RecapCap1

2023-12-27

Apprendimento statistico - MASTRANTONIO

METODI SIMULATIVI

Si scrivono di seguito paccehetti utili per svolgere i calcoli su R.

```
library(datasets)
library(catdata)
```

```
## Caricamento del pacchetto richiesto: MASS
```

```
library(dslabs)
```

```
## Warning: il pacchetto 'dslabs' è stato creato con R versione 4.2.3
```

```
library(mvtnorm)
library(patchwork)
```

```
##
## Caricamento pacchetto: 'patchwork'
```

```
## Il seguente oggetto è mascherato da 'package:MASS':
##
##     area
```

```
library(paletteer)
library(tidyverse)
```

```
## Warning: il pacchetto 'tidyverse' è stato creato con R versione 4.2.3
```

```
## Warning: il pacchetto 'ggplot2' è stato creato con R versione 4.2.3
```

```
## Warning: il pacchetto 'tibble' è stato creato con R versione 4.2.3
```

```
## Warning: il pacchetto 'tidyr' è stato creato con R versione 4.2.3
```

```
## Warning: il pacchetto 'readr' è stato creato con R versione 4.2.3
```

```
## Warning: il pacchetto 'purrr' è stato creato con R versione 4.2.3
```

```
## Warning: il pacchetto 'dplyr' è stato creato con R versione 4.2.3
```

```
## Warning: il pacchetto 'forcats' è stato creato con R versione 4.2.3
```

```
## Warning: il pacchetto 'lubridate' è stato creato con R versione 4.2.3
```

```
## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
## ✓ dplyr     1.1.4     ✓ readr     2.1.4
## ✓ forcats   1.0.0     ✓ stringr   1.5.0
## ✓ ggplot2   3.4.4     ✓ tibble    3.2.1
## ✓ lubridate 1.9.3     ✓ tidyr     1.3.0
## ✓ purrr     1.0.2
```

```
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
## ✖ dplyr::select() masks MASS::select()
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to be
come errors
```

NUMERI PSEUDO CASUALI: Simulazione da uniformi - seed

Dall'esempio sottostante si può notare che nei primi due casi in cui si setta il seme, *seed*, i numeri "casuali" generati risultano essere gli stessi. Invece, nella terza prova i risultati sono differenti.

```
set.seed(100) # Setta il seme
s1 = runif(10, 0, 1) # Simulo un campione di 10 elementi dalla v.a. uniforme U(0,1)
set.seed(100) # Setta lo stesso seme
s2 = runif(10, 0, 1) # Simulo nuovamente un campione dalla v.a. uniforme
```

```
s3 = runif(10, 0, 1) # Simulo un campione dalla v.a. uniforme
```

```
t(s1) # t() traspone il vettore s1
```

```
##           [,1]      [,2]      [,3]       [,4]      [,5]      [,6]      [,7]
## [1,] 0.3077661 0.2576725 0.5523224 0.05638315 0.4685493 0.4837707 0.8124026
##           [,8]      [,9]     [,10]
## [1,] 0.3703205 0.5465586 0.1702621
```

```
t(s2)
```

```
##           [,1]      [,2]      [,3]       [,4]      [,5]      [,6]      [,7]
## [1,] 0.3077661 0.2576725 0.5523224 0.05638315 0.4685493 0.4837707 0.8124026
##           [,8]      [,9]     [,10]
## [1,] 0.3703205 0.5465586 0.1702621
```

```
t(s3)
```

```
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.6249965 0.8821655 0.2803538 0.3984879 0.7625511 0.6690217 0.2046122
##           [,8]      [,9]     [,10]
## [1,] 0.3575249 0.3594751 0.6902905
```

SIMULAZIONE DA UNA UNIFORME

Simuliamo da una $U(0, 1)$, con seme 10.

```
set.seed(10)
x = runif(3, 0, 1)
x
```

```
## [1] 0.5074782 0.3067685 0.4269077
```

```
set.seed(10)
runif(3, 0, 1)
```

```
## [1] 0.5074782 0.3067685 0.4269077
```

Implementiamo la funzione per la simulazione di un uniforme $U(0, 1)$.

Il codice definisce una funzione chiamata **r_unif01** che implementa un generatore di numeri pseudo-casuali uniformemente distribuiti tra 0 e 1 utilizzando un metodo lineare congruenziale (LCG).

```
# n è il numero di numeri da generare
# m è il modulo
# c è l'incremento
# a è il moltiplicatore
r_unif01 = function(n, seed, m = 100, c = 2, a = 4){
    ret = rep(NA,n) # Crea un vettore per memorizzare i numeri generati
    val_prec = seed
    for(i in 1:n){
      # Utilizza la formula LCG per generale i numeri
      ret[i] = (a * val_prec + c) %% m
      val_prec = ret[i]
    }
    # Normalizza i numeri
    ret = ret/m
    return(ret)
}
```

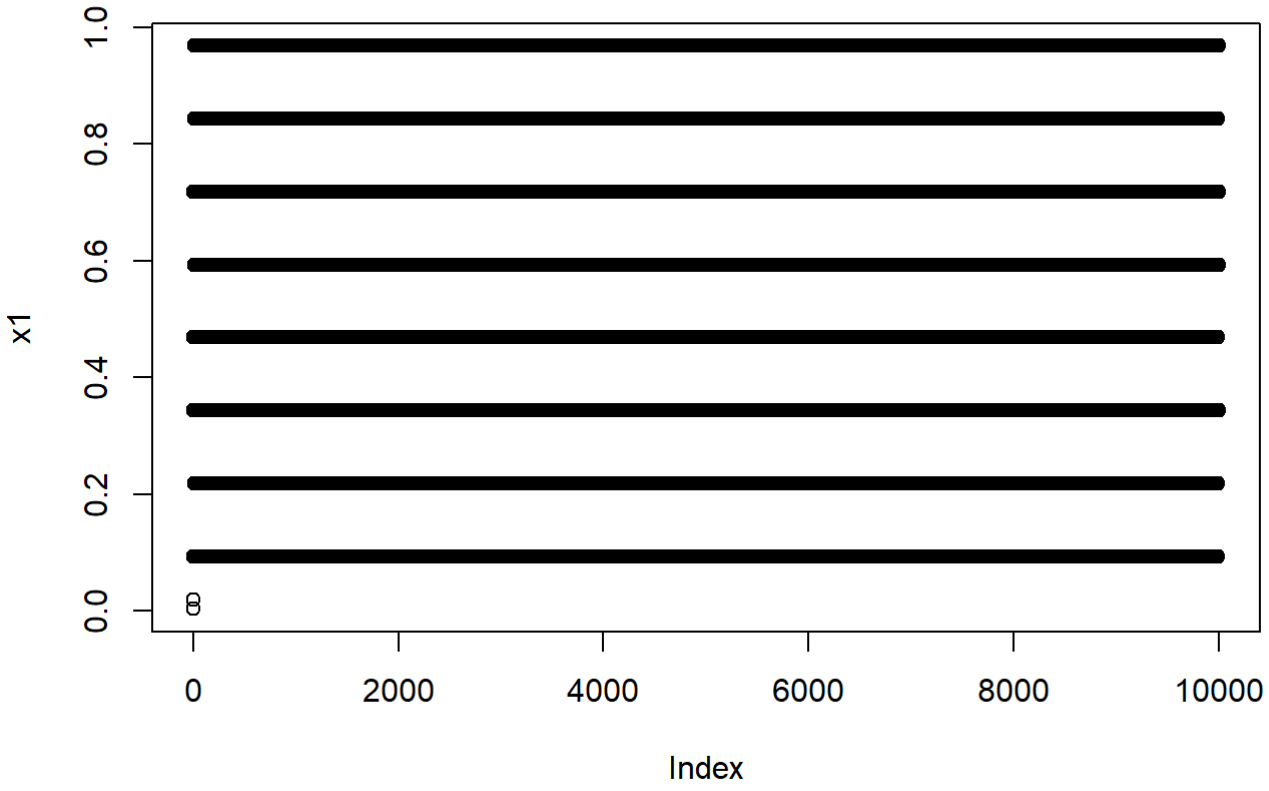
In sintesi, la funzione **r_unif01** restituisce un vettore di lunghezza n contenente numeri pseudo-casuali uniformemente distribuiti tra 0 e 1, generati utilizzando un LCG con i parametri specificati o con valori predefiniti se non specificati.

Facciamo dei test.

```
x1 = r_unif01(n = 10000, seed = 0, m = 1000, c = 3, a = 5 ) # Si genera x1 con la funzione de
finita precedentemente
x1[1:25] # Stampati i primi 25 elementi del vettore x1
```

```
##  [1] 0.003 0.018 0.093 0.468 0.343 0.718 0.593 0.968 0.843 0.218 0.093 0.468
## [13] 0.343 0.718 0.593 0.968 0.843 0.218 0.093 0.468 0.343 0.718 0.593 0.968
## [25] 0.843
```

```
plot(x1)
```



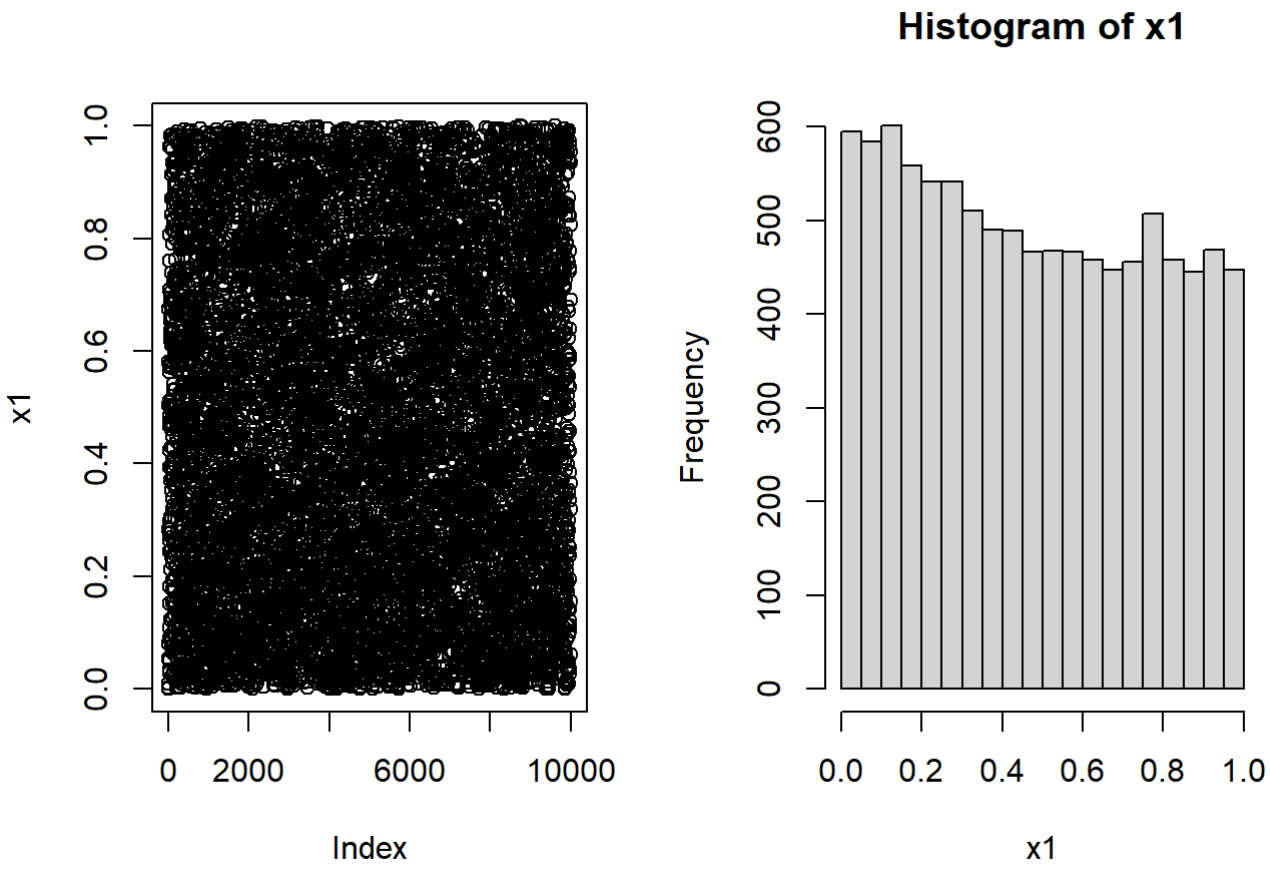
Il grafico rappresenta la distribuzione dei 10000 numeri pseudo-casuali generati dal tuo generatore lineare congruenziale con i parametri specificati.

Si vede facilmente che è la simulazione entra in loop, continuando a generare sempre gli stessi valori. Proviamo ad ovviare il problema.

```
x1 = r_unif01(n = 10000, seed = 0, m = 10000000, c=3.7, a =5.3 )
x1[1:25]
```

```
##  [1] 3.700000e-07 2.331000e-06 1.272430e-05 6.780879e-05 3.597566e-04
##  [6] 1.907080e-03 1.010789e-02 5.357221e-02 2.839331e-01 5.048456e-01
## [11] 6.756820e-01 5.811149e-01 7.990951e-02 4.235208e-01 2.446604e-01
## [16] 2.967003e-01 5.725120e-01 3.431392e-02 1.818642e-01 9.638804e-01
## [21] 1.085665e-01 5.754026e-01 4.963417e-02 2.630615e-01 3.942261e-01
```

```
par(mfrow = c(1,2))
plot(x1) # Mostra la distribuzione dei numeri pseudo-casuali, si visualizza la sequenza nel suo insieme
hist(x1) # Istrogramma che mostra la distribuzione dei numeri pseudo-casuali in bin (in un intervallo specifico in cui i dati vengono raggruppati)
```



Il risultato è una rappresentazione visuale della distribuzione dei numeri pseudo-casuali generati con i parametri specificati.

UN ESEMPIO DI GENERATORE PSEUDO CASUALE

Otteniamo un generatore pseudo casuale $U(0, 1)$ creando le variabili

$$d_i = (ad_{i-1} + c)\ mod\ m$$

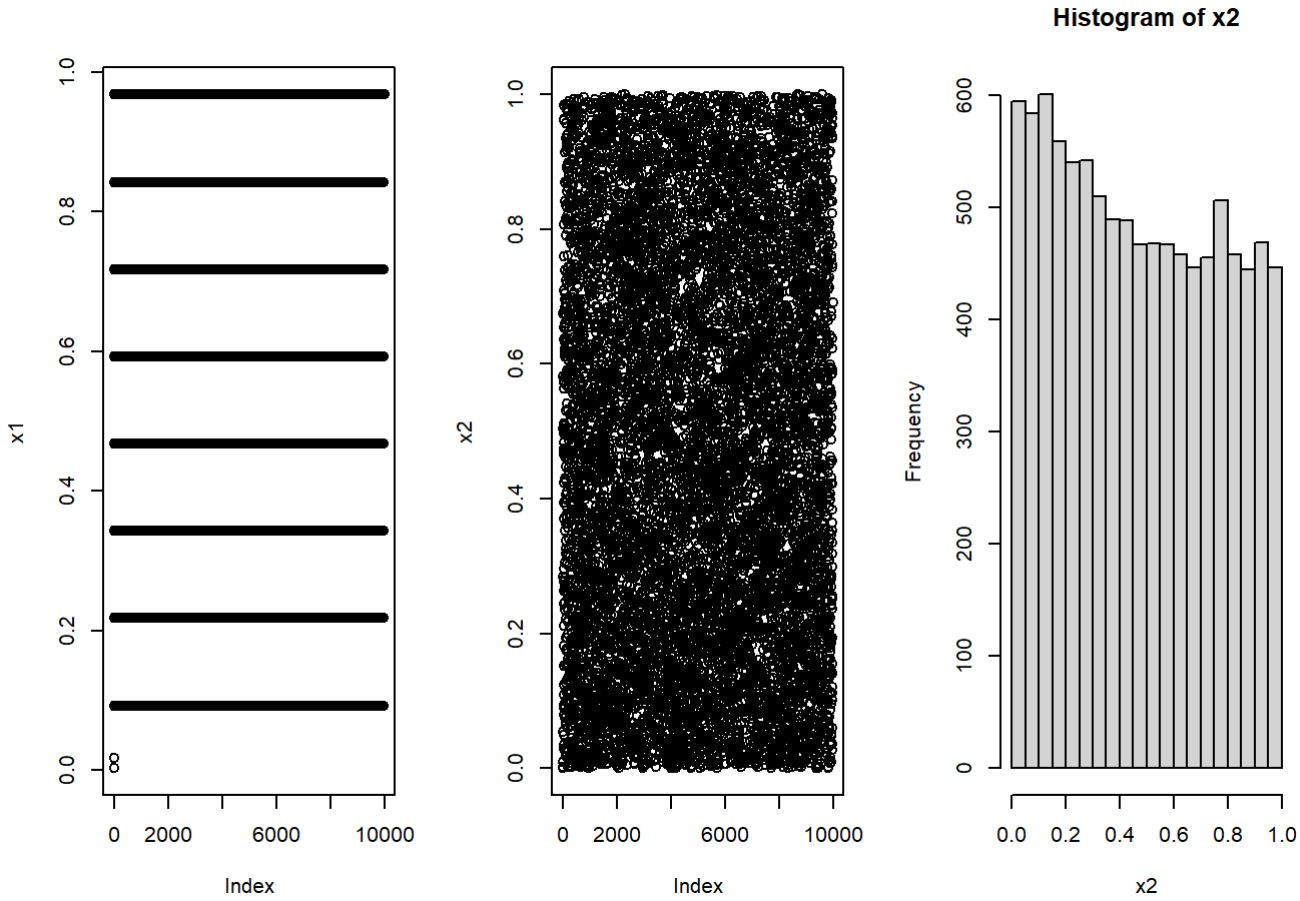
dove $0 \leq c \leq m$ è l'incremento e $0 < a < m$ è il moltiplicatore. Dopo si prende $u_i = \frac{d_i}{m}$.

Una volta implementata la funzione **gen_pseudo_casuale_uniforme** facciamo due test:

1. Inseriamo valori interi e plottiamo: entra in un loop e genera sempre gli stessi numeri.
2. Inseriamo valori non interi e plottiamo: si può vedere che tutti i numeri generati sono differenti.

```r
# Definizione della funzione
gen_pseudo_U = function(n, seed, m = 100, c = 2, a = 4){
    u_i = rep(NA,n) # Replica il valore (NA) per n volte creando un array
    val_prec = seed # Alla 1° iter. setto il valore prec. come il seme
    for(i in 1:n){
        # Applico la formula con %% il modulo
        u_i[i] = (a * val_prec + c) %% m
        val_prec = u_i[i] # Aggiorno il d_i-1
    }
    u_i = u_i / m # Applica la formula per prendere gli "u_i"
    return(u_i)
}

# Chiamo la funzione per vedere il risultato
x = gen_pseudo_U(n = 10000, seed = 0, m = 1000, c = 3, a = 5)
# x per stampare il risultato
```

```r
# "Test"
# Generatore con interi che entra in loop
x1 = gen_pseudo_U(n = 10000, seed = 0, m = 1000, c = 3, a = 5 )
# Generatore che non entra in loop
x2 = gen_pseudo_U(n = 10000, seed = 0, m = 10000000, c = 3.7, a = 5.3 )
# per il momento lascio così, imparerò a farli più piccoli
par(mfrow=c(1,3))
plot(x1)
plot(x2)
hist(x2)
```



Marginali e congiunte: SIMULO DUE VARIABILI CHE POSSONO ASSUMERE VALORI IN $[1, 5]$.

```
K = 5
probmat = matrix(runif(K^2, 0,1), nrow=K, ncol=K)
# Matrice di probabilità 5x5
probmat = probmat/sum(probmat)
probmat
```

```
##             [,1]        [,2]        [,3]       [,4]       [,5]
## [1,] 0.059423198 0.052798207 0.051091735 0.03419038 0.03048473
## [2,] 0.007299144 0.036837945 0.030697462 0.07168607 0.03479560
## [3,] 0.019327838 0.055869785 0.036764032 0.07413699 0.06058447
## [4,] 0.023536911 0.048675071 0.004449938 0.05275732 0.07187070
## [5,] 0.023346111 0.009731708 0.022649307 0.06645415 0.02054121
```

```
# Simuliamo nsim simulazioni
nsim = 10000
# rowSums(probmat) somma tutti i valori di probabilità sulle righe
# Otteniamo così nsim campioni da x1
x1 = sample(1:K, nsim, prob = rowSums(probmat), replace=T)

# Simuliamo ora dalla condizionata usando la formula di Bayes
# f(x2 = j | x1 = i) = f(x2 = j, x1 = i) / f(x1 = i)
x2 = rep(NA, nsim)
for(i in 1:nsim){
x2[i] = sample(1:K, 1, prob = probmat[x1[i], ]/sum(probmat[x1[i], ]), replace=T)
}
# Affianco in colonna x1 e x2
conti = matrix(0, nrow=K, ncol=K)
# "Conti" è la matrice dei conteggi che mi dice quanto spesso esce una
# determinata combinazione
for(i in 1:nsim){
conti[x1[i], x2[i]] = conti[x1[i], x2[i]] + 1
}
conti/nsim # Matrice 5x5
```

```
##        [,1]   [,2]   [,3]   [,4]   [,5]
## [1,] 0.0593 0.0563 0.0534 0.0362 0.0296
## [2,] 0.0079 0.0374 0.0286 0.0714 0.0335
## [3,] 0.0190 0.0565 0.0365 0.0721 0.0610
## [4,] 0.0249 0.0438 0.0059 0.0509 0.0670
## [5,] 0.0216 0.0083 0.0240 0.0736 0.0213
```

```
probmat # Matrice 5x5
```

```
##             [,1]        [,2]        [,3]       [,4]       [,5]
## [1,] 0.059423198 0.052798207 0.051091735 0.03419038 0.03048473
## [2,] 0.007299144 0.036837945 0.030697462 0.07168607 0.03479560
## [3,] 0.019327838 0.055869785 0.036764032 0.07413699 0.06058447
## [4,] 0.023536911 0.048675071 0.004449938 0.05275732 0.07187070
## [5,] 0.023346111 0.009731708 0.022649307 0.06645415 0.02054121
```

PLOT DENSITA', CUMULATA, QUANTILE E CAMPIONE DALLA NORMALE

Si definisca $X \sim N(\mu, \sigma^2)$. In R la normale è chiamata *"norm"*, inoltre

1. La funzione che calcola la densità si chiama **rnorm()**
2. La funzione che calcola la cumulata è **pnorm()**
3. La funzione che calcola il quantile è **qnorm()**
4. La funzione che estrae un campion è **rnorm()**

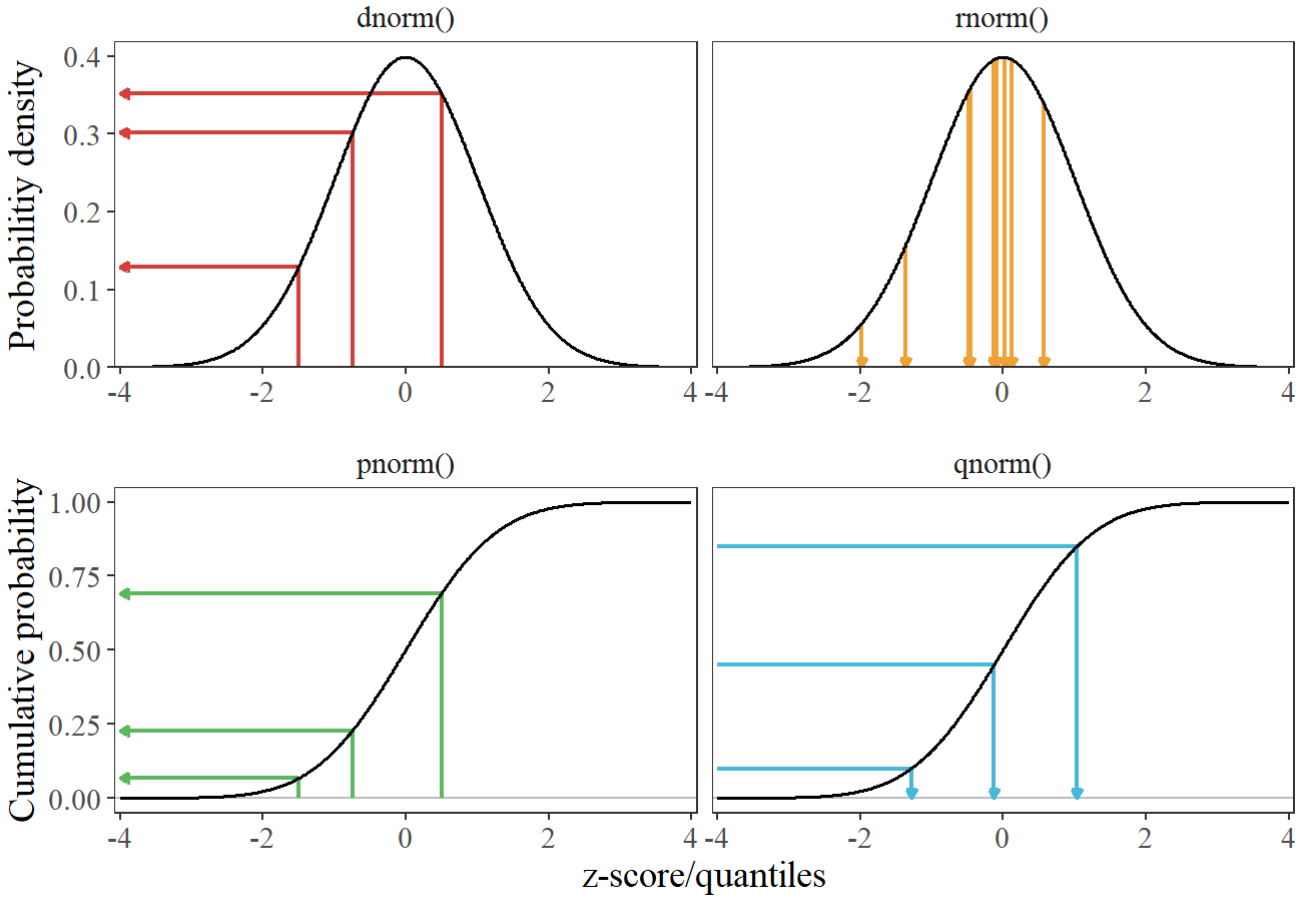
```
z_scores <- seq(-4, 4, by = 0.01)
mu <- 0
sd <- 1
# Functions
# Using `dnorm` and `pnorm` to setup the "skeleton" of related plots.
normal_dists <- list(`dnorm()` = ~ dnorm(., mu, sd),
`rnorm()` = ~ dnorm(., mu, sd),
`pnorm()` = ~ pnorm(., mu, sd),
`qnorm()` = ~ pnorm(., mu, sd))
# Apply functions to data and parameter combinations
df <- tibble(z_scores, mu, sd) %>%
mutate_at(.vars = vars(z_scores), .funs = normal_dists) %>%
# "Lengthen" the data
pivot_longer(cols = -c(z_scores, mu, sd), names_to = "func",
values_to = "prob") %>%
# Categorize based on shape of distribution -- need to split up the dataframe
# for plotting later.
mutate(distribution = ifelse(func == "pnorm()" | func == "qnorm()",
"Cumulative probability", "Probability density"))
# Split up the data into different pieces that can then be added to a plot.
# Probabilitiy density distrubitions
df_pdf <- df %>%
filter(distribution == "Probability density") %>%
rename(`Probabilitiy density` = prob)
# Cumulative density distributions
df_cdf <- df %>%
filter(distribution == "Cumulative probability") %>%
rename(`Cumulative probability` = prob)
# dnorm segments
# Need to make lines that represent examples of how values are mapped -- there
# is probably a better way to do this, but quick and dirty is fine for now.
df_dnorm <- tibble(z_start.line_1 = c(-1.5, -0.75, 0.5),
pd_start.line_1 = 0) %>%
mutate(z_end.line_1 = z_start.line_1,
pd_end.line_1 = dnorm(z_end.line_1, mu, sd),
z_start.line_2 = z_end.line_1,
pd_start.line_2 = pd_end.line_1,
z_end.line_2 = min(z_scores),
pd_end.line_2 = pd_start.line_2,
id = 1:n()) %>%
pivot_longer(-id) %>%
separate(name, into = c("source", "line"), sep = "\\.") %>%
pivot_wider(id_cols = c(id, line), names_from = source) %>%
mutate(func = "dnorm()",
size = ifelse(line == "line_1", 0, 0.03))
# rnorm segments
# Make it reproducible
set.seed(20200209)
df_rnorm <- tibble(z_start = rnorm(10, mu, sd)) %>%
mutate(pd_start = dnorm(z_start, mu, sd),
z_end = z_start,
pd_end = 0,
func = "rnorm()")
# pnorm segments
df_pnorm <- tibble(z_start.line_1 = c(-1.5, -0.75, 0.5),
```

```
pd_start.line_1 = 0) %>%
mutate(z_end.line_1 = z_start.line_1,
pd_end.line_1 = pnorm(z_end.line_1, mu, sd),
z_start.line_2 = z_end.line_1,
pd_start.line_2 = pd_end.line_1,
z_end.line_2 = min(z_scores),
pd_end.line_2 = pd_start.line_2,
id = 1:n()) %>%
pivot_longer(-id) %>%
separate(name, into = c("source", "line"), sep = "\\.") %>%
pivot_wider(id_cols = c(id, line), names_from = source) %>%
mutate(func = "pnorm()",
size = ifelse(line == "line_1", 0, 0.03))
# qnorm segments
df_qnorm <- tibble(z_start.line_1 = min(z_scores),
pd_start.line_1 = c(0.1, 0.45, 0.85)) %>%
mutate(z_end.line_1 = qnorm(pd_start.line_1),
pd_end.line_1 = pd_start.line_1,
z_start.line_2 = z_end.line_1,
pd_start.line_2 = pd_end.line_1,
z_end.line_2 = z_end.line_1,
pd_end.line_2 = 0,
id = 1:n()) %>%
pivot_longer(-id) %>%
separate(name, into = c("source", "line"), sep = "\\.") %>%
pivot_wider(id_cols = c(id, line), names_from = source) %>%
mutate(func = "qnorm()",
size = ifelse(line == "line_1", 0, 0.03))
cp <- paletteer_d("ggsci::default_locuszoom", 4, )
names(cp) <- c("dnorm()", "rnorm()", "pnorm()", "qnorm()")
# Probabilitiy density
p_pdf <- df_pdf %>%
ggplot(aes(z_scores, `Probabilitiy density`)) +
geom_segment(data = df_dnorm,
aes(z_start, pd_start, xend = z_end, yend = pd_end),
arrow = arrow(length = unit(df_dnorm$size, "npc"), type = "closed"),
size = 0.8, color = cp["dnorm()"]) +
geom_segment(data = df_rnorm,
aes(z_start, pd_start, xend = z_end, yend = pd_end),
arrow = arrow(length = unit(0.03, "npc"), type = "closed"),
size = 0.8, color = cp["rnorm()"]) +
geom_line(size = 0.6) +
facet_wrap(~ func, nrow = 1) +
theme_bw() +
theme(panel.grid = element_blank(),
axis.title.x = element_blank(),
strip.background = element_blank(),
text = element_text(family = "serif", size = 14)) +
scale_y_continuous(expand = expand_scale(c(0, 0.05))) +
scale_x_continuous(expand = c(0.01, 0))
```

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## ℹ Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```
## Warning: `expand_scale()` was deprecated in ggplot2 3.3.0.
## ℹ Please use `expansion()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

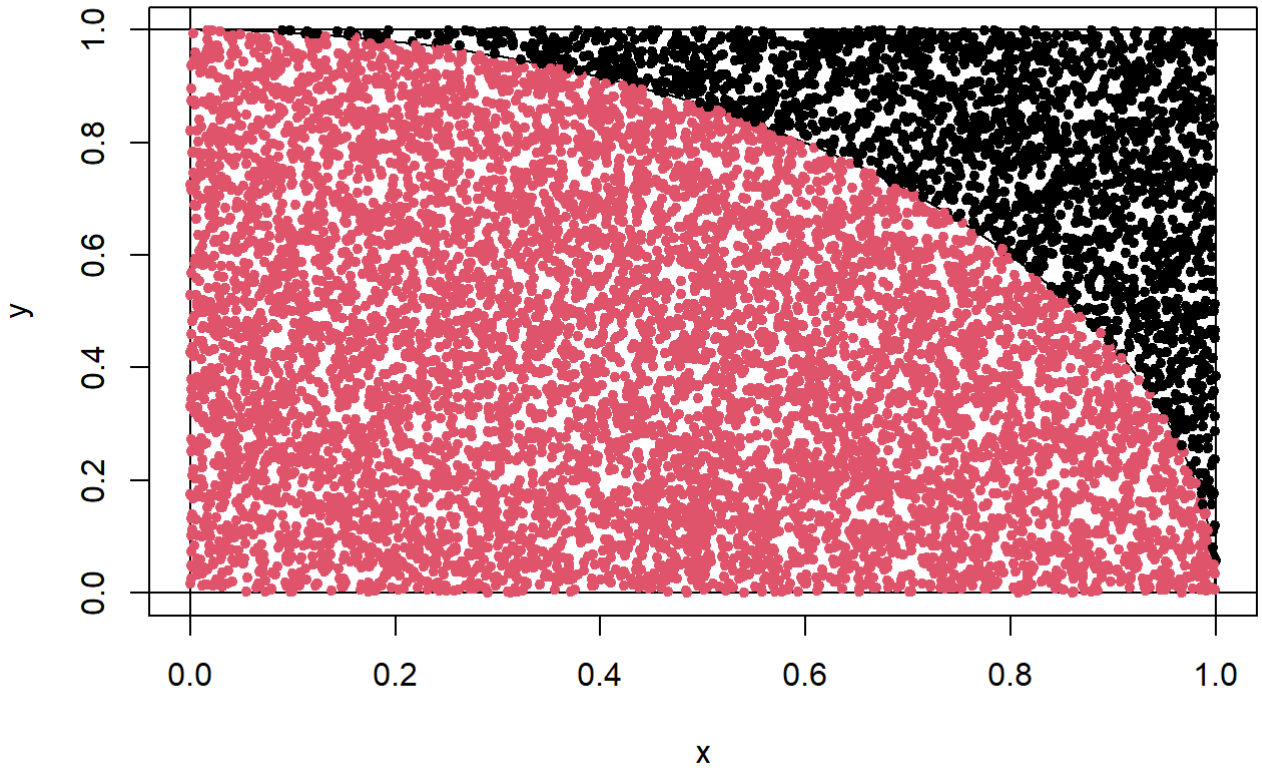
```
# Cumulative probability
p_cdf <- df_cdf %>%
ggplot(aes(z_scores, `Cumulative probability`)) +
geom_hline(yintercept = 0, color = "grey") +
geom_segment(data = df_pnorm,
aes(z_start, pd_start, xend = z_end, yend = pd_end),
arrow = arrow(length = unit(df_dnorm$size, "npc"), type = "closed"),
size = 0.8, color = cp["pnorm()"]) +
geom_segment(data = df_qnorm,
aes(z_start, pd_start, xend = z_end, yend = pd_end),
arrow = arrow(length = unit(df_qnorm$size, "npc"), type = "closed"),
size = 0.8, color = cp["qnorm()"]) +
geom_line(size = 0.6) +
facet_wrap(~ func, nrow = 1) +
labs(x = "z-score/quantiles") +
theme_bw() +
theme(panel.grid = element_blank(),
strip.background = element_blank(),
text = element_text(family = "serif", size = 14)) +
scale_x_continuous(expand = c(0.01, 0))
# Combine the plots
p_pdf + p_cdf + plot_layout(ncol = 1)
```



Stessa cosa vale per altre distribuzioni, per esempio la Poisson è dpois(), ppois(), dpois() e rpois(). Si chiama dpois anche se in realtà calcola la probabilità e non la densità.

CALCOLO π

```
x <- seq(0,1, by = 0.01)
y <- sqrt(1-x^2) # Calcola la circonferenza nel primo quadrante
plot(x,y, type="l")
# Creo una scatola [0,1]x[0,1]
abline(v = 0)
abline(v = 1)
abline(h = 1)
abline(h = 0)
nsim <- 10000
coord1 <- runif(nsim, 0,1)
coord2 <- runif(nsim, 0,1)
points(coord1,coord2, pch = 20, col = c(1,2)[ (coord1^2+ coord2^2 <= 1) +1])
```



Avendo tanti campioni casuali, mi aspetto che la probabilità di selezionare un punto e vedere che è sotto la curva (cioè nella zona rossa) è la proporzione di quanta area è presenta nella zona rossa e quanta nella zona nera.

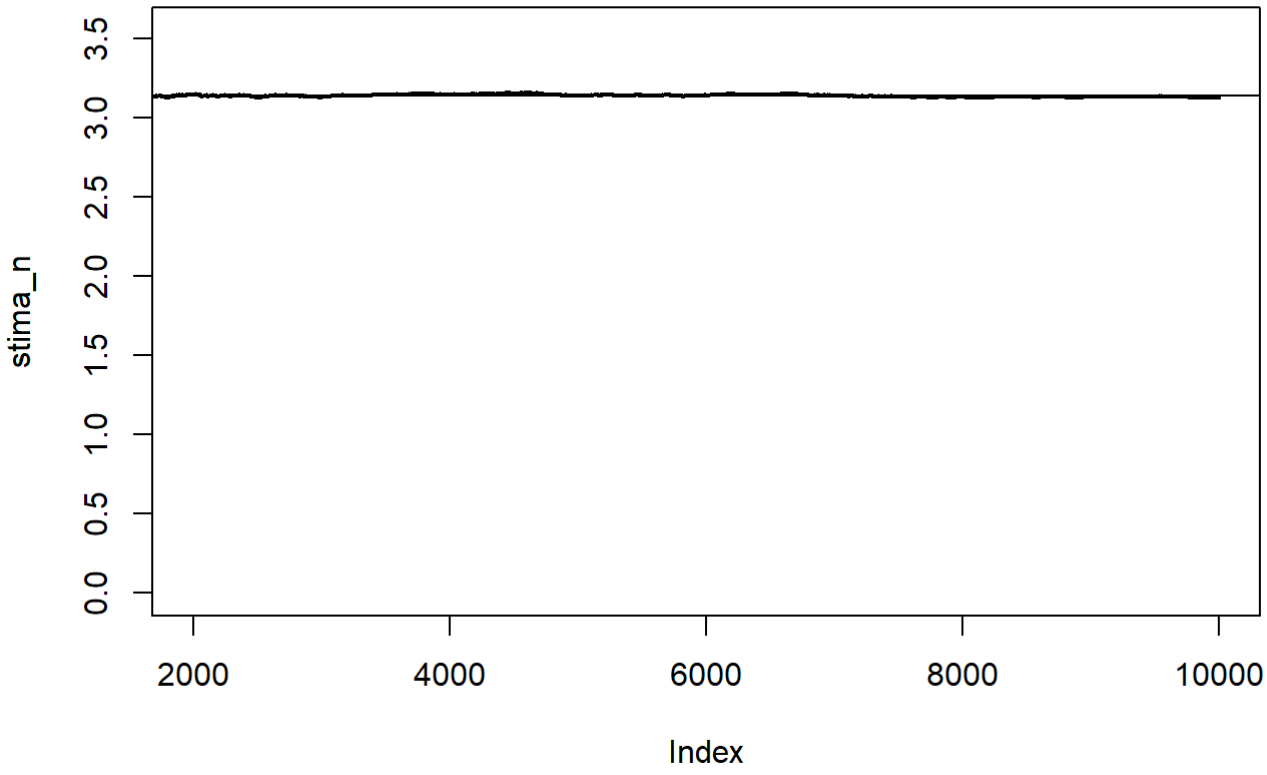
Contiamo i punti dentro al cerchio.

```
punti_in_cerchio <- sum( (coord1^2+ coord2^2 <= 1))/nsim*4 # Sommo tutti gli "1" che sono pre
senti e divido per i punti che ho
punti_in_cerchio
```

```
## [1] 3.128
```

L'obbiettivo è calcolare π approssimando il numero di punti dentro la circonferenza. Aumentando le iterazioni il valore di π migliorerà sempre di più, infatti per la legge dei grandi numeri la media dell'integrale tende all'area dell'oggetto studiato.

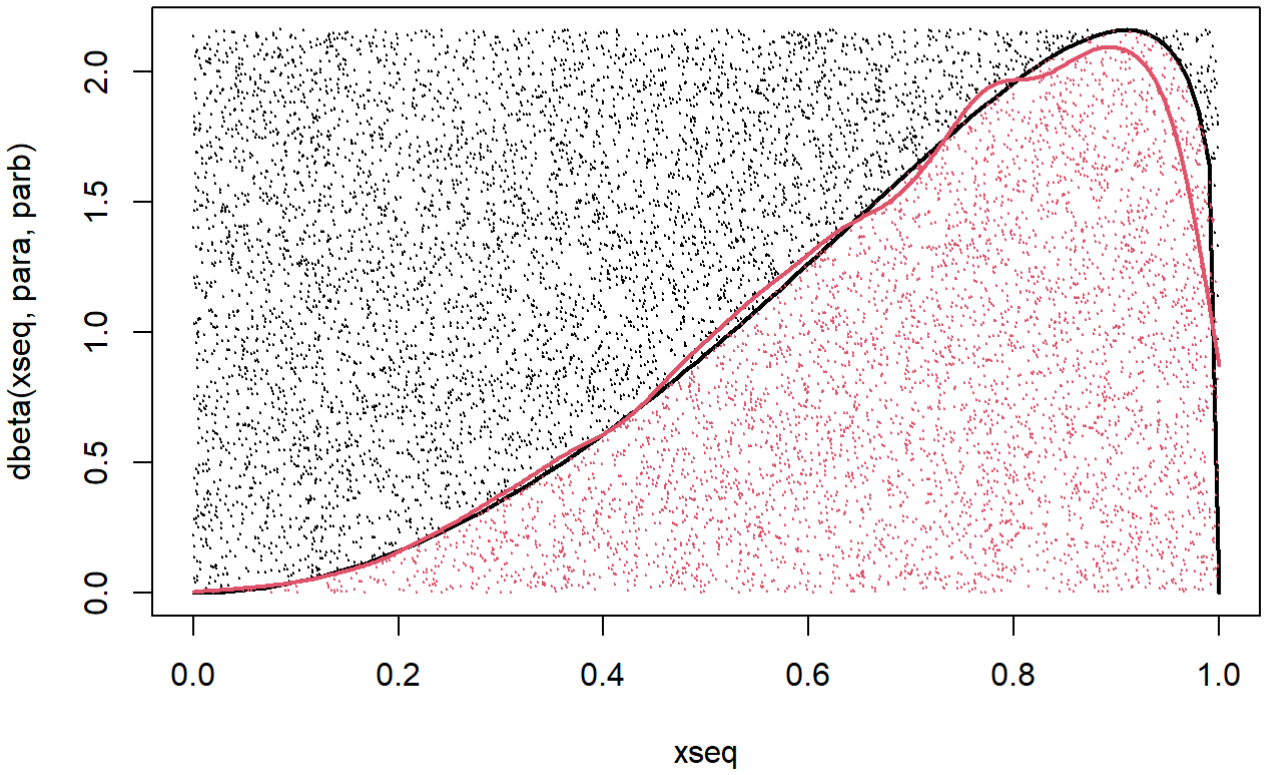
```
stima_n <- rep(NA, nsim)
stima_n[1] <- 0

for(i in 2:nsim){
    stima_n[i] <- sum( (coord1[1:i]^2 + coord2[1:i]^2 <= 1))/i*4
}
plot(stima_n, lwd = 2, type = "l", xlim = c(2000, 10000))
abline(h = pi)
```



ESEMPIO DI ACCEPT-REJECT

Questo codice esegue la simulazione di campioni da una distribuzione beta e ne visualizza i risultati attraverso un grafico. Cioè il codice mostra la generazione e la visualizzazione di campioni da una distribuzione beta e fornisce un confronto visivo tra la distribuzione originale e i campioni generati.

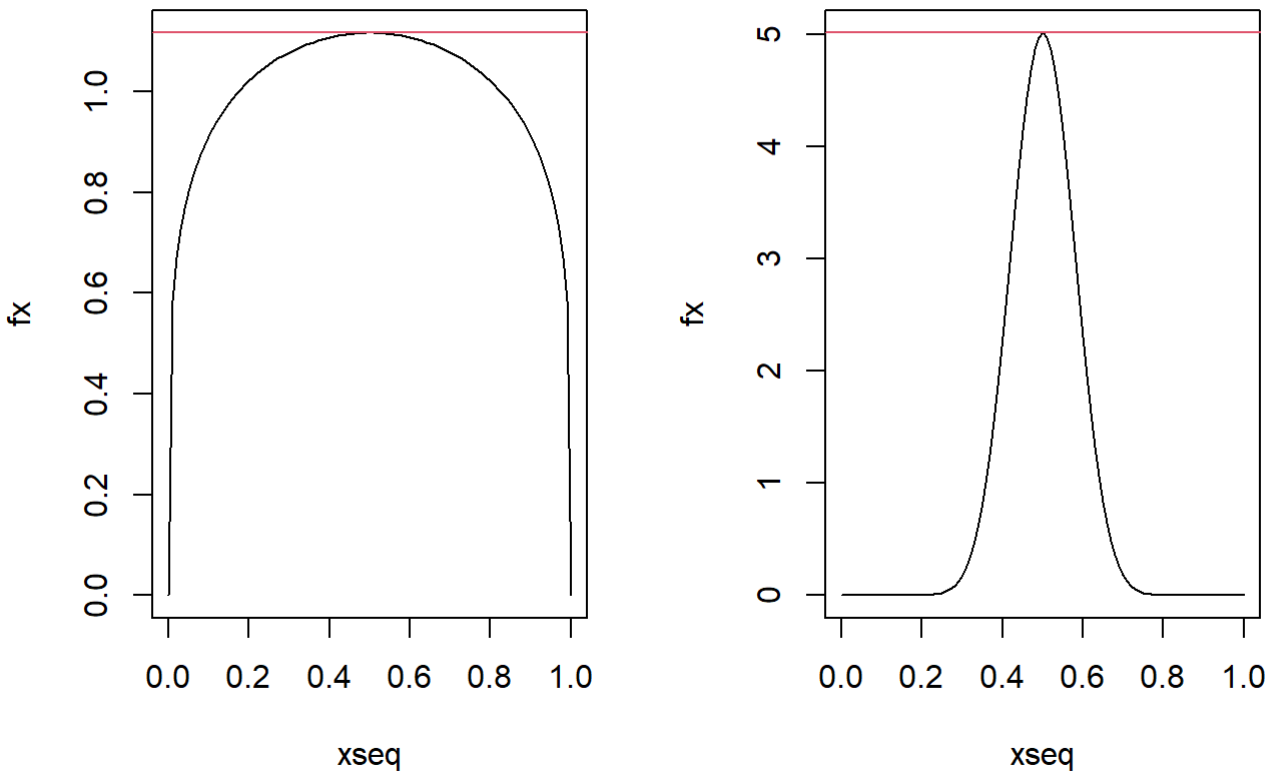
```
# parametri della distribuzione beta
para <- 3
parb <- 1.2

# calcoliamo il massimo
moda <- (para - 1) / (para + parb - 2) # moda della distribuzione
m <- dbeta (moda, para, parb) # densità in corrispondenza della moda

# generazione di due vettori casuali Y e U con 10000 campioni ciascuno
# da una distribuzione uniforme tra 0 e 1
n <- 10000 # numero di campioni
Y <- runif (n, 0, 1)
U <- runif (n, 0, m)

# Viene filtrato il vettore Y mantenendo solo i valori per i quali
# U è minore della densità di probabilità calcolata in precedenza
# Vengono creati vettori U_X e X contenenti rispettivamente
# i valori di U e X corrispondenti al filtro applicato:
# U < dbeta (Y,para , parb )
X <- Y[U < dbeta(Y, para, parb)]
U_X <- U[U < dbeta(Y, para, parb)]

# grafico della densità di probabilità della distribuzione beta
# vengono evidenziati i punti che soddisfano il filtro con un
# colore diverso e viene tracciata la densità stimata di X
xseq = seq(0, 1, by = 0.01)
plot(xseq , dbeta(xseq, para, parb), ylim = c(0,m),
     type ="l", lwd =2, )
points(Y,U, pch = 20 , cex = 0.1)
points(X,U_X, pch = 20 , cex = 0.1, col = 2)
lines(density(X, from = 0, to = 1), col =2, lwd =2)
```



ESEMPI DI β E DELLE SCATOLE

Si creano due grafici per illustrare la densità di probabilità di due distribuzioni beta diverse. Il codice visualizza graficamente le funzioni di densità di probabilità di due distribuzioni beta diverse e identifica il massimo valore della PDF per ognuna di esse.

```r
# xseq è utilizzato come sequenza di valori per valutare le funzioni di densità
# di probabilità (PDF) delle distribuzioni beta
xseq <- seq(0, 1, by = 0.01)

# fx è la PDF della distribuzione beta con parametri 1.2 e 1.2
# calcolata per la sequenza di valori in xseq
fx <- dbeta(xseq, 1.2, 1.2)

# si calcola il massimo valore della PDF
m <- max(fx)

par(mfrow = c(1,2))
# PDF della distribuzione beta con parametri 1.2 e 1.2
plot(xseq, fx, type = "l")
# la linea orizzontale in rosso rappresenta il massimo valore della PDF
abline(h = m, col = 2)

# PDF della distribuzione beta con parametri 20 e 20
fx = dbeta(xseq, 20, 20)
# il suo massimo
m <- max(fx)

# PDF della distribuzione beta con parametri 20 e 20
# e la linea orizzontale colorata in rosso rappresenta
# il massimo valore della PDF
plot(xseq, fx, type="l")
abline(h = m, col = 2)
```

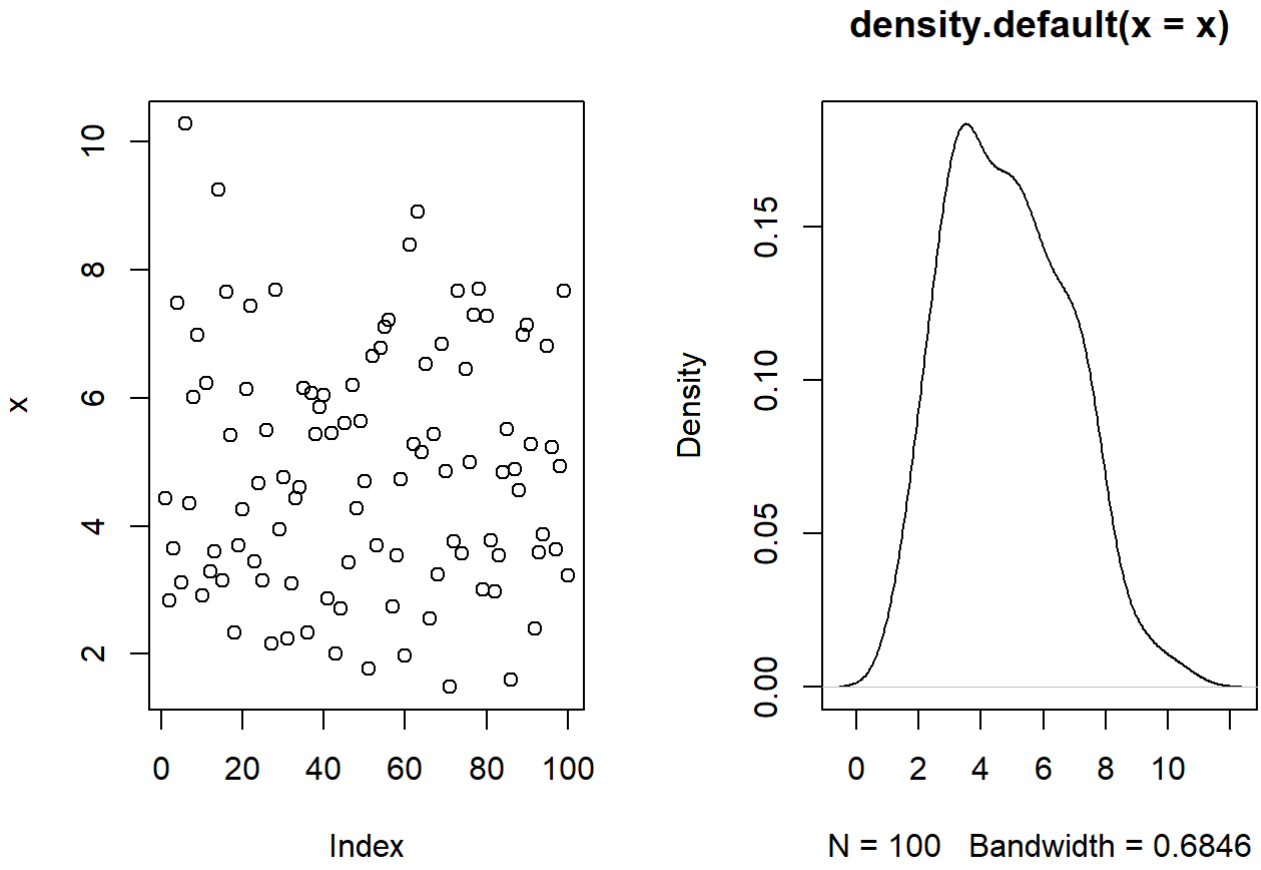


STIMA DI DENSITA': SIMULAZIONI DA $\Gamma(5, 1)$

Alle volte abbiamo dei campioni di cui non conosciamo la densità. Facciamo quindi un esempio.

La funzione **density** dà la stima della densità del campione di input.

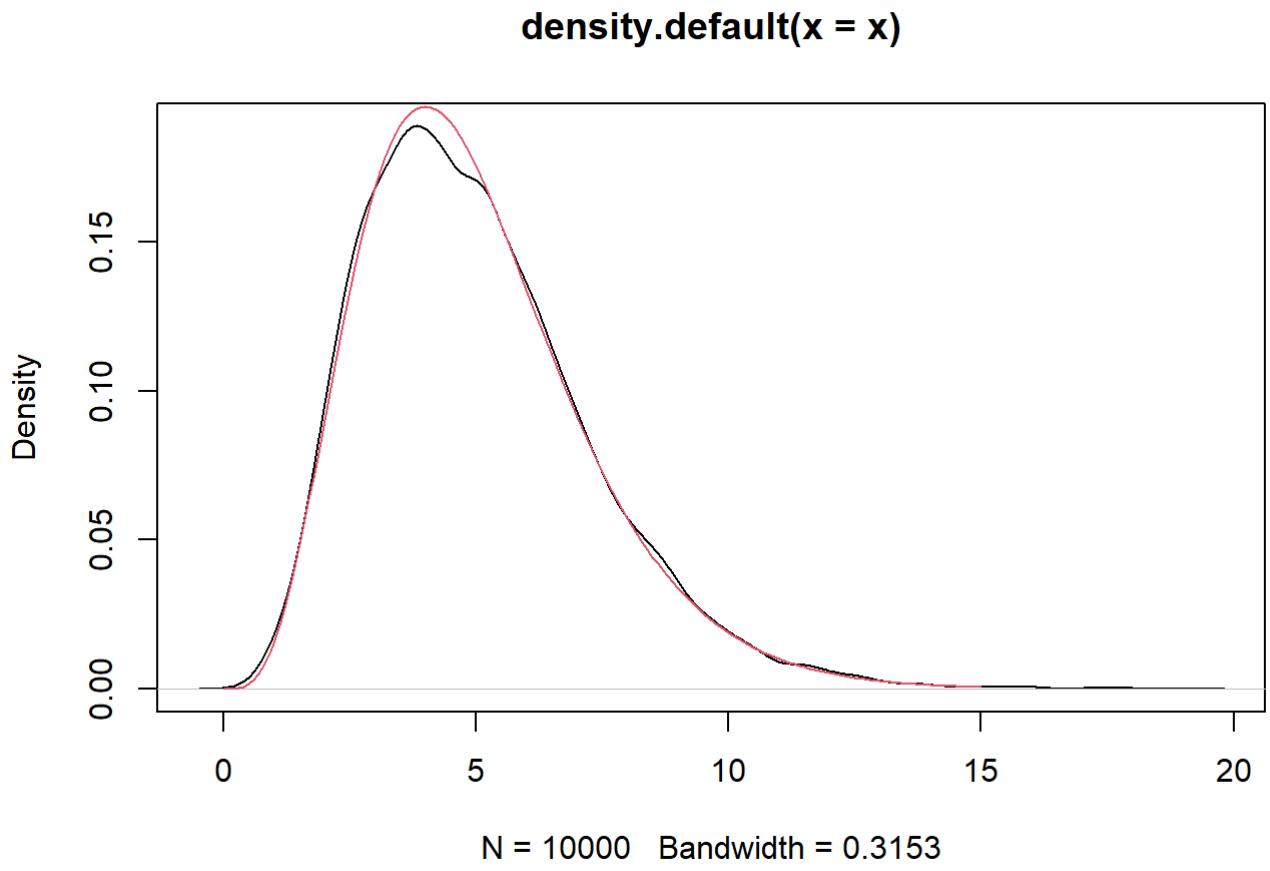
```
x = rgamma(100, 5, 1) # Simulo 100 osservazioni
par(mfrow=c(1,2))
plot(x) # Vedo i dati simulati
plot(density(x)) # Vedo una possibile funione di densità dei dati
```



Per calcolare la stima di densità è necessario avere un kernel. Prendiamo un kernel normal $\phi(x;\mu,\sigma^2)$. La stima di densità $\hat{f}$ in x, la definisco come $\hat{f}(x) = \frac{\sum_{i=1}^{n}\phi(x_i;x,\sigma^2)}{n}$.

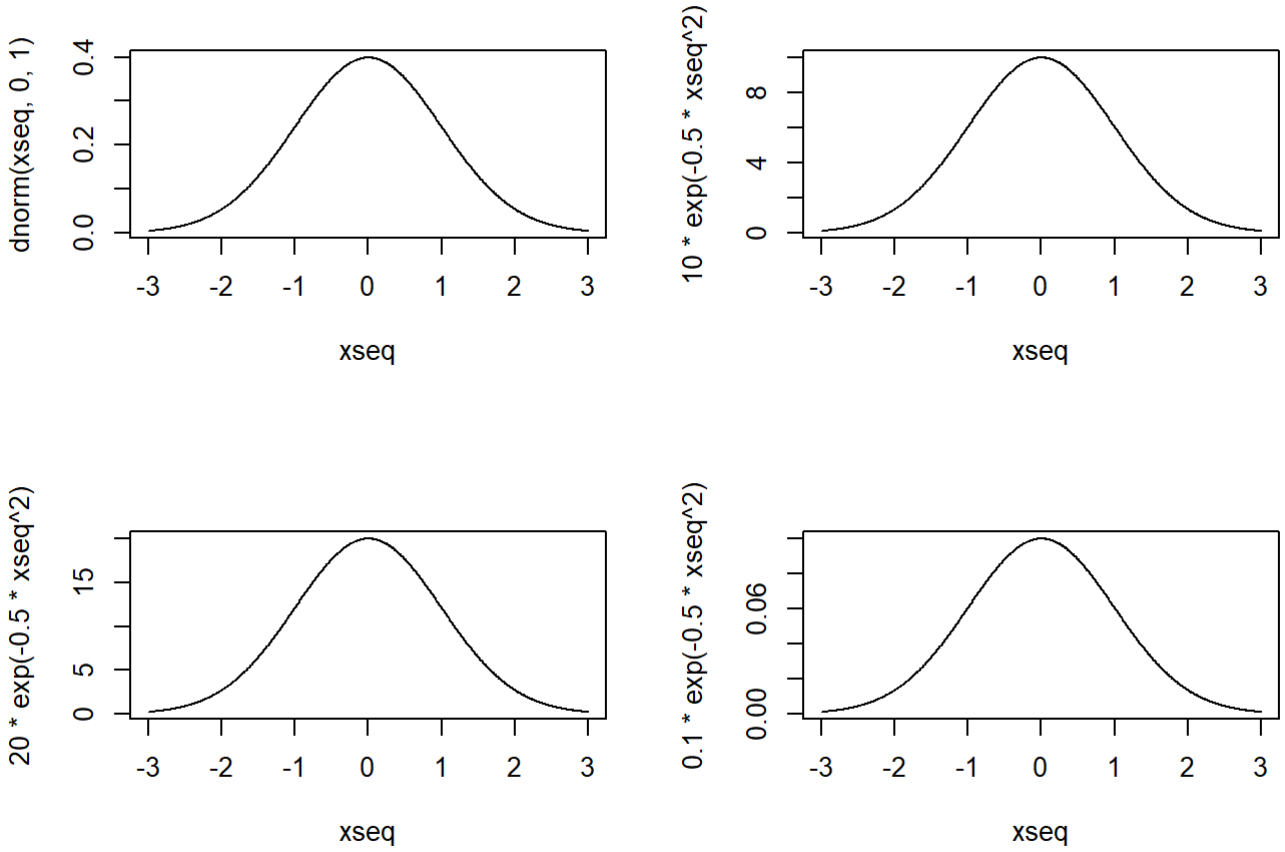
Simuliamo dalle gamma, e confrontiamo la stima con la densità vera.

```
x = rgamma(10000,5,1)

plot(density(x))
xseq = seq(0,15, by=0.1)
lines(xseq, dgamma(xseq, 5,1), col=2)
```



COSTANTI DI NORMALIZZAZIONE: ESEMPI DI KERNEL

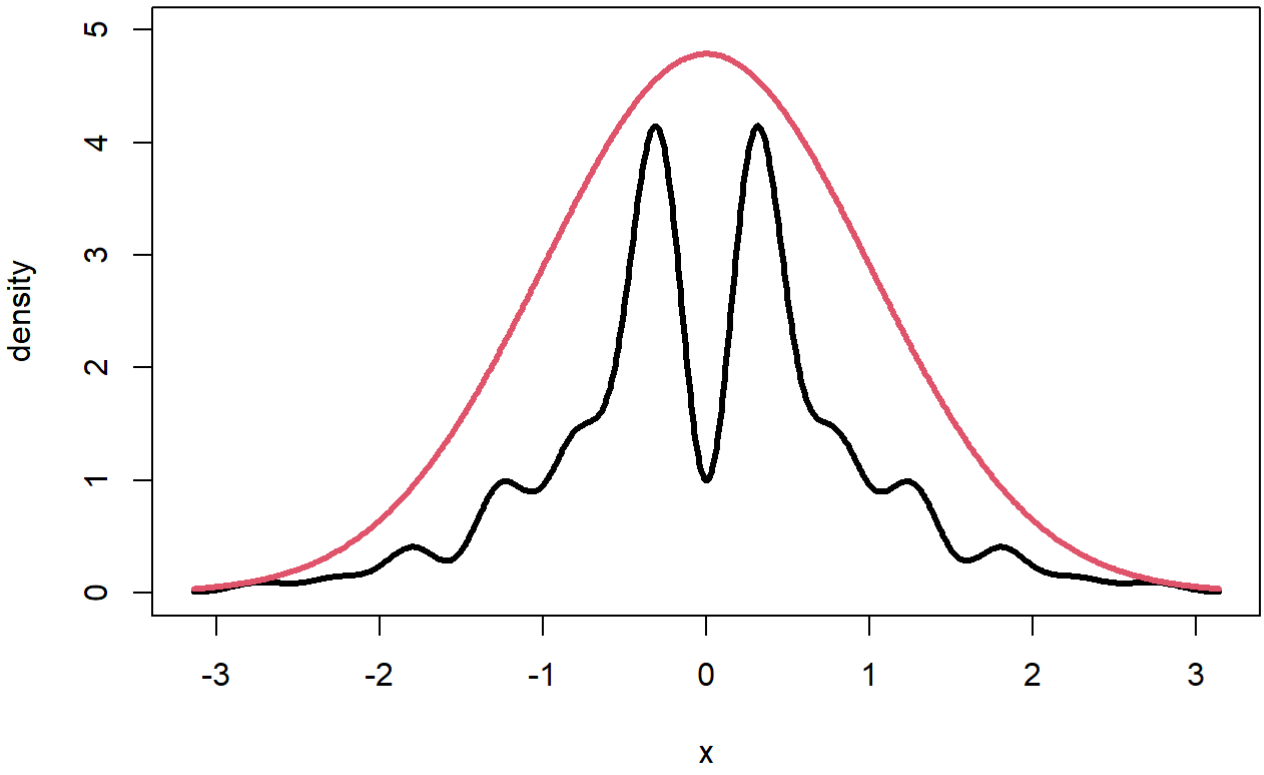
```
xseq = seq(-3,3, by=0.01)

par(mfrow=c(2,2))
plot(xseq, dnorm(xseq, 0,1), type="l")
plot(xseq, 10*exp(-0.5*xseq^2), type="l")
plot(xseq, 20*exp(-0.5*xseq^2), type="l")
plot(xseq, 0.1*exp(-0.5*xseq^2), type="l")
```



KERNEL E DENSITA' NORMALE

Il codice genera un grafico che visualizza una densità di probabilità complessa e la confronta con la densità di probabilità di una distribuzione normale standard moltiplicata per 12.

```
xseq = seq(-pi, pi, by = 0.01) # Crea una sequenza di valori da -pi a pi con un passo di 0.01
# Calcola i valori della densità di probabilità in base a una funzione data
dens = exp(-xseq^2/2)*(sin(6*xseq)^2+3*cos(xseq)^2*sin(4*xseq)^2+1)
# Crea un grafico della densità di probabilità
plot (
  xseq, dens, type = "l", ylim = c(0,5),
  lwd = 3, xlab = "x", ylab = "density"
)
# Sovrapponi al grafico precedente una curva che rappresenta la densità di probabilità di una distribuzione normale standard (con media 0 e deviazione standard 1), moltiplicata per 12.
lines(xseq, 12*dnorm(xseq), col = 2, lwd = 3)
```

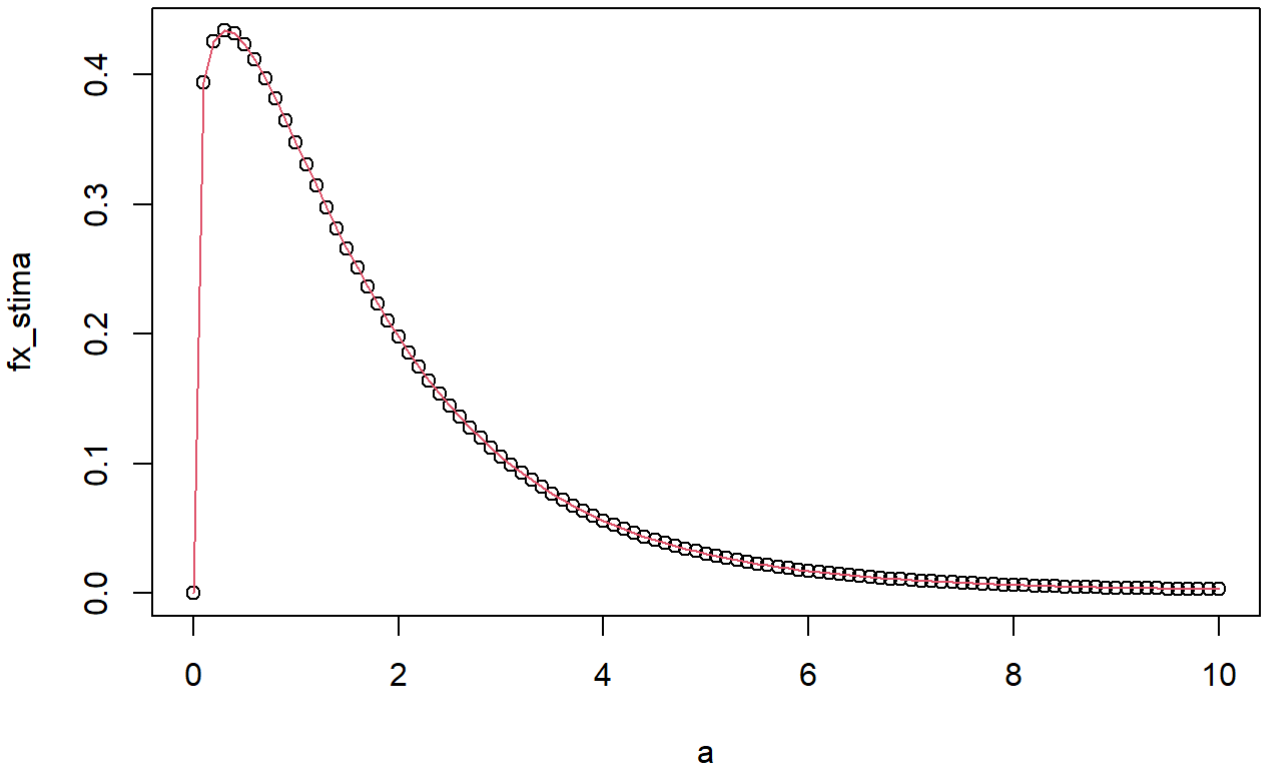


ESEMPIO APPROSSIMAZIONE MONTE CARLO

Ipotizziamo che $Y \sim Exp(\lambda)$ e $X \mid Y = y \sim G(exp, 1)$. Ci chiediamo com'è fatta la densità marg di X? Questo è un caso in cui non possiamo dire nulla riguardo la distribuzione ma possiamo usare l'approssimazione MC.

Il codice esegue una stima della densità di probabilità condizionata di X, dato un campione estratto da una distribuzione esponenziale, e quindi produce un grafico della stima.

```r
n = 1000 # Imposta la dimensione del campione casuale da estrarre dalla distribuzione esponenziale
a = seq(0, 10, by = 0.1) # Questi sono i valori di x su cui verrà calcolata la stima della densità
lambda = 2 # Imposta il parametro lambda per la distribuzione esponenziale
fx_stima = c() # Crea un vettore vuoto fx_stima che verrà utilizzato per immagazzinare le stime della densità di probabilità
# Genera un campione casuale di dimensione n da una distribuzione esponenziale con parametro lambda.
y = rexp (n, lambda )
# Ciclo for per calcolare la stima della densità di probabilità per ogni valore in a
for (i in 1: length (a)){
  # Calcola la densità di probabilità condizionata di a[i] dato il campione y estratto dalla distribuzione esponenziale
  fzgiveny = dgamma (a[i], exp (y) ,1)
  # Calcola la stima della densità di probabilità di X nel punto a[i] come la media delle densità di probabilità condizionate calcolate sopra
  fx_stima[i] = sum ( fzgiveny )/n
}
plot(a, fx_stima) # Crea un grafico della stima della densità di probabilità in funzione di a
lines(a, fx_stima, col = 2) # Sovrapponi al grafico precedente una linea della stima della densità di probabilità
```



DIMOSTRAZIONE NUMERICA LGN

Calcoliamo la media di $log(x)$ con $X \sim G(1, 5)$ La calcoliamo come

$$\frac{\sum_{i=1}^{n} log(x_i)}{n} \approx \int_{R^+} log(x) f(x) dx$$

Il codice fornisce una stima della media dei logaritmi dei valori di un campione estratto da una distribuzione $G(1, 5)$.

```
n = 10000 # Dimensione del campione casuale
# Genera un campione casuale di dimensione n da una distribuzione gamma con forma 1 e tasso (rate) 5. La funzione rgamma restituisce valori casuali da una distribuzione gamma
x = rgamma(n, 1, rate = 5)
sum(log(x))/n
```

```
## [1] -2.185776
```

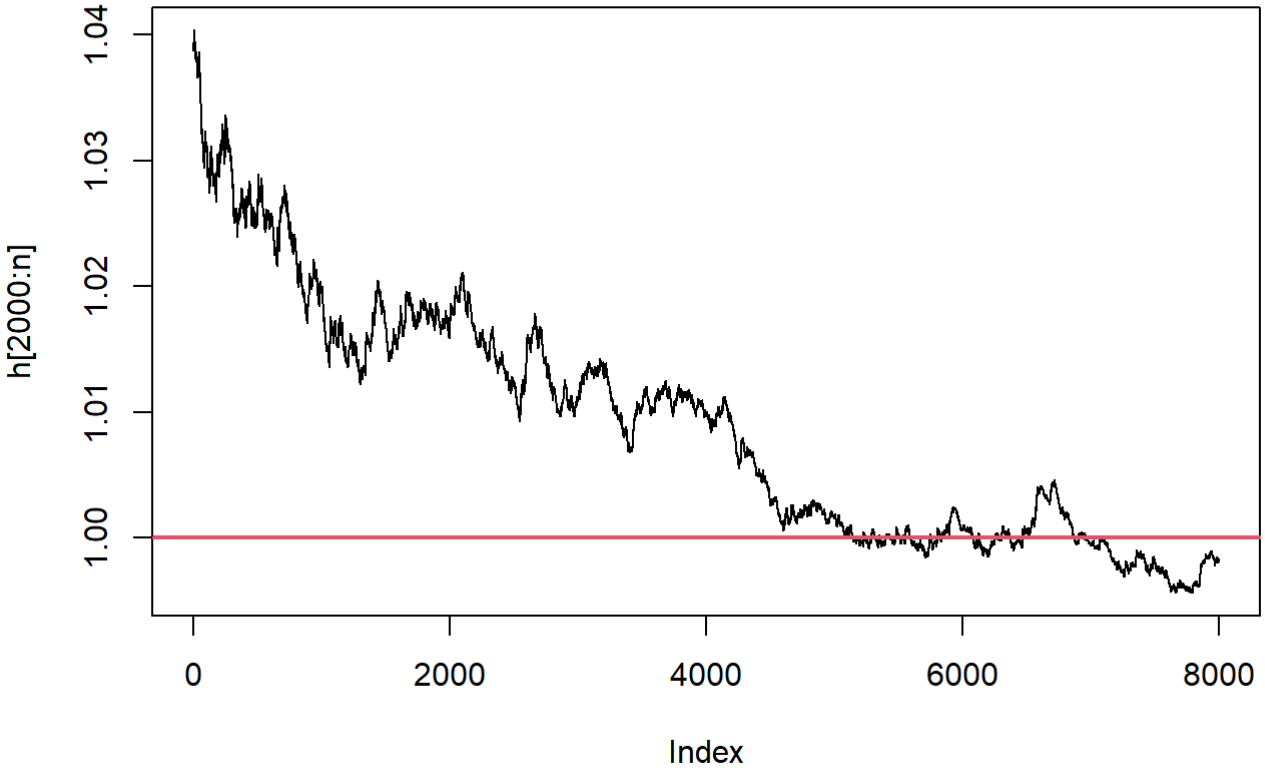
```
mean(log(x)) # Calcola la media dei logaritmi degli elementi nel campione
```

```
## [1] -2.185776
```

“Dimostriamo” la legge dei grandi numeri, numericamente. Assumiamo $X \sim G(1, 1)$. Io so che la media è 1.

Il codice genera un grafico della media cumulativa di un campione casuale di dimensione 10000 estratto da una distribuzione $G(1, 1)$. Viene anche sovrapposta una linea orizzontale a $y = 1$ per evidenziarne il confronto.

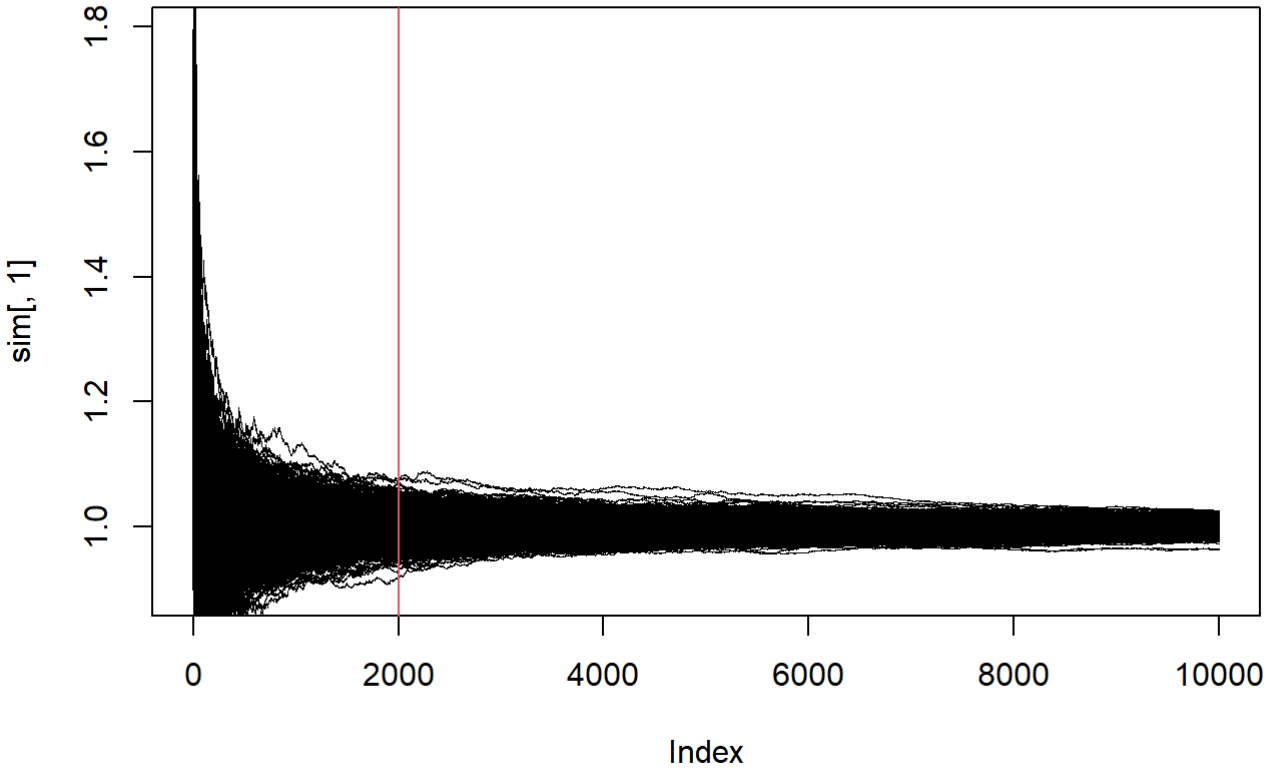
```
n = 10000
# Imposta il seme del generatore di numeri casuali per garantire la riproducibilità dei risultati
set.seed(190)
x = rgamma(n,1,1)
# Calcola la media cumulativa dei valori nel campione
# cumsum(x) restituisce la somma cumulativa degli elementi di x, e poi ogni valore è diviso per il numero corrispondente nella sequenza da 1 a n
h = cumsum(x)/(1:n)
# Crea un grafico della media cumulativa dal 2000-esimo al n-esimo elemento della sequenza h
plot(h[2000:n], type="l")
abline(h = 1, col=2, lwd=2) # Sovrapponi al grafico una linea orizzontale a y = 1
```



Il grafico mostra come la media cumulativa dei primi n valori del campione gamma evolve nel tempo. La linea orizzontale a $y = 1$ serve come riferimento per valutare se la media cumulativa converge a un certo valore.

Vediamo la variabilità. Il codice genera un grafico che rappresenta l'evoluzione della media cumulativa attraverso più simulazioni. Ogni simulazione coinvolge l'estrazione di un campione casuale di dimensione 10000 da una distribuzione $G(1, 1)$.

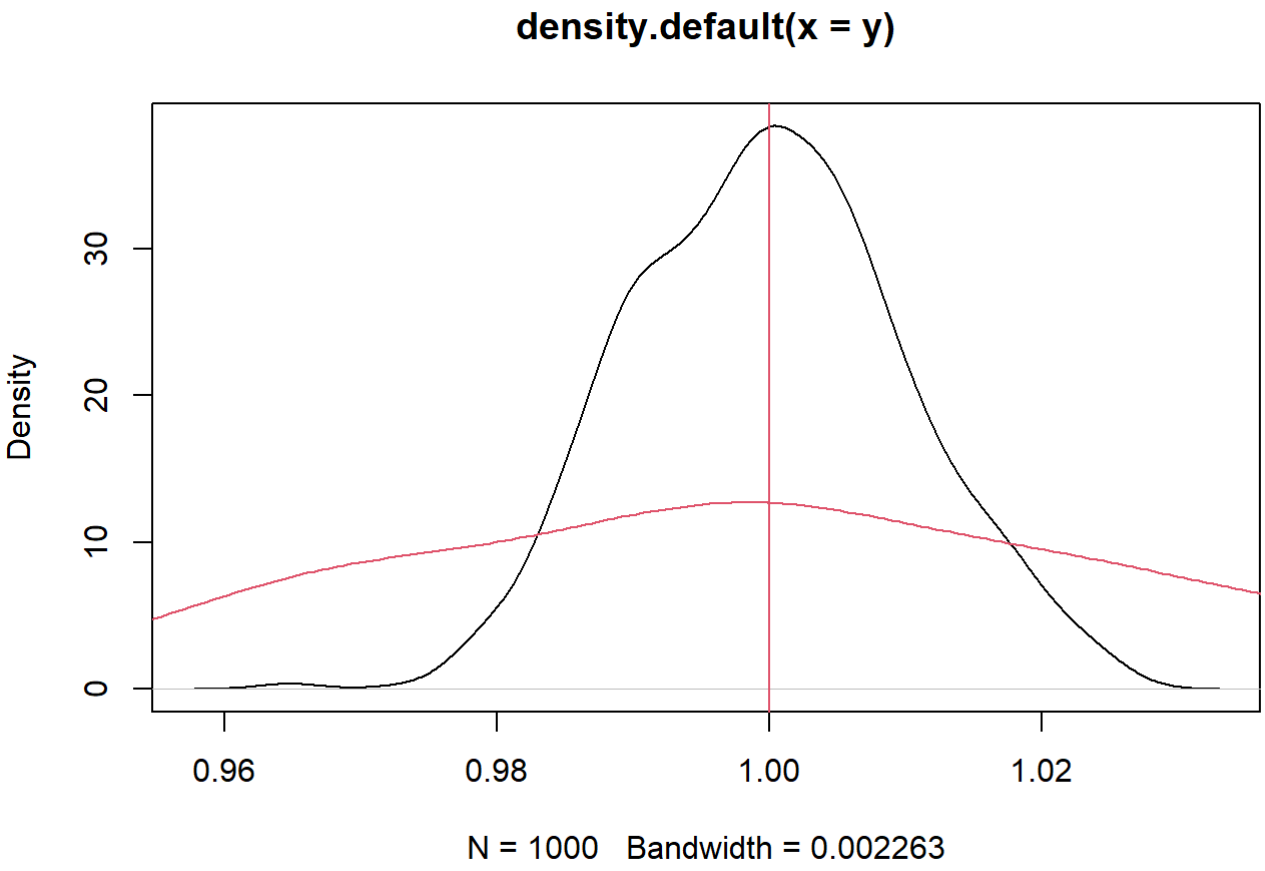
```
n = 10000 # Imposta la dimensione del campione casuale che verrà generato dalla distribuzione gamma
nsim = 1000 # Imposta il numero di simulazioni da eseguire
set.seed(190)
# Crea una matrice sim di dimensioni n x nsim inizializzata con valori mancanti (NA)
sim = matrix(NA, nrow = n, ncol = nsim)
# Esegue le simulazioni
for(i in 1:nsim){
    x = rgamma(n, 1, 1) #  Genera un campione casuale da una distribuzione G(1,1)
    h = cumsum(x)/(1:n) #  Calcola la media cumulativa dei valori nel campione
    sim[,i] = h # Memorizza i risultati della simulazione nella colonna i della matrice sim
}
# Crea un grafico della media cumulativa della prima simulazione.
plot(sim[,1], type="l")
# Sovrappone le linee delle medie cumulative delle simulazioni successive usando la funzione lines
for(i in 2:nsim){
    lines(sim[,i], lwd=0.1)
}
abline(v = 2000, col=2)
```



Il grafico rappresenta l'andamento della media cumulativa attraverso diverse simulazioni e include una linea verticale al punto 2000 sulla scala x. Questo può essere utile per valutare la variabilità delle medie cumulative a diversi momenti.

Il codice crea un grafico della densità di probabilità stimata per due differenti insiemi di dati presenti nella matrice sim.

```
y = sim[10000,] # Rappresenta i valori corrispondenti alla simulazione finale (alla 10000-esima iterazione)
# Crea un grafico della densità di probabilità stimata per i valori di y utilizzando la funzione density. La funzione density stima la densità di probabilità non parametrica basandosi su i dati forniti
plot(density(y))
# Sovrapponi al grafico precedente un'altra curva della densità di probabilità stimata, questa volta per i valori della simulazione alla 1000-esima iterazione
lines(density(sim[1000,]), col=2)
abline(v = 1, col=2)
```



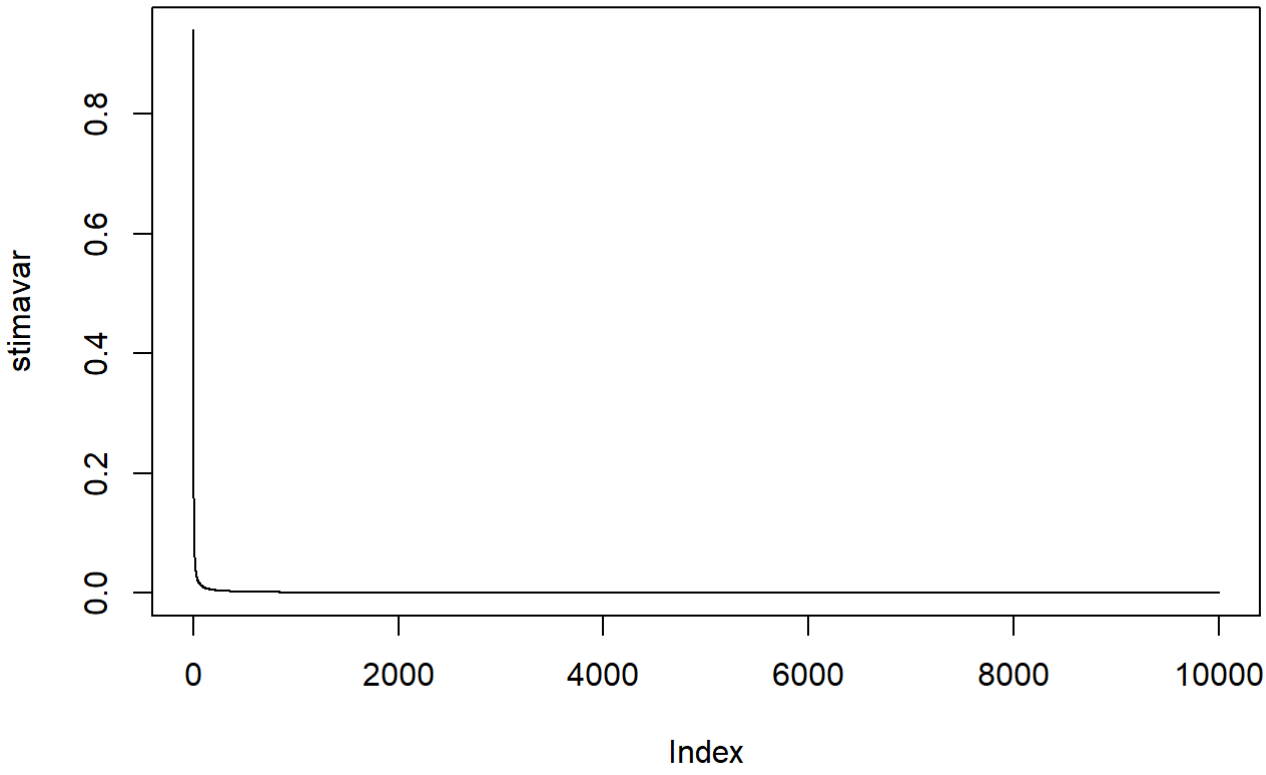
Il grafico mostra le stime della densità di probabilità per due diverse simulazioni (alla 10000-esima e alla 1000-esima iterazione) e include una linea verticale in corrispondenza del valore 1 sull'asse x. Questo può essere utilizzato per confrontare la distribuzione di probabilità tra due momenti diversi della simulazione.

Calcoliamo la stima della varianza

$$\frac{\sum_{i=1}^{n}(x_i - \hat{E}(x_i))^2}{n}$$

Il codice calcola la varianza stimata per ciascuna delle n simulazioni e crea un grafico dell'andamento di queste varianze stimare nel corso delle simulazioni.

```
# Crea un vettore stimavar di lunghezza n inizializzato con valori mancanti (NA). Questo vett
ore sarà utilizzato per memorizzare le varianze stimate
stimavar = rep(NA, n)
stimavar[1] = 0
# Ciclo for itera attraverso le simulazioni (da 1 a n
for(i in 1:n){
    xbar = mean(sim[i,]) # Calcola la media dei valori della simulazione corrente
    # Calcola la varianza stimata per la simulazione corrente e la memorizza nel vettore stim
avar. La varianza stimata è calcolata come la somma dei quadrati delle differenze tra i valor
i e la media, diviso per il numero di colonne nella matrice sim (che rappresenta il numero di
campioni in ogni simulazione)
    stimavar[i] = sum((sim[i,]-xbar)^2)/ncol(sim)
}
plot( stimavar, type="l") # Crea un grafico dell'andamento delle varianze stimate nel corso d
elle simulazioni
```




```
dim(sim) # Restituisce le dimensioni della matrice sim
```

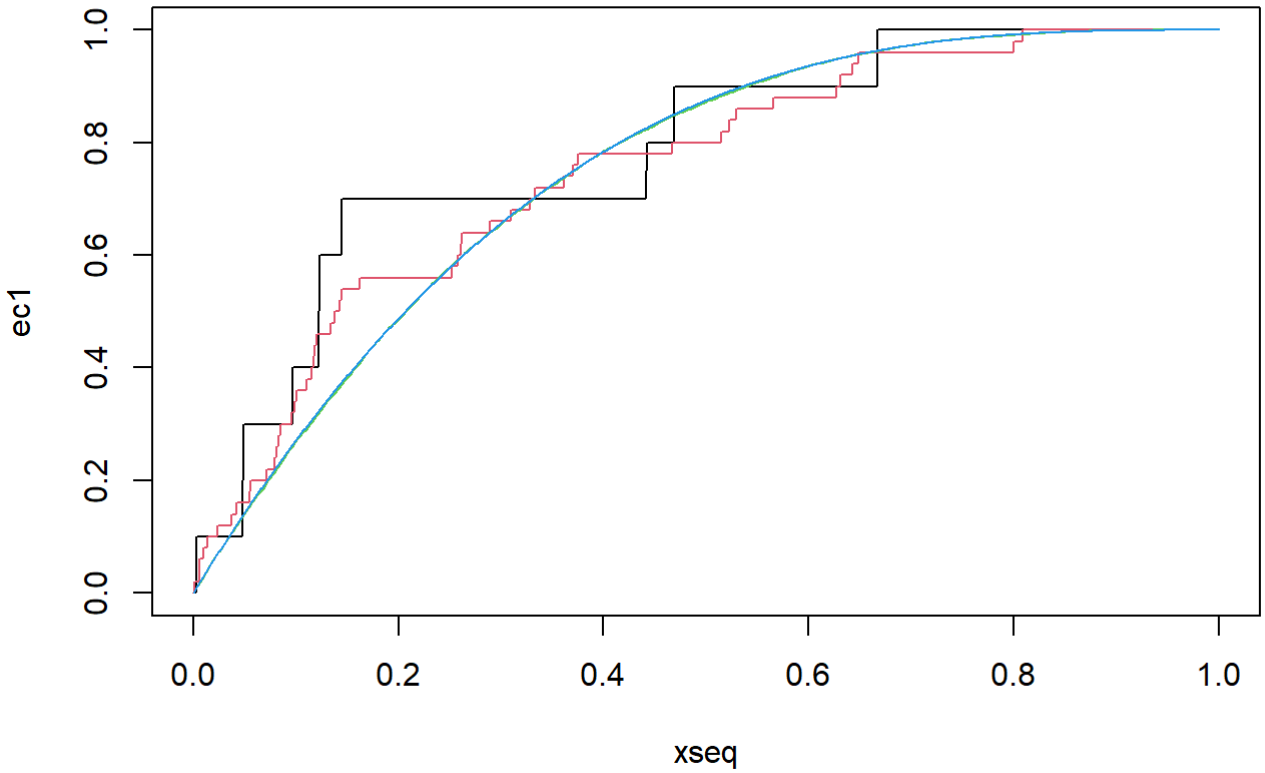
```
## [1] 10000  1000
```

Il codice genera un grafico che mostra come variano le varianze stimare attraverso le diverse simulazioni. Questo può essere utile per osservare la stabilità o la variabilità delle varianze stimare nel corso del processo di simulazione.

FUNZIONE DI RIPARTIZIONE DI UNA $B(1, 3)$

Calcoliamo la funzione di ripartizione di una $B(1, 3)$. Il codice genera tre distribuzioni cumulative empiriche (ECDF) per campioni estratti da una distribuzione $B(1, 3)$. Inoltre, viene sovrapposta la funzione di distribuzione cumulativa teorica (CDF) per la distribuzione beta con gli stessi parametri.

```
# Genera un campione di dimensione n1, n2, n3 da una distribuzione B(1,3)
x1 = rbeta(10, 1, 3)
x2 = rbeta(50, 1, 3)
x3 = rbeta(10000, 1, 3)

xseq = seq(0,1, by = 0.001) # Questi valori saranno utilizzati per valutare le ECDF e la CDF teorica
# Calcola la ECDF per i campioni
ec1 = ecdf(x1)(xseq)
ec2 = ecdf(x2)(xseq)
ec3 = ecdf(x3)(xseq)

plot(xseq, ec1, type = "l") # Crea un grafico della ECDF per il primo campione
lines(xseq, ec2, col = 2) # Sovrapponi al grafico una linea della ECDF per il secondo campione
lines(xseq, ec3, col = 3) # Sovrapponi al grafico una linea della ECDF per il terzo campione
lines(xseq,  pbeta(xseq, 1, 3), col = 4) # Sovrapponi al grafico una linea della CDF teorica della distribuzione B(1,3)
```



Come si vede dalla la funzione è a scalini con scalini di altezza $\frac{1}{n}$ e per $X_{i-1} < a \leq X_i$ è costante. Il codice mostra come variano le ECDF per campioni di dimensioni diverse estratti dalla stessa distribuzione beta e le confronta con la CDF teorica corrispondente.

AREA PORZIONE DEL CERCHIO

Calcoliamo l'area di una porzione del cerchio. Io voglio calcolare

$$\int_{-1}^{1}(\sqrt{1-x^2})dx$$

Opzione 1

$$\int_{-1}^{1}(\sqrt{1-x^2})2\frac{1}{2}dx$$

Definisco $h(x) = (\sqrt{1-x^2})2$ e allora

$$\int_{-1}^{1}h(x)\frac{1}{2}dx$$

Se $X \sim U(-1, 1)$ allora

$$\frac{\sum(1-x_i^2)2}{n} \approx \int_{-1}^{1}h(x)\frac{1}{2}dx$$

Opzione 2

$$\int_{-1}^{1}(\sqrt{1-x^2})dx = \int_{-1}^{1}\int_{0}^{\sqrt{1-x^2}}f(y)dydx$$

$f(y)$ densità di un'uniforme e quindi

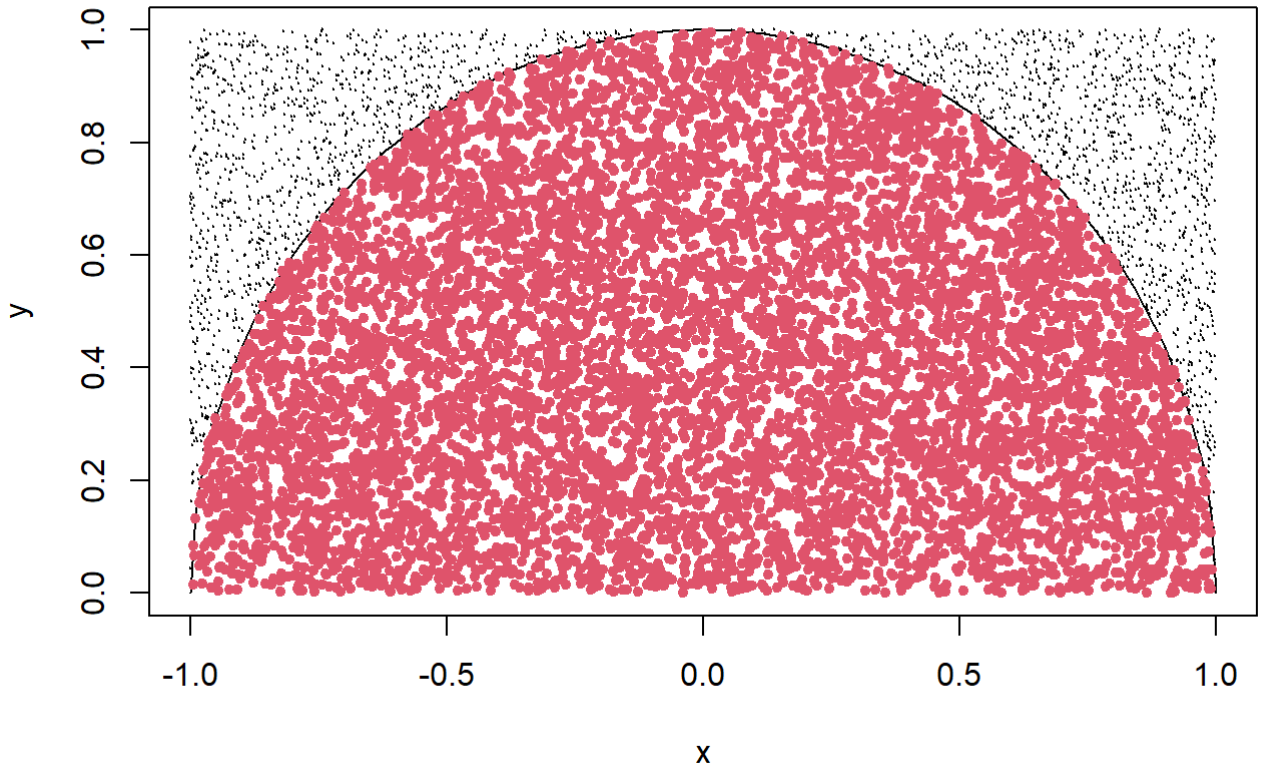
$$\int_{-1}^{1}\int_{0}^{1}1_{y<\sqrt{1-x^2}}f(y)dydx$$

calcoliamo il primo integrale

```
x = 0.5
y = runif(10000, 0,1)
sum(y < sqrt(1-x^2))/10000
```

```
## [1] 0.8676
```

```
# valvoliamola
x = runif(n, -1,1)
y = runif(10000, 0,1)

plot(x,y, cex = 0.1, pch=20)
xseq = seq(-1,1, by = 0.01)
lines(xseq, sqrt(1-xseq^2))
points(x[(x^2+y^2)<1],y[(x^2+y^2)<1], col=2, pch=20)
```




```
2*sum( (x^2+y^2)<1 )/10000
```

```
## [1] 1.5728
```

ESEMPIO DI METODI SIMULATIVI

Assumiamo che $X \sim Exp(\lambda)$ e volgiamo calcolare

$$F(a;\lambda) = \int_0^a f(x)dx = \int_0^\infty 1_{<a}(x) f(x)dx \approx \frac{\sum_{i=1}^n 1_{\leq a}(x_i)}{n} = \frac{\sum_{b=1}^B 1_{\leq a}(x^b)}{B}$$

Il codice genera un campione di variabili casuali da una distribuzione esponenziale, stimando quindi la CDF empirica al punto specificato a e confronta questo valore con il valore esatto della CDF al punto a calcolato dalla funzione di distribuzione esponenziale

```
n = 1000 # numero di variabili casuali generate
lambda = 1 #  tasso del processo esponenziale
a = 2 # punto in cui si desidera valutare la funzione di distribuzione cumulativa (CDF)

# Genera n variabili casuali da una distribuzione esponenziale con tasso lambda utilizzando l
a funzione rexp(n, lambda). Queste variabili casuali seguono una distribuzione esponenziale
x = rexp(n, lambda)
Fa_stimata = sum(x <= a)/n # Calcola la funzione di distribuzione cumulativa (CDF) empirica s
timata nel punto a
Fa_stimata
```

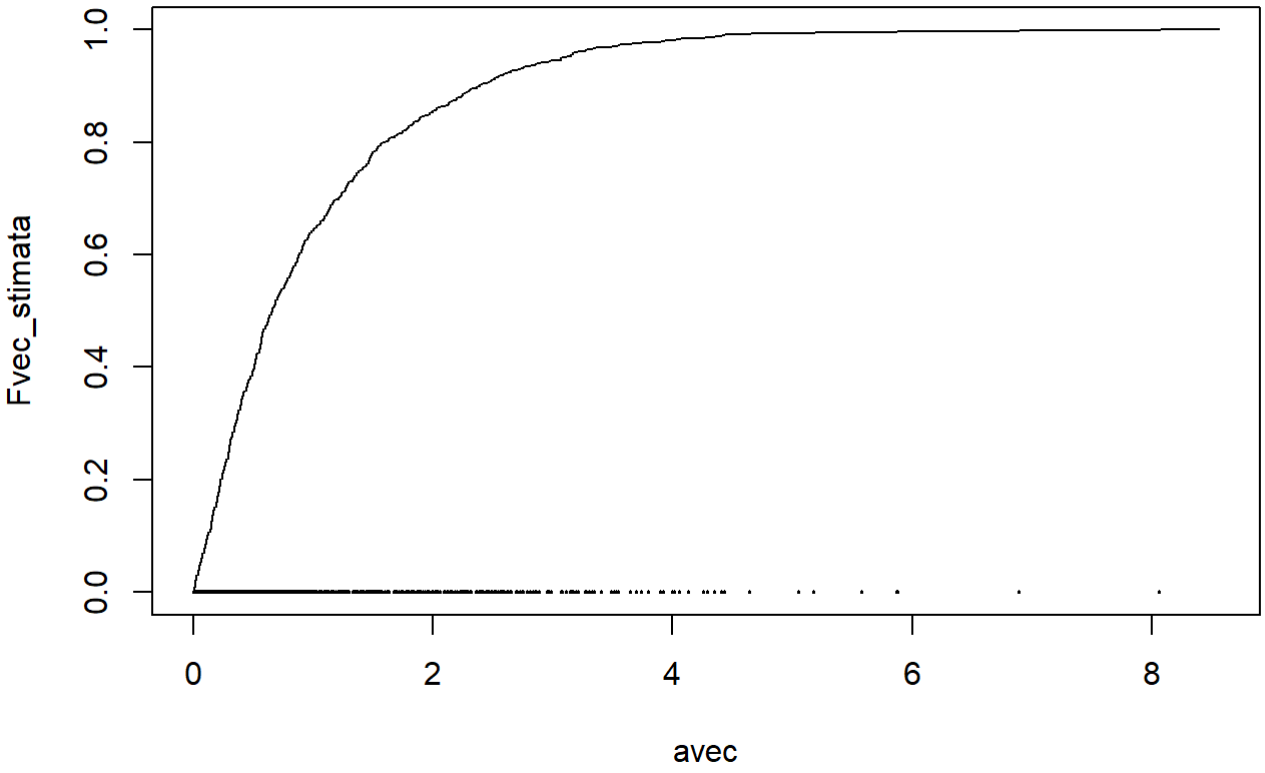
```
## [1] 0.855
```

```
pexp(a,lambda) # Calcola il valore esatto della CDF al punto a utilizzando la funzione pexp (a, lambda) della distribuzione esponenziale con il tasso lambda
```

```
## [1] 0.8646647
```

Calcoliamo l'integrale su diversi valori. Il codice estende il concetto di calcolare la funzione di distribuzione cumulativa (CDF) empirica stimata, ma su una griglia più ampia di valori.

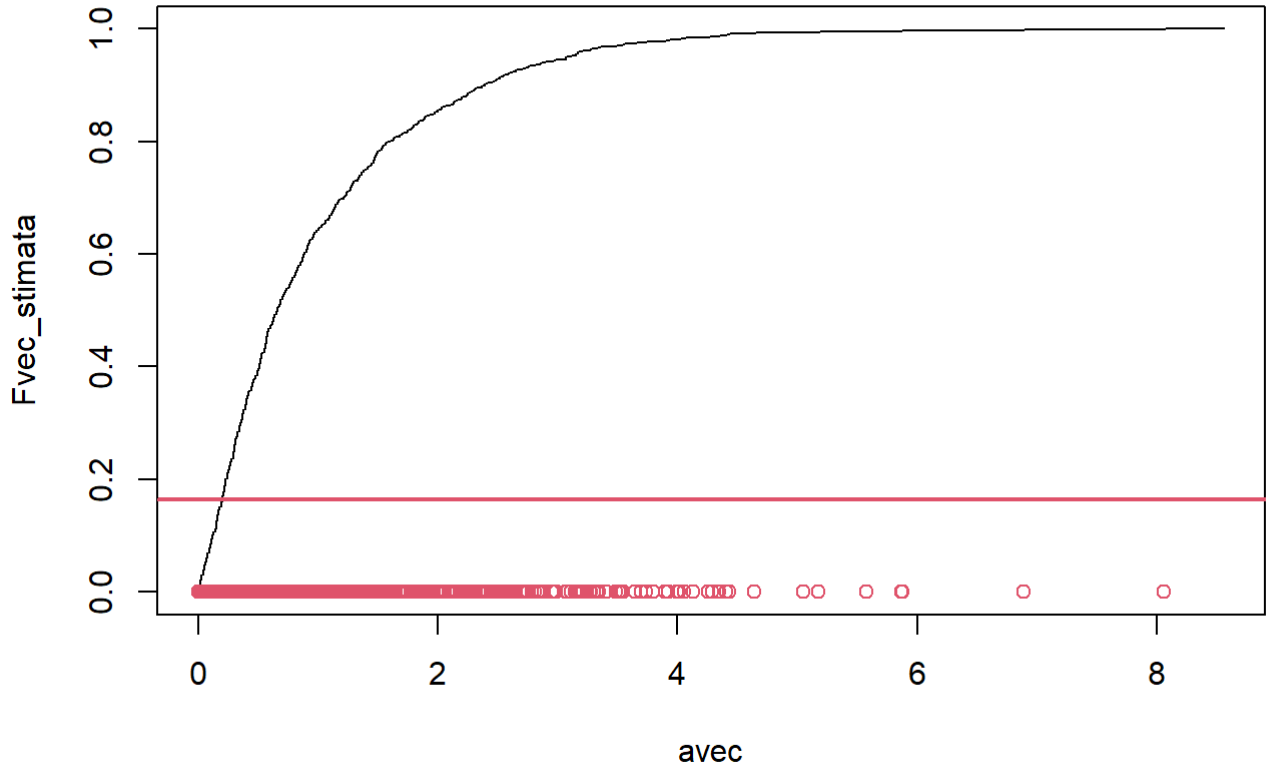
```
# Genera un campione di n variabili casuali da una distribuzione esponenziale con tasso lambda
x = rexp(n, lambda)
avec = seq(0.00000000001, max(x)+0.5, by = 0.01)

Fvec_stimata = c() # vettore che verrà utilizzato per memorizzare i valori stimati della CDF per ciascun valore in avec

# Calcola la CDF empirica stimata per ciascun valore in avec
for(i in 1:length(avec)){
    Fvec_stimata[i] = sum(x <= avec[i])/n
}

plot(avec, Fvec_stimata, type="s") # Plotta la CDF empirica stimata
points(x, rep(0, n), cex = 0.3, pch = 20) # Aggiunge al grafico dei punti corrispondenti alle osservazioni nel campione x
```



Il risultato finale è un grafico che visualizza la CDF empirica stimata su una griglia più ampia di valori insieme alle singole osservazioni nel campione.

Aggiungiamo alcune funzionalità simulando da $\hat{F}(x)$.

```
u = runif(1, 0,1) # Genera un numero casuale uniforme u nell'intervallo [0, 1]
plot(avec, Fvec_stimata, type="s" ) # Genera un numero casuale uniforme u nell'intervallo [0, 1]
abline(h = u, col = 2, lwd = 2) # Aggiunge una linea orizzontale nel grafico al valore di u
points(x, rep(0, n), col = 2) # Aggiunge i punti corrispondenti alle osservazioni nel campione x sul grafico
```




```
Fvec_stimata_x = c(1:n)/n

x_ord = x[order(x)] # Crea un vettore Fvec_stimata_x che rappresenta i valori della CDF empirica stimata per i dati ordinat
```

Il codice sembra finalizzato a visualizzare la CDF empirica stimata, evidenziando un valore casuale u sulla scala della CDF e confrontando la CDF empirica stimata con una rappresentazione ordinata.

Simuliamo nsim punti. Il codice esegue una simulazione di campionamento inverso utilizzando la CDF empirica stimata.

```
nsim = 10000 # numero di simulazioni da eseguire
xsim = c()
for(i in 1:nsim){
    u = runif(1, 0,1) # Genera un numero casuale uniforme u nell'intervallo [0, 1]
    w = which(Fvec_stimata_x >= u)[1] # Trova il più piccolo indice in Fvec_stimata_x tale ch
e il valore corrispondente è maggiore o uguale a u
    xsim[i] = x_ord[w] # Usa l'indice trovato per selezionare il corrispondente valore ordina
to da x_ord e memorizzarlo in xsim
}
table(xsim) # tabella delle frequenze
```

```
## xsim
## 0.000122875354193619  0.00119423493742943  0.00306829786004922
##                    6                   10                   13
##  0.00344218461894917  0.00465072225779295  0.00466586276888847
##                   18                    6                    7
##  0.00526811894098949  0.00627049105241895  0.00685970857739449
##                    3                    6                    8
##  0.00797855947166681   0.0117612440432197   0.0118096554651856
##                    8                   13                    9
##   0.0120712933130562     0.01225866864545   0.0137125224396302
##                    7                    4                    6
##    0.015223877504468   0.0167840230029482   0.0168489362113178
##                    9                   14                    8
##   0.0173210174922566   0.0177569668740034   0.0181542133760773
##                    9                   10                   12
##   0.0184811226348965   0.0213477086107098   0.0236349399201572
##                   13                   15                   11
##   0.0243637276051112    0.025243419688195   0.0261380760930479
##                    8                   11                    9
##   0.0263544041739978   0.0274642519708259   0.0286007247299706
##                   13                    9                    8
##   0.0296244618498965   0.0303968014195561   0.0332814729772508
##                   10                    9                   10
##   0.0333377250810375   0.0354159637354314   0.0363425487352649
##                    7                   12                    9
##   0.0381560519852626   0.0382213643752038   0.0385611010715365
##                   10                   12                    6
##   0.0387183567873122   0.0408348333457482    0.040879852604121
##                   14                    5                    9
##   0.0418957276269794   0.0441684105769079    0.046578967012465
##                   10                   10                   10
##   0.0485354908742011    0.049780047645661   0.0499911750666797
##                    9                    8                    6
##   0.0500280819833279   0.0510787381790578   0.0553949508423433
##                   10                   14                    8
##    0.057328789960593   0.0582069577649236   0.0601430144160986
##                   13                   10                    6
##   0.0604381649694188   0.0640529650263488   0.0656950203701854
##                   12                   12                    9
##   0.0660844570957124   0.0671731265263159   0.0693300988427624
##                   10                   11                   14
##   0.0702539747580886    0.070277173537761   0.0711922971531749
##                    9                   10                    9
##   0.0713326493278146   0.0726107130758464   0.0736139425837343
##                   11                    7                    5
##   0.0740763419307768   0.0745831481181085   0.0749274937668561
##                    5                    8                    5
##   0.0757859368615294   0.0802031750899304   0.0828170294486845
##                   13                    7                   13
##   0.0833529438823462   0.0842128074727952   0.0859892538597579
##                    9                   12                   16
##   0.0863132113703893   0.0864062793552876   0.0893614515447213
##                   10                    9                   13
##   0.0907954494468868   0.0916888422116506   0.0932731716893613
##                   11                    6                   10
```

```
##   0.0934560943595224   0.0951884309761226   0.0976602998562157
##                    3                   11                    6
##    0.097730222158134    0.104362591690819    0.105275224298859
##                   15                    8                   15
##    0.105530726723373    0.105950771830976    0.106264309121594
##                   11                    8                    6
##    0.107360366977066     0.10927766234363     0.10961552336812
##                   10                    5                   13
##    0.109989697113633    0.110959011679644     0.11141163809225
##                   10                    5                    9
##    0.112385522108525    0.117012342903763     0.11885629221797
##                    5                   11                    5
##    0.118949301540852    0.119529671501368    0.121080764569342
##                    5                    8                   15
##    0.121110870136776    0.122051862121446    0.123063447044171
##                   14                   14                    7
##    0.123532914766218    0.128765096887946    0.130533816569503
##                   12                   10                   10
##    0.130831114904006    0.131609904579818    0.132709746714681
##                    9                   14                    7
##    0.135438369121403    0.137493314221501    0.140308632981032
##                    8                    8                   12
##    0.140815068036318    0.142419444279577     0.14266883675009
##                   10                   13                    7
##     0.14372505992651    0.144201203729149    0.144647991284728
##                   12                   11                    6
##    0.144841302186251    0.147199084050953    0.147317360620946
##                   11                   12                   11
##    0.148223316168114    0.148483104222448    0.148938231515819
##                   11                    9                    4
##    0.150559628382325    0.150637533981353    0.151006819389359
##                    9                    8                   13
##    0.151951749512863    0.153176794245335    0.153563125990331
##                   11                   15                   10
##     0.15522053772724    0.155830970965326    0.156082683242857
##                    8                    8                    7
##    0.157736952925547    0.158192383301646    0.159796990919858
##                    8                    9                    7
##    0.160274996255804    0.160404849331826     0.16157574808425
##                   13                    6                   11
##    0.162126503884792    0.162951864302158    0.163704753837359
##                   16                    6                    7
##    0.163940087731173       0.167874077335    0.169401579536498
##                   11                    6                   10
##    0.173583108466119    0.174130833242089    0.174304882995784
##                   12                    4                   10
##    0.178523656536279    0.181060782633722    0.182149556897287
##                    9                   18                    8
##    0.182259440887719    0.182274839840829    0.182789310310603
##                   18                    8                   16
##    0.183110354002565    0.184499741066247    0.186147733385872
##                    5                   13                    9
##    0.189557261244335    0.191426327724517    0.192011529114097
##                    6                   10                   19
##    0.192624556366354     0.19309428660199    0.194879357237369
##                    6                   16                   18
```

```
##     0.19498479925096    0.195522408001125    0.195903779517931
##                   12                    9                   10
##    0.197583772728192    0.198295950164033    0.198422409594059
##                    6                   11                   10
##    0.199185142293572    0.204517501406372    0.205275862012058
##                    7                    9                   12
##    0.205627778282679    0.207821355666965    0.208755017986816
##                   14                   11                   13
##    0.209818526636809    0.209985854569823    0.210716644302011
##                    9                    6                    6
##    0.212347107939422    0.213620149062661    0.215536586008966
##                    4                    5                   15
##    0.217138169333339    0.217908271122724    0.218188273254782
##                   13                    9                   11
##    0.218812826555222    0.219302966514162    0.219482461456209
##                   12                    7                   17
##     0.22026285452881    0.220959621947259    0.222415586467832
##                   12                   14                    9
##     0.22292352700606    0.224550512153655    0.225372339356656
##                   10                   13                   17
##    0.226158214267343    0.226848366204649    0.227362925652415
##                    8                   12                    7
##    0.228077217936516    0.229038792196661    0.229756203480065
##                   12                   14                    9
##    0.230554490815848    0.232597986524581    0.233738522510976
##                   10                    7                   12
##    0.234364341478795    0.236611137166619    0.237651167903095
##                    7                   13                   17
##    0.238247933331877    0.238723804248454    0.238777300770749
##                   15                    5                   11
##    0.239505642093718    0.239595555700362    0.243647587485611
##                   14                   12                    9
##    0.244449027944299     0.24627014156431    0.246692469537127
##                    8                    4                   12
##    0.248474845803624    0.249332272005715    0.252331375982112
##                    9                   12                   14
##    0.254762570839375     0.25735796475783    0.258152514709925
##                   10                    9                   10
##    0.259132107254118    0.259405793156475    0.263486433750387
##                   10                    5                    8
##     0.26443713484332    0.264437427278608    0.264490574132651
##                    4                    8                   17
##    0.265378053765744    0.266497666200256      0.2690107378054
##                    4                    9                   10
##    0.270994944963604    0.271235228981823    0.271600299712311
##                   10                    5                   10
##    0.272527023246419    0.273069133516401    0.277616722509265
##                   10                    9                   11
##    0.280550672551265    0.282735730987042    0.283764032938231
##                   10                   12                   18
##    0.284041685517877    0.284822432789952    0.285189752466977
##                    9                    7                   12
##    0.285321427043527     0.28546515153721    0.286301342304796
##                    6                   12                   12
##    0.286858360748738     0.28710761340335    0.288006059359759
##                   15                   12                    8
```

```
##    0.289983164984733    0.290472919213201    0.291019963100553
##                   16                    6                   16
##    0.292006806936115     0.29241937585175    0.292501187562138
##                   11                    5                   15
##    0.293100579490516    0.294144498184323    0.294848444405943
##                   13                    8                   12
##    0.296380575280637    0.298095051664859    0.298837054055184
##                    8                   13                   10
##    0.298968496266752    0.300619213376194    0.301214527804404
##                   12                   11                   11
##    0.301779856126215    0.302674782462418    0.304444617893203
##                   10                    7                   10
##    0.305191268810653    0.305965680847186    0.307077031116933
##                    8                    6                    6
##    0.307439037133008    0.308398368877009    0.308802509214729
##                    8                    8                   11
##    0.310422906652093    0.318925791885704    0.319932162761688
##                   14                    8                    4
##    0.321177102159709    0.321605215780437     0.32255927240476
##                    8                   11                    7
##     0.32264662720263    0.322844776770454    0.327595471870154
##                    8                   10                   14
##    0.327916010748595    0.329048180122225    0.329902512487024
##                   15                    8                   15
##    0.329992950150913    0.330981768667698    0.331065197937009
##                   13                    5                    9
##    0.331362851895392    0.331400929018855    0.331739543937147
##                    6                    4                   10
##    0.334417644422501    0.336223995196197    0.336915963243571
##                   10                   15                    9
##    0.338217686396092    0.343491103965789    0.343904289416969
##                    5                   11                    8
##    0.344081573653966    0.347685357090086    0.349401911720634
##                    9                   10                   14
##    0.349904409144074    0.350232750177383    0.350899526692477
##                   10                   11                    8
##    0.352109367027879    0.354069305583835    0.355617507553349
##                    7                    7                    5
##    0.356673767324537    0.360693916678429    0.361876557249045
##                   11                    8                   11
##    0.362140615470707    0.362931320443749    0.363173617981374
##                    8                    9                    7
##    0.365215362049639    0.366378211881965    0.366956988740873
##                    7                   10                   10
##    0.368966187816113    0.369046851311234    0.371231575063608
##                   11                   12                   13
##    0.372592127881944    0.372622911818326    0.375000061932951
##                   13                   12                    5
##    0.375412760768086    0.375509622307059    0.377641418483108
##                   16                   12                   15
##    0.379332203883678    0.383002201095223    0.384334652201345
##                    7                   11                   11
##    0.384788711555302    0.384867326356471    0.385687477421016
##                    7                    9                    9
##    0.386438497342169     0.38873229882652    0.389181585982442
##                   11                    8                   13
```

```
##    0.392605190792133    0.392911706119776     0.39315488608554
##                   18                   11                   11
##    0.393612553365529    0.394615489058197    0.396425155922771
##                    5                    6                   11
##    0.396522799241309    0.397283902391791    0.398369313683361
##                   10                   11                   13
##    0.399902125820518    0.400014572311193    0.401947268284857
##                    9                    7                    7
##    0.404043264687061    0.405771784096026    0.408313937485218
##                   14                    5                    7
##    0.408799365628511    0.409286366309971    0.411668735788966
##                   15                    9                    7
##    0.412032031454146    0.413288537705469    0.417692068964243
##                    9                   11                    6
##    0.418532866053283    0.418759868612674    0.420981747563928
##                    8                    6                   11
##    0.421056282240897    0.423319708555724    0.431208542082459
##                    8                   12                   14
##    0.432973414193839     0.43476129649207    0.436326683498919
##                    6                   10                   11
##    0.438880071975291    0.439591523259878    0.441678404812013
##                    9                    8                    7
##    0.443797775544226     0.44444127904144    0.445466769393533
##                   12                    9                    7
##    0.447972908616066    0.448140853550285    0.449421667959541
##                   10                   11                   15
##    0.452462511602789    0.452985872514546    0.454725660849363
##                    7                   11                   12
##    0.454750352539122    0.455133982468396    0.462286192923784
##                   15                   15                   14
##    0.464664885308594    0.465217414134851    0.470161077100784
##                   11                   10                   13
##    0.472358622099684    0.477249150630087    0.478464193176478
##                    6                   10                    8
##    0.479419887531549    0.480128421913832    0.480516442097723
##                   11                   11                   10
##    0.480727427760608    0.480758648365736    0.480854813009501
##                    9                   15                   16
##    0.481342273298651    0.482819461616924    0.485174386762083
##                   13                   10                   12
##     0.48901060083881    0.491625429576521    0.494944024365395
##                    6                   15                   13
##    0.497167780529708     0.49822349473834    0.500518454704434
##                    4                    9                   15
##    0.501123412977904    0.502540763933212    0.502936955541372
##                   10                    8                   11
##    0.504063190426677    0.504929831717163    0.505273228045553
##                    6                   10                   10
##    0.507516035810113    0.509216097649187    0.512009875848889
##                    9                    5                   12
##    0.512978326529264       0.514028087724    0.515428580809385
##                   10                   12                    5
##    0.517739427275956    0.519389958120883    0.519542030058801
##                    9                   10                   12
##    0.519764196593314    0.520456686615944    0.520963337272406
##                    8                    8                   14
```

```
##    0.520967609243371    0.521294551901519    0.522910927422345
##                    8                   10                    6
##    0.524279766716063    0.525622042361647    0.525845746509731
##                    9                    7                   11
##     0.52918558428064    0.535107435658574    0.537299547344446
##                   14                    8                    8
##    0.540480333380401    0.542676644865423    0.543395972810686
##                   13                   10                    9
##    0.544606171082705    0.544946365058422    0.547060955315828
##                    9                    9                    7
##    0.547176248859614    0.550413612276316    0.552079930511321
##                   25                   10                   10
##     0.55215441249311    0.552562096621841    0.553404209204018
##                   10                    8                   14
##    0.553744060453027    0.554568774998188    0.556336301844567
##                   16                   16                    8
##    0.558760480955243    0.560128264112986    0.560601223725826
##                    8                   10                   10
##    0.562340235337615    0.562701905611902    0.562717433087528
##                   10                   10                   14
##    0.563564176671207    0.563695745542645    0.564987863879651
##                    7                    3                    9
##    0.565743230748922    0.565798300123646    0.566340053919703
##                   12                   11                    4
##    0.566753161605448    0.569598302307014    0.570058441720903
##                    9                   11                   17
##    0.570185162127018    0.570878281723708    0.571037979796529
##                   10                    9                   19
##    0.571163111366332    0.574151284992695    0.574524201147067
##                    8                    5                   10
##    0.575776313897222    0.578148595523089    0.581274567637593
##                    5                   11                   18
##    0.582711852155626    0.583959034178406    0.586031898390502
##                   12                    7                   11
##    0.586180571932346    0.590567151052186    0.596245939843357
##                    7                    6                   11
##    0.599043748807162     0.59987036138773    0.600290522910655
##                    8                    8                    7
##    0.603123138193041    0.608745950274169    0.611335589651825
##                    8                    9                    8
##     0.61464109364897    0.616616424638778    0.618775254581124
##                    9                   13                    6
##    0.619479048065841     0.62029971042648    0.622640496119857
##                    8                   11                    8
##    0.624185904860497    0.625006453709391    0.626160759944469
##                   10                   11                   15
##    0.626344923861325    0.629024119116366    0.630111274775118
##                   11                    5                   12
##    0.633233909029514    0.633692207746208    0.634635034482926
##                   15                    9                   11
##    0.636938420590013    0.639873905484328    0.642375284805894
##                    9                   15                    7
##    0.646904501598328    0.648160624783486    0.648428855929524
##                   13                    8                   14
##    0.650263628456742    0.652123309206218    0.652190891560167
##                    8                    2                   11
```

```
##    0.653622725512832    0.653756458777934    0.653972766827792
##                   11                    9                   15
##    0.656341042369604    0.658565026242286    0.668802330736071
##                   14                    8                   14
##    0.674756390973926    0.677205930929631    0.678234168328345
##                   10                   17                    7
##    0.678334814030677    0.680991292465478    0.681109356693923
##                   14                   12                    9
##     0.68318208027631    0.683421638794243    0.684479780029505
##                   11                   15                   12
##    0.684651835821569     0.68540078913793    0.687138519249856
##                   12                   10                   15
##     0.68786188820377    0.688727342523634    0.690513669513166
##                    7                    7                   16
##    0.693491754080069    0.699216096357559    0.699990783718444
##                    5                    9                    9
##    0.700371202970755     0.70346904325133    0.705471359294399
##                    9                    9                    7
##    0.709684747229231    0.710802437781561    0.711012404716183
##                   15                   10                   10
##    0.712405278305423    0.717323950987112    0.721699715620309
##                   13                   17                   10
##    0.723261550026846    0.724731202195299    0.725071551316689
##                   11                    6                    9
##    0.734219817402673      0.7342659337026    0.743009093303527
##                    9                    9                   14
##    0.747754333050547    0.750062389524767    0.750197290400875
##                    4                    6                    9
##    0.750275880159221    0.751007478613387    0.754960815380228
##                    7                    8                   11
##    0.761974239055169    0.762715874260103    0.766181896398676
##                   12                    8                    7
##     0.76690556885367    0.773841689870853    0.774531252299939
##                   11                   12                   11
##    0.775437032623874    0.779140545944584     0.78290218110652
##                   12                   10                   12
##    0.783020584802044    0.783095770473492    0.784072867079109
##                   11                    9                   14
##    0.789053964386594    0.794767496349833    0.794974112913083
##                    8                   11                    5
##     0.79527001358093    0.797239866065693    0.801048337439907
##                   13                    4                    9
##    0.804568271069398    0.805131560486849    0.807597521650505
##                   12                    6                    9
##    0.808501807016231    0.811824683989299    0.815074712917429
##                    5                   14                    9
##    0.815178421436096    0.818157198857859    0.821212697732163
##                   10                   10                   15
##    0.823461614507151    0.823729471645176    0.828492210204613
##                   15                    8                    5
##    0.828931353944254    0.833509142468584    0.833822812843037
##                   11                    9                    9
##    0.835119620260095     0.83835085849337    0.841421321829482
##                    8                   13                   11
##    0.844626462186803    0.845553428445311    0.845764080020499
##                    2                   10                    7
```

```
##    0.845928052737793    0.852979899342072    0.861565184875937
##                    8                    9                   13
##     0.86275438587637    0.863376310053182    0.863437987822008
##                    8                   13                   14
##    0.865337749841733    0.866169034270905    0.866922731469286
##                   11                   11                    9
##    0.870033309752953    0.871918298262072    0.876241016065841
##                   16                    9                   14
##    0.876830756794376    0.877517884555173    0.877690055365575
##                   11                    9                    9
##    0.880696502472174    0.888936137073373    0.889539845245791
##                   10                   11                   11
##    0.892106865437114    0.893557431134296    0.894295406288895
##                    9                   10                   12
##    0.894404234936759    0.899129344374405    0.899204490170204
##                   12                   12                    6
##    0.899724709759367    0.900446481875089    0.901973554822517
##                    8                    3                   16
##    0.904190784618571    0.906066957819176    0.908295678181848
##                   10                   15                   10
##    0.908918043556524    0.912152412305486    0.917054740670037
##                   11                   10                   11
##    0.917924849655622    0.920637245279164    0.924097659955633
##                    8                    7                    9
##    0.925473132422238    0.925519017460656    0.927205992820573
##                   14                    7                   12
##     0.92951761554451    0.936687619383347     0.93873042609067
##                   12                    9                    9
##    0.940853076944959    0.941531873266113     0.94260788239063
##                   11                   15                    8
##    0.943741014570833    0.954269530782839    0.954989121253503
##                    9                   19                   10
##    0.958134080241335     0.95832603020057    0.959164231973601
##                    7                    7                    6
##    0.965542987544484    0.967381149488402    0.968567667486978
##                   12                   12                   12
##    0.970349967437697     0.98116380292217    0.982718887696874
##                   10                    9                    9
##    0.987982849548471    0.993462570639078     0.99597588270427
##                   17                    9                    7
##    0.996453186455083     1.00523375185796     1.00649201493235
##                    5                   10                    7
##     1.01139826043965     1.02114295955272     1.02307908326488
##                    7                    8                   12
##     1.02336914683068     1.03248332813764     1.04425825544613
##                   13                   11                    7
##     1.04504753924431     1.04576484079342      1.0531578650278
##                    7                    7                   11
##     1.05560367550225      1.0570138496858      1.0587652734992
##                   12                    8                   14
##     1.06678481352671     1.08574092756481      1.0859916562956
##                   16                    4                   11
##     1.08672035201908     1.08765442478926      1.0906232734722
##                    5                    4                   11
##     1.09253440979393     1.09576100199939      1.0970533271421
##                   13                   10                    6
```

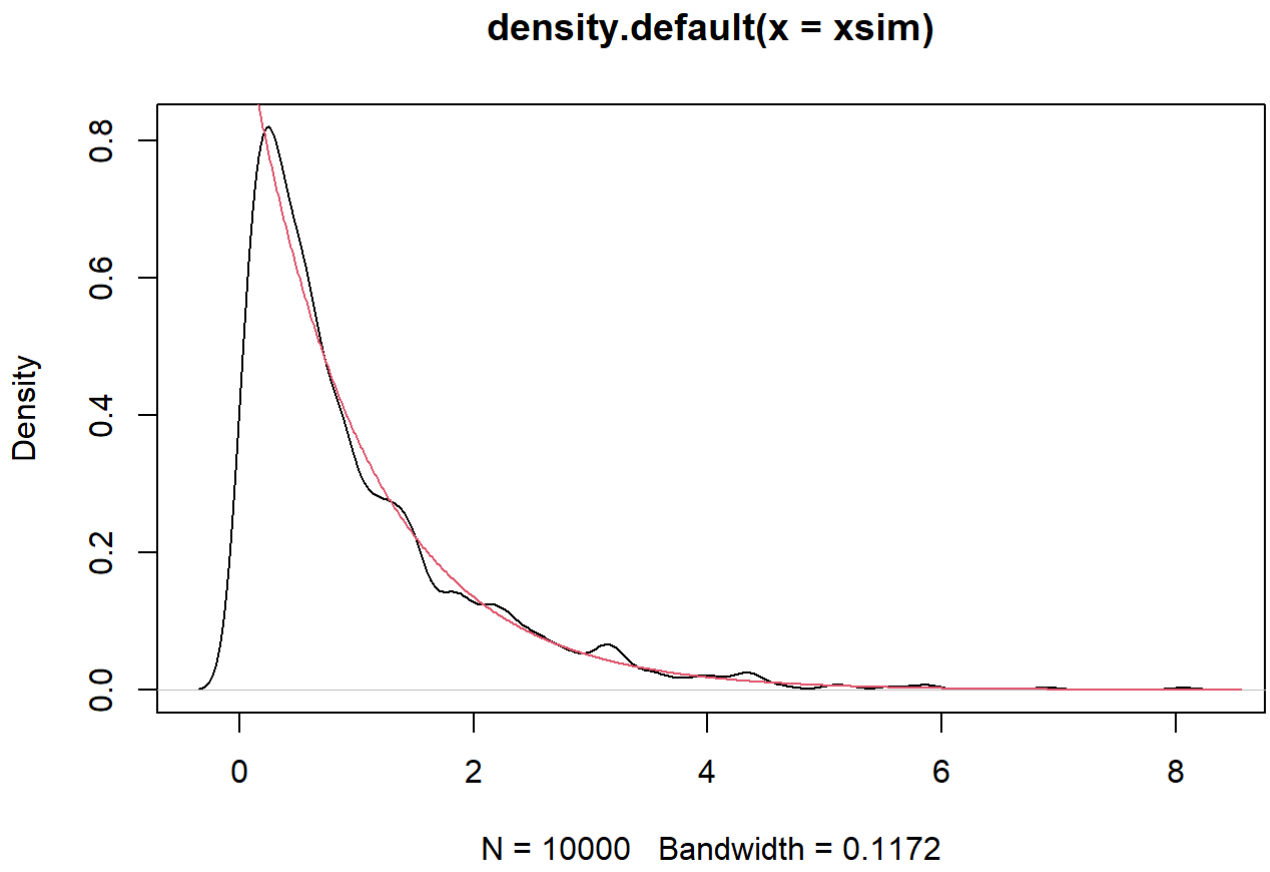
```
##     1.09740187695958     1.10030627871908     1.10368799139418
##                   10                   15                   15
##     1.10779943480438     1.11011646594874     1.11291040941597
##                   14                    7                   11
##     1.11625855691255     1.11934329565139     1.12320606526545
##                    7                   10                   13
##     1.12544713165135      1.1266936598432     1.12966897800089
##                   11                    7                   12
##     1.13024065825613     1.13244730136813     1.13303027111561
##                    6                    5                    9
##     1.14231541593932     1.14357230711365     1.14540855415636
##                    3                    9                   13
##     1.14610811784388     1.14866496618362     1.15110172979058
##                    6                   13                   10
##     1.15227581463279     1.15879201832247     1.16039950488062
##                   14                   14                   11
##       1.163647538009     1.16398594524236     1.16591396778458
##                    9                   13                   15
##     1.16985932919652     1.17258274756434     1.18412832539053
##                    9                    8                    6
##     1.20721573285283     1.20762932110102     1.21061453591557
##                    8                   10                   12
##     1.21495383414896     1.21628481518002     1.22086901492895
##                    7                   11                    8
##     1.22572063743593     1.23007517024399     1.23185111298682
##                   11                    8                    8
##     1.23397244788231     1.23572183303059     1.23881661142678
##                   12                    8                   11
##     1.24418086461247     1.25414226479353     1.25433078497173
##                   11                   15                    5
##     1.25727379410983     1.26069618571074     1.26173347975583
##                    9                    7                   12
##     1.26331722002957     1.26344728279383     1.26924417349966
##                   11                   11                   16
##     1.27170608095349      1.2728728185919     1.27359780478747
##                   10                    9                   11
##     1.28070901330085     1.28147668107868      1.2833238383335
##                    9                    8                   13
##     1.28689922853829     1.29219914037259     1.29301549583198
##                   11                    8                   16
##     1.29729819852383     1.29966516974838     1.32812948148222
##                   16                    5                    9
##     1.32820989746125     1.32913678414644     1.33306633676858
##                   10                   11                   10
##     1.33506576433302     1.33509115870566     1.34016233026506
##                   15                    7                    9
##     1.34089173838021     1.34118037946047     1.34358834750475
##                   12                   16                   14
##     1.35036979052694     1.35113190762223     1.35629088334264
##                   11                   15                   13
##     1.35718415673347     1.35726634129406     1.36066991552235
##                    6                   10                    8
##     1.37039164822073     1.37365399673579      1.3922970573509
##                   11                   11                    9
##     1.39343501432779     1.40362098306044     1.40743771373274
##                   21                   13                    9
```

```
##     1.40767991875978     1.40881388019287     1.41438566575614
##                    7                    6                    4
##     1.42536024356974      1.4357419767197     1.43843809802728
##                   12                    9                   14
##     1.44141640484775     1.44316489975752     1.44402970365229
##                   11                   12                    7
##     1.44946970348676     1.45292758595122      1.4552465127344
##                   11                    7                   10
##     1.45649912013652     1.45813848600403     1.46229654542151
##                   12                   10                   19
##     1.46428551661393     1.46562677791818     1.46615919518048
##                    7                    8                    3
##     1.46718516564908     1.46925610659996     1.47475307798924
##                   13                   11                    9
##     1.47578267215262     1.47747886172476     1.48553413169728
##                    8                   17                   17
##      1.4862920421865     1.48725451520624     1.48978787465932
##                   10                    9                    9
##     1.49019951551201     1.49338533534648     1.49642720623853
##                   10                    6                    6
##     1.50422433755341     1.50986113390865      1.5214527337028
##                   10                    7                    7
##     1.52560607370066     1.52708251579645     1.52995488970521
##                   13                    9                   10
##     1.53160057767274     1.53411781034412      1.5386789316506
##                    6                   10                    5
##     1.54051542087438     1.54170666812899     1.54691560938068
##                   13                   10                    6
##     1.55346736489775      1.5575226464519     1.55879247770941
##                    5                    9                   13
##     1.56745750929551     1.57073905869367     1.59346325197297
##                   13                    9                    6
##     1.59816797628048     1.61049548900666     1.61089385501299
##                    9                    8                    7
##     1.62668428189101     1.62862567290062     1.62879523897215
##                   11                   11                   10
##     1.63039499326589     1.66879247306826     1.67727193131195
##                   15                    6                   11
##     1.67809204935076     1.67987035206499     1.69132376003002
##                   13                    9                    6
##     1.69996085382047     1.70147425486686     1.71130561261418
##                    8                   10                   12
##     1.73320792420747     1.73412686577978     1.74059893182459
##                   10                    4                   13
##     1.74629842295291     1.74949401049855     1.76993144108894
##                    6                    5                    9
##     1.77162484816256     1.77196330759601     1.78280059612336
##                    9                   10                   13
##     1.78673862098696     1.79111592276933     1.79473600341919
##                   16                   12                    9
##     1.79816801651361     1.79877706236186      1.8144333850685
##                    7                   11                    7
##     1.82208775792558     1.82334659798982     1.82824082291665
##                   11                    8                   14
##      1.8316667395735     1.83383821501377     1.83572234577479
##                    9                   11                    6
```

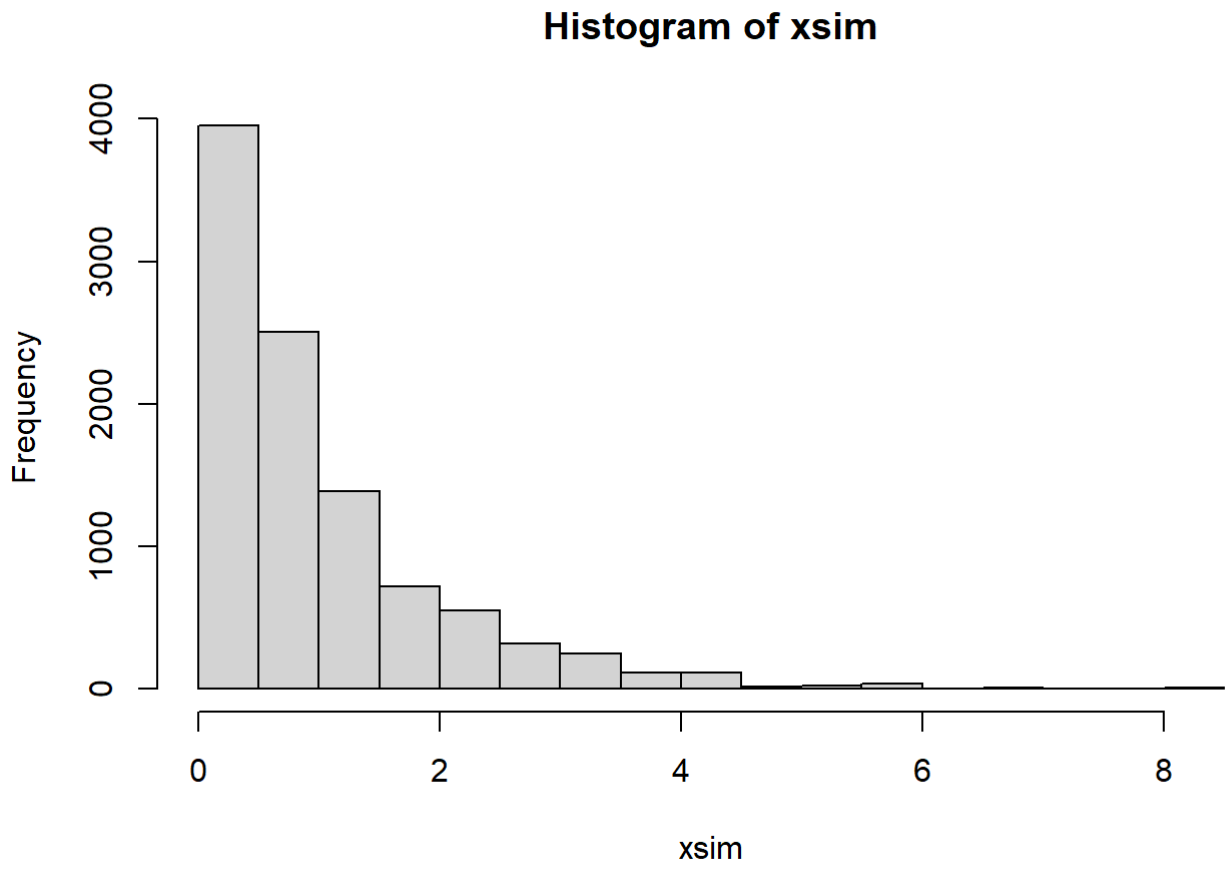
```
##     1.86123500324041      1.8730506504977      1.8774539361799
##                   10                   10                   12
##     1.87872512443903     1.88648266150596     1.88920352227035
##                   10                    5                    9
##     1.88946031584384     1.89677002458813      1.8976469746882
##                   11                   21                    9
##     1.91178729357328     1.93451458766105     1.95128729186001
##                   13                   14                   17
##     1.95550976685228     1.96227084091845     1.96316196202281
##                   14                    8                   12
##     1.96357019050958     1.99014565712752     1.99512521355393
##                   10                   14                    6
##     1.99909551060739     2.00216150647941     2.01431691824439
##                    7                   12                    8
##     2.01560030682924     2.02413065909388     2.02843917897883
##                    7                   14                   10
##     2.04141021138969     2.05981996088269     2.08660192729305
##                    8                    9                   11
##     2.09862637693498      2.1018023471019     2.11260818863723
##                    9                    9                    7
##     2.11509477893326     2.12202176454779     2.12757934589209
##                    7                   12                   14
##     2.12806496931146     2.13861123273585     2.15332760090444
##                    9                   22                    9
##     2.15793194785761     2.17962831586683     2.18012156511161
##                    8                   14                   11
##     2.19790093401883     2.19880122865055     2.19892700771122
##                   10                   12                   10
##     2.20060420955036     2.22154335106227     2.22686640346859
##                   11                    9                    9
##      2.2340629416189     2.23783663461421     2.24489683099034
##                   14                   16                    3
##     2.24726377228711     2.25509092757914     2.26609204510663
##                   14                    7                    8
##     2.27257421041343     2.27605762225521     2.28017248527133
##                   15                    3                    9
##     2.29370425921014     2.30754672164533     2.30837672377441
##                    6                   12                   12
##     2.31596313778844     2.31640241468761     2.36135829539634
##                   14                   10                   12
##     2.36331135208223     2.36345524887059     2.38411195377324
##                    6                    7                    7
##      2.3856418969712     2.38761715019558     2.40394268428002
##                   10                    6                    8
##     2.40875326643202     2.41695438988315     2.44035560438845
##                    9                   11                    8
##     2.44980197200987     2.46038262347195     2.46761848933876
##                   10                    5                   11
##     2.47966304967616     2.48941057721589      2.4952148812494
##                    9                   10                    9
##     2.51267386028979       2.515170904432     2.51756365025852
##                   12                    6                    7
##     2.52732652414297      2.5385590213022     2.54176498005602
##                    7                    8                    5
##     2.56105033854458     2.57133982131693     2.57222744623635
##                    9                   10                    7
```

```
##     2.58104634458635     2.59039806748245     2.61973153325001
##                   13                    9                   13
##     2.62680279264662     2.64534638845466     2.64709184119914
##                    8                    4                   12
##      2.6505586298785     2.69380293031186     2.70152107799861
##                    9                    6                   14
##     2.72354084054444     2.73904318163607     2.74501322756026
##                   15                    5                    8
##     2.75311377073143     2.78626132739668     2.80742175860291
##                   11                   15                   14
##     2.83650933189278     2.84924210543142     2.85074815788245
##                   11                    7                    4
##     2.85328237184798     2.85669118149166     2.88647155427646
##                   12                   12                   13
##     2.95397093130236     2.95894574360978     2.97189930064564
##                    4                    9                    9
##     2.98865008337383     3.06465483486853     3.06706365569837
##                   11                    8                    8
##      3.0692036373723     3.06944789503421     3.07661846070887
##                    7                   14                   14
##     3.10795669609818     3.10834027809982     3.14145196212416
##                   11                    9                    9
##      3.1456730094202     3.14672774059658     3.14850050194149
##                   12                    5                   12
##     3.16466844810133     3.16698217181462     3.17093192769175
##                   14                    6                   15
##     3.19811002148275     3.20704871399288     3.26863268746023
##                   11                   11                    6
##     3.27460230870133     3.27462280524855     3.30664114637261
##                   12                   14                   11
##     3.32727185619955     3.34140089882498     3.40215720964556
##                   11                    8                    9
##     3.48406895273782     3.51032257399682     3.53156667689169
##                    9                    9                   11
##     3.54123561622059     3.54188104511654     3.64680514277375
##                   10                   13                    9
##     3.69619048179543      3.7339367175237     3.79205388368523
##                   11                    4                   10
##     3.89415817500508     3.91941971124566     3.92314735592759
##                   11                   11                    3
##     3.99270222128785     4.01467024327672     4.05557784975922
##                   10                   13                   12
##     4.13040867985642     4.25405915428825      4.2874721472425
##                    9                   17                   11
##     4.34753898286474     4.35093720604606     4.40425567676253
##                   13                   12                   12
##     4.42763078828575     4.42815872757336     4.64063586641975
##                   11                    6                   14
##      5.0514505372236     5.17842672922113      5.5760367956377
##                   12                   13                   14
##     5.86629007723103     5.88163875009785     6.88797972872298
##                   14                    7                   10
##     8.06243369868381
##                    9
```

```
plot(density(xsim)) # stima della densità
lines(avec, dexp(avec, 1), col = 2) # sovrapposizione di una densità esponenziale con tasso 1 sull'istogramma dei risultati
```




```
hist(xsim) # istogramma dei risultati
```



Calcoliamo

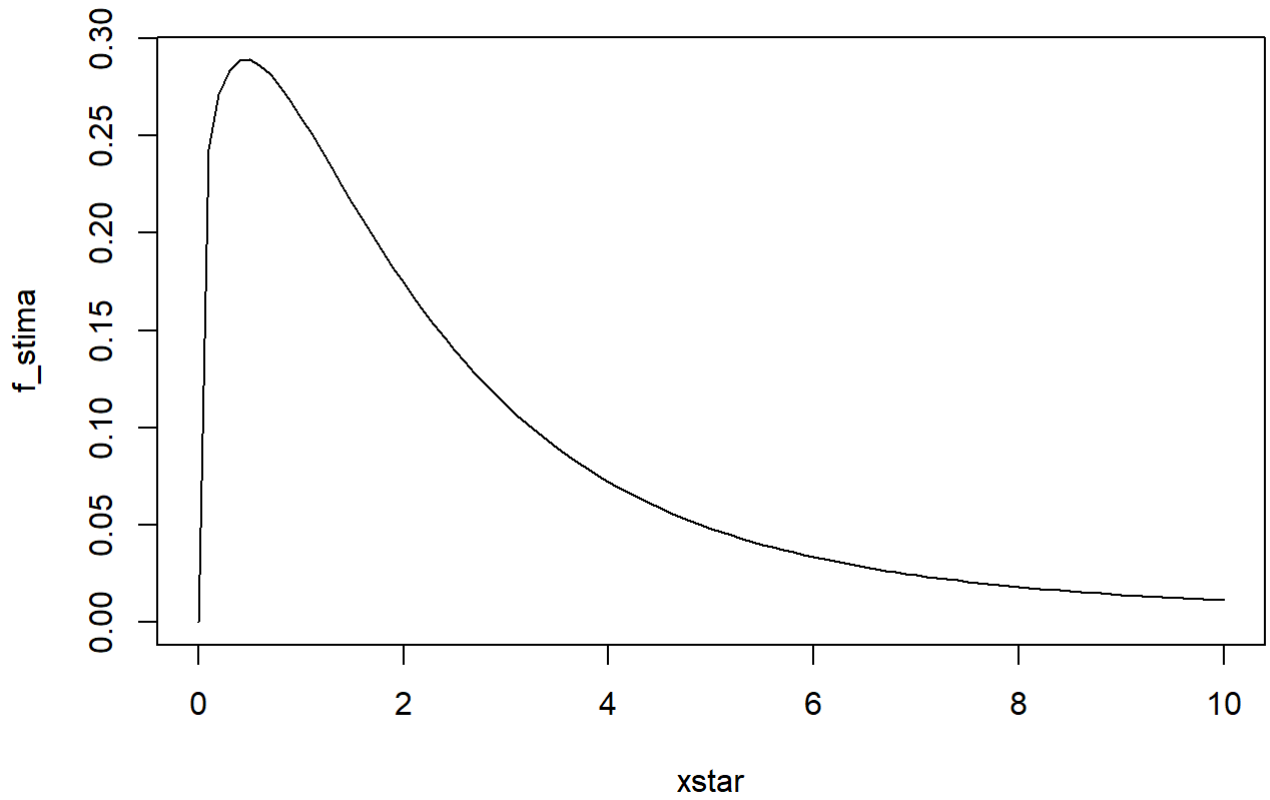
$$f(x^*) = \int_{\mathrm{R}^+} f(x \mid y) f(y) dy$$

con $Y \sim Exp(\lambda)$ e $X \mid Y = y \sim G(\exp y, 1)$ Lo stimiamo come

$$\frac{\sum f(x^* \mid y_i)}{n}$$

Il codice stima la funzione di densità di probabilità (PDF) di una variabile casuale distribuita secondo una distribuzione gamma, ma con i parametri della distribuzione (shape e scale) dati dal valore esponenziale di un campione di variabili casuali esponenziali.

```r
n = 10000 # numero di variabili casuali esponenziali generate
xstar = seq(0, 10, by = 0.1) #  sequenza di valori da 0 a 10 con passo 0.1 che verranno utilizzati come punti di valutazione per la PDF stimata.

f_stima = rep(NA,length(xstar)) # verrà utilizzato per memorizzare i risultati della stimata della PDF
y = rexp(n, 1)

for(i_x in 1:length(xstar)){
    xs = xstar[i_x]
    f_stima[i_x] = 0
    # Per ogni variabile casuale esponenziale generata (y), calcola il valore della PDF della distribuzione gamma al punto xs con i parametri shape e scale dati da exp(y)
    for(i in 1:n){
        f_stima[i_x] = f_stima[i_x] + dgamma(xs, exp(y[i]))
    }
    # Divide f_stima[i_x] per il numero di variabili casuali esponenziali generate (n).
    f_stima[i_x] = f_stima[i_x]/n
}
plot(xstar, f_stima, type = "l") # la stima della PDF
```



In sostanza, questo codice stima la PDF di una variabile casuale distribuita secondo una distribuzione gamma, dove i parametri della distribuzione sono ottenuti tramite il valore esponenziale di un campione di variabili casuali esponenziali.

ESEMPIO RAO-BLACKWELL

Voglio calcolare

$$E(h(X))$$

con $X \sim N(\mu_x, \sigma_x^2)$, potrei approssimarlo con

$$\frac{\sum h(x_i)}{n}$$

Il codice esegue un'approssimazione dell'aspettativa di una variabile casuale X tramite il metodo di Monte Carlo. In particolare, sembra stimare l'aspettativa di X utilizzando un campione casuale di dimensione nsim dalla distribuzione normale con media mu e deviazione standard sigma.

```
nsim = 1000 # Definisce il numero di campioni generati per l'approssimazione di Monte Carlo
mu = 0
sigma = 1

# Assumo h(X) = X
x = rnorm(nsim, mu, sigma) # Genera un campione di dimensione nsim da una distribuzione norma
le con media mu e deviazione standard sigma.

approx_mc = sum(x)/nsim # Calcola l'approssimazione dell'aspettativa di X come la somma di tu
tti i valori nel campione divisa per il numero di campioni (nsim).
approx_mc
```

```
## [1] 0.04684971
```

Assumiamo che $(X, Y)^T \sim N_2(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ con

$$\boldsymbol{\mu} = (\mu_x, \mu_y)^T$$

con $diag(\boldsymbol{\Sigma}) = (\sigma_x^2, \sigma_y^2)^T$ e l'extradiagonale è $\sigma_{x,y}$

Calcoliamo l'approssimazione

$$E(E(X|\,Y)) \approx \frac{\sum E(X|y_i)}{n}$$

il valore atteso condizionato è

$$E(X|y_i) = \mu_x + \sigma_{x,y}(\sigma_y^2)^{-1}(y_i - \mu_y)$$

e varianza condizionata

$$Var(X|y_i) = \sigma_x^2 - \sigma_{x,y}(\sigma_y^2)^{-1}\sigma_{x,y}^T$$

Il codice implementa una simulazione Monte Carlo per calcolare l'approssimazione condizionata di una media.

```
mu = c(0,2) # vettore di medie
Sigma = matrix(c(1, 0.9, 0.9, 1), ncol = 2) # matrice di covarianza con valori 1 sulla diagon
ale principale e 0.9 negli altri elementi

# Definizione di una funzione cond_mean
# La funzione prende un parametro y e calcola una media condizionata utilizzando i parametri
mu e Sigma. La formula è basata su una distribuzione normale condizionata.
cond_mean = function(y){
    mu[1]+Sigma[1,2]*(y-mu[2])/Sigma[2,2]
}

# Simuliamo y
# Viene generato un vettore y di dati casuali dalla distribuzione normale con media mu[2] e d
eviazione standard Sigma[2,2]^0.5.
y = rnorm(nsim, mu[2], Sigma[2,2]^0.5)
mean_cond_vec = c()
# Viene utilizzato un ciclo for per calcolare la media condizionata per ogni valore di y gene
rato.
for(i in 1:nsim){
    mean_cond_vec[i] = cond_mean(y[i]) # L'approssimazione Monte Carlo viene ottenuta calcola
ndo la media delle medie condizionata
}
approx_mc2 = sum(mean_cond_vec)/nsim #  l'approssimazione della media condizionata basata sul
la simulazione Monte Carlo
approx_mc2
```

```
## [1] -0.04481569
```

Il codice simula dati da una distribuzione normale, quindi calcola la media condizionata utilizzando una formula specifica e restituisce un'approssimazione Monte Carlo di questa media condizionata. La precisione dell'approssimazione dipenderà dalla dimensione del campione (nsim).

Approssimiamo le varianze di

$$Var(X) \approx \frac{\sum var(x_i)}{n}$$

e

$$Var(E(X|Y)) \approx \frac{\sum var(E(X|y_i))}{n}$$

con metodi Monte Carlo.

Una stima della varianza, se ho dei campioni z qualsiasi, viene dato da

$$\sum_{i=1}^{n} (z_i - \bar{z})^2 / n$$

Il codice è un esempio di simulazione Monte Carlo per valutare la varianza di stimatori in due situazioni:

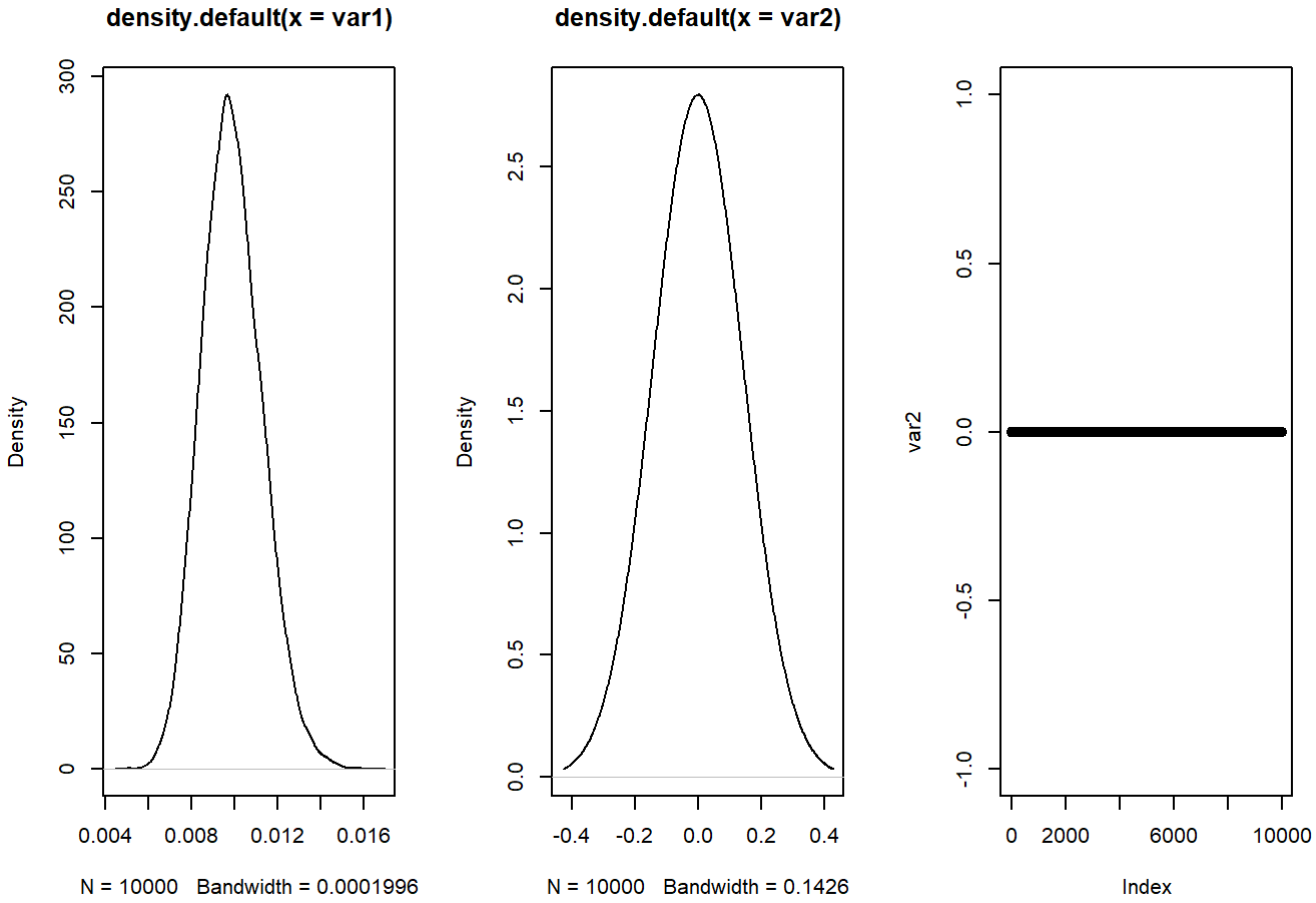
1. **Stima della Varianza per Variabili Marginali**. Viene generata una variabile casuale x_{marg} da una distribuzione normale standard $N(0, 1)$ con dimensione $n = 100$. Viene calcolato uno stimatore per la media di x_{marg}. Viene calcolata la varianza di questo stimatore.

2. **Stima della Varianza per Variabili Condizionali (Rao-Blackwellizando)**. Viene generata una variabile casuale y dalla parte marginale della seconda componente di una distribuzione normale multivariata con media $mu = c(0, 2)$ e matrice di covarianza $Sigma = matrix(c(1, 0.0, 0.0, 1), ncol = 2)$. Viene definita una funzione $cond_{mean}$ per calcolare la media condizionale della prima componente condizionata a y. Viene calcolato uno stimatore per la media condizionale. Viene calcolata la varianza di questo stimatore.

Vengono generati tre grafici: 1. La densità delle varianze ottenute per le variabili marginali (var1). 2. La densità delle varianze ottenute per le variabili condizionali (var2). 3. Un plot della varianza ottenuta per le variabili condizionali (var2).

Questo tipo di analisi può essere utilizzato per confrontare la precisione degli stimatori nelle due situazioni e verificare se l'approccio Rao-Blackwellizzante porta a una riduzione della varianza rispetto all'uso di variabili marginali.

```
var_mc1 = sum((x-mean(x))^2)/nsim^2 # calcola una stima della varianza di un vettore di dati
x
# Dallo script precedente, mean_cond_vec è la media condizionata per ogni valore di y generat
o
z = mean_cond_vec
var_mc2 = sum((z-mean(z))^2)/nsim^2 # calcola una stima della varianza di un vettore di dati
z

var1 = c()
var2 = c()
for(i in 1:10000){
    # Simuliamo dalla normali
    n = 100
    x_marg = rnorm(n, 0,1)

    # Stimatore
    mean(x_marg)
    # E la sua varianza
    sum( (x_marg-mean(x_marg))^2)/nsim^2

    # Facciamo la stessa cosa usando le normali multivariate e condizionando (Rao-Blackwelliz
ando)

    # settiamo i parametri della congiunta
    mu = c(0,2)
    Sigma = matrix(c(1,0.0,0.0,1), ncol=2)
    # Siamo interessati alla prima componente N(0,1) )

    # Funzione della media cond per calcolare la media condizionale della prima componente co
ndizionata a y
    cond_mean = function(y){
        mu[1]+Sigma[1,2]*(y-mu[2])/Sigma[2,2]
    }

    # Simulo dalla marginale della seconda componente
    y = rnorm(n, mu[2], Sigma[2,2]^0.5)

    # Calcolo lo stimatore
    mean(cond_mean(y))
    # E la sua varianza
    sum(  (cond_mean(y)-mean(cond_mean(y)))^2 )/n^2

    # Confronto le varianze
    var1[i] =  sum((x_marg-mean(x_marg))^2)/n^2
    var2[i] =  sum((cond_mean(y)-mean(cond_mean(y)))^2)/n^2
}
par(mfrow=c(1,3))
plot(density(var1))
plot(density(var2))
plot(var2)
```



Vediamo l'effetto della dipendenza. Il codice è un esempio di simulazione di variabili casuali da una distribuzione normale multivariata

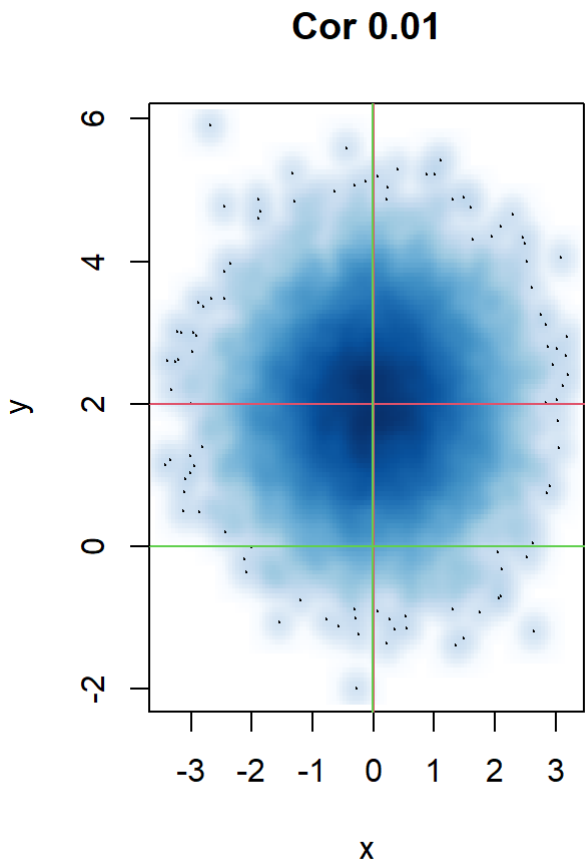
1. **Definizione della Funzione rmnorm**. *rmnorm* è una funzione che simula dati da una distribuzione normale multivariata. Gli argomenti sono il numero di osservazioni n, il vettore medio $mean$, e la matrice di covarianza $varcov$. Viene verificato il numero di colonne nella matrice di covarianza d, e poi viene generata una matrice z di numeri casuali da una distribuzione normale standard. La matrice z viene moltiplicata per la decomposizione di Cholesky della matrice di covarianza $chol(varcov)$. Il risultato viene trasformato per ottenere i dati della distribuzione normale multivariata richiesta.

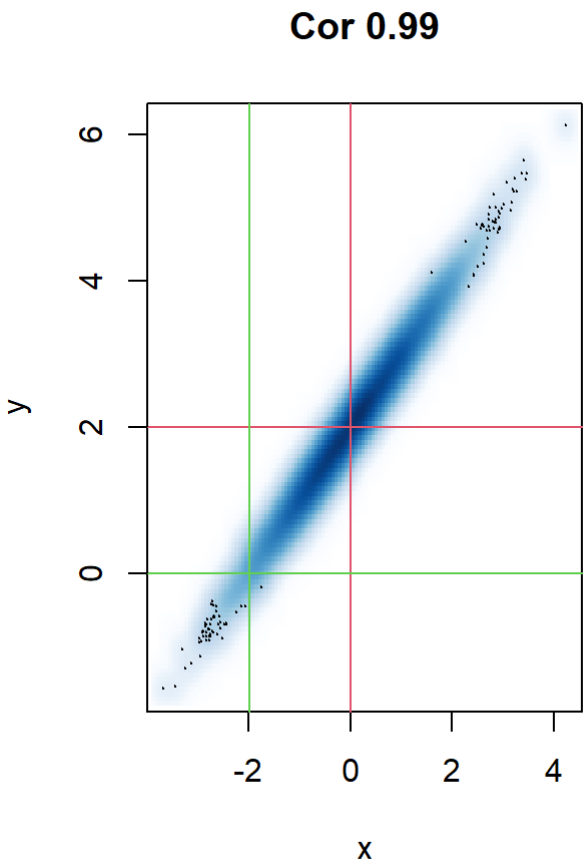
2. **Prima Simulazione con Correlazione 0.01**. Viene specificato un vettore medio mu_{sim} e una matrice di covarianza $Sigma_{sim}$ con correlazione 0.01. Viene utilizzata la funzione rmnorm per generare 10000 osservazioni da questa distribuzione. Viene creato uno *scatter plot* dei dati con smoothScatter. Vengono tracciate linee verticali *(abline(v= mu[1], col=2))* e orizzontali *(abline(h= mu[2], col=2))* alla posizione della media della distribuzione di riferimento mu. Vengono tracciate anche linee verticali *(abline(v = mu_sim[1]+Sigma_sim[1,2]*(0-mu_sim[2])/Sigma_sim[2,2], col=3))** e orizzontali *(abline(h = 0, col=3))* alla posizione della media condizionale quando $y = 0$.

3. **Seconda Simulazione con Correlazione 0.99**. Viene specificato un nuovo vettore medio mu_{sim} e una nuova matrice di covarianza $Sigma_{sim}$ con correlazione 0.99. Viene nuovamente utilizzata la funzione rmnorm per generare 10000 osservazioni da questa nuova distribuzione. Viene creato un secondo *scatter plot* dei dati con smoothScatter. Vengono tracciate le stesse linee di riferimento e le linee della media condizionale rispetto alla prima simulazione.

In entrambe le simulazioni, si sta esaminando l'effetto di variare la correlazione tra le variabili nella distribuzione normale multivariata, e si stanno confrontando le distribuzioni condizionali in base a diverse correlazioni. La funzione *cond_mean(0)* calcola la media condizionale della prima componente quando $y = 0$.

```r
# Definzione funzione rmnorm che simula dati da una distribuzione normale multivariata
rmnorm=function(n = 1, mean = rep(0, d), varcov){
  # Numero di colonne nella matrice di covarianza
  d <- if (is.matrix(varcov))
    ncol(varcov)
  else 1
  # Generazione matrice numeri casuali da una distribuzione normale standard per la decomposi
zione di Cholesky della matrice di covarianza
  z <- matrix(rnorm(n * d), n, d) %*% chol(varcov)
  y <- t(mean + t(z))
  return(y)
}

# Simulazione 1
simbi = 10000
mu_sim = c(0, 2)
Sigma_sim = matrix(c(1, 0.01, 0.01, 1), ncol = 2)
z = rmnorm(n = simbi, mean = mu_sim, Sigma_sim)

par(mfrow=c(1,2))
smoothScatter(z[,1],z[,2], pch=20, cex=.2, main= "Cor 0.01", xlab="x", ylab="y")
abline(v= mu[1], col=2)
abline(h= mu[2], col=2)
# Assumo y = 0
abline(h = 0, col=3)
abline(v = mu_sim[1]+Sigma_sim[1,2]*(0-mu_sim[2])/Sigma_sim[2,2], col=3)

# Simulazione 2
simbi = 10000
mu_sim = c(0,2)
Sigma_sim = matrix(c(1,0.99,0.99,1), ncol=2)
z = rmnorm(n = simbi, mean = mu_sim, Sigma_sim)

smoothScatter(z[,1],z[,2], pch=20, cex=.2, main="Cor 0.99", xlab="x", ylab="y")
abline(v= mu[1], col=2)
abline(h= mu[2], col=2)
# Assumo y = 0
abline(h = 0, col=3)
abline(v = mu_sim[1]+Sigma_sim[1,2]*(0-mu_sim[2])/Sigma_sim[2,2], col=3)
```





IMPORTANCE SAMPLING

Importance sampling per la media di una normal, scegliendo come g diverse t di student.

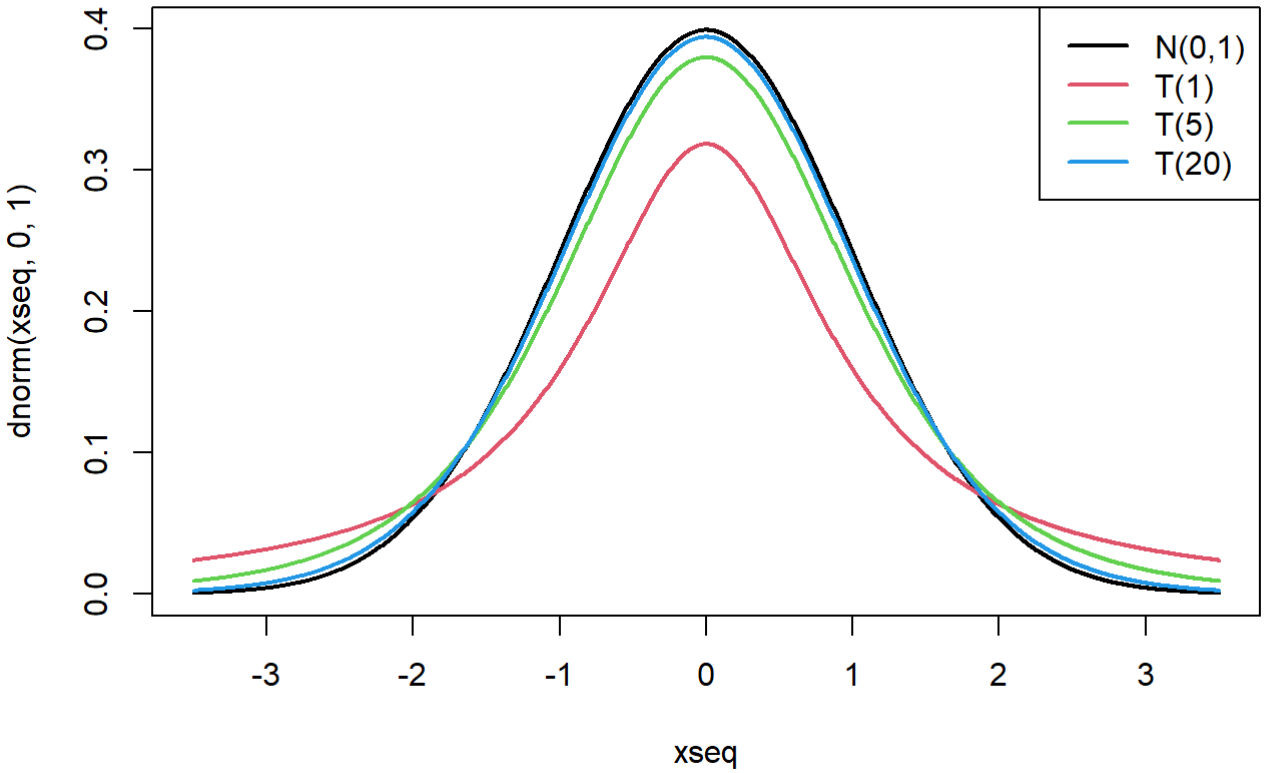
Il codice genera un grafico delle funzioni di densità di probabilità (pdf) della distribuzione normale standard $N(0, 1)$ e di diverse distribuzioni t di Student con diversi gradi di libertà. Successivamente, vengono simulate variabili casuali da queste distribuzioni t e vengono calcolati degli stimatori.

1. **Grafico delle Funzioni di Densità di Probabilità.**
    - `xseq` è una sequenza di valori da -3.5 a 3.5.
    - `dnorm(xseq, 0, 1)` calcola i valori della pdf della distribuzione normale standard per la sequenza `xseq` .
    - `dt(xseq, df)` calcola i valori della pdf della distribuzione t di Student con `df` gradi di libertà per la sequenza `xseq` .
    - Viene creato un grafico delle funzioni di densità di probabilità per la distribuzione normale standard e tre distribuzioni t di Student con diversi gradi di libertà (1, 5, 20).
    - La legenda mostra le etichette delle distribuzioni.
2. **Simulazione di Variabili Casuali e Calcolo degli Stimatori.**
    - Vengono simulate tre campioni di dimensione 1000 da distribuzioni t di Student con gradi di libertà 1, 5, e 20.
    - Gli stimatori m1, m2, e m3 vengono calcolati come il prodotto di ciascuna variabile casuale t con il rapporto tra la pdf della distribuzione normale standard e la pdf della distribuzione t.
    - Viene calcolata la media di ciascuno degli stimatori m1, m2, e m3.
    - Viene calcolata la varianza campionaria normalizzata di ciascuno degli stimatori (var(m1)/n, var(m2)/n, var(m3)/n), dove n è la dimensione del campione.

In generale, questo codice esplora il comportamento di stimatori basati su distribuzioni t di Student in confronto con la distribuzione normale standard. La varianza campionaria normalizzata fornisce una stima dell'errore quadratico medio degli stimatori. Il confronto può essere utile per comprendere come il cambiamento della

forma della distribuzione influisca sulla qualità degli stimatori.

```
xseq = seq(-3.5, 3.5,length.out = 1000)
# dnorm calcola i valori della pdf della distribuzione normale standard per la sequenza xseq
# dt calcola i valori della pdf della distribuzione t di Student con df gradi di libertà per la sequenza xseq
plot(xseq, dnorm(xseq, 0,1), col=1, lwd=2, type="l")
lines(xseq, dt(xseq,1), col=2, lwd=2)
lines(xseq, dt(xseq,5), col=3, lwd=2)
lines(xseq, dt(xseq,20), col=4, lwd=2)
legend("topright", c("N(0,1)", "T(1)", "T(5)", "T(20)"),col=1:4,lwd=2)
```




```
n = 1000
# Simulazione tre campioni da t di Student con diversi gradi di libertà
x_t1 = rt(n,1)
x_t5 = rt(n,5)
x_t20 = rt(n,20)

# Calcoliamo gli stimatori
m1 = x_t1*dnorm(x_t1,0,1)/dt(x_t1,1)
m2 = x_t5*dnorm(x_t5,0,1)/dt(x_t5,5)
m3 = x_t20*dnorm(x_t20,0,1)/dt(x_t20,20)

# Media
mean(m1)
```

```
## [1] -0.1116139
```

```
mean(m2)
```

```
## [1] 0.004627629
```

```
mean(m3)
```

```
## [1] 0.04063783
```

```
# Realizzazione dell'approssimazione della Var(X)
var(m1)/n
```

```
## [1] 0.001136346
```

```
var(m2)/n
```

```
## [1] 0.0009778513
```

```
var(m3)/n
```

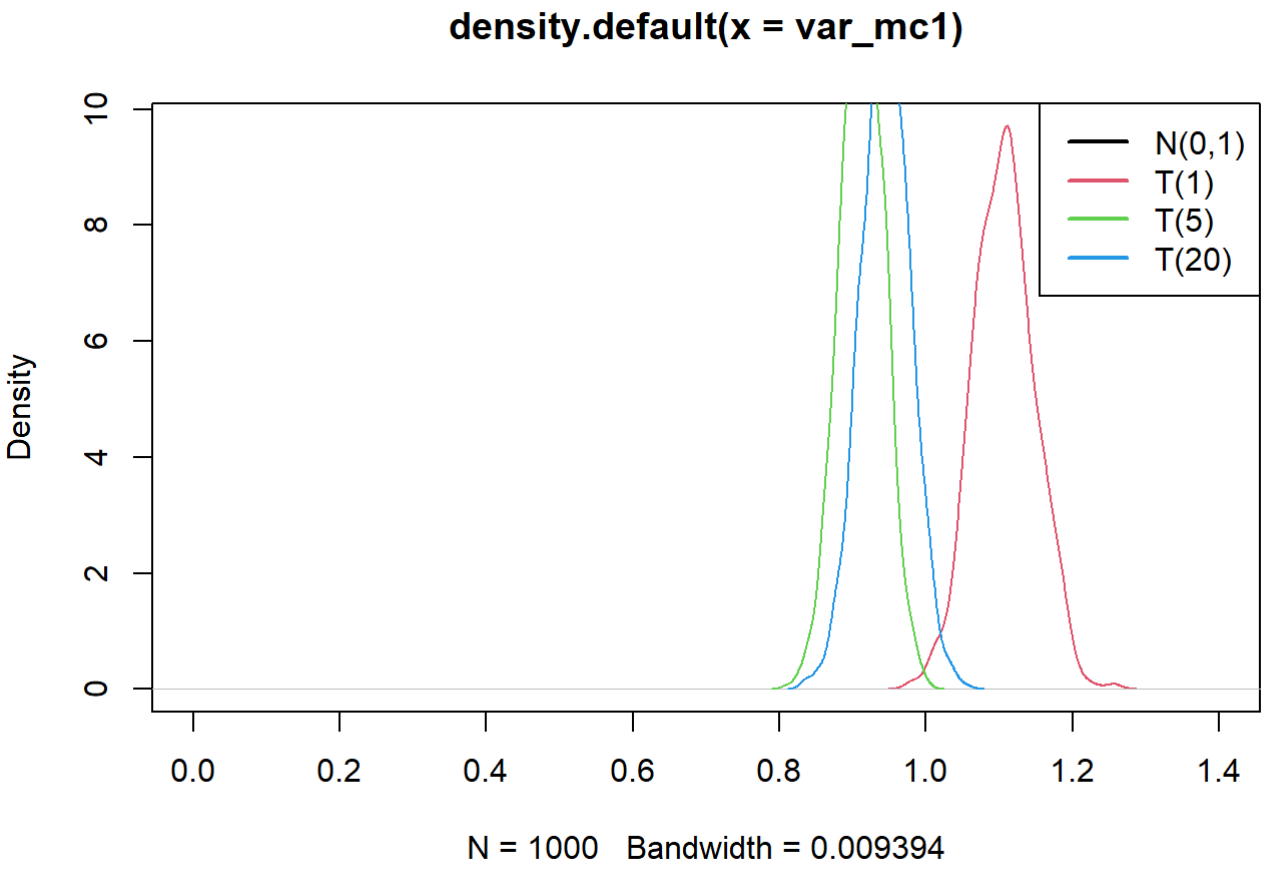
```
## [1] 0.0009502816
```

Se voglio stimare la varianza, posso fare una simulazione. Il codice esegue una simulazione Monte Carlo per valutare la varianza degli stimatori calcolati da distribuzioni t di Student con diversi gradi di libertà. 1. **Ciclo For** per ogni iterazione del ciclo (isim da 1 a nvar): - Viene generato un campione da una distribuzione t di Student con gradi di libertà 1, 5, e 20 (x_t1, x_t5, x_t20) - Vengono calcolati gli stimatori m1, m2, e m3 usando la stessa logica descritta nel codice precedente. - Le varianze degli stimatori vengono calcolate e salvate nei rispettivi vettori var_mc1, var_mc2, e var_mc3.

2. **Grafico delle Densità delle Varianze degli Stimatori**. Viene creato un grafico delle densità delle varianze degli stimatori ottenuti nelle simulazioni per le distribuzioni normale standard (blu), t di Student con df=1 (verde), t di Student con df=5 (rosso) e t di Student con df=20 (viola).

Il grafico finale fornisce una rappresentazione visiva delle densità delle varianze degli stimatori per ciascuna distribuzione in considerazione. Questo può essere utile per confrontare la precisione degli stimatori nelle diverse distribuzioni t di Student con diversi gradi di libertà.

```r
nvar = 1000 # Numero di simulazioni da eseguire

# Vettori in cui vengono salvate le varianze degli stimatori ottenuti in ciascuna simulazione
var_mc1 = c()
var_mc2 = c()
var_mc3 = c()

for(isim in 1:nvar){
    n = 1000
    # Generazione del campione
    x_t1 = rt(n,1)
    x_t5 = rt(n,5)
    x_t20 = rt(n,20)

    # Calcolo degli stimatori
    m1 = x_t1*dnorm(x_t1,0,1)/dt(x_t1,1)
    m2 = x_t5*dnorm(x_t5,0,1)/dt(x_t5,5)
    m3 = x_t20*dnorm(x_t20,0,1)/dt(x_t20,20)

    # Calcolo varianze degli stimatori
    var_mc1[isim] = var(m1)
    var_mc2[isim] = var(m2)
    var_mc3[isim] = var(m3)
}

# Calcolo degli Stimatori Medi
stimatore1 = mean(var_mc1)
stimatore2 = mean(var_mc2)
stimatore3 = mean(var_mc3)

# Grafico delle Densità delle Varianze degli Stimatori
plot(density(var_mc1), col=2, xlim=c(0,1.4))
lines(density(var_mc2), col=3)
lines(density(var_mc3), col=4)

legend("topright", c("N(0,1)", "T(1)", "T(5)", "T(20)"),col=1:4,lwd=2)
```

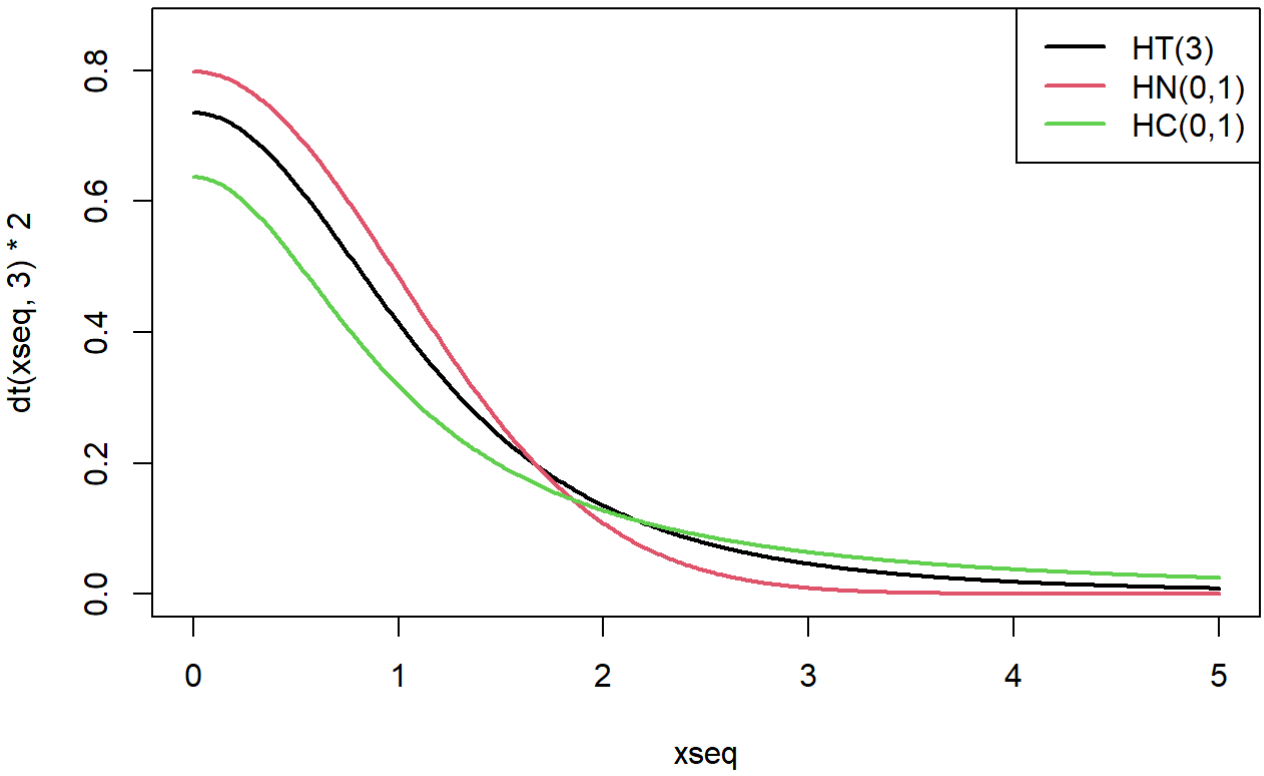


Vediamo un caso un po' più complesso. Il codice esegue una simulazione Monte Carlo per confrontare gli stimatori della varianza di una distribuzione half t(3) (definita solo sui positivi) rispetto a due distribuzioni di riferimento: half normal(0,1) e half cauchy(0,1).

1. **Grafico delle Densità**
   - Viene creato un grafico delle densità di probabilità per una half t(3) (HT(3)), una half normal(0,1) (HN(0,1)), e una half cauchy(0,1) (HC(0,1)).
   - La densità di HT(3) viene moltiplicata per 2 per confrontarla meglio con le altre due distribuzioni.
2. **Simulazione Monte Carlo.**
   - Vengono eseguite 10000 simulazioni.
   - Per ciascuna simulazione:
     - `x_norm` viene generato da una distribuzione normale standard.
     - `x_cauchy` viene generato da una distribuzione cauchy standard.
     - Vengono calcolati gli stimatori della varianza come la media di `(|x| * f(x))/g(x)` , dove `f(x)` è la densità di HT(3) e `g(x)` è la densità di riferimento (dnorm per `x_norm` e dt con df=1 per `x_cauchy`).
     - I risultati vengono salvati nella struttura dati `results` .
3. **Calcolo delle Medie e delle Varianze.**
   - Vengono calcolate le medie degli stimatori per ciascuna distribuzione di riferimento (`apply(results, 2, mean)`).
   - Viene calcolata la varianza campionaria normalizzata degli stimatori per ciascuna distribuzione di riferimento (`var`).
4. **Stampa delle Varianze.**
   - Vengono stampate le varianze campionarie normalizzate degli stimatori.

In generale, il codice esplora la qualità degli stimatori della varianza per la distribuzione half t(3) rispetto a due distribuzioni di riferimento. La varianza campionaria normalizzata degli stimatori fornisce una misura della precisione degli stimatori.

```r
# Voglaimo simulare da una half t(3) (t distribution definita solo sui positivi) usando come
g uno half normal(0,1), e half cauchy(0,1) (una half t(1))

xseq = seq(0, 5, by = 0.01)
# Grafico delle Densità
plot(xseq,dt(xseq, 3)*2, col=1, lwd=2, type="l",ylim= c(0,0.43*2))
lines(xseq, dnorm(xseq)*2, col=2, lwd=2)
lines(xseq, dt(xseq,1)*2, col=3, lwd=2)
legend("topright", c("HT(3)", "HN(0,1)", "HC(0,1)"), col=1:4, lwd=rep(2,4), lty=rep(1,4))
```




```r
# Simulazione Monte Carlo:
nsim = 10000
n = 100
results = data.frame(list(normal=rep(0,nsim), Cauchy=rep(0,nsim), t3=rep(0,nsim)))

for (i in 1:nsim) {
   x_norm = rnorm(n) # generato da una distribuzione normale standard
   x_cauchy = rt(n, 1) # generato da una distribuzione cauchy standard

   # Stimatori della varianza
   results[i,1] = mean(abs(x_norm)*dt(x_norm, df=3)/dnorm(x_norm))
   results[i,2] = mean(abs(x_cauchy)*dt(x_cauchy, df=3)/dt(x_cauchy, df=1))
   results[i,3] = mean(abs(rt(100, df=3)))
}

# Calcolo delle Medie e delle Varianze
apply(results,2,mean)
```

```
##   normal   Cauchy       t3
## 1.064958 1.102346 1.102830
```

```
var = c()
var[1] = sum((results[,1]-mean(results[,1]))^2)/n^2
var[2] = sum((results[,2]-mean(results[,2]))^2)/n^2
var[3] = sum((results[,3]-mean(results[,3]))^2)/n^2

# Stampa delle Varianze
print("Var")
```

```
## [1] "Var"
```

```
var
```

```
## [1] 71.591443273  0.005179006  0.017644787
```

STIMA DELLA VARIANZA

Assumiamo che $X \sim F(\boldsymbol{\theta})$ e sono interessato a

$$E(h(X))$$

che lo approssimo con

$$Y = \frac{\sum_{i=1}^{n} h(X_i)}{n}$$

con $X_i \sim F(\theta)$, iid. Abbiamo che $Y \sim H(\theta)$. Lo scopo è stimare la varianza

$$Var(Y) = \frac{Var(h(X))}{n}$$

La varianza è

$$Var(Q) = E(Q^2) - E^2(Q) \approx \frac{\sum Q_i^2}{n} - \left(\frac{\sum Q_i}{n}\right)^2 = \frac{\sum (Q_i - \bar{Q})^2}{n}$$

- **Metodo 1 - approssimo** $Var(h(X))$

$$Var(h(X))) \approx \frac{\sum (h(X_i) - \bar{h(X)})^2}{n}$$

e

$$Var(Y) = \frac{\frac{\sum (h(X_i) - \bar{h(X)})^2}{n}}{n}$$

Il codice genera un vettore x di lunghezza n da una distribuzione $G(p_1, p_2)$. Successivamente, calcola la media campionaria $mean_{mc}$ e la prima stima della varianza var_{mc1}.

```
# Metodo 1
n = 10 # Definisce la lunghezza del vettore x, ovvero il numero di campioni che verranno gene
rati dalla distribuzione gamma

# X ~ G(p1,p2)
p1 = 3 # Parametro di forma della distribuzione gamma
p2 = 0.3 # Parametro di tasso della distribuzione gamma

x = rgamma(n, p1, p2)

mean_mc = sum(x)/n # Calcola la media campionaria dei valori nel vettore x


# Prima stima varianza
var_x = sum((x-mean(x))^2)/n # Calcola la varianza campionaria di x
var_mc1 = var_x/n # Calcola una prima stima della varianza dividendo la varianza campionaria
per il numero di campioni n
```

In sintesi, il codice genera un campione di lunghezza n da una distribuzione gamma con parametri dati, quindi calcola la media campionaria e una prima stima della varianza del campione.

- **Metodo 2 - approssimo** $Var(Y)$. So che $Y \sim H(\theta)$, e assumiamo che ha una varianza e una media, finite. Se voglio approssimare la varianza di Y, devo calcolare

  $$Var(Y) \approx \frac{\sum_{j=1}^{m}(Y_j - \bar{Y})^2}{m}$$

  Ho bisogno di campioni iid $Y_j \sim H(\theta)$, che posso ottenere come

  $$Y_j = \frac{\sum_{i}^{n} h(X_{j,i})}{n}$$

  con $X_{j,i} \sim F(\theta)$ con $i = 1, \ldots, n$

Il codice effettua una seconda stima della varianza del campione mediante simulazione Monte Carlo.

```
# Metodo 2
msim = 10000  # Numero di simulazioni Monte Carlo
y = c() # Memorizzerà le medie campionarie dei campioni generati nelle simulazioni.

# Vengono generate nuove realizzazioni xj della distribuzione gamma. Per ciascuna realizzazio
ne, calcola la media campionaria e la salva nel vettore y
for (jsim in 1:msim) {
  xj = rgamma(n, p1, p2)  # Genera un nuovo campione dalla distribuzione gamma
  y[jsim] = mean(xj)  # Calcola la media campionaria del nuovo campione e la salva in un vett
ore
}

var_mc2 = sum((y - mean(y))^2) / msim  # Calcola la seconda stima della varianza utilizzando
le medie campionarie memorizzate nel vettore y
# Questa stima viene ottenuta sommando i quadrati delle deviazioni dalla media e dividendo pe
r il numero di simulazioni Monte Carlo
```

- **Metodo 3** Definisco

$$W = \frac{\frac{\sum (h(X_i) - \bar{h(X)})^2}{n}}{n}$$

che ha una distribuzione $W \sim L(\theta)$, e voglio approssimare

$$E(W)$$

con

$$\frac{\sum_{p=1}^{s} W_p}{s}$$

con

$$W_p = \frac{\frac{\sum (h(X_{p,i}) - \bar{h(X_p)})^2}{n}}{n}$$

Il codice implementa un terzo metodo per stimare la varianza del campione mediante simulazione Monte Carlo.

```
# Metodo 3
ssim = msim  # Numero di simulazioni Monte Carlo
w = c() # Verrà utilizzato per memorizzare i contributi individuali alla varianza calcolati i
n ogni simulazione

# In ogni iterazione, viene generato un nuovo campione dalla distribuzione gamma. Successivam
ente, viene calcolato il contributo individuale alla varianza, che è la somma dei quadrati de
lle deviazioni dalla media, diviso per n^2, e questo valore viene memorizzato nel vettore w
for (psim in 1:ssim) {
  xp = rgamma(n, p1, p2)  # Genera un nuovo campione dalla distribuzione gamma
  w[psim] = sum((xp - mean(xp))^2) / n^2  # Calcola e salva il contributo individuale alla va
rianza
}

var_mc3 = mean(w)  # Calcola la terza stima della varianza media
# Calcola la terza stima della varianza media prendendo la media dei contributi individuali a
lla varianza memorizzati nel vettore w

var_mc1
```

```
## [1] 4.529281
```

```
var_mc2
```

```
## [1] 3.395653
```

```
var_mc3
```

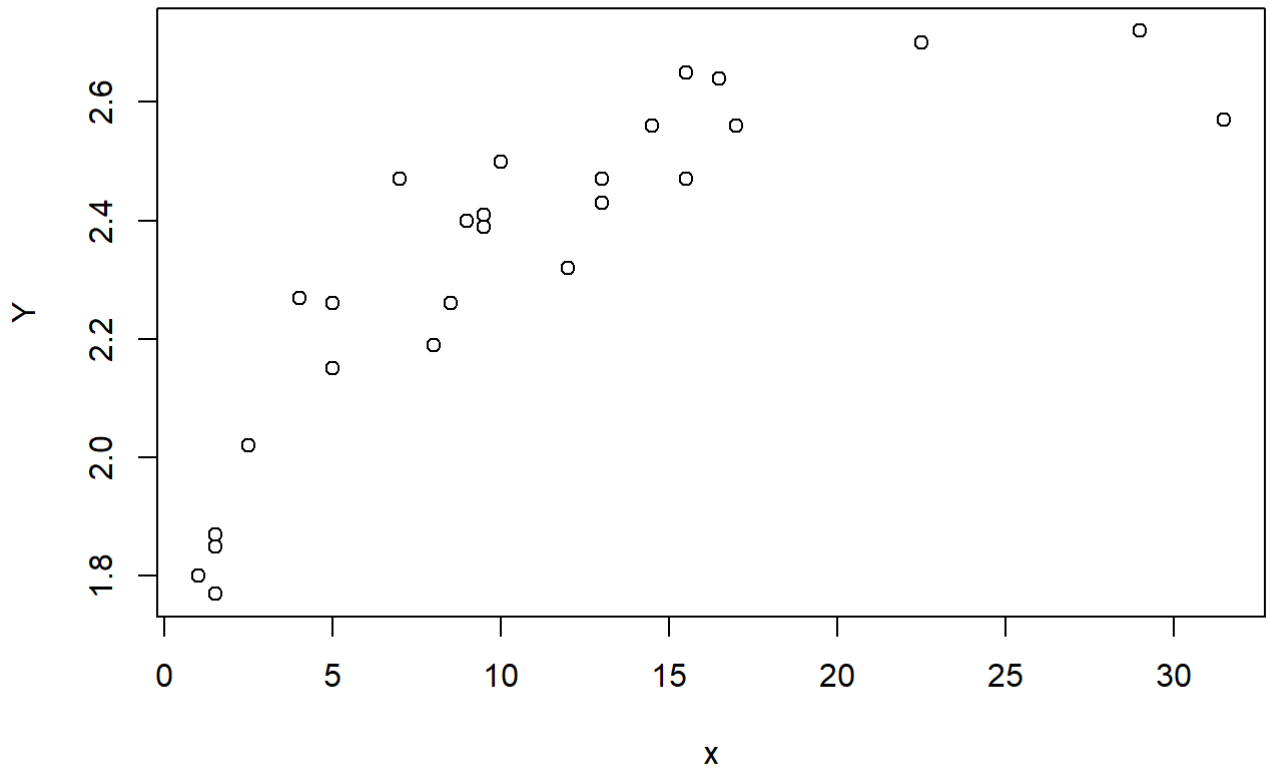
```
## [1] 3.009622
```

BOOTSTRAP

Descrizione del dataset. The data are length (Y) and age (x) measurements for 27 captured dugongs (seacows- mucche di mare). Vogliamo capire la relazione tra i due.

DATASET

```
x = c( 1.0,  1.5,  1.5,  1.5, 2.5,   4.0,  5.0,  5.0,  7.0,
               8.0,  8.5,  9.0,  9.5, 9.5,  10.0, 12.0, 12.0, 13.0,
              13.0, 14.5, 15.5, 15.5, 16.5, 17.0, 22.5, 29.0, 31.5)
# valori y
Y = c(1.80, 1.85, 1.87, 1.77, 2.02, 2.27, 2.15, 2.26, 2.47,
              2.19, 2.26, 2.40, 2.39, 2.41, 2.50, 2.32, 2.32, 2.43,
              2.47, 2.56, 2.65, 2.47, 2.64, 2.56, 2.70, 2.72, 2.57)


plot(x,Y)
```



Assumiamo che

$$Y_i \sim N(\alpha - \beta\gamma^{x_i}, \sigma^2)$$

con $\alpha \in \mathrm{R}^+$ $\beta \in \mathrm{R}^+$, $\gamma \in [0, 1]$. Potrei definire

$$z_i = \gamma^{x_i}$$

$$\lambda = -\beta$$

allora ho che

$$y_i = \alpha + \lambda z_i + \epsilon_i \quad \epsilon_i \sim N(0, \sigma^2)$$

quindi ho che

$$\hat{\alpha} = \bar{y} - \hat{\lambda}\bar{z}$$

$$\hat{\lambda} = \frac{\sum (z_i - \bar{z})(y_i - \bar{y})}{\sum (z_i - \bar{z})^2}$$

Se voglio trovare stimatore ML di γ devo minimizzare

$$\inf_\gamma \sum (y_i - (\alpha - \beta\gamma^{x_i}))^2 =$$

$$\inf_\gamma (\sum (y_i - \alpha)^2 + \sum (\beta\gamma^{x_i})^2 - 2\sum (y_i - \alpha)(\beta\gamma^{x_i}))$$

Siccome non so calcolarlo, faccio ottimizzazione creo la funzione da minimizzare.

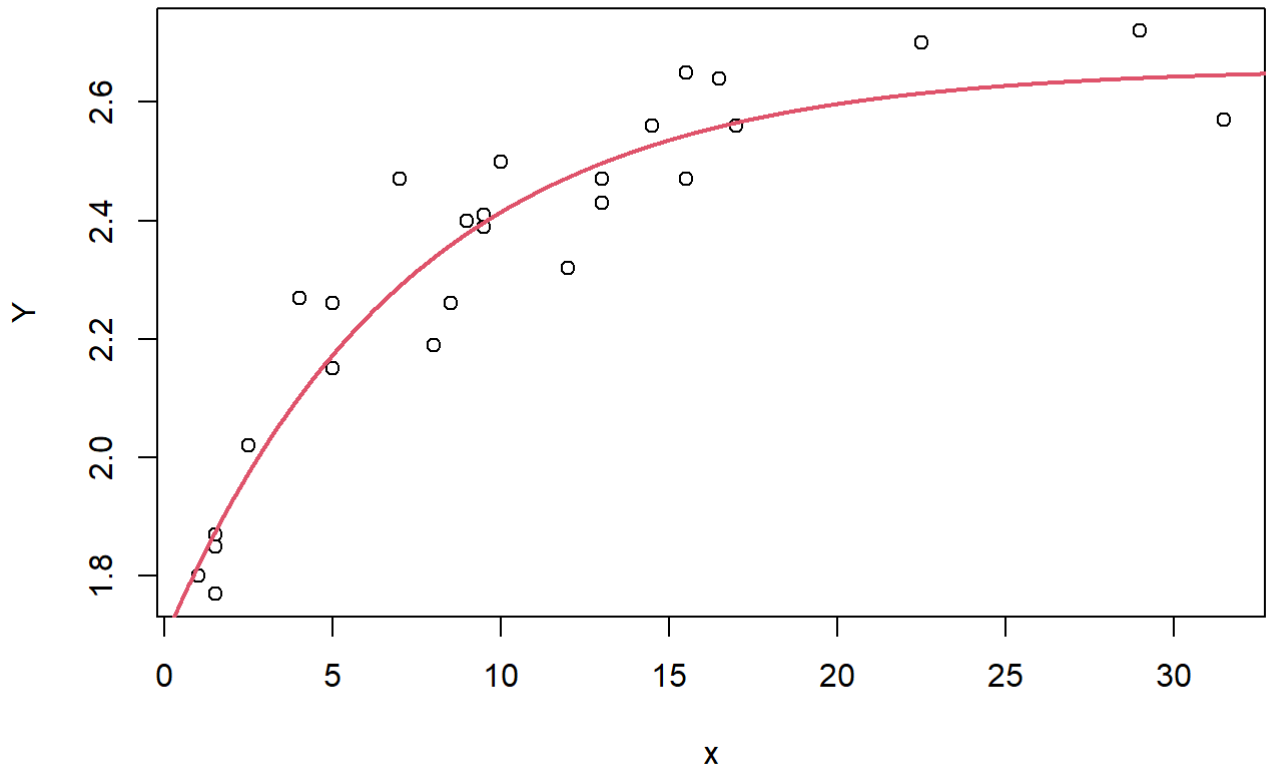
Il codice definisce una funzione **fff** che prende in input un vettore param e restituisce il negativo del logaritmo della funzione di verosimiglianza. Successivamente, utilizza la funzione **optim** per massimizzare questa funzione di verosimiglianza negativa e ottenere stime dei parametri del modello.

```
fff = function(param) {
  a = exp(param[1])
  b = exp(param[2])
  g = exp(param[3]) / (1 + exp(param[3]))
  s = exp(param[4])
  m = a - b * g^xb
  # La funzione di verosimiglianza negativa è calcolata come il negativo del logaritmo della funzione di densità di probabilità normale (dnorm). La somma di questi logaritmi è memorizzata nella variabile like
  like = sum(dnorm(Yb, m, s^0.5, log = TRUE))
  # La funzione restituisce il negativo della somma del logaritmo della funzione di verosimiglianza, poiché la funzione optim minimizza, non massimizza
  return(-like)
}

Yb = Y
xb = x

# Ottimizza la funzione di verosimiglianza negativa
param = optim(c(0, 0, 0, 0), fff, method = "BFGS")$par
# La funzione optim viene utilizzata per massimizzare la funzione di verosimiglianza negativa. I parametri stimati vengono poi estratti dalla parte optim(...)$par

# Estrai i parametri stimati
# I parametri stimati sono ottenuti esponenziando gli elementi del vettore param
a = exp(param[1])
b = exp(param[2])
g = exp(param[3]) / (1 + exp(param[3]))
s = exp(param[4])
```

In sintesi, questo codice esegue l'ottimizzazione dei parametri di un modello utilizzando la massimizzazione della verosimiglianza negativa e restituisce le stime dei parametri.

Vediamo se le stima hanno senso.

```
plot(x,Y)
fseq = seq(0, 40, by = 0.01)
lines(fseq, a-b*g^fseq, col=2, lwd=2)
```




```
a
```

```
## [1] 2.658058
```

```
b
```

```
## [1] 0.9635148
```

```
g
```

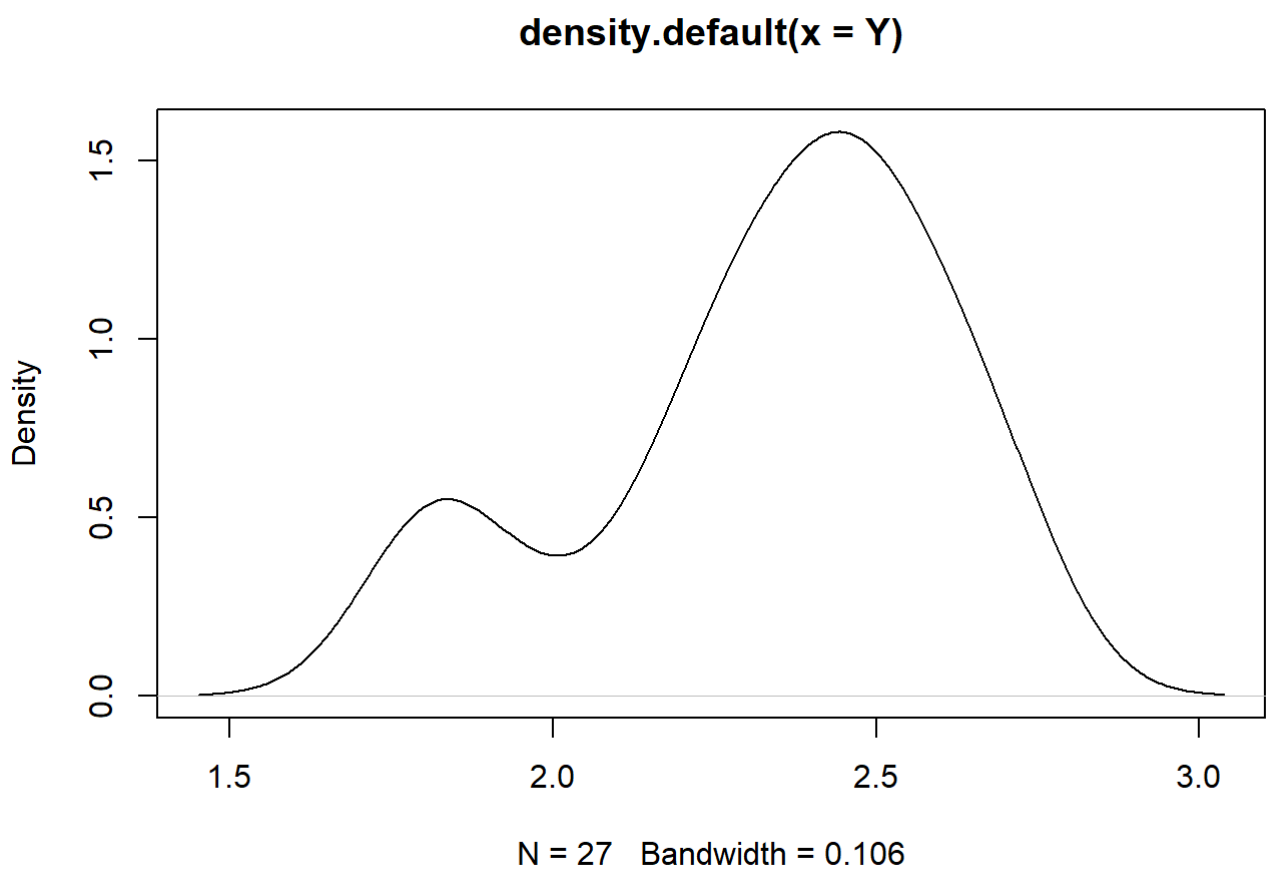
```
## [1] 0.8714516
```

```
s
```

```
## [1] 0.008063963
```

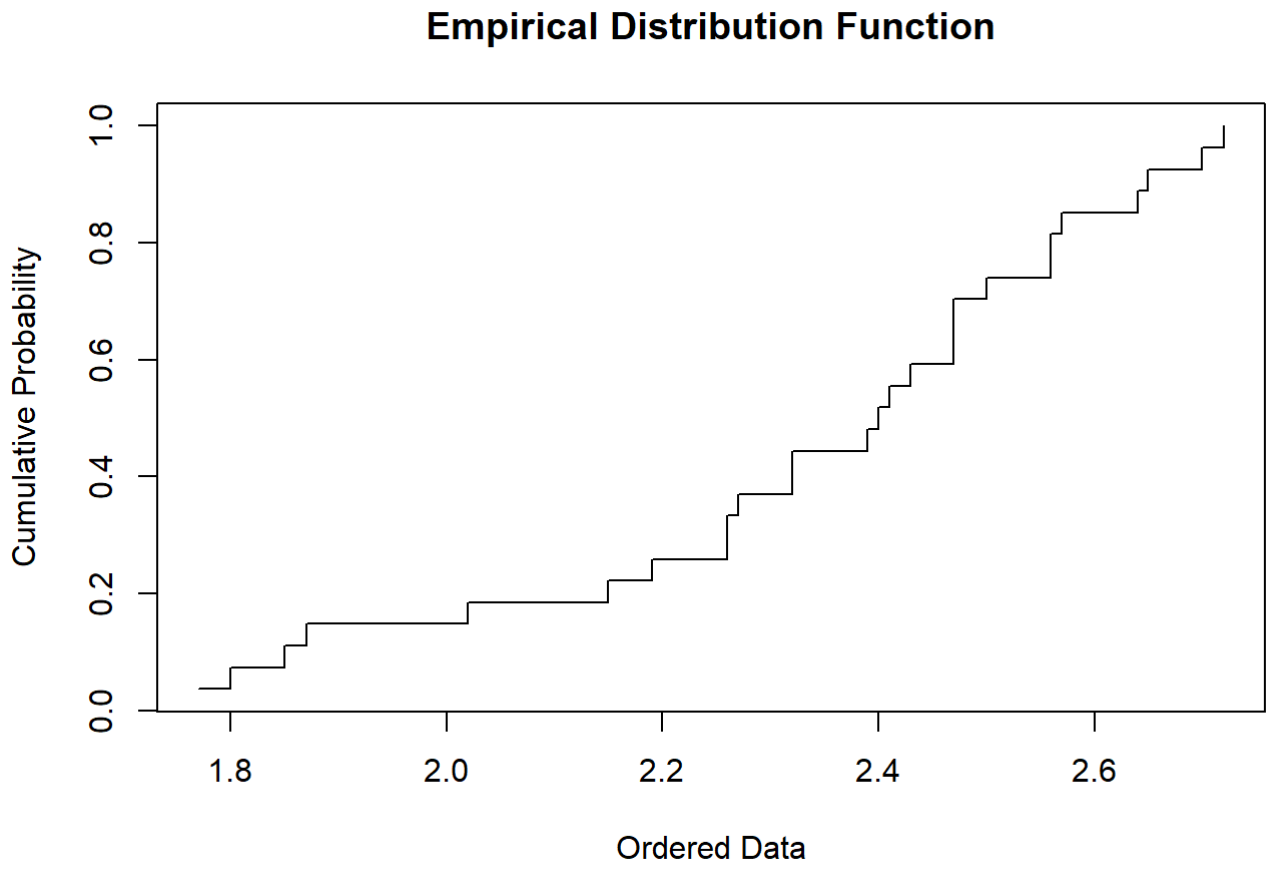
Vediamo com’è fatta la distribuzione di y.

```
plot(density(Y))
```



Posso trovare una stima di F di y. Il codice crea un plot della funzione di distribuzione empirica dei dati ordinati.

```
# Ordina i dati Y in ordine crescente
yord = Y[order(Y)]

# Calcola la lunghezza dei dati
n = length(Y)

# Calcola la funzione di distribuzione empirica, che rappresenta la probabilità cumulativa di
ciascun dato ordinato. La variabile d contiene queste probabilità cumulative
d = (1:(n)) / n

# Crea il plot della funzione di distribuzione empirica
plot(yord, d, type = "s", main = "Empirical Distribution Function", xlab = "Ordered Data", yl
ab = "Cumulative Probability")
```



In sintesi, il grafico mostra la funzione di distribuzione empirica dei dati ordinati, dove l'asse x rappresenta i dati ordinati e l'asse y rappresenta la probabilità cumulativa associata a ciascun dato. La linea a gradini collega i punti della funzione di distribuzione empirica.

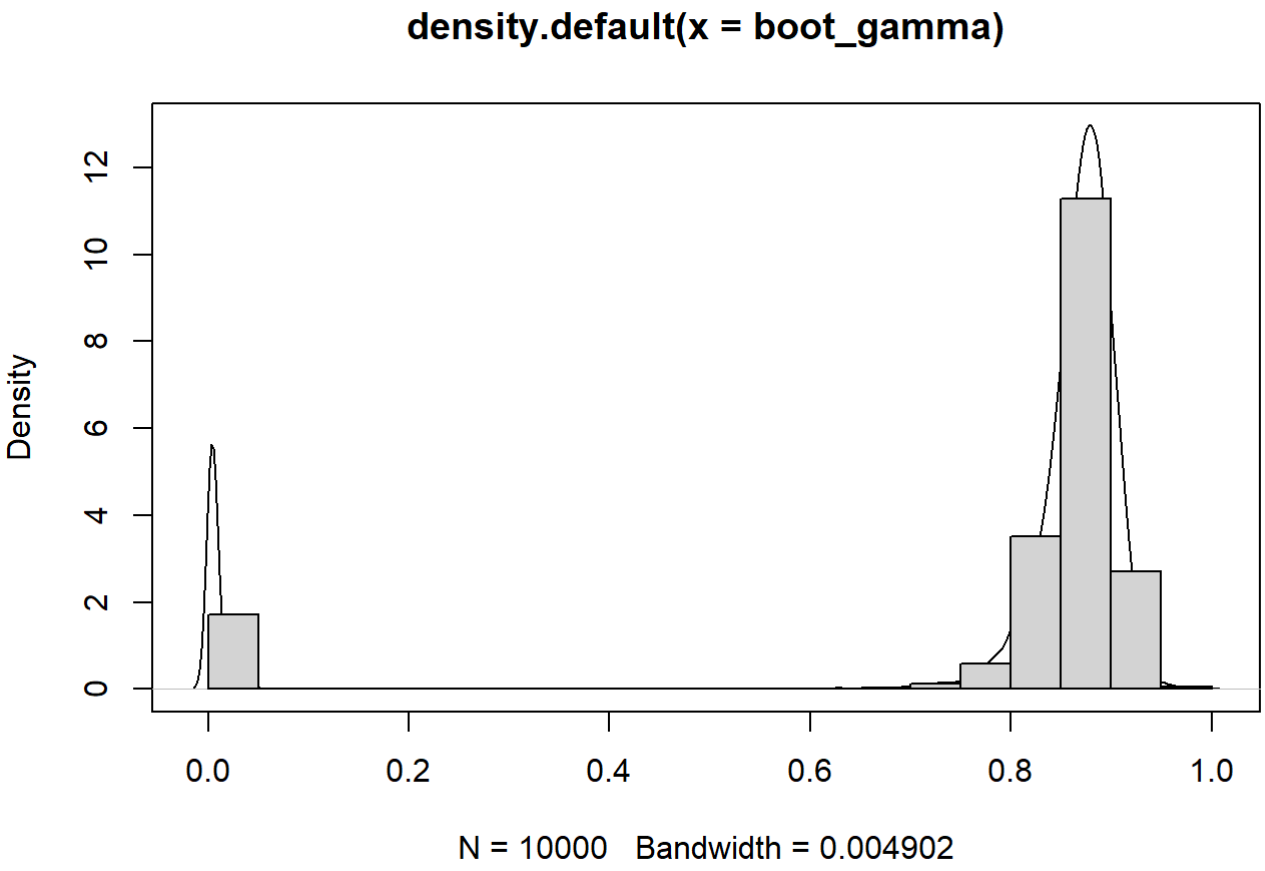
Simulo e ri-stimo gamma.

Il codice esegue un bootstrap per stimare la distribuzione dei parametri del modello, in particolare il parametro g, attraverso 10000 campionamenti bootstrap.

```
# Inizializza un vettore vuoto per memorizzare le stime di g ottenute con il bootstrap
boot_gamma = c()

# Esegui il campionamento bootstrap
for (ib in 1:10000) {
  # Campionamento bootstrap selezionando casualmente con sostituzione indici da 1 a n
  samp = sample(1:n, n, replace = TRUE)
  # Crea i vettori di dati di bootstrap utilizzando gli indici campionati.
  Yb = Y[samp]
  xb = x[samp]

  # Ottimizza la funzione di verosimiglianza negativa con i dati di bootstrap
  param = optim(c(0, 0, 0, 0), fff, method = "BFGS")$par

  # Estrai il parametro g stimato
  g = exp(param[3]) / (1 + exp(param[3]))

  # Memorizza la stima di g nel vettore boot_gamma
  boot_gamma[ib] = g
}
# Alla fine del ciclo, il vettore boot_gamma conterrà le stime di g ottenute tramite il boots
trap
```

Il codice plotta una stima della densità della distribuzione di g ottenuta attraverso il bootstrap utilizzando la funzione density, e l'altro è un istogramma della distribuzione di g con la sovrapposizione della stima della densità.

```
# Stima della densità di boot_gamma
plot(density(boot_gamma))
# Istogramma di boot_gamma con sovrapposizione della stima della densità
hist(boot_gamma, add=T, freq=F)
```

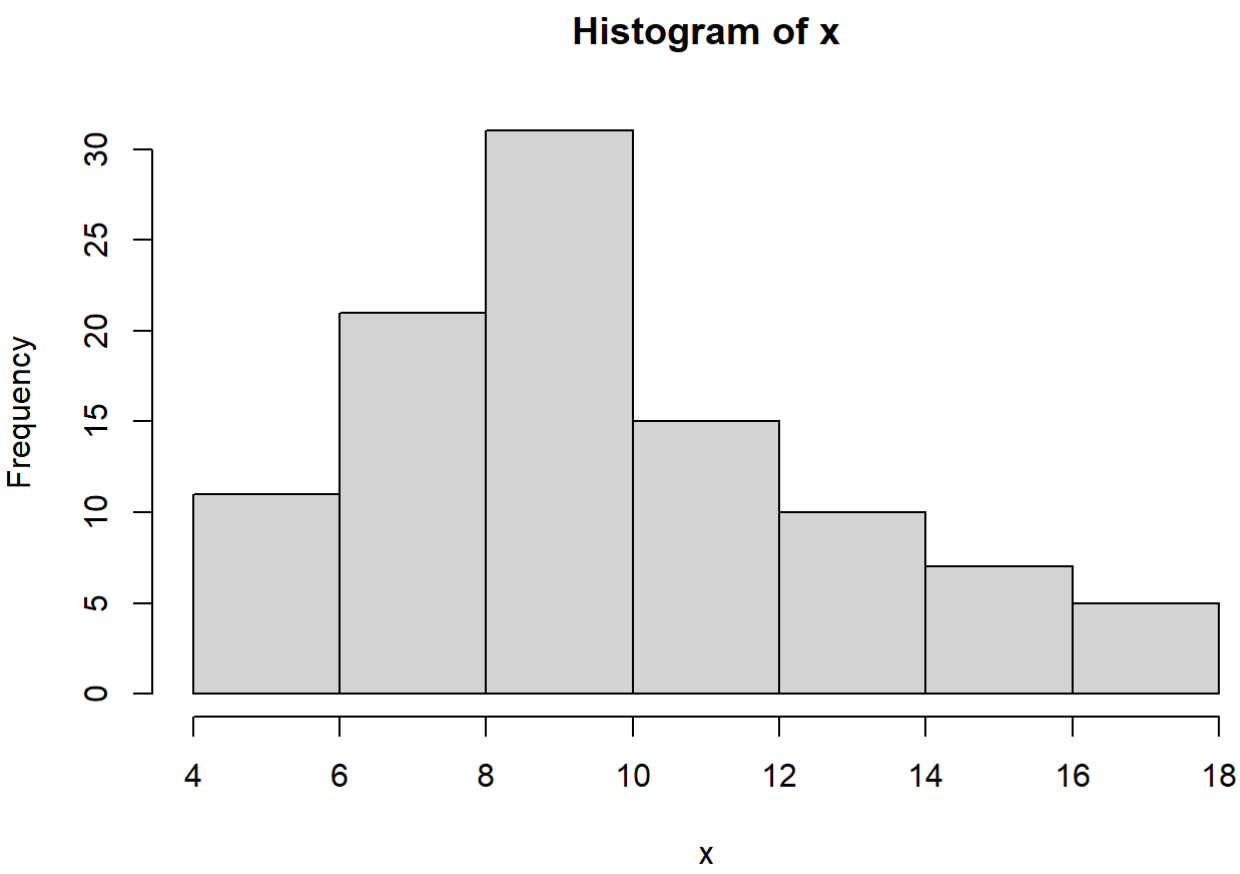


ESEMPIO SIMULATO BOOTSTRAP

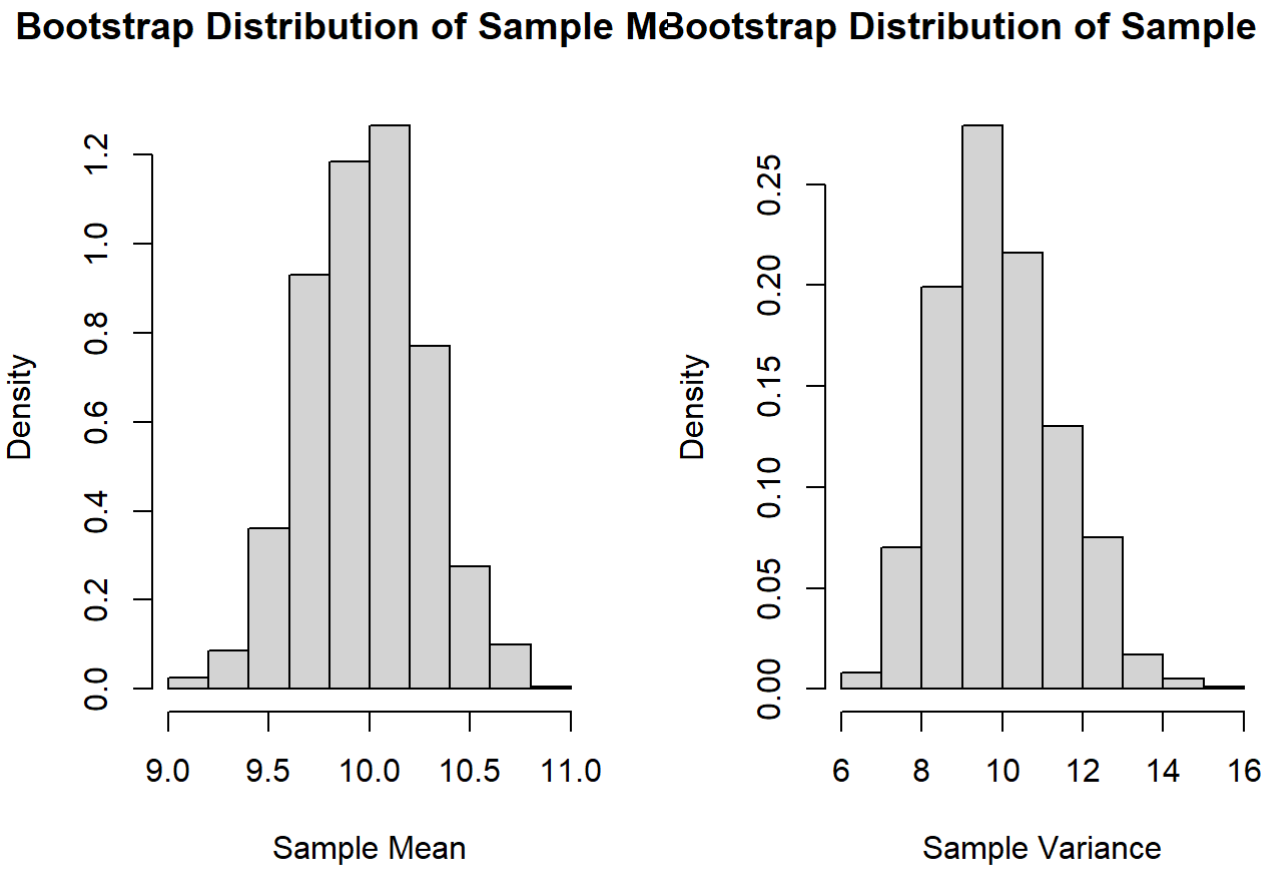
Simuliamo da una Poisson.

Il codice R simula un campione da una distribuzione di Poisson e quindi calcola la media campionaria (xbar) e la varianza campionaria (varhat) del campione originale. Successivamente, esegue un processo di bootstrap 1000 volte, calcolando la media campionaria e la varianza campionaria per ciascun campionamento, e infine crea un layout a due colonne con istogrammi delle distribuzioni delle medie campionarie e delle varianze campionarie ottenute tramite bootstrap.

```
# Simula un campione da una distribuzione di Poisson
n = 100
lambda = 10
x = rpois(n, lambda)
xreal = x

# Istogramma del campione originale
hist(x)
```




```r
# Calcola la media campionaria e la varianza campionaria del campione originale
xbar = mean(x)
varhat = var(x)

# Inizializza vettori per memorizzare medie campionarie e varianze campionarie del bootstrap
xbar_vec = c()
varhat_vec = c()

# Esegui il processo di bootstrap
for (i in 1:1000) {
  x = rpois(n, lambda)  # Campionamento bootstrap
  xbar = mean(x)  # Calcola la media campionaria del campione di bootstrap
  varhat = var(x)  # Calcola la varianza campionaria del campione di bootstrap
  xbar_vec[i] = xbar  # Memorizza la media campionaria nel vettore
  varhat_vec[i] = varhat  # Memorizza la varianza campionaria nel vettore
}

# Crea un layout a due colonne per i grafici
par(mfrow=c(1,2))

# Istogramma delle medie campionarie ottenute tramite bootstrap
hist(xbar_vec, freq = FALSE, main = "Bootstrap Distribution of Sample Mean", xlab = "Sample M
ean")

# Istogramma delle varianze campionarie ottenute tramite bootstrap
hist(varhat_vec, freq = FALSE, main = "Bootstrap Distribution of Sample Variance", xlab = "Sa
mple Variance")
```

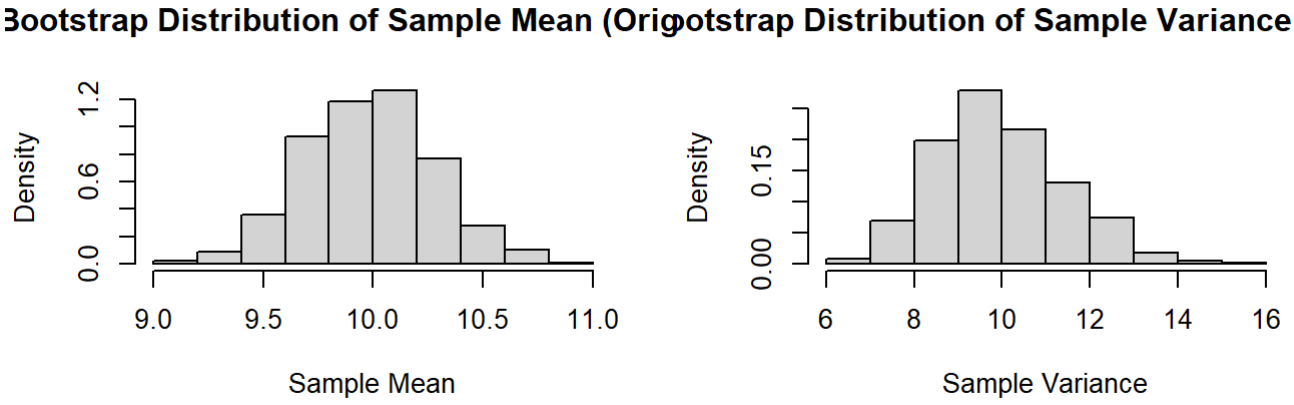


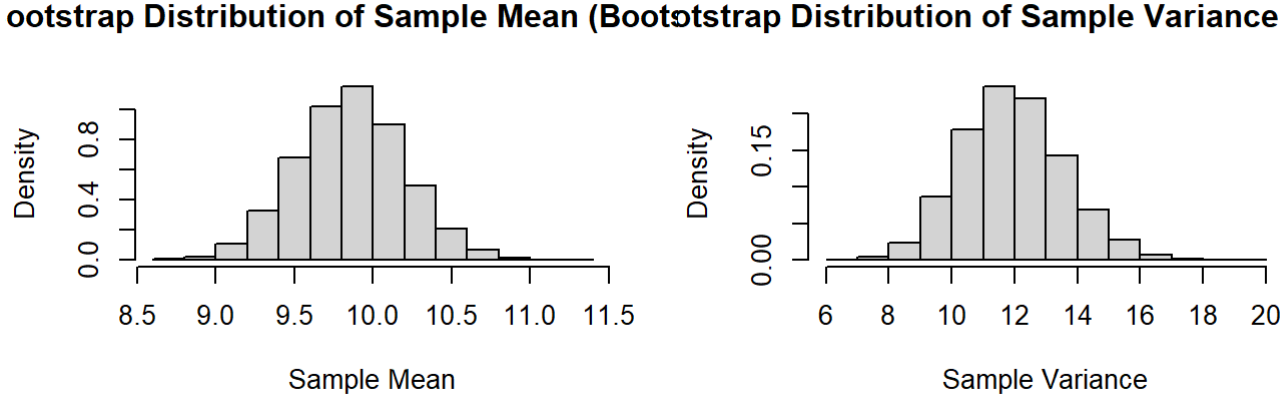
In sintesi, il codice simula un campione da una distribuzione di Poisson, calcola la media campionaria e la varianza campionaria del campione originale, quindi esegue un processo di bootstrap per ottenere le distribuzioni campionarie della media e della varianza. I risultati vengono visualizzati attraverso istogrammi in un layout a due colonne.

Io vorrei che gli stimatori bootstrap avessero queste “forme”.

Il codice esegue un bootstrap, selezionando casualmente con sostituzione indici dall’insieme degli indici del campione originale per ottenere campioni di bootstrap. Vengono calcolate le medie campionarie (xbar_boot) e le varianze campionarie (varhat_boot) per ciascun campionamento bootstrap, e successivamente vengono mostrati in un layout a due colonne con istogrammi.

```r
nb = 10000  # Numero di campionamenti bootstrap

# Inizializza vettori per memorizzare medie campionarie e varianze campionarie del bootstrap
xbar_boot = c()
varhat_boot = c()

# Esegui il processo di bootstrap
for (i in 1:nb) {
  ind = sample(1:n, n, replace = TRUE)  # Campionamento bootstrap di indici
  xb = x[ind]  # Seleziona il campione di bootstrap

  xbar = mean(xb)  # Calcola la media campionaria del campione di bootstrap
  varhat = var(xb)  # Calcola la varianza campionaria del campione di bootstrap

  xbar_boot[i] = xbar  # Memorizza la media campionaria nel vettore
  varhat_boot[i] = varhat  # Memorizza la varianza campionaria nel vettore
}

# Crea un layout a due colonne per i grafici
par(mfrow = c(2, 2))

# Istogramma delle medie campionarie ottenute tramite bootstrap (originale)
hist(xbar_vec, freq = FALSE, main = "Bootstrap Distribution of Sample Mean (Original)", xlab
= "Sample Mean")

# Istogramma delle varianze campionarie ottenute tramite bootstrap (originale)
hist(varhat_vec, freq = FALSE, main = "Bootstrap Distribution of Sample Variance (Original)",
xlab = "Sample Variance")

# Istogramma delle medie campionarie ottenute tramite bootstrap (bootstrap)
hist(xbar_boot, freq = FALSE, main = "Bootstrap Distribution of Sample Mean (Bootstrap)", xla
b = "Sample Mean")

# Istogramma delle varianze campionarie ottenute tramite bootstrap (bootstrap)
hist(varhat_boot, freq = FALSE, main = "Bootstrap Distribution of Sample Variance (Bootstra
p)", xlab = "Sample Variance")
```





In sintesi, questo codice esegue il bootstrap selezionando casualmente con sostituzione indici dall'insieme degli indici del campione originale, calcola le medie campionarie e le varianze campionarie per ciascun campionamento bootstrap e visualizza i risultati attraverso istogrammi in un layout a due colonne.