Data Science per psicologi

Corrado Caudek

2021-09-13

# Indice

*Capitolo 1*

# Approssimazione della distribuzione a posteriori

In generale, in un problema bayesiano i dati $y$ provengono da una densità $p(y \mid \theta)$ e al parametro $\theta$ viene assegnata una densità a priori $p(\theta)$. Dopo avere osservato un campione $Y = y$, la funzione di verosimiglianza è uguale a $\mathcal{L}(\theta) = p(y \mid \theta)$ e la densità a posteriori diventa

$$p(\theta \mid y) = \frac{p(\theta)\mathcal{L}(\theta)}{\int p(\theta)\mathcal{L}(\theta)d\theta}. \tag{1.1}$$

Si noti che, quando usiamo il teorema di Bayes per calcolare la distribuzione a posteriori del parametro di un modello statistico, al denominatore troviamo un integrale. Se vogliamo trovare la distribuzione a posteriori con metodi analitici è necessario usare distribuzioni a priori coniugate per la verosimiglianza. Per quanto "semplice" in termini formali, questo approccio, però, limita di molto le possibili scelte del ricercatore. Nel senso che non è sempre sensato, dal punto di vista teorico, utilizzare distribuzioni a priori coniugate per la verosimiglianza per i parametri di interesse. Però, se usiamo delle distribuzioni a priori non coniutate per la verosimiglianza, ci troviamo in una condizione nella quale, per determinare la distribuzione a posteriori, è necessario calcolare un integrale che, nella maggior parte dei casi, non si può risolvere analiticamente. In altre parole: è possibile ottenere analiticamenre la distribuzione a posteriori solo per alcune specifiche combinazioni di distribuzioni a priori e verosimiglianza, il che limita considerevolmente la flessibilità della modellizzazione.

Inoltre, i sommari della distribuzione a posteriori sono espressi come rapporto di integrali. Ad esempio, la media a posteriori di $\theta$ è data da

$$\mathbb{E}(\theta \mid y) = \frac{\int \theta p(\theta)\mathcal{L}(\theta)d\theta}{\int p(\theta)\mathcal{L}(\theta)d\theta}. \tag{1.2}$$

Il calcolo del valore atteso a posteriori richiede dunque la valutazione di due integrali, ciascuno dei quali non esprimibile in forma chiusa. Per questa ragione,

la strada principale che viene seguita nella modellistica bayesiana è quella che porta a determinare la distribuzione a posteriori non per via analitica, ma bensì mediante metodi numerici. La simulazione fornisce dunque la strategia generale del calcolo bayesiano.

A questo fine vengono usati i metodi di campionamento detti Monte-Carlo Markov-Chain (MCMC). Tali metodi costituiscono una potente e praticabile alternativa per la costruzione della distribuzione a posteriori per modelli complessi e consentono di decidere quali distribuzioni a priori e quali distribuzioni di verosimiglianza usare sulla base di considerazioni teoriche soltanto, senza dovere preoccuparsi di altri vincoli.

Dato che è basata su metodi computazionalmente intensivi, la stima numerica della funzione a posteriori può essere svolta soltanto mediante software. In anni recenti i metodi Bayesiani di analisi dei dati sono diventati sempre più popolari proprio perché la potenza di calcolo necessaria per svolgere tali calcoli è ora alla portata di tutti. Questo non era vero solo pochi decenni fa.

In questo Capitolo vedremo come sia possibile calcolare in maniera approssimata la distribuzione a posteriori. Presenteremo tre diverse tecniche che possono essere utilizzate a questo scopo:

1. il metodo basato su griglia,
2. il metodo dell'approssimazione quadratica,
3. il metodo di Monte Carlo basato su Catena di Markov (MCMC).

## 1.1 Metodo basato su griglia

Il metodo basato su griglia (*grid-based*) è un metodo di approsimazione numerica basato su una griglia di punti uniformemente spaziati. Anche se la maggior parte dei parametri è continua (ovvero, in linea di principio ciascun parametro può assumere un numero infinito di valori), possiamo ottenere un'eccellente approssimazione della distribuzione a posteriori considerando solo una griglia finita di valori dei parametri. In un tale metodo, la densità di probabilità a posteriori può dunque essere approssimata tramite le densità di probabilità cacolate in ciascuna cella della griglia.

Il metodo basato su griglia si sviluppa in quattro fasi:

- Fissare una griglia discreta di possibili valori $\theta$.[1]
- Valutare la distribuzione a priori $p(\theta)$ e la funzione di verosimiglianza $\mathcal{L}(y \mid \theta)$ in corrispondenza di ciascun valore $\theta$ della griglia.
- Ottenere un'approssimazione discreta della densità a posteriori: (a) calcolare il prodotto $p(\theta)\mathcal{L}(y \mid \theta)$ per ciascun valore $\theta$ della griglia; e (b) normalizzare i prodotti così ottenuti in modo tale che la loro somma sia 1.
- Selezionare $N$ valori casuali della griglia in modo tale da ottenere un campione casuale delle densità a posteriori normalizzate.

[1] È chiaro che, per ottenere buone approssimazioni, è necessaria una griglia molto densa.

### 1.1.1 Modello Beta-Binomiale

Supponiamo di avere osservato 9 sucessi in 10 prove Bernoulliane indipendenti.[2] Imponiamo alla distribuzione a priori su $\theta$ (proabilità di successo in una singola prova) una Beta(2, 2) per descrivere la nostra incertezza sul parametro prima di avere osservato i dati. Dunque, il modello diventa:

$$\begin{aligned} Y \mid \theta &\sim \text{Bin}(10, \pi) \\ \theta &\sim \text{Beta}(2, 2). \end{aligned} \tag{1.3}$$

In queste circostanze, l'aggiornamento bayesiano produce una distribuzione a posteriori Beta di parametri 11 ($y + \alpha = 9 + 2$) e 3 ($n - y + \beta = 10$ - $9 + 2$):

$$\theta \mid (y = 9) \sim \text{Beta}(11, 3) \ . \tag{1.4}$$

Per approssimare la distribuzione a posteriori, fissiamo una griglia di $n = 6$ valori equispaziati: $\theta \in \{0, 0.2, 0.4, 0.6, 0.8, 1\}$ (in seguito aumenteremo $n$):

```
grid_data <- tibble(
  theta_grid = seq(from = 0, to = 1, length = 6)
)
```

In corrispondenza di ciascun valore della griglia, valutiamo la distribuzione a priori Beta$(2, 2)$ e la verosimiglianza Bin$(10, \theta)$ con $y = 9$.

```
grid_data <- grid_data %>%
  mutate(
    prior = dbeta(theta_grid, 2, 2),
    likelihood = dbinom(9, 10, theta_grid)
  )
```

In ciascuna cella della griglia, calcoliamo il prodotto della verosimiglianza e della distribuzione a priori. Troviamo così un'approssimazione discreta e non normalizzata della distribuzione a posteriori (`unnormalized`). Normalizziamo poi questa approssimazione dividendo ciascun valore del vettore `unnormalized` per la somma di tutti i valori del vettore:

```
grid_data <- grid_data %>%
  mutate(
    unnormalized = likelihood * prior,
    posterior = unnormalized / sum(unnormalized))
```

La somma dei valori così trovati è uguale a 1:

[2] La discussione del modello Beta-Binomiale segue molto da vicino la presentazione di Johnson et al. (2022) utilizzando anche lo stesso codice R.

```
grid_data %>%
  summarize(
    sum(unnormalized),
    sum(posterior)
  )
#> # A tibble: 1 x 2
#>   `sum(unnormalized)` `sum(posterior)`
#>                 <dbl>            <dbl>
#> 1               0.318                1
```

Abbiamo dunque ottenuto la seguente distribuzione a posteriori discretizzata $p(\theta \mid y)$:

```
round(grid_data, 2)
#> # A tibble: 6 x 5
#>   theta_grid prior likelihood unnormalized posterior
#>        <dbl> <dbl>      <dbl>        <dbl>     <dbl>
#> 1        0    0          0            0         0
#> 2        0.2  0.96       0            0         0
#> 3        0.4  1.44       0            0         0.01
#> 4        0.6  1.44       0.04         0.06      0.18
#> 5        0.8  0.96       0.27         0.26      0.81
#> 6        1    0          0            0         0
```

La figura 1.1 mostra un grafico della distribuzione a posteriori discretizzata che è stata ottenuta:

```
grid_data %>%
  ggplot(
    aes(x = theta_grid, y = posterior)
  ) +
  geom_point() +
  geom_segment(
    aes(
      x = theta_grid,
      xend = theta_grid,
      y = 0,
      yend = posterior)
  )
```

L'ultimo passo della simulazione è il campionamento dalla distribuzione a posteriori discretizzata:

```
set.seed(84735)
post_sample <- sample_n(
```

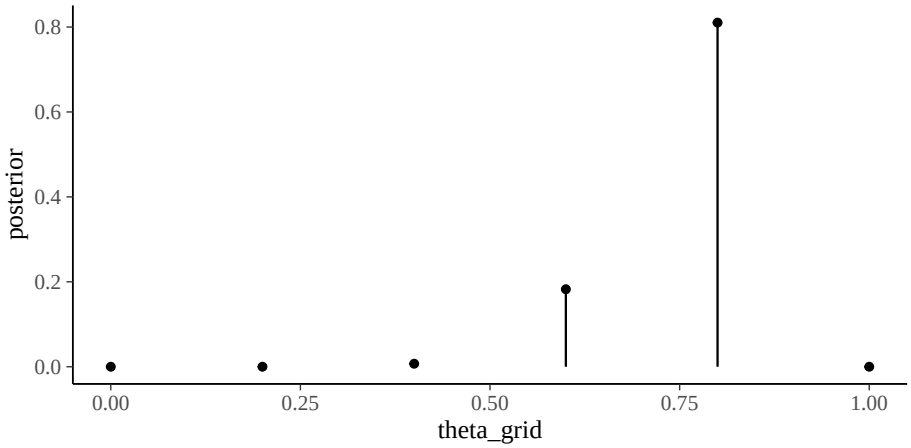



***Figura 1.1:*** *Distribuzione a posteriori discretizzata ottenuta con il metodo grid-based per $y = 9$ successi in 10 prove Bernoulliane, con distribuzione a priori Beta$(2, 2)$. È stata utilizzata una griglia di solo $n = 6$ punti.*

```
  grid_data,
  size = 1e5,
  weight = posterior,
  replace = TRUE
)
```

È facile intuire che i valori estratti con rimessa dalla distribuzione a posteriori discretizzata saranno quasi sempre uguali a 0.6 o 0.8. Questa intuizione è confermata dal grafico 1.2 a cui è stata sovrapposta la vera distribuzione a posteriori Beta$(11, 3)$:

```
ggplot(post_sample, aes(x = theta_grid)) +
  geom_histogram(aes(y = ..density..), color = "white") +
  stat_function(fun = dbeta, args = list(11, 3)) +
  lims(x = c(0, 1))
```

La figura 1.2 mostra che, con una griglia così sparsa abbiamo ottenuto una versione estremamente approssimata della vera distribuzione a posteriori. Possiamo ottenere un risultato migliore con una griglia più densa, come indicato nella figura 1.3:

```
grid_data  <- tibble(
  theta_grid = seq(from = 0, to = 1, length.out = 100)
)

grid_data <- grid_data %>%
```

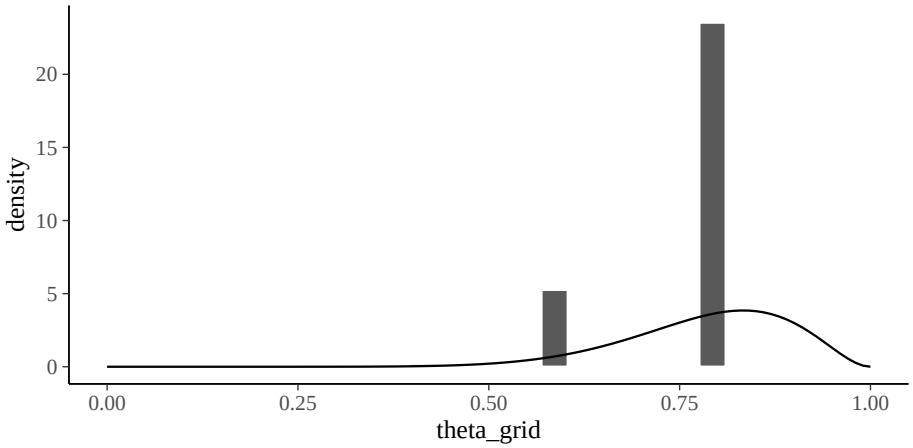



***Figura 1.2:*** *Campionamento dalla distribuzione a posteriori discretizzata ottenuta con il metodo grid-based per $y = 9$ successi in 10 prove Bernoulliane, con distribuzione a priori $Beta(2, 2)$. È stata utilizzata una griglia di solo $n = 6$ punti.*

```
  mutate(
    prior = dbeta(theta_grid, 2, 2),
    likelihood = dbinom(9, 10, theta_grid)
  )

grid_data <- grid_data %>%
  mutate(
    unnormalized = likelihood * prior,
    posterior = unnormalized / sum(unnormalized)
  )

grid_data %>%
ggplot(
  aes(x = theta_grid, y = posterior)
) +
  geom_point() +
  geom_segment(
    aes(
      x = theta_grid,
      xend = theta_grid,
      y = 0,
      yend = posterior)
  )
```

Campioniamo ora 10000 punti:

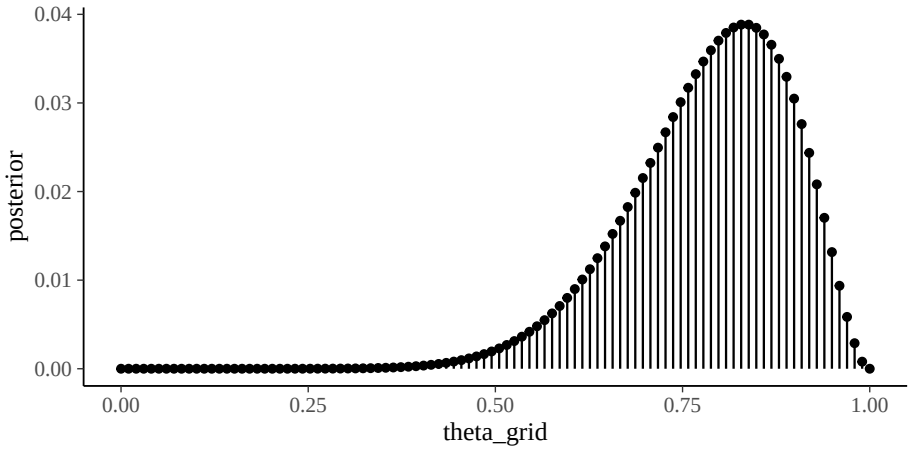



***Figura 1.3:*** *Distribuzione a posteriori discretizzata ottenuta con il metodo grid-based per $y = 9$ successi in 10 prove Bernoulliane, con distribuzione a priori Beta$(2, 2)$. È stata utilizzata una griglia di $n = 100$ punti.*

```
# Set the seed
set.seed(84735)
post_sample <- sample_n(
  grid_data,
  size = 1e4,
  weight = posterior,
  replace = TRUE
)
```

Con il campionamento dalla distribuzione a posteriori discretizzata costruita mediante una griglia più densa ($n = 100$) otteniamo un risultato soddisfacente (figura 1.4): la distribuzione dei valori prodotti dalla simulazione ora approssima molto bene la corretta distribuzione a posteriori $p(\theta \mid y) = \text{Beta}(11, 3)$.

```
post_sample %>%
  ggplot(aes(x = theta_grid)) +
  geom_histogram(
    aes(y = ..density..),
    color = "white",
    binwidth = 0.05
  ) +
  stat_function(fun = dbeta, args = list(11, 3)) +
  lims(x = c(0, 1))
```

Possiamo concludere dicendo che il metodo basato su griglia è molto intuitivo e non richiede particolari competenze di programmazione per essere

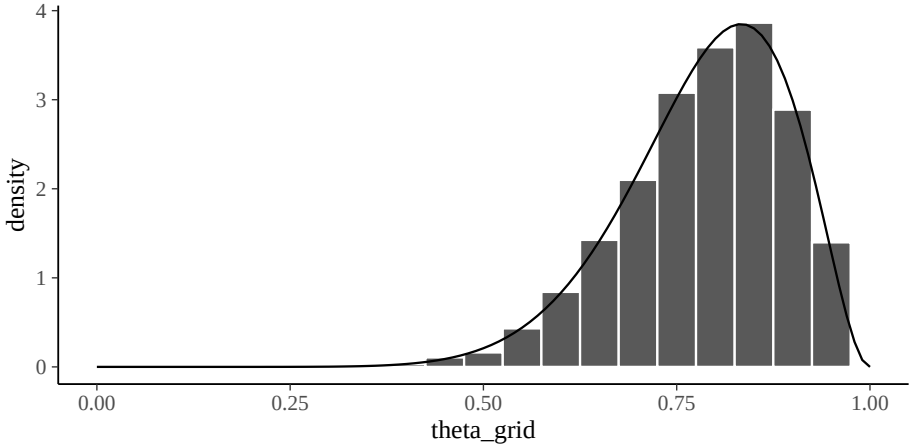



***Figura 1.4:*** *Campionamento dalla distribuzione a posteriori discretizzata ottenuta con il metodo grid-based per $y = 9$ successi in 10 prove Bernoulliane, con distribuzione a priori Beta$(2, 2)$. È stata utilizzata una griglia di $n = 100$ punti. All'istogramma è stata sovrapposta lacorretta distribuzione a posteriori, ovvero la densità Beta$(11, 3)$.*

implementato. Inoltre, fornisce un risultato che, per tutti gli scopi pratici, può essere considerato come un campione casuale estratto da $p(\theta \mid y)$. Tuttavia, anche se tale metodo fornisce risultati accuratissimi, esso ha un uso limitato. A causa della *maledizione della dimensionalità*[3], infatti, il metodo basato su griglia può essere solo usato nel caso di semplici modelli statistici, con non più di due parametri. Nella pratica concreta tale metodo viene dunque sostituito da altre tecniche più efficienti in quanto, anche nei più comuni modelli utilizzati in psicologia, vengono solitamente stimati centinaia se non migliaia di parametri.

## 1.2 Approssimazione quadratica

L'approssimazione quadratica è uno dei metodi che possono essere usati per superare il problema della "maledizione della dimensionalità". La motivazione di tale metodo è la seguente. Sappiamo che, in generale, la regione della distribuzione a posteriori che si trova in prossimità del suo massimo può essere ben approssimata dalla forma di una distribuzione Normale.[4]

L'approssimazione quadratica si pone due obiettivi.

[3] Che cos'è la *maledizione della dimensionalità*? È molto facile da capire. Supponiamo di utilizzare una griglia di 100 punti equispaziati. Nel caso di un solo parametro, sarà necessario calcolare 100 valori. Per due parametri devono essere calcolari $100^2$ valori. Ma già per 10 parametri avremo bisogno di calcolare $10^{10}$ valori – è facile capire che una tale quantità di calcoli è troppo grande anche per un computer molto potente. Per modelli che richiedono la stima di un numero non piccolo di parametri è dunque necessario procedere in un altro modo.

[4] Descrivere la distribuzione a posteriori mediante la distribuzione Normale significa utilizzare un'approssimazione che viene, appunto, chiamata "quadratica" (tale approssimazione si dice quadratica perché il logaritmo di una distribuzione gaussiana forma una parabola e la

1. Trovare la moda della distribuzione a posteriori. Ci sono varie procedure di ottimizzazione, implementate in R, in grado di trovare il massimo di una distribuzione.

2. Stimare la curvatura della distribuzione in prossimità della moda. Una stima della curvatura è sufficiente per trovare un'approssimazione quadratica dell'intera distribuzione. In alcuni casi, questi calcoli possono essere fatti seguendo una procedura analitica, ma solitamente vengono usate delle tecniche numeriche.

Una descrizione della distribuzione a posteriori ottenuta mediante l'approssimazione quadratica si ottiene mediante la funzione `quap()` contenuta nel pacchetto `rethinking`:[5]

```
suppressPackageStartupMessages(library("rethinking"))

mod <- quap(
  alist(
  N ~ dbinom(N + P, p), # verosimiglianza binomiale
  p ~ dbeta(2, 10) # distribuzione a priori Beta(2, 10)
  ),
  data = list(N = 23, P = 7)
)
```

Un sommario dell'approssimazione quadratica è fornito da

```
precis(mod, prob = 0.95)
#>        mean        sd      2.5%     97.5%
#> p 0.5999999 0.0774593 0.4481824 0.7518173
```

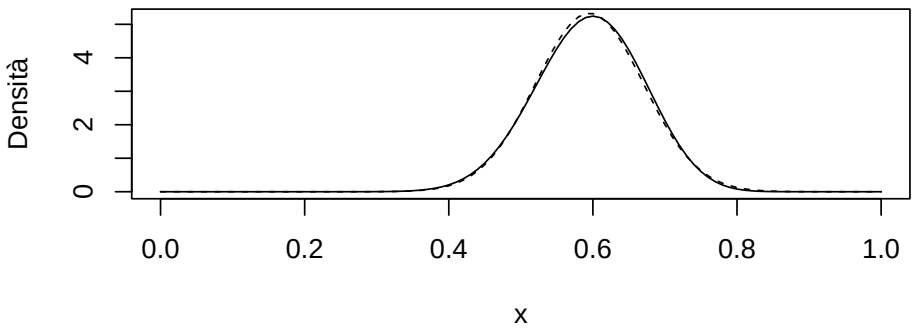
Qui sotto è fornito un confronto tra la corretta distribuzione a posteriori (linea continua) e l'approssimazione quadratica (linea trateggiata).

```
N <- 23
P <- 7
a <- N + 2
b <- P + 10
curve(dbeta(x, a, b), from=0, to=1, ylab="Densità")
# approssimazione quadratica
curve(
  dnorm(x, a/(a+b), sqrt((a*b)/((a+b)^2*(a+b+1)))),
  lty = 2,
```

parabola è una funzione quadratica – dunque, mediante questa approssimazione descriviamo il logaritmo della distribuzione a posteriori mediante una parabola).

[5] Il pacchetto `rethinking` è stato creato da McElreath (2020) per accompagnare il suo testo *Statistical Rethinking*[2]. Per l'installazione si veda https://github.com/rmcelreath/rethinking.

```
  add = TRUE
)
```




Il grafico precedente mostra che l'approssimazione quadratica fornisce risultati soddisfacenti. Tali risultati sono simili (o identici) a quelli ottenuti con il metodo *grid-based*, con il vantaggio aggiuntivo di disporre di una serie di funzioni R in grado di svolgere i calcoli per noi. In realtà, però, l'approssimazione quadratica è poco usata perché, per problemi complessi, è più conveniente fare ricorso ai metodi Monte Carlo basati su Catena di Markov (MCMC) che verranno descritti nel Paragrafo successivo.

## 1.3 Metodo Monte Carlo

Una verosimiglianza Binomiale e una distribuzione a priori Beta producono una distribuzione a posteriori Beta (si veda il capitolo **??**). Con una simulazione R è dunque facile ricavare dei campioni causali dalla distribuzione a posteriori. Maggiore è il numero di campioni, migliore sarà l'approssimazione della distribuzione a posteriori.

Continuando con l'esempio precedente,

$$\begin{aligned} y \mid \theta, n &\sim \text{Bin}(y = 23, n = 30 \mid \theta) \\ \theta_{prior} &\sim \text{Beta}(2, 10) \\ \theta_{post} &\sim \text{Beta}(y + a = 23 + 2 = 25, n - y + b = 30 - 23 + 10 = 17), \end{aligned}$$

stimiamo, ad esempio, il valore della media a posteriori di $\theta$:

```
set.seed(7543897)
print(mean(rbeta(1e2, shape1 = 25, shape2 = 17)), 6)
#> [1] 0.587548
```

L'approssimazione migliora all'aumentare del numero di campioni estratto dalla distribuzione a posteriori:

```
print(mean(rbeta(1e3, shape1 = 25, shape2 = 17)), 6)
#> [1] 0.597659
```

```
print(mean(rbeta(1e4, shape1 = 25, shape2 = 17)), 6)
#> [1] 0.595723
```

```
print(mean(rbeta(1e5, shape1 = 25, shape2 = 17)), 6)
#> [1] 0.595271
```

È lo stesso ragionamento che abbiamo fatto in riferiento alla relazione tra campione e popolazione: al crescere della numerosità campionaria il campione approssima sempre meglio le proprietà della popolazione da cui è stato estratto (legge dei grandi numeri). Nel caso presente, il risultato esatto è

$$\bar{\theta}_{post} = \frac{\alpha}{\alpha + \beta} = \frac{25}{25 + 17} \approx 0.5952.$$

Quando il numero di campioni tratti dalla distribuzione a posteriori è molto grande, quindi, la distribuzione dei campioni converge alla densità della popolazione (si veda l'Appendice **??**).[6]

Inoltre, le statistiche descrittive (es. media, moda, varianza, eccetera) dei campioni estratti dalla distribuzione a posteriori convergeranno ai corrispondenti valori della distribuzione a posteriori. La figura @ref{fig:mcmc-chains-1} mostra come, all'aumentare del numero di repliche, la media, la mediana, la deviazione standard e l'asimmetria convergono ai veri valori della distribuzione a posteriori (linee rosse tratteggiate).

## 1.4 Metodi MC basati su Catena di Markov

Nel Paragrafo 1.3 la simulazione Monte Carlo funzionava perché

(a) sapevamo che la distribuzione a posteriori era una Beta$(25, 17)$,
(b) era possibile usare le funzioni R per estrarre campioni casuali da tale distribuzione.

Tuttavia, capita raramente di usare una distribuzione a priori coniugata alla verosimiglianza, quindi in generale non valgono né la condizione (a) né la condizione (b) descritte sopra. Ad esempio, nel caso di una verosimiglianza binomiale e una distribuzione a priori Normale, la distribuzione a posteriori di $\theta$ è

$$p(\theta \mid y) = \frac{\mathrm{e}^{-(\theta-1/2)^2}\theta^y(1-\theta)^{n-y}}{\int_0^1 \mathrm{e}^{-(t-1/2)^2}t^y(1-t)^{n-y}dt}.$$

[6] Si noti, naturalmente, che il numero dei campioni di simulazione è controllato dal ricercatore; è totalmente diverso dalla dimensione del campione che è fissa ed è una proprietà dei dati.

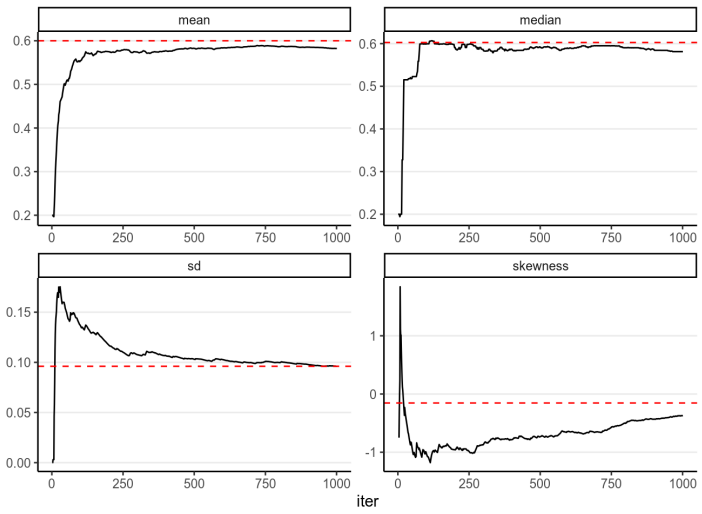



***Figura 1.5:*** *Convergenza delle simulazioni Monte Carlo.*

Una tale distribuzione non è implementata in R e dunque non possiamo campionare da $p(\theta \mid y)$.

Per fortuna, esiste un algoritmo chiamato Monte Carlo basato su catena di Markov (*Markov Chain Monte Carlo*, MCMC) che consente il campionamento da una distribuzione a posteriori senza che sia necessario conoscere la rappresentazione analitica di una tale distribuzione. I metodi Monte Carlo basati su catena di Markov consentono di costruire sequenze di punti (le "catene") nello spazio dei parametri le cui densità sono proporzionali alla distribuzione a posteriori — in altre parole, dopo aver simulato un grande numero di passi della catena si possono usare i valori così generati come se fossero un campione casuale della distribuzione a posteriori. Le tecniche MCMC sono attualmente il metodo computazionale maggiormente utilizzato per risolvere i problemi di inferenza bayesiana.

### 1.4.1 Catene di Markov

Per introdurre il concetto di catena di Markov, supponiamo che una persona esegua una passeggiata casuale sulla retta dei numeri naturali considerando solo i valori 1, 2, 3, 4, 5, 6.[7] Se la persona è collocata su un valore interno dei valori possibili (ovvero, 2, 3, 4 o 5), nel passo successivo è altrettanto probabile che rimanga su quel numero o si sposti su un numero adiacente. Se si muove, è ugualmente probabile che si muova a sinistra o a destra. Se la persona si trova su uno dei valori estremi (ovvero, 1 o 6), nel passo successivo è altrettanto probabile che rimanga rimanga su quel numero o si sposti nella posizione adiacente.

[7] Seguiamo qui la presentazione fornita da Bob Carpenter.

Questo è un esempio di una catena di Markov discreta. Una catena di Markov descrive il movimento probabilistico tra un numero di stati. Nell'esempio ci sono sei possibili stati, da 1 a 6, i quali corrispondono alle possibili posizioni della passeggiata casuale. Data la sua posizione corrente, la persona si sposterà nelle altre posizioni possibili con delle specifiche probabilità. La probabilità che si sposti in un'altra posizione dipende solo dalla sua posizione attuale e non dalle posizioni visitate in precedenza.

È possibile descrivere il movimento tra gli stati nei termini delle cosiddette *probabilità di transizione*, ovvero le probabilità di movimento tra tutti i possibili stati in un unico passaggio di una catena di Markov. Le probabilità di transizione sono riassunte in una *matrice di transizione* $P$:

```
p <- c(0, 0, 1, 0, 0, 0)

P <- matrix(
  c(.5, .5, 0, 0, 0, 0,
    .25, .5, .25, 0, 0, 0,
    0, .25, .5, .25, 0, 0,
    0, 0, .25, .5, .25, 0,
    0, 0, 0, .25, .5, .25,
    0, 0, 0, 0, .5, .5
    ),
  nrow = 6, ncol = 6, byrow = TRUE)

kableExtra::kable(P)
```

| | | | | | |
|---|---|---|---|---|---|
| 0.50 | 0.50 | 0.00 | 0.00 | 0.00 | 0.00 |
| 0.25 | 0.50 | 0.25 | 0.00 | 0.00 | 0.00 |
| 0.00 | 0.25 | 0.50 | 0.25 | 0.00 | 0.00 |
| 0.00 | 0.00 | 0.25 | 0.50 | 0.25 | 0.00 |
| 0.00 | 0.00 | 0.00 | 0.25 | 0.50 | 0.25 |
| 0.00 | 0.00 | 0.00 | 0.00 | 0.50 | 0.50 |

La prima riga della matrice di transizione $P$ fornisce le probabilità di passare a ciascuno degli stati da 1 a 6 in un unico passaggio a partire dalla posizione 1; la seconda riga fornisce le probabilità di transizione in un unico passaggio dalla posizione 2 e così via. Per esempio, il valore $P[1, 1]$ ci dice che, se la persona è nello stato 1, avrà una probabilità di 0.5 di rimanere in quello stato; $P[1, 2]$ ci dice che c'è una probabilità di 0.5 di passare dallo stato 1 allo stato 2. Gli altri elementi della prima riga sono 0 perché, in un unico passaggio, non è possibile passare dallo stato 1 agli stati 3, 4, 5 e 6. Il valore $P[2, 1]$ ci dice che, se la persona è nello stato 1 (seconda riga), avrà una probabilità di 0.25 di passare allo stato 1; avra una probabilità di 0.5 di rimanere in quello stato, $P[2, 2]$; e avrà una probabilità di 0.25 di passare allo stato 3, $P[2, 3]$; eccetera.

Si notino alcune importanti proprietà di questa particolare catena di Markov.

- È possibile passare da ogni stato a qualunque altro stato in uno o più passaggi: una catena di Markov con questa proprietà si dice *irriducibile*.
- Dato che la persona si trova in un particolare stato, se può tornare a questo stato solo a intervalli regolari, si dice che la catena di Markov è *periodica*. In questo esempio la catena è *aperiodica* poiché la passeggiata casuale non può eitornare allo stato attuale a intervalli regolari.

Un'importante proprietà di una catena di Markov irriducibile e aperiodica è che il passaggio ad uno stato del sistema dipende unicamente dallo stato immediatamente precedente e non dal come si è giunti a tale stato (dalla storia). Per questo motivo si dice che un processo markoviano è senza memoria. Tale "assenza di memoria" può essere interpretata come la proprietà mediante cui è possibile ottenere un insieme di campioni casuali da una distribuzione di interesse. Nel caso dell'inferenza bayesiana, la distribuzione di interesse è la distribuzione a posteriori, $p(\theta \mid y)$. Le catene di Markov consentono di stimare i valori di aspettazione di variabili rispetto alla distribuzione a posteriori.

La matrice di transizione che si ottiene dopo un enorme numero di passi di una passeggiata casuale markoviana si chiama *distribuzione stazionaria*. Se una catena di Markov è irriducibile e aperiodica, allora ha un'unica distribuzione stazionaria $w$. La distribuzione limite di una tale catena di Markov, quando il numero di passi tende all'infinito, è uguale alla distribuzione stazionaria $w$.

### 1.4.2 Simulare una catena di Markov

Un metodo per dimostrare l'esistenza della distribuzione stazionaria di una catena di Markov è quello di eseguire un esperimento di simulazione. Iniziamo una passeggiata casuale partendo da un particolare stato, diciamo la posizione 3, e quindi simuliamo molti passaggi della catena di Markov usando la matrice di transizione $P$. Al crescere del numero di passi della catena, le frequenze relative che descrivono il passaggio a ciascuno dei sei possibili nodi della catena approssimano sempre meglio la distribuzione stazionaria $w$.

Senza entrare nei dettagli della simulazione, la figura 1.6 mostra i risultati ottenuti in 10,000 passi di una passeggiata casuale markoviana. Si noti che, all'aumentare del numero di iterazioni, le frequenze relative approssimano sempre meglio le probabilità nella distribuzione stazionaria $w = (0.1, 0.2, 0.2, 0.2, 0.2, 0.1)$.

```
set.seed(123)
s <- vector("numeric", 10000)
s[1] <- 3
for (j in 2:10000){
  s[j] <- sample(1:6, size=1, prob=P[s[j - 1], ])
```

```
}
S <- data.frame(Iterazione = 1:10000,
                Location = s)

S %>% mutate(L1 = (Location == 1),
             L2 = (Location == 2),
             L3 = (Location == 3),
             L4 = (Location == 4),
             L5 = (Location == 5),
             L6 = (Location == 6))  %>%
  mutate(Proporzione_1 = cumsum(L1) / Iterazione,
         Proporzione_2 = cumsum(L2) / Iterazione,
         Proporzione_3 = cumsum(L3) / Iterazione,
         Proporzione_4 = cumsum(L4) / Iterazione,
         Proporzione_5 = cumsum(L5) / Iterazione,
         Proporzione_6 = cumsum(L6) / Iterazione) %>%
  select(Iterazione, Proporzione_1, Proporzione_2, Proporzione_3,
         Proporzione_4, Proporzione_5, Proporzione_6) -> S1

gather(S1, Outcome, Probability, -Iterazione) -> S2

ggplot(S2, aes(Iterazione, Probability)) +
  geom_line() +
  facet_wrap(~ Outcome, ncol = 3) +
  ylim(0, .4) +
  ylab("Frequenza relativa") +
  # theme(text=element_text(size=14))  +
  scale_x_continuous(breaks = c(0, 3000, 6000, 9000))
```

### 1.4.3 Campionamento mediante algoritmi MCMC

Il metodo di campionamento utilizzato dagli algoritmi Monte Carlo a catena di Markov (MCMC) crea una catena di Markov irriducibile e aperiodica, la cui distribuzione stazionaria equivale alla distribuzione a posteriori $p(\theta \mid y)$. Un modo generale per ottenere una tale catena di Markov è quello di usare l'algoritmo di Metropolis. L'algoritmo di Metropolis è il primo algoritmo MCMC che è stato proposto, ed è applicabile ad una grande varietà di problemi inferenziali di tipo bayesiano. Tale algoritmo è stato in seguito sviluppato allo scopo di renderlo via via più efficiente. Lo presentiamo qui in una forma intuitiva.

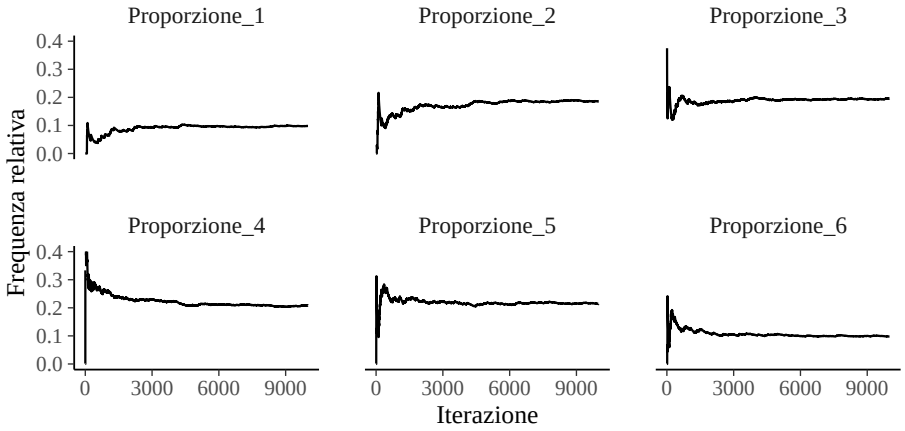



***Figura 1.6:*** *Frequenze relative degli stati da 1 a 6 in funzione del numero di iterazioni per la simulazione di una catena di Markov.*

### 1.4.4 Una passeggiata casuale sui numeri naturali

Per introdurre l'algoritmo di di Metropolis considereremo il campionamento da una distribuzione discreta.[8] Supponiamo di definire una distribuzione di probabilità discreta sugli interi $1, \dots, K$. Scriviamo in R la funzione `pd()` che assegna ai valori $1, \dots, 8$ delle probabilità proporzionali a 5, 10, 4, 4, 20, 20, 12 e 5.

```
pd <- function(x){
  values <- c(5, 10, 4, 4, 20, 20, 12, 5)
  ifelse(
    x %in% 1:length(values),
    values[x] / sum(values),
    0
  )
}

prob_dist <- tibble(
  x = 1:8,
  prob = pd(1:8)
)
```

La figura 1.7 illustra la distribuzione di probabilità che è stata generata.

```
x <- 1:8
prob_dist %>%
```

[8] Seguiamo qui la trattazione di Albert and Hu (2019). Per una presentazione intuitiva dell'algoritmo di Metropolis, si vedano anche Kruschke (2014); McElreath (2020).

```
ggplot(aes(x = x, y = prob)) +
geom_bar(stat = "identity", width = 0.06) +
scale_x_continuous("x", labels = as.character(x), breaks = x) +
labs(
  y = "Probabilità",
  x = "X"
)
```

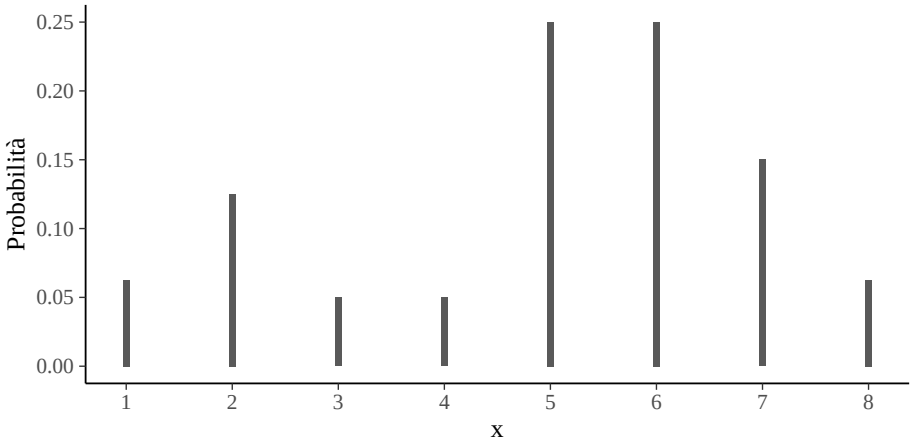



***Figura 1.7:*** *Distribuzione di massa di probabilità per una variabile casuale avente valori 1, 2, ..., 8.*

L'algoritmo di Metropolis corrisponde alla seguente passeggiata casuale.

1. L'algoritmo inizia con un valore iniziale qualsiasi da 1 a $K = 8$ della variabile casuale.

2. Per simulare il valore successivo della sequenza, lanciamo una moneta equilibrata. Se esce testa, consideriamo come valore candidato il valore immediatamente precedente al valore corrente nella sequenza $1, \ldots, 8$; se esce croce, il valore candidato sarà il valore immediatamente successivo al valore corrente nella sequenza.

3. Calcoliamo il rapporto tra la probabilità del valore candidato e la probabilità del valore corrente:

$$R = \frac{pd(\text{valore candidato})}{pd(\text{valore corrente})}.$$

4. Estraiamo un numero a caso $\in [0, 1]$. Se tale valore è minore di $R$ accettiamo il valore candidato come valore successivo della catena markoviana; altrimenti il valore successivo della catena rimane il valore corrente.

I passi da 1 a 4 definiscono una catena di Markov irriducibile e aperiodica sui valori di stato $\{1, 2, \dots, 8\}$, dove il passo 1 fornisce il valore iniziale della catena e i passi da 2 a 4 definiscono la matrice di transizione $P$. Un modo di campionare da una distribuzione di massa di probabilità `pd` consiste nell'iniziare da una posizione qualsiasi e eseguire una passeggiata casuale costituita da un grande numero di passi, ripetendo le fasi 2, 3 e 4 dell'algoritmo di Metropolis. Dopo un grande numero di passi, la distribuzione dei valori della catena markoviana approssimerà la distribuzione di probabilità `pd`.

La funzione `random_walk()` implementa l'algoritmo di Metropolis. Tale funzione richiede in input la distribuzione di probabilità `pd`, la posizione di partenza `start` e il numero di passi dell'algoritmo `num_steps`.

```
random_walk <- function(pd, start, num_steps){
  y <- rep(0, num_steps)
  current <- start
  for (j in 1:num_steps){
    candidate <- current + sample(c(-1, 1), 1)
    prob <- pd(candidate) / pd(current)
    if (runif(1) < prob)
      current <- candidate
    y[j] <- current
  }
  return(y)
}
```

Di seguito, implementiamo l'algoritmo di Metropolis utilizzando, quale valore iniziale, $X = 4$. Ripetiamo la simulazione 10,000 volte.

```
out <- random_walk(pd, 4, 1e4)

S <- tibble(out) %>%
  group_by(out) %>%
  summarize(
    N = n(),
    Prob = N / 10000
  )

prob_dist2 <- rbind(
  prob_dist,
  tibble(
    x = S$out,
    prob = S$Prob
    )
  )
prob_dist2$Type <- rep(
```

```
  c("Prob. corrette", "Prob. simulate"),
  each = 8
  )
```

```
x <- 1:8
prob_dist2 %>%
  ggplot(aes(x = x, y = prob, fill = Type)) +
  geom_bar(stat = "identity", width = 0.1, position = position_dodge(0.3)) +
  scale_x_continuous("x", labels = as.character(x), breaks = x) +
  scale_fill_manual(values = c("black", "gray80")) +
  theme(legend.title = element_blank()) +
  labs(
    y = "Probabilità",
    x = "X"
  )
```

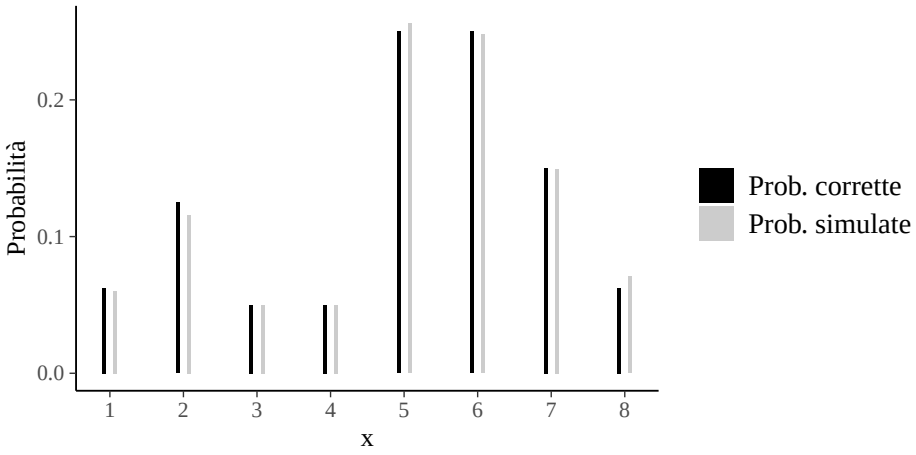



***Figura 1.8:*** *L'istogramma confronta i valori prodotti dall'algoritmo di Metropolis con i corretti valori della distribuzione di massa di probabilità.*

La figura 1.8 confronta l'istogramma dei valori simulati dalla passeggiata casuale con l'effettiva distribuzione di probabilità `pd`. Si noti la somiglianza tra le due distribuzioni.

### 1.4.5 L'algoritmo di Metropolis

Vediamo ora come l'algoritmo di Metropolis possa venire usato per generare una catena di Markov irriducibile e aperiodica per la quale la distribuzione stazionaria è uguale alla distribuzione a posteriori di interesse.[9]

[9] Una illustrazione visiva di come si svolge il processo di "esplorazione" dell'algoritmo di Metropolis è fornita in questo post.

In termini generali, l'algoritmo di Metropolis include due fasi.

- *Fase 1* La selezione di un valore candidato $\theta'$ del parametro mediante il campionamento da una distribuzione proposta.
- *Fase 2* La decisione tra la possibilità di accettare il valore candidato

$$\theta^{(m+1)} = \theta'$$

o di mantenere il valore corrente

$$\theta^{(m+1)} = \theta$$

sulla base del seguente criterio:

  - se $\mathcal{L}(\theta' \mid y)p(\theta') > \mathcal{L}(\theta \mid y)p(\theta)$ il valore candidato viene sempre accettato;
  - altrimenti il valore candidato viene accettato solo in una certa proporzione di casi.

Esaminiamo ora nei dettagli il funzionamento dell'algoritmo di Metropolis.

(a) Si inizia con un punto arbitrario $\theta^{(1)}$, quindi il primo valore della catena di Markov $\theta^{(1)}$ può corrispondere semplicemente ad un valore a caso tra i valori possibili del parametro.

(b) Per ogni passo successivo della catena, $m + 1$, si campiona un valore candidato $\theta'$ da una distribuzione proposta: $\theta' \sim \Pi(\theta)$. La distribuzione proposta può essere qualunque distribuzione, anche se, idealmente, è meglio che sia simile alla distribuzione a posteriori. In pratica la distribuzione a posteriori è sconosciuta e quindi il valore $\theta'$ viene campionato da una qualche distribuzione simmetrica centrata sul valore corrente $\theta^{(m)}$ del parametro. Nell'esempio che verrà discusso qui sotto, useremo la distribuzione Normale. Tale distribuzione sarà centrata sul valore corrente della catena e avrà una appropriata deviazione standard: $\theta' \sim \mathcal{N}(\theta^{(m)}, \sigma)$. In pratica, questo significa che, se $\sigma$ è piccola, il valore candidato $\theta'$ sarà simile al valore corrente $\theta^{(m)}$.

(c) Una volta generato il valore candidato $\theta'$ si calcola il rapporto tra la densità della distribuzione a posteriori non normalizzata nel punto $\theta'$ [ovvero, il prodotto tra la verosimiglianza $\mathcal{L}(y \mid \theta')$ nel punto $\theta'$ e la distribuzione a priori nel punto $\theta'$] e la densità della distribuzione a posteriori non normalizzata nel punto $\theta^{(m)}$ [ovvero, il prodotto tra la verosimiglianza $\mathcal{L}(y \mid \theta^{(m)})$ nel punto $\theta^{(m)}$ e la distribuzione a priori nel punto $\theta^{(m)}$]:

$$\alpha = \frac{p(y \mid \theta')p(\theta')}{p(y \mid \theta^{(m)})p(\theta^{(m)})}. \tag{1.5}$$

Si noti che, essendo un rapporto, la (1.5) cancella la costante di normalizzazione.

(d) Il rapporto $\alpha$ viene utilizzato per decidere se accettare il valore candidato $\theta'$, oppure se campionare un diverso candidato. Possiamo pensare al rapporto $\alpha$ come alla risposta alla seguente domanda: alla luce dei dati, è più plausibile il valore candidato del parametro o il valore corrente? Se $\alpha$ è maggiore di 1 ciò significa che il valore candidato è più plausibile del valore corrente; in tali circostanze il valore candidato viene sempre accettato. Altrimenti, si decide di accettare il valore candidato con una probabilità minore di 1, ovvero non sempre, ma soltanto con una probabilità uguale ad $\alpha$. Se $\alpha$ è uguale a 0.10, ad esempio, questo significa che la plausibilità a posteriori del valore candidato è 10 volte più piccola della plausibilità a posteriori del valore corrente. Dunque, il valore candidato verrà accettato solo nel 10% dei casi. Come conseguenza di questa strategia di scelta, l'algoritmo di Metropolis ottiene un campione casuale dalla distribuzione a posteriori, dato che la probabilità di accettare il valore candidato è proporzionale alla densità del candidato nella distribuzione a posteriori. Dal punto di vista algoritmico, la procedura descritta sopra viene implementata confrontando il rapporto $\alpha$ con un valore casuale estratto da una distribuzione uniforme Unif$(0, 1)$. Se $\alpha > u \sim$ Unif$(0, 1)$ allora il punto candidato $\theta'$ viene accettato e la catena si muove in quella nuova posizione, ovvero $\theta^{(m+1)} = \theta'^{(m+1)}$. Altrimenti $\theta^{(m+1)} = \theta^{(m)}$ e si campiona un nuovo valore candidato $\theta'$.

(e) Il passaggio finale dell'algoritmo calcola l'*accettanza* in una specifica esecuzione dell'algoritmo, ovvero la proporzione dei valori candidati $\theta'$ che sono stati accettati come valori successivi nella sequenza.

L'algoritmo di Metropolis prende come input il numero $M$ di passi da simulare, la deviazione standard $\sigma$ della distribuzione proposta e la densità a priori, e ritorna come output la sequenza $\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(M)}$. La chiave del successo dell'algoritmo di Metropolis è il numero di passi fino a che la catena approssima la stazionarietà. Tipicamente i primi da 1000 a 5000 elementi sono scartati. Dopo un certo periodo $k$ (detto di *burn-in*), la catena di Markov converge ad una variabile casuale che è distribuita secondo la distribuzione a posteriori. In altre parole, i campioni del vettore $\left(\theta^{(k+1)}, \theta^{(k+2)}, \ldots, \theta^{(M)}\right)$ diventano campioni di $p(\theta \mid y)$.

### 1.4.6 Una applicazione concreta

Per fare un esempio concreto, consideriamo nuovamente i 30 pazienti esaminati da Zetsche et al. (2019) e discussi nel Paragrafo 2. Di essi, 23 hanno manifestato aspettative distorte negativamente sul loro stato d'animo futuro. Utilizzando l'algoritmo di Metropolis, ci poniamo il problema di ottenere la stima a posteriori di $\theta$ (probabilità di manifestare un'aspettativa distorta negativamente), dati 23 "successi" in 30 prove. Verrà imposta a $\theta$ la stessa distribuzione a priori usata nel Paragrafo 2.5.

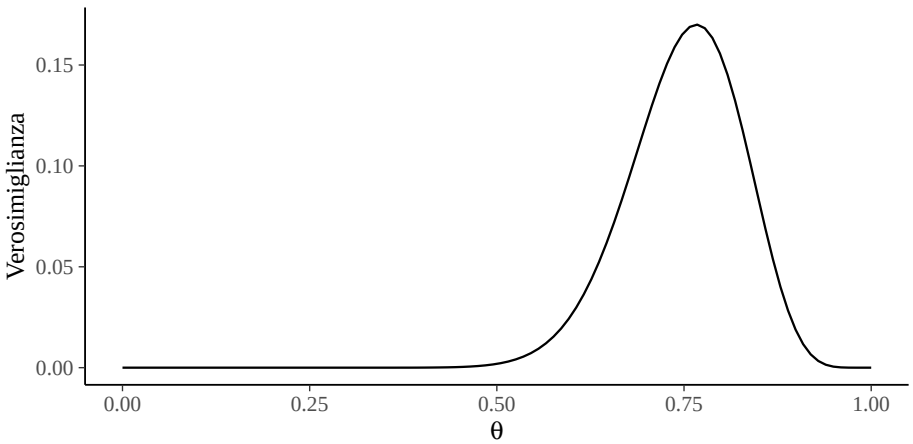
#### 1.4.6.1 Verosimiglianza

Per calcolare la funzione di verosimiglianza per i 30 valori di Zetsche et al. (2019) useremo la funzione `likelihood()`:

```
x <- 23
N <- 30
param <- seq(0, 1, length.out = 100)

likelihood <- function(param, x = 23, N = 30) {
  dbinom(x, N, param)
}

tibble(
  x = param,
  y = likelihood(param)
) %>%
  ggplot(aes(x, y)) +
  geom_line() +
  labs(
    x = expression(theta),
    y = "Verosimiglianza"
  )
```



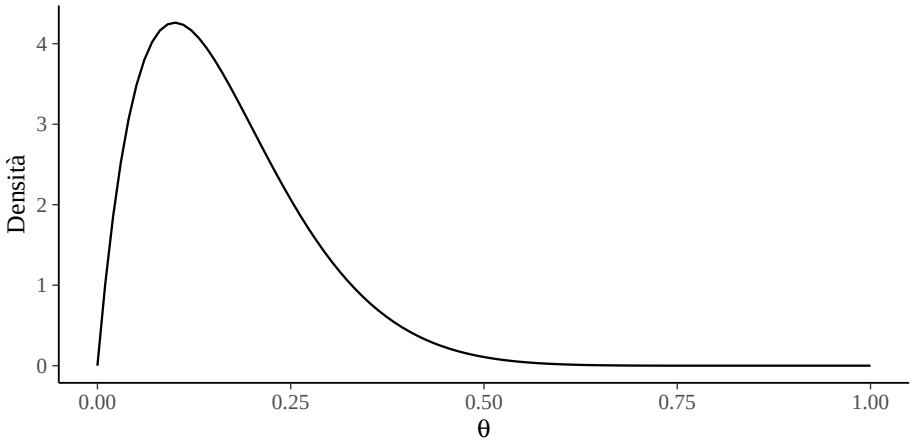

La funzione `likelihood()` ritorna l'ordinata della verosimiglianza binomiale per ciascun valore del vettore `param` in input.

#### 1.4.6.2 Distribuzione a priori

Quale distribuzione a priori utilizzeremo una Beta$(2, 10)$ implementata nella funzione `prior()`:

```r
prior <- function(param, alpha = 2, beta = 10) {
  param_vals <- seq(0, 1, length.out = 100)
  dbeta(param, alpha, beta) # / sum(dbeta(param_vals, alpha, beta))
}

tibble(
  x = param,
  y = prior(param)
) %>%
  ggplot(aes(x, y)) +
  geom_line() +
  labs(
    x = expression(theta),
    y = "Densità"
  )
```



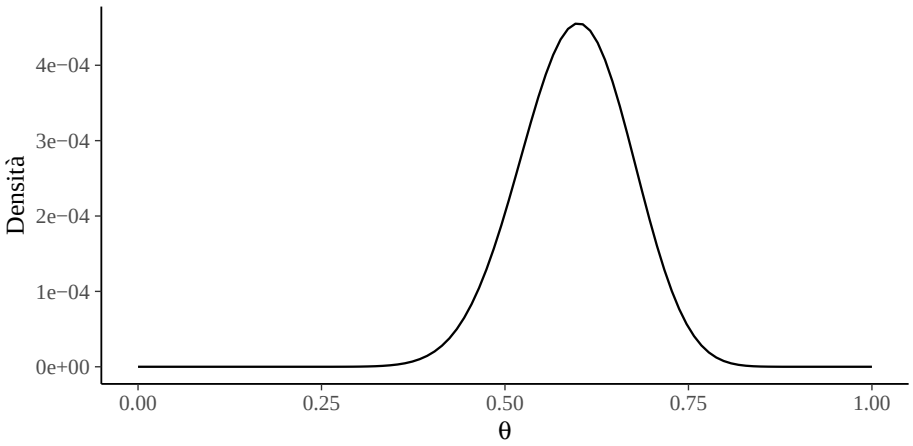

#### 1.4.6.3 Distribuzione a posteriori

La funzione a posteriori è data dal prodotto della densità a priori e della verosimiglianza ed è implementata nella funzione `posterior()`:

```r
posterior <- function(param) {
  likelihood(param) * prior(param)
}

tibble(
  x = param,
  y = posterior(param)
) %>%
```

```
  ggplot(aes(x, y)) +
  geom_line() +
  labs(
    x = expression(theta),
    y = "Densità"
  )
```




Il risultato della figura precedente è stato ottenuto con il metodo basato su griglia. Vogliamo ora replicare tale risultato usando l'algoritmo di Metropolis. La figura indica che la media a posteriori è pari a circa 0.6. Questo è il valore a posteriori più verosimile per il parametro $\theta$ alla luce dei dati osservati se assumiamo una distribuzione a priori Beta$(2, 10)$. Ci poniamo ora il problema di replicare tale risultato usando l'algoritmo di Metropolis.

#### 1.4.6.4 Implementazione

Per implementare l'algoritmo di Metropolis utilizzeremo una distribuzione proposta Normale. Il valore candidato sarà dunque un valore selezionato a caso da una Normale di parametri $\mu$ uguale al valore corrente nella catena e $\sigma = 0.9$. In questo esempio, la deviazione standard $\sigma$ è stata scelta empiricamente in modo tale da ottenere una accettanza adeguata. L'accettanza ottimale è di circa 0.20 e 0.30 — se l'accettanza è troppo grande, l'algoritmo esplora uno spazio troppo ristretto della distribuzione a posteriori.[10]

```
proposal_distribution <- function(param) {
  while(1) {
    res = rnorm(1, mean = param, sd = 0.9)
```

[10] L'accettanza dipende dalla distribuzione proposta: in generale, tanto più la distribuzione proposta è simile alla distribuzione target, tanto più alta diventa l'accettanza.

```
    if (res > 0 & res < 1)
      break
  }
  res
}
```

Nella presente implementazione del campionamento dalla distribuzione proposta è stato inserito un controllo che impone al valore candidato di essere incluso nell'intervallo $[0, 1]$.[11]

L'algoritmo di Metropolis viene implementato nella seguente funzione:

```
run_metropolis_MCMC <- function(startvalue, iterations) {
  chain <- vector(length = iterations + 1)
  chain[1] <- startvalue
  for (i in 1:iterations) {
    proposal <- proposal_distribution(chain[i])
    r <- posterior(proposal) / posterior(chain[i])
    if (runif(1) < r) {
      chain[i + 1] <- proposal
    } else {
      chain[i + 1] <- chain[i]
    }
  }
  chain
}
```

Avendo definito le funzioni precedenti, generiamo una catena di valori $\theta$:

```
set.seed(123)
startvalue <- runif(1, 0, 1)
niter <- 1e4
chain <- run_metropolis_MCMC(startvalue, niter)
```

Mediante le istruzioni precedenti otteniamo una catena di Markov costituita da 10,001 valori. Escludiamo i primi 5,000 valori considerati come burn-in. Ci restano dunque con 5,001 valori che verranno considerati come un campione casuale estratto dalla distribuzione a posteriori $p(\theta \mid y)$.

L'accettanza è pari a

```
burnIn <- niter / 2
acceptance <- 1 - mean(duplicated(chain[-(1:burnIn)]))
```

[11] Si possono trovare implementazioni dell'algoritmo di Metropolis più eleganti di quella presentata qui. Lo scopo dell'esercizio è quello di illustrare la logica soggiacente all'algoritmo di Metropolis, non quello di proporre un'implementazione efficente dell'algoritmo.

```
acceptance
#> [1] 0.2511498
```

il che conferma la bontà della deviazione standard ($\sigma = 0.9$) scelta per la distribuzione proposta.

A questo punto è facile ottenere una stima a posteriori del parametro $\theta$. Per esempio, la stima della media a posteriori è:

```
mean(chain[-(1:burnIn)])
#> [1] 0.5921799
```

Una figura che mostra l'approssimazione di $p(\theta \mid y)$ ottenuta con l'algoritmo di Metropolis, insieme ad un *trace plot* dei valori della catena di Markov, viene prodotta usando le seguenti istruzioni:

```
p1 <- tibble(
  x = chain[-(1:burnIn)]
  ) %>%
  ggplot(aes(x)) +
  geom_histogram() +
  labs(
    x = expression(theta),
    y = "Frequenza",
    title = "Distribuzione a posteriori"
  ) +
  geom_vline(
    xintercept = mean(chain[-(1:burnIn)])
  )

p2 <- tibble(
  x = 1:length(chain[-(1:burnIn)]),
  y = chain[-(1:burnIn)]
  ) %>%
  ggplot(aes(x, y)) +
  geom_line() +
  labs(
    x = "Numero di passi",
    y = expression(theta),
    title = "Valori della catena"
  ) +
  geom_hline(
    yintercept = mean(chain[-(1:burnIn)]),
    colour = "gray"
  )
```

```
p1 + p2
```

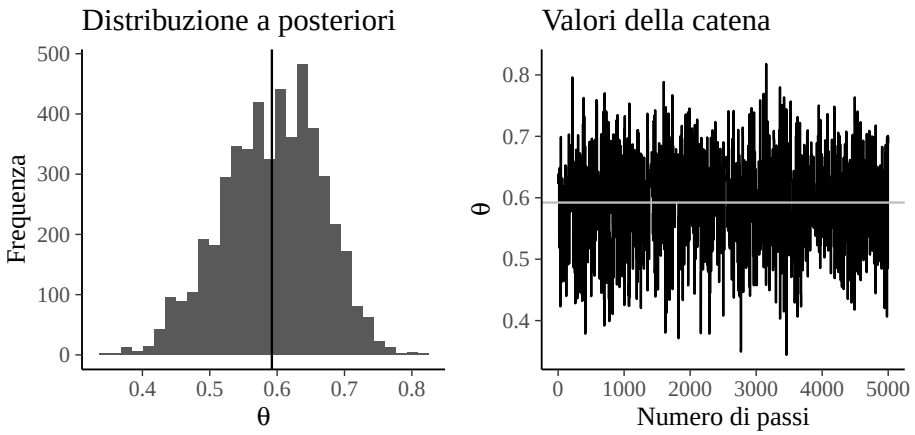



***Figura 1.9:*** *Sinistra. Stima della distribuzione a posteriori della probabilità di una aspettativa futura distorta negativamente per i dati di @zetschefuture2019. Destra. Trace plot dei valori della catena di Markov escludendo il periodo di burn-in.*

### 1.4.7 Input

Negli esempi discussi in questo Capitolo abbiamo illustrato l'esecuzione di una singola catena in cui si parte un unico valore iniziale e si raccolgono i valori simulati da molte iterazioni. È possibile che i valori di una catena siano influenzati dalla scelta del valore iniziale. Quindi una raccomandazione generale è di eseguire l'algoritmo di Metropolis più volte utilizzando diversi valori di partenza. In questo caso, si avranno più catene di Markov. Confrontando le proprietà delle diverse catene si esplora la sensibilità dell'inferenza alla scelta del valore di partenza. I software MCMC consentono sempre all'utente di specificare diversi valori di partenza e di generare molteplici catene di Markov.

## 1.5 Stazionarietà

Un punto importante da verificare è se il campionatore ha raggiunto la sua distribuzione stazionaria. La convergenza di una catena di Markov alla distribuzione stazionaria viene detta "mixing".

### 1.5.1 Autocorrelazione

Informazioni sul "mixing" della catena di Markov sono fornite dall'autocorrelazione. L'autocorrelazione misura la correlazione tra i valori successivi di una catena di Markov. Il valore $m$-esimo della serie ordinata viene confrontato con

un altro valore ritardato di una quantità $k$ (dove $k$ è l'entità del ritardo) per verificare quanto si correli al variare di $k$. L'autocorrelazione di ordine 1 (*lag 1*) misura la correlazione tra valori successivi della catena di Markow (cioè, la correlazione tra $\theta^{(m)}$ e $\theta^{(m-1)}$); l'autocorrelazione di ordine 2 (*lag 2*) misura la correlazione tra valori della catena di Markow separati da due "passi" (cioè, la correlazione tra $\theta^{(m)}$ e $\theta^{(m-2)}$); e così via.

L'autocorrelazione di ordine $k$ è data da $\rho_k$ e può essere stimata come:

$$\begin{aligned}\rho_k &= \frac{\text{Cov}(\theta_m, \theta_{m+k})}{\text{Var}(\theta_m)} \\ &= \frac{\sum_{m=1}^{n-k}(\theta_m - \bar{\theta})(\theta_{m-k} - \bar{\theta})}{\sum_{m=1}^{n-k}(\theta_m - \bar{\theta})^2} \qquad \text{con} \quad \bar{\theta} = \frac{1}{n}\sum_{m=1}^{n}\theta_m. \end{aligned} \tag{1.6}$$

Per fare un esempio pratico, simuliamo dei dati autocorrelati con la funzione R `colorednoise::colored_noise()`:

```
suppressPackageStartupMessages(library("colorednoise"))
set.seed(34783859)
rednoise <- colored_noise(
  timesteps = 30, mean = 0.5, sd = 0.05, phi = 0.3
)
```

L'autocorrelazione di ordine 1 è semplicemente la correlazione tra ciascun elemento e quello successivo nella sequenza. Nell'esempio, il vettore `rednoise` è una sequenza temporale di 30 elementi. Il vettore `rednoise[-length(rednoise)]` include gli elementi con gli indici da 1 a 29 nella sequenza originaria, mentre il vettore `rednoise[-1]` include gli elementi 2:30. Gli elementi delle coppie ordinate dei due vettori avranno dunque gli indici $(1, 2), (2, 3), \ldots (29, 30)$ degli elementi della sequenza originaria. La correlazione di Pearson tra i vettori `rednoise[-length(rednoise)]` e `rednoise[-1]` corrisponde dunque all'autocorrelazione di ordine 1 della serie temporale.

```
cor(rednoise[-length(rednoise)], rednoise[-1])
#> [1] 0.3967366
```

Il Correlogramma è uno strumento grafico usato per la valutazione della tendenza di una catena di Markov nel tempo. Il correlogramma si costruisce a partire dall'autocorrelazione $\rho_k$ di una catena di Markov in funzione del ritardo (*lag*) $k$ con cui l'autocorrelazione è calcolata: nel grafico ogni barretta verticale riporta il valore dell'autocorrelazione (sull'asse delle ordinate) in funzione del ritardo (sull'asse delle ascisse). In R, il correlogramma può essere prodotto con una chiamata a `acf()`:

```
acf(rednoise)
```


Il correlogramma precedente mostra come l'autocorrelazione di ordine 1 sia circa pari a 0.4 e diminuisce per lag maggiori; per lag di 4, l'autocorrelazione diventa negativa e aumenta progressivamente fino ad un lag di 8; eccetera.

In situazioni ottimali l'autocorrelazione diminuisce rapidamente ed è effettivamente pari a 0 per piccoli lag. Ciò indica che i valori della catena di Markov che si trovano a più di soli pochi passi di distanza gli uni dagli altri non risultano associati tra loro, il che fornisce conferma del "mixing" della catena di Markov, ossia di convergenza alla distribuzione stazionaria. Nelle analisi bayesiane, una delle strategie che consentono di ridurre l'autocorrelazione è quella di assottigliare l'output immagazzinando solo ogni $m$-esimo punto dopo il periodo di burn-in. Una tale strategia va sotto il nome di *thinning*.

### 1.5.2 Test di convergenza

Un test di convergenza può essere svolto in maniera grafica mediante le tracce delle serie temporali (*trace plot*), cioè il grafico dei valori simulati rispetto al numero di iterazioni. Se la catena è in uno stato stazionario le tracce mostrano assenza di periodicità nel tempo e ampiezza costante, senza tendenze visibili o andamenti degni di nota. Un esempio di *trace plot* è fornito nella figura 1.9 (destra).

Ci sono inoltre alcuni test che permettono di verificare la stazionarietà del campionatore dopo un dato punto. Uno è il test di Geweke che suddivide il campione, dopo aver rimosso un periodo di burn in, in due parti. Se la catena è in uno stato stazionario, le medie dei due campioni dovrebbero essere uguali. Un test modificato, chiamato Geweke z-score, utilizza un test $z$ per confrontare i due subcampioni ed il risultante test statistico, se ad esempio è più alto di 2, indica che la media della serie sta ancora muovendosi da un punto ad un altro e quindi è necessario un periodo di burn-in più lungo.

## Considerazioni conclusive

In generale, la distribuzione a posteriori dei parametri di un modello statistico non può essere determinata per via analitica. Tale problema, invece, viene affrontato facendo ricorso ad una classe di algoritmi per il campionamento da distribuzioni di probabilità che sono estremamente onerosi dal punto di vista computazionale e che possono essere utilizzati nelle applicazioni pratiche solo grazie alla potenza di calcolo dei moderni computer. Lo sviluppo di software che rendono sempre più semplice l'uso dei metodi MCMC, insieme all'incremento della potenza di calcolo dei computer, ha contribuito a rendere sempre più popolare il metodo dell'inferenza bayesiana che, in questo modo, può essere estesa a problemi di qualunque grado di complessità.

Nel 1989 un gruppo di statistici nel Regno Unito si pose il problema di simulare le catene di Markov su un personal computer. Nel 1997 ci riuscirono con il primo rilascio pubblico di un'implementazione Windows dell'inferenza bayesiana basata su Gibbs sampling, detta BUGS. Il materiale presentato in questo capitolo descrive gli sviluppi contemporanei del percorso che è iniziato in quel periodo.

*Capitolo 2*

# Aspettative degli individui depressi

Per fare pratica, applichiamo il metodo basato su griglia ad un campione di dati reali. Zetsche et al. (2019) si sono chiesti se gli individui depressi manifestino delle aspettative accurate circa il loro umore futuro, oppure se tali aspettative siano distorte negativamente. Esamineremo qui i 30 partecipanti dello studio di Zetsche et al. (2019) che hanno riportato la presenza di un episodio di depressione maggiore in atto. All'inizio della settimana di test, a questi pazienti è stato chiesto di valutare l'umore che si aspettavano di esperire nei giorni seguenti della settimana. Mediante una app, i partecipanti dovevano poi valutare il proprio umore in cinque momenti diversi di ciascuno dei cinque giorni successivi. Lo studio considera diverse emozioni, ma qui ci concentriamo solo sulla tristezza.

Sulla base dei dati forniti dagli autori, abbiamo calcolato la media dei giudizi relativi al livello di tristezza raccolti da ciascun partecipante tramite la app. Tale media è stata poi sottratta dall'aspettativa del livello di tristezza fornita all'inizio della settimana. Per semplificare l'analisi abbiamo considerato la discrepanza tra aspettative e realtà come un evento dicotomico: valori positivi di tale differenza indicano che le aspettative circa il livello di tristezza sono maggiori del livello di tristezza che in seguito viene effettivamente esperito; ciò significa che le aspettative sono negativamente distorte (evento codificato con "1"). Si può dire il contrario (le aspettative sono positivamente distorte) se tale differenza assume valori negativi (evento codificato con "0").

Nel campione dei 30 partecipanti clinici esaminati da Zetsche et al. (2019), 23 partecipanti manifestano delle aspettative negativamente distorte e 7 partecipanti manifestano delle aspettative positivamente distorte. Nella seguente discussione, chiameremo $\theta$ la probabilità dell'evento "le aspettative del partecipante sono distorte negativamente". Il problema che ci poniamo è quello di ottenere la stima a posteriori di $\theta$, avendo osservato 23 "successi" in 30 prove, ovvero $\hat{\theta} = 23/30 = 0.77$.

## 2.1 La griglia

Fissiamo una griglia di $n = 50$ valori equispaziati nell'intervallo [0, 1] per il parametro $\theta$:

```
n_points <- 50
p_grid <- seq(from = 0, to = 1, length.out = n_points)
p_grid
#>  [1] 0.00000000 0.02040816 0.04081633 0.06122449 0.08163265
#>  [6] 0.10204082 0.12244898 0.14285714 0.16326531 0.18367347
#> [11] 0.20408163 0.22448980 0.24489796 0.26530612 0.28571429
#> [16] 0.30612245 0.32653061 0.34693878 0.36734694 0.38775510
#> [21] 0.40816327 0.42857143 0.44897959 0.46938776 0.48979592
#> [26] 0.51020408 0.53061224 0.55102041 0.57142857 0.59183673
#> [31] 0.61224490 0.63265306 0.65306122 0.67346939 0.69387755
#> [36] 0.71428571 0.73469388 0.75510204 0.77551020 0.79591837
#> [41] 0.81632653 0.83673469 0.85714286 0.87755102 0.89795918
#> [46] 0.91836735 0.93877551 0.95918367 0.97959184 1.00000000
```

## 2.2 Distribuzione a priori

Supponiamo di avere scarse credenze a priori sulla tendenza di un individuo clinicamente depresso a manifestare delle aspettative distorte negativamente circa il suo umore futuro. Imponiamo quindi una distribuzione non informativa sulla distribuzione a priori di $\theta$ — ovvero, una distribuzione uniforme nell'intervallo [0, 1]. Dato che consideriamo soltanto $n = 50$ valori possibili per il parametro $\theta$, creiamo un vettore di 50 elementi che conterrà i valori della distribuzione a priori scalando ciascun valore del vettore per $n$ in modo tale che la somma di tutti i valori sia uguale a 1.0:

```
prior1 <- dbeta(p_grid, 1, 1) / sum(dbeta(p_grid, 1, 1))
prior1
#>  [1] 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02
#> [12] 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02
#> [23] 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02
#> [34] 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.02
#> [45] 0.02 0.02 0.02 0.02 0.02 0.02
```

Verifichiamo:

```
sum(prior1)
#> [1] 1
```

La distribuzione a priori così costruita è rappresentata nella figura 2.1.

```
p1 <- data.frame(p_grid, prior1) %>%
  ggplot(aes(x=p_grid, xend=p_grid, y=0, yend=prior1)) +
  geom_line() +
  geom_segment() +
  ylim(0, 0.17) +
  labs(
    x = "Parametro \U03B8",
    y = "Probabilità a priori",
    title = "50 punti"
  )
p1
```

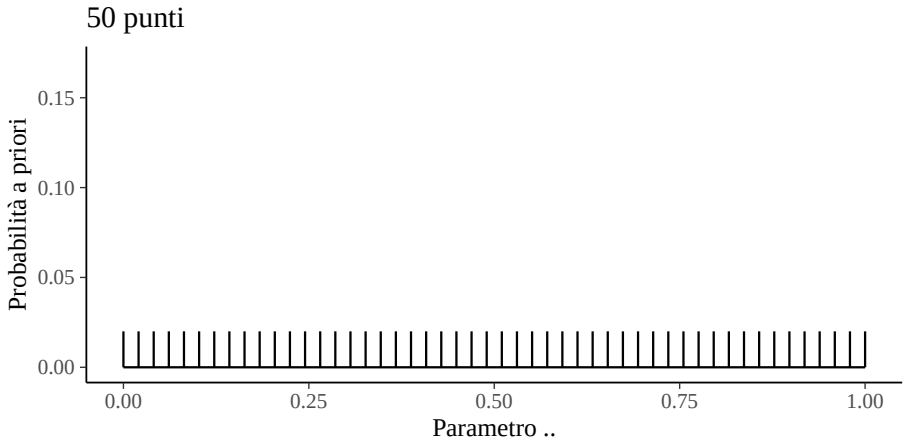



***Figura 2.1:*** *Rappresentazione grafica della distribuzione a priori per il parametro heta, ovvero la probabilità di aspettative future distorte negativamente.*

## 2.3 Funzione di verosimiglianza

Calcoliamo ora la funzione di verosimiglianza utilizzando i 50 valori $\theta$ definiti in precedenza. A ciascuno dei valori della griglia applichiamo la formula binomiale, tendendo costanti i dati (ovvero 23 "successi" in 30 prove). Ad esempio, in corrispondenza del valore $\theta = 0.816$, l'ordinata della funzione di verosimiglianza diventa

$$\binom{30}{23} \cdot 0.816^{23} \cdot (1 - 0.816)^{7} = 0.135.$$

Per $\theta = 0.837$, l'ordinata della funzione di verosimiglianza sarà

$$\binom{30}{23} \cdot 0.837^{23} \cdot (1 - 0.837)^{7} = 0.104.$$

Dobbiamo svolgere questo calcolo per tutti gli elementi della griglia. Usando R, tale risultato si trova nel modo seguente:

```
likelihood <- dbinom(x = 23, size = 30, prob = p_grid)
likelihood
#>  [1] 0.000000e+00 2.352564e-33 1.703051e-26 1.644169e-22
#>  [5] 1.053708e-19 1.525217e-17 8.602222e-16 2.528440e-14
#>  [9] 4.606907e-13 5.819027e-12 5.499269e-11 4.105534e-10
#> [13] 2.520191e-09 1.311195e-08 5.919348e-08 2.362132e-07
#> [17] 8.456875e-07 2.749336e-06 8.196948e-06 2.259614e-05
#> [21] 5.798673e-05 1.393165e-04 3.148623e-04 6.720574e-04
#> [25] 1.359225e-03 2.611870e-03 4.778973e-03 8.340230e-03
#> [29] 1.390025e-02 2.214199e-02 3.372227e-02 4.909974e-02
#> [33] 6.830377e-02 9.068035e-02 1.146850e-01 1.378206e-01
#> [37] 1.568244e-01 1.681749e-01 1.688979e-01 1.575211e-01
#> [41] 1.348746e-01 1.043545e-01 7.133007e-02 4.165680e-02
#> [45] 1.972669e-02 6.936821e-03 1.535082e-03 1.473375e-04
#> [49] 1.868105e-06 0.000000e+00
```

La funzione `dbinom(x, size, prob)` richiede che vengano specificati tre parametri: il numero di "successi", il numero di prove e la probabilità di successo. Nella chiamata precedente, `x` (numero di successi) e `size` (numero di prove bernoulliane) sono degli scalari e `prob` è il vettore `p_grid`. In tali circostanze, l'output di `dbinom()` è il vettore che abbiamo chiamato `likelihood`. Gli elementi di tale vettore sono stati calcolati applicando la formula della distribuzione binomiale a ciascuno dei 50 elementi della griglia, tenendo sempre costanti i dati [ovvero, `x` (il numero di successi) e `size` (numero di prove bernoulliane)]; ciò che varia è il valore `prob`, che assume valori diversi (`p_grid`) in ciascuna cella della griglia.

La chiamata a `dbinom()` produce dunque un vettore i cui valori corrispondono all'ordinata della funzione di verosimiglianza per per ciascun valore $\theta$ specificato in `p_grid`. La verosimiglianza discretizzata così ottenuta è riportata nella figura 2.2.

```
p2 <- data.frame(p_grid, likelihood) %>%
  ggplot(aes(x=p_grid, xend=p_grid, y=0, yend=likelihood)) +
  geom_segment() +
  ylim(0, 0.17) +
  labs(
    x = "Parametro \U03B8",
    y = "Verosimiglianza"
  )
p2
```

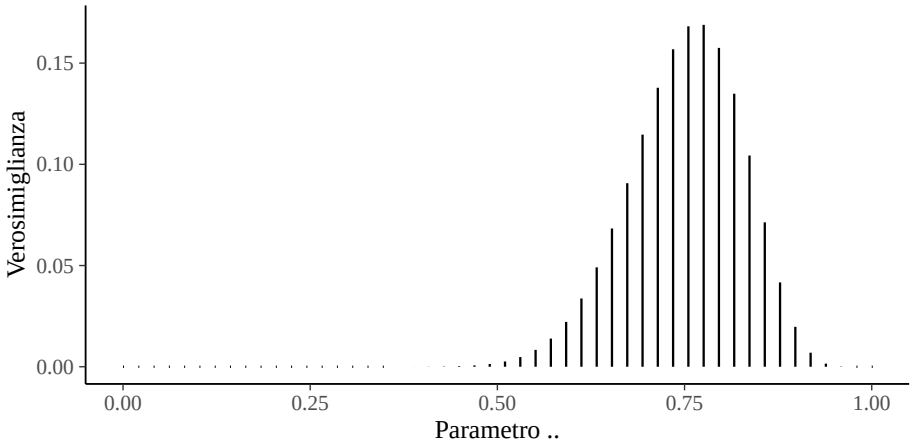



***Figura 2.2:*** *Rappresentazione della funzione di verosimiglianza per il parametro $\theta$, ovvero la probabilità di aspettative future distorte negativamente.*

## 2.4 Distribuzione a posteriori

L'approssimazione discretizzata della distribuzione a posteriori $p(\theta \mid y)$ si ottiene facendo il prodotto della verosimiglianza e della distribuzione a priori per poi scalare tale prodotto per una costante di normalizzazione. Il prodotto $p(\theta)\mathcal{L}(y \mid \theta)$ produce la distribuzione a posteriori *non standardizzata*.

Nel caso di una distribuzione a priori non informativa (ovvero una distribuzione uniforme), per ottenere la funzione a posteriori non standardizzata è sufficiente moltiplicare ciascun valore della funzione di verosimiglianza per 0.02. Per esempio, per il primo valore della funzione di verosimiglianza usato quale esempio poco sopra, abbiamo $0.135 \cdot 0.02$; per il secondo valore dell'esempio abbiamo $0.104 \cdot 0.02$; e così via. Possiamo svolgere tutti i calcoli usando R nel modo seguente:[1]

```
unstd_posterior <- likelihood * prior1
unstd_posterior
#>  [1] 0.000000e+00 4.705127e-35 3.406102e-28 3.288337e-24
#>  [5] 2.107415e-21 3.050433e-19 1.720444e-17 5.056880e-16
#>  [9] 9.213813e-15 1.163805e-13 1.099854e-12 8.211068e-12
#> [13] 5.040382e-11 2.622390e-10 1.183870e-09 4.724263e-09
#> [17] 1.691375e-08 5.498671e-08 1.639390e-07 4.519229e-07
#> [21] 1.159735e-06 2.786331e-06 6.297247e-06 1.344115e-05
#> [25] 2.718450e-05 5.223741e-05 9.557946e-05 1.668046e-04
```

[1] Ricordiamo il principio dell'aritmetica vettorializzata: i vettori `likelihood` e `prior1` sono entrambi costituiti da 50 elementi. Se facciamo il prodotto tra i due vettori otteniamo un vettore di 50 elementi, ciascuno dei quali uguale al prodotto dei corrispondenti elementi dei vettori `likelihood` e `prior1`.

```
#> [29] 2.780049e-04 4.428398e-04 6.744454e-04 9.819948e-04
#> [33] 1.366075e-03 1.813607e-03 2.293700e-03 2.756411e-03
#> [37] 3.136488e-03 3.363497e-03 3.377958e-03 3.150422e-03
#> [41] 2.697491e-03 2.087091e-03 1.426601e-03 8.331361e-04
#> [45] 3.945339e-04 1.387364e-04 3.070164e-05 2.946751e-06
#> [49] 3.736209e-08 0.000000e+00
```

Avendo calcolato i valori della funzione a posteriori non standardizzata è poi necessario dividere per una costante di normalizzazione. Nel caso discreto, trovare il denominatore del teorema di Bayes è facile: esso è uguale alla somma di tutti i valori della distribuzione a posteriori non normalizzata. Per i dati presenti, tale costante di normalizzazione è uguale a 0.032:

```
sum(unstd_posterior)
#> [1] 0.0316129
```

La standardizzazione dei due valori usati come esempio è data da: $0.135 \cdot 0.02/0.032$ e da $0.104 \cdot 0.02/0.032$. Usiamo R per svolgere questo calcolo su tutti i 50 valori di `unstd_posterior` così che la somma dei 50 i valori di `posterior` sia uguale a 1.0:

```
posterior <- unstd_posterior / sum(unstd_posterior)
posterior
#>  [1] 0.000000e+00 1.488357e-33 1.077440e-26 1.040188e-22
#>  [5] 6.666313e-20 9.649330e-18 5.442222e-16 1.599625e-14
#>  [9] 2.914574e-13 3.681425e-12 3.479129e-11 2.597379e-10
#> [13] 1.594406e-09 8.295316e-09 3.744893e-08 1.494410e-07
#> [17] 5.350268e-07 1.739376e-06 5.185824e-06 1.429552e-05
#> [21] 3.668548e-05 8.813904e-05 1.991986e-04 4.251792e-04
#> [25] 8.599178e-04 1.652408e-03 3.023432e-03 5.276472e-03
#> [29] 8.794033e-03 1.400820e-02 2.133450e-02 3.106310e-02
#> [33] 4.321259e-02 5.736920e-02 7.255582e-02 8.719259e-02
#> [37] 9.921545e-02 1.063963e-01 1.068538e-01 9.965619e-02
#> [41] 8.532881e-02 6.602021e-02 4.512719e-02 2.635430e-02
#> [45] 1.248015e-02 4.388601e-03 9.711744e-04 9.321354e-05
#> [49] 1.181862e-06 0.000000e+00
```

Verifichiamo:

```
sum(posterior)
#> [1] 1
```

La distribuzione a posteriori così trovata non è altro che la versione normalizzata della funzione di verosimiglianza: questo avviene perché la distribuzione a priori uniforme non ha aggiunto altre informazioni oltre a quelle che erano

già fornite dalla funzione di verosimiglianza. L'approssimazione discretizzata di $p(\theta \mid y)$ che abbiamo appena trovato è riportata nella figura 2.3.

```
p3 <- data.frame(p_grid, posterior) %>%
  ggplot(aes(x=p_grid, xend=p_grid, y=0, yend=posterior)) +
  geom_segment() +
  ylim(0, 0.17) +
  labs(
    x = "Parametro \U03B8",
    y = "Probabilità a posteriori"
  )
p3
```

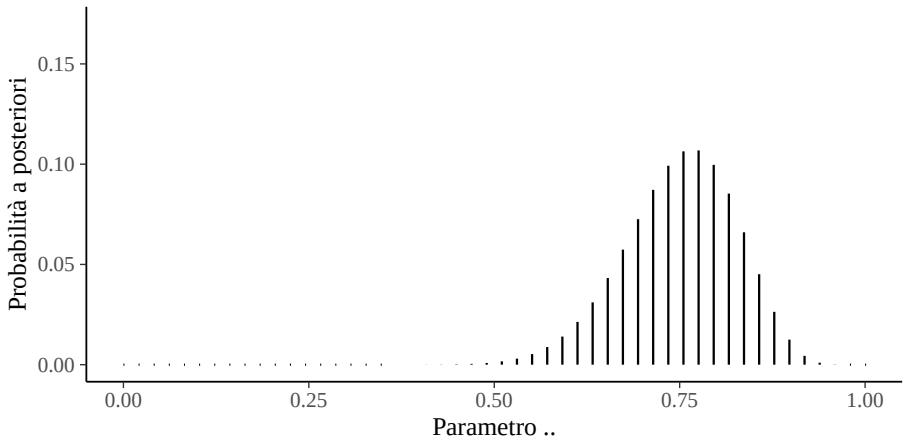



***Figura 2.3:*** *Rappresentazione della distribuzione a posteriori per il parametro $\theta$, ovvero la probabilità di aspettative future distorte negativamente.*

I grafici delle figure 2.1, 2.2 e 2.3 sono state calcolati utilizzando una griglia di 50 valori equi-spaziati per il parametro $\theta$. I segmenti verticali rappresentano l'intensità della funzione in corrispondenza di ciascuna modalità parametro $\theta$. Nella figura 2.1 e nella figura 2.3 la somma delle lunghezze dei segmenti verticali è uguale a 1.0; ciò non si verifica, invece, nel caso della figura 2.3 (la funzione di verosimiglianza non è mai una funzione di probabilità, né nel caso discreto né in quello continuo).

## 2.5 La stima della distribuzione a posteriori (versione 2)

Continuiamo l'analisi di questi dati esaminiamo l'impatto di una distribuzione a priori informativa sulla distribuzione a posteriori. Una distribuzione a priori informativa riflette un alto grado di certezza a priori sui valori dei parametri del modello. Un ricercatore utilizza una distribuzione a priori informativa

per introdurre nel processo di stima informazioni pre-esistenti alla raccolta dei dati, introducendo così delle restrizioni sulla possibile gamma di valori del parametro.

Nel caso presente, supponiamo che la letteratura psicologica fornisca delle informazioni su $\theta$ (la probabilità che le aspettative future di un individuo clinicamente depresso siano distorte negativamente). Per fare un esempio, supponiamo (irrealisticamente) che tali conoscenze pregresse possano essere rappresentate da una Beta di parametri $\alpha = 2$ e $\beta = 10$. Tali ipotetiche conoscenze pregresse ritengono molto plausibili valori $\theta$ bassi e considerano implausibili valori $\theta > 0.5$. Questo è equivalente a dire che ci aspettiamo che le aspettative relative all'umore futuro siano distorte negativamente solo per pochissimi individui clinicamente depressi — ovvero, ci aspettiamo che la maggioranza degli individui clinicamente depressi sia inguaribilmente ottimista. Questa è, ovviamente, una credenza a priori del tutto irrealistica. La esamino qui, non perché abbia alcun senso nel contesto dei dati di Zetsche et al. (2019), ma soltanto per fare un esempio nel quale risulta chiaro come la distribuzione a posteriori sia una sorta di "compromesso" tra la distribuzione a priori e la verosimiglianza.

Con calcoli del tutto simili a quelli descritti sopra si giunge alla distribuzione a posteriori rappresentata nella figura 2.4. Useremo ora una griglia di 100 valori per il parametro $\theta$:

```
n_points <- 100
p_grid <- seq(from = 0, to = 1, length.out = n_points)
```

Per la distribuzione a priori scegliamo una Beta(2, 10):

```
alpha <- 2
beta <- 10
prior2 <- dbeta(p_grid, alpha, beta) / sum(dbeta(p_grid, alpha, beta))
sum(prior2)
#> [1] 1
```

Tale distribuzione a priori è rappresentata nella figura 2.4:

```
plot_df <- data.frame(p_grid, prior2)
p4 <- plot_df %>%
  ggplot(aes(x=p_grid, xend=p_grid, y=0, yend=prior2)) +
  geom_segment() +
  ylim(0, 0.17) +
  labs(
    x = "",
    y = "Probabilità a priori"
  )
p4
```

Calcoliamo il valore di verosimiglianza per ciascun punto della griglia:

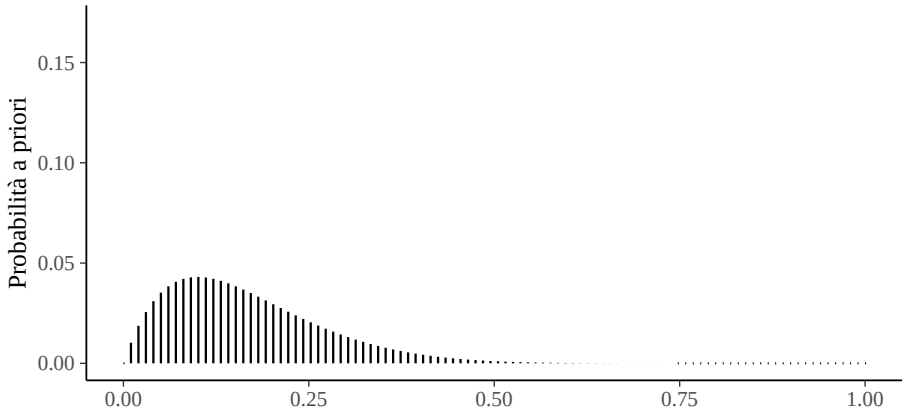



***Figura 2.4:*** *Rappresentazione di una funzione a priori informativa per il parametro $\theta$.*

```
likelihood <- dbinom(23, size = 30, prob = p_grid)
```

Per ciascun punto della griglia, il prodotto tra la verosimiglianza e distribuzione a priori è dato da:

```
unstd_posterior2 <- likelihood * prior2
```

È necessario normalizzare la distribuzione a posteriori discretizzata:

```
posterior2 <- unstd_posterior2 / sum(unstd_posterior2)
```

Verifichiamo:

```
sum(posterior2)
#> [1] 1
```

La nuova funzione a posteriori discretizzata è rappresentata nella figura 2.5:

```
plot_df <- data.frame(p_grid, posterior2)
p5 <- plot_df %>%
  ggplot(aes(x = p_grid, xend = p_grid, y = 0, yend = posterior2)) +
  geom_segment() +
  ylim(0, 0.17) +
  labs(
    x = "Parametro \U03B8",
    y = "Probabilità a posteriori"
  )
p5
```

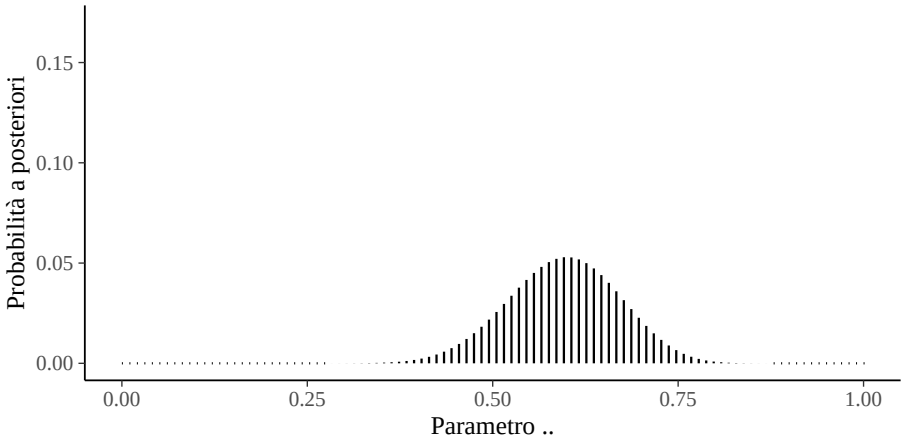



***Figura 2.5:*** *Rappresentazione della funzione a posteriori per il parametro $\theta$ calcolata utilizzando una distribuzione a priori informativa.*

Facendo un confronto tra le figure 2.4 e 2.5 notiamo una notevole differenza tra la distribuzione a priori e la distribuzione a posteriori. In particolare, la distribuzione a posteriori risulta spostata verso destra su posizioni più vicine a quelle della verosimiglianza [figura 2.2]. Si noti inoltre che, a causa dell'effetto della distribuzione a priori, le distribuzioni a posteriori delle figure 2.3 e 2.5 sono molto diverse tra loro.

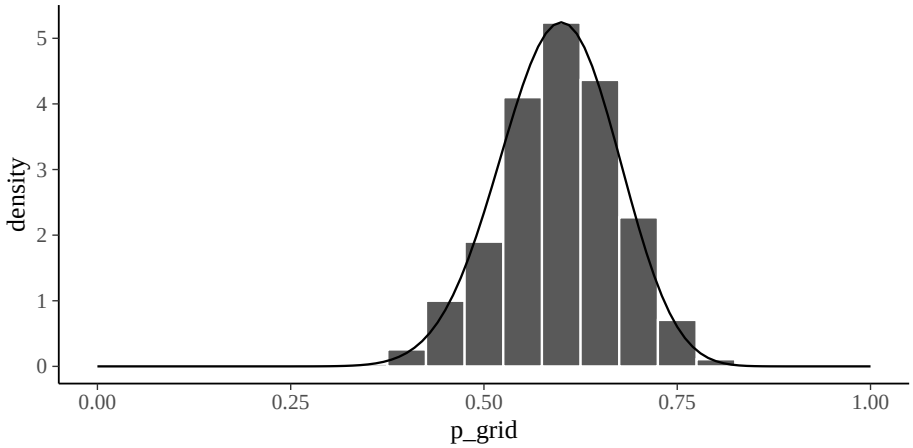
Campioniamo ora 10,000 punti dall'approssimazione discretizzata della distribuzione a posteriori:

```
# Set the seed
set.seed(84735)

df <- data.frame(
  p_grid,
  posterior2
)
# Step 4: sample from the discretized posterior
post_samples <- df %>%
  slice_sample(
  n = 1e5,
  weight_by = posterior2,
  replace = TRUE
)
```

Una rappresentazione grafica del campione casuale estratto dalla distribuzione a posteriori $p(\theta \mid y)$ è data da:

```
post_samples %>%
  ggplot(aes(x = p_grid)) +
  geom_histogram(
    aes(y = ..density..),
    color = "white",
    binwidth = 0.05
  ) +
  stat_function(fun = dbeta, args = list(25, 17)) +
  lims(x = c(0, 1))
```




All'istogramma è stata sovrapposta la corretta distribuzione a posteriori, ovvero una Beta di parametri 25 ($y + \alpha = 23 + 2$) e 17 ($n - y + \beta = 30$ - $23 + 10$).

La stima della moda a posteriori si ottiene con

```
df$p_grid[which.max(df$posterior2)]
#> [1] 0.5959596
```

e corrisponde a

$$\text{Mo} = \frac{\alpha - 1}{\alpha + \beta - 2} = \frac{25 - 1}{25 + 17 - 2} = 0.6.$$

La stima della media a posteriori si ottiene con

```
mean(post_samples$p_grid)
#> [1] 0.5953337
```

e corrisponde a

$$\bar{\theta} = \frac{\alpha}{\alpha + \beta} = \frac{25}{25 + 17} \approx 0.5952.$$

La stima della mediana a posteriori si ottiene con

```
median(post_samples$p_grid)
#> [1] 0.5959596
```

e corrisponde a

$$\text{Me} = \frac{\alpha - \frac{1}{3}}{\alpha + \beta - \frac{2}{3}} \approx 0.5968.$$

# Bibliografia


Albert, J. and Hu, J. (2019). *Probability and Bayesian Modeling*. Chapman and Hall/CRC.

Johnson, A. A., Ott, M., and Dogucu, M. (2022). *Bayes Rules! An Introduction to Bayesian Modeling with R*. CRC Press.

Kruschke, J. (2014). *Doing Bayesian data analysis: A tutorial with R, JAGS, and Stan*. Academic Press.

McElreath, R. (2020). *Statistical rethinking: A Bayesian course with examples in R and Stan*. CRC Press, Boca Raton, Florida, 2nd edition edition.

Zetsche, U., Bürkner, P.-C., and Renneberg, B. (2019). Future expectations in clinical depression: Biased or realistic? *Journal of Abnormal Psychology*, 128(7):678–688.