3 Markov Chain Monte Carlo

Diego

Table of contents

```
pacman::p_load(tidyverse,cowplot,here,tidybayes,brms,patchwork)
```

```
hdi_of_icdf <- function(name, width = .95, tol = 1e-8, ... ) {

  incredible_mass <- 1.0 - width
  interval_width <- function(low_tail_prob, name, width, ...) {
    name(width + low_tail_prob, ...) - name(low_tail_prob, ...)
  }
  opt_info <- optimize(interval_width, c(0, incredible_mass),
                       name = name, width = width,
                       tol = tol, ...)
  hdi_lower_tail_prob <- opt_info$minimum
  return(c(name(hdi_lower_tail_prob, ...),
           name(width + hdi_lower_tail_prob, ...)))

}
```

1 Approximating a distribution with a large sample

The concept of representing a distribution by a large representative sample is foundation for the approach to Bayesian analysis of complex models. The larger the sample, the better the estimation.

The `hdi_of_icdf()` function will compute the analytic 95% HDIs for the distribution under consideration, such as Beta($\theta|15, 7$).

```
h <- hdi_of_icdf(name = qbeta,
                 shape1=15,shape2=7)
h
## [1] 0.4907001 0.8639305
```

We can compute the corresponding mode, by:

$$\omega = \frac{a-1}{a+b-2}$$

```
(omega <- (15-1)/(15+7-2))
## [1] 0.7
```

To visualize this distribution with 95% HDI

```r
tibble(theta = seq(from = 0, to = 1, length.out = 100)) %>%
  mutate(density = dbeta(theta, shape1 = 15, shape2 = 7)) %>%

  ggplot() +
  geom_area(aes(x = theta, y = density),
            fill = "steelblue") +
  geom_segment(aes(x = h[1], xend = h[2], y = 0, yend = 0),
               size = .75) +
  geom_point(aes(x = omega, y = 0),
             size = 1.5, shape = 19) +
  annotate(geom = "text", x = .675, y = .4,
           label = "95% HDI", color = "white") +
  scale_x_continuous(expression(theta),
                     breaks = c(0, h, omega, 1),
                     labels = c("0", h %>% round(2), omega, "1")) +
  ggtitle("Exact distribution") +
  ylab(expression(p(theta))) +
  theme_cowplot()
```

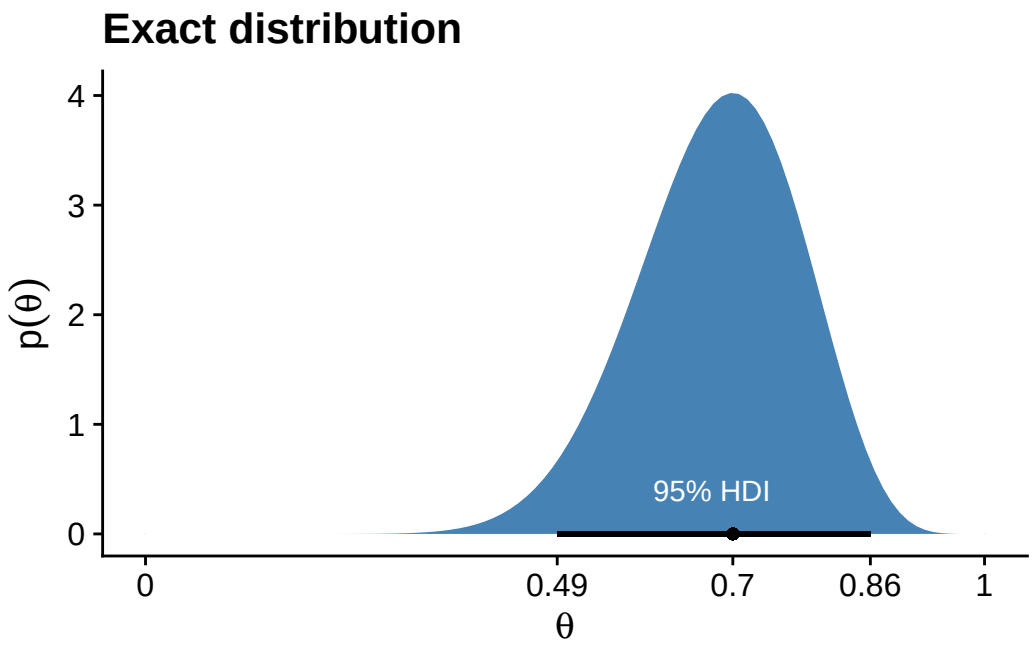

Figure 1: Beta distribution with 95% HDI

Suppose we have three sample with the same beta distribution above, but with different sample size ($n = 500, 5000, 50000$).

```r
set.seed(7)

d <- tibble(n=c(500,5000,50000)) %>%
  mutate(theta = map(n, ~rbeta(., shape1=15,shape2=7))) %>%
  unnest(theta) %>%
  mutate(key = str_c("Sample N = ", n))
head(d)
## # A tibble: 6 x 3
##       n theta key
##   <dbl> <dbl> <chr>
## 1   500 0.806 Sample N = 500
## 2   500 0.756 Sample N = 500
## 3   500 0.727 Sample N = 500
## 4   500 0.784 Sample N = 500
## 5   500 0.782 Sample N = 500
## 6   500 0.590 Sample N = 500
```

So, we will visualize these three samples with their 95% HDI

```r
d %>%
  ggplot(aes(x = theta, y = 0)) +
  # stat_histinterval() shows the mode and its 95% HDI
  stat_histinterval(point_interval = mode_hdi, .width = .95,
                    breaks = 30, fill = "steelblue") +
  scale_x_continuous(expression(theta), limits = c(0, 1)) +
  scale_y_continuous(NULL, breaks = NULL) +
  theme_cowplot() +
  facet_wrap(~ key, ncol = 3, scales = "free")
```

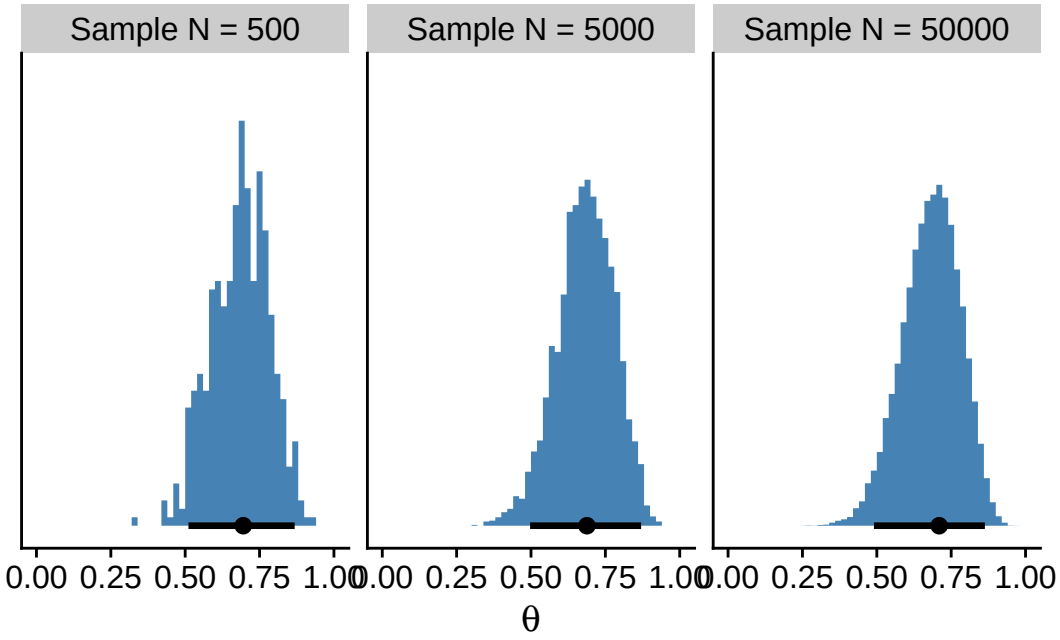

Figure 2: Three Beta distributions with 95% HDI

To get the exact number, we can do:

```
d %>% group_by(key) %>% mode_hdi(theta)
## # A tibble: 3 x 7
##   key              theta .lower .upper .width .point .interval
##   <chr>            <dbl>  <dbl>  <dbl>  <dbl> <chr>  <chr>
## 1 Sample N = 500   0.695  0.511  0.868   0.95 mode   hdi
## 2 Sample N = 5000  0.688  0.497  0.870   0.95 mode   hdi
## 3 Sample N = 50000 0.710  0.490  0.863   0.95 mode   hdi
```

Mentioned earlier, the larger the sample size, the better the estimation. So, let's do the estimation based on the beta distribution Beta($\theta|15, 7$) with varying N values.

```
my_mode_simulation <- function(seed) {

  set.seed(seed)

  tibble(n = c(500, 5000, 50000)) %>%
    mutate(theta = map(n, ~rbeta(., shape1 = 15, shape2 = 7))) %>%
    unnest(theta) %>%
    mutate(key = str_c("Sample N = ", n)) %>%
```

```
    group_by(key) %>%
    mode_hdi(theta)

}
```

Simulate, and visualize the outputs.

```
# we need an index of the values we set our seed with in our `my_mode_simulation()` functi
sim <-
  tibble(seed = 1:1e3) %>%
  group_by(seed) %>%
  # inserting our subsamples
  mutate(modes = map(seed, my_mode_simulation)) %>%
   # unnesting allows us to access our model results
  unnest(modes)

sim %>%
  ggplot(aes(x = theta, y = key)) +
  geom_vline(xintercept = .7, color = "white") +
  stat_histinterval(.width = c(.5, .95), breaks = 20, fill = "steelblue") +
  labs(title = expression("Variability of the mode for simulations of "*beta(theta*'|'*15*
       subtitle = "For each sample size, the dot is the median, the inner thick line is th
       x = "mode",
       y = NULL) +
  coord_cartesian(xlim = c(.6, .8),
                  ylim = c(1.25, 3.5)) +
  theme_cowplot(font_size = 11.5) +
  theme(axis.text.y = element_text(hjust = 0),
        axis.ticks.y = element_blank())
```

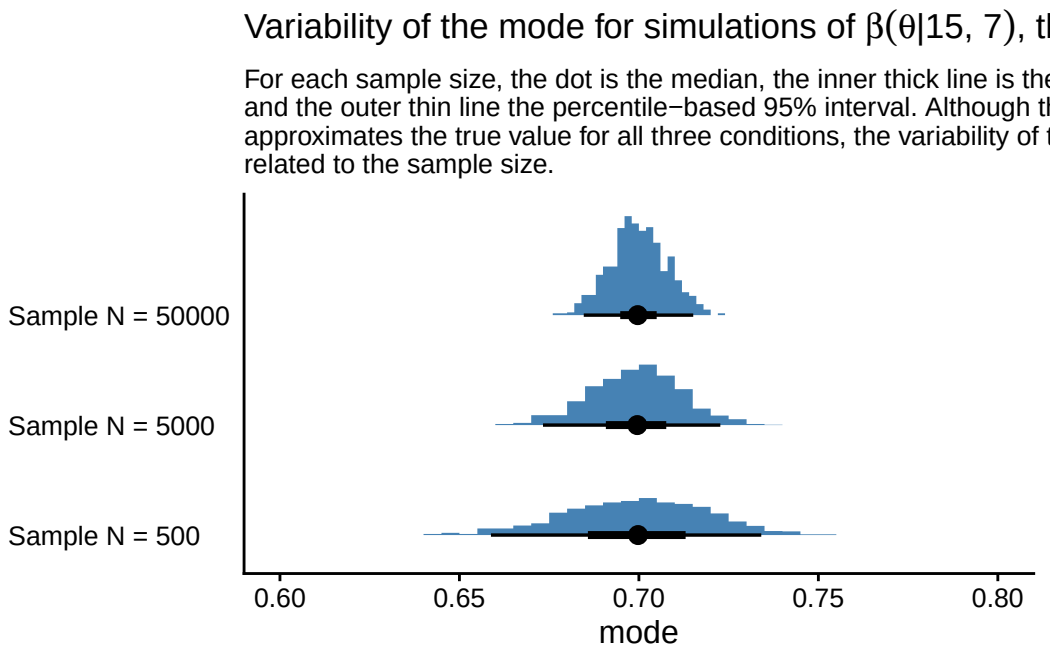

Figure 3: Estimate three Beta distributions with 95% HDI

2 A simple case of the Metropolis algorithm

> **Note**
>
> The goal in Bayesian inference is to get an accurate representation of the posterior distribution. One way to do that is to sample a large number of representative points from the posterior.

2.1 A politician stumbles on the Metropolis algorithm

Denote P_{proposed} as the population of the proposed island, and P_{current} as the population of the current island.

$$p_{\text{move}} = \frac{P_{\text{proposed}}}{P_{\text{current}}}$$

In the long run, the probability that the politician is on any one of the islands exactly matches the relative population of the island.

2.2 A random walk

```
set.seed(7)

num_days  <- 5e4
positions <- rep(0, num_days)
current   <- 4
for (i in 1:num_days) {
  # record current position
  positions[i] <- current
  # flip coin to generate proposal
  proposal <- current + sample(c(-1, 1), size = 1)
  # now make sure he loops around from 7 back to 1
  if (proposal < 1) proposal <- 7
  if (proposal > 7) proposal <- 1
  # move?
  prob_accept_the_proposal <- proposal/current
  current <- ifelse(runif(1) < prob_accept_the_proposal, proposal, current)
}
```

The `positions` is the main product of the simulation. Let's visualize it.

```
tibble(theta = positions) %>%

  ggplot(aes(x = theta)) +
  geom_bar(fill = "steelblue") +
  scale_x_continuous(expression(theta), breaks = 1:7) +
  scale_y_continuous(expand = expansion(mult = c(0, 0.05))) +
  theme_cowplot()
```

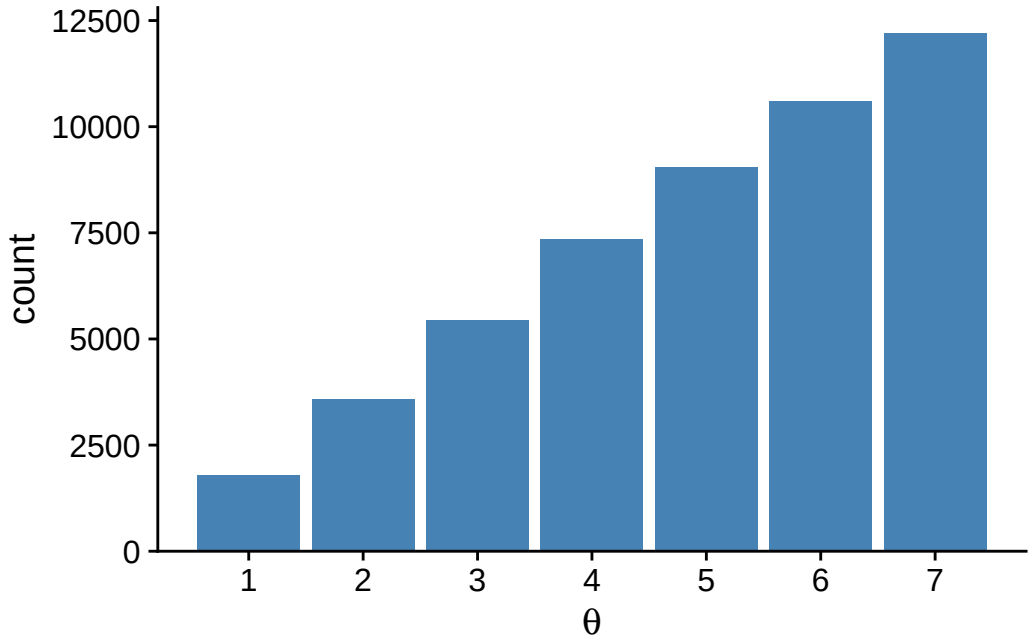

Figure 4: Number of positions with varying theta

To show the θ varying at different time points,

```
tibble(t     = 1:5e4,
       theta = positions) %>%
  slice(1:500) %>%

  ggplot(aes(x = theta, y = t)) +
  geom_path(size = 1/4, color = "steelblue") +
  geom_point(size = 1/2, alpha = 1/2, color = "steelblue") +
  scale_x_continuous(expression(theta), breaks = 1:7) +
  scale_y_log10("Time Step", breaks = c(1, 2, 5, 20, 100, 500)) +
  theme_cowplot()
```

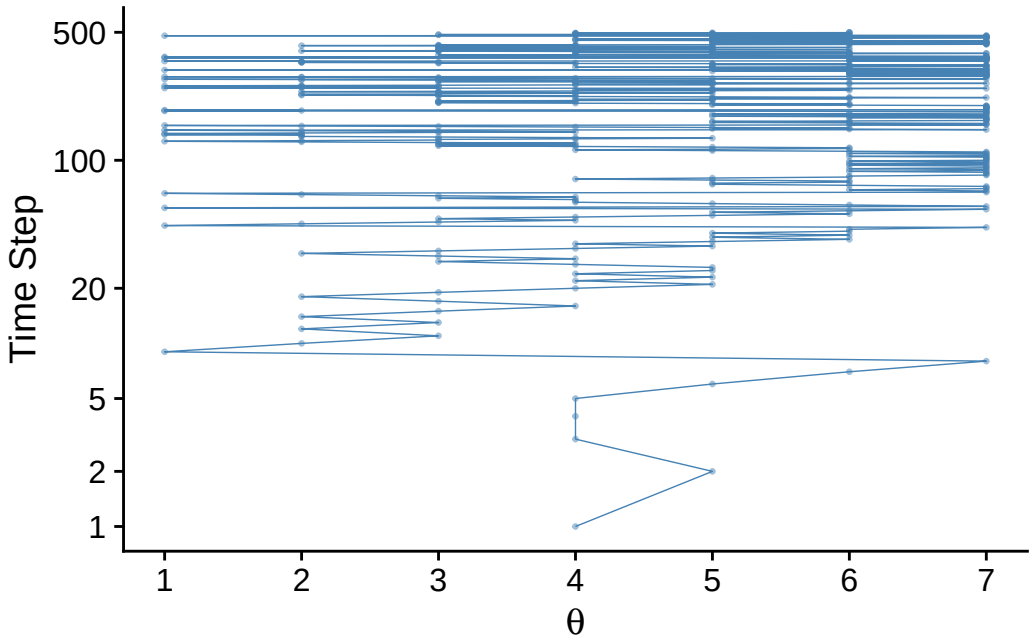

Figure 5: Time step against theta

To show the relative proportion of θ

```
tibble(x = 1:7,
       y = 1:7) %>%

  ggplot(aes(x = x, y = y)) +
  geom_col(width = .2, fill = "steelblue") +
  scale_x_continuous(expression(theta), breaks = 1:7) +
  scale_y_continuous(expression(p(theta)), expand = expansion(mult = c(0, 0.05))) +
  theme_cowplot()
```

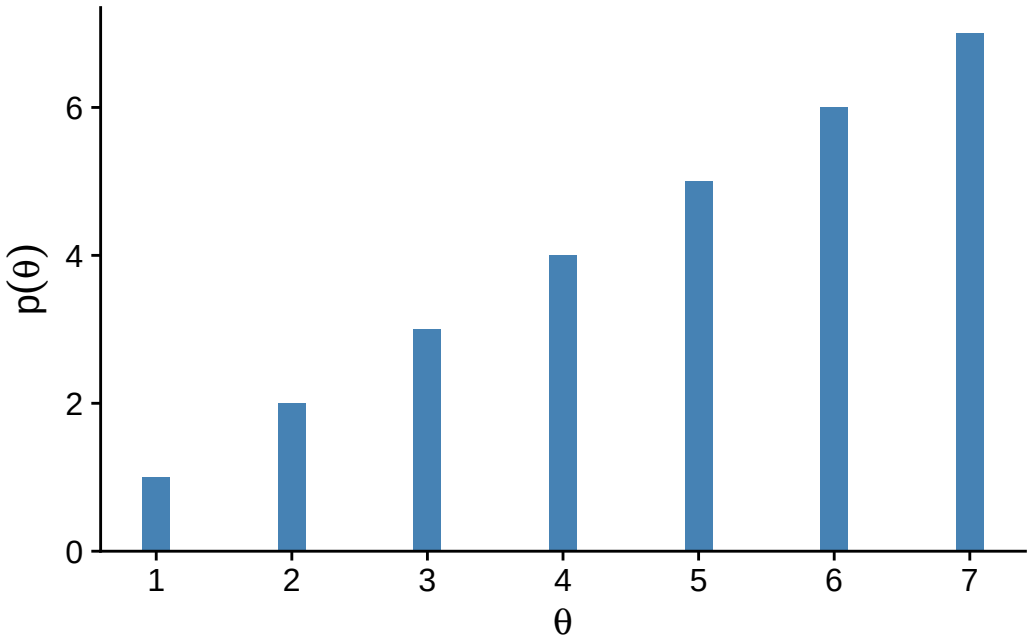

Figure 6: Relative population by theta

2.3 General properties of a random walk

The trajectory shown in Figure 5 is just one possible sequence of positions. At each time step, the direction of the proposed move is random, and if the relative probability of the proposed position is less than that of the current position, the acceptance of the proposed move is also random. However, in the long run, the relative frequency of visits mimics the target distribution.

Figure 7 shows the probability of being in each position as a function of time.

First, simulate:

```
nslots   <- 7
p_target <- 1:7
p_target <- p_target / sum(p_target)

# construct the transition matrix
proposal_matrix <- matrix(0, nrow = nslots, ncol = nslots)

for(from_idx in 1:nslots) {
  for(to_idx in 1:nslots) {
    if(to_idx == from_idx - 1) {proposal_matrix[from_idx, to_idx] <- 0.5}
```

```
    if(to_idx == from_idx + 1) {proposal_matrix[from_idx, to_idx] <- 0.5}
  }
}

# construct the acceptance matrix
acceptance_matrix <- matrix(0, nrow = nslots, ncol = nslots)

for(from_idx in 1:nslots) {
  for(to_idx in 1:nslots) {
    acceptance_matrix[from_idx, to_idx] <- min(p_target[to_idx] / p_target[from_idx], 1)
  }
}

# compute the matrix of move probabilities
move_matrix <- proposal_matrix * acceptance_matrix

# compute the transition matrix, including the probability of staying in place
transition_matrix <- move_matrix
for (diag_idx in 1:nslots) {
  transition_matrix[diag_idx, diag_idx] = 1.0 - sum(move_matrix[diag_idx, ])
}

# specify starting position vector:
position_vec <- rep(0, nslots)
position_vec[round(nslots / 2)] <- 1.0

p <- list()
data <-
  tibble(position = 1:nslots,
         prob     = position_vec)

# loop through the requisite time indexes
# update the data and transition vector
for(time_idx in 1:99) {

  p[[time_idx]] <- data

  # update the position vec
  position_vec <- position_vec %*% transition_matrix

  # update the data
```

```
  data <- NULL
  data <-
    tibble(position = 1:nslots,
           prob     = t(position_vec))
}
```

Then, plot

```
p %>%
  as_tibble_col() %>%
  mutate(facet = str_c("italic(t)==", 1:99)) %>%
  slice(c(1:14, 99)) %>%
  unnest(value) %>%
  bind_rows(
    tibble(position = 1:nslots,
           prob     = p_target,
           facet    = "target")
  ) %>%
  mutate(facet = factor(facet, levels = c(str_c("italic(t)==", c(1:14, 99)), "target"))) %

  # plot!
  ggplot(aes(x = position, y = prob, fill = facet == "target")) +
  geom_col(width = .2) +
  scale_fill_manual(values = c("steelblue", "goldenrod2"), breaks = NULL) +
  scale_x_continuous(expression(theta), breaks = 1:7) +
  scale_y_continuous(expression(italic(p)(theta)), expand = expansion(mult = c(0, 0.05))) +
  theme_cowplot() +
  facet_wrap(~ facet, scales = "free_y", labeller = label_parsed)
```

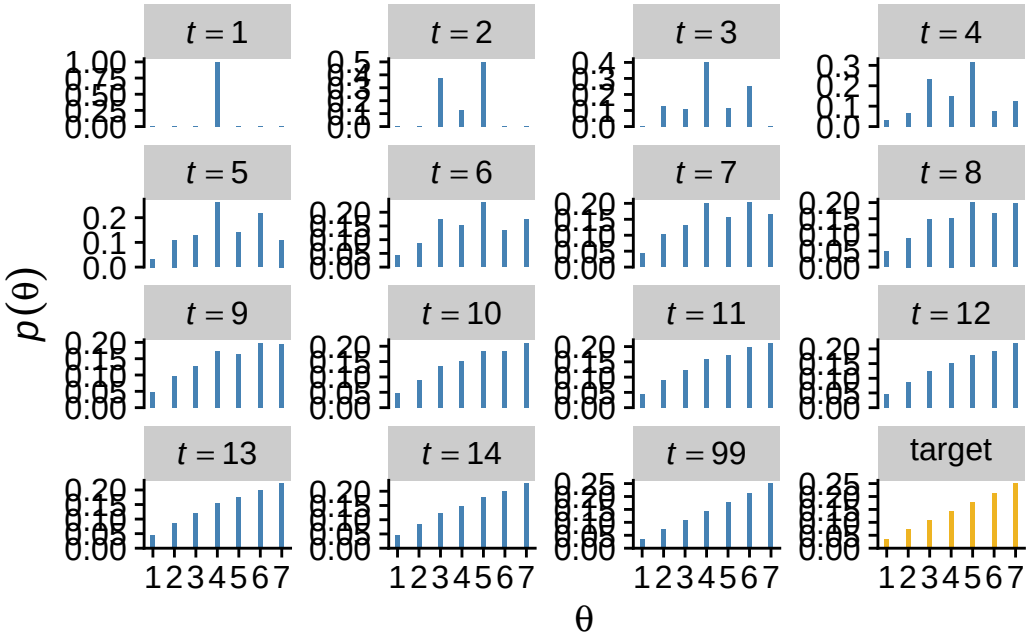

Figure 7: Probability of being in each position as a function of time

2.4 Through the random walk procedure

We can generate random samples from the target distribution, by doing indirectly. Moreover, we can generate those random samples from the target distribution even when the target distribution is not normalized.

In particular, when the target distribution $P(\theta)$ is a posterior proportional to $p(D|\theta)p(\theta)$, we can generate random representative values from the posterior distribution merely by evaluating $p(D|\theta)p(\theta)$.

3 The Metropolis algorithm more generally

Here, we will generate a proposed jump from a zero-mean normal distribution with a standard deviation of 0.2.

```
rnorm(1,mean = 0,sd=0.2)
## [1] -0.1985524
```

To know what draws from normal distribution looks like in the long run,

```r
mu    <- 0
sigma <- 0.2

# how many proposals would you like?
n  <- 500

set.seed(7)
tibble(proposed_jump = rnorm(n, mean = mu, sd = sigma)) %>%

  ggplot(aes(x = proposed_jump, y = 0)) +
  geom_jitter(width = 0, height = .1,
              size = 1/2, alpha = 1/2, color = "steelblue") +
  # this is the idealized distribution
  stat_function(fun = dnorm, args = list(mean = mu, sd = sigma),
                color = "steelblue") +
  scale_x_continuous(breaks = seq(from = -0.6, to = 0.6, length.out = 7)) +
  scale_y_continuous(NULL, breaks = NULL) +
  labs(title = "Jump proposals",
       subtitle = "The blue line shows the data generating distribution.") +
  theme_cowplot()
```

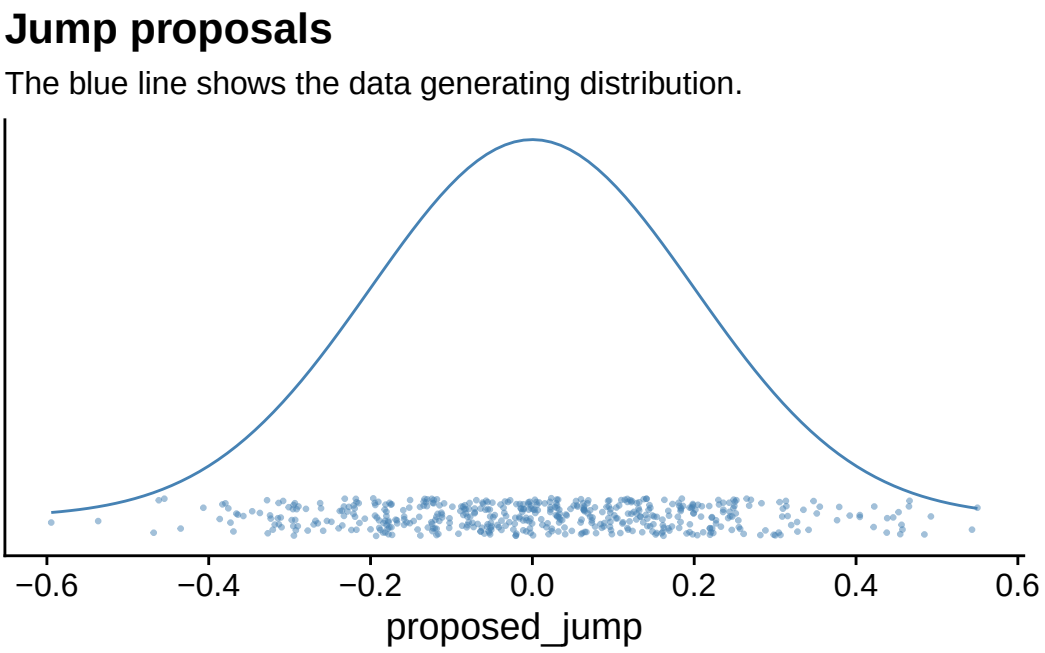

Figure 8: Data generated from a normal distribution

3.1 Metropolis algorithm applied to Bernoulli likelihood and beta prior

Here we have a function to apply Metropolis algorithm to Bernoulli likelihood

```
# specify the data, to be used in the likelihood function.
my_data <- c(rep(0, 6), rep(1, 14))

# define the Bernoulli likelihood function, p(D|theta).
# the argument theta could be a vector, not just a scalar
likelihood <- function(theta, data) {
  z <- sum(data)
  n <- length(data)
  p_data_given_theta <- theta^z * (1 - theta)^(n - z)
  # the theta values passed into this function are generated at random,
  # and therefore might be inadvertently greater than 1 or less than 0.
  # the likelihood for theta > 1 or for theta < 0 is zero
  p_data_given_theta[theta > 1 | theta < 0] <- 0
  return(p_data_given_theta)
}

# define the prior density function.
prior_d <- function(theta) {
  p_theta <- dbeta(theta, 1, 1)
  # the theta values passed into this function are generated at random,
  # and therefore might be inadvertently greater than 1 or less than 0.
  # the prior for theta > 1 or for theta < 0 is zero
  p_theta[theta > 1 | theta < 0] = 0
  return(p_theta)
}

# define the relative probability of the target distribution,
# as a function of vector theta. for our application, this
# target distribution is the unnormalized posterior distribution
target_rel_prob <- function(theta, data) {
  target_rel_prob <- likelihood(theta, data) * prior_d(theta)
  return(target_rel_prob)
}

# specify the length of the trajectory, i.e., the number of jumps to try:
traj_length <- 50000 # this is just an arbitrary large number

# initialize the vector that will store the results
```

```
trajectory <- rep(0, traj_length)

# specify where to start the trajectory:
trajectory[1] <- 0.01 # another arbitrary value

# specify the burn-in period
burn_in <- ceiling(0.0 * traj_length) # arbitrary number, less than `traj_length`

# initialize accepted, rejected counters, just to monitor performance:
n_accepted <- 0
n_rejected <- 0
```

Next function will make it easier to plug into the **purrr** function.

```
my_metropolis <- function(proposal_sd) {

  # now generate the random walk. the 't' index is time or trial in the walk.
  # specify seed to reproduce same random walk
  set.seed(47405)


  ## I'm taking this section out and will replace it

  # # specify standard deviation of proposal distribution
  # proposal_sd <- c(0.02, 0.2, 2.0)[2]

  ## end of the section I took out


  for (t in 1:(traj_length - 1)) {
    current_position <- trajectory[t]
    # use the proposal distribution to generate a proposed jump
    proposed_jump <- rnorm(1, mean = 0, sd = proposal_sd)
    # compute the probability of accepting the proposed jump
    prob_accept <- min(1,
                       target_rel_prob(current_position + proposed_jump, my_data)
                       / target_rel_prob(current_position, my_data))
    # generate a random uniform value from the interval [0, 1] to
    # decide whether or not to accept the proposed jump
    if (runif(1) < prob_accept) {
      # accept the proposed jump
```

```
      trajectory[t + 1] <- current_position + proposed_jump
      # increment the accepted counter, just to monitor performance
      if (t > burn_in) {n_accepted <- n_accepted + 1}
    } else {
      # reject the proposed jump, stay at current position
      trajectory[t + 1] <- current_position
      # increment the rejected counter, just to monitor performance
      if (t > burn_in) {n_rejected <- n_rejected + 1}
    }
  }

  # extract the post-burn_in portion of the trajectory
  accepted_traj <- trajectory[(burn_in + 1) : length(trajectory)]

  tibble(accepted_traj = accepted_traj,
         n_accepted    = n_accepted,
         n_rejected    = n_rejected)
  # end of Metropolis algorithm

}
```

So, run the analysis

```
d <- tibble(proposal_sd = c(0.02,0.2,2.0)) %>%
  mutate(accepted_traj = map(proposal_sd, my_metropolis)) %>%
  unnest(accepted_traj)
glimpse(d)
## Rows: 150,000
## Columns: 4
## $ proposal_sd   <dbl> 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.~
## $ accepted_traj <dbl> 0.01000000, 0.01000000, 0.01000000, 0.01000000, 0.011491~
## $ n_accepted    <dbl> 46801, 46801, 46801, 46801, 46801, 46801, 46801, 46801, ~
## $ n_rejected    <dbl> 3198, 3198, 3198, 3198, 3198, 3198, 3198, 3198, 3198, 31~
```

Visualize the model

```
d <-
  d %>%
  mutate(proposal_sd = str_c("Proposal SD = ", proposal_sd),
         iter        = rep(1:50000, times = 3))

d %>%
```

```
  ggplot(aes(x = accepted_traj, y = 0)) +
  stat_histinterval(point_interval = mode_hdi, .width = .95,
                    fill = "steelblue", slab_color = "white", outline_bars = T,
                    breaks = 40, normalize = "panels") +
  scale_x_continuous(expression(theta), breaks = 0:5 * 0.2) +
  scale_y_continuous(NULL, breaks = NULL) +
  theme_cowplot() +
  # make it easier to differentiate among subplots by adding lightweight panel borders
  panel_border() +
  facet_wrap(~ proposal_sd, ncol = 3)
```

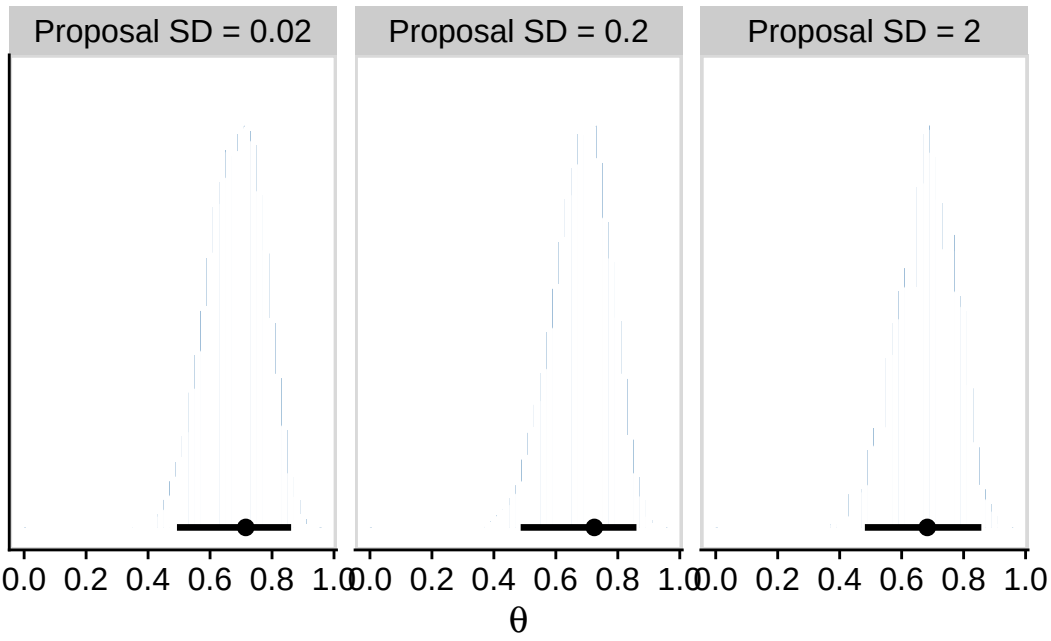

Figure 9: Theta distribution with varying sd

We will need the estimates for the effectiveness sample size for each chain.

```
d %>%
  select(proposal_sd, accepted_traj) %>%
  group_by(proposal_sd) %>%
  summarise(Eff.Sz. = posterior::ess_mean(accepted_traj))
## # A tibble: 3 x 2
##   proposal_sd        Eff.Sz.
##   <chr>                <dbl>
```

```
## 1 Proposal SD = 0.02    500.
## 2 Proposal SD = 0.2   11340.
## 3 Proposal SD = 2      2105.
```

3.2 Summary of Metropolis algorithm

> **Note**
>
> This idea is that we get a handle on the posterior distribution by generating a large sample of representative values. The larger the sample, the more accurate is the approximation. This is a sample of representative credible parameter values from the posterior distribution; it is not a resampling of data.
> Representative parameter values can be randomly sampled from complicated posterior distributions without solving the integral in Bayes' rule, and by using only simple proposal distributions for which efficient random number generators already exist.

4 Toward Gibbs sampling: Estimating two coin biases

4.1 Prior, likelihood and posterior for two biases

We can considering situations in which there are two underlying biases, namely θ_1 and θ_2 for two coins. We want to determine what we should believe about these biases after we have observed some data from two coins.

$$p(\theta_1, \theta_2|D) = \frac{p(D|\theta_1, \theta_2)p(\theta_1, \theta_2)}{p(D)}$$

4.2 Hamiltonian Monte Carlo sampling

```
# set parameters
# coin 1
n1 <- 8; z1 <- 6
# coin 2
n2 <- 7; z2 <- 2

# make the data
theta_1_data <- rep(0:1, times = c(n1 - z1, z1))
```

```
theta_2_data <- rep(0:1, times = c(n2-z2,z2))

theta_1_data;theta_2_data
## [1] 0 0 1 1 1 1 1 1
## [1] 0 0 0 0 0 1 1
```

4.2.1 Uneven multivariate Bernoulli via `resp_subset()`

```
d <- tibble(
  y1=theta_1_data,
  y2=c(theta_2_data,NA)
)
d <- d %>%
  mutate(subset = if_else(is.na(y2), FALSE, TRUE))
```

In this case, we will need to build the model with only intercept. Here, we define two models as separate objects.

```
model_1 <- bf(y1 ~ 1)
# the subset variable in the data is the logical variable for criterion y2
model_2 <- bf(y2 | subset(subset) ~1)
```

By default, `brms` package assumes an unbounded parameter space for the standard intercept priors. Here we know the beta distribution imposes boundaries within the range of $[0, 1]$. So we can set it manually. `resp=y1` means the prior is connected to the `y1` criterion.

```
fit3.1a <- brm(data = d,
               family = bernoulli(link = identity),
               model_1+model_2,
               prior = c(prior(beta(2,2), class = Intercept, lb=0,ub=1, resp = y1),
                         prior(beta(2,2), class = Intercept, lb=0,ub=1,resp=y2)),
               iter = 3000, warmup = 500, cores = 3, chains = 3,
               seed = 7, file = "fits/fit03.01a")
draws.a <- as_draws_df(fit3.1a); head(draws.a)
## # A draws_df: 6 iterations, 1 chains, and 4 variables
##   b_y1_Intercept b_y2_Intercept lprior lp__
## 1           0.60           0.42  0.743  -11
## 2           0.73           0.39  0.524  -11
## 3           0.69           0.33  0.518  -11
## 4           0.64           0.59  0.690  -12
```

```
## 5           0.63           0.60  0.694  -13
## 6           0.69           0.17  0.075  -12
## # ... hidden reserved variables {'.chain', '.iteration', '.draw'}
```

To reduce the overplotting, we only look at the first 500 post-warmup draws.

```
p1 <- draws.a %>%
  filter(.draw<501) %>%
  ggplot(aes(x=b_y1_Intercept,y=b_y2_Intercept))+
  geom_point(alpha=1/4, color="steelblue")+
  geom_path(size=1/10, alpha=1/2, color="steelblue")+
  scale_x_continuous(expression(theta[1]),
                     breaks = 0:5/5, expand = c(0,0), limits = 0:1)+
  scale_y_continuous(expression(theta[2]),
                     breaks = 0:5/5, expand = c(0,0), limits = 0:1)+
  labs(subtitle = "fit3.1a (resp_subset() method)")+
  coord_equal()+
  theme_cowplot()
p1
```

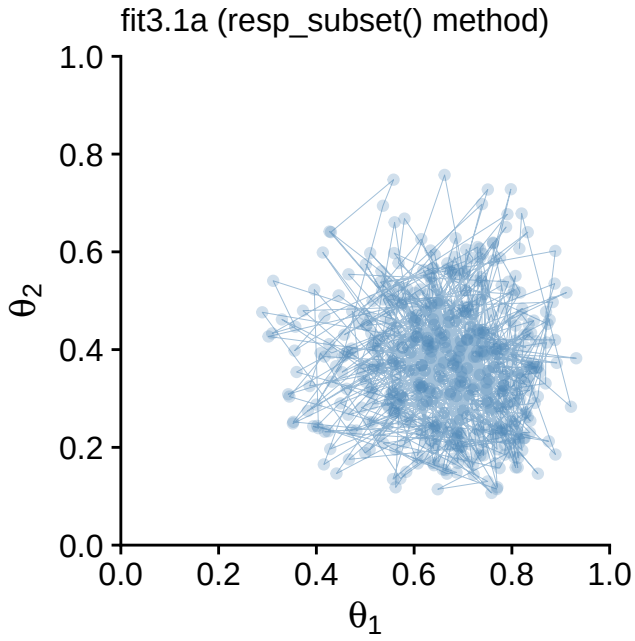

Figure 10: Posterior distribution of two bernoulli theta

4.2.2 Uneven multivariate Bernoulli via the aggregated binomial approach

```
d <- tibble(n1=n1,
            z1=z1,
            n2=n2,
            z2=z2)
```

`resp_trials()` allows to incorporate additional information about the criterion variable into the model. In this case, we will use `trials()`. We have `z1==6` heads out of `n1==8` trials. The expression as an intercept-only model is `z1|trials(n1)~1` .

```
model_1 <- bf(z1 | trials(n1)~1)
model_2 <- bf(z2 | trials(n2)~1)
```

Because the Bernoulli function is the special case of the binomial function for which $N = 1$, for which each data point is a discrete trial. So we will set `family=binomial(link="identity")`, when each data point is an aggregated of multiple trials.

```
fit3.1b <- brm(data = d,
               family = binomial(link = "identity"),
               model_1+model_2,
               prior = c(prior(beta(2,2),class=Intercept, lb = 0, ub = 1, resp = z1),
                         prior(beta(2,2),class=Intercept, lb = 0, ub = 1, resp = z2)),
      iter = 3000, warmup = 500, cores = 3, chains = 3,
      seed = 7,
      file = "fits/fit03.01b")
draws.b <- as_draws_df(fit3.1b);head(draws.b)
## # A draws_df: 6 iterations, 1 chains, and 4 variables
##   b_z1_Intercept b_z2_Intercept lprior lp__
## 1           0.82           0.12  -0.54 -7.7
## 2           0.64           0.19   0.23 -5.9
## 3           0.62           0.16   0.11 -6.3
## 4           0.43           0.30   0.61 -6.4
## 5           0.48           0.31   0.66 -5.8
## 6           0.57           0.44   0.77 -5.2
## # ... hidden reserved variables {'.chain', '.iteration', '.draw'}
```

```
p2 <- draws.b %>%
  filter(.draw < 501) %>%

  ggplot(aes(x = b_z1_Intercept, y = b_z2_Intercept)) +
```

```
  geom_point(alpha = 1/4, color = "steelblue") +
  geom_path(size = 1/10, alpha = 1/2, color = "steelblue") +
  scale_x_continuous(expression(theta[1]),
                     breaks = 0:5 / 5, expand = c(0, 0), limits = 0:1) +
  scale_y_continuous(expression(theta[2]),
                     breaks = 0:5 / 5, expand = c(0, 0), limits = 0:1) +
  labs(subtitle = "fit3.1b (aggregated binomial method)") +
  coord_equal() +
  theme_cowplot()

p1+p2
```

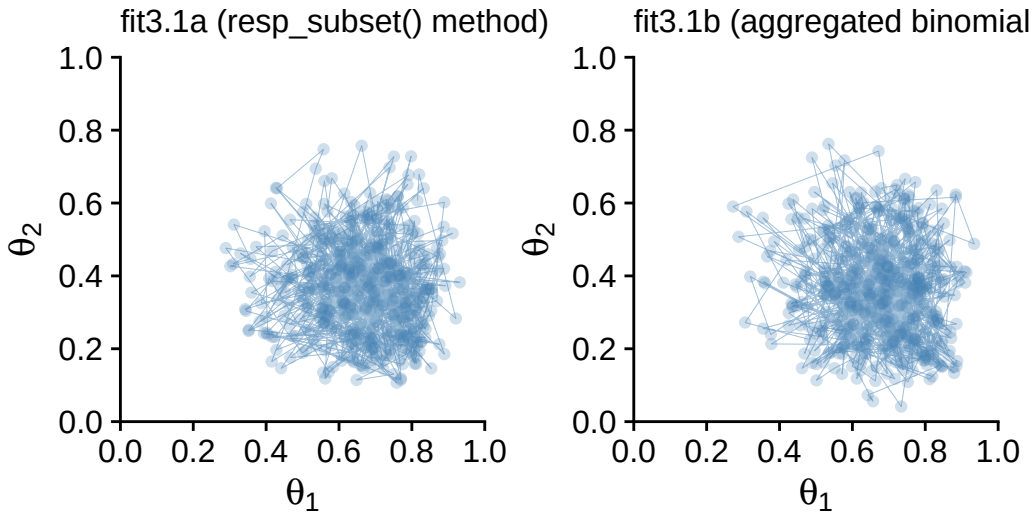

Figure 11: Posterior distribution of two bernoulli theta

We might want to compare the two model types by plotting the marginal posterior densities.

```
draws <-
  bind_rows(
    draws.a %>% transmute(`theta[1]` = b_y1_Intercept, `theta[2]` = b_y2_Intercept),
    draws.b %>% transmute(`theta[1]` = b_z1_Intercept, `theta[2]` = b_z2_Intercept)
  ) %>%
  mutate(fit = rep(c("fit3.1a", "fit3.1b"), each = n() / 2))

# wrangle
draws %>%
  pivot_longer(-fit) %>%
```

```
# plot
ggplot(aes(x = value, y = 0)) +
stat_halfeye(point_interval = mode_hdi, .width = .95, fill = "steelblue") +
scale_x_continuous("posterior", breaks = 0:5 / 5,
                   expand = c(0, 0), limits = 0:1) +
scale_y_continuous(NULL, breaks = NULL) +
theme_cowplot() +
theme(panel.spacing.x = unit(0.75, "cm")) +
panel_border() +
facet_grid(fit ~ name, labeller = label_parsed)
```

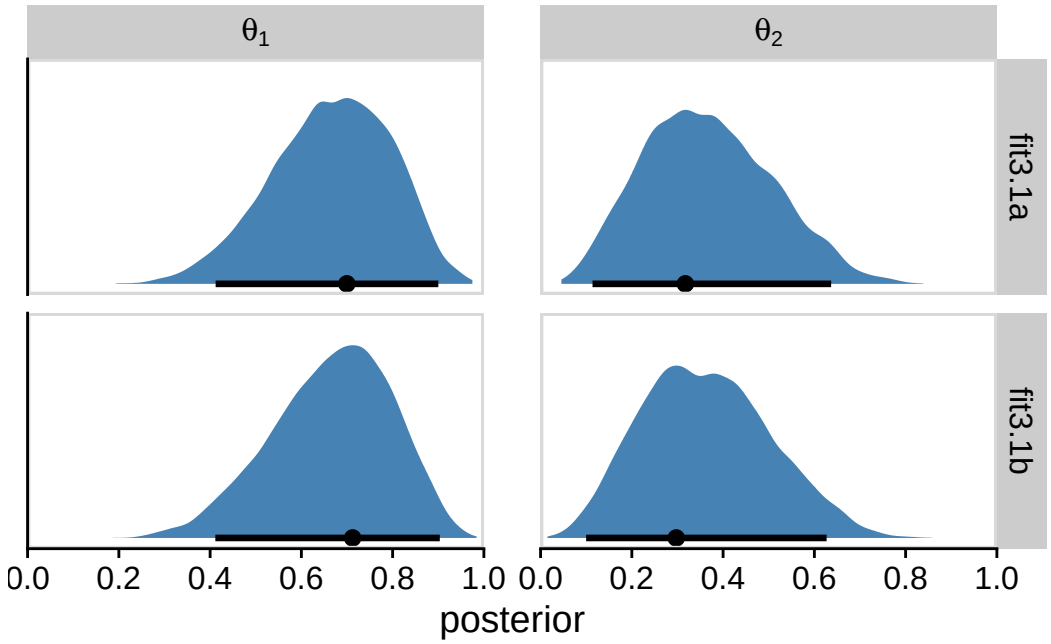

Figure 12: Posterior samples from the two models

4.3 Difference between biases

```
draws %>%
  mutate(dif = `theta[1]` - `theta[2]`,
         fit = if_else(fit == "fit3.1a",
                       "fit3.1a (resp_subset() method)",
                       "fit3.1b (aggregated binomial method)")) %>%
```

```
  ggplot(aes(x = dif, y = fit)) +
  stat_histinterval(point_interval = mode_hdi, .width = .95,
                    fill = "steelblue2", slab_color = "steelblue4", outline_bars = T,
                    breaks = 40, normalize = "panels") +
  geom_vline(xintercept = 0, linetype = 3) +
  labs(x = expression(theta[1]-theta[2]),
       y = NULL) +
  coord_cartesian(ylim = c(1.5, 2.4)) +
  theme_cowplot()
```

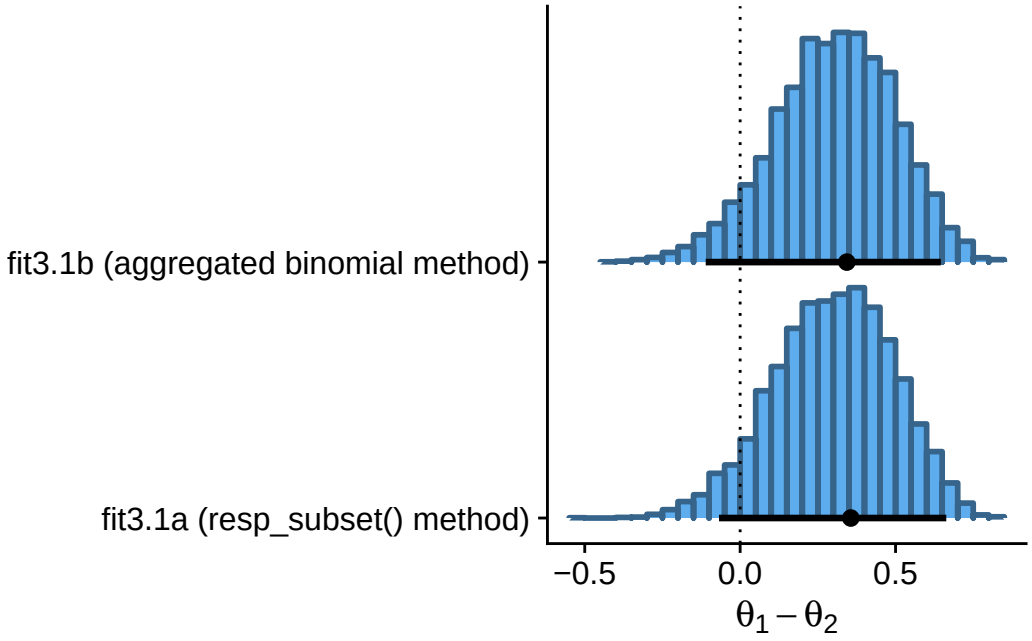

Figure 13: Differences between theta1 and theta2

The exact estimates of the mode and 95% HDIs for the difference distribution $\theta_1 - \theta_2$ is:

```
draws %>%
  mutate(dif = `theta[1]`-`theta[2]`) %>%
  group_by(fit) %>%
  mode_hdi(dif) %>%
  mutate_if(is.double, round, digits = 3)
## # A tibble: 2 x 7
##   fit       dif .lower .upper .width .point .interval
##   <chr>   <dbl>  <dbl>  <dbl>  <dbl> <chr>  <chr>
```

```
## 1 fit3.1a 0.356 -0.068  0.663   0.95 mode   hdi
## 2 fit3.1b 0.343 -0.111  0.646   0.95 mode   hdi
```

Among the three primary measures of central tendency, modes are particularly sensitive to things like sample variance. If we compare the two methods with the mean, we will get:

```
draws %>%
  mutate(dif = `theta[1]`-`theta[2]`) %>%
  group_by(fit) %>%
  summarise(mean_of_the_dif = mean(dif) %>% round(digits=3))
## # A tibble: 2 x 2
##   fit     mean_of_the_dif
##   <chr>             <dbl>
## 1 fit3.1a           0.3
## 2 fit3.1b           0.304
```

5 MCMC representativeness, accuracy, and efficiency

> ! Three main goals in generating an MCMC sample from the posterior distribution
>
> 1. The values in the chain must be *representative* of the posterior distribution
>   1. should not be unduly influenced by the arbitrary initial value of the chain
>   2. should fully explore the range of the posterior distribution without getting stuck
> 2. The chain should be of sufficient size so that estimates are *accurate and stable.*
>   1. Estimates of the central tendency and the limits of the 95% HDI should not be much different if the MCMC analysis is run again, using different seeds
> 3. The chain should be generated *efficiently*, with as few steps as possible

5.1 MCMC representativeness

Suppose a trial following a Bernoulli distribution, occurring 35 out of 50.

```
z <- 35;n <- 50
d <- tibble(y=rep(0:1, times=c(n-z,z)))
```

```
fit3.2 <- brm(data = d,
              family = bernoulli(link=identity),
              y~1,
              prior(beta(2,2),class=Intercept,lb=0,ub=1),
              iter=10000, warmup=500, cores=3, chains=3,
              seed=7, file=here("fits","fit03.02"))
```

i burn-in period

Burn-in period is the preliminary steps, during which the chain moves from its unrepresentative initial value to the model region of the posterior.
In reality, it is routine to apply a burn-in period of several hundred to thousand steps.

The first `n` iterations for each MCMC chain are "warmups". HMC warmups are discarded and not used to describe the posterior, somewhat analogous to but not synonymous with burn-in iterations.

```
draws <- as_draws_df(fit3.2)

draws %>%
  mutate(chain = factor(.chain)) %>%

  ggplot(aes(x = b_Intercept, y = chain, fill = chain)) +
  stat_halfeye(point_interval = mode_hdi,
               .width = .95) +
  scale_fill_brewer(direction = -1) +
  scale_y_discrete(expand = expansion(mult = 0.035)) +
  theme_cowplot() +
  theme(legend.position = "none")
```

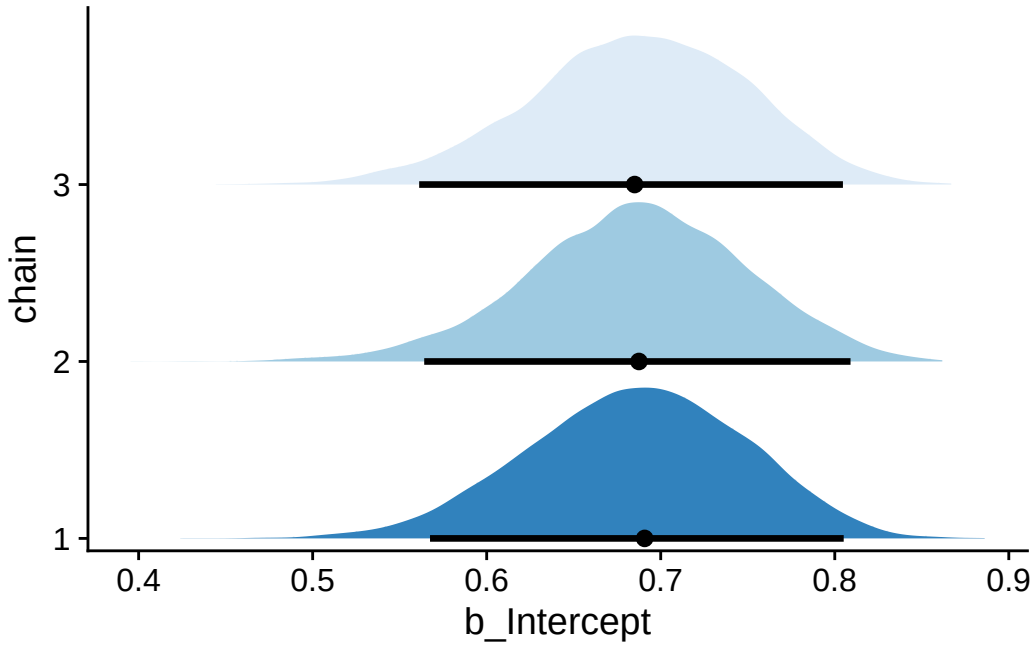

Figure 14: Posterior distribuition with 95% HDI for three chains-I

```
draws %>%
  mutate(chain = factor(.chain)) %>%

  ggplot(aes(x = b_Intercept, y = chain, fill = chain)) +
  stat_halfeye(point_interval = mode_hdi, .width = .95,
               height = 9, alpha = 3/4) +
  scale_fill_brewer() +
  scale_y_discrete(expand = expansion(mult = 0.2)) +
  theme_cowplot() +
  theme(legend.position = "none")
```

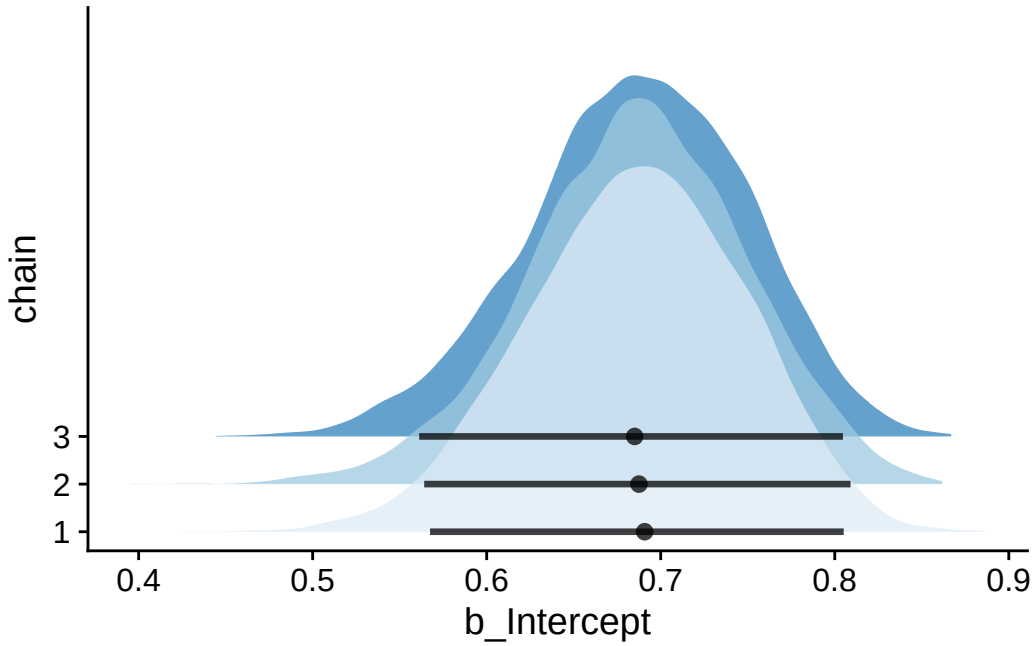

Figure 15: Posterior distribuition with 95% HDI for three chains-II

5.2 MCMC accuracy

> **Note**
>
> When calculating the chain length and accuracy, we should take the clumpiness of the chain into account. We will measure clumpiness as *autocorrelation*, which is simply the correlation of the chain values with the chain values k steps ahead. For each choice of k, the autocorrelation is different.