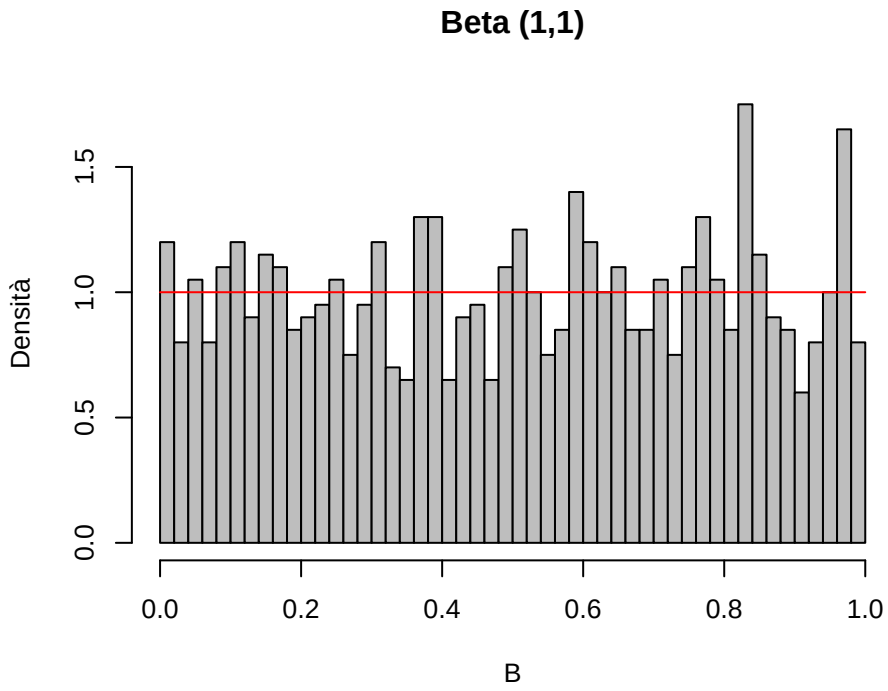
Si generano 1000 osservazioni dalla distribuzione beta standard con $\alpha = 1$ e $\beta = 1$. Si tratta della distribuzione uniforme in $(0, 1)$

```
n<-1000
set.seed(27732)
alpha <- 1
beta<- 1
```

```
B <- rbeta(n,alpha,beta)
summary(B)
#>      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
#> 0.0006886 0.2503327 0.5137247 0.5056517 0.7635183 0.9983702
```

Si disegna l'istogramma

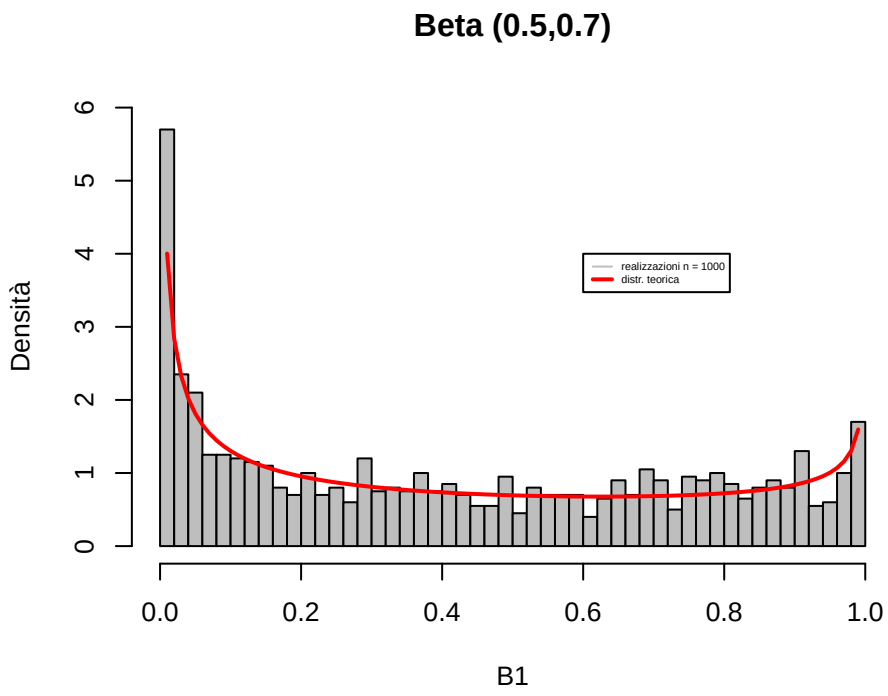
```
hist(B,
     breaks = 50,
     freq=FALSE,
     main="Beta (1,1)",
     col ="grey",
     ylab="Densità"
     )

curve(dbeta(x,1,1),
     col = "red",
    add = TRUE )
```



Si nota dalle statistiche descrittive che i numeri possono essere assunte come determinazioni pseudo-casuali.

- Si ottengono ancora 1000 realizzazioni ponendo $\alpha = 0.5$ e $\beta = 0.7$ e si sovrappone la **curva teorica** generata attraverso la funzione `dbeta`.

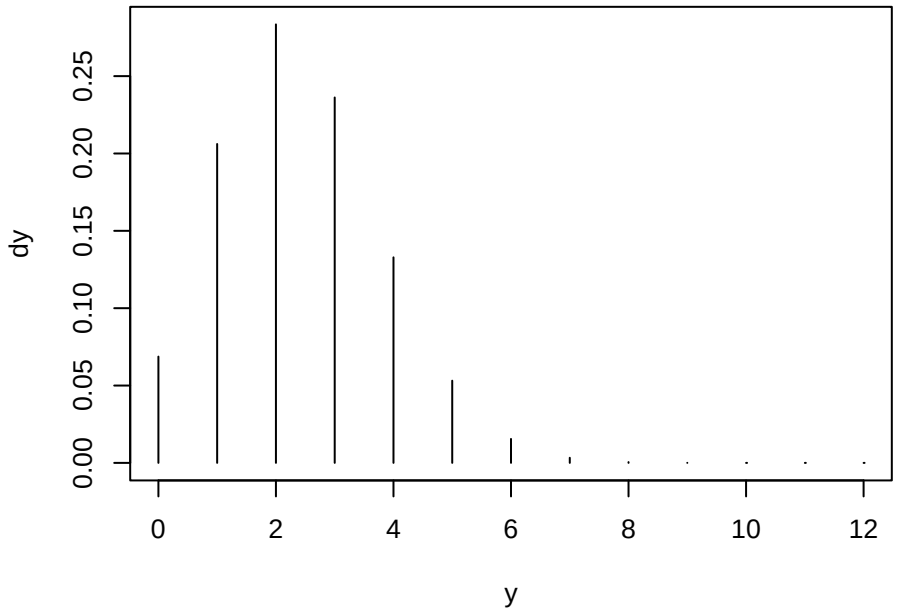
```r
alpha <- 0.5
beta <- 0.7
B1 <- rbeta(n,alpha,beta)
#
hist(B1,
     breaks = 50,
     freq=FALSE,
     main="Beta (0.5,0.7)",
     col ="grey",
     ylab="Densità",
     ylim=c(0,6)
     )
#
curve(dbeta(x,alpha,beta),
     col = "red",
    add = TRUE, lwd=2 )
legend(0.6,4, c("realizzazioni n = 1000",
              "distr. teorica"),
       col = c("grey", "red"),
       lty  = c(1,1),
       lwd = c(1,2),
       cex = 0.4)
```



Generazione di pseudo-derminazioni dalla v.c. Binomiale

Si genera una sequenza di valori interi da 0 a 12 ($n = 2$), si fissa la probabilità di successo pari a 0.3 e si determinano le probabilità corrispondenti ai valori di x, si rappresentano graficamente

```
y= seq(0, 12, 1); y
#>  [1]  0  1  2  3  4  5  6  7  8  9 10 11 12
dy <- dbinom(y,12,0.2); dy
#>  [1] 6.871948e-02 2.061584e-01 2.834678e-01 2.362232e-01 1.328756e-01
#>  [6] 5.315022e-02 1.550215e-02 3.321889e-03 5.190451e-04 5.767168e-05
#> [11] 4.325376e-06 1.966080e-07 4.096000e-09
plot(y, dy, type="h")
```



Per generare dei numeri dalla distribuzione Binomiale si utilizza `rbinom`. Con il seguente codice si genera una realizzazione del risultato di un esperimento un cui la probabilità dell'evento successo è 0.2 e si ripete l'esperimento per 8 volte

```
set.seed(123)
 rbinom(1, 8, 0.20)
#> [1] 1
```

ovvero su 8 prove si riscontra solo un successo.

Il seguente codice permette di simulare 8 realizzazioni di un esperimento in cui la probabilità di successo è sempre 0.2

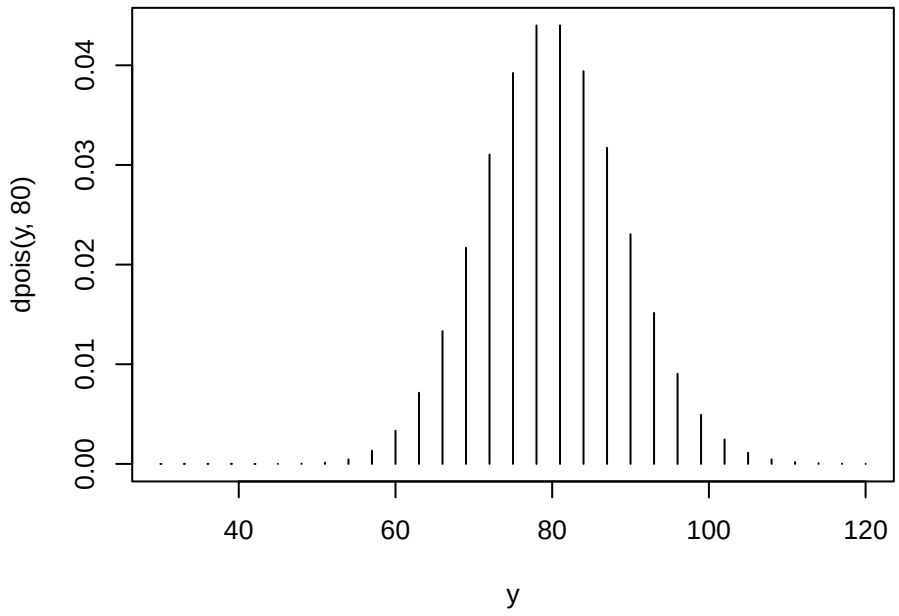
```
rbinom(8, 1, 0.20)
#> [1] 0 0 1 1 0 0 1 0
```

il risultato è che si verifica un succeso alla terza, quarta e settima prova.

Generazione di pseudo-derminazioni dalla v.c. di Poisson

Si genera una sequenza di valori tra 30 e 120 con incremento di 1.

```
y = seq(30, 120, 3)
plot(y, dpois(y, 80), type='h')
```



Si rappresenta la densità di Poisson fissando la media a 80

Per generare 100 realizzazioni dalla distribuzione di Poisson con parametro $\lambda = 22$ si utilizza:

```
set.seed(163)
y <- rpois(100, 22); y
#>   [1] 19 21 14 20 19 26 13 23 19 25 19 25 30 17 22 22 30 25 22 17 17 25 17 29 16
#>  [26] 24 18 19 26 19 25 27 18 25 22 25 22 26 21 21 21 16 17 28 13 22 18 25 24 23
#>  [51] 31 24 23 22 25 17 20 17 19 15 16 27 25 27 23 15 17 23 27 24 13 15 20 20 17
#>  [76] 20 29 21 25 23 24 22 22 11 25 19 22 18 19 25 26 27 27 25 22 20 14 22 20 19
mean(y)
#> [1] 21.46
var(y)
#> [1] 18.57414
```

Modello lineare generalizzato per i conteggi basato sulla distribuzione Binomiale Negativa

Si considerino i seguenti dati riguardanti dei grachi particolari (per sapere cosa sono si rimanda al seguente video https://www.youtube.com/watch?v=6gydJh6rP50). Si tratta di 173 femmine di granchi. Durante la stagione della deposizione delle uova, una femmina migra verso la riva per riprodursi. Con un maschio attaccato alla sua spina dorsale posteriore, si scava nella sabbia e depone grappoli di uova. Le uova sono fecondate esternamente. Durante la deposizione delle uova, altri granchi maschi, chiamati satelliti, possono raggrupparsi intorno alla coppia e fecondare anche le uova. Per ogni granchio femmina si intende modellizzare il numero di satelliti (`sat`). Le variabili esplicative sono il colore del granchio femmina (`color` 1 = medio chiaro; 2 = medio; 3 = medio scuro; 4 = scuro), la condizione della spina dorsale (`spine` 1 = entrambe buone; 2 = una consumata o rotta; 3 = entrambe consumate o rotte), la larghezza del carapace (`width` in cm) e il peso (`weight`in kg). Il colore è un surrogato dell'età del granchio, poiché i granchi più vecchi tendono ad avere un colore più scuro.

Si caricano i dati dal seguente sito

```
Crabs <- read.table("http://stat4ds.rwth-aachen.de/data/Crabs.dat", header=TRUE)
head(Crabs)
```

crab	sat	y	weight	width	color	spine
1	8	1	3.05	28.3	2	3
2	0	0	1.55	22.5	3	3
3	9	1	2.30	26.0	1	1
4	0	0	2.10	24.8	3	3
5	4	1	2.60	26.0	3	3
6	0	0	2.10	23.8	2	3

```
require(skimr)
skimr::skim_without_charts(Crabs)
```

Table 5: Data summary

Name	Crabs
Number of rows	173
Number of columns	7
Column type frequency:	
numeric	7
Group variables	None

Variable type: numeric

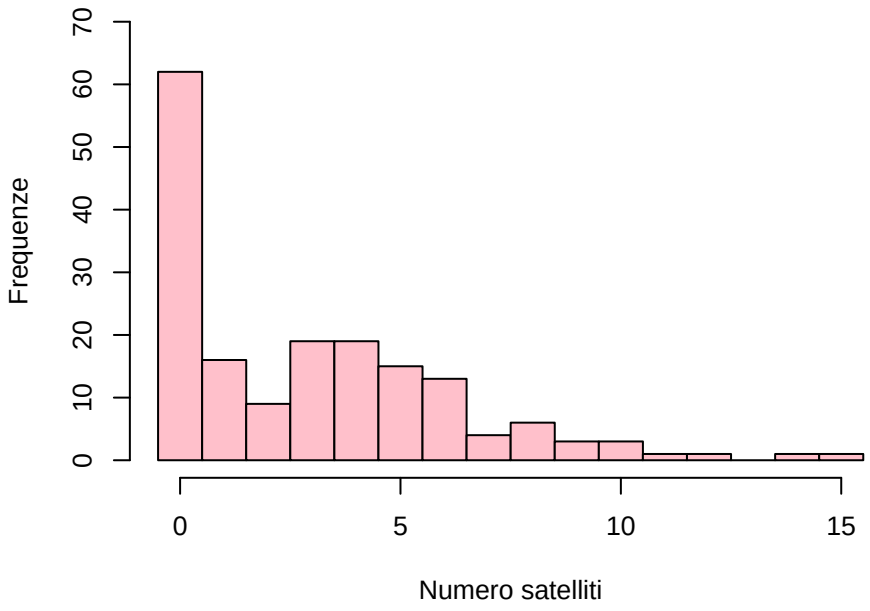
skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100
crab	0	1	87.00	50.08	1.0	44.0	87.00	130.00	173.0
sat	0	1	2.92	3.15	0.0	0.0	2.00	5.00	15.0
y	0	1	0.64	0.48	0.0	0.0	1.00	1.00	1.0
weight	0	1	2.44	0.58	1.2	2.0	2.35	2.85	5.2
width	0	1	26.30	2.11	21.0	24.9	26.10	27.70	33.5
color	0	1	2.44	0.80	1.0	2.0	2.00	3.00	4.0
spine	0	1	2.49	0.83	1.0	2.0	3.00	3.00	3.0

```
table(Crabs$sat)
#>
#>  0  1  2  3  4  5  6  7  8  9 10 11 12 14 15
#> 62 16  9 19 19 15 13  4  6  3  3  1  1  1  1
```

Si nota che ben 62 femmine non hanno satelliti e che la media è 2.92 mentre la varianza è 9.9 per cui molto superiore alla media. Poche (4) hanno un numero di satelliti superiore a 10.

La distribuzione empirica è la seguente

```
hist(Crabs$sat, breaks=c(0:16)-0.5,
     ylim = c(0,70),
     col = "pink", ylab = "Frequenze",
     xlab = "Numero satelliti",
     main = " ")
```



Utilizzando la distribuzione BN si stima il seguente modello lineare generalizzato considerando come variabili esplicative il peso e il colore.

```
library(MASS)
stima <- glm.nb(sat ~ weight + factor(color), link=log, data=Crabs)
summary(stima)
#>
#> Call:
#> glm.nb(formula = sat ~ weight + factor(color), data = Crabs,
#>     link = log, init.theta = 0.9596400657)
#>
#> Deviance Residuals:
#>     Min       1Q   Median       3Q      Max
#> -1.8680  -1.3680  -0.3149   0.4263   2.2791
#>
#> Coefficients:
#>                Estimate Std. Error z value Pr(>|z|)
#> (Intercept)     -0.4263     0.5382  -0.792    0.428
#> weight           0.7121     0.1615   4.410 1.04e-05 ***
#> factor(color)2  -0.2527     0.3486  -0.725    0.468
#> factor(color)3  -0.5218     0.3799  -1.373    0.170
#> factor(color)4  -0.4804     0.4282  -1.122    0.262
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> (Dispersion parameter for Negative Binomial(0.9596) family taken to be 1)
#>
#>     Null deviance: 220.01  on 172  degrees of freedom
#> Residual deviance: 196.56  on 168  degrees of freedom
#> AIC: 757.94
#>
#> Number of Fisher Scoring iterations: 1
#>
#>
#>               Theta:  0.960
#>           Std. Err.:  0.175
#>
#>  2 x log-likelihood:  -745.935
```

e si nota che il colore non risulta significativo ma solo peso è rilevante. La stima parametro di dispersione (forma della distribuzione) è 0.9596 e pertanto l'overdispersion è

```
1/0.9596
#> [1] 1.042101
```

Analisi della serie storica dei conteggi riferiti a COVID-19

Modello autoregressivo di Poisson non omogeneo

Nel presente esempio (tratto da Bartolucci e Pennoni, 2020) si stima un modello di Poisson non omogeneo nel tempo considerando la serie dei conteggi dei casi di pazienti affetti da COVID-19.

I dati sono quelli forniti giornalmente a livello ufficiale dal **Dipartimento della Protezione Civile** sull'andamento del COVID-19 in Italia.

La serie inizia con i conteggi del giorno 24 Febbraio 2020 (primo giorno di rilevazione) e viene incrementata ogni giorno.

Nel seguente esempio si considerano i dati riferiti al periodo temporale più recente dal 1 Agosto fino al 20 Ottobre 2022.

In particolare si seleziona la serie storica dei conteggi di interesse e si stima il modello di Poisson non omogeneo con un *trend temporale lineare*, *quadratico* e *log lineare* inserendo anche una *componente autoregressiva* del primo ordine (si vedano le dispense di teoria) espresso come segue:

$$\log \lambda_t(y_{(t-1)}) = \beta_0 + t\beta_1 + (t^2/100)\beta_2 + \log(t)\beta_3 + z_{t-1}\rho,$$

dove ρ è il coefficiente autoregressivo.

I dati sono di libero accesso nella piattaforma GITHUB e con la seguente sintassi si importano nel workspace direttamente dal repository e si selezionano alcune delle categorie disponibili.

Si noti che con il seguente chunck prima si importa il dataframe con i conteggi di tutte le date disponibili fino al giorno corrente e poi con la funzione `subset` si seleziona il perido di interesse

```
repository <- "https://raw.githubusercontent.com/pcm-dpc/COVID-19/master/"
overall.dataset <- "dati-andamento-nazionale/dpc-covid19-ita-andamento-nazionale.csv"
overall.filename<-paste(repository,overall.dataset,sep="")
Italy<-read.csv(overall.filename)
names(Italy)
#>  [1] "data"
#>  [2] "stato"
#>  [3] "ricoverati_con_sintomi"
#>  [4] "terapia_intensiva"
#>  [5] "totale_ospedalizzati"
#>  [6] "isolamento_domiciliare"
#>  [7] "totale_positivi"
#>  [8] "variazione_totale_positivi"
```

```
#>  [9] "nuovi_positivi"
#> [10] "dimessi_guariti"
#> [11] "deceduti"
#> [12] "casi_da_sospetto_diagnostico"
#> [13] "casi_da_screening"
#> [14] "totale_casi"
#> [15] "tamponi"
#> [16] "casi_testati"
#> [17] "note"
#> [18] "ingressi_terapia_intensiva"
#> [19] "note_test"
#> [20] "note_casi"
#> [21] "totale_positivi_test_molecolare"
#> [22] "totale_positivi_test_antigenico_rapido"
#> [23] "tamponi_test_molecolare"
#> [24] "tamponi_test_antigenico_rapido"
library(dplyr)
dataItaly<- Italy %>%
 select(data,
   dimessi_guariti,
        isolamento_domiciliare,
        ricoverati_con_sintomi,
        terapia_intensiva,deceduti)
mydate <- as.Date(as.POSIXct(dataItaly$data, format="%Y-%m-%dT %H:%M:%S"))
df<-data.frame(mydate,dataItaly$isolamento_domiciliare)
colnames(df)<-c("date", "Isolati")
df1 <- subset(df, date >= as.Date('2022-08-01') & date <= as.Date('2022-10-20'))
```

Si nota che vi sono 24 campi disponibili per i dati nazionali.

Il dataframe denominato `df` è stato creato considerando solo la data e la serie dei conteggi che si intende usare: ovvero ad esempio i pazienti posti in isolamento domiciliare.

Si vede che i giorni di rilevazione vanno dal 24 febbraio al 20 ottobre 2020

```
min(df$date)
#> [1] "2020-02-24"
max(df$date)
#> [1] "2022-10-29"
```

Il data frame df1 invece considera l'ultimo periodo.

L'utilizzo del modelli di Poisson permette anche di tenere conto in modo adeguato delle irregolarità presenti nei dati, e pertanto i valori interpolati creano una sorta di effetto smoothing o lisciamento ai dati osservati.

```
require(skimr)
skimr::skim_without_charts(df1$Isolati)
```

Table 7: Data summary

Name	df1$Isolati
Number of rows	81
Number of columns	1
_____	
Column type frequency:	
numeric	1
_____	
Group variables	None

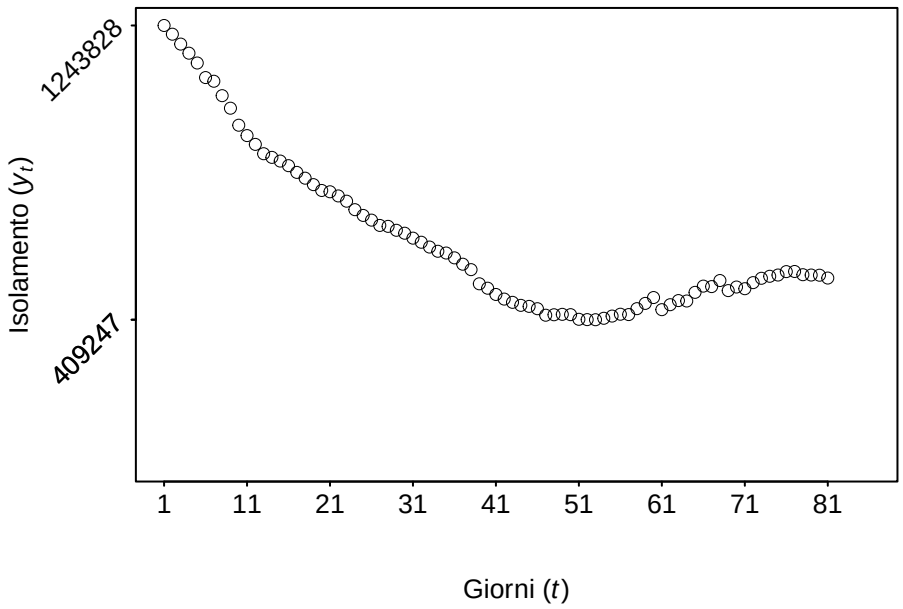
Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100
data	0	1	640634.8	227705.8	409247	463524	536906	772106	1243828

Si **rappresenta graficamente** la serie osservata

```
n<-length(df1$Isolati)
ma<-max(df1$Isolati)
mi<-min(df1$Isolati)
ytick<-c(mi,409247,ma)
xtick<-c(1,11, 21, 31, 41, 51, 61, 71, 81)
plot(df1$Isolati,
     ylab=expression("Isolamento ("*italic(y[t])*")"),
     xlab=expression("Giorni ("*italic(t)*")"),
     yaxt="n",
     xaxt="n",
     xlim =c(1,n+5),
```

```
     ylim = c(0,ma+10),
     lwd = 0.5,
     lty = 1,col ="black" )
axis(side=2, at=ytick, labels = FALSE,
     cex.lab = 0.5, padj = 2,tck=-0.009)
text(par("usr")[1],
     ytick,
     labels = ytick, srt = 45,
     pos = 2, xpd = TRUE, cex.lab = 0.1)
axis(side=1, at=xtick,
     labels = FALSE, cex=0.1,tck=-0.009)
text(x=xtick,
     par("usr")[3],
     labels = xtick,
     cex.lab = 0.5,
     pos = 1, xpd = TRUE)
```



Il modello di Poisson non omogeneo si stima definendo le **covariate** riferite al tempo e menzionate in precedenza: il *trend lineare*, *quadratico* e *log-lineare* nel modo seguente

```
regressors1Italy <- cbind(linearTrend=seq(along=df1$Isolati),
                          quadTrend = seq(along=df1$Isolati)^2/100,
                          linlogTrend = log(seq(along=df1$Isolati)))
head(regressors1Italy)
```

```
#>      linearTrend quadTrend linlogTrend
#> [1,]           1      0.01   0.0000000
#> [2,]           2      0.04   0.6931472
#> [3,]           3      0.09   1.0986123
#> [4,]           4      0.16   1.3862944
#> [5,]           5      0.25   1.6094379
#> [6,]           6      0.36   1.7917595
```

Per stimare il modello con il metodo della massima versosimiglianza si utilizza la funzione `tscount::tsglm` della libreria (*da installare*, tscount proposta da Liboschik et al., 2020) (Analysis of Count Time Series)

Si noti che la prima opzione del chunk seguente serve per *rimuovere la notazione scientifica* che alcune volte può essere stampata in output.

```
options(scipen = 100)
require(tscount)
M3Italy <- tsglm(ts=df1$Isolati,
                 link = "log",
                 model=list(past_obs=1),
                 xreg=regressors1Italy,
                 distr = "poisson")
```

Come già sperimentato per la funzione `glm` occorre fornire:

- i dati osservati come serie univariata dei conteggi,
- la funzione `link` per il modello log-lineare per λ_t,
- definire l'ordine del ritardi con `past_obs`,
- la matrice delle covariate temporali `xreg` e la distribuzione di riferimento.

```
summary(M3Italy)
#>
#> Call:
#> tsglm(ts = df1$Isolati, model = list(past_obs = 1), xreg = regressors1Italy,
#>     link = "log", distr = "poisson")
#>
#> Coefficients:
#>              Estimate  Std.Error  CI(lower)  CI(upper)
#> (Intercept)   0.86375   0.034280    0.79656    0.93093
#> beta_1        0.93892   0.002431    0.93415    0.94368
#> linearTrend   0.00619   0.000122    0.00595    0.00643
#> quadTrend    -0.00489   0.000103   -0.00509   -0.00469
#> linlogTrend  -0.05969   0.000516   -0.06070   -0.05868
#> Standard errors and confidence intervals (level =  95 %) obtained
#> by normal approximation.
#>
#> Link function: log
#> Distribution family: poisson
#> Number of coefficients: 5
#> Log-likelihood: -29574.38
#> AIC: 59158.76
#> BIC: 59170.73
#> QIC: 59158.76
```

Viene fornito in output il valore massimo della funzione di log-verosimiglianza a convergenza, il numero dei parametri stimati con il modello, ed i valori degli indici di selezione BIC (criterio d'informazione Bayesiano), AIC (criterio di Akaike) e una sua versione aggiustata definita QIC (quasi information criterion).

Dalla sintesi dei risultati delle stime del modello notiamo che l'*intercetta* del modello ha un valore positivo e corrisponde al valore del logaritmo di lambda quando tutte le covariate hanno valore 0.

Le stime dei coefficienti di regressione sono fornite insieme all'**intervallo di confidenza** calcolato al livello del 95% utilizzando il metodo asintotico (basato sull'approssimazione dello stimatore alla distribuzione Normale).

Si noti che tutti i coefficienti sono positivi tranne i due coefficienti riferito al trend quadratico e al trend log-lineare nel tempo.

Si noti che è di particolare interesse confrontare i valori osservati con i **valori interpolati**.

E' inoltre di interesse fare delle **previsioni**: calcolare i conteggi attesi in base al modello stimato per gli isolati di oggi (20 ottobre 2022) e per quelli dei prossimi 4/5 giorni.

Come per la funzione `glm` i valori previsti si ottengono con la funzione `predict`.

Si noti che la funzione `tsglm::predict` implementa il `method="bootstrap"` ovvero la distribuzione dei valori previsti è approssimata con il bootstrap parametrico basato di default su `B = 1000` traiettorie simulate dal modello stimato. Si noti che in questo modo si associa alla stima puntuale della previsione un intervallo definito *intervallo predittivo* calcolato con una confidenza di 0.95.

```
go <- 5

TT <- length(df1$date)

P3Italy <-predict(M3Italy,
                  newxreg = data.frame(linearTrend = ((TT+1):(TT+go)),
                                       quadTrend = ((TT+1):(TT+go))^2/100,
                                       linlogTrend = log((TT+1):(TT+go))),
                  n.ahead=go,
                  method="bootstrap" )
```

La seguente tabella riporta i valori previsti degli isolati, insieme ai limiti dell'**intervallo predittivo** al 95%, per oggi e per i prossimi 4 giorni

```
pred<-data.frame(cbind(P3Italy$pred,P3Italy$interval))
colnames(pred)<-c("previsti", "PIinf", "PIsup")
pred
```

previsti	PIinf	PIsup
514103.7	512780	515516
500611.5	498653	502346
486952.6	484739	488954
473149.1	470835	475373
459224.2	456703	461674

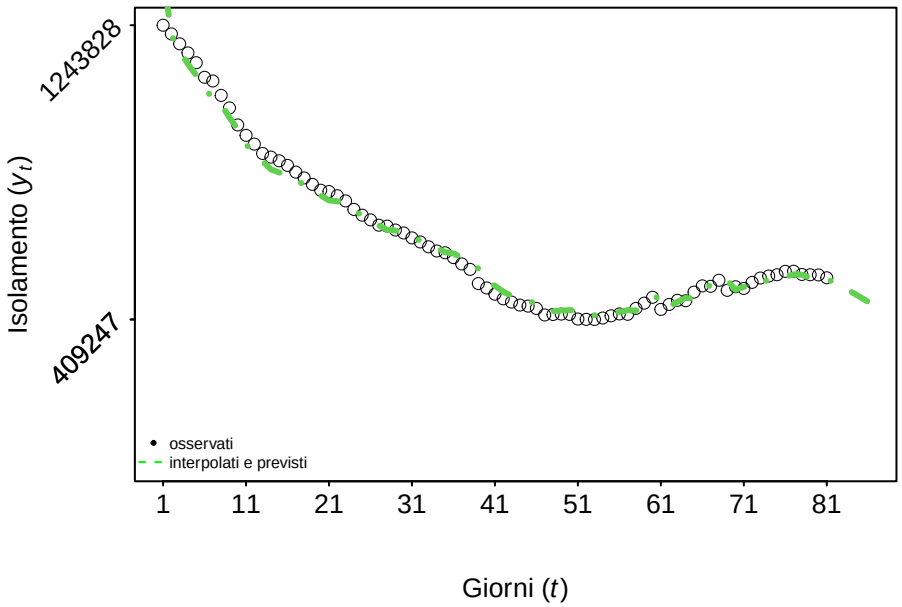
Si nota che il numero degli isolati atteso per domani è di 514103 ed il trend è quello che iil numero decresca nei giorni successivi.

Si nota che le ampiezze degli intervalli sono modeste e che queste crescono al crescere del giorno previsto. Ciò denota, come ci si aspetta, un aumento dell'**incertezza** delle previsioni via via che ci si allontana dall'ultima data di osservazione.

Le seguente figura mostra i conteggi **osservati** dei pazienti isolati per tutto il periodo considerato insieme ai valori **interpolati** con il modello stimato a cui vengono aggiunti i valori **previsti** per il periodo temporale riportato che sono visualizzati dopo la retta grigia che indica il giorno di ieri

```
ytick<-c(mi,409247,ma)
xtick<-c(1,11, 21, 31, 41, 51, 61, 71, 81)
plot(df1$Isolati,
     ylab=expression("Isolamento ("*italic(y[t])*")"),
     xlab=expression("Giorni ("*italic(t)*")"),
     yaxt="n",
     xaxt="n",
     xlim =c(1,n+5),
     ylim = c(0,ma),
     lwd = 0.5,
     lty = 1,col ="black" )
abline(v=240, col = "gray")
axis(side=2, at=ytick, labels = FALSE,
     cex.lab = 0.5, padj = 2,tck=-0.009)
text(par("usr")[1],
     ytick,
     labels = ytick, srt = 45,
     pos = 2, xpd = TRUE, cex.lab = 0.1)

axis(side=1, at=xtick,
     labels = FALSE, cex=0.1,tck=-0.009)
text(x=xtick,
     par("usr")[3],
     labels = xtick,
     cex.lab = 0.5,
     pos = 1, xpd = TRUE)

lines(c(M3Italy$fitted.values,
        pred$previsti),
      lwd=3, col = 3, lty = 4)
legend("bottomleft",
```

```
       pch = c(20,NA,NA,NA,NA),
       lty = c(NA,2),
       legend = c("osservati", "interpolati e previsti"),
       col = c("black", "green"),
       bty = "n",
       x.intersp = 0.1,
       cex= 0.6,  pt.cex = .5,
       xpd = TRUE,
       text.width = 0.0001)
```



Modello autoregressivo con distribuzione Binomiale Negativa

Nel presente esempio si consida un modello basato sulla distribuzione Binomiale Negativa per la serie dei conteggi dei casi di pazienti affetti da COVID-19.

Nell' applicazione precedente la stima del parametro autoregressivo è diversa da zero $\hat{\rho} = 0.99$ (beta1).

Il modello basato sulla distribuzione Binomiale Negativa include il parametro di *overdispersion* in modo che la il valore atteso dei conteggi è $\hat{\lambda}_t$ mentre la varianza è diversa dal valore atteso e risulta $\hat{\lambda} \cdot (1 + \hat{\lambda} \cdot \hat{\sigma}^2)$.

```
options(scipen = 100)
require(tscount)
M4Italy <- tsglm(ts=df1$Isolati,
                 link = "log",
```

```
                  model=list(past_obs=1),
                  xreg=regressors1Italy,
                  distr = "nbinom")
```

ovvero specificando la distribuzione Binomiale Negativa con `distr = "nbinom"`.

In questo caso il modello presenta un parametro aggiuntivo riferito a σ^2.

```
summary(M4Italy)
#>
#> Call:
#> tsglm(ts = df1$Isolati, model = list(past_obs = 1), xreg = regressors1Italy,
#>     link = "log", distr = "nbinom")
#>
#> Coefficients:
#>              Estimate  Std.Error  CI(lower)  CI(upper)
#> (Intercept)   0.86375    0.87043  -0.842271   2.569761
#> beta_1        0.93892    0.06192   0.817555   1.060276
#> linearTrend   0.00619    0.00354  -0.000759   0.013134
#> quadTrend    -0.00489    0.00295  -0.010667   0.000887
#> linlogTrend  -0.05969    0.01779  -0.094550  -0.024833
#> sigmasq       0.00105         NA         NA         NA
#> Standard errors and confidence intervals (level =  95 %) obtained
#> by normal approximation.
#>
#> Link function: log
#> Distribution family: nbinom (with overdispersion coefficient 'sigmasq')
#> Number of coefficients: 6
#> Log-likelihood: -914.3805
#> AIC: 1840.761
#> BIC: 1855.128
#> QIC: 67318.39
```

Dalla sintesi dei risultati del modello notiamo che la stima del parametro σ^2 è 0.00375 che risulta pertanto postiva indicando *overdispersion*.

Per calcolare l'errore standard occorre applicare il bootstrap che verrà introdotto nei giorni seguenti.

I valori previsti per i cinque giorni successivi (dal 19 al 23 ottobre) con questo modello si ottengono con la funzione `predict` ed in modo analogo a quanto mostrato per il modello stimato con la distribuzione di Poisson possiamo calcolarle dopo aver specificato le covariate temporali.

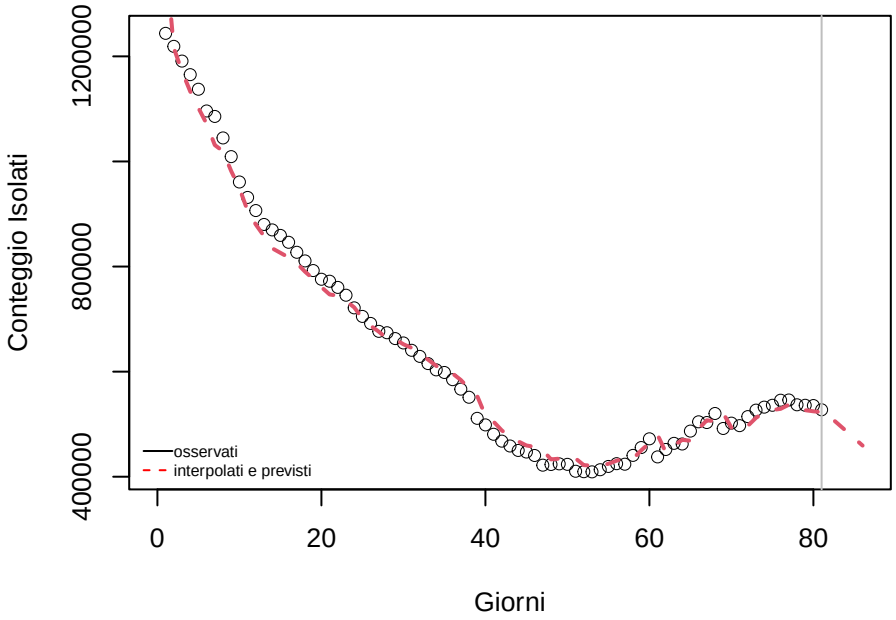
```
go <- 5

TT <- length(df1$date)

P4Italy <-predict(M4Italy,
                  newxreg = data.frame(linearTrend = ((TT+1):(TT+go)),
                                       quadTrend = ((TT+1):(TT+go))^2/100,
                                       linlogTrend = log((TT+1):(TT+go))),
                  n.ahead=go,
                  method="bootstrap" )
pred<-cbind(P4Italy$pred,P4Italy$interval)
colnames(pred)<-c("previsti", "PIinf", "PIsup")
pred
#>      previsti  PIinf  PIsup
#> [1,] 514103.7 481137 547798
#> [2,] 500611.5 458864 547757
#> [3,] 486952.6 440011 540023
#> [4,] 473149.1 421656 530036
#> [5,] 459224.2 405014 519757
```

E' interessante notare che il valore puntuale del conteggio previsto è simile a quello ottenuto con il modello di Poisson tuttavia in base all'overdispersion l'itervallo predittivo calcolato con una fiducia di 0.95 ha maggiore ampiezza.

I conteggi **osservati** dei pazienti isolati per tutto il periodo considerato insieme ai valori **interpolati** con il modello stimato e i valori **previsti** possono essere rappresentati graficamente

```
plot(df1$Isolati, lwd = 0.5,
     lty = 1,col ="black",
     xlab = "Giorni",
     ylab= "Conteggio Isolati",
     ylim = c(mi,ma+2), xlim = c(0,n+5))
abline(v=81, col = "gray")
lines(c(M4Italy$fitted.values,P4Italy$pred),
      lwd=2, col = 2, lty = 2)
legend("bottomleft",
       lty = c(1,2),
```

```
       legend = c("osservati", "interpolati e previsti"),
       col = c("black", "red"),
       bty = "n",
       x.intersp = 0.1,
       cex= 0.6,  pt.cex = .5,
       xpd = TRUE,
       text.width = 0.0001)
```



Valutazione della prevalenza nel tempo

Considerando la popolazione complessiva è possibile stimare in base al modello la *prevalenza* che misura la proporzione dell'evento (Isolamento nel presente caso) in una popolazione in un dato momento.

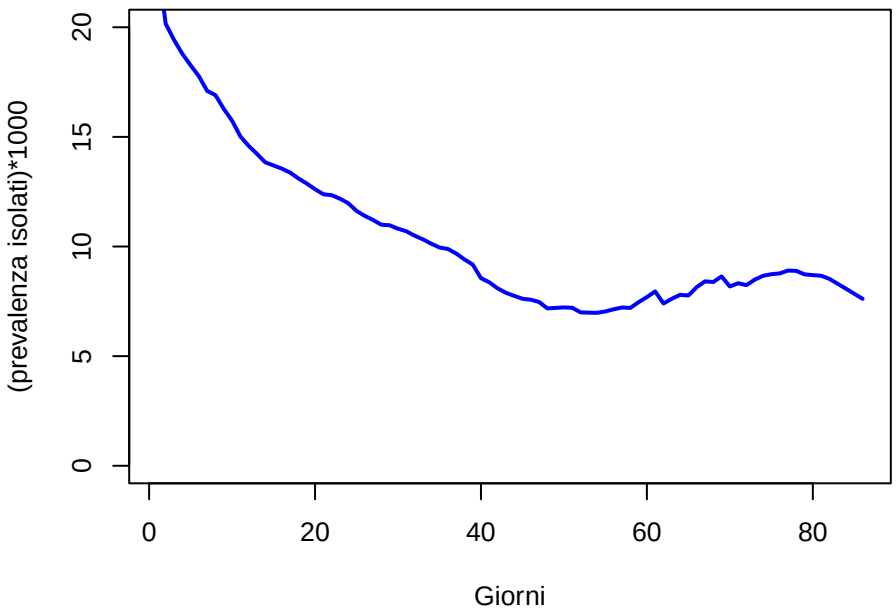
Per l'Italia la numerorità della popolazione totale è reperibile dal sito dell'Istituto Nazionale di Statistica che riporta che popolazione residente al 1 Gennario 2020 è di 60317000 unità https://www.istat.it/it/archivio/238447

Utilizzando quest'informazione è possibile calcolare la prevalenza dividendo i valori per questo numero (e moltiplicando per mille)

```
prev<-c(M4Italy$fitted.values,P4Italy$pred)/60317000
prev<-prev*1000
```

E verificare in base al modello stimato come questa si è modificata nel tempo ad esempio con il seguente grafico

```
plot(prev, type = "l",
     xlab = "Giorni",
     ylim = c(0,20),
     lwd=2,
     col = "blue", ylab = "(prevalenza isolati)*1000")
```



che permette di evidenziare la prevalenza di casi ogni 1000 abitanti. In questo modo è possibile anche fare confronti tra diversi paesi.

Metodi di ricampionamento: il bootstrap

Dati nervo

Nel libro di Cox, D. R., & Lewis, P. A. W. (1966). *The statistical analysis of Series of Events*, Methuen's Monographs on Applied Probability

vengono utilizzati i dati della serie di 799 tempi di attesa tra pulsazioni successive lungo la fibra del nervo, analizzati nel seguito.

```
nervo<-read.table("nervo.dat", header = TRUE)
head(nervo$A)
#> [1] 0.21 0.18 0.02 0.15 0.15 0.24
require(skimr)
skim_without_charts(nervo)
```

Table 10: Data summary

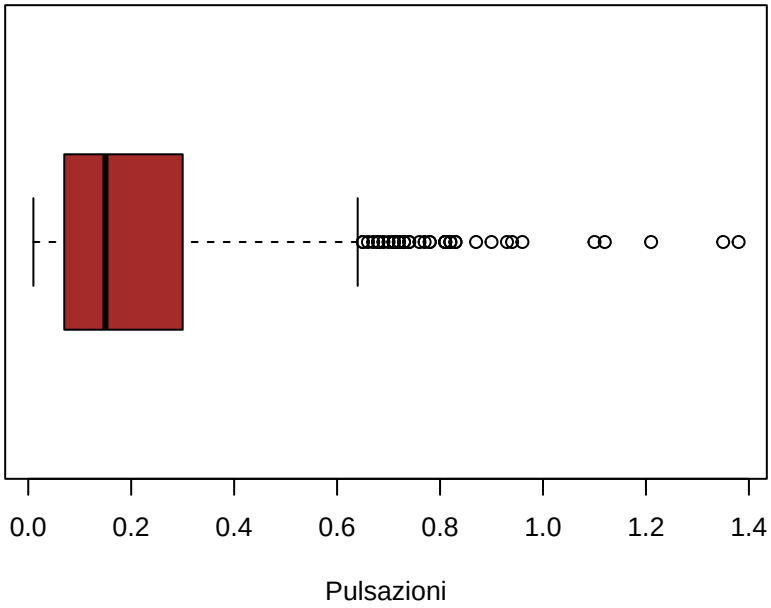
Name	nervo
Number of rows	799
Number of columns	1
Column type frequency:	
numeric	1
Group variables	None

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100
A	0	1	0.22	0.21	0.01	0.07	0.15	0.3	1.38

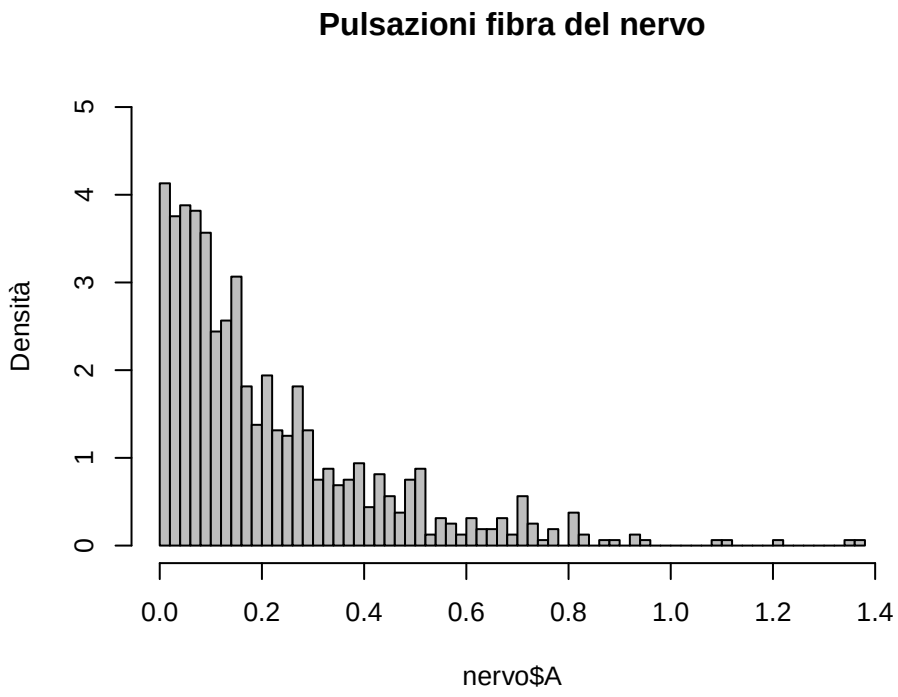
I tempi variano da 0.01 a 1.38 mille secondi e si nota l'asimmetria della distribuzione in quanto la media è superiore alla mediana.

```
boxplot(nervo$A, xlab="Pulsazioni", horizontal=TRUE, col = "brown")
```



Si nota che la mediana di 0.15 è maggiormente rappresentata nei dati rispetto alla media.

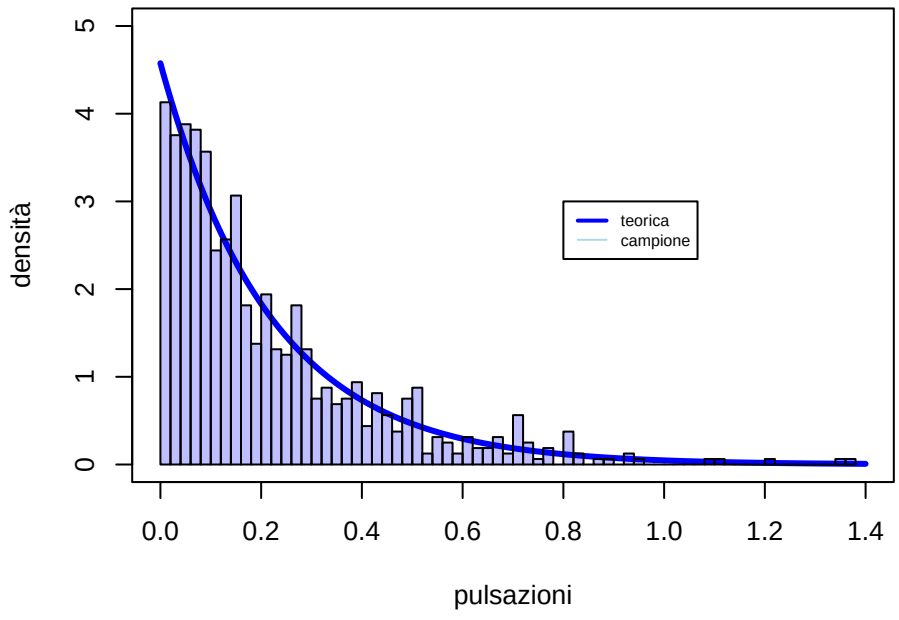
```
hist(nervo$A,
     breaks = 50,
     ylim=c(0,5),
     main = "Pulsazioni fibra del nervo",
     ylab="Densità",
     freq =FALSE,
     col = "grey")
```



Per verifica si disegna la curva della densità esponenziale con media (1/tasso) pari a quella osservata nei dati campionari e si aggiunge la distribuzione empirica, si calcola il tasso

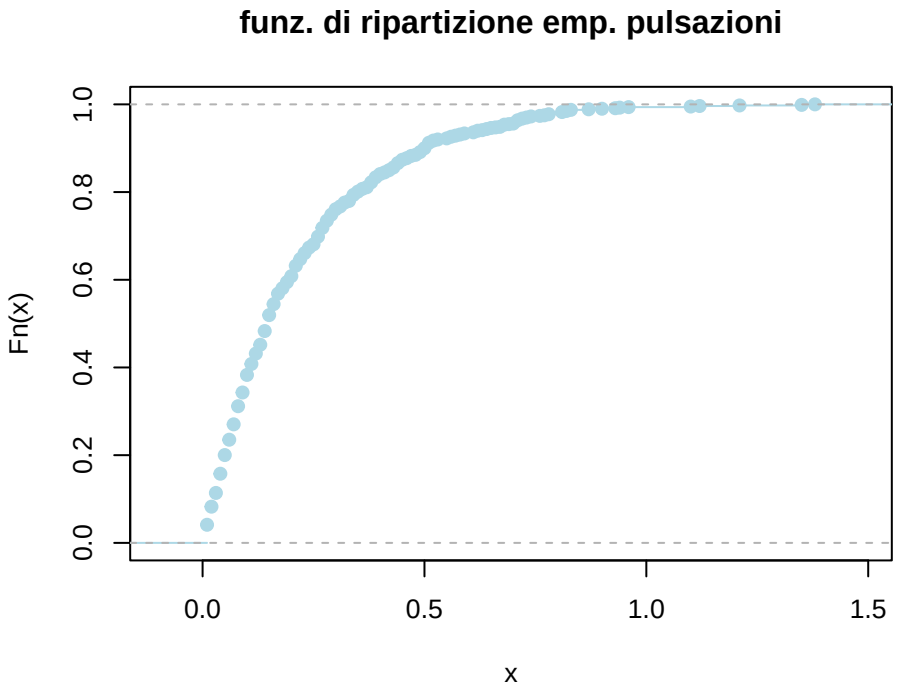
```
rateA <- 1/mean(nervo$A); rateA
#> [1] 4.575126
```

```
x<-seq(0,1.4,length=799)
h<-dexp(x,rate =  rateA)
plot(x,h,
    ylim=c(0,5),
     type="l", col = "blue",
     lwd = 3,
     xlab = "pulsazioni",
    ylab = "densità")
hist(nervo$A,
     breaks = 50,
     col = rgb(0,0,1,1/4),
     freq =FALSE, add=T)
legend(0.8,3, c("teorica", "campione"),
       col = c("blue", "lightblue"),
       lty  = c(1,1),
       lwd = c(2,1),
       cex = 0.6)
```



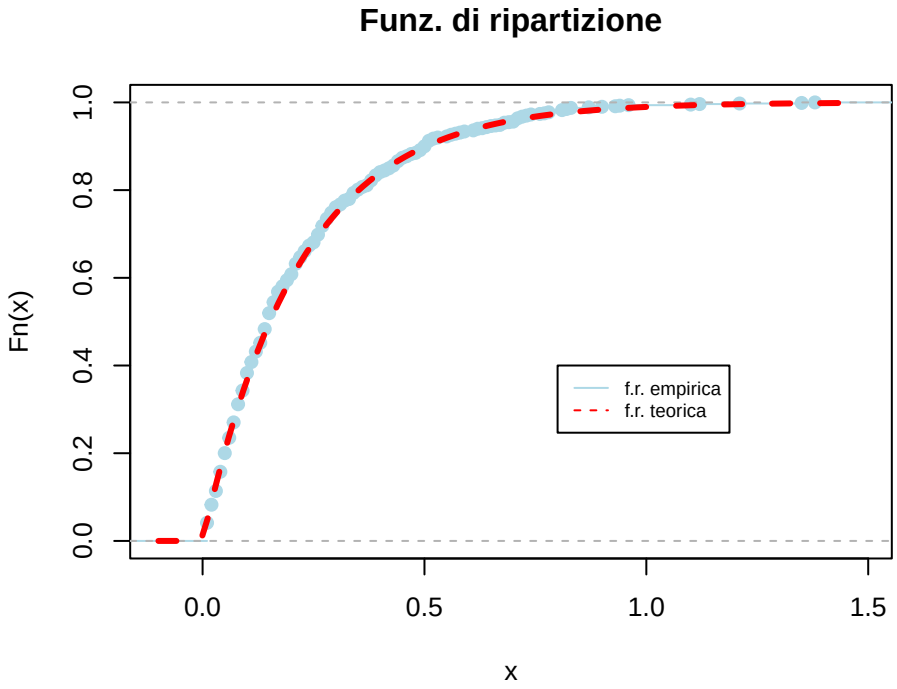
Si disegna la funzione di ripartizione empirica che rappresenta lo stimatore di sostituzione **plug-in** della funzione di ripartizione teorica

```
plot(ecdf(nervo$A),
     col="lightblue",
     main= "funz. di ripartizione emp. pulsazioni")
```



Si aggiunge alla funzione empirica la funzione teorica

```
plot(ecdf(nervo$A),
     col="lightblue",
     main= "Funz. di ripartizione")
#
curve(pexp(x,rate=rateA),
      lty='dashed',
      col='red',
      lwd='3',
      add=TRUE)
#
legend(0.8,0.4, col=c("lightblue","red"),
c("f.r. empirica","f.r. teorica"), lty=c(1,2),
cex=0.7)
```



Indice di asimmetria

Considerando che per una variabile casuale X con media μ e varianza σ^2 l'asimmetria è:

$$\kappa = E\left[\frac{(X-\mu)^3}{\sigma^3}\right] = \frac{\int (x-\mu)^3 dF(x)}{\left(\int (x-\mu)^2 dF(x)\right)^{3/2}},$$

e $\kappa = 0$ nella distribuzione Normale.

L'usuale stima dell'indice di asimmetria si ottiene con lo **stimatore di sostituzione o plug-in** che è il seguente

$$\hat{\kappa} = T(\hat{F}_n) = \frac{\int (x-\mu)^3 d\hat{F}_n(x)}{\hat{\sigma}^3} = \frac{1/n \sum_i^n (X_i - \bar{X}_n)^3}{\hat{\sigma}^3}.$$

Nella libreria `e1071` è presente la funzione che calcola questo indice

```
require(e1071)
skewness(nervo$A)
#> [1] 1.757943
```

Bootstrap

Dato che lo stimatore κ non ha distribuzione nota si può associare alla stima puntuale dell'assimetria ottenuta con lo stimatore di sostituzione una misura della sua variabilità.

A questo punto occorre applicare il metodo bootstrap attraverso i seguenti passi:

1. Si determina $X_1^*, \dots, X_n^* \sim \hat{F}_n$;
2. Si calcola per ogni determinazione il valore della statistica $T_n^* = g(X_1^*, \dots, X_n^*)$;
3. Si ripetono B volte i passi 1 e 2 e si ottengono i valori $T_{n,1}^*, \dots, T_{n,B}^*$;
4. Si calcola la deviazione standard

$$sd_{boot} = \sqrt{\frac{1}{B-1}\sum_{r=1}^{B}\left(T_{n,b}^* - \frac{1}{B}\sum_{r=1}^{B}(T_{n,r}^*)\right)^2}.$$

- PASSO 1: si ottiene il **primo campione** dal campione originario con la la funzione `sample` che pemette di ottenere `n` replicazioni con **reimmissione** specificando '{replace=TRUE}

```
n <- length(nervo$A)
B1 <- sample(nervo$A, n, replace = TRUE)
summary(B1)
#>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#>  0.0100  0.0700  0.1600  0.2332  0.3000  1.3800
```

- Si ottiene il **secondo campione**

```
B2 <- sample(nervo$A, n, replace = TRUE)
summary(B2)
#>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#>  0.0100  0.0700  0.1400  0.2136  0.2800  1.3800
```

- Si ottiene il **terzo campione**

```
B3 <- sample(nervo$A, n, replace = TRUE)
summary(B3)
#>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#>  0.0100  0.0700  0.1500  0.2152  0.2900  1.3500
```

- PASSO 2: si calcola il valore dell'indice di asimmetria in ogni campione ed il valori realizzati per ogni replicazione bootstrap si salvano nel vettore `s`

```
s1<-skewness(B1)
s2<-skewness(B2)
s3<-skewness(B3)
s <- c(s1,s2,s3); s
#> [1] 1.439517 1.912271 1.424755
```

oppure con la funzione **apply**

```
BB<-cbind(B1,B2,B3)
apply(BB,2,skewness)
#>       B1       B2       B3
#> 1.439517 1.912271 1.424755
```

- PASSO 3: si calcola la **deviazione standard** delle replicazioni bootstrap si tratta della stima per l'errore standard da associare al valore dell'indice di asimmetria calcolato sul campione dei dati originari attraverso lo stimatore plug-on che è pari a 1.76.

Utilizzando solo 3 replicazioni bootstrap si ha un errore standard pari a

```
sd(s)
#> [1] 0.2773044
```

Ovviamente $B = 3$ non è abbastanza e procedura deve essere ripetuta almeno con $B = 200$ campioni bootstrap.

Utilizzo del ciclo for

Nel seguente codice si campiona 1000 volte dal vettore nervo e si salva il valore realizzato su ogni campione dell'indice di asimmnetria in `Tboot`. Poi si calcola la deviazione standard di queste realizzazioni

```
B <- 1000
n <- length(nervo$A)
Tboot <- rep(0, B)
set.seed(16253)
for (i in 1:B) {
      Xstar <- sample(nervo$A,
                      n,
                      replace = TRUE)
      Tboot[i] <- e1071::skewness(Xstar)
}

head(Tboot)
#> [1] 1.608102 1.532254 1.751126 1.847124 1.608976 1.736952
summary(Tboot)
#>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#>   1.223   1.630   1.742   1.743   1.850   2.444

seTboot <- sd(Tboot); seTboot
#> [1] 0.1694565
```

L'accuratezza per il valore di simmetria calcolato con lo stimatore plug-in sul campione originario è stimata con il metodo bootstrap in base a 1000 replicazioni del campione originario ed è pari a 0.16 (come si vede è minore di quello calcolato con $B = 3$).

Ovviamente come per qualsiasi altra procedura statistica, è **fondamentale che i dati siano sufficientemente informativi** sul fenomeno di interesse.

Intervalli di confidenza bootstrap

Come nell'inferenza classica anche in questo contesto è preferibile riportare accanto alla stima puntuale per il parametro θ un intervallo di confidenza e la probabilità di copertura chiamata livello di confidenza.

Metodo del percentile

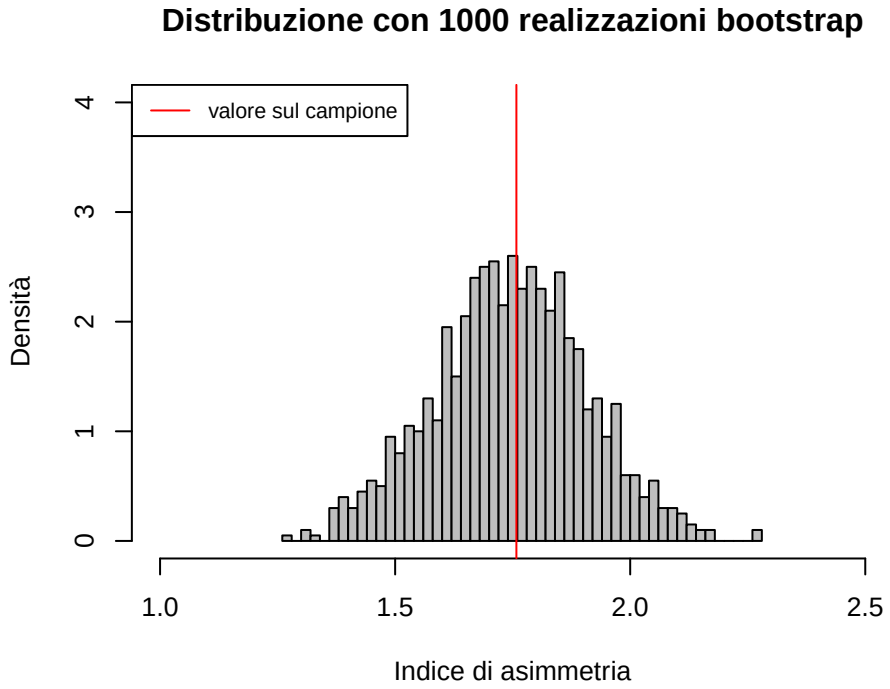
Si considerano i dati dei tempi delle pulsazioni e la stima dell'indice di asimmetria

```
nervo<-read.table("nervo.dat", header = TRUE)
```

Si ottiene la distribuzione bootstrap utilizzando B=1000 campioni e si aggiunge il valore della stima calcolato nei dati osservati

```
B <- 1000
n <- length(nervo$A)
Tboot <- rep(0, B)
for (i in 1:B) {
      Xstar <- sample(nervo$A,
                      n,
                      replace = TRUE)
      Tboot[i] <- e1071::skewness(Xstar)
}
```

```
sk <- e1071::skewness(nervo$A)
hist(Tboot,
     breaks=50,
     freq=FALSE,
     main = "Distribuzione con 1000 realizzazioni bootstrap",
     xlab = "Indice di asimmetria",
     ylim = c(0,4),
     col= "gray",
     ylab = "Densità",
     xlim = c(1,2.5))
abline(v=sk, col="red")
legend("topleft", 2,
       c("valore sul campione"),
       col = "red",
       lty= 1,
       cex = 0.8)
```



L'intervallo di confidenza bootstrap ottenuto il metodo del percentile si calcola in base alla distribuzione empirica delle replicazioni bootstrap.

Quando si utilizza il metodo del percentile come estremi dell'intervallo di confidenza si prendono i quantili della distribuzione ad un fissato il livello di confidenza ad esempio 0.95.

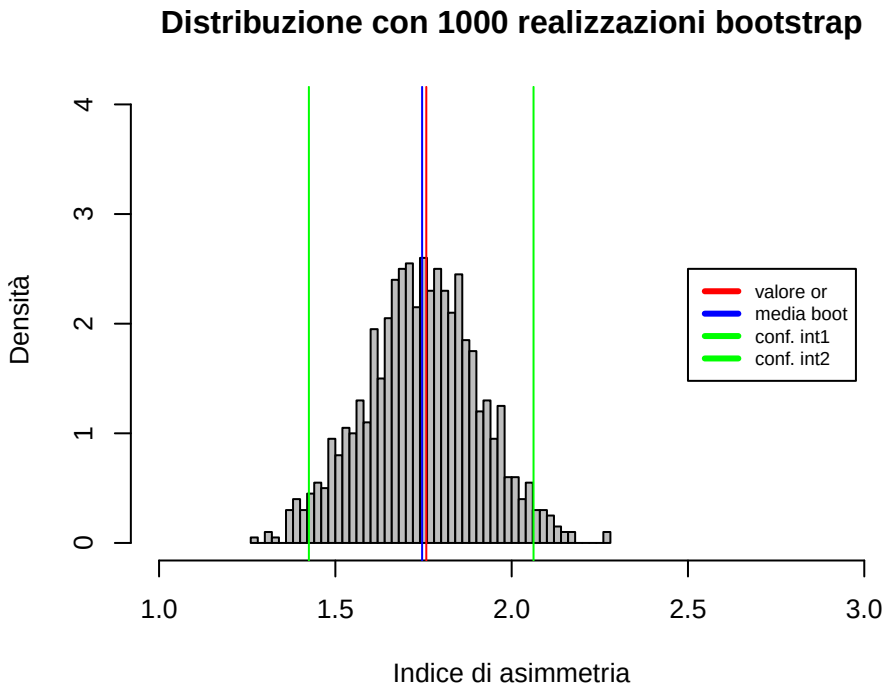
Pertanto questo è un intervallo di confidenza al 95% per l'indice di asimmetria.

Ricordiamo che:

- L'intervallo di confidenza è un intervallo di valori intorno alla stima puntuale del parametro entro il quale si ritiene plausibile che sia contenuto il vero valore del parametro con un certo livello di confidenza.
- Il **livello di confidenza** è una probabilità che va riferita ad un intervallo di confidenza con estremi aleatori nel campionamento ripetuto.
- Il valore 0.95 invece **non** è la probabilità che θ appartenga all'intervallo osservato.
- Abbiamo fiducia che θ appartenga all'intervallo osservato perchè la procedura lo comprende nel 95% dei casi.

L'istogramma della distribuzione bootstrap può essere completato con il valore medio ottenuto dalle replicazioni bootstrap e con gli estremi dell'intervallo di confidenza

```r
sB<- mean(Tboot)
Q <- quantile(Tboot, c(0.025, 0.975))

Q[1]; Q[2]
#>    2.5%
#> 1.42489
#>    97.5%
#> 2.062046
#
hist(Tboot,
      breaks = 60,
      freq=FALSE,
      main = "Distribuzione con 1000 realizzazioni bootstrap",
      xlab = "Indice di asimmetria",
     ylim = c(0,4),
      col = "gray",
       ylab = " Densità",
     xlim = c(1, 3))
#
abline( v = c(sk,sB,Q[1], Q[2]),
       col = c("red", "blue", "green", "green"))
#
legend(2.5,2.5,
       c("valore or", "media boot",
         "conf. int1", "conf. int2"),
      col = c("red", "blue", "green", "green"),
       lty = c(1,1,1,1),
       lwd = c(3,3,3,3),
       cex = 0.7)
```



Metodo Bias Corrected Accelerated Bootstrap

L'intervallo di confidenza con il metodo **Bias Corrected Accellerated bootstrap** si calcola con la funzione `bcanon` presente nella libreria `bootstrap` che richiede come input il vettore dei dati, il numero di replicazioni bootstrap e la funzione (theta) riferita allo stimatore

```
n
#> [1] 799
B<-1000
theta<-e1071::skewness
require(bootstrap)
set.seed(1013)
CIbca<-bcanon(nervo$A, B,
              theta,
              alpha = c(0.025, 0.975))
CIbca$confpoints
#>      alpha bca point
#> [1,] 0.025  1.470134
#> [2,] 0.975  2.135836
```

Si nota che gli estremi inferiore e superiore dell'intervallo di confidenza al 95% ottenuti con il metodo bca sono 1.47 e 2.13, rispettivamente. Differiscono da quelli calcolati con il metodo del percentile. Infatti la funzione restituisce anche i valori stimati delle due costanti:

- **accellerazione** a
- **di correzione per la distorsione** z_0

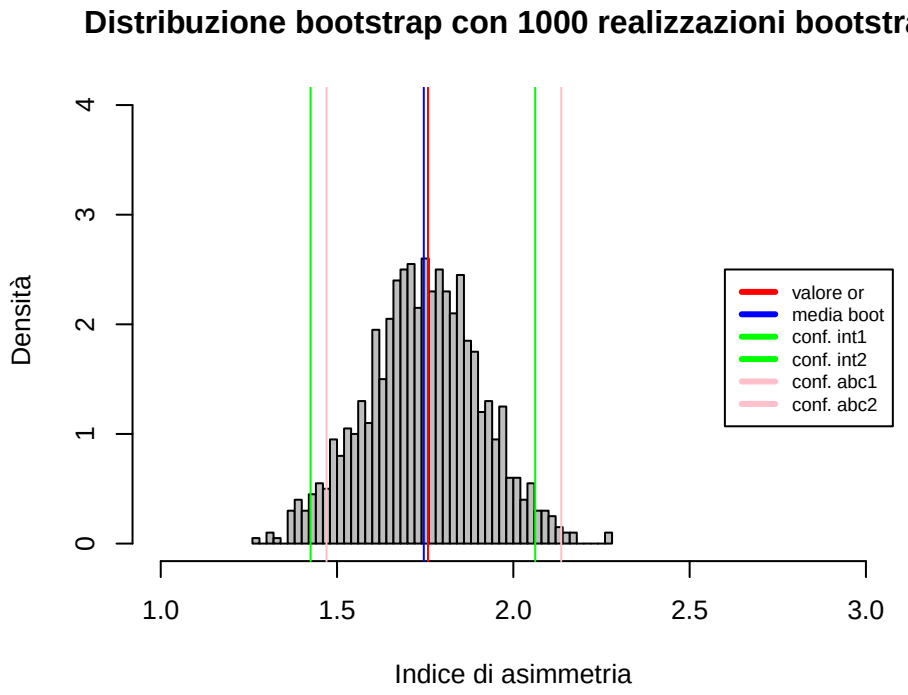
e questi sono diversi da zero

```
CIbca$acc
#> [1] 0.05746515
CIbca$z0
#> [1] 0.06019541
```

Gli estremi dell'intervallo ottenuti con questo metodo possono essere aggiunti al grafico nel modo seguente:

```
hist(Tboot,
      breaks = 60,
      freq=FALSE,
      main = "Distribuzione bootstrap con 1000 realizzazioni bootstrap",
      xlab = "Indice di asimmetria",
     ylim = c(0,4),
      col = "gray",
       ylab = " Densità",
     xlim = c(1, 3))
#
abline( v = c(sk,
              sB,
              Q[1],
              Q[2],
              CIbca[["confpoints"]][3],
              CIbca[["confpoints"]][4]),
       col = c("red",
               "blue",
               "green",
               "green",
               "pink",
               "pink"))
#
legend(2.6,2.5,
       c("valore or",
         "media boot",
         "conf. int1",
         "conf. int2",
         "conf. abc1",
         "conf. abc2"),
```

```
      col = c("red",
              "blue",
              "green",
              "green",
              "pink",
               "pink"),
       lty = c(1,1,1,1,1,1),
       lwd = c(3,3,3,3,3,3),
      cex = 0.7)
```



Il valore dell'indice di asimmetria per le pulsazioni consecutive lungo la fibra del nervo è compreso tra questi due estremi con una fiducia di 0.95.

Stimatore del rischio relativo: intervalli di confidenza bootstrap

La seguente tabella di contingenza riporta le frequenze assolute (conteggi) per due caratteri binari dove Y è intesa come variabile risposta e X è intesa come covariata. Ad esempio Y è la variabile casuale che rappresenta l'evento infarto e X l'assunzione di placebo o di aspirina in un campione casuale di persone.

I numeri $N_{11}, N_{01}, N_{10}, N_{00}$ rappresentano le frequenze congiunte di $X_i = i$, $i = 0, 1$ e $Y_i = j$, $j = 0, 1$

X		Y	
	1	0	
1	N_{11}	N_{10}	$N_{1.}$
0	N_{01}	N_{00}	$N_{0.}$
	$N_{.1}$	$N_{.0}$	$N_{..}$

Le frequenze esterne (marginali) sono le somme corrispondenti di riga e di colonna.

Negli studi di coorte vengono confrontati individui che presentano e non presentano una certa qualità/patologia. Nell'ambito del campionamento prospettivo (che si differenzia dal campionamento retrospettivo caso-controllo) si assumono come campioni indipendenti: il primo formato da individui **esposti** ed il secondo da persone **non esposte**. In questo modo si può stimare la probabilità dell'evento data l'esposizione.

Il **Rischio Relativo** (RR, relative risk o risk ratio) è una misura d'associazione spesso utilizzata in ambito medico per confrontare ad esempio una risposta rispetto a due trattamenti.

Si calcola come **rapporto** tra la frequenza di coloro che presentano questa caratteristica (hanno avuto l'infarto) e sono stati esposti o non sono stati esposti (all'aspirina). La misurà è definita come rapporto tra:

- la proporzione degli **esposti**: coloro che hanno avuto l'infarto e hanno preso l'aspirina N_{11} sul totale di coloro che hanno preso l'aspirina $N_{1.}$;
- la proporzione dei **non esposti**: ovvero coloro che hanno avuto l'infarto e non hanno assunto aspirina N_{01} sul totale di coloro che non hanno assunto aspirina $N_{0.}$

$$RR = \frac{\frac{N_{11}}{N_{1.}}}{\frac{N_{01}}{N_{0.}}}.$$

Si dimostra che $RR = 1$ è condizione sufficiente e necessaria per l'indipendenza tra Y e X.

I conteggi sono considerati delle v.c. aventi una distribuzione Binomiale:

$$N_{01} \sim Bin(N_{0.}, p_0) \qquad N_{11} \sim Bin(N_{1.}.p_1)$$

dove $p_0 = P(Y = 1|X = 0)$ e $p_1 = P(Y = 1|X = 1)$.

Gli stimatori di massima verosimiglianza per queste probabilità si ottengono come

$$\hat{p}_0 = \frac{N_{01}}{N_{0.}} \qquad \hat{p}_1 = \frac{N_{11}}{N_{1.}}$$

e lo stimatore di massima verosimiglianza per il rischio relativo (RR) si ottiene

$$\hat{RR} = \frac{\hat{p}_1}{\hat{p}_0}.$$

Se le probabilità sono basse il RR è simile al rapporto delle quote o odds ratio (OR)

$$OR = \frac{p_1(1-p_1)}{p_0(1-p_0)}.$$

Negli studi **retrospettivi** occorre conoscere la probabilità di esposizione per stimare RR. Se quest'ultima non è nota il rischio relativo non può essere calcolato ed occorre utilizzare l'OR.

Un'altra misura di associazione è la differenza delle probabilità la cui stima di massima verosimiglianza si ottiene come

$$\hat{D} = \hat{p}_1 - \hat{p}_0.$$

Si dimostra che le due variabili casuali sono **statisticamente indipendenti** se

$$RR = 1 \quad D = 0 \quad OR = 1.$$

La distribuzione del rischio relativo non è nota e anche asintoticamente non è approssimabile a quella Normale. Nel seguito si calcola il valore del rischio relativo e si utilizza il bootstrap per ottenere una stima intervallare.

Esempio

Si suppone che vi siano 11034 persone trattate con **placebo** e 11037 trattate con aspirina (campioni indipendenti) e che 104 pazienti hanno avuto l'infarto tra quelli trattati con aspirina mentre 189 hanno avuto l'infarto tra quelli trattati con placebo. Si considerano le variabili casuali X e Y riferite all'evento **aspirina** (X) e **infarto** (Y).

La stima di massima verosimiglianza del rischio relativo RR per coloro che sono trattati con placebo è

$\hat{p}_1 = 189/11034 = 0.0171$

mentre l'altra proporzione per i trattati con aspirina è $\hat{p}_0 = 104/11037 = 0.009$.

Da cui il rischio relativo

$$\hat{RR} = \frac{0.0171}{0.009} = 1.81$$

risulta superiore a 1: c'è un'associazione tra il placebo e l'infarto, infatti in base a questa stima puntuale, la probabilità di avere l'infarto è circa 2 volte superiore per il gruppo dei controlli (ovvero i pazienti trattati con placebo) rispetto al gruppo dei trattati (con aspirina).

Nel seguente codice si crea il data set con le osservazioni:

```
X <- rep(c("aspirina","placebo"), c(11037,11034))
head(X)
#> [1] "aspirina" "aspirina" "aspirina" "aspirina" "aspirina" "aspirina"
table(X)
#> X
#> aspirina  placebo
#>    11037    11034
```

La tabella si ottiene con la funzione `table` che restituisce la tabella con le frequenze assolute.

```
Y = rep(c("NO", "SI", "SI", "NO"),
          c(11037-104, 104, 189,11034-189))
head(Y)
#> [1] "NO" "NO" "NO" "NO" "NO" "NO"
table(Y)
#> Y
#>    NO    SI
#> 21778   293
```

Si costruisce il dataframe

```
dataR <- data.frame(X,Y)
head(dataR)
```

X	Y
aspirina	NO
aspirina	NO
aspirina	NO
aspirina	NO
aspirina	NO
aspirina	NO

```
table(dataR)
#>           Y
#> X             NO    SI
#>   aspirina 10933   104
#>   placebo  10845   189
```

Le stime delle due proporzioni possono essere ricavate attraverso la tabella di contingenza in cui si dividono le frequenze assolute per il valore marginale di riga (1)

```
CC <- prop.table(table(dataR),1); CC
#>           Y
#> X                   NO         SI
#>   aspirina 0.99057715 0.00942285
#>   placebo  0.98287113 0.01712887
```

La stima del rischio relativo si ottiene come rapporto tra le due frequenze relative corrispondenti

```
RR <- CC[4]/CC[3]; RR
#> [1] 1.817802
```

Una misura di accuratezza per questa stima si ottiene con il **bootstrap**. Occorre campionare dai dati di partenza ogni volta la stessa unità per entrambe le colonne. Utilizzando il ciclo iterativo `for`

```
B <- 2000
RRB <- rep(0,B)
n <- dim(dataR)[1]
set.seed(1023)
for(i in 1:B){
  ind <-sample(1:n,
               size = n,
               replace = TRUE)
  datB <- dataR[ind,]
  CC <- prop.table(table(datB),1)
  RR <- CC[4]/CC[3]
  RRB[i]<-RR
}
head(RRB)
#> [1] 1.985886 1.871075 1.628552 1.933696 1.563510 1.854757
```

Il vettore RRB contiene i valori realizzati dello stimatore RR per ogniuna delle 2000 replicazioni bootstrap.

I valori sono sempre maggiori di 1 e il minimo è 1.3. Si noti che nelle replicazioni il valore stimato assume un massimo pari a circa 2.8 e che la distribuzione bootstrap è leggermente asimmetrica.

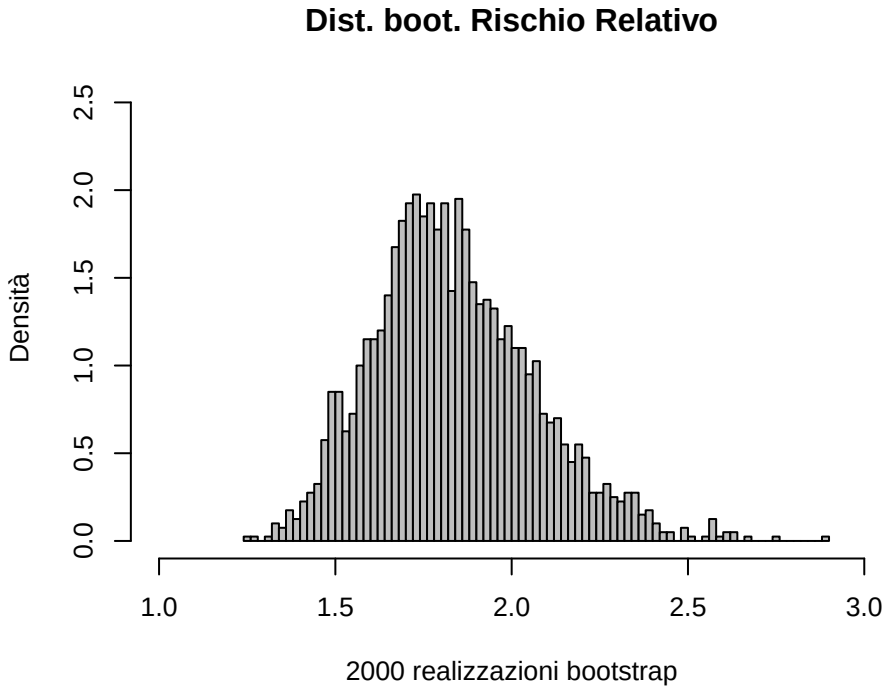
```
summary(RRB)
#>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#>   1.250   1.679   1.808   1.836   1.978   2.888
```

L' errore standard associato alla stima è 0.23.

```
sd(RRB)
#> [1] 0.2281224
```

Si disegna la **distribuzione bootstrap** attraverso l'istogramma

```
sB<- mean(RRB)
hist(RRB,
      breaks = 60,
      freq=FALSE,
      main = "Dist. boot. Rischio Relativo",
      xlab = "2000 realizzazioni bootstrap",
     ylim = c(0,2.5),
      col = "gray",
       ylab = " Densità",
     xlim = c(1, 3))
```



Metodo del percentile

Si calcola l'intervallo di confidenza al 95% con il **metodo del percentile**.

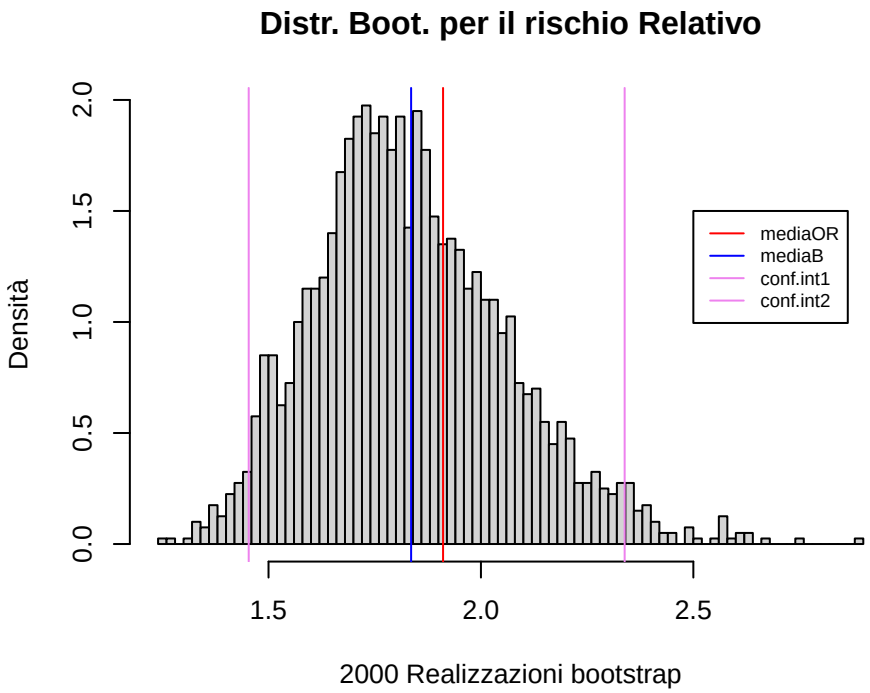
Utilizzando il metodo del percentile l'intervallo di confidenza è il 95% centrale nella distribuzione delle replicazioni bootstrap. Ovvero le stime plausibili sono quelle che ricadono tra il 2.5-esimo percentile ed il 97.5-esimo percentile della distribuzione bootstrap.

```
Q <- quantile(RRB, c(0.025,0.975)); Q
#>     2.5%    97.5%
#> 1.453375 2.338311
```

Si rappresenta graficamente la distribuzione dello stimatore rischio relativo ottenuta con il bootstrap

```
Tm <- mean(RRB); Tm
#> [1] 1.835642
hist(RRB,
     main = "Distr. Boot. per il rischio Relativo",
     breaks=60,
     freq=FALSE,
     ylab="Densità",
     xlab="2000 Realizzazioni bootstrap")
abline(v=c(RR, Tm, Q[1], Q[2]),
```

```
       col=c("red","blue", "violet","violet"))
legend(2.5, 1.5,
       c("mediaOR","mediaB", "conf.int1", "conf.int2"),
       col=c("red","blue", "violet", "violet"),
       lty=c(1,1,1,1),
       cex=0.7)
```



Si nota la **distorsione** in questo caso è data dalla distanza tra media delle replicazioni bootstrap e la stima del rischio relativo (RR) calcolata nei dati di origine.

Metodo Bca

Essendo grande la distanza tra la media osservata nei dati di origine e quella nelle replicazioni bootstrap ed essendo presente asimmetria nella distribuzione bootstrap è opportuno considerare anche l'intervallo ottenuto con il metodo BCa.

Per applicare la funzione `bootstrap::bcanon` occorre implementare la funzione `thetaR` per il calcolo del rischio relativo nel modo seguente

```
thetaR <- function(ind){
  datB <- dataR[ind,]
  CC <- prop.table(table(datB),1)
  CC[4]/CC[3]
}
```

Per ottenere un intervallo a livello di confidenza $1 - \alpha = 0.95$ si applica la funzione `bcanon`

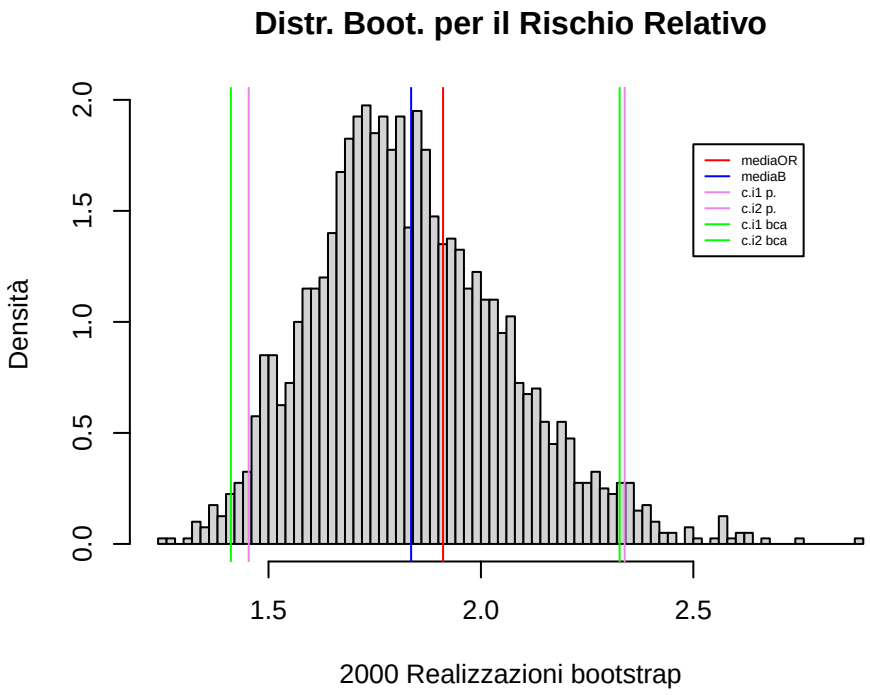
```
require(bootstrap)
set.seed(1023)
CIBca <- bcanon(1:n,
                B,
                thetaR,
                alpha=c(0.025,0.975))
CIBca$confpoints
#>      alpha bca point
#> [1,] 0.025  1.411361
#> [2,] 0.975  2.326449
CIBca$acc
#> [1] -0.00603286
CIBca$z0
#> [1] -0.01504034
```

Con una fiducia di 0.95 affermiamo che la probabilità di avere l'infarto per coloro che non sono stati trattati è superiore di un valore compreso tra 1.4 e 2.3 rispetto a quello di coloro che sono stati trattati con aspirina.

Aggiungiamo gli estremi Bca al grafico precedente

```
Tm <- mean(RRB); Tm
#> [1] 1.835642
hist(RRB,
     main = "Distr. Boot. per il Rischio Relativo",
     breaks=60,
     freq=FALSE,
     ylab="Densità",
     xlab="2000 Realizzazioni bootstrap")
abline(v=c(RR, Tm, Q[1], Q[2],
           CIBca[["confpoints"]][3],
           CIBca[["confpoints"]][4]),
       col=c("red","blue", "violet","violet", "green", "green"))
legend(2.5, 1.8,
       c("mediaOR",
         "mediaB",
         "c.i1 p.",
         "c.i2 p.",
         "c.i1 bca",
         "c.i2 bca"),
       col=c("red","blue", "violet", "violet", "green", "green"),
```

```
       lty=c(1,1,1,1),
       cex=0.5)
```



Si noti che l'intervallo bca è leggermente traslato a sinistra rispetto a quello ottenuto con il metodo del percentile.