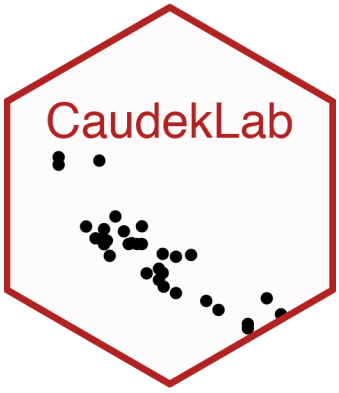
Psicometria

Corrado Caudek

2021-12-24

Questo documento è stato realizzato con:

- LaTeX e la classe memoir (http://www.ctan.org/pkg/memoir);
- R (http://www.r-project.org/) e RStudio (http://www.rstudio.com/);
- bookdown (http://bookdown.org/) e memoiR (https://ericmarcon.github.io/memoiR/).



Nel blog della mia pagina personale sono forniti alcuni approfondimenti degli argomenti qui trattati. https://ccaudek.github.io/caudeklab/

Indice

Data della versione presente: Dicembre 24, 2021.

Prefazione

Data Science per psicologi contiene il materiale delle lezioni dell'insegnamento di *Psicometria B000286* (A.A. 2021/2022) rivolto agli studenti del primo anno del Corso di Laurea in Scienze e Tecniche Psicologiche dell'Università degli Studi di Firenze. *Psicometria* si propone di fornire agli studenti un'introduzione all'analisi dei dati in psicologia. Le conoscenze/competenze che verranno sviluppate in questo insegnamento sono quelle della Data science, ovvero un insieme di conoscenze/competenze che si pongono all'intersezione tra statistica (ovvero, richiedono la capacità di comprendere teoremi statistici) e informatica (ovvero, richiedono la capacità di sapere utilizzare un software).

La psicologia e la Data science

> *It's worth noting, before getting started, that this material is hard. If you find yourself confused at any point, you are normal. Any sense of confusion you feel is just your brain correctly calibrating to the subject matter. Over time, confusion is replaced by comprehension […]*
>
> — Richard McElreath

Sembra sensato spendere due parole su un tema che è importante per gli studenti: quello indicato dal titolo di questo Capitolo. È ovvio che agli studenti di psicologia la statistica non piace. Se piacesse, forse studierebbero Data science e non psicologia; ma non lo fanno. Di conseguenza, gli studenti di psicologia si chiedono: "perché dobbiamo perdere tanto tempo a studiare queste cose quando in realtà quello che ci interessa è tutt'altro?" Questa è una bella domanda.

C'è una ragione molto semplice che dovrebbe farci capire perché la Data science è così importante per la psicologia. Infatti, a ben pensarci, la psicologia è una disciplina intrinsecamente statistica, se per statistica intendiamo quella disciplina che studia la variazione delle caratteristiche degli individui nella popolazione. La psicologia studia *gli individui* ed è proprio la variabilità inter- e intra-individuale ciò che vogliamo descrivere e, in certi casi, predire. In questo senso, la psicologia è molto diversa dall'ingegneria, per esempio. Le proprietà di un determinato ponte sotto certe condizioni, ad esempio, sono molto simili a quelle di un altro ponte, sotto le medesime condizioni. Quindi, per un ingegnere la statistica è poco importante: le proprietà dei materiali sono unicamente dipendenti dalla loro composizione e restano costanti. Ma lo stesso non può dirsi degli individui: ogni individuo è unico e cambia nel tempo. E le variazioni tra gli individui, e di un individuo nel tempo, sono l'oggetto di studio proprio della psicologia: è dunque chiaro che i problemi che la psicologia si pone sono molto diversi da quelli affrontati, per esempio, dagli ingegneri. Questa è la ragione per cui abbiamo tanto bisogno della Data science in psicologia: perché la Data science ci consente di descrivere la variazione e il cambiamento. E queste sono appunto le caratteristiche di base dei fenomeni psicologici.

Sono sicuro che, leggendo queste righe, a molti studenti sarà venuta in mente la seguente domanda: perché non chiediamo a qualche esperto di fare il "lavoro sporco" (ovvero le analisi statistiche) per noi, mentre noi (gli psicologi) ci occupiamo solo di ciò che ci interessa, ovvero dei problemi psicologici slegati dai dettagli "tecnici" della Data

science? La risposta a questa domanda è che non è possibile progettare uno studio psicologico sensato senza avere almeno una comprensione rudimentale della Data science. Le tematiche della Data science non possono essere ignorate né dai ricercatori in psicologia né da coloro che svolgono la professione di psicologo al di fuori dell'Università. Infatti, anche i professionisti al di fuori dall'università non possono fare a meno di leggere la letteratura psicologica più recente: il continuo aggiornamento delle conoscenze è infatti richiesto dalla deontologia della professione. Ma per potere fare questo è necessario conoscere un bel po' di Data science! Basta aprire a caso una rivista specialistica di psicologia per rendersi conto di quanto ciò sia vero: gli articoli che riportano i risultati delle ricerche psicologiche sono zeppi di analisi statistiche e di modelli formali. E la comprensione della letteratura psicologica rappresenta un requisito minimo nel bagaglio professionale dello psicologo.

Le considerazioni precedenti cercano di chiarire il seguente punto: la Data science non è qualcosa da studiare a malincuore, in un singolo insegnamento universitario, per poi poterla tranquillamente dimenticare. Nel bene e nel male, gli psicologi usano gli strumenti della Data science in tantissimi ambiti della loro attività professionale: in particolare quando costruiscono, somministrano e interpretano i test psicometrici. È dunque chiaro che possedere delle solide basi di Data science è un tassello imprescindibile del bagaglio professionale dello psicologo. In questo insegnamento verrano trattati i temi base della Data science e verrà adottato un punto di vista bayesiano, che corrisponde all'approccio più recente e sempre più diffuso in psicologia.

Come studiare

> *I know quite certainly that I myself have no special talent. Curiosity, obsession and dogged endurance, combined with self-criticism, have brought me to my ideas.*
>
> — Albert Einstein

Il giusto metodo di studio per prepararsi all'esame di Psicometria è quello di seguire attivamente le lezioni, assimilare i concetti via via che essi vengono presentati e verificare in autonomia le procedure presentate a lezione. Incoraggio gli studenti a farmi domande per chiarire ciò che non è stato capito appieno. Incoraggio gli studenti a utilizzare i forum attivi su Moodle e, soprattutto, a svolgere gli esercizi proposti su Moodle. I problemi forniti su Moodle rappresentano il livello di difficoltà richiesto per superare l'esame e consentono allo studente di comprendere se le competenze sviluppate fino a quel punto sono sufficienti rispetto alle richieste dell'esame.

La prima fase dello studio, che è sicuramente individuale, è quella in cui è necessario acquisire le conoscenze teoriche relative ai problemi che saranno presentati all'esame. La seconda fase di studio, che può essere facilitata da scambi con altri e da incontri di gruppo, porta ad acquisire la capacità di applicare le conoscenze: è necessario capire come usare un software (R) per applicare i concetti statistici alla specifica situazione del problema che si vuole risolvere. Le due fasi non sono però separate: il saper fare molto spesso ci aiuta a capire meglio.

Sviluppare un metodo di studio efficace

> *Memorization is not learning.*
>
> — Richard Phillips Feynman

Avendo insegnato molte volte in passato un corso introduttivo di analisi dei dati ho notato nel corso degli anni che gli studenti con l'atteggiamento mentale che descriverò qui sotto generalmente ottengono ottimi risultati. Alcuni studenti sviluppano naturalmente questo approccio allo studio, ma altri hanno bisogno di fare uno sforzo per maturarlo.

Fornisco qui sotto una breve descrizione del "metodo di studio" che, nella mia esperienza, è il più efficace per affrontare le richieste di questo insegnamento (Burger & Starbird, 2012).

- Dedicate un tempo sufficiente al materiale di base, apparentemente facile; assicuratevi di averlo capito bene. Cercate le lacune nella vostra comprensione. Leggere presentazioni diverse dello stesso materiale (in libri o articoli diversi) può fornire nuove intuizioni.

- Gli errori che facciamo sono i nostri migliori maestri. Istintivamente cerchiamo di dimenticare subito i nostri errori. Ma il miglior modo di imparare è apprendere dagli errori che commettiamo. In questo senso, una soluzione corretta è meno utile di una soluzione sbagliata. Quando commettiamo un errore questo ci fornisce un'informazione importante: ci fa capire qual è il materiale di studio sul quale dobbiamo ritornare e che dobbiamo capire meglio.

- C'è ovviamente un aspetto "psicologico" nello studio. Quando un esercizio o problema ci sembra incomprensibile, la cosa migliore da fare è dire: "mi arrendo", "non ho idea di cosa fare!". Questo ci rilassa: ci siamo già arresi, quindi non abbiamo niente da perdere, non dobbiamo più preoccuparci. Ma non dobbiamo fermarci qui. Le cose "migliori" che faccio (se ci sono) le faccio quando non ho voglia di lavorare. Alle volte, quando c'è qualcosa che non so fare e non ho idea di come affontare, mi dico: "oggi non ho proprio voglia di fare fatica", non ho voglia di mettermi nello stato mentale per cui "in 10 minuti devo risolvere il problema perché dopo devo fare altre cose". Però ho voglia di *divertirmi* con quel problema e allora mi dedico a qualche aspetto "marginale" del problema, che so come affrontare, oppure considero l'aspetto più difficile del problema, quello che non so come risolvere, ma invece di cercare di risolverlo, guardo come altre persone hanno affrontato problemi simili, opppure lo stesso problema in un altro contesto. Non mi pongo l'obiettivo "risolvi il problema in 10 minuti", ma invece quello di farmi un'idea "generale" del problema, o quello di capire un caso più specifico e più semplice del problema. Senza nessuna pressione. Infatti, in quel momento ho deciso di non lavorare (ovvero, di non fare fatica). Va benissimo se "parto per la tangente", ovvero se mi metto a leggere del materiale che sembra avere poco a che fare con il problema centrale (le nostre intuizioni e la nostra curiosità solitamente ci indirizzano sulla strada giusta). Quando faccio così, molto spesso trovo la soluzione del problema che mi ero posto e, paradossalmente, la trovo in un tempo minore di quello che, in precedenza, avevo dedicato a "lavorare" al problema. Allora perché non faccio sempre così? C'è ovviamente l'aspetto dei "10 minuti" che non è sempre facile da dimenticare. Sotto pressione, possiamo solo agire in maniera automatica, ovvero possiamo solo applicare qualcosa che già sappiamo fare. Ma se dobbiamo imparare qualcosa di nuovo, la pressione è un impedimento.

- È utile farsi da soli delle domande sugli argomenti trattati, senza limitarsi a cercare di risolvere gli esercizi che vengono assegnati. Quando studio qualcosa mi viene in mente: "se questo è vero, allora deve succedere quest'altra cosa". Allora verifico se questo è vero, di solito con una simulazione. Se i risultati della simulazione sono quelli che mi aspetto, allora vuol dire che ho capito. Se i risultati sono diversi da quelli che mi aspettavo, allora mi rendo conto di non avere capito e ritorno indietro a studiare con più attenzione la teoria che pensavo di avere capito – e ovviamente mi rendo conto che c'era un aspetto che avevo frainteso. Questo tipo di verifica è qualcosa che dobbiamo fare da soli, in prima persona: nessun altro può fare questo al posto nostro.

- Non aspettatevi di capire tutto la prima volta che incontrate un argomento nuovo.[1] È utile farsi una nota mentalmente delle lacune nella vostra comprensione e tornare su di esse in seguito per carcare di colmarle. L'atteggiamento naturale, quando non capiamo i dettagli di qualcosa, è quello di pensare: "non importa, ho capito in maniera approssimativa questo punto, non devo preoccuparmi del resto". Ma in realtà non è vero: se la nostra comprensione è superficiale, quando il problema verrà presentato in una nuova forma, non riusciremo a risolverlo. Per cui i dubbi che ci vengono quando studiamo qualcosa sono il nostro alleato più prezioso: ci dicono esattamente quali sono gli aspetti che dobbiamo approfondire per potere migliorare la nostra preparazione.
- È utile sviluppare una visione d'insieme degli argomenti trattati, capire l'obiettivo generale che si vuole raggiungere e avere chiaro il contributo che i vari pezzi di informazione forniscono al raggiungimento di tale obiettivo. Questa organizzazione mentale del materiale di studio facilita la comprensione. È estremamente utile creare degli schemi di ciò che si sta studiando. Non aspettate che sia io a fornirvi un riepilogo di ciò che dovete imparare: sviluppate da soli tali schemi e tali riassunti.
- Tutti noi dobbiamo imparare l'arte di trovare le informazioni, non solo nel caso di questo insegnamento. Quando vi trovate di fronte a qualcosa che non capite, o ottenete un oscuro messaggio di errore da un software, ricordatevi: "Google is your friend".

Corrado Caudek

[1] Ricordatevi inoltre che gli individui tendono a sottostimare la propria capacità di apprendere (Horn & Loewenstein, 2021).

Capitolo 1

Approssimazione della distribuzione a posteriori

In questo Capitolo ci occuperemo di metodi numerici per l'approssimazione della distribuzione a posteriori.

1.1 Stima 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. Questo però non è sempre giustificato dal punto di vista teorico. 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 analiticamente la distribuzione a posteriori solo per alcune specifiche combinazioni di distribuzioni a priori e verosimiglianza, il che limita considerevolmente la flessibilità della modellizzazione. 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).

1.2 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 calcolate in ciascuna cella della griglia.

Il metodo basato su griglia si sviluppa in quattro fasi:

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

Modello Beta-Binomiale

Per fare un esempio, consideriamo il modello Beta-Binomiale di cui conosciamo la soluzione esatta. 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:

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

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

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

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 $y = 9$.

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

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

Calcoliamo poi, in ciascuna cella della griglia, il prodotto della verosimiglianza e della distribuzione a priori. Troviamo così un'approssimazione discreta e non normalizzata della distribuzione a posteriori (`unnormalized`). Normalizziamo infine 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 sarà uguale a 1:

```
grid_data %>%
  summarize(
    sum(unnormalized),
    sum(posterior)
  )
#> # A tibble: 1 × 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 × 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)
  )
```

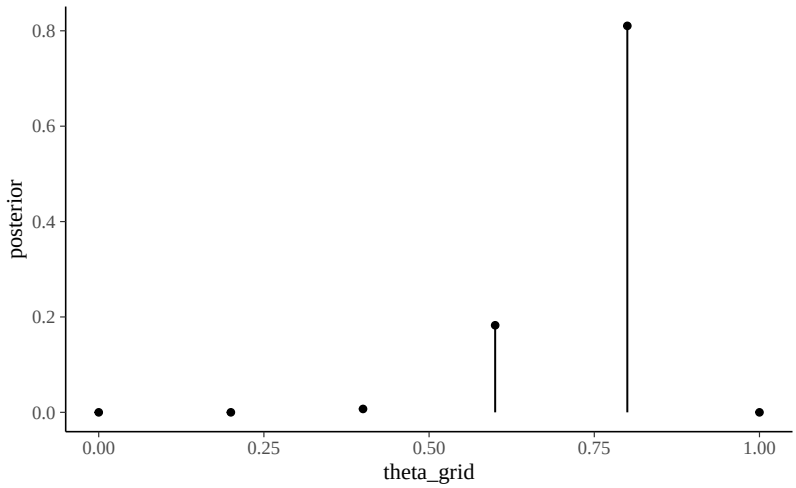


Figura 1.1: *Distribuzione a posteriori discretizzata ottenuta con il metodo grid-based per $y = 9$ successi in 10 prove Bernoulliane, con distribuzione a priori Beta$(2, 2)$. È stata utilizzata una griglia di solo $n = 6$ punti.*

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

```
set.seed(84735)
post_sample <- sample_n(
  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 %>%
  mutate(
    prior = dbeta(theta_grid, 2, 2),
```

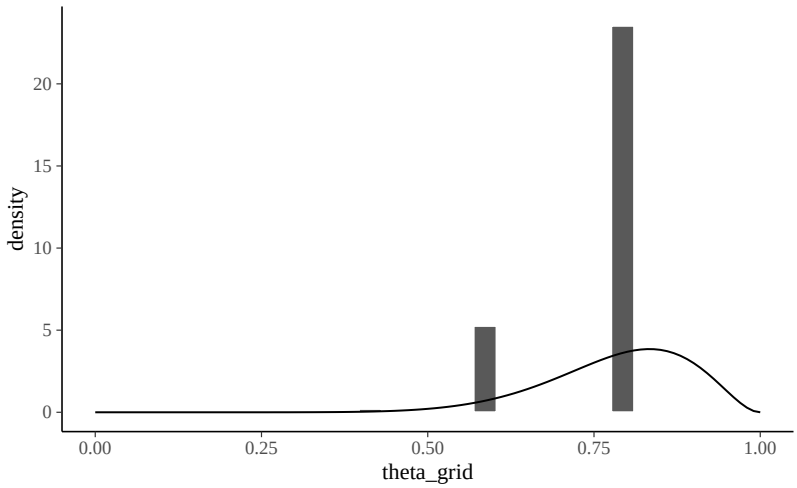


Figura 1.2: *Campionamento dalla distribuzione a posteriori discretizzata ottenuta con il metodo grid-based per $y = 9$ successi in 10 prove Bernoulliane, con distribuzione a priori Beta$(2, 2)$. È stata utilizzata una griglia di solo $n = 6$ punti.*

```
    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:

```
# 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 $p(\theta \mid y) = \text{Beta}(11, 3)$.

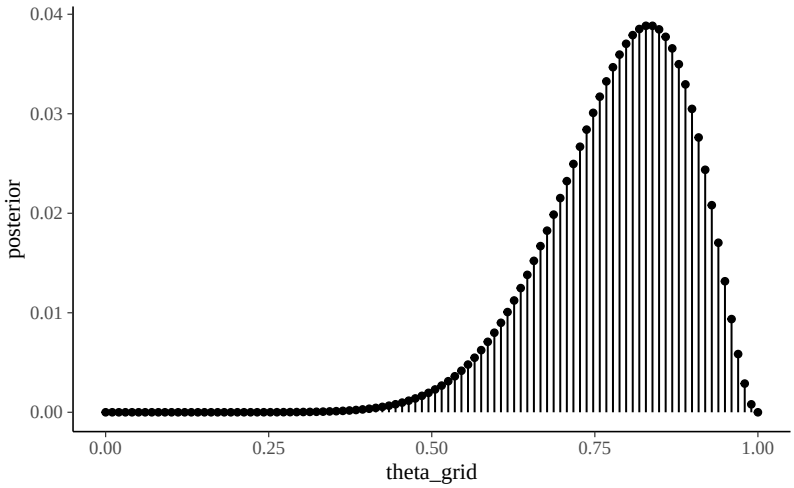


Figura 1.3: *Distribuzione a posteriori discretizzata ottenuta con il metodo grid-based per $y = 9$ successi in 10 prove Bernoulliane, con distribuzione a priori $Beta(2, 2)$. È stata utilizzata una griglia di $n = 100$ punti.*

```
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))
```

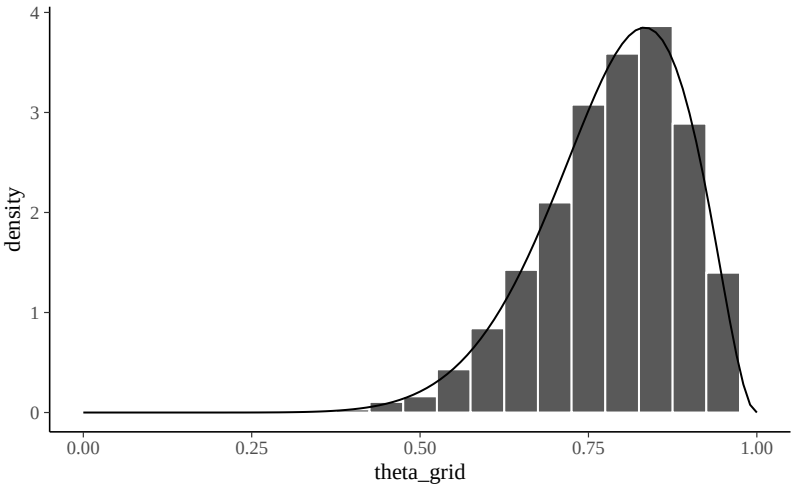


Figura 1.4: *Campionamento dalla distribuzione a posteriori discretizzata ottenuta con il metodo grid-based per $y = 9$ successi in 10 prove Bernoulliane, con distribuzione a priori $Beta(2, 2)$. È stata utilizzata una griglia di $n = 100$ punti. All'istogramma è stata sovrapposta la corretta distribuzione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

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

1.3 Approssimazione quadratica

L'approssimazione quadratica è un altro metodo che può essere usato 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.

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  0.6 0.0775 0.448 0.752
```

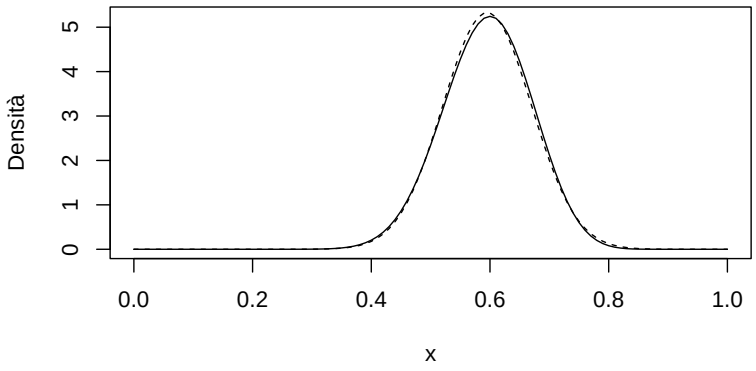
[3] Che cos'è la *maledizione della dimensionalità*? È molto facile da capire. Supponiamo di utilizzare una griglia di 100 punti equispaziati. Nel caso di un solo parametro, sarà necessario calcolare 100 valori. Per due parametri devono essere calcolari 100^2 valori. Ma già per 10 parametri avremo bisogno 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 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.

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,
  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.

1.4 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.

Consideriamo nuovamente i dati di Zetsche et al. (2019) (23 "successi" in 30 prove Bernoulliane) e applichiamo a quei dati lo stesso modello del Capitolo **??**:

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

Poniamoci il problema di stimare il valore della media a posteriori di θ. Nel caso presente, il risultato esatto è

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

Dato che la distribuzione a posteriori di θ è Beta$(25, 17)$, possiamo estrarre un campione casuale di osservazioni da tale distribuzione e calcolare la media:

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

È ovvio che l'approssimazione migliora all'aumentare del numero di osservazioni estratte dalla distribuzione a posteriori (legge dei grandi numeri):

```
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
```

Quando il numero di osservazioni (possiamo anche chiamarle "campioni") tratte dalla distribuzione a posteriori è molto grande, la distribuzione di tali 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).

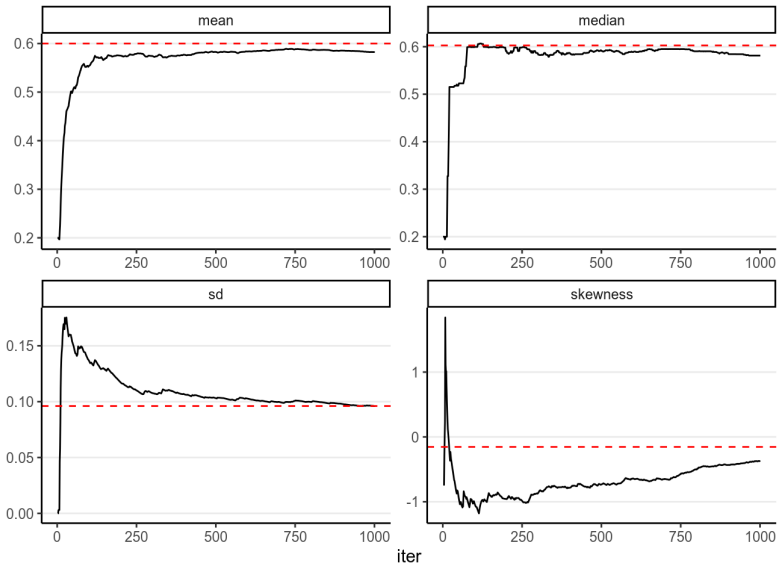


Figura 1.5: *Convergenza delle simulazioni Monte Carlo.*

1.5 Metodi MC basati su Catena di Markov

Nel Paragrafo 1.4 la simulazione Monte Carlo funzionava perché

[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.

- sapevamo che la distribuzione a posteriori era una Beta$(25, 17)$,
- 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 le due condizioni descritte sopra non si applicano. Ad esempio, nel caso di una verosimiglianza binomiale e una distribuzione a priori Normale, la distribuzione a posteriori di θ è

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

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.

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.

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,
```

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

```
    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.

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))
```

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.

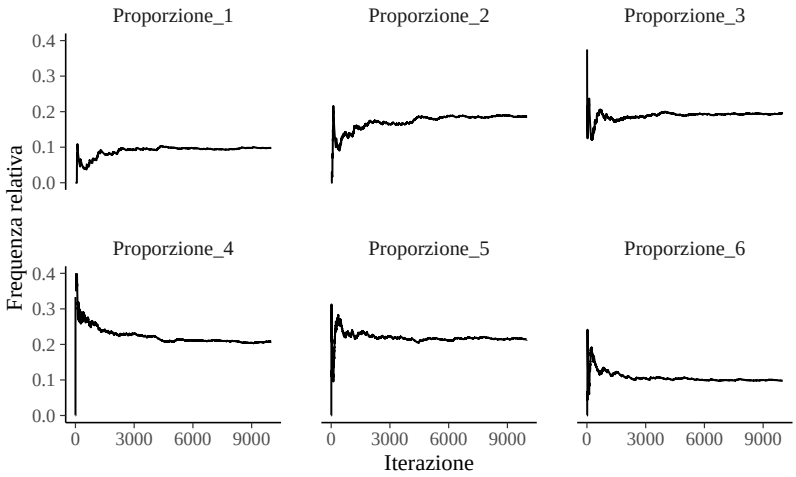


Figura 1.6: *Frequenze relative degli stati da 1 a 6 in funzione del numero di iterazioni per la simulazione di una catena di Markov.*

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 %>%
  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"
  )
```

L'algoritmo di Metropolis corrisponde alla seguente passeggiata casuale.

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

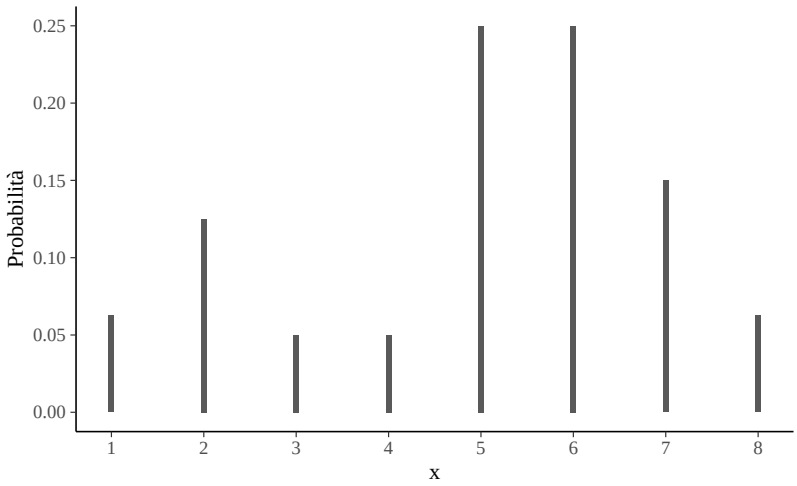


Figura 1.7: *Distribuzione di massa di probabilità per una variabile casuale avente valori 1, 2, ..., 8.*

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 $1, \dots, 8$; 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 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"
  )
```

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.

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

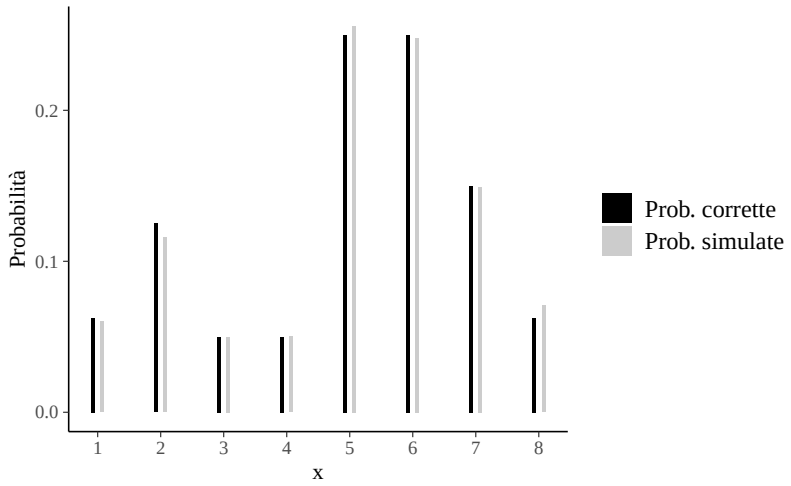


Figura 1.8: *L'istogramma confronta i valori prodotti dall'algoritmo di Metropolis con i corretti valori della distribuzione di massa di probabilità.*

uguale alla distribuzione a posteriori di interesse.[9] 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 $\theta^{(m+1)} = \theta'$ 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 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, però, 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 qui discusso, useremo la distribuzione gaussiana. 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 σ è piccola, il valore candidato θ' sarà simile al valore corrente $\theta^{(m)}$.

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

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

il prodotto tra la verosimiglianza $\mathcal{L}(y \mid \theta^{(m)})$ nel punto $\theta^{(m)}$ e la distribuzione a priori nel punto $\theta^{(m)}$]:

$$\alpha = \frac{p(y \mid \theta')p(\theta')}{p(y \mid \theta^{(m)})p(\theta^{(m)})}. \tag{1.2}$$

Si noti che, essendo un rapporto, la (1.2) cancella la costante di normalizzazione.

(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 risposta 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 $\text{Unif}(0, 1)$. Se $\alpha > u \sim \text{Unif}(0, 1)$ 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 $\theta^{(1)}, \theta^{(2)}, \dots, \theta^{(M)}$. 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 $\left(\theta^{(k+1)}, \theta^{(k+2)}, \dots, \theta^{(M)}\right)$ diventano campioni di $p(\theta \mid y)$.

Una applicazione concreta

Per fare un esempio concreto, consideriamo nuovamente i 30 pazienti esaminati da Zetsche et al. (2019). 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, imponendo su θ la stessa distribuzione a priori usata nel Capitolo **??**, ovvero $\text{Beta}(2, 10)$.

Per calcolare la funzione di verosimiglianza, avendo fissato i dati di Zetsche et al. (2019), definiamo la funzione `likelihood()`

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

che ritorna l'ordinata della verosimiglianza binomiale per ciascun valore del vettore `param` in input.

La distribuzione a priori Beta$(2, 10)$ è implementata nella funzione `prior()`:

```
prior <- function(param, alpha = 2, beta = 10) {
  dbeta(param, alpha, beta)
}
```

Il prodotto della densità a priori e della verosimiglianza è implementato nella funzione `posterior()`:

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

L'Appendice **??** mostra come un'approssimazione della distribuzione a posteriori $p(\theta \mid y)$ per questi dati possa essere ottenuta mediante il metodo basato su griglia. Qui vogliamo invece usare l'algoritmo di Metropolis.

Implementazione

Per implementare l'algoritmo di Metropolis utilizzeremo una distribuzione proposta gaussiana. Il valore candidato sarà dunque un valore selezionato a caso da una gaussiana di parametri μ uguale al valore corrente nella catena e $\sigma = 0.9$. In questo esempio, la deviazione standard 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.[10]

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

[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.

[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.

```
      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 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)]))
acceptance
#> [1] 0.251
```

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 θ. Per esempio, la stima della media a posteriori è:

```
mean(chain[-(1:burnIn)])
#> [1] 0.592
```

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
```

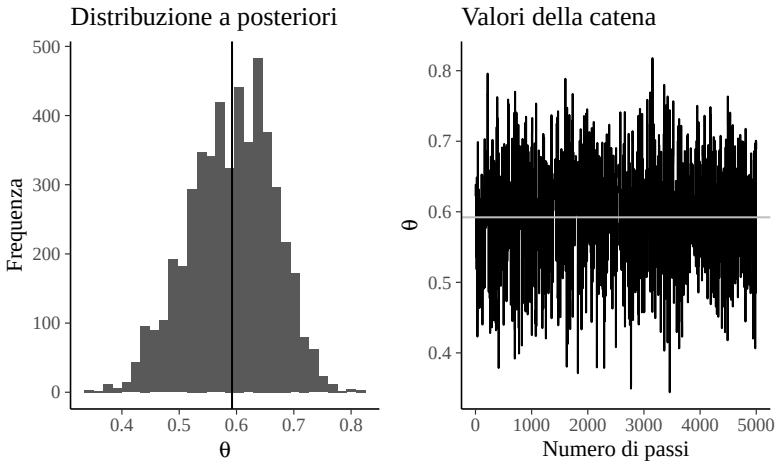


Figura 1.9: *Sinistra. Stima della distribuzione a posteriori della probabilità di una aspettativa futura distorta negativamente per i dati di Zetsche et al. (2019). Destra. Trace plot dei valori della catena di Markov escludendo il periodo di burn-in.*

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.

1.6 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".

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 ρ_k e può essere stimata come:

$$\begin{aligned} \rho_k &= \frac{\text{Cov}(\theta_m, \theta_{m+k})}{\text{Var}(\theta_m)} \\ &= \frac{\sum_{m=1}^{n-k}(\theta_m - \bar{\theta})(\theta_{m-k} - \bar{\theta})}{\sum_{m=1}^{n-k}(\theta_m - \bar{\theta})^2} \qquad \text{con} \qquad \bar{\theta} = \frac{1}{n}\sum_{m=1}^{n}\theta_m. \end{aligned} \tag{1.3}$$

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

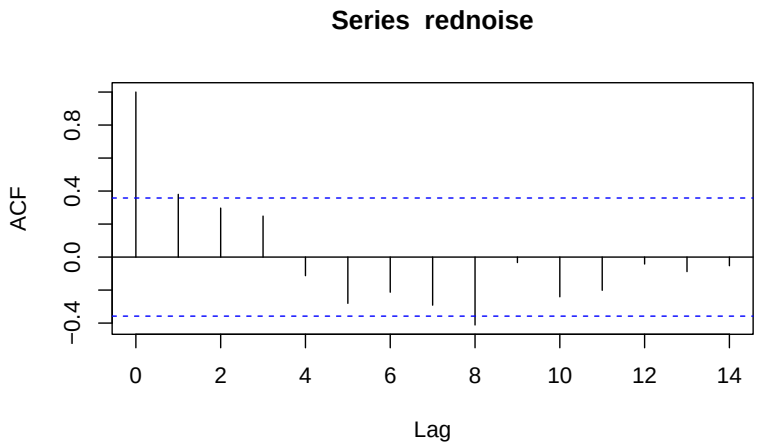
```
suppressPackageStartupMessages(library("colorednoise"))
set.seed(34783859)
rednoise <- colored_noise(
  timesteps = 30, mean = 0.5, sd = 0.05, phi = 0.3
)
```

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] 0.397
```

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*.

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.

Bibliografia

Albert, J. & Hu, J. (2019). *Probability and Bayesian Modeling*. Chapman; Hall/CRC. (Cit. a p. 13).

Burger, E. B. & Starbird, M. (2012). *The 5 elements of effective thinking*. Princeton University Press. (Cit. a p. ix).

Horn, S. & Loewenstein, G. (2021). Underestimating Learning by Doing. *Available at SSRN 3941441* (cit. a p. x).

Johnson, A. A., Ott, M. & Dogucu, M. (2022). *Bayes Rules! An Introduction to Bayesian Modeling with R*. CRC Press. (Cit. a p. 2).

Kruschke, J. (2014). *Doing Bayesian data analysis: A tutorial with R, JAGS, and Stan*. Academic Press. (Cit. a p. 13).

McElreath, R. (2020). *Statistical rethinking: A Bayesian course with examples in R and Stan* (2nd Edition). CRC Press. (Cit. alle pp. 7, 13).

Zetsche, U., Bürkner, P.-C. & Renneberg, B. (2019). Future expectations in clinical depression: Biased or realistic? *Journal of Abnormal Psychology*, *128*(7), 678–688 (cit. alle pp. 8, 17).

Elenco delle figure

Abstract This document contains the material of the lessons of Psicometria B000286 (2021/2022) aimed at students of the first year of the Degree Course in Psychological Sciences and Techniques of the University of Florence, Italy.

Keywords Data science, Bayesian statistics.