Data Science per psicologi

Corrado Caudek

2021-10-10

# Indice

*Appendice* $A$

# Aspettative degli individui depressi

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

Sulla base dei dati forniti dagli autori, abbiamo calcolato la media dei giudizi relativi al livello di tristezza raccolti da ciascun partecipante tramite la app. Tale media è stata poi sottratta dall'aspettativa del livello di tristezza fornita all'inizio della settimana. La discrepanza tra aspettative e realtà è stata considerata come un evento dicotomico: valori positivi di tale differenza indicano che le aspettative circa il livello di tristezza erano maggiori del livello di tristezza effettivamente esperito — ciò significa che le aspettative future risultano negativamente distorte (evento codificato con "1"). Viceversa, si ha che le aspettative risultano positivamente distorte se la differenza descritta in precedenza assume un valore negativo (evento codificato con "0").

Nel campione dei 30 partecipanti clinici di Zetsche et al. (2019), le aspettative future di 23 partecipanti risultano distorte negativamente e quelle di 7 partecipanti risultano distorte positivamente. Chiameremo $\theta$ la probabilità dell'evento "le aspettative del partecipante sono distorte negativamente". Ci poniamo il problema di ottenere una stima a posteriori di $\theta$ usando il metodo basato su griglia.

## A.1 La griglia

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

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

## A.2 Distribuzione a priori

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

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

Verifichiamo:

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

La distribuzione a priori così costruita è rappresentata nella figura A.1.

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

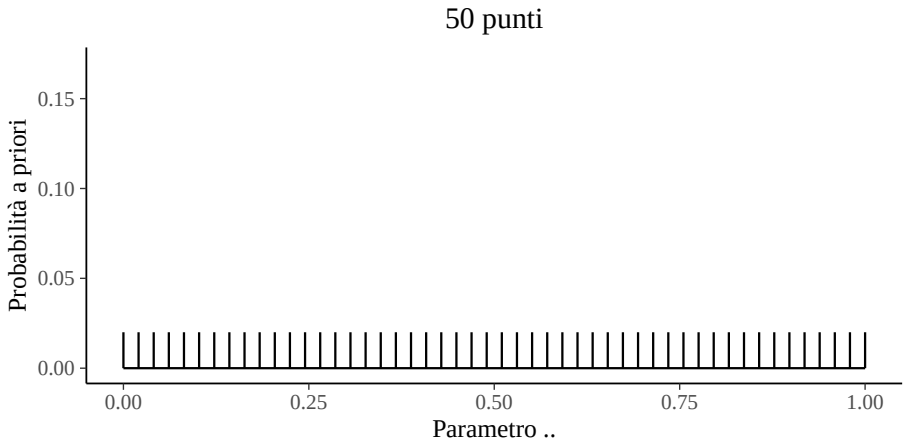



***Figura A.1:*** *Rappresentazione grafica della distribuzione a priori per il parametro heta, ovvero la probabilità di aspettative future distorte negativamente.*

## A.3 Funzione di verosimiglianza

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

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

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

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

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

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

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

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

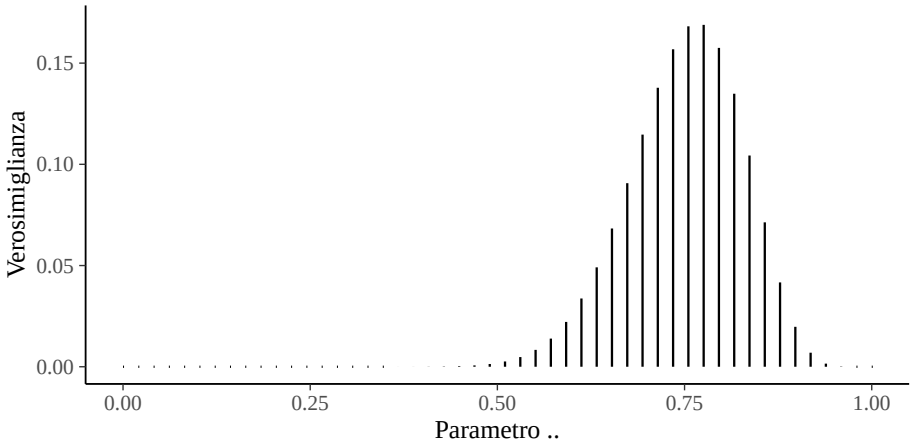



***Figura A.2:*** *Rappresentazione della funzione di verosimiglianza per il parametro $\theta$, ovvero la probabilità di aspettative future distorte negativamente.*

## A.4 Distribuzione a posteriori

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

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

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

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

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

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

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

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

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

Verifichiamo:

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

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

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

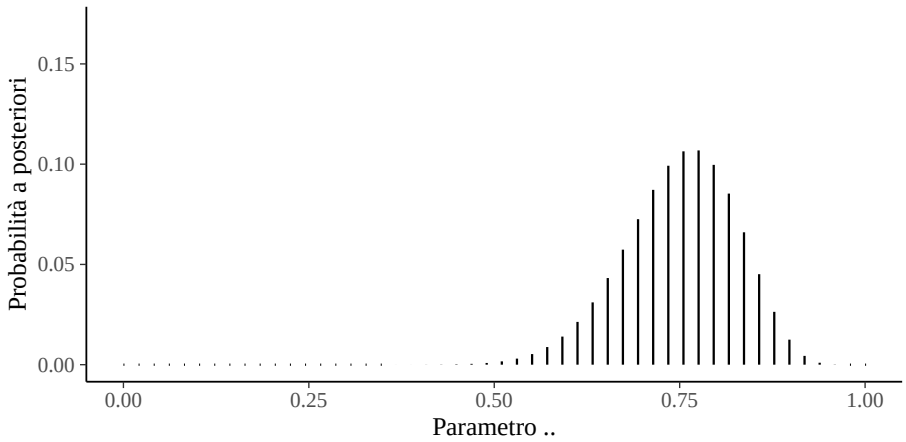



***Figura A.3:*** *Rappresentazione della distribuzione a posteriori per il parametro $\theta$, ovvero la probabilità di aspettative future distorte negativamente.*

I grafici delle figure A.1, A.2 e A.3 sono state calcolati utilizzando una griglia di 50 valori equi-spaziati per il parametro $\theta$. I segmenti verticali rappresentano l'intensità della funzione in corrispondenza di ciascuna modalità parametro $\theta$. Nella figura A.1 e nella figura A.3 la somma delle lunghezze dei segmenti verticali è uguale a 1.0; ciò non si verifica, invece, nel caso della figura A.3 (la funzione di verosimiglianza non è mai una funzione di probabilità, né nel caso discreto né in quello continuo).

## A.5 La stima della distribuzione a posteriori (versione 2)

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

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

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

Con calcoli del tutto simili a quelli descritti sopra si giunge alla distribuzione a posteriori rappresentata nella figura A.4. Useremo ora una griglia di 100 valori per il parametro $\theta$:

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

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

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

Tale distribuzione a priori è rappresentata nella figura A.4:

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

Calcoliamo il valore di verosimiglianza per ciascun punto della griglia:

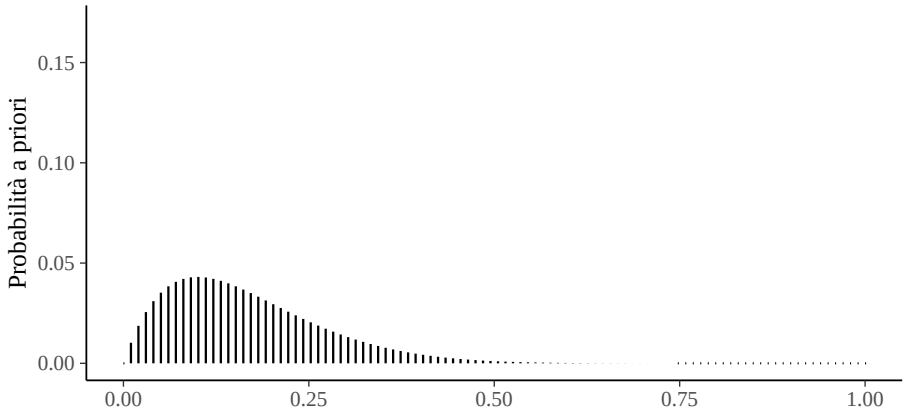



***Figura A.4:*** *Rappresentazione di una funzione a priori informativa per il parametro $\theta$.*

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

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

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

È necessario normalizzare la distribuzione a posteriori discretizzata:

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

Verifichiamo:

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

La nuova funzione a posteriori discretizzata è rappresentata nella figura A.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
```

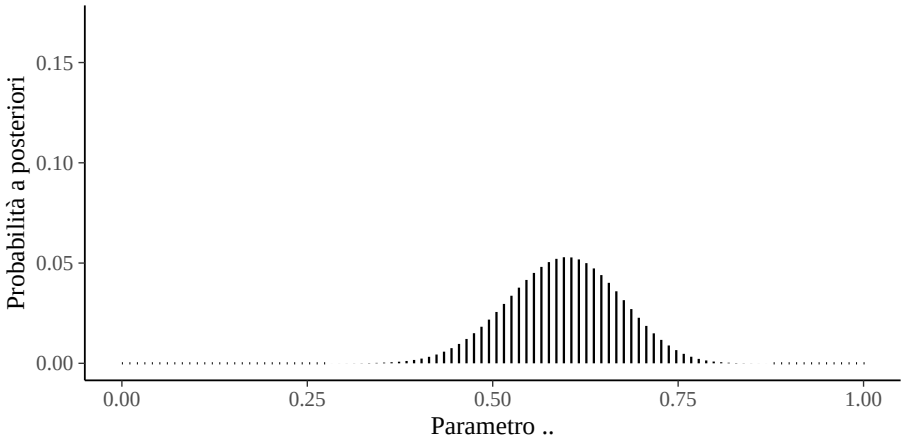



***Figura A.5:*** *Rappresentazione della funzione a posteriori per il parametro $\theta$ calcolata utilizzando una distribuzione a priori informativa.*

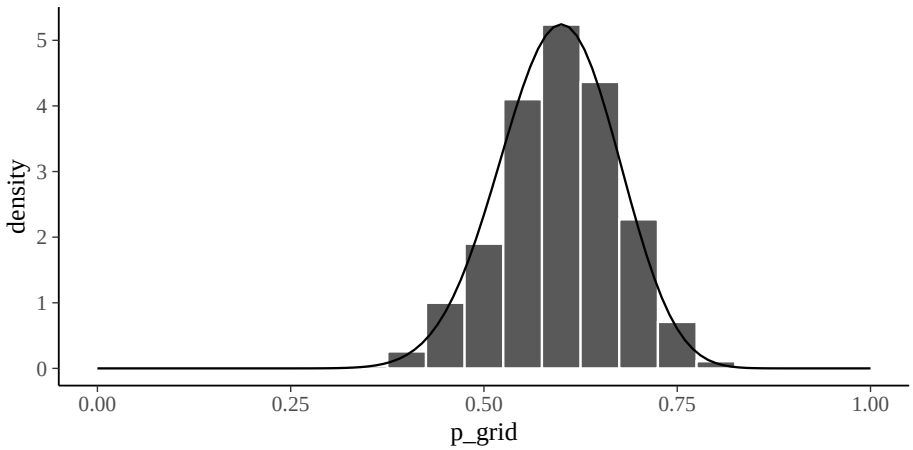
Facendo un confronto tra le figure A.4 e A.5 notiamo una notevole differenza tra la distribuzione a priori e la distribuzione a posteriori. In particolare, la distribuzione a posteriori risulta spostata verso destra su posizioni più vicine a quelle della verosimiglianza [figura A.2]. Si noti inoltre che, a causa dell'effetto della distribuzione a priori, le distribuzioni a posteriori delle figure A.3 e A.5 sono molto diverse tra loro.

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

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

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

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

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




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

La stima della moda a posteriori si ottiene con

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

e corrisponde a

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

La stima della media a posteriori si ottiene con

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

e corrisponde a

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

La stima della mediana a posteriori si ottiene con

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

e corrisponde a

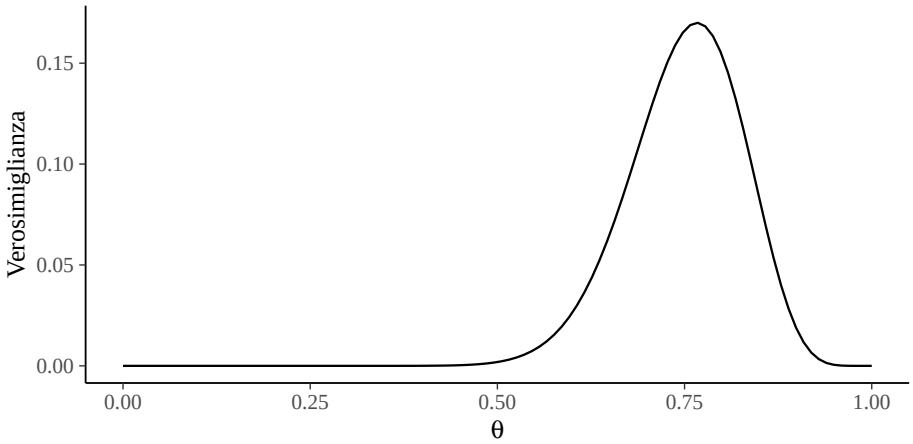
$$\text{Me} = \frac{\alpha - \frac{1}{3}}{\alpha + \beta - \frac{2}{3}} \approx 0.5968.$$

## A.6 Versione 2

Possiamo semplificare i calcoli precedenti definendo le funzioni `likelihood()`, `prior()` e `posterior()`.

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

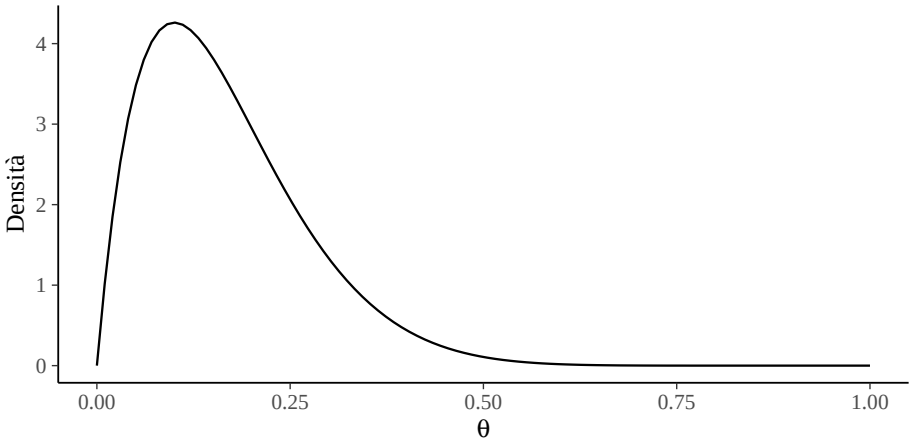
```
x <- 23
N <- 30
param <- seq(0, 1, length.out = 100)

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

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




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

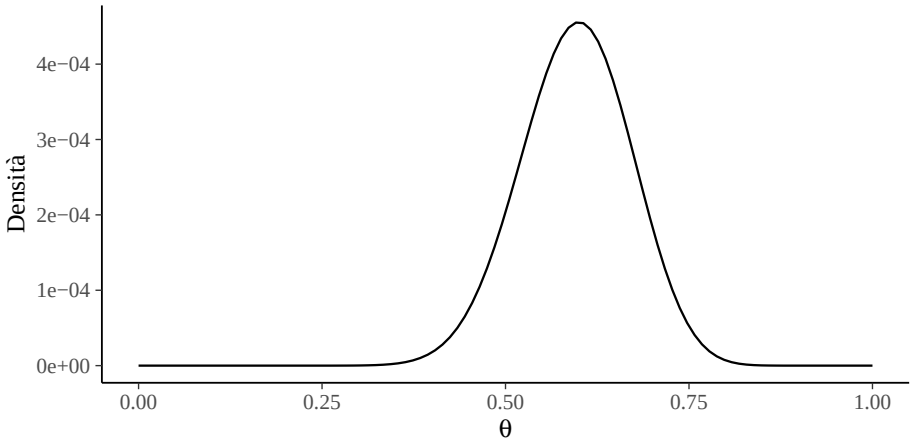
Quale distribuzione a priori utilizzeremo una Beta$(2, 10)$ che è implementata nella funzione `prior()`:

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

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




La funzione `posterior()` ritorna il prodotto della densità a priori e della verosimiglianza:

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

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




La distribuzione a posteriori non normalizzata mostrata nella figura replica il risultato ottenuto con il codice utilizzato nella prima parte di questo Capitolo. Per l'implementazione dell'algoritmo di Metropolis non è necessaria la normalizzazione della distribuzione a posteriori.

*Appendice* $B$

# Integrazione di Monte Carlo

Il termine Monte Carlo si riferisce al fatto che la computazione fa ricorso ad un ripetuto campionamento casuale attraverso la generazione di sequenze di numeri casuali. Una delle sue applicazioni più potenti è il calcolo degli integrali mediante simulazione numerica. Sia l'integrale da calcolare

$$\int_a^b h(y)dy.$$

Se decomponiamo $h(y)$ nel prodotto di una funzione $f(y)$ e una funzione di densità di probabilità $p(y)$ definita nell'intervallo $(a, b)$ avremo:

$$\int_a^b h(y)dy = \int_a^b f(y)p(y)dy = \mathbb{E}[f(y)],$$

così che l'integrale può essere espresso come una funzione di aspettazione $f(y)$ sulla densità $p(y)$. Se definiamo un gran numero di variabili casuali $y_1, y_2, \dots, y_n$ appartenenti alla densità di probabilità $p(y)$ allora avremo

$$\int_a^b h(y)dy = \int_a^b f(y)p(y)dy = \mathbb{E}[f(y)] \approx \frac{1}{n}\sum_{i=1}^{n} f(y)$$

che è l'integrale di Monte Carlo.

L'integrazione con metodo Monte Carlo trova la sua giustificazione nella *Legge forte dei grandi numeri*. Data una successione di variabili casuali $Y_1, Y_2, \dots, Y_n, \dots$ indipendenti e identicamente distribuite con media $\mu$, ne segue che

$$P\left(\lim_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n} Y_i = \mu\right) = 1.$$

Ciò significa che, al crescere di $n$, la media delle realizzazioni di $Y_1, Y_2, \dots, Y_n, \dots$ converge con probabilità 1 al vero valore $\mu$.

Possiamo fornire un esempio intuitivo della legge forte dei grandi numeri facendo riferimento ad una serie di lanci di una moneta dove $Y = 1$ significa "testa" e $Y = 0$ significa "croce". Per la legge forte dei grandi numeri, nel caso di una moneta equilibrata la proporzione di eventi "testa" converge alla vera probabilità dell'evento "testa"

$$\frac{1}{n}\sum_{i=1}^{n} Y_i \to \frac{1}{2}$$

con probabilità di uno.

Quello che è stato detto sopra non è che un modo sofisticato per dire che, se vogliamo calcolare un'approssimazione del valore atteso di una variabile casuale, non dobbiamo fare altro che la media aritmetica di un grande numero di realizzazioni della variabile casuale. Come è facile intuire, l'approssimazione migliora al crescere del numero di dati che abbiamo a disposizione.

L'integrazione di Monte Carlo può essere usata per approssimare la distribuzione a posteriori richiesta da una analisi Bayesiana: una stima di $p(\theta \mid y)$ può essere ottenuta mediante un grande numero di campioni casuali della distribuzionea a posteriori.

*Appendice* $C$

# Programmare in Stan

## C.1 Che cos'è Stan?

STAN è un linguaggio di programmazione probabilistico usato per l'inferenza bayesiana (Carpenter et al., 2017). Prende il nome da uno dei creatori del metodo Monte Carlo, Stanislaw Ulam (Eckhardt, 1987). Stan consente di generare campioni da distribuzioni di probabilità basati sulla costruzione di una catena di Markov avente come distribuzione di equilibrio (o stazionaria) la distribuzione desiderata.

Nelle parole degli autori:

> Stan is a state-of-the-art platform for statistical modeling and high-performance statistical computation. Thousands of users rely on Stan for statistical modeling, data analysis, and prediction in the social, biological, and physical sciences, engineering, and business. Users specify log density functions in Stan's probabilistic programming language and get: full Bayesian statistical inference with MCMC sampling (NUTS, HMC); approximate Bayesian inference with variational inference (ADVI): penalized maximum likelihood estimation with optimization (L-BFGS).

È possibile accedere al linguaggio Stan tramite diverse interfacce:

- `CmdStan`: eseguibile da riga di comando,
- `RStan` - integrazione con il linguaggio R;
- `PyStan` - integrazione con il linguaggio di programmazione Python;
- `MatlabStan` - integrazione con MATLAB;
- `Stan.jl` - integrazione con il linguaggio di programmazione Julia;
- `StataStan` - integrazione con Stata.

Inoltre, vengono fornite interfacce di livello superiore con i pacchetti che utilizzano Stan come backend, principalmente in Linguaggio R:

- `shinystan`: interfaccia grafica interattiva per l'analisi della distribuzione a posteriori e le diagnostiche MCMC;

- `bayesplot`: insieme di funzioni utilizzabili per creare grafici relativi all'analisi della distribuzione a posteriori, ai test del modello e alle diagnostiche MCMC;

- `brms`: fornisce un'ampia gamma di modelli lineari e non lineari specificando i modelli statistici mediante la sintassi usata in R;
- `rstanarm`: fornisce un sostituto per i modelli frequentisti forniti da base R e `lme4` utilizzando la sintassi usata in R per la specificazione dei modelli statistici;
- `edstan`: modelli Stan per la Item Response Theory;
- `cmdstanr`, un'interfaccia R per `CmdStan`.

## C.2 Interfaccia `cmdstanr`

Negli esempi di questa dispensa verrà usata l'interfaccia `cmdstanr`. Il pacchetto `cmdstanr` non è ancora disponibile su CRAN, ma può essere installato come indicato su questo link. Una volta che è stato installato, il pacchetto `cmdstanr` può essere caricato come un qualsiasi altro pacchetto R.

Si noti che `cmdstanr` richiede un'installazione funzionante di `CmdStan`, l'interfaccia shell per Stan. Se `CmdStan` non è installato, `cmdstanr` lo installerà automaticamente se il computer dispone di una *Toolchain* adatta. Stan richiede infatti che sul computer su cui viene installato siano presenti alcuni strumenti necessari per gestire i file C++. Tra le altre ragioni, questo è dovuto al fatto che il codice Stan viene tradotto in codice C++ e compilato. Il modo migliore per ottenere il software necessario per un computer Windows o Mac è quello di installare `RTools`. Per un computer Linux, è necessario installare `build-essential` e una versione recente dei compilatori g++ o clang++. I requisiti sono descritti nella Guida di CmdStan.

Per verificare che la Toolchain sia configurata correttamente è possibile utilizzare la funzione `check_cmdstan_toolchain()`:

```
check_cmdstan_toolchain()
```

Se la toolchain è configurata correttamente, `CmdStan` può essere installato mediante la funzione `install_cmdstan()`:

```
# do not run!
# install_cmdstan(cores = 2)
```

Prima di poter utilizzare `CmdStanR`, è necessario specificare dove si trova l'installazione di `CmdStan`.

```
# do not run!
# set_cmdstan_path(PATH_TO_CMDSTAN)
```

Sul mio computer `PATH_TO_CMDSTAN` è la seguente stringa (incluse le virgolette):

```
cmdstan_path()
#> [1] "/Users/corrado/.cmdstanr/cmdstan-2.28.0"
```

La versione installata di `CmdStan` si ottiene con:

```
cmdstan_version()
#> [1] "2.28.0"
```

## C.3 Codice Stan

Qualunque sia l'interfaccia che viene usata, i modelli sottostanti sono sempre scritti nel linguaggio Stan, il che significa che lo stesso codice Stan è valido per tutte le interfacce possibili. Il codice Stan è costituito da una serie di blocchi che vengono usati per specificare un modello statistico. In ordine, questi blocchi sono: `data`, `transformed data`, `parameters`, `transformed parameters`, `model`, e `generated quantities`.

Un programma Stan contiene tre "blocchi" obbligatori: blocco `data`, blocco `parameters`, blocco `model`.

### C.3.1 Blocco `data`

Qui vengono dichiarate le variabili che saranno passate a Stan. Devono essere elencati i nomi delle variabili che saranno utilizzate nel programma, il *tipo di dati* da registrare per ciascuna variabile, per esempio: - *int* = intero, - *real* = numeri reali (ovvero, numeri con cifre decimali), - *vector* = sequenze ordinate di numeri reali unidimensionali, - *matrix* = matrici bidimensionali di numeri reali, - *array* = sequenze ordinate di dati multidimensionali.

Devono anche essere dichiarate le dimensioni delle variabili e le eventuali restrizioni sulle variabili (es. `upper = 1 lower = 0`, che fungono da controlli per Stan). Tutti i nomi delle variabili assegnate qui saranno anche usati negli altri blocchi del programma.

Per esempio, l'istruzione seguente dichiaria la variabile `Y` – la quale rappresenta, ad esempio, l'altezza di 10 persone – come una variabile di tipo `real[10]`. Ciò significa che specifichiamo un array di lunghezza 10, i cui elementi sono variabili continue definite sull'intervallo dei numeri reali $[-\infty, +\infty]$.

```
data {
  real Y[10]; // heights for 10 people
}
```

Invece, con l'istruzione

```
data {
  int Y[10]; // qi for 10 people
}
```

dichiariamo la variabile `Y` – la quale rappresenta, ad esempio, il QI di 10 persone – come una variabile di tipo `int[10]`, ovvero un array di lunghezza 10, i cui elementi sono numeri naturali, cioè numeri interi non negativi $\{0, +1, +2, +3, +4, ...\}$.

Un altro esempio è

```
data {
  real<lower=0, upper=1> Y[10]; // 10 proportions
}
```

nel quale viene specificato un array di lunghezza 10, i cui elementi sono delle variabili continue definite sull'intervallo dei numeri reali $[0, 1]$ — per esempio, delle proporzioni.

Si noti che i tipi `vector` e `matrix` contengono solo elementi di tipo `real`, ovvero variabili continue, mentre gli `array` possono contenere dati di qualsiasi tipo. I dati passati a Stan devono essere contenuti in un oggetto del tipo `list`.

### C.3.2 Blocco `parameters`

I parametri che vengono stimati sono dichiarati nel blocco `parameters`. Per esempio, l'istruzione

```
parameters {
  real mu; // mean height in population
  real<lower=0> sigma; // sd of height distribution
}
```

dichiaria la variabile `mu` che codifica l'altezza media nella popolazione, che è una variabile continua in un intervallo illimitato di valori, e la deviazione standard `sigma`, che è una variabile continua non negativa. Avremmo anche potuto specificare un limite inferiore di zero su `mu` perché deve essere non negativo.

Per una regressione lineare semplice, ad esempio, devono essere dichiarate le variabili corrispondenti all'intercetta (`alpha`), alla pendenza (`beta`) e alla deviazione standard degli errori attorno alla linea di regressione (`sigma`). In altri termini, nel blocco `parameters` devono essere elencati tutti i parametri che dovranno essere stimati dal modello. Si noti che parametri discreti non sono possibili. Infatti, Stan attualmente non supporta i parametri con valori interi, almeno non direttamente.

### C.3.3 Blocco `model`

Nel blocco `model` vengono elencate le dichiarazioni relative alla verosimiglianza dei dati e alle distribuzioni a priori dei parametri, come ad esempio, nelle istruzioni seguenti.

```
model {
  for(i in 1:10) {
    Y[i] ~ normal(mu, sigma);
  }
  mu ~ normal(170, 15); // prior for mu
  sigma ~ cauchy(0, 20); // prior for sigma
}
```

Mediante l'istruzione all'interno del ciclo `for`, ciascun valore dell'altezza viene concepito come una variable casuale proveniente da una distribuzione Normale di parametri $\mu$ e $\sigma$ (i parametri di interesse nell'inferenza). Il ciclo `for` viene ripetuto 10 volte perché i dati sono costituiti da un array di 10 elementi (ovvero, il campione è costituito da 10 osservazioni).

Le due righe che seguno il ciclo `for` specificano le distribuzioni a priori dei parametri su cui vogliamo effettuare l'inferenza. Per $\mu$ assumiamo una distribuzione a priori Normale di parametri $\mu = 170$ e $\sigma = 15$; per $\sigma$ assumiamo una distribuzione a priori Cauchy(0, 20).

Se non viene definita alcuna distribuzione a priori, Stan utilizzerà la distribuzione a priori predefinita $Unif(-\infty, +\infty)$. Raccomandazioni sulle distribuzioni a priori sono fornite in questo link.

La precedente notazione di campionamento può anche essere espressa usando la seguente notazione alternativa:

```
  for(i in 1:10) {
    target += normal_lpdf(Y[i] | mu, sigma);
  }
```

Questa notazione rende trasparente il fatto che, in pratica, Stan esegue un campionamento nello spazio

$$\log p(\theta \mid y) \propto \log p(y \mid \theta) + \log p(\theta) = \sum_{i=1}^{n} \log p(y_i \mid \theta) + \log p(\theta).$$

Per ogni passo MCMC, viene ottenuto un nuovo valore di $\mu$ e $\sigma$ eviene valutata la log densità a posteriori non normalizzata. Ad ogni passo MCMC, Stan calcola un nuovo valore della densità a posteriori su scala logaritmica partendo da un valore di 0 e incrementandola ogni volta che incontra un'istruzione ~. Quindi, le istruzioni precedenti aumentano la log-densità di una quantità pari a $\log(p(Y[i])) \propto -\frac{1}{2}\log(\sigma^2) - (Y[i] - \mu)^2/2\sigma^2$ per le altezze si ciascuno degli $i = 1 \dots, 10$ individui – laddove la formula esprime, in termini logaritmici, la densità Normale da cui sono stati esclusi i termini costanti.

Oppure, in termini vettorializzati, il modello descritto sopra può essere espresso come

```
model {
  Y ~ normal(mu, sigma);
}
```

dove il termine a sinistra di $\sim$ è un array. Questa notazione più compatta è anche la più efficiente.

### C.3.4 Blocchi opzionali

Ci sono inoltre tre blocchi opzionali:

- Il blocco `transformed data` consente il pre-processing dei dati. È possibile trasformare i parametri del modello; solitamente ciò viene fatto nel caso dei modelli più avanzati per consentire un campionamento MCMC più efficiente.
- Il blocco `transformed parameters` consente la manipolazione dei parametri prima del calcolo della distribuzione a posteriori.
- Il blocco `generated quantities` consente il post-processing riguardante qualsiasi quantità che non fa parte del modello ma può essere calcolata a partire dai parametri del modello, per ogni iterazione dell'algoritmo. Esempi includono la generazione dei campioni a posteriori e le dimensioni degli effetti.

### C.3.5 Sintassi

Si noti che il codice Stan richiede i punti e virgola (;) alla fine di ogni istruzione di assegnazione. Questo accade per le dichiarazioni dei dati, per le dichiarazioni dei parametri e ovunque si acceda ad un elemento di un tipo `data` e lo si assegni a qualcos'altro. I punti e virgola non sono invece richiesti all'inizio di un ciclo o di un'istruzione condizionale, dove non viene assegnato nulla.

In STAN, qualsiasi stringa che segue `//` denota un commento e viene ignorata dal programma.

Stan è un linguaggio estremamente potente e consente di implementare quasi tutti i modelli statistici, ma al prezzo di un certo sforzo di programmazione. Anche l'adattamento di semplici modelli statistici mediante il linguaggio STAN a volte può essere laborioso. Per molti modelli comunemente usati, come i modelli di regressione e multilivello, tale processo può essere semplificato usando le funzioni del pacchetto `brms`. D'altra parte, per modelli veramente complessi, non ci sono molte alternative all'uso di STAN. Per chi è curioso, il manuale del linguaggio Stan è accessibile al seguente link.

## C.4 Workflow

Se usiamo `cmdstanr`, dobbiamo prima scrivere il codice con il modello, poi compilare il codice con la funzione `cmdstan_model()`, poi eseguire il campionamento MCMC con il metodo `$sample()` e, infine, creare un sommario dei risultati usando, per esempio, usando il metodo `$summary()`.

# Bibliografia

Carpenter, B., Gelman, A., Hoffman, M. D., Lee, D., Goodrich, B., Betancourt, M., Brubaker, M., Guo, J., Li, P., and Riddell, A. (2017). Stan: A probabilistic programming language. *Journal of statistical software*, 76(1):1–32.

Eckhardt, R. (1987). Stan Ulam, John Von Neumann and the Monte Carlo Method. *Los Alamos Science Special Issue*.

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.