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 θ 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 θ 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:

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

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

- Selezionare N valori casuali della griglia in modo tale da ottenere un campione casuale delle densità a posteriori normalizzate.

1.1.1 Modello Beta-Binomiale

Supponiamo di avere osservato 9 sucessi in 10 prove Bernoulliane indipendenti.[2] Imponiamo alla distribuzione a priori su θ (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 ($n - y + \beta = 10 - 9 + 2$):

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

Iniziamo a definire i valori θ della griglia: $\theta \in \{0, 0.2, 0.4, 0.6, 0.8, 1\}$ (in questo primo esempio considereremo solo 6 valori):

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

In corrispondenza di ciascuno dei 6 valori 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 non normalizzata del distribuzione a posteriori. Successivamente normalizziamo questa approssimazione dividendo ciascun valore a posteriori non normalizzato per la somma di tutti i valori a posteriori non normalizzati:

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

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

Confermiamo che la somma dei valori a posteriori sia 1:

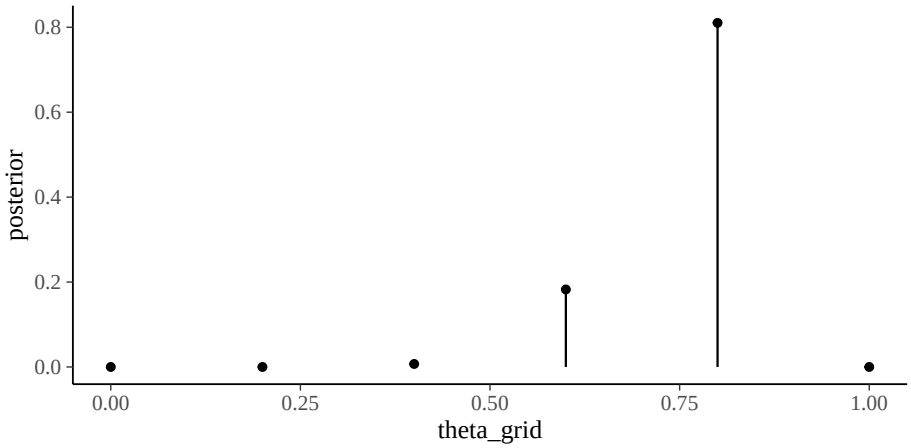
```
grid_data %>%
  summarize(
    sum(unnormalized),
    sum(posterior)
  )
#> # A tibble: 1 x 2
#>   `sum(unnormalized)` `sum(posterior)`
#>                 <dbl>            <dbl>
#> 1               0.318                1
```

Otteniamo dunque 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
```

Un grafico della distribuzione a posteriori discretizzata è dato da:

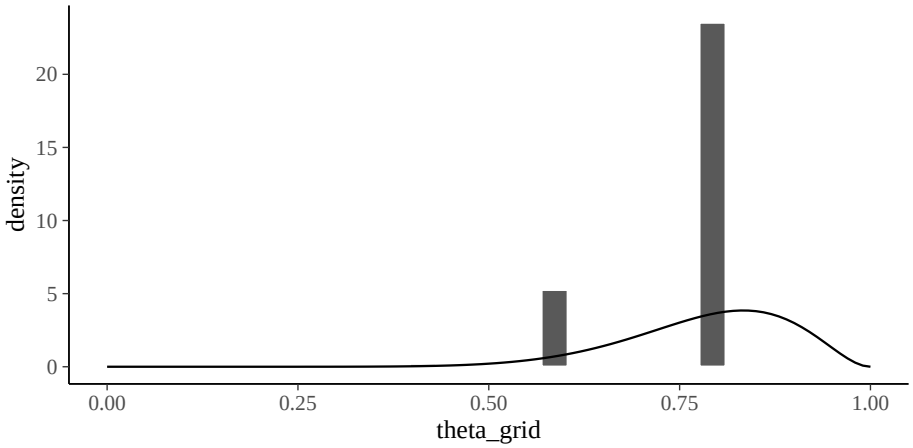
```
grid_data %>%
  ggplot(
    aes(x = theta_grid, y = posterior)
  ) +
  geom_point() +
  geom_segment(
    aes(
      x = theta_grid,
      xend = theta_grid,
      y = 0,
      yend = posterior)
  )
```



Ora possiamo campionare dalla distribuzione a posteriori discretizzata. 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 successivo a cui è statsa sovrapposta la vera distribuzione a posteriori Beta$(11, 3)$:

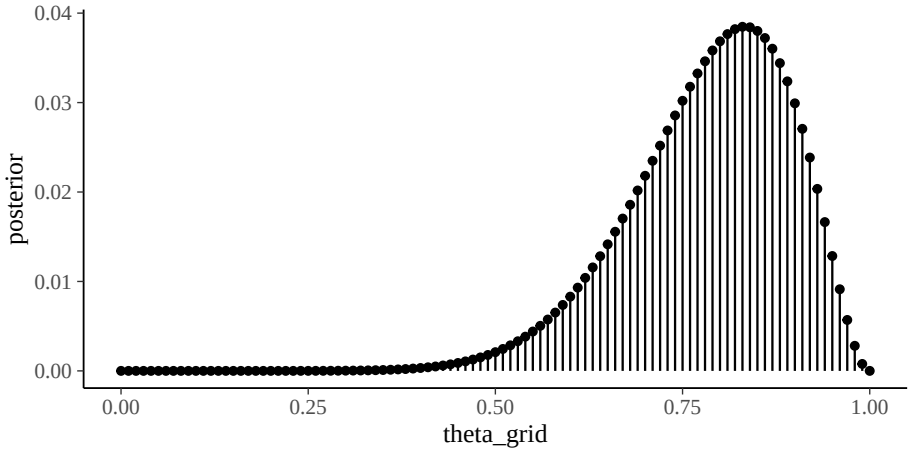
```
set.seed(84735)

post_sample <- sample_n(
  grid_data,
  size = 1e5,
  weight = posterior,
  replace = TRUE
)
```

```
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 precedente 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:

```
grid_data  <- data.frame(theta_grid = seq(from = 0, to = 1, length = 101))

grid_data <- grid_data %>%
  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))

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

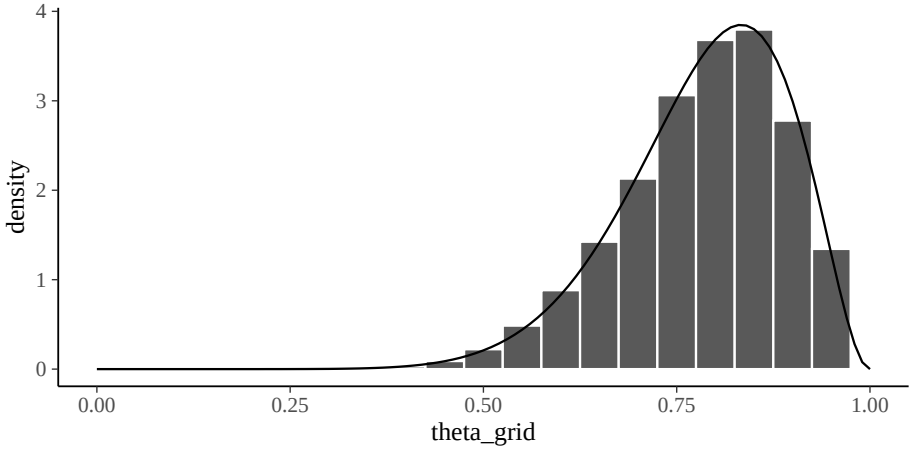


Campioniamo ora 10000 punti:

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

# Step 4: sample from the discretized posterior
post_sample <- sample_n(
  grid_data,
  size = 1e4,
  weight = posterior,
  replace = TRUE
)
```

Con una griglia più densa abbiamo ottenuto un risultato soddisfacente: la posizione e la distribuzione dei valori prodotti dalla simulazione corrispondono da vicino a quelli della distribuzione a posteriori $p(\theta \mid y)$.

```
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))
```



Come messo in evidenza da Johnson et al. (2022), il metodo basato su griglia è molto intuitivo e non richiede particolari competenze di programmazione per essere implementato. Soprattutto, il risultato è costituito da un campione che, per tutti gli scopi pratici, può essere inteso come un campione casuale estratto da $p(\theta \mid y)$. Tuttavia, anche se tale metodo può fornire risultati accuratissimi, il suo uso è molto limitato. A causa della *maledizione della dimensionalità*[3], infatti, possiamo fare ricorso a tale procedura numerica per la stima di $p(\theta \mid y)$ solo nel caso di modelli statistici semplici, con non più di due parametri. Nella pratica concreta tale metodo viene sostituito da altre tecniche più efficienti in quanto, anche nei comuni modelli utilizzati in psicologia, vengono solitamente stimati centinaia se non migliaia di parametri.

1.1.2 Aspettative degli individui depressi

Per fare pratica, applichiamo il metodo basato su griglia anche ad un'altro set di dati. 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

[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.

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 θ 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 θ, avendo osservato 23 "successi" in 30 prove, ovvero $\hat{\theta} = 23/30 = 0.77$.

Iniziamo a costruire la griglia di valori del parametro θ. In questo esempio considereremo 50 valori egualmente spaziati nell'intervallo [0, 1]: 0.000, 0.0204, ..., 0.978, 1.000. Per ottenere i valori griglia procediamo nel modo seguente:

```
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
```

1.1.2.1 Distribuzione a priori

Supponiamo che le nostre credenze a priori sulla tendenza di un individuo clinicamente depresso a manifestare delle aspettative distorte negativamente circa il suo umore futuro siano molto scarse. Imponiamo quindi su θ una distribuzione a priori non informativa – ovvero, ipotizziamo che la distribuzione a priori sia una distribuzione uniforme nell'intervallo [0, 1]. Dato che consideriamo soltanto $n = 50$ valori possibili per il parametro θ, 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 (in questo modo viene definita una funzione di massa di probabilità):

```
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
#> [12] 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
#> [34] 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 1.1.

```
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
```

1.1.2.2 Funzione di verosimiglianza

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

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

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

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

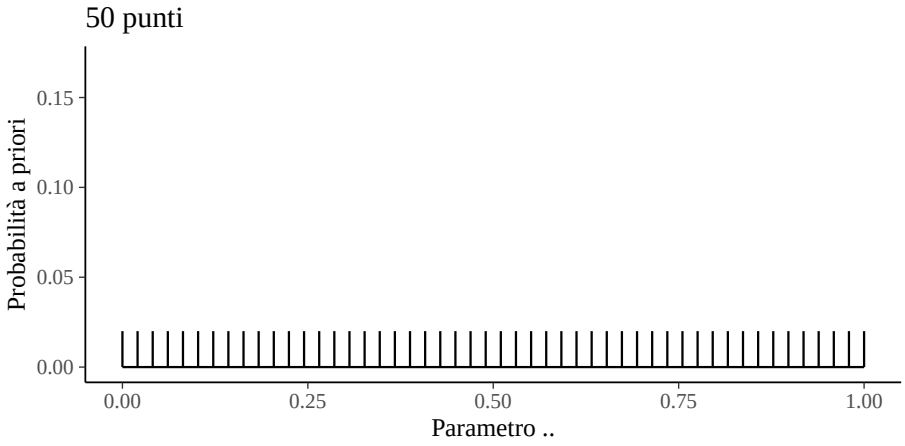


Figura 1.1: *Rappresentazione grafica della distribuzione a priori per il parametro heta, ovvero la probabilità di aspettative future distorte negativamente.*

Dobbiamo svolgere questo calcolo per tutti gli elementi della griglia. Usando R il risultato cercato 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
```

Il vettore `likelihood` è stato ottenuto passando alla funzione `dbinom()` un vettore di valori, ovvero gli elementi della griglia `p_grid`. 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. Dato che `x` (numero di successi) e `size` (numero di prove bernoulliane) sono degli scalari e `prob` è un vettore, questo significa che la formula della probabilità binomiale verrà applicata a ciascun elemento di `p_grid` tenendo costanti i valori di `x` e `size`, ovvero i dati. In questo modo otteniamo in output un vettore i cui valori corrispondono all'ordinata della funzione di verosimiglianza per il corrispondente valore gri-

glia di θ. La funzione di verosimiglianza così ottenuta è riportata nella figura 1.2.

```
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
```

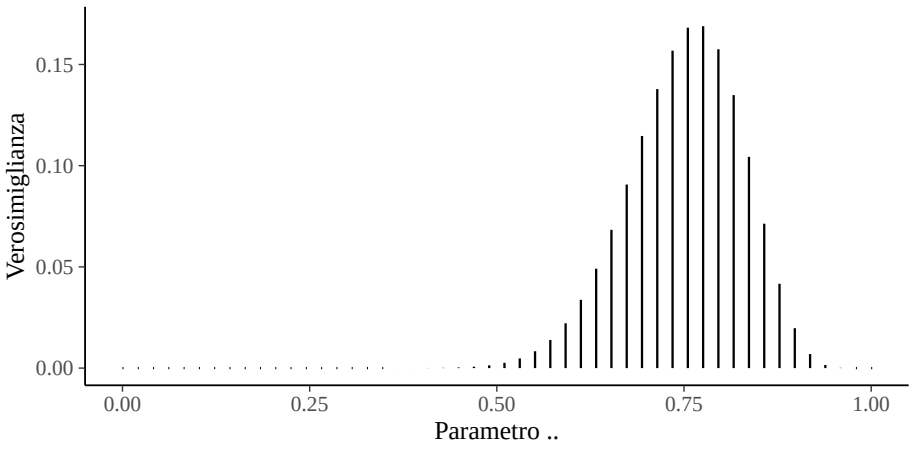


Figura 1.2: *Rappresentazione della funzione di verosimiglianza per il parametro θ, ovvero la probabilità di aspettative future distorte negativamente.*

1.1.2.3 Distribuzione a posteriori

L'approssimazione discretizzata della distribuzione a posteriori $p(\theta \mid y)$ si ottiene facendo prima il prodotto della verosimiglianza e della distribuzione a priori, e poi scalando tale prodotto per una costante di normalizzazione. Quindi, 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 in precedenza, abbiamo $0.135 \cdot 0.02$; per il secondo valore della funzione di verosimiglianza usato come esempio abbiamo $0.104 \cdot 0.02$; e così via. Possiamo svolgere tutti i calcoli usando R nel modo seguente:

```
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
#> [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
```

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`.

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 negli esempi sopra si ottiene con: $0.135 \cdot 0.02/0.032$ e $0.104 \cdot 0.02/0.032$. Usiamo R per svolgere questo calcolo su tutti i 50 valori di `unstd_posterior`:

```
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
```

In questo modo la somma di tutti e 50 i valori di `posterior` è uguale a 1.0. Verifichiamo:

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

Nell'esempio, la distribuzione a posteriori trovata come descritto sopra 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 1.3.

```
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
```

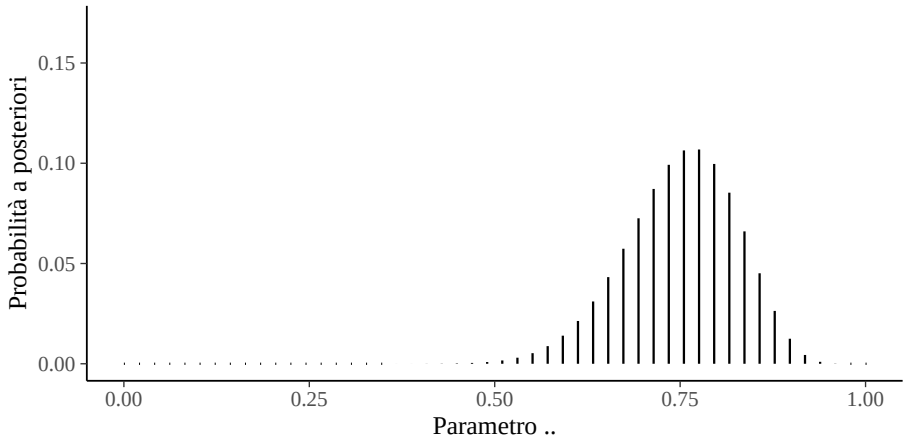


Figura 1.3: *Rappresentazione della distribuzione a posteriori per il parametro θ, ovvero la probabilità di aspettative future distorte negativamente.*

I grafici delle figure 1.1, 1.2 e 1.3 sono state calcolati utilizzando una griglia di 50 valori equi-spaziati per il parametro θ. I segmenti verticali rappresentano l'intensità della funzione in corrispondenza di ciascuna modalità parametro θ. Nella figura 1.1 e nella figura 1.3 la somma delle lunghezze dei segmenti verticali è pari ad 1.0; ciò non si verifica, invece, nel caso della figura 1.3 (la funzione di verosimiglianza non è mai una funzione di probabilità, né nel caso discreto né in quello continuo).

1.1.2.4 La stima della distribuzione a posteriori (versione 2)

Continuiamo la discussione dell'esempio precedente ed esaminiamo l'impatto sulla distribuzione a posteriori di una distribuzione a priori informativa. Una distribuzione a priori informativa riflette un alto grado di certezza sui parametri del modello da stimare. Un ricercatore può utilizzare una distribuzione a priori informativa per introdurre nel processo di stima le informazioni pre-esistenti alla raccolta dei dati, le quali introducono delle restrizioni sulla possibile gamma di valori possibili del parametro.

Nel caso presente, supponiamo che la letteratura psicologica fornisca delle informazioni su θ, ovvero sulla probabilità che le aspettative future di un individuo clinicamente depresso siano distorte negativamente. In tali circostanze, anziché imporre su $p(\theta)$ una distribuzione a priori non informativa, il ricercatore può utilizzare una distribuzione a priori informativa. 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 θ 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 1.4. Useremo una griglia di 100 valori per il parametro θ:

```
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 1.4:

```
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
```

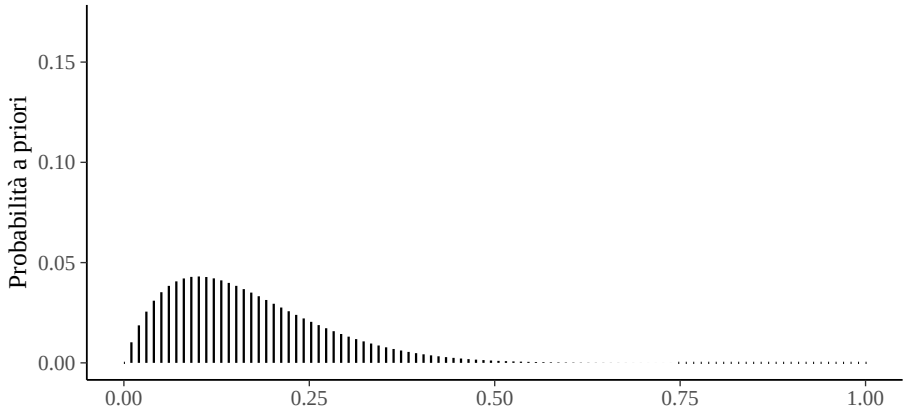


Figura 1.4: *Rappresentazione di una funzione a priori informativa per il parametro θ.*

Calcoliamo ora il valore della funzione di verosimiglianza in corrispondenza di ciascun punto della griglia:

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

Il prodotto tra la verosimiglianza e la distribuzione a priori per ciascun punto della griglia si ottiene con:

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

È necessario normalizzare la distribuzione a posteriori in modo tale che la somma dei valori sia 1:

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

Verifichiamo:

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

La nuova funzione a posteriori è rappresentata nella figura 1.5:

```
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
```

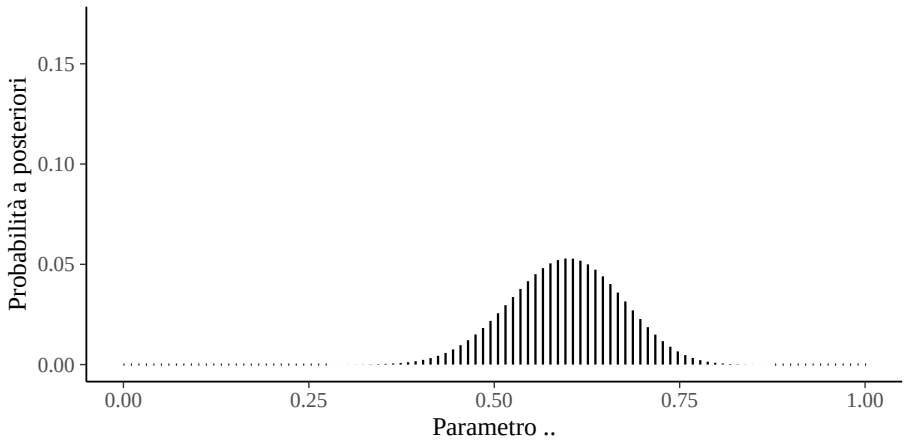


Figura 1.5: *Rappresentazione della funzione a posteriori per il parametro θ calcolata utilizzando una distribuzione a priori informativa.*

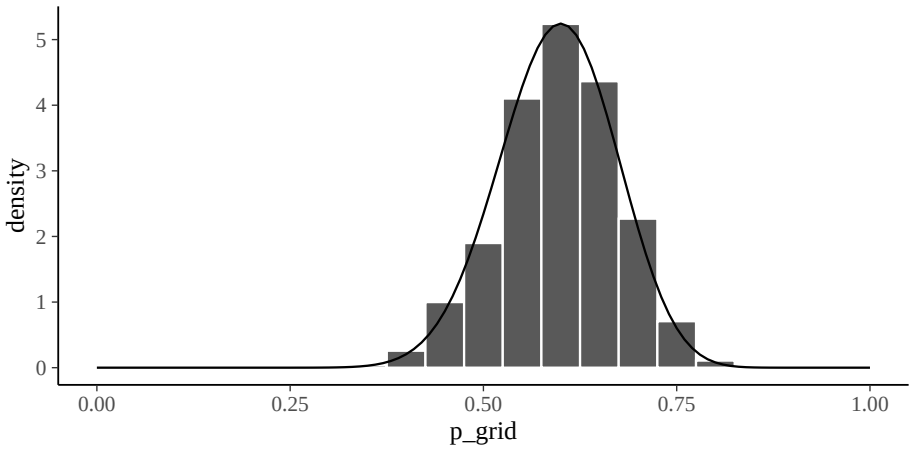
Facendo un confronto tra le figure 1.4 e 1.5 notiamo la differenza tra la distribuzione a priori θ e la distribuzione a posteriori per θ. In particolare, la distribuzione a posteriori rappresentata nella 1.5 risulta spostata verso destra su posizioni più vicine a quelle della verosimiglianza, rappresentata nella figura 1.2. Si noti inoltre che, a causa dell'effetto della distribuzione a priori, le distribuzioni a posteriori riportate nelle figure 1.3 e 1.5 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 abbiamo sovrapposto una curva nera che rappresenta la corretta distribuzione a posteriori che è una Beta di parametri 25 ($y + \alpha = 23 + 2$) e 17 ($n - y + \beta = 30$ - 23 + 10).

La stima della moda della distribuzione a posteriori si ottiene con

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

e corrisponde a

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

Una stima della media della distribuzione 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 mediana a posteriori si ottiene con

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

e corrisponde a

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

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

L'approssimazione quadratica si pone due obiettivi.

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`. Tale pacchetto, creato da Richard McElreath per accompagnare il suo testo *Statistical Rethinking*[2], può essere scaricato utilizzando le istruzioni seguenti:

```
install.packages(c("coda", "mvtnorm", "devtools", "loo", "dagitty"))
library("devtools")
devtools::install_github("rmcelreath/rethinking")
```

È possibile che, per i diversi sistemi operativi, sia necessaria l'installazione di componenti ulteriori. Si veda https://github.com/rmcelreath/rethinking.

L'approssimazione quadratica fornisce risultati simili (o identici) a quelli ottenuti con il metodo *grid-based*. Il vantaggio dell'approssimazione quadratica è che disponiamo di una serie di funzioni R che svolgono tutti i calcoli per noi. Per trovare la funzione a posteriori di θ per i dati discussi nelle sezioni precedenti possiamo usare la funzione `rethinking::quap()`. La sintassi è la seguente:

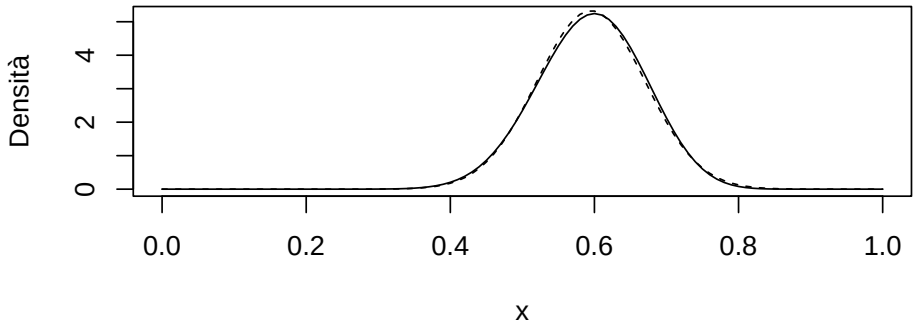
```
suppressPackageStartupMessages(library("rethinking"))

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 0.6000005 0.07745926 0.4481831 0.7518178
```

Qui sotto è fornita una rappresentazione grafica della distribuzione a posteriori corretta (linea continua) e quella ottenuta mediante l'approssimazione quadratica (linea trateggiata).

```r
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,
  add = TRUE
)
```



In realtà, l'approssimazione quadratica è poco usata in pratica, perché per problemi complessi è più conveniente usare i metodi Monte Carlo basati su Catena di Markov (MCMC) che verranno descritti nella successiva sezione.

1.3 Simulazione Monte Carlo

Nel capitolo **??** abbiamo visto che, nel caso di una verosimiglianza è Binomiale e una distribuzione a priori Beta, la distribuzione a posteriori è anch'essa una Beta, e con una simulazione R possiamo facilmente ricavare dei campioni causali dalla distribuzione a posteriori. Più campioni otteniamo, meglio possiamo approssimare la distribuzione a posteriori basata sui campioni prodotti dalla simulazione. È la stessa ragione per cui, se abbiamo un campione molto ampio, possiamo descrivere in modo molto preciso la popolazione da cui il campione è stato estratto; nel caso presente, la vera distribuzione a posteriori corrisponde alla popolazione e i campioni prodotti dalla simulazione sono i campioni casuali estratti dalla popolazione. Con 10,000 o 100,000 campioni possiamo descrivere la distribuzione a posteriori molto accuratamente.

Per esempio, possiamo ottenere il valore della media della distribuzione a posteriori di θ prendendo un grande numero di campioni dalla distribuzione a posteriori:

```
set.seed(7543897)
print(mean(rbeta(1e3, shape1 = 25, shape2 = 17)), 6)
#> [1] 0.596229
```

L'approssimazione migliora all'aumentare del numero di campioni:

```
print(mean(rbeta(1e4, shape1 = 25, shape2 = 17)), 6)
#> [1] 0.595709
```

```
print(mean(rbeta(1e5, shape1 = 25, shape2 = 17)), 6)
#> [1] 0.595282
```

Quindi possiamo dire che quando il numero di campioni tratti dalla distribuzione a posteriori è molto grande, la distribuzione dei campioni converge alla densità della popolazione (si veda l'Appendice **??**). Il metodo Monte Carlo funziona in molte situazioni. Si noti, naturalmente, che il numero dei campioni di simulazione è controllato dal ricercatore; è totalmente diverso dalla dimensione del campione, che è fisso ed è una proprietà dei dati.

Inoltre, la maggior parte delle statistiche descrittive (es. media, *DS*) del campione convergeranno ai corrispondenti valori della vera distribuzione a posteriori. I grafici seguenti mostrano come, all'aumentare del numero di repliche, la media, la mediana, la *DS* e l'asimmetria convergono al vero valore (linee rosse tratteggiate).

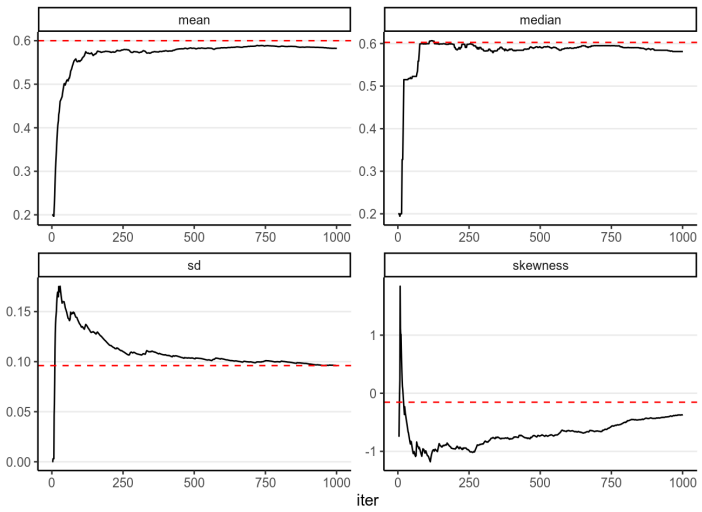


Figura 1.6: *Convergenza delle simulazioni Monte Carlo.*

1.4 Metodi MC basati su Catena di Markov

Nel Paragrafo 1.3 la simulazione Monte Carlo funzionava perché (a) sapevamo esattamente che la distribuzione a posteriori era una distribuzione Beta e (b) era possibile usare le funzioni R per estrarre campioni casuali da tale distribuzione. Tuttavia, capita raramente di potere 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, se nel caso di una verosimiglianza binomiale usiamo una distribuzione Normale per la distribuzione a priori di θ, la distribuzione a posteriori diventa

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

e non è chiaro come sia possibile estrarre dei campioni di simulazione da una tale distribuzione a posteriori.

Per fortuna, abbiamo a disposizione un algoritmo intelligente 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 seguiamo la discussione di tale tema fornita da Bob Carpenter. Supponiamo che una persona esegua una passeggiata casuale sulla retta dei numeri naturali sui valori 1, 2, 3, 4, 5, 6. Se la persona è collocata su un valore interno (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 (1 o 6), nel passo successivo è altrettanto probabile che rimanga rimanga su quel numero o si sposti nella posizione adiacente.

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, corrispondenti 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 *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 possono essere riassunte per mezzo di 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)
print(P)
#>      [,1] [,2] [,3] [,4] [,5] [,6]
#> [1,] 0.50 0.50 0.00 0.00 0.00 0.00
#> [2,] 0.25 0.50 0.25 0.00 0.00 0.00
#> [3,] 0.00 0.25 0.50 0.25 0.00 0.00
#> [4,] 0.00 0.00 0.25 0.50 0.25 0.00
#> [5,] 0.00 0.00 0.00 0.25 0.50 0.25
#> [6,] 0.00 0.00 0.00 0.00 0.50 0.50
```

La prima riga di P fornisce le probabilità di passare a tutti gli 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.

Ci sono diverse proprietà importanti 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 la persona può tornare in 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 tornare 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 lo strumento mediante il quale 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 possono quindi essere utilizzate per 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. Inoltre, come illustrato sopra, la distribuzione limite di una tale catena di Markov, quando il numero di passi tende all'infinito, sarà uguale a questa distribuzione stazionaria w.

1.4.2 Simulare una catena di Markov

Un altro metodo per dimostrare l'esistenza della distribuzione stazionaria di una catena di Markov è quello di eseguire un esperimento di simulazione. Iniziamo la nostra 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 in ciascuno dei sei possibili nodi della catena approssimano sempre meglio la distribuzione stazionaria w. Senza entrare nei dettagli della simulazione, la figura in basso raffigura 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))
```

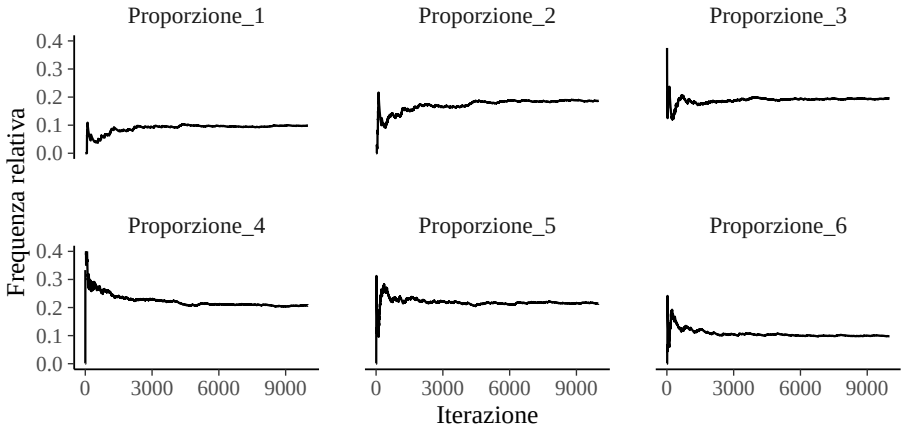


Figura 1.7: *Frequenze relative degli stati da 1 a 6 in funzione del numero di iterazioni per la simulazione di una catena di Markov.*

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 $p(\theta \mid y)$ cercata. Un modo generale per ottenerla è usare l'algoritmo di Metropolis, un algoritmo che, nelle sue varianti, risulta applicabile ad una grande varietà di problemi inferenziali di tipo bayesiano.

Presentiamo ora, in una forma intuitiva, l'algoritmo di Metropolis, ovvero il primo algoritmo MCMC che è stato proposto. Tale algoritmo è stato sviluppato in seguito allo scopo di renderlo via via più efficiente. Il nostro obiettivo qui però è solo quello di illustrare la logica soggiacente.

1.4.4 Una passeggiata casuale sui numeri naturali

Per introdurre l'algoritmo di campionamento a catena di Markov considereremo il campionamento da una distribuzione discreta.[4] Supponiamo di definire una

[4]Seguiamo qui la trattazione di Albert and Hu (2019); si vedano anche Kruschke (2014); McElreath (2020).

distribuzione di probabilità discreta sugli interi $1, \dots, K$. Ad esempio, scriviamo in R una funzione `pd()` che assegna ai valori $1, \dots, 8$ 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 <- data.frame(
  x = 1:8,
  prob = pd(1:8)
)
```

La figura 1.8 illustra la distribuzione di probabilità che abbiamo generato.

```
x <- 1:8
prob_dist %>%
  ggplot(aes(x = x, y = prob)) +
  geom_bar(stat = "identity", width = 0.06) +
  scale_x_continuous("x", labels = as.character(x), breaks = x)
```

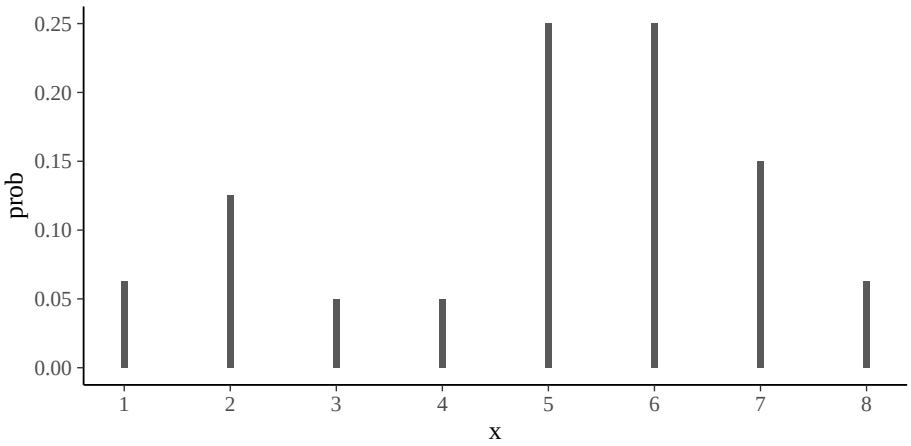


Figura 1.8: *Distribuzione di massa di probabilità per una variabile casuale avente valori 1, 2, ..., 8.*

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 successivo valore della sequenza, lanciamo una moneta equilibrata. Se esce testa, il valore candidato sarà il valore immediatamente precedente al valore corrente nella sequenza $1, 2, \dots, 8$; 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 casuale $\in [0, 1]$. 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 e la matrice di transizione P è definita dai passi da 2 a 4. Un modo di campionare da una distribuzione di massa di probabilità `pd` consiste nell'iniziare da qualsiasi posizione e eseguire un grande numero di "passegggiate casuali" ripetendo i passaggi 2, 3 e 4 (proporre una valore candidato, calcolare il rapporto e decidere se accettare il valore candidato). Se questo processo viene ripetuto per un numero elevato di volte, la distribuzione dei valori così ottenuti approssima 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 `s`.

```
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 <- data.frame(out) %>%
  group_by(out) %>%
  summarize(
    N = n(),
    Prob = N / 10000
  )

prob_dist2 <- rbind(
  prob_dist,
  data.frame(
    x = S$out,
    prob = S$Prob
    )
  )
prob_dist2$Type <- rep(c("actual", "simulated"), 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"))
```

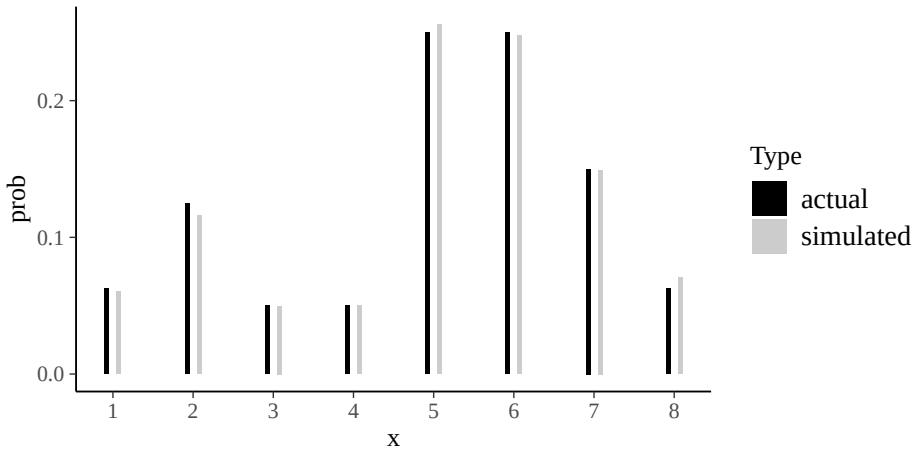


Figura 1.9: *L'istogramma confronta i valori prodotti dall'algoritmo di Metropolis con i reali valori della distribuzione.*

La Figura mette a confronto l'istogramma dei valori simulati dalla passeggiata casuale con l'effettiva distribuzione di probabilità. Si noti che le probabilità

prodotte dalla simulazione sono molto simili alle vere probabilità.

1.4.5 L'algoritmo di Metropolis

Un modo popolare di simulare da una distribuzione a posteriori consiste in una generalizzazione dell'algoritmo descritto nel Paragrafo precedente. La strategia di campionamento Monte Carlo a catena di Markov genera una catena di Markov irriducibile e aperiodica per la quale la distribuzione stazionaria è uguale alla distribuzione a posteriori di interesse. Uno dei metodi che possono essere usati a questo scopo è chiamato algoritmo Metropolis.[5]

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

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

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

o di mantenere il valore corrente

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

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 può corrispondere semplicemente all'origine, $\theta^{(1)} = 0$.

(b) Per ogni passo successivo della catena, $m + 1$, si campiona un valore candidato θ' 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 θ' viene campionato da una qualche distribuzione simmetrica centrata sul valore corrente $\theta^{(m)}$ del parametro. Nell'esempio discuso qui sotto, useremo la distribuzione Normale con una appropriata deviazione standard: $\theta' \sim \mathcal{N}(\theta^{(m)}, \sigma)$. In pratica, questo significa che, con σ piccola, il valore candidato θ' sarà simile al valore corrente $\theta^{(m)}$.

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

(c) Una volta generato il valore candidato θ' si calcola il rapporto di densità nel punto θ' e nel punto corrente $\theta^{(m)}$ che cancella la costante di normalizzazione. La (1.5) corrisponde al rapporto tra la densità della distribuzione a posteriori non normalizzata nel punto θ' [ovvero, al prodotto tra l'ordinata della verosimiglianza $\mathcal{L}(y \mid \theta')$ nel punto θ' e l'ordinata della distribuzione a priori nel punto θ'] e la densità della distribuzione a posteriori non normalizzata nel punto θ [ovvero, al prodotto tra l'ordinata della verosimiglianza $\mathcal{L}(y \mid \theta)$ nel punto θ e l'ordinata della distribuzione a priori nel punto θ]:

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

(d) Il rapporto α viene utilizzato per decidere se accettare il valore candidato θ', oppure se campionare un diverso candidato. Possiamo pensare al rapporto α come alla seguente domanda: alla luce dei dati, è più plausibile il valore candidato del parametro o il valore corrente? Se α 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 α. Se α 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 α con un valore casuale estratto da una distribuzione uniforme $U(0, 1)$. Se $\alpha > u \sim U(0, 1)$ allora il punto candidato θ' 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 θ'.

(e) Il passaggio finale dell'algoritmo calcola l'*accettanza* in una specifica esecuzione dell'algoritmo, ovvero la proporzione dei valori candidati θ' 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 σ 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 simulata converge

dunque ad una variabile casuale che è distribuita secondo la distribuzione a posteriori. In altre parole, i campioni del vettore $(\theta^{(k+1)}, \theta^{(k+2)}, \dots, \theta^{(M)})$ diventano campioni di $p(\theta \mid y)$.

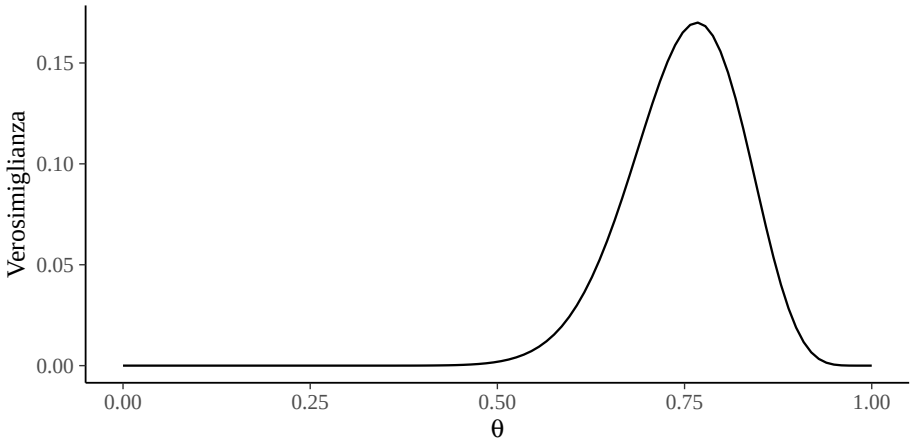
1.5 Una applicazione concreta

Per fare un esempio concreto, consideriamo nuovamente i 30 pazienti esaminati da Zetsche et al. (2019) e discussi nel Paragrafo 1.1.2. 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 θ (probabilità di manifestare un'aspettativa distorta negativamente), dati 23 "successi" in 30 prove. Verrà imposta a θ la stessa distribuzione a priori usata nel Paragrafo 1.1.2.4.

1.5.1 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)
}

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

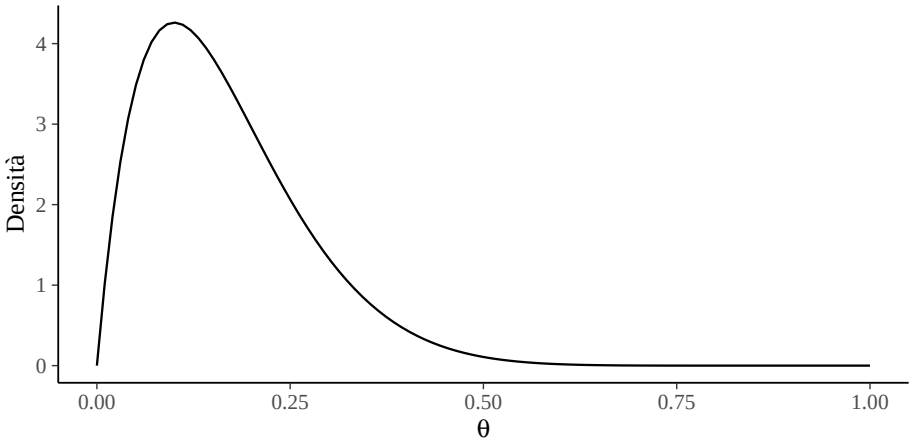


La funzione `likelihood()` ritorna l'ordinata della verosimiglianza binomiale per ciascun valore del vettore `param` in input.

1.5.2 Distribuzione a priori

Utilizziamo anche in questo esempio la distribuzione a priori informativa Beta(2, 10):

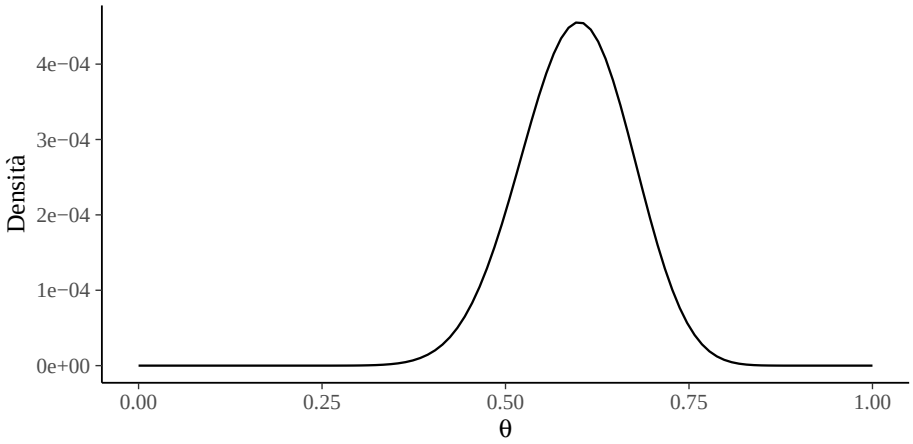
```
prior <- function(param, alpha = 2, beta = 10) {
  param_vals <- seq(0, 1, length.out = 100)
  dbeta(param, alpha, beta) # / sum(dbeta(param_vals, alpha, beta))
}

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



1.5.3 Distribuzione a posteriori

La funzione a posteriori è data dal prodotto della densità a priori e della verosimiglianza:

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

data.frame(x = param, y = posterior(param)) %>%
  ggplot(aes(x, y)) +
  geom_line() +
  labs(
    x = expression(theta),
    y = "Densità"
  )
```



Questo è il risultato che vogliamo ottenere utilizzando l'algoritmo di Metropolis. Dalla figura precedente vediamo che la moda della distribuzione a posteriori è pari a circa 0.6. Questo è il valore più verosimile a posteriori per il parametro θ considerando i dati e la distribuzione a priori Beta(2, 10).

1.5.4 Implementazione

Implementiamo ora l'algoritmo di Metropolis utilizzando la Normale quale distribuzione proposta. Il valore candidato corrisponderà dunque ad un valore selezionato a caso da una Normale di parametri μ uguale al valore corrente nella catena e σ. In questo esempio, σ 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.[6]

```
proposal_distribution <- function(param) {
  while(1) {
    res = rnorm(1, mean = param, sd = 0.9)
    if (res > 0 & res < 1)
      break
  }
  res
}
```

In questa implementazione del campionamento dalla distribuzione proposta sono stati inseriti dei controlli tali per cui il valore candidato è incluso nell'intervallo [0, 1].[7]

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

[7] Si possono trovare implementazioni migliori dell'algoritmo di Metropolis di quella presentata qui. Lo scopo di questo esercizio è solo quello di illustrare la logica soggia-

L'algoritmo di Metropolis viene implementato nella funzione seguente:

```
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 θ:

```
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 4,000 valori. Dei 4,000 valori ottenuti, escludiamo i primi 2,000 valori considerati come burn-in. Ci restano dunque con 2,000 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)]))
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 puntuale a posteriori del parametro θ. Per esempio, possiamo calcolare la media del campione casuale di $p(\theta \mid y)$ prodotto dall'algoritmo di Metropolis:

---

cente all'algoritmo di Metropolis, non quello di proporre un'implementazione efficente dell'algoritmo.

```
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 <- data.frame(
  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 <- data.frame(
  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
```

1.5.5 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

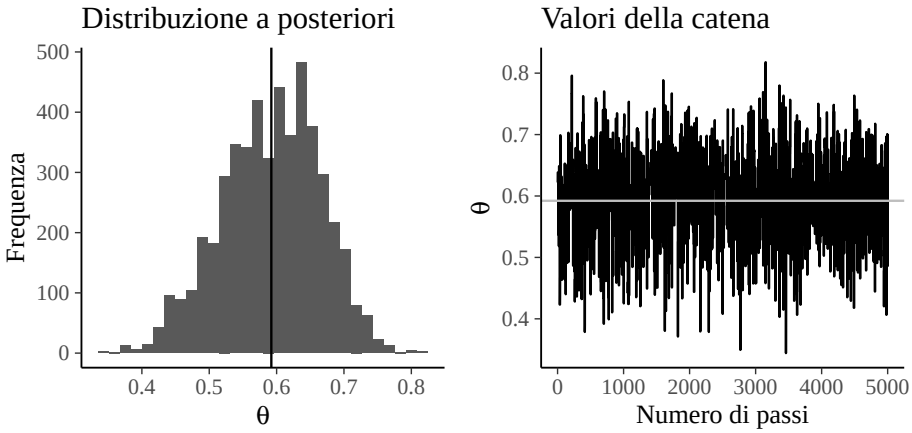


Figura 1.10: *Sinistra. Stima della distribuzione a posteriori della probabilità di una aspettativa futura distorta negativamente per i dati di @zetschefuture2019. Destra. Trace plot dei valori della catena di Markov escludendo il periodo di burn-in.*

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.

1.6 Stazionarietà

L'ultimo punto 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".

1.6.1 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 i-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 $\theta^{(i)}$ e $\theta^{(i-1)}$); l'autocorrelazione di ordine 2 (*lag 2*) misura la correlazione tra valori della catena di Markow separati da due "passi" (cioè, la correlazione tra $\theta^{(i)}$ e $\theta^{(i-2)}$); e così via.

L'autocorrelazione di ordine k è data da ρ_k e può essere stimata come:

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

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

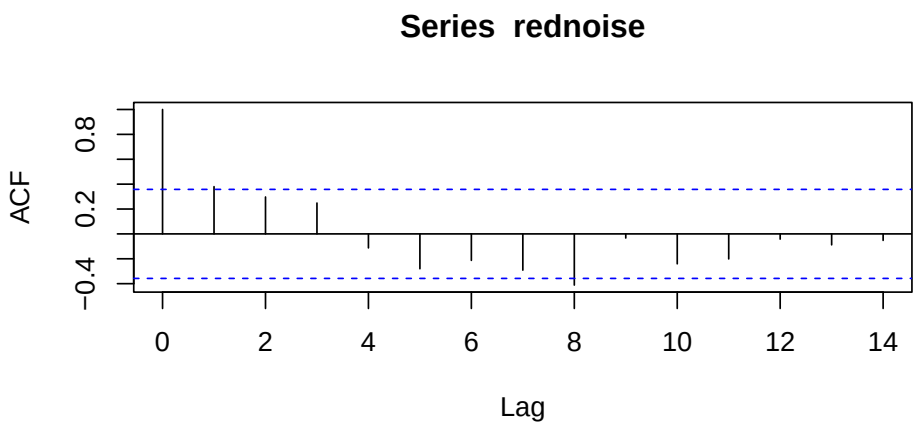
```
suppressPackageStartupMessages(library("colorednoise"))
set.seed(34783859)
rednoise <- colored_noise(
  timesteps = 30, mean = 0.5, sd = 0.05, phi = 0.3
)
```

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 ρ_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*.

1.6.2 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 1.10 (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.

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.