CHAPTER 7

Markov Chain Monte Carlo

Contents

7.1. Approximating a Distribution with a Large Sample . . . 145
7.2. A Simple Case of the Metropolis Algorithm . . . 146
  7.2.1 A politician stumbles upon the Metropolis algorithm . . . 146
  7.2.2 A random walk . . . 147
  7.2.3 General properties of a random walk . . . 149
  7.2.4 Why we care . . . 152
  7.2.5 Why it works . . . 152
7.3. The Metropolis Algorithm More Generally . . . 156
  7.3.1 Metropolis algorithm applied to Bernoulli likelihood and beta prior . . . 157
  7.3.2 Summary of Metropolis algorithm . . . 161
7.4. Toward Gibbs Sampling: Estimating Two Coin Biases . . . 162
  7.4.1 Prior, likelihood and posterior for two biases . . . 163
  7.4.2 The posterior via exact formal analysis . . . 165
  7.4.3 The posterior via the Metropolis algorithm . . . 168
  7.4.4 Gibbs sampling . . . 170
  7.4.5 Is there a difference between biases? . . . 176
  7.4.6 Terminology: MCMC . . . 177
7.5. MCMC Representativeness, Accuracy, and Efficiency . . . 178
  7.5.1 MCMC representativeness . . . 178
  7.5.2 MCMC accuracy . . . 182
  7.5.3 MCMC efficiency . . . 187
7.6. Summary . . . 188
7.7. Exercises . . . 189

You furtive posterior: coy distribution.
Alluring, curvaceous, evading solution.
Although I can see what you hint at is ample,
I'll settle for one representative sample.[1]

[1] This chapter is about methods for approximating a posterior distribution by collecting from it a large representative sample. These methods are important because complex posterior distributions are otherwise very challenging to get a handle on. The poem says merely that complexly shaped posterior distributions are evasive, but instead of demanding a precise solution, we will do practical analysis with a representative sample. Some people have suggested that the poem seems to allude to something else, but I don't know what they could mean.

Doing Bayesian Data Analysis, 2nd Edition
http://dx.doi.org/10.1016/B978-0-12-405888-0.00007-6

This chapter introduces the methods we will use for producing accurate approximations to Bayesian posterior distributions for realistic applications. The class of methods is called Markov chain Monte Carlo (MCMC), for reasons that will be explained later in the chapter. It is MCMC algorithms and software, along with fast computer hardware, that allow us to do Bayesian data analysis for realistic applications that would have been effectively impossible 30 years ago. This chapter explains some of the essential ideas of MCMC methods that are crucial to understand before using the sophisticated software that make MCMC methods easy to apply.

In this chapter, we continue with the goal of inferring the underlying probability θ that a coin comes up heads, given an observed set of flips. In Chapter 6, we considered the scenario when the prior distribution is specified by a function that is conjugate to the likelihood function, and thus yields an analytically solvable posterior distribution. In Chapter 5, we considered the scenario when the prior is specified on a dense grid of points spanning the range of θ values, and thus the posterior is numerically generated by summing across the discrete values.

But there are situations in which neither of those previous methods will work. We already recognized the possibility that our prior beliefs about θ could not be adequately represented by a beta distribution or by any function that yields an analytically solvable posterior function. Grid approximation is one approach to addressing such situations. When we have just one parameter with a finite range, then approximation by a grid is a useful procedure. But what if we have several parameters? Although we have, so far, only been dealing with models involving a single parameter, it is much more typical, as we will see in later chapters, to have models involving several parameters. In these situations, the parameter space cannot be spanned by a grid with a reasonable number of points. Consider, for example, a model with, say, six parameters. The parameter space is, therefore, a six-dimensional space that involves the *joint* distribution of all *combinations* of parameter values. If we set up a comb on each parameter that has 1,000 values, then the six-dimensional parameter space has $1{,}000^6 = 1{,}000{,}000{,}000{,}000{,}000{,}000$ combinations of parameter values, which is too many for any computer to evaluate. In anticipation of those situations when grid approximation will not work, we explore a new method called Markov chain Monte Carlo, or MCMC for short. In this chapter, we apply MCMC to the simple situation of estimating a single parameter. In real research you would probably not want to apply MCMC to such simple one-parameter models, instead going with mathematical analysis or grid approximation. But it is very useful to *learn* about MCMC in the one-parameter context.

The method described in this chapter assumes that the prior distribution is specified by a function that is easily evaluated. This simply means that if you specify a value for θ, then the value of $p(\theta)$ is easily determined, especially by a computer. The method also assumes that the value of the likelihood function, $p(D|\theta)$, can be computed for any

specified values of D and θ. (Actually, all that the method really demands is that the prior and likelihood can be easily computed up to a multiplicative constant.) In other words, the method does *not* require evaluating the difficult integral in the denominator of Bayes' rule.

What the method produces for us is an approximation of the posterior distribution, $p(\theta|D)$, in the form of a large number of θ values sampled from that distribution. This heap of representative θ values can be used to estimate the central tendency of the posterior, its highest density interval (HDI), etc. The posterior distribution is estimated by randomly generating a lot of values from it, and therefore, by analogy to the random events at games in a casino, this approach is called a Monte Carlo method.

7.1. APPROXIMATING A DISTRIBUTION WITH A LARGE SAMPLE

The concept of representing a distribution by a large representative sample is foundational for the approach we take to Bayesian analysis of complex models. The idea is applied intuitively and routinely in everyday life and in science. For example, polls and surveys are founded on this concept: By randomly sampling a subset of people from a population, we estimate the underlying tendencies in the entire population. The larger the sample, the better the estimation. What is new in the present application is that the population from which we are sampling is a mathematically defined distribution, such as a posterior probability distribution.

Figure 7.1 shows an example of approximating an exact mathematical distribution by a large random sample of representative values. The upper-left panel of Figure 7.1 plots an exact beta distribution, which could be the posterior distribution for estimating the underlying probability of heads in a coin. The exact mode of the distribution and the 95% HDI are also displayed. The exact values were obtained from the mathematical formula of the beta distribution. We can approximate the exact values by randomly sampling a large number of representative values from the distribution. The upper-right panel of Figure 7.1 shows a histogram of 500 random representative values. With only 500 values, the histogram is choppy, not smooth, and the estimated mode and 95% HDI are in the vicinity of the true values, but unstable in the sense that a different random sample of 500 representative values would yield noticeably different estimates. The lower-left panel shows a larger sample size of 5,000 representative values, and the lower-right panel shows a larger sample yet, with 50,000 representative values. You can see that for larger samples sizes, the histogram becomes smoother, and the estimated values become closer (on average) to the true values. (The computer generates pseudorandom values; see Footnote 4, p. 74.)

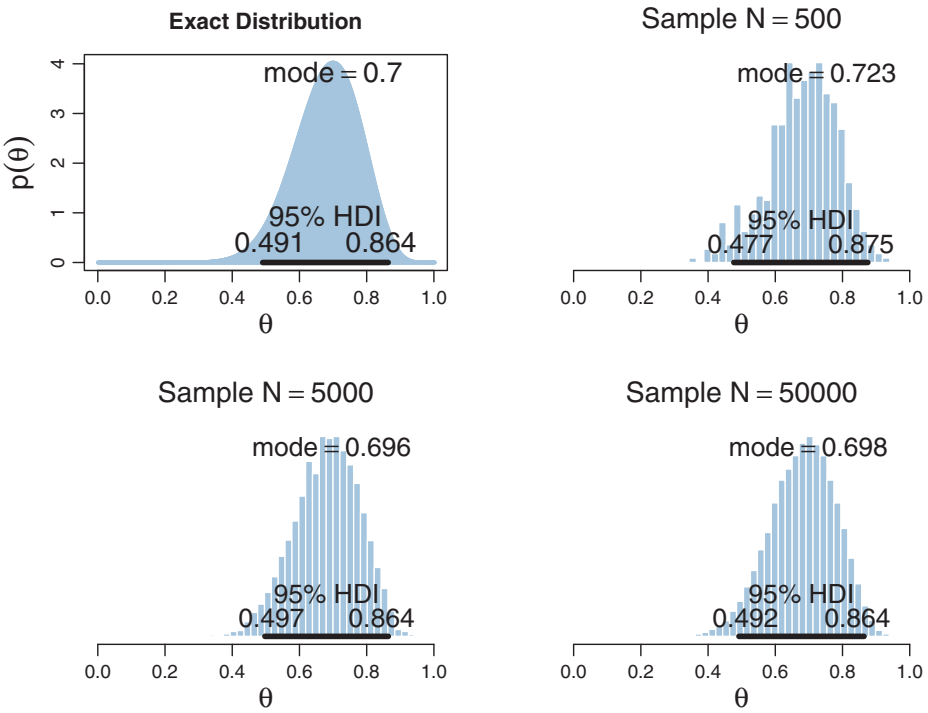

Figure 7.1 Large representative samples approximate the continuous distribution in the upper-left panel. The larger the sample, the more accurate the approximation. (This happens to be a beta(θ|15, 7) distribution.)

7.2. A SIMPLE CASE OF THE METROPOLIS ALGORITHM

Our 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. The question then becomes this: How can we sample a large number of representative values from a distribution? For an answer, let's ask a politician.

7.2.1. A politician stumbles upon the Metropolis algorithm

Suppose an elected politician lives on a long chain of islands. He is constantly traveling from island to island, wanting to stay in the public eye. At the end of a grueling day of photo opportunities and fundraising,[2] he has to decide whether to (i) stay on the current island, (ii) move to the adjacent island to the west, or (iii) move to the adjacent island to

[2] Maybe I shouldn't blithely make cynical jokes about politicians, because I believe that most elected representatives really do try to do some good for their constituencies. But saying so isn't as entertaining as the cheap joke.

the east. His goal is to visit all the islands proportionally to their relative population, so that he spends the most time on the most populated islands, and proportionally less time on the less populated islands. Unfortunately, he holds his office despite having no idea what the total population of the island chain is, and he doesn't even know exactly how many islands there are! His entourage of advisers is capable of some minimal information gathering abilities, however. When they are not busy fundraising, they can ask the mayor of the island they are on how many people are on the island. And, when the politician proposes to visit an adjacent island, they can ask the mayor of that adjacent island how many people are on that island.

The politician has a simple heuristic for deciding whether to travel to the proposed island: First, he flips a (fair) coin to decide whether to propose the adjacent island to the east or the adjacent island to the west. If the proposed island has a larger population than the current island, then he definitely goes to the proposed island. On the other hand, if the proposed island has a smaller population than the current island, then he goes to the proposed island only probabilistically, to the extent that the proposed island has a population as big as the current island. If the population of the proposed island is only half as big as the current island, the probability of going there is only 50%.

In more detail, denote the population of the proposed island as P_{proposed}, and the population of the current island as P_{current}. Then he moves to the less populated island with probability $p_{\text{move}} = P_{\text{proposed}} / P_{\text{current}}$. The politician does this by spinning a fair spinner marked on its circumference with uniform values from zero to one. If the pointed-to value is between zero and p_{move}, then he moves.

What's amazing about this heuristic is that it works: In the long run, the probability that the politician is on any one of the islands exactly matches the relative population of the island!

7.2.2. A random walk

Let's consider the island hopping heuristic in a bit more detail. Suppose that there are seven islands in the chain, with relative populations as shown in the bottom panel of Figure 7.2. The islands are indexed by the value θ, whereby the leftmost, western island is $\theta = 1$ and the rightmost, eastern island is $\theta = 7$. The relative populations of the islands increase linearly such that $P(\theta) = \theta$. Notice that uppercase $P(\,)$ refers to the *relative* population of the island, not its absolute population and not its probability mass. To complete your mental picture, you can imagine islands to the left of 1 and to the right of 7 that have populations of zero. The politician can propose to jump to those islands, but the proposal will always be rejected because the population is zero.

The middle panel of Figure 7.2 shows one possible trajectory taken by the politician. Each day corresponds to one-time increment, indicated on the vertical axis. The plot of the trajectory shows that on the first day ($t = 1$), the politician happens to start on the

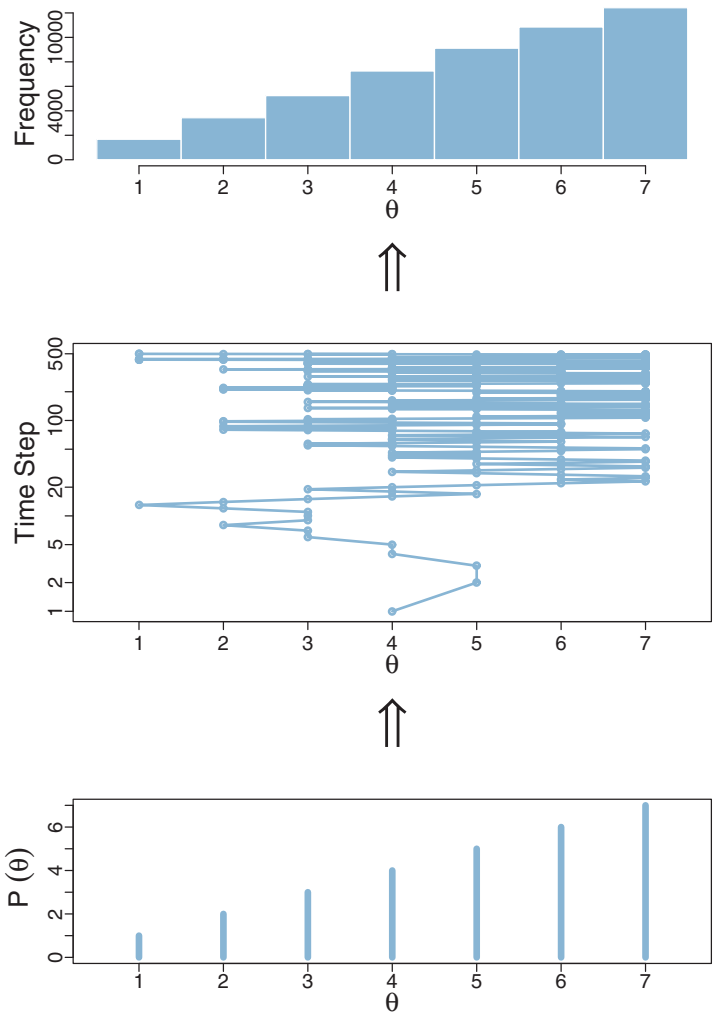

Figure 7.2 Illustration of a simple Metropolis algorithm. The bottom panel shows the values of the target distribution. The middle panel shows one random walk, at each time step proposing to move either one unit right or one unit left, and accepting the proposed move according the heuristic described in the main text. The top panel shows the frequency distribution of the positions in the walk.

middle island in the chain, hence $\theta_{\text{current}}=4$. To decide where to go on the second day, he flips a coin to propose moving either one position left or one position right. In this case, the coin proposed moving right, hence $\theta_{\text{proposed}}=5$. Because the relative population at the proposed position is greater than the relative population at the current position (i.e., $P(5) > P(4)$), the proposed move is accepted. The trajectory shows this move, because when $t=2$, then $\theta=5$.

Consider the next day, when $t = 2$ and $\theta = 5$. The coin flip proposes moving to the left. The probability of accepting this proposal is $p_{\text{move}} = P(\theta_{\text{proposed}})/P(\theta_{\text{current}}) = 4/5 = 0.80$. The politician then spins a fair spinner that has a circumference marked from 0 to 1, which happens to come up with a value greater than 0.80. Therefore, the politician rejects the proposed move and stays at the current island. Hence the trajectory shows that θ is still 5 when $t = 3$. The middle panel of Figure 7.2 shows the trajectory for the first 500 steps in this random walk across the islands. The scale of the time step is plotted logarithmically so you can see the details of the early steps but also see the trend of the later steps. There are many thousands of steps in the simulation.

The upper panel of Figure 7.2 shows a histogram of the frequencies with which each position is visited during this junket. Notice that the sampled relative frequencies closely mimic the actual relative populations in the bottom panel! In fact, a sequence generated this way will converge, as the sequence gets longer, to an arbitrarily close approximation of the actual relative probabilities.

7.2.3. General properties of a random walk

The trajectory shown in Figure 7.2 is just one possible sequence of positions when the movement heuristic is applied. 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, then acceptance of the proposed move is also random. Because of the randomness, if the process were started over again, then the specific trajectory would almost certainly be different. Regardless of the specific trajectory, in the long run the relative frequency of visits mimics the target distribution.

Figure 7.3 shows the probability of being in each position as a function of time. At time $t = 1$, the politician starts at $\theta = 4$. This starting position is indicated in the upper-left panel of Figure 7.3, labeled $t = 1$, by the fact that there is 100% probability of being at $\theta = 4$.

We want to derive the probability of ending up in each position at the next time step. To determine the probabilities of positions for time $t = 2$, consider the possibilities from the movement process. The process starts with the flip of a fair coin to decide which direction to propose moving. There is a 50% probability of proposing to move right, to $\theta = 5$. By inspecting the target distribution of relative probabilities in the lower-right panel of Figure 7.3, you can see that $P(\theta{=}5) > P(\theta{=}4)$, and therefore, a rightward move is always accepted whenever it is proposed. Thus, at time $t = 2$, there is a 50% probability of ending up at $\theta = 5$. The panel labeled $t = 2$ in Figure 7.3 plots this probability as a bar of height 0.5 at $\theta = 5$. The other 50% of the time, the proposed move is to the left, to $\theta = 3$. By inspecting the target distribution of relative probabilities in the lower-right panel of Figure 7.3, you can see that $P(\theta{=}3) = 3$, whereas $P(\theta{=}4) = 4$, and therefore, a leftward move is accepted only 3/4 of the times it is proposed. Hence, at time $t = 2$,

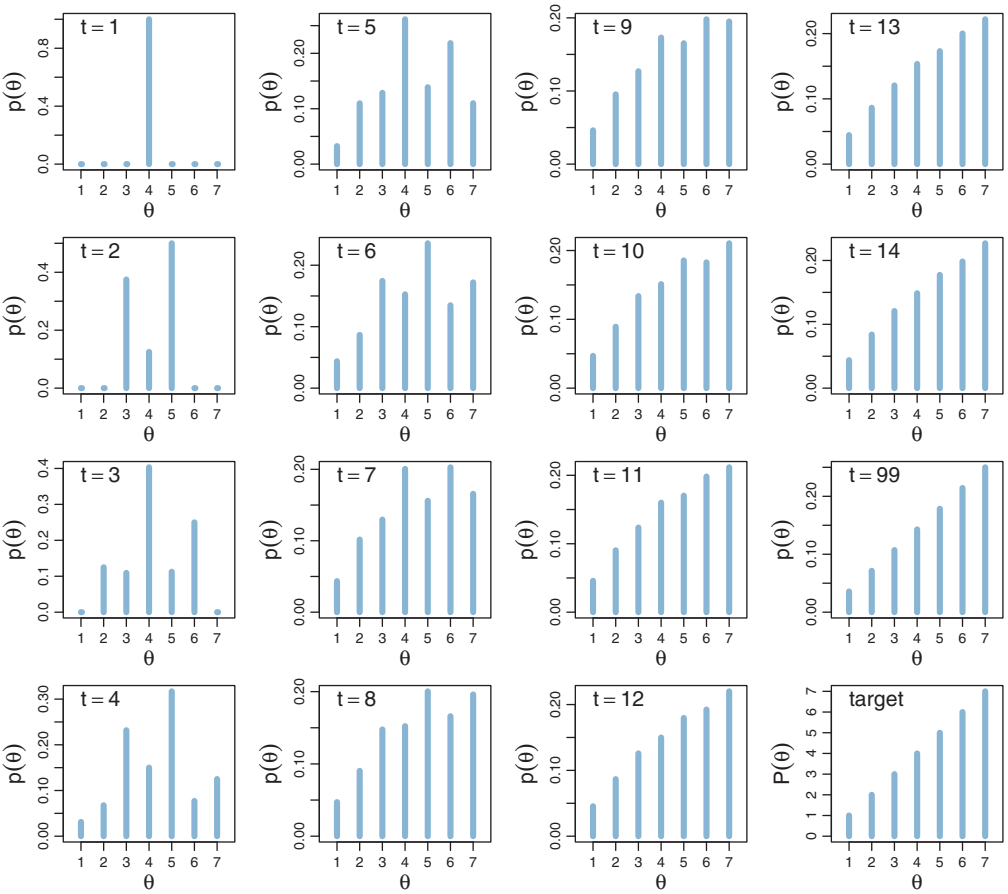

Figure 7.3 The probability of being at position θ, as a function of time t, when a simple Metropolis algorithm is applied to the target distribution in the lower-right panel. The time in each panel corresponds to the step in a random walk, an example of which is shown in Figure 7.2. The target distribution is shown in the lower-right panel.

the probability of ending up at $\theta = 3$ is $50\% \cdot 3/4 = 0.375$. The panel labeled $t = 2$ in Figure 7.3 shows this as a bar of height 0.375 at $\theta = 3$. Finally, if a leftward move is proposed but not accepted, we just stay at $\theta = 4$. The probability of this happening is only $50\% \cdot (1 - 3/4) = 0.125$.

This process repeats for the next time step. I won't go through the arithmetic details for each value of θ. But it is important to notice that after two proposed moves, i.e., when $t = 3$, the politician could be at any of the positions $\theta = 2$ through $\theta = 6$, but not yet at $\theta = 1$ or $\theta = 7$, because he could be at most two positions away from where he started.

The probabilities continue to be computed the same way at every time step. You can see that in the early time steps, the probability distribution is not a straight incline like

the target distribution. Instead, the probability distribution has a bulge over the starting position. As you can see in Figure 7.3, by time $t = 99$, the position probability is virtually indistinguishable from the target distribution, at least for this simple distribution. More complex distributions require a longer duration to achieve a good approximation to the target.

The graphs of Figure 7.3 show the *probability* that the moving politician is at each value of θ. But remember, at any given time step, the politician is at only one particular position, as shown in Figure 7.2. To approximate the target distribution, we let the politician meander around for many time steps while we keep track of where he has been. When we have a long record of where the traveler has been, we can approximate the target probability at each value of θ by simply counting the relative number times that the traveler visited that value.

Here is a summary of our algorithm for moving from one position to another. We are currently at position θ_{current}. We then propose to move one position right or one position left. The specific proposal is determined by flipping a coin, which can result in 50% heads (move right) or 50% tails (move left). The range of possible proposed moves, and the probability of proposing each, is called the *proposal distribution*. In the present algorithm, the proposal distribution is very simple: It has only two values with 50-50 probabilities.

Having proposed a move, we then decide whether or not to accept it. The acceptance decision is based on the value of the target distribution at the proposed position, relative to the value of the target distribution at our current position. Specifically, if the target distribution is greater at the proposed position than at our current position, then we definitely accept the proposed move: We always move higher if we can. On the other hand, if the target distribution is less at the proposed position than at our current position, we accept the move probabilistically: We move to the proposed position with probability $p_{\text{move}} = P(\theta_{\text{proposed}})/P(\theta_{\text{current}})$, where $P(\theta)$ is the value of the target distribution at θ. We can combine these two possibilities, of the target distribution being higher or lower at the proposed position than at our current position, into a single expression for the probability of moving to the proposed position:

$$p_{\text{move}} = \min\left(\frac{P(\theta_{\text{proposed}})}{P(\theta_{\text{current}})}, 1\right) \tag{7.1}$$

Notice that Equation 7.1 says that when $P(\theta_{\text{proposed}}) > P(\theta_{\text{current}})$, then $p_{\text{move}} = 1$. Notice also that the target distribution, $P(\theta)$, does not need to be normalized, which means it does not need to sum to 1 as a probability distribution must. This is because what matters for our choice is the *ratio*, $P(\theta_{\text{proposed}})/P(\theta_{\text{current}})$, not the absolute magnitude of $P(\theta)$. This property was used in the example of the island-hopping politician: The target distribution was the population of each island, not a normalized probability.

Having proposed a move by sampling from the proposal distribution and having then determined the *probability* of accepting the move according to Equation 7.1, we then actually accept or reject the proposed move by sampling a value from a uniform distribution over the interval [0, 1]. If the sampled value is between 0 and p_{move}, then we actually make the move. Otherwise, we reject the move and stay at our current position. The whole process repeats at the next time step.

7.2.4. Why we care

Notice what we must be able to do in the random-walk process:

- We must be able to generate a random value from the proposal distribution, to create θ_{proposed}.
- We must be able to evaluate the target distribution at any proposed position, to compute $P(\theta_{\text{proposed}})/P(\theta_{\text{current}})$.
- We must be able to generate a random value from a uniform distribution, to accept or reject the proposal according to p_{move}.

By being able to do those three things, we are able to do *indirectly* something we could not necessarily do directly: We can generate random samples from the target distribution. Moreover, we can generate those random samples from the target distribution even when the target distribution is not normalized.

This technique is profoundly useful when the target distribution $P(\theta)$ is a posterior proportional to $p(D|\theta)p(\theta)$. Merely by evaluating $p(D|\theta)p(\theta)$, without normalizing it by $p(D)$, we can generate random representative values from the posterior distribution. This result is wonderful because the method obviates direct computation of the evidence $p(D)$, which, as you'll recall, is one of the most difficult aspects of Bayesian inference. By using MCMC techniques, we can do Bayesian inference in rich and complex models. It has only been with the development of MCMC algorithms and software that Bayesian inference is applicable to complex data analysis, and it has only been with the production of fast and cheap computer hardware that Bayesian inference is accessible to a wide audience.

7.2.5. Why it works

In this section, I'll explain a bit of the mathematics behind why the algorithm works. Despite the mathematics presented here, however, *you will not need to use any of this mathematics for applied data analysis later in the book*. The math is presented here only to help you understand why MCMC works. Just as you can drive a car without being able to build an engine from scratch, you can apply MCMC methods without being able to program a Metropolis algorithm from scratch. But it can help you interpret the results of MCMC applications if you know a bit of what is going on "under the hood."

To get an intuition for why this algorithm works, consider two adjacent positions and the probabilities of moving from one to the other. We'll see that the relative transition probabilities, between adjacent positions, exactly match the relative values of the target distribution. Extrapolate that result across all the positions, and you can see that, in the long run, each position will be visited proportionally to its target value. Now the details: Suppose we are at position θ. The probability of moving to $\theta + 1$, denoted $p(\theta \rightarrow \theta + 1)$, is the probability of proposing that move times the probability of accepting it if proposed, which is $p(\theta \rightarrow \theta + 1) = 0.5 \cdot \min(P(\theta + 1)/P(\theta), 1)$. On the other hand, if we are presently at position $\theta + 1$, the probability of moving to θ is the probability of proposing that move times the probability of accepting it if proposed, which is $p(\theta + 1 \rightarrow \theta) = 0.5 \cdot \min(P(\theta)/P(\theta + 1), 1)$. The ratio of the transition probabilities is

$$\begin{aligned}
\frac{p(\theta \rightarrow \theta+1)}{p(\theta+1 \rightarrow \theta)} &= \frac{0.5 \min(P(\theta+1)/P(\theta), 1)}{0.5 \min(P(\theta)/P(\theta+1), 1)} \\
&= \begin{cases} \frac{1}{P(\theta)/P(\theta+1)} & \text{if } P(\theta+1) > P(\theta) \\ \frac{P(\theta+1)/P(\theta)}{1} & \text{if } P(\theta+1) < P(\theta) \end{cases} \\
&= \frac{P(\theta+1)}{P(\theta)} \qquad (7.2)
\end{aligned}$$

Equation 7.2 tells us that during transitions back and forth between adjacent positions, the relative probability of the transitions exactly matches the relative values of the target distribution. That might be enough for you to get the intuition that, in the long run, adjacent positions will be visited proportionally to their relative values in the target distribution. If that's true for adjacent positions, then, by extrapolating from one position to the next, it must be true for the whole range of positions.

To make that intuition more defensible, we have to fill in some more details. To do this, I'll use matrix arithmetic. This is the only place in the book where matrix arithmetic appears, so if the details here are unappealing, feel free to skip ahead to the next section (Section 7.3, p. 156). What you'll miss is an explanation of the mathematics underlying Figure 7.3, which depicts the key idea that the target distribution is *stable*: If the current probability of being in a position matches the target probabilities, then the Metropolis algorithm keeps it that way.

Consider the probability of transitioning from position θ to some other position. The proposal distribution, in the present simple scenario, considers only positions $\theta + 1$ and $\theta - 1$. The probability of moving to position $\theta - 1$ is the probability of proposing that position times the probability of accepting the move if it is proposed: $0.5 \min(P(\theta - 1)/P(\theta), 1)$. The probability of moving to position $\theta + 1$ is the probability of proposing that position times the probability of

accepting the move if it is proposed: $0.5 \min(P(\theta+1)/P(\theta), 1)$. The probability of staying at position θ is simply the complement of those two move-away probabilities: $0.5\,[1 - \min(P(\theta-1)/P(\theta), 1)] + 0.5\,[1 - \min(P(\theta+1)/P(\theta), 1)]$.

We can put those transition probabilities into a matrix. Each row of the matrix is a possible current position, and each column of the matrix is a candidate moved-to position. Below is a *sub*matrix from the full transition matrix T, showing rows $\theta - 2$ to $\theta + 2$, and columns $\theta - 2$ to $\theta + 2$:

$$\begin{bmatrix} \ddots & p(\theta-2 \to \theta-1) & 0 & 0 & 0 \\ \ddots & p(\theta-1 \to \theta-1) & p(\theta-1 \to \theta) & 0 & 0 \\ 0 & p(\theta \to \theta-1) & p(\theta \to \theta) & p(\theta \to \theta+1) & 0 \\ 0 & 0 & p(\theta+1 \to \theta) & p(\theta+1 \to \theta+1) & \ddots \\ 0 & 0 & 0 & p(\theta+2 \to \theta+1) & \ddots \end{bmatrix}$$

which equals

$$\begin{bmatrix} \ddots & 0.5\min\left(\frac{P(\theta-1)}{P(\theta-2)}, 1\right) & 0 & 0 & 0 \\ \ddots & \begin{array}{l} 0.5\left[1-\min\left(\frac{P(\theta-2)}{P(\theta-1)}, 1\right)\right] \\ +\,0.5\left[1-\min\left(\frac{P(\theta)}{P(\theta-1)}, 1\right)\right] \end{array} & 0.5\min\left(\frac{P(\theta)}{P(\theta-1)}, 1\right) & 0 & 0 \\ 0 & 0.5\min\left(\frac{P(\theta-1)}{P(\theta)}, 1\right) & \begin{array}{l} 0.5\left[1-\min\left(\frac{P(\theta-1)}{P(\theta)}, 1\right)\right] \\ +\,0.5\left[1-\min\left(\frac{P(\theta+1)}{P(\theta)}, 1\right)\right] \end{array} & 0.5\min\left(\frac{P(\theta+1)}{P(\theta)}, 1\right) & 0 \\ 0 & 0 & 0.5\min\left(\frac{P(\theta)}{P(\theta+1)}, 1\right) & \begin{array}{l} 0.5\left[1-\min\left(\frac{P(\theta)}{P(\theta+1)}, 1\right)\right] \\ +\,0.5\left[1-\min\left(\frac{P(\theta+2)}{P(\theta+1)}, 1\right)\right] \end{array} & \ddots \\ 0 & 0 & 0 & 0.5\min\left(\frac{P(\theta+1)}{P(\theta+2)}, 1\right) & \ddots \end{bmatrix} \tag{7.3}$$

The usefulness of putting the transition probabilities into a matrix is that we can then use matrix multiplication to get from any current location to the probability of the next locations. Here's a reminder of how matrix multiplication operates. Consider a matrix T. The value in its rth row and cth column is denoted T_{rc}. We can multiply the matrix on its *left* side by a *row* vector w, which yields another row vector. The cth component of the

product wT is $\sum_r w_r T_{rc}$. In other words, to compute the cth component of the result, take the row vector w and multiply its components by the corresponding components in the cth column of T, and sum up those component products.[3]

To use the transition matrix in Equation 7.3, we put the *current* location probabilities into a row vector, which I will denote w because it indicates *w*here we are. For example, if at the current time, we are definitely in location $\theta = 4$, then w has 1.0 in its $\theta = 4$ component, and zeros everywhere else. To determine the probability of the locations at the next time step, we simply multiply w by T. Here's a key example to think through: When $w = [\ldots, 0, 1, 0, \ldots]$ with a 1 only in the θ position, then wT is simply the row of T corresponding to θ, because the cth component of wT is $\sum_r w_r T_{rc} = T_{\theta c}$, where I'm using the subscript θ to stand for the index that corresponds to value θ.

Matrix multiplication is a very useful procedure for keeping track of position probabilities. At every time step, we just multiply the current position probability vector w by the transition probability matrix T to get the position probabilities for the next time step. We keep multiplying by T, over and over again, to derive the long-run position probabilities. This process is exactly what generated the graphs in Figure 7.3.

Here's the climactic implication: When the vector of position probabilities is the target distribution, it stays that way on the next time step! In other words, the position probabilities are stable at the target distribution. We can actually prove this result without much trouble. Suppose the current position probabilities are the target probabilities, i.e., $w = [\ldots, P(\theta-1), P(\theta), P(\theta+1), \ldots]/Z$, where $Z = \sum_\theta P(\theta)$ is the normalizer for the target distribution. Consider the θ component of wT. We will demonstrate that the θ component of wT is the same as the θ component of w, for any component θ. The θ component of wT is $\sum_r w_r T_{r\theta}$. Look back at the transition matrix in Equation 7.3, and you can see then that the θ component of wT is

$$\begin{aligned}\sum_r w_r T_{r\theta} = {} & P(\theta-1)/Z \cdot 0.5\min\left(\frac{P(\theta)}{P(\theta-1)}, 1\right) \\ & + P(\theta)/Z \cdot \left(0.5\left[1-\min\left(\frac{P(\theta-1)}{P(\theta)}, 1\right)\right] + 0.5\left[1-\min\left(\frac{P(\theta+1)}{P(\theta)}, 1\right)\right]\right) \\ & + P(\theta+1)/Z \cdot 0.5\min\left(\frac{P(\theta)}{P(\theta+1)}, 1\right) \qquad (7.4)\end{aligned}$$

To simplify that equation, we can consider separately the four cases: Case 1: $P(\theta) > P(\theta-1)$ and $P(\theta) > P(\theta+1)$; Case 2: $P(\theta) > P(\theta-1)$ and $P(\theta) < P(\theta+1)$; Case 3: $P(\theta) < P(\theta-1)$ and $P(\theta) > P(\theta+1)$; Case 4: $P(\theta) < P(\theta-1)$ and $P(\theta) < P(\theta+1)$. In each case, Equation 7.4 simplifies to $P(\theta)/Z$. For example, consider Case 1, when $P(\theta) > P(\theta-1)$ and $P(\theta) > P(\theta+1)$. Equation 7.4 becomes

[3] Although we don't do it here, we can also multiply a matrix on its *right* side by a *column* vector, which yields another column vector. For a column vector v, the rth component of Tv is $\sum_c T_{rc} v_c$.

$$
\begin{aligned}
\sum_r w_r T_{r\theta} &= P(\theta-1)/Z \cdot 0.5 \\
&\quad + P(\theta)/Z \cdot \left(0.5\left[1-\left(\frac{P(\theta-1)}{P(\theta)}\right)\right] + 0.5\left[1-\left(\frac{P(\theta+1)}{P(\theta)}\right)\right]\right) \\
&\quad + P(\theta+1)/Z \cdot 0.5 \\
&= 0.5\,P(\theta-1)/Z \\
&\quad + 0.5P(\theta)/Z - 0.5P(\theta)/Z\frac{P(\theta-1)}{P(\theta)} + 0.5P(\theta)/Z - 0.5P(\theta)/Z\frac{P(\theta+1)}{P(\theta)} \\
&\quad + 0.5\,P(\theta+1)/Z \\
&= P(\theta)/Z
\end{aligned}
$$

If you work through the other cases, you'll find that it always reduces to $P(\theta)/Z$. In conclusion, when the θ component starts at $P(\theta)/Z$, it stays at $P(\theta)/Z$.

We have shown that the target distribution is stable under the Metropolis algorithm, for our special case of island hopping. To prove that the Metropolis algorithm realizes the target distribution, we would need to show that the process actually gets us to the target distribution regardless of where we start. Here, I'll settle for intuition: You can see that no matter where you start, the distribution will naturally diffuse and explore other positions. Examples of this were shown in Figures 7.2 and 7.3. It's reasonable to think that the diffusion will settled into *some* stable state and we've just shown that the target distribution is *a* stable state. To make the argument complete, we'd have to show that there are no other stable states, and that the target distribution is actually an attractor into which other states flow, rather than a state that is stable if it is ever obtained but impossible to actually attain. This complete argument is far beyond what would be useful for the purposes of this book, but if you're interested you could take a look at the book by Robert and Casella (2004).

7.3. THE METROPOLIS ALGORITHM MORE GENERALLY

The procedure described in the previous section was just a special case of a more general procedure known as the Metropolis algorithm, named after the first author of a famous article (Metropolis, Rosenbluth, Rosenbluth, Teller, & Teller, 1953).[4] In the previous section, we considered the simple case of (i) discrete positions, (ii) on one dimension, and (iii) with moves that proposed just one position left or right. That simple situation made it relatively easy (believe it or not) to understand the procedure and how it works. The general algorithm applies to (i) continuous values, (ii) on any number of dimensions, and (iii) with more general proposal distributions.

[4] Nicholas Metropolis (1915–1999) was first author of the article, but history suggests that he had little to do with the invention of the algorithm itself, which might better be attributed to the second and third authors of the article, Marshall and Arianna Rosenbluth (Gubernatis, 2005).

The essentials of the general method are the same as for the simple case. First, we have some target distribution, $P(\theta)$, over a multidimensional continuous parameter space from which we would like to generate representative sample values. We must be able to compute the value of $P(\theta)$ for any candidate value of θ. The distribution, $P(\theta)$, does not have to be normalized, however. It merely needs to be nonnegative. In typical applications, $P(\theta)$ is the unnormalized posterior distribution on θ, which is to say, it is the product of the likelihood and the prior.

Sample values from the target distribution are generated by taking a random walk through the parameter space. The walk starts at some arbitrary point, specified by the user. The starting point should be someplace where $P(\theta)$ is nonzero. The random walk progresses at each time step by proposing a move to a new position in parameter space and then deciding whether or not to accept the proposed move. Proposal distributions can take on many different forms, with the goal being to use a proposal distribution that efficiently explores the regions of the parameter space where $P(\theta)$ has most of its mass. Of course, we must use a proposal distribution for which we have a quick way to generate random values! For our purposes, we will consider the generic case in which the proposal distribution is normal, centered at the current position. (Recall the discussion of the normal distribution back in Section 4.3.2.2, p. 83.) The idea behind using a normal distribution is that the proposed move will typically be near the current position, with the probability of proposing a more distant position dropping off according to the normal curve. Computer languages such as R have built-in functions for generating pseudorandom values from a normal distribution. For example, if we want to generate a proposed jump from a normal distribution that has a mean of zero and a standard deviation (SD) of 0.2, we could command R as follows: `proposedJump=rnorm(1, mean=0, sd=0.2)`, where the first argument, 1, indicates that we want a single random value, not a vector of many random values.

Having generated a proposed new position, the algorithm then decides whether or not to accept the proposal. The decision rule is exactly what was already specified in Equation 7.1. In detail, this is accomplished by computing the ratio $p_{\text{move}} = P(\theta_{\text{proposed}})/P(\theta_{\text{current}})$. Then a random number from the uniform interval [0, 1] is generated; in R, this can be accomplished with the command `runif(1)`. If the random number is between 0 and p_{move}, then the move is accepted. The process repeats and, in the long run, the positions visited by the random walk will closely approximate the target distribution.

7.3.1. Metropolis algorithm applied to Bernoulli likelihood and beta prior

In the scenario of the island-hopping politician, the islands represented candidate parameter values, and the relative populations represented relative posterior probabilities. In the scenario of coin flipping, the parameter θ has values that range on a continuum from zero to one, and the relative posterior probability is computed as likelihood times prior. To apply the Metropolis algorithm, we conceive of the parameter dimension

as a dense chain of infinitesimal islands, and we think of the (relative) population of each infinitesimal island as its (relative) posterior probability density. And, instead of the proposed jump being only to immediately adjacent islands, the proposed jump can be to islands farther away from the current island. We need a proposal distribution that will let us visit any parameter value on the continuum. For this purpose, we will use the familiar normal distribution (recall Figure 4.4, p. 83).

We will apply the Metropolis algorithm to the following familiar scenario. We flip a coin N times and observe z heads. We use a Bernoulli likelihood function, $p(z, N|\theta) = \theta^z (1-\theta)^{(N-z)}$. We start with a prior $p(\theta) = \text{beta}(\theta|a, b)$. For the proposed jump in the Metropolis algorithm, we will use a normal distribution centered at zero with standard deviation (SD) denoted as σ. We denote the proposed jump as $\Delta\theta \sim \text{normal}(\mu = 0, \sigma)$, where the symbol "$\sim$" means that the value is randomly sampled from the distribution (cf. Equation 2.2, p. 27). Thus, the proposed jump is usually close to the current position, because the mean jump is zero, but the proposed jump can be positive or negative, with larger magnitudes less likely than smaller magnitudes. Denote the current parameter value as θ_{cur} and the proposed parameter value as $\theta_{\text{pro}} = \theta_{\text{cur}} + \Delta\theta$.

The Metropolis algorithm then proceeds as follows. Start at an arbitrary initial value of θ (in the valid range). This is the current value, denoted θ_{cur}. Then:

1. Randomly generate a proposed jump, $\Delta\theta \sim \text{normal}(\mu = 0, \sigma)$ and denote the proposed value of the parameter as $\theta_{\text{pro}} = \theta_{\text{cur}} + \Delta\theta$.
2. Compute the probability of moving to the proposed value as Equation 7.1, specifically expressed here as

$$
\begin{aligned}
p_{\text{move}} &= \min\left(1, \frac{P(\theta_{\text{pro}})}{P(\theta_{\text{cur}})}\right) && \text{generic Metropolis form} \\
&= \min\left(1, \frac{p(D|\theta_{\text{pro}})p(\theta_{\text{pro}})}{p(D|\theta_{\text{cur}})p(\theta_{\text{cur}})}\right) && P \text{ is likelihood times prior} \\
&= \min\left(1, \frac{\text{Bernoulli}(z, N|\theta_{\text{pro}})\text{beta}(\theta_{\text{pro}}|a, b)}{\text{Bernoulli}(z, N|\theta_{\text{cur}})\text{beta}(\theta_{\text{cur}}|a, b)}\right) \\
& && \text{Bernoulli likelihood and beta prior} \\
&= \min\left(1, \frac{\theta_{\text{pro}}^z(1-\theta_{\text{pro}})^{(N-z)}\theta_{\text{pro}}^{(a-1)}(1-\theta_{\text{pro}})^{(b-1)}/B(a, b)}{\theta_{\text{cur}}^z(1-\theta_{\text{cur}})^{(N-z)}\theta_{\text{cur}}^{(a-1)}(1-\theta_{\text{cur}})^{(b-1)}/B(a, b)}\right) \\
& && \text{by Equations 6.2 and 6.3}
\end{aligned}
$$

   If the proposed value θ_{pro} happens to land outside the permissible bounds of θ, the prior and/or likelihood is set to zero, hence p_{move} is zero.
3. Accept the proposed parameter value if a random value sampled from a $[0, 1]$ uniform distribution is less than p_{move}, otherwise reject the proposed parameter value and tally the current value again.

Repeat the above steps until it is judged that a sufficiently representative sample has been generated. The judgment of "sufficiently representative" is not trivial and is an issue that will be discussed later in this chapter. For now, the goal is to understand the mechanics of the Metropolis algorithm.

Figure 7.4 shows specific examples of the Metropolis algorithm applied to a case with a beta($\theta|1, 1$) prior, $N = 20$, and $z = 14$. There are three columns in Figure 7.4, for three different runs of the Metropolis algorithm using three different values for σ in the proposal distribution. In all three cases, θ was arbitrarily started at 0.01, merely for purposes of illustration.

The middle column of Figure 7.4 uses a moderately sized SD for the proposal distribution, namely $\sigma = 0.2$, as indicated in the title of the upper-middle panel where it says "Prpsl.SD = 0.2." This means that at any step in the chain, for whatever value θ happens to be, the proposed jump is within ± 0.2 of θ about 68% of the time (because a normal distribution has about 68% of its mass between -1 and $+1$ SDs). In this case, the proposed jumps are accepted roughly half the time, as indicated in the center panel by the annotation $N_{acc}/N_{pro} = 0.495$, which is the number of accepted proposals divided by

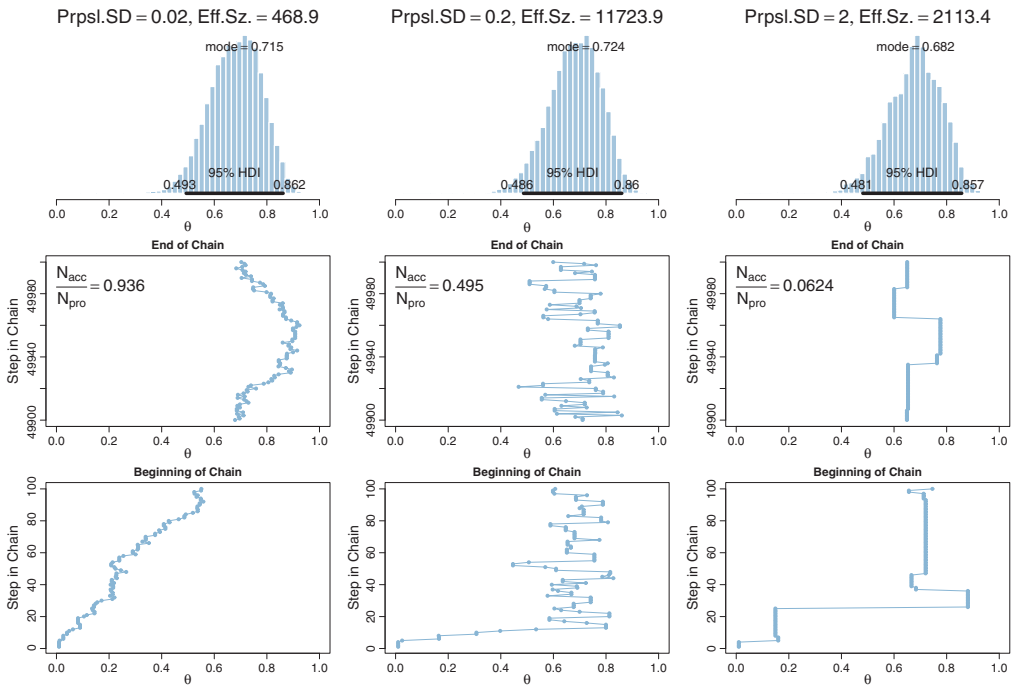

Figure 7.4 Metropolis algorithm applied to Bernoulli likelihood with beta($\theta|1, 1$) prior and $z = 14$ with $N = 20$. For each of the three columns, there are 50,000 steps in the chain, but for the left column, the proposal standard deviation (SD) is 0.02, for the middle column SD = 0.2, and for the right column SD = 2.0.

the total number of proposals in the chain. This setting of the proposal distribution allows the chain to move around parameter space fairly efficiently. In particular, you can see in the lower-middle panel that the chain moves away quickly from the unrepresentative starting position of $\theta = 0.01$. And, the chain visits a variety of representative values in relatively few steps. That is, the chain is not very clumpy. The upper-middle panel shows a histogram of the chain positions after 50,000 steps. The histogram looks smooth and is an accurate approximation of the true underlying posterior distribution, which we know in this case is a beta(θ|15, 7) distribution (cf. Figure 7.1, p. 146). Although the chain is not very clumpy and yields a smooth-looking histogram, it does have some clumpiness because each step is linked to the location of the previous step, and about half the steps don't change at all. Thus, there are not 50,000 independent representative values of the posterior distribution in the chain. If we take into account the clumpiness, then the so-called "effective size" of the chain is less, as indicated in the title of upper-middle panel where it says "Eff.Sz. = 11723.9." This is the equivalent number of values if they were sampled independently of each other. The technical meaning of effective size will be discussed later in the chapter.

The left column of Figure 7.4 uses a relatively small proposal SD, namely $\sigma = 0.02$, as indicated in the title of the upper-left panel where it says "Prpsl.SD = 0.02." You can see that successive steps in the chain make small moves because the proposed jumps are small. In particular, in the lower-left panel you can see that it takes many steps for the chain to move away from the unrepresentative starting position of $\theta = 0.01$. The chain explores values only very gradually, producing a snake-like chain that lingers around particular values, thereby producing a form of clumpiness. In the long run, the chain will explore the posterior distribution thoroughly and produce a good representation, but it will require a very long chain. The title of the upper-left panel indicates that the effective size of this 50,000 step chain is only 468.9.

The right column of Figure 7.4 uses a relatively large proposal SD, namely $\sigma = 2$, as indicated in the title of the upper-left panel where it says "Prpsl.SD = 2." The proposed jumps are often far away from the bulk of the posterior distribution, and therefore, the proposals are often rejected and the chain stays at one value for many steps. The process accepts new values only occasionally, producing a very clumpy chain. In the long run, the chain will explore the posterior distribution thoroughly and produce a good representation, but it will require a very long chain. The title of the upper-right panel indicates that the effective size of this 50,000 step chain is only 2113.4.

Regardless of which proposal distribution in Figure 7.4 is used, the Metropolis algorithm will eventually produce an accurate representation of the posterior distribution, as is suggested by the histograms in the upper row of Figure 7.4. What differs is the efficiency of achieving a good approximation. The moderate proposal distribution will achieve a good approximation in fewer steps than either of the extreme proposal

distributions. Later in the chapter, we will discuss criteria for deciding that a chain has produced a sufficiently good approximation, but for now suppose that our goal is to achieve an effective size of 10,000. The proposal distribution in the middle column of Figure 7.4 has achieved this goal. For the proposal distribution in the left column of Figure 7.4, we would need to run the chain more than 20 times longer because the effective size is less than 1/20th of the desired size. Sophisticated implementations of the Metropolis algorithm have an automatic preliminary phase that adjusts the width of the proposal distribution so that the chain moves relatively efficiently. A typical way to do this is to adjust the proposal distribution so that the acceptance ratio is a middling value such as 0.5. The steps in the adaptive phase are not used to represent the posterior distribution.

7.3.2. Summary of Metropolis algorithm

In this section, I recapitulate the main ideas of the Metropolis algorithm and explicitly point out the analogy between the island-hopping politician and the θ-hopping coin bias.

The motivation for methods like the Metropolis algorithm is that they provide a high-resolution picture of the posterior distribution, even though in complex models we cannot explicitly solve the mathematical integral in Bayes' rule. The 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 our approximation. As emphasized previously, this is a sample of representative credible parameter values from the posterior distribution; it is not a resampling of data (there is a fixed data set).

The cleverness of the method is that 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. All we need to decide whether to accept a proposed parameter value is the mathematical formulas for the likelihood function and prior distribution, and these formulas can be directly evaluated from their definitions.

We have seen two examples of the Metropolis algorithm in action. One was the island-hopping politician in Figure 7.2. The other was the θ-hopping coin bias in Figure 7.4. The example of the island-hopping politician was presented to demonstrate the Metropolis algorithm in its most simple form. The application involved only a finite space of discrete parameter values (i.e., the islands), the simplest possible proposal distribution (i.e., a single step right or left), and a target distribution that was directly evaluated (i.e., the relative population of the island) without any reference to mathematical formulas for likelihood functions and prior distributions. The simple forms of the discrete space and proposal distribution also allowed us to explore some of the basic mathematics of transition probabilities, to get some sense of why the Metropolis algorithm works.

The example of the θ-hopping coin bias in Figure 7.4 used the Metropolis algorithm in a more realistic setting. Instead of a finite space of discrete parameter values, there was a continuum of possible parameter values across the interval from zero to one. This is like an infinite string of adjacent infinitesimal islands. Instead of a proposal distribution that could only go a single step left or right, the normal proposal distribution could jump anywhere on the continuum, but it favored nearby values as governed by the SD of its bell shape. And, instead of a simple "relative population" at each parameter value, the target distribution was the relative density of the posterior distribution, computed by evaluating the likelihood function times the prior density.

7.4. TOWARD GIBBS SAMPLING: ESTIMATING TWO COIN BIASES

The Metropolis method is very useful, but it can be inefficient. Other methods can be more efficient in some situations. In particular, another type of sampling method that can be very efficient is *Gibbs sampling*. Gibbs sampling typically applies to models with multiple parameters, and therefore, we need to introduce an example in which there is more than one parameter. A natural extension of previous examples, which involved estimating the bias of a single coin, is estimating the biases of two coins.

In many real-world situations we observe two proportions, which differ by some amount in the specific random samples, but for which we want to infer what underlying difference is credible for the broader population from which the sample came. After all, the observed flips are just a noisy hint about the actual underlying biases. For example, suppose we have a sample of 97 people suffering from a disease. We give a random subset of them a promising drug, and we give the others a placebo. After 1 week, 12 of the 51 drug-treated people have gotten better, and 5 of the 46 placebo-treated people have gotten better. Did the drug actually work better than the placebo, and by how much? In other words, based on the observed difference in proportions, 12/51 versus 5/46, what underlying differences are actually credible? As another example, suppose you want to find out if mood affects cognitive performance. You manipulate mood by having 83 people sit through mood-inducing movies. A random subset of your participants is shown a bittersweet film about lovers separated by circumstances of war but who never forget each other. The other participants are shown a light comedy about high school pranks. Immediately after seeing the film, all participants are given some cognitive tasks, including an arithmetic problem involving long division. Of the 42 people who saw the war movie, 32 correctly solved the long division problem. Of the 41 people who saw the comedy, 27 correctly solved the long division problem. Did the induced mood actually affect cognitive performance? In other words, based on the observed difference in proportions, 32/42 versus 27/41, what underlying differences are actually credible?

To discuss the problem more generally and with mathematical precision, we need to define some notation. To make the notation generic, we will talk about heads and tails of flips of coins, instead of outcomes of participants in groups. What was called a group is now called a coin, and what was the outcome of a participant is now called the outcome of a flip. We'll use the same sort of notation that we've been using previously, but with subscripts to indicate which of the two coins is being referred to. Thus, the hypothesized bias for heads in coin j ,where $j = 1$ or $j = 2$, is denoted θ_j. The actual number of heads observed in a sample of N_j flips is z_j, and the ith individual flip in coin j is denoted y_{ji}.

We assume that the data from the two coins are independent: The performance of one coin has no influence on the performance of the other. Typically, we design research to make sure that the assumption of independence holds. In the examples above, we assumed independence in the disease-treatment scenario because we assumed that social interaction among the patients was minimal. We assumed independence in the mood-induction experiment because the experiment was designed to enforce zero social interaction between participants after the movie. The assumption of independence is crucial for all the mathematical analyses we will perform. If you have a situation in which the groups are not independent, the methods of this section do not directly apply. In situations where there are dependencies in the data, a model can try to formally express the dependency, but we will not be pursuing those situations here.

7.4.1. Prior, likelihood and posterior for two biases

We are considering situations in which there are *two* underlying biases, namely θ_1 and θ_2, for the two coins. We are trying to determine what we should believe about these biases after we have observed some data from the two coins. Recall that I am using the term "bias" as the name of the parameter θ, and *not* to indicate that the value of θ deviates from 0.5. The colloquial meaning of bias, as unfair, might also sneak into my writing from time to time. Thus, a colloquially "unbiased" coin technically "has a bias of 0.5."

To estimate the biases of the coins in a Bayesian framework, we must first specify what we believe about the biases without the data. Our prior beliefs are about *combinations* of parameter values. To specify a prior belief, we must specify the credibility, $p(\theta_1, \theta_2)$, for every combination θ_1, θ_2. If we were to make a graph of $p(\theta_1, \theta_2)$, it would be three dimensional, with θ_1 and θ_2 on the two horizontal axes, and $p(\theta_1, \theta_2)$ on the vertical axis. Because the credibilities form a probability density function, their integral across the parameter space must be one: $\int\int \mathrm{d}\theta_1 \mathrm{d}\theta_2 p(\theta_1, \theta_2) = 1$.

For simplicity in these examples, we will assume that our beliefs about θ_1 are independent of our beliefs about θ_2. For example, if I have a coin from Canada and a coin from India, my belief about bias in the coin from Canada could be completely separate from my belief about bias in the coin from India. Independence of attributes was discussed in Section 4.4.2, p. 92. Mathematically, independence means that $p(\theta_1, \theta_2) = p(\theta_1)p(\theta_2)$ for every value of θ_1 and θ_2, where $p(\theta_1)$ and $p(\theta_2)$ are the

marginal belief distributions. When beliefs about two parameters are independent, the mathematical manipulations can be greatly simplified. Beliefs about the two parameters do not need to be independent, however. For example, perhaps I believe that coins are minted in similar ways across countries, and so if a Canadian coin is biased (i.e., has a θ value different than 0.5), then an Indian coin should be similarly biased. At the extreme, if I believe that θ_1 always exactly equals θ_2, then the two parameter values are completely dependent upon each other. Dependence does not imply direct causal relationship, it merely implies that knowing the value of one parameter constrains beliefs about the value of the other parameter.

Along with the prior beliefs, we have some observed data. We assume that the flips within a coin are independent of each other and that the flips across coins are independent of each other. The veracity of this assumption depends on exactly how the observations are actually made, but, in properly designed experiments, we have reasons to trust this assumption. Notice that we will always assume independence of *data* within and across groups, regardless of whether we assume independence in our *beliefs* about the biases in the groups. Formally, the assumption of independence in the data means the following. Denote the result of a flip of coin 1 as y_1, where the result can be $y_1 = 1$ (heads) or $y_1 = 0$ (tails). Similarly, the result of a flip of coin 2 is denoted y_2. Independence of the data across the two coins means that the data from coin 1 depend only on the bias in coin 1, and the data from coin 2 depend only on the bias in coin 2, which can be expressed formally as $p(y_1|\theta_1, \theta_2) = p(y_1|\theta_1)$ and $p(y_2|\theta_1, \theta_2) = p(y_2|\theta_2)$.

From one coin we observe the data D_1 consisting of z_1 heads out of N_1 flips, and from the other coin we observe the data D_2 consisting of z_2 heads out of N_2 flips. In other words, $z_1 = \sum_{i=1}^{N_1} y_{1i}$, where y_{1i} denotes the ith flip of the first coin. Notice that $z_1 \in \{0, \ldots, N_1\}$ and $z_2 \in \{0, \ldots, N_2\}$. We denote the whole set of data as $D = \{z_1, N_1, z_2, N_2\}$. Because of independence of sampled flips, the probability of D is the product of the Bernoulli distribution functions for the individual flips:

$$\begin{aligned} p(D|\theta_1, \theta_2) &= \prod_{y_{1i} \in D_1} p(y_{1i}|\theta_1, \theta_2) \prod_{y_{2j} \in D_2} p(y_{2j}|\theta_1, \theta_2) \\ &= \prod_{y_{1i} \in D_1} \theta_1^{y_{1i}} (1-\theta_1)^{(1-y_{1i})} \prod_{y_{2j} \in D_2} \theta_2^{y_{2j}} (1-\theta_2)^{(1-y_{2j})} \\ &= \theta_1^{z_1} (1-\theta_1)^{(N_1-z_1)} \, \theta_2^{z_2} (1-\theta_2)^{(N_2-z_2)} \end{aligned} \tag{7.5}$$

The posterior distribution of our beliefs about the underlying bias is derived in the usual way by applying Bayes' rule, but now the functions involve two parameters:

$$\begin{aligned} p(\theta_1, \theta_2|D) &= p(D|\theta_1, \theta_2) p(\theta_1, \theta_2) \,/\, p(D) \\ &= p(D|\theta_1, \theta_2) p(\theta_1, \theta_2) \Big/ \iint \mathrm{d}\theta_1 \mathrm{d}\theta_2 p(D|\theta_1, \theta_2) p(\theta_1, \theta_2) \end{aligned} \tag{7.6}$$

Remember, as always in the expression of Bayes' rule, the θ_j's in left side of the equation and in the numerator of the right side are referring to specific values of θ_j, but the θ_j's in the integral in the denominator range over all possible values of θ_j.

What has just been described in the previous few paragraphs is the general Bayesian framework for making inferences about two biases when the likelihood function consists of independent Bernoulli distributions. What we have to do next is specify a particular mathematical form for the prior distribution. We will work through the mathematics of a particular case for two reasons: First, it will allow us to explore graphical displays of two-dimensional parameter spaces, which will inform our intuitions about Bayes' rule and sampling from the posterior distribution. Second, the mathematics will set the stage for a specific example of Gibbs sampling. Later in the book when we do applied Bayesian analysis, we will *not* be doing any of this sort of mathematics. We are doing the math now, for simple cases, to understand how the methods work so we can properly interpret their outputs in realistically complex cases.

7.4.2. The posterior via exact formal analysis

Suppose we want to pursue a solution to Bayes' rule, in Equation 7.6 above, using formal analysis. What sort of prior probability function would make the analysis especially tractable? Perhaps you can guess the answer by recalling the discussion of Chapter 6. We learned there that the beta distribution is conjugate to the Bernoulli likelihood function. This suggests that a product of beta distributions would be conjugate to a product of Bernoulli functions.

This suggestion turns out to be true. We pursue the same logic as was used for Equation 6.8 (p. 132). First, recall that a beta distribution has the form $\text{beta}(\theta|a,b) = \theta^{(a-1)}(1-\theta)^{(b-1)}/B(a,b)$, where $B(a,b)$ is the beta normalizing function, which by definition is $B(a,b) = \int_0^1 \mathrm{d}\theta\, \theta^{(a-1)}(1-\theta)^{(b-1)}$. We assume a $\text{beta}(\theta_1|a_1,b_1)$ prior on θ_1, and an independent $\text{beta}(\theta_2|a_2,b_2)$ prior on θ_2, so that $p(\theta_1,\theta_2) = \text{beta}(\theta_1|a_1,b_1) \cdot \text{beta}(\theta_2|a_2,b_2)$. Then:

$$
\begin{aligned}
p(\theta_1,\theta_2|D) &= p(D|\theta_1,\theta_2)p(\theta_1,\theta_2)/p(D) && \text{general form of Bayes' rule} \\
&= \theta_1^{z_1}(1-\theta_1)^{(N_1-z_1)}\,\theta_2^{z_2}(1-\theta_2)^{(N_2-z_2)}p(\theta_1,\theta_2)/p(D) \\
& && \text{Bernoulli likelihood from Equation 7.5} \\
&= \frac{\theta_1^{z_1}(1-\theta_1)^{(N_1-z_1)}\,\theta_2^{z_2}(1-\theta_2)^{(N_2-z_2)}\,\theta_1^{(a_1-1)}(1-\theta_1)^{(b_1-1)}\,\theta_2^{(a_2-1)}(1-\theta_2)^{(b_2-1)}}{p(D)B(a_1,b_1)B(a_2,b_2)} \\
& && \text{independent beta prior} \\
&= \frac{\theta_1^{(z_1+a_1-1)}(1-\theta_1)^{(N_1-z_1+b_1-1)}\theta_2^{(z_2+a_2-1)}(1-\theta_2)^{(N_2-z_2+b_2-1)}}{p(D)B(a_1,b_1)B(a_2,b_2)} && (7.7)
\end{aligned}
$$

We know that the left side of Equation 7.7 must be a probability density function, and we see that the numerator of the right side has the form of a product of beta distributions, namely $\text{beta}(\theta_1|z_1+a_1, N_1-z_1+b_1)$ times $\text{beta}(\theta_2|z_2+a_2, N_2-z_2+b_2)$. Therefore, the denominator of Equation 7.7 must be the corresponding normalizer for the product of beta distributions:[5]

$$p(D)B(a_1,b_1)B(a_2,b_2) = B(z_1+a_1, N_1-z_1+b_1)\, B(z_2+a_2, N_2-z_2+b_2) \tag{7.9}$$

Recapitulation: When the prior is a product of independent beta distributions, the posterior is also a product of independent beta distributions, with each beta obeying the update rule we derived in Chapter 6. Explicitly, if the prior is $\text{beta}(\theta_1|a_1,b_1) \cdot \text{beta}(\theta_2|a_2,b_2)$, and the data are z_1, N_1, z_2, N_2, then the posterior is $\text{beta}(\theta_1|z_1+a_1, N_1-z_1+b_1) \cdot \text{beta}(\theta_2|z_2+a_2, N_2-z_2+b_2)$.

One way of understanding the posterior distribution is to visualize it. We want to plot the probability densities as a function of *two* parameters, θ_1 and θ_2. One way to do this is by placing the two parameters, θ_1 and θ_2, on two horizontal axes, and placing the probability density, $p(\theta_1,\theta_2)$, on a vertical axis. The surface can then be displayed in a picture as if it were a landscape viewed in perspective. Alternatively, we can place the two parameter axes flat on the drawing plane and indicate the probability density with level contours, such that any one contour marks points that have the same specific level. An example of these plots is described next.

Figure 7.5 shows graphs for an example of Bayesian updating from Equation 7.6. In this example, the prior begins with mild beliefs that each bias is about 0.50, using a $\text{beta}(\theta|2,2)$ distribution for both biases. The upper panels show a perspective plot and a contour plot for the prior distribution. Notice that it is gently peaked at the center of the parameter space, which reflects the prior belief that the two biases are around 0.5, but without much certainty. The perspective plot shows vertical slices of the prior density parallel to θ_1 and θ_2. Consider slices parallel to the θ_1 axis, with different slices at different values of θ_2. The profile of the density on every slice has the same shape, with merely different heights. In particular, at every level of θ_2, the profile of the slice along θ_1 is a $\text{beta}(\theta_1|2,2)$ shape, with merely an overall height that depends on the level of θ_2. When the shape of the profile on the slices does not change, as exemplified here, then the joint distribution is constructed from the product of independent marginal distributions.

[5] By rearranging terms of Equation 7.9, a convenient consequence is that

$$p(D) = \frac{B(z_1+a_1, N_1-z_1+b_1)\, B(z_2+a_2, N_2-z_2+b_2)}{B(a_1,b_1)B(a_2,b_2)} \tag{7.8}$$

This is analogous to the result we found previously for one parameter, in Equation 6.8 and Footnote 5 on p. 132.

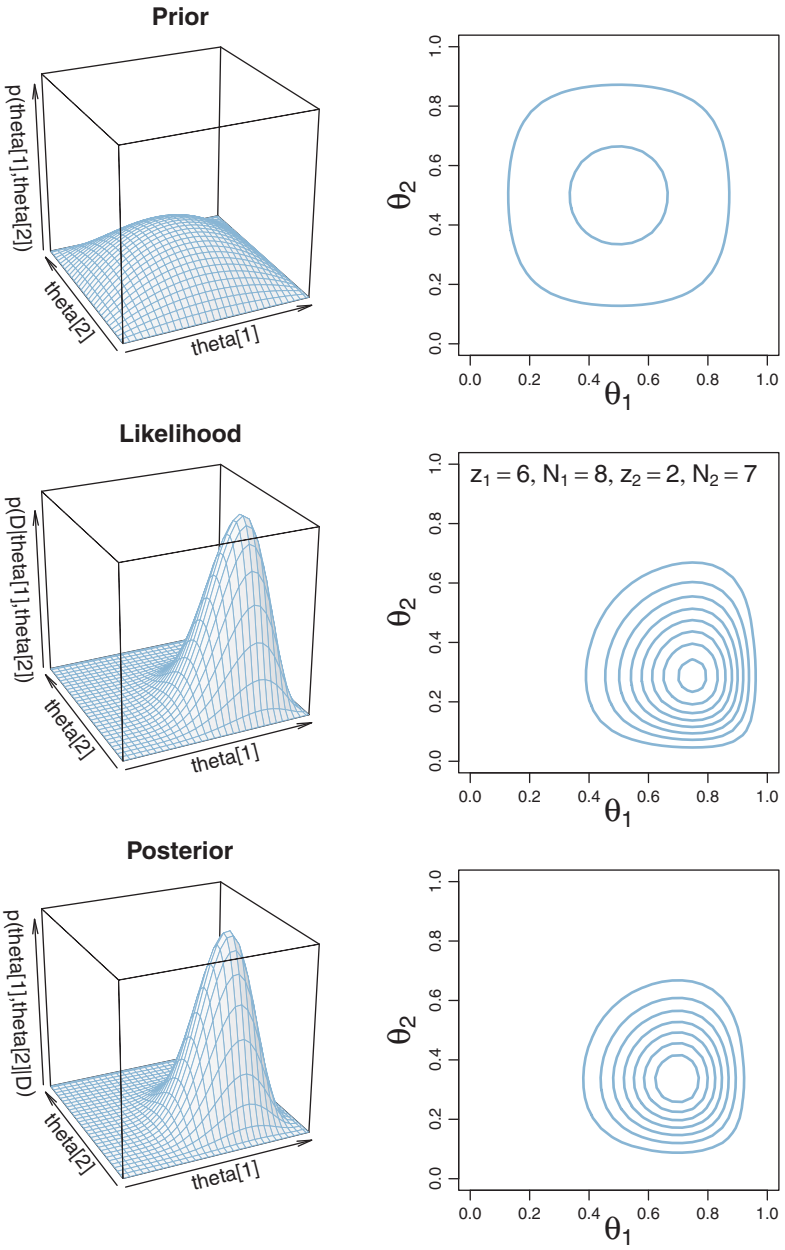

Figure 7.5 Bayesian updating of independent beta$(\theta|2,2)$ priors with the data annotated in the middle-right panel. Left panels show perspective surface plots; right panels show contour plots of the same distributions.

The contour plot in the upper-right panel of Figure 7.5 shows the same distribution as the upper-left panel. Instead of showing vertical slices through the distribution, the contour plot shows horizontal slices. Each contour corresponds to a slice at a particular height. Contour plots can be challenging to interpret because it is not immediately obvious what the heights of the contours are, or even whether two adjacent contours belong to different heights. Contours can be labeled with numerical values to indicate their heights, but then the plot can become very cluttered. Therefore, if the goal is a quick intuition about the general layout of the distribution, then a perspective plot is preferred over a contour plot. If the goal is instead a more detailed visual determination of the parameter values for a particular peak in the distribution, then a contour plot may be preferred.

The middle row of Figure 7.5 shows the likelihood functions for the specific data displayed in the panels. The plots show the likelihood for each possible combination of θ_1 and θ_2. Notice that the likelihood is maximized at θ values that match the proportions of heads in the data.

The resulting posterior distribution is shown in the lowest row of Figure 7.5. At each point in the θ_1, θ_2 parameter space, the posterior is the product of the prior and likelihood values at that point, divided by the normalizer, $p(D)$.

Summary: This section provided a graphical display of a prior, likelihood, and posterior on a two-parameter space, in Figure 7.5. In this section, we emphasized the use of mathematical forms, with priors that are conjugate to the likelihood. The particular mathematical form, involving beta distributions, will be used again in a subsequent section that introduces Gibbs sampling, and that is another motivation for including the mathematical formulation of this section. Before getting to Gibbs sampling, however, we first apply the Metropolis algorithm to this two-parameter situation. We will see later that Gibbs sampling generates a posterior sample more efficiently than the Metropolis algorithm.

7.4.3. The posterior via the Metropolis algorithm

Although we have already solved the present example with exact mathematical analysis, we will apply the Metropolis algorithm to expand our understanding of the algorithm for a two-parameter case and subsequently to compare the Metropolis algorithm with Gibbs sampling. Recall that the Metropolis algorithm is a random walk through the parameter space that starts at some arbitrary point. We propose a jump to a new point in parameter space, with the proposed jump randomly generated from a proposal distribution from which it is easy to generate values. For our present purposes, the proposal distribution is a *bivariate* normal. You can visualize a bivariate normal distribution by imagining a one-dimensional normal (as in Figure 4.4, p. 83), sticking a pin down vertically through its peak, and spinning it around the pin, to form a bell-shaped surface. The use of a bivariate

normal proposal distribution implies that the proposed jumps will usually be near the current position. Notice that proposed jumps can be in any direction in parameter space relative to the current position.[6] The proposed jump is definitely accepted if the posterior is taller (more dense) at the proposed position than at the current position, and the proposed jump is probabilistically accepted if the posterior is shorter (less dense) at the proposed position than at the current position. If the proposed jump is rejected, the current position is counted again. Notice that a position in parameter space represents a *combination* of jointly credible parameter values, $\langle \theta_1, \theta_2 \rangle$.

Figure 7.6 shows the Metropolis algorithm applied to the case of Figure 7.5 (p. 167), so that you can directly compare the results of the Metropolis algorithm with the results of formal analysis and grid approximation. By comparing with the contour plot in the lower-right panel of Figure 7.5 (p. 167), you can see that the points do indeed appear to explore the bulk of the posterior distribution. The sampled points in Figure 7.6 are connected by lines so that you can get a sense of the trajectory taken by the random

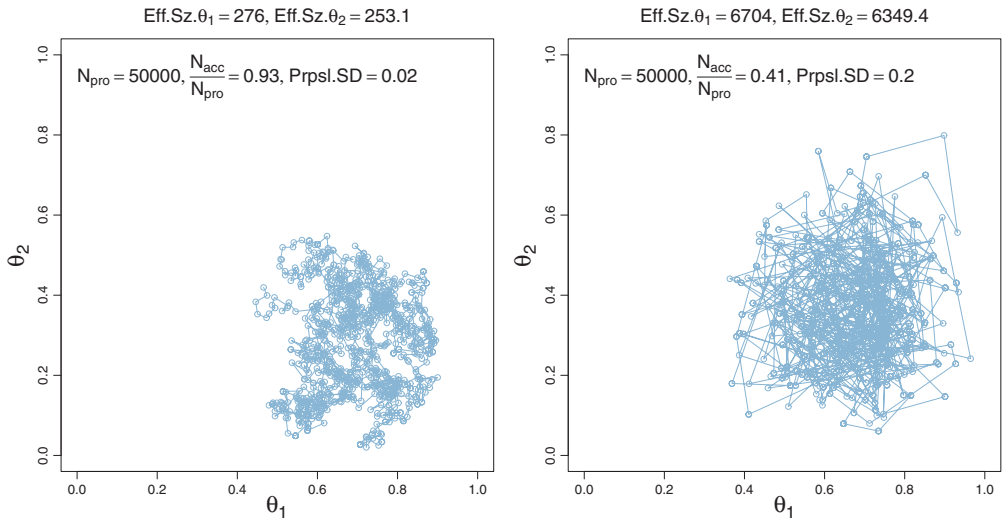

Figure 7.6 Metropolis algorithm applied to the prior and likelihood shown in Figure 7.5, p. 167. The left panel shows a chain with a narrow proposal distribution and the right panel shows a chain with a moderate-width proposal distribution, as indicated by annotation "Prpsl.SD" in each panel. N_{pro} is the number of proposed jumps, and N_{acc} is the number of accepted proposals. *Many of the plotted points have multiple superimposed symbols where the chain lingered during rejected proposals.* Notice that the effective size of the chain, indicated at the top of the plot, is far less than the length of the chain (N_{pro}). Only 1,000 of the N_{pro} steps are displayed here.

[6] The proposal distribution does not have to be a rotationally symmetric bivariate normal. For example, it could be a bivariate normal with nonzero covariances, so that proposals are more likely to be made in some diagonal directions more than others. The proposal distribution could even be nonnormal. It is only for the present illustrative purposes that we assume a simple symmetric proposal distribution.

walk. But the ultimate estimates regarding the posterior distribution do not care about the sequence in which the sampled points appeared, and the trajectory is irrelevant to anything but your understanding of the Metropolis algorithm.

The two panels of Figure 7.6 show results from using two different widths for the proposal distribution. The left panel shows results from a relatively narrow proposal distribution, that had a standard deviation of only 0.02. You can see that there were only tiny changes from one step to the next. Visually, the random walk looks like a clumpy strand that gradually winds through the parameter space. Because the proposed jumps were so small, the proposals were usually accepted, but each jump yielded relatively little new information and consequently the effective sample size (ESS) of the chain is very small, as annotated at the top of the panel.

The right panel of Figure 7.6 shows results from using a moderate width for the proposal distribution, with a SD of 0.2. The jumps from one position to the next are larger than the previous example, and the random walk explores the posterior distribution more efficiently than the previous example. The ESS of the chain is much larger than before. But the ESS of the chain is still far less than the number of proposed jumps, because many of the proposed jumps were rejected, and even when accepted, the jumps tended to be near the previous step in the chain.

In the limit of infinite random walks, the Metropolis algorithm yields arbitrarily accurate representations of the underlying posterior distribution. The left and right panels of Figure 7.6 would eventually converge to an identical and highly accurate approximation to the posterior distribution. But in the real world of finite random walks, we care about how efficiently the algorithm generates an accurate representative sample. We prefer to use the proposal distribution from the right panel of Figure 7.6 because it will, typically, produce a more accurate approximation of the posterior than the proposal distribution from left panel, for the same number of proposed jumps.

7.4.4. Gibbs sampling

The Metropolis algorithm is very general and broadly applicable. One problem with it, however, is that the proposal distribution must be properly tuned to the posterior distribution if the algorithm is to work well. If the proposal distribution is too narrow or too broad, a large proportion of proposed jumps will be rejected and the trajectory will get bogged down in a localized region of the parameter space. Even at its most efficient, the effective size of the chain is far less than the number of proposed jumps. It would be nice, therefore, if we had another method of sample generation that was more efficient. Gibbs sampling is one such method.

Gibbs sampling was introduced by Geman and Geman (1984), who were studying how computer vision systems could infer properties of an image from its pixelated camera input. The method is named after the physicist Josiah Willard Gibbs (1839–1903), who

studied statistical mechanics and thermodynamics. Gibbs sampling can be especially useful for hierarchical models, which we explore in Chapter 9. It turns out that Gibbs sampling is a special case of the Metropolis-Hastings algorithm, which is a generalized form of the Metropolis algorithm, as will be briefly discussed later. An influential article by Gelfand and Smith (1990) introduced Gibbs sampling to the statistical community; a brief review is provided by Gelfand (2000), and interesting details of the history of MCMC can be found in the book by McGrayne (2011).

The procedure for Gibbs sampling is a type of random walk through parameter space, like the Metropolis algorithm. The walk starts at some arbitrary point, and at each point in the walk, the next step depends only on the current position, and on no previous positions. What is different about Gibbs sampling, relative to the Metropolis algorithm, is how each step is taken. At each point in the walk, one of the component parameters is selected. The component parameter could be selected at random, but typically the parameters are cycled through, in order: $\theta_1, \theta_2, \theta_3, \ldots, \theta_1, \theta_2, \theta_3, \ldots$. The reason that parameters are cycled rather than selected randomly is that for complex models with many dozens or hundreds of parameters, it would take too many steps to visit every parameter by random chance alone, even though they would be visited about equally often in the long run. Suppose that parameter θ_i has been selected. Gibbs sampling then chooses a new value for that parameter by generating a random value directly from the conditional probability distribution $p(\theta_i|\{\theta_{j\neq i}\}, D)$. The new value for θ_i, combined with the unchanged values of $\theta_{j\neq i}$, constitutes the new position in the random walk. The process then repeats: Select a component parameter and select a new value for that parameter from its conditional posterior distribution.

Figure 7.7 illustrates this process for a two-parameter example. In the first step, we want to select a new value for θ_1. We conditionalize on the values of all the other parameters, $\theta_{j\neq 1}$, from the previous step in the chain. In this example, there is only one other parameter, namely θ_2. The upper panel of Figure 7.7 shows a slice through the joint distribution at the current value of θ_2. The heavy curve is the shape of the posterior distribution conditional on this value of θ_2, denoted $p(\theta_1|\{\theta_{j\neq 1}\}, D)$, which is $p(\theta_1|\theta_2, D)$ in this case because there is only one other parameter. If the mathematical form of the conditional distribution is appropriate, a computer can directly generate a random value of θ_1. Having thereby generated a new value for θ_1, we then conditionalize on it and determine the conditional distribution of the next parameter, θ_2, as shown in the lower panel of Figure 7.7. The conditional distribution is denoted formally as $p(\theta_2|\{\theta_{j\neq 2}\}, D)$, which equals $p(\theta_2|\theta_1, D)$ in this case of only two parameters. If the mathematical form is convenient, our computer can directly generate a random value of θ_2 from the conditional distribution. We then conditionalize on the new value θ_2, and the cycle repeats.

Gibbs sampling can be thought of as just a special case of the Metropolis algorithm, in which the proposal distribution depends on the location in parameter space and the

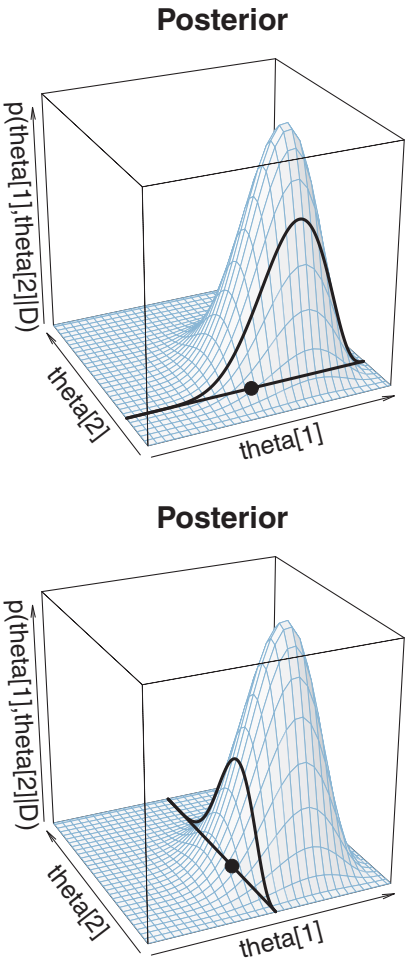

Figure 7.7 Two steps in a Gibbs sampling. In the upper panel, the heavy lines show a slice through the posterior conditionalized on a particular value of θ_2, and the large dot shows a random value of θ_1 sampled from the conditional density. The lower panel shows a random generation of a value for θ_2, conditional on the value for θ_1 determined by the previous step. The heavy lines show a slice through the posterior at the conditional value of θ_1, and the large dot shows the random value of θ_2 sampled from the conditional density.

component parameter selected. At any point, a component parameter is selected, and then the proposal distribution for that parameter's next value *is* the conditional posterior probability of that parameter. *Because the proposal distribution exactly mirrors the posterior probability for that parameter, the proposed move is always accepted*. A rigorous proof of this idea requires development of a generalized form of the Metropolis algorithm, called the Metropolis-Hastings algorithm (Hastings, 1970). A technical overview of the relation is provided by Chib and Greenberg (1995), and an accessible mathematical tutorial is given by Bolstad (2009).

Gibbs sampling is especially useful when the complete joint posterior, $p(\{\theta_i\}|D)$, cannot be analytically determined and cannot be directly sampled, but all the conditional distributions, $p(\theta_i|\{\theta_{j\neq i}\}, D)$, can be determined and directly sampled. We will not encounter such a situation until later in the book, but the process of Gibbs sampling can be illustrated now for a simpler situation.

We continue now with estimating two coin biases, θ_1 and θ_2, when assuming that the prior belief distribution is a product of beta distributions. The posterior distribution is again a product of beta distributions, as was derived in Equation 7.7 (p. 165). This joint posterior is easily dealt with directly, and so there is no real need to apply Gibbs sampling, but we will go ahead and do Gibbs sampling of this posterior distribution for purposes of illustration and comparison with other methods.

To accomplish Gibbs sampling, we must determine the conditional posterior distribution for each parameter. By definition of conditional probability,

$$\begin{aligned} p(\theta_1|\theta_2, D) &= p(\theta_1, \theta_2|D)/p(\theta_2|D) && \text{conditional is joint divided by marginal} \\ &= p(\theta_1, \theta_2|D) \Big/ \int \mathrm{d}\theta_1\, p(\theta_1, \theta_2|D) && \text{marginal is integral of joint} \end{aligned}$$

For our current application, the joint posterior is a product of beta distributions, as was derived in Equation 7.7, p. 165. Therefore, substituting into the equation above, we have

$$\begin{aligned} p(\theta_1|\theta_2, D) &= p(\theta_1, \theta_2|D) \Big/ \int \mathrm{d}\theta_1\, p(\theta_1, \theta_2|D) \\ &= \frac{\mathrm{beta}(\theta_1|z_1+a_1, N_1-z_1+b_1)\,\mathrm{beta}(\theta_2|z_2+a_2, N_2-z_2+b_2)}{\int \mathrm{d}\theta_1\,\mathrm{beta}(\theta_1|z_1+a_1, N_1-z_1+b_1)\,\mathrm{beta}(\theta_2|z_2+a_2, N_2-z_2+b_2)} \\ &= \frac{\mathrm{beta}(\theta_1|z_1+a_1, N_1-z_1+b_1)\,\mathrm{beta}(\theta_2|z_2+a_2, N_2-z_2+b_2)}{\mathrm{beta}(\theta_2|z_2+a_2, N_2-z_2+b_2)\int \mathrm{d}\theta_1\,\mathrm{beta}(\theta_1|z_1+a_1, N_1-z_1+b_1)} \\ &\qquad\qquad \text{because } \mathrm{beta}(\theta_2|\ldots) \text{ is constant w.r.t. } \theta_1 \\ &= \frac{\mathrm{beta}(\theta_1|z_1+a_1, N_1-z_1+b_1)\,\mathrm{beta}(\theta_2|z_2+a_2, N_2-z_2+b_2)}{\mathrm{beta}(\theta_2|z_2+a_2, N_2-z_2+b_2)} \\ &\qquad\qquad \text{because integral of prob. distrib. must be 1} \\ &= \mathrm{beta}(\theta_1|z_1+a_1, N_1-z_1+b_1) \end{aligned} \tag{7.10}$$

From these considerations, you can also see that the other conditional posterior probability is $p(\theta_2|\theta_1, D) = \mathrm{beta}(\theta_2|z_2+a_2, N_2-z_2+b_2)$. We have just derived what may have already been intuitively clear: Because the posterior is a product of independent beta distributions, it makes sense that $p(\theta_1|\theta_2, D) = p(\theta_1|D)$. Nevertheless, the derivation illustrates the sort of analytical procedure that is needed in general. In general, the

posterior distribution will not be a product of independent marginal distributions as in this simple case. Instead, the formula for $p(\theta_i|\theta_{j\neq i}, D)$ will typically involve the values of $\theta_{j\neq i}$.

The conditional distributions just derived were displayed graphically in Figure 7.7. The upper panel shows a slice conditional on θ_2, and the heavy curve illustrates $p(\theta_1|\theta_2, D)$ which is beta$(\theta_1|z_1+a_1, N_1-z_1+b_1)$. The lower panel shows a slice conditional on θ_1, and the heavy curve illustrates $p(\theta_2|\theta_1, D)$, which is beta$(\theta_2|z_2+a_2, N_2-z_2+b_2)$.

Having successfully derived the conditional posterior probability distributions, we now figure out whether we can directly sample from them. In this case, the answer is yes, we can, because the conditional probabilities are beta distributions, and our computer software comes prepackaged with generators of random beta values.

Figure 7.8 shows the result of applying Gibbs sampling to this scenario. The left panel shows every step, changing one parameter at a time. Notice that each step in the random walk is parallel to a parameter axis, because each step changes only one component parameter. You can also see that at each point, the walk direction changes to the other parameter, rather than doubling back on itself or continuing in the same direction. This is because the walk cycled through the component parameters, $\theta_1, \theta_2, \theta_1, \theta_2, \theta_1, \theta_2, \ldots$, rather than selecting them randomly at each step.

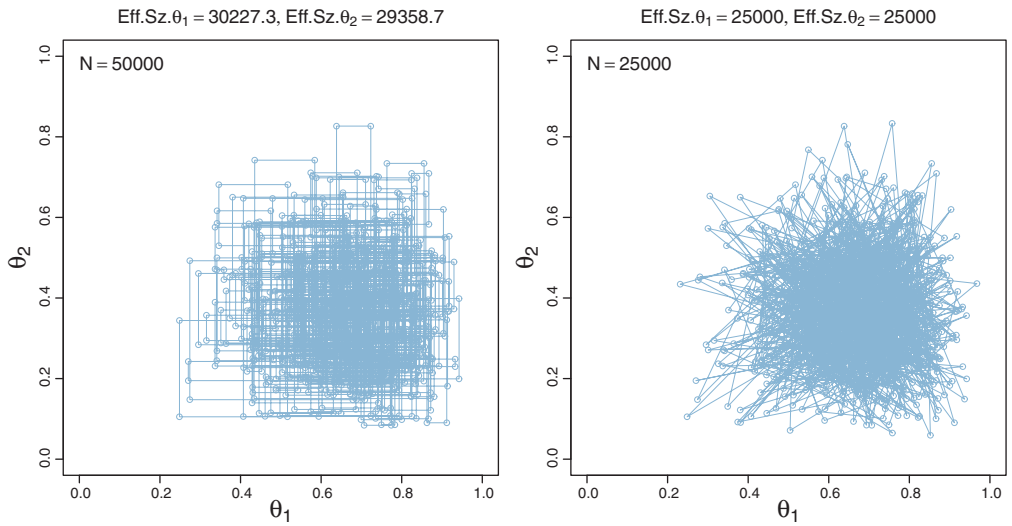

Figure 7.8 Gibbs sampling applied to the posterior shown in Figure 7.5, p. 167. The left panel shows all the intermediate steps of chain, changing one parameter at a time. The right panel shows only the points after complete sweeps through all (two) parameters. Both are valid samples from the posterior distribution. Only 1,000 of the *N* steps are displayed here. Compare with the results of the Metropolis algorithm in Figure 7.6. Notice that the effective size of the Gibbs sample is larger than the effective size of the Metropolis sample for the same length of chain.

The right panel of Figure 7.8 only shows the steps after each complete cycle through the parameters. It is the same chain as in the left panel, only with the intermediate steps not shown. Both the left and the right panels are representative samples from the posterior distribution, and they would converge to the same continuous distribution in the limit of an infinitely long chain. The software we will use later in the book (called JAGS) records only complete cycles, not intermediate single-parameter steps.

If you compare the results of Gibbs sampling in Figure 7.8 with the results of the Metropolis algorithm in Figure 7.6 (p. 169), you will see that the points produced by Gibbs sampling and the Metropolis algorithm look similar in where they fall, if not in the trajectory of the walk that produces them. In fact, in the infinitely long run, the Gibbs and Metropolis methods converge to the same distribution. What differs is the efficiency of getting to any desired degree of approximation accuracy in a finite run. Notice that the effective size of the Gibbs sampler, displayed in the panel annotations of Figure 7.8, is much greater than the effective size of the Metropolis algorithm displayed in Figure 7.6. (The right panel of Figure 7.8 shows that the effective size is the same as the number of complete cycles of the Gibbs sampler in this case, but this is not generally true and happens here because the conditional distributions are independent of each other.)

By helping us visualize *how* Gibbs sampling works, Figure 7.8 also helps us understand better *why* it works. Imagine that instead of changing the component parameter at every step, we linger a while on a component. Suppose, for example, that we have a fixed value of θ_1, and we keep generating new random values of θ_2 for many steps. In terms of Figure 7.8, this amounts to lingering on a vertical slice of the parameter space, lined up over the fixed value of θ_1. As we continue sampling within that slice, we build up a good representation of the posterior distribution for that value of θ_1. Now we jump to a different value of θ_1, and again linger a while at the new value, filling in a new vertical slice of the posterior. If we do that enough, we will have many vertical slices, each representing the posterior distribution along that slice. We can use those vertical slices to represent the posterior, *if* we have also lingered in each slice proportionally to the posterior probability of being in that slice! Not all values of θ_1 are equally likely in the posterior, so we visit vertical slices according to the conditional probability of θ_1. Gibbs sampling carries out this process, but lingers for only one step before jumping to a new slice.

So far, I have emphasized the advantages of Gibbs sampling over the Metropolis algorithm, namely, that there is no need to tune a proposal distribution and no inefficiency of rejected proposals. I also mentioned restrictions: We must be able to derive the conditional probabilities of each parameter on the others and be able to generate random samples from those conditional probability distributions.

But there is one other disadvantage of Gibbs sampling. Because it only changes one parameter value at a time, its progress can be stalled by highly correlated parameters. We will encounter applications later in which credible combinations of parameter values can be very strongly correlated; see, for example, the correlation of slope and intercept parameters in Figure 17.3, p. 481. Here I hope merely to plant the seeds of an intuition for later development. Imagine a posterior distribution over two parameters. Its shape is a narrow ridge along the *diagonal* of the parameter space, and you are *inside, within,* this narrow ridge, like being inside a long-narrow hallway that runs diagonally to the parameter axes. Now imagine doing Gibbs sampling from this posterior. You are in the hallway, and you are contemplating a step along a parameter axis. Because the hallway is narrow and diagonal, a step along a parameter axis quickly encounters a wall, and your step size must be small. This is true no matter which parameter axis you face along. Therefore, you can take only small steps and only very gradually explore the length of the diagonal hallway. On the other hand, if you were stepping according to a Metropolis sampler, whereby your proposal distribution included changes of both parameters at once, then you could jump in the diagonal direction and quickly explore the length of the hallway.

7.4.5. Is there a difference between biases?

Until this point, we have focused on the mechanics of Gibbs sampling (and the Metropolis algorithm) without answering the question, How different are the biases of the two coins? Fortunately, the representative sample from the joint posterior distribution gives us an answer. Notice that every step in the chain (either from Gibbs or Metropolis) is a combination of θ_1 and θ_2 values that are jointly credible. Therefore, at each step in the chain, a credible difference is $\theta_1 - \theta_2$. Across the full chain, we have a large representative sample of credible differences. (We cannot get a sense of the difference of parameters merely by looking at the marginal distributions of those parameters; see Section 12.1.2.1, p. 340.)

Figure 7.9 shows histograms of $\theta_1 - \theta_2$ from the posterior distribution, generated by each of the chains in Figures 7.6 and 7.8. In the limit of an infinite chain, they would all converge to the same true underlying posterior distribution, but any finite chain is an approximation. In principle, on average, the results from a chain with larger effective size should be more accurate than a smaller effective size, although the smaller might get lucky and happen to be more accurate on a given random walk. In this case, therefore, the results from the Gibbs sampling are probably more accurate than the results from the Metropolis algorithm.

Figure 7.9 reveals that a difference of zero is clearly among the 95% most credible differences (i.e., within the 95% HDI), and we would not want to declare that there is a difference. The Bayesian posterior gives complete information about credible differences

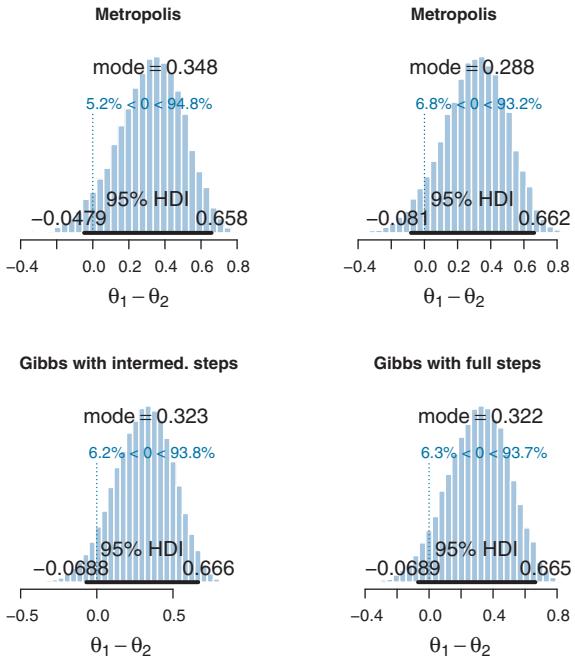

Figure 7.9 Credible posterior differences between biases. Upper panels come from results of Metropolis algorithm in Figure 7.6 using two different proposal standard deviations; lower panels come from results of Gibbs sampling in Figure 7.8. The four distributions are nearly the same, and in the limit for infinitely long chains, should be identical. For these finite chains, the ones with longer effective size (i.e., the Gibbs sampled) are more accurate on average.

and indicates that there is a suggestion of difference, but our uncertainty is large because the amount of data is small. Notice that the decision rule is a separate process from Bayes' rule, and the result of a decision rule leaves behind most of the posterior information. See Section 12.1.1, p. 336, for complete discussion of HDI and decisions.

7.4.6. Terminology: MCMC

Assessing the properties of a target distribution by generating representative random values is a case of a Monte Carlo simulation. Any simulation that samples a lot of random values from a distribution is called a Monte Carlo simulation, named after the dice and spinners and shufflings of the famous casino locale. The appellation "Monte Carlo" is attributed (Eckhardt, 1987) to the mathematicians Stanislaw Ulam (1909–1984) and John von Neumann (1903–1957).

The Metropolis algorithm and Gibbs sampling are specific types of Monte Carlo process. They generate random walks such that each step in the walk is completely

independent of the steps before the current position. Any such process in which each step has no memory for states before the current one is called a (first-order) Markov process, and a succession of such steps is a Markov chain (named after the mathematician Andrey Markov, 1856–1922). The Metropolis algorithm and Gibbs sampling are examples of a *Markov chain Monte Carlo (MCMC)* process. There are many others. It is the invention of MCMC algorithms, along with software for automatically creating samplers for complex models (such as BUGS, JAGS, and Stan), and fast cheap computers, that allow us to do Bayesian data analysis for complex realistic data.

7.5. MCMC REPRESENTATIVENESS, ACCURACY, AND EFFICIENCY

We have 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. They should not be unduly influenced by the arbitrary initial value of the chain, and they 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*. In particular, the estimates of the central tendency (such as median or mode), and the limits of the 95% HDI, should not be much different if the MCMC analysis is run again (using different seed states for the pseudorandom number generators).
3. The chain should be generated *efficiently*, with as few steps as possible, so not to exceed our patience or computing power.

These goals are achieved only more or less, not absolutely, in real applications. In principle, the mathematics of MCMC guarantee that infinitely long chains will achieve a perfect representation of the posterior distribution, but unfortunately we do not have infinite time or computer memory. Therefore, we must check the quality of finite MCMC chains to determine whether there are signs of *un*representativeness or *in*stability. For typical models of moderate complexity such as those in this book, the MCMC chains are usually well behaved, and checking them is routine. As models increase in complexity, their MCMC chains may be more problematic and checking them can be more important and more challenging.

7.5.1. MCMC representativeness

Checks for unrepresentativeness usually look for lingering influence of the initial value and for orphaned chains that have somehow settled into unusual regions of parameter space. There is (at the time of this writing) no single universally best method for conducting these checks. Current practice often focuses on two methods: visual examination of the trajectory, and consideration of a numerical description of convergence.

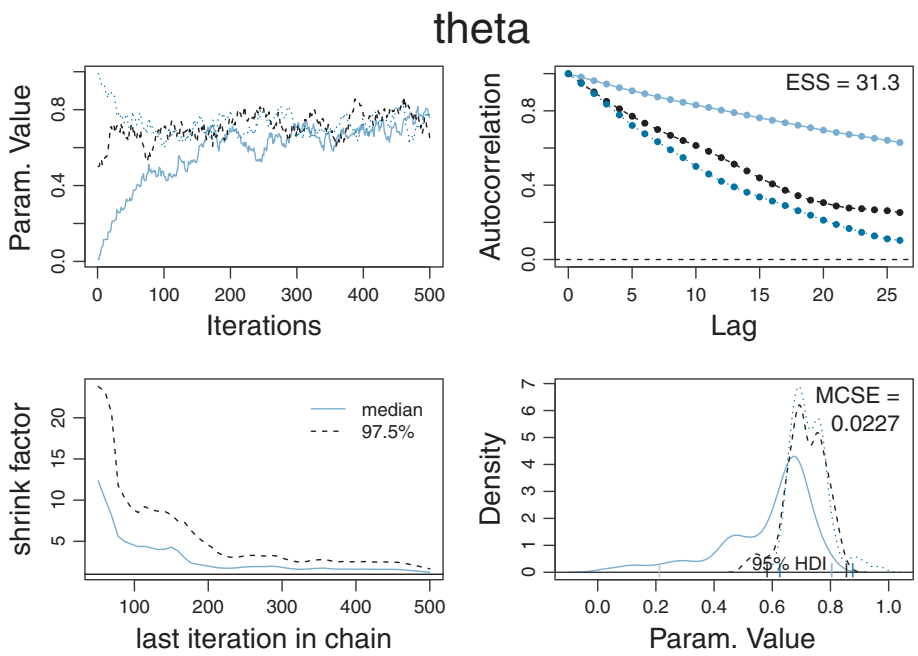

Figure 7.10 Illustration of MCMC diagnostics. Three chains were generated by starting a Metropolis algorithm at different initial values, with proposal SD=0.02 (cf. Figure 7.4) for data $z = 35, N = 50$. Only steps 1–500 are shown here. See Figure 7.11 for later steps in the chain.

The first method to detect unrepresentativeness is a visual examination of the chain trajectory. A graph of the sampled parameter values as a function of step in the chain is called a *trace plot*. Examples appeared previously in Figure 7.4, and new examples appear in Figures 7.10 and 7.11. One way to enhance the visibility of unrepresentative parts of the chain is to superimpose two or more chains (that have been sampled with independent pseudo-random numbers). If the chains are all representative of the posterior distribution, they should overlap each other. Figure 7.10 shows the early steps of three MCMC trajectories started at different initial values. The upper-left panel of Figure 7.10 shows the trace plot. The vertical axis is the parameter value, and the horizontal axis is the step in the chain, labeled as "iterations." The trajectories reveal that it takes a few hundred steps for the three chains to converge to the same region of the parameter. This visual assessment suggests that the first several hundred steps of the chain should be excluded from the sample because they are not representative. The preliminary steps, during which the chain moves from its unrepresentative initial value to the modal region of the posterior, is called the *burn-in* period. For realistic applications, it is routine to apply a burn-in period of several hundred to several thousand steps. Later steps in the chains are shown in Figure 7.11. In particular, the upper-left panel of Figure 7.11

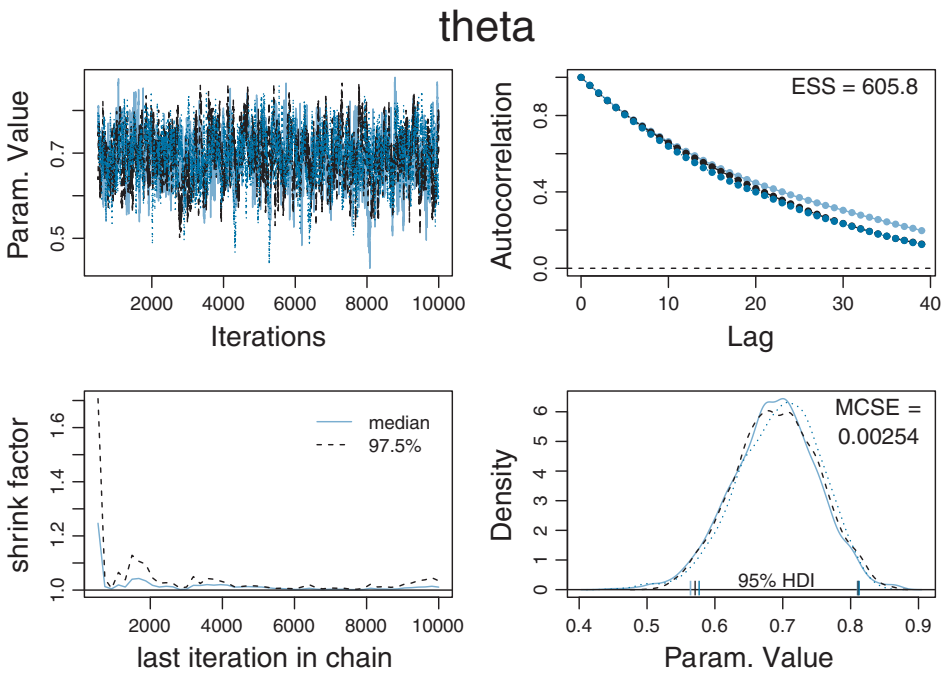

Figure 7.11 Illustration of MCMC diagnostics. Three chains were generated by starting a Metropolis algorithm at different initial values, with proposal SD=0.02 (cf. Figure 7.4) for data $z = 35$, $N = 50$. Steps 500–10,000 are shown here. See Figure 7.10 for earlier steps in the chain.

shows the trace plot, where it can be seen that the three chains meander fairly smoothly and overlap each other. If any chain were isolated from the others, it would be a sign that convergence had not been achieved. If any chain lingered for extended durations at (nearly) the same value, or changed values only very gradually, it might also be a sign of failure to converge. In the present case, fortunately, the chains are thoroughly overlapping, which is a good sign that suggests representativeness. But, the chains do meander relatively slowly, which is a sign of inefficiency (to be discussed more below). It is important to understand that the logic of checking for representativeness goes only one way: If the chains are representative, then they should overlap and mix well. But the converse is not necessarily true. The chains might overlap, but all be stuck in the same unrepresentative part of parameter space. Fortunately, this is rarely a problem for moderately complex models with chains started at reasonable initial values.

Another useful visual representation appears in the lower-right panels of Figures 7.10 and 7.11. These plots show smoothed histograms of the parameter values sampled in each chain. Smoothed histograms are also called *density plots*. Unlike histograms, which show the exact proportion of points in each histogram bin, density plots average across overlapping intervals to produce a smooth representation of probability density. (You can

learn about the details by typing `?density` at the command line in R.) The lower-right panel of Figure 7.10 shows that the density plots of the three chains do *not* overlap very well during the burn-in period. This is a clear visual indicator that the chains have not converged. On the other hand, the lower-right panel of Figure 7.11 shows that the density plots of the three chains *do* overlap well after the burn-in period. This suggests, but does not guarantee, that the chains are producing representative values from the posterior distribution. Notice also that the density plots display the estimated 95% HDI for each chain. The HDI limits are slightly different for each chain, of course, because each chain is a finite random sample from the posterior distribution. In the limit, for infinite chains, the HDI limits of different chains will all converge to the same values. For finite chains, the display provides a visual impression of the variability in the estimates due to the finite sample size of the chain. (The meaning of "MCSE," displayed in the density plots, is explained below.)

Besides the visual checks of convergence, there are also numerical checks. One popular numerical check is a measure of how much variance there is between chains relative to how much variance there is within chains. The idea is that if all the chains have settled into a representative sampling, then the average difference between the chains should be the same as the average difference (across steps) within the chains. But, if one or more chains is orphaned or stuck, it will increase the between-chain variance relative to the within-chain variance. The lower-left panels of Figures 7.10 and 7.11 show plots of this measure. You can see that during the burn-in period (Figure 7.10), the measure greatly exceeds 1.0. After the burn-in period (Figure 7.11), the measure quickly gets very close to 1.0.

The specific numerical measure is called the Gelman-Rubin statistic (Gelman & Rubin, 1992), or the Brooks-Gelman-Rubin statistic (Brooks & Gelman, 1998), or the "potential scale reduction factor," or simply the "shrink factor" as plotted in Figures 7.10 and 7.11. Intuitively, its value is 1.0 if the chains are fully converged, but its value is larger than 1.0 if there are orphaned or stuck chains. As a heuristic, if the Gelman-Rubin statistic is greater than 1.1 or so, you should worry that perhaps the chains have not converged adequately. The exact definition of the Gelman-Rubin statistic involves a lot of details that are not essential for the purposes of this book. Therefore, interested readers are encouraged to learn more by reading the original articles or secondary summaries such as Gill (2009, p. 478) or Ntzoufras (2009, p. 143). The plots in the left columns of Figures 7.10 and 7.11 were produced by functions from the `coda` package created by Martyn Plummer et al. Later in the book, you will be introduced to the JAGS system for MCMC sampling, also created by Martyn Plummer, and the `coda` package will come along with the JAGS system. Meanwhile, another way to learn more about the Gelman-Rubin statistic is by installing the `coda` package into R. At R's command line, type `install.packages("coda")`, then load it into memory by typing `library(coda)`, and then find out about the diagnostic measure by typing `?gelman.diag`.

7.5.2. MCMC accuracy

After we have some assurance that the chains are genuinely representative samples from the posterior distribution, the second main goal is to have a large enough sample for stable and accurate numerical estimates of the distribution. The larger the sample, the more stable and accurate (on average) will be the estimates of the central tendency and HDI limits. But, as we saw in Figure 7.4 (p. 159), some chains can be more clumpy than others. Successive steps in a clumpy chain do not provide independent information about the parameter distribution.

What we need, therefore, are measures of chain length and accuracy that take into account the clumpiness of the chain. And for that, we need a measure of clumpiness. We will measure clumpiness as *autocorrelation*, which is simply the correlation of the chain values with the chain values k steps ahead. There is a different autocorrelation for each choice of k.

Figure 7.12 shows an example of computing autocorrelation. The upper panels show an MCMC chain of 70 steps, superimposed with the same chain translated a certain number of steps ahead. The vertical axis plots the parameter value, and the horizontal axis plots the step in the chain, here labeled as the "index." The number of steps between the chain and its superimposed copy is called the *lag*. The three columns show three different lags. The middle row shows the values of the original chain (on the vertical axis) plotted against the lagged values of the chain (on the horizontal axis). The scatter plot makes it easy to visualize the correlation of the original and lagged values. To help understand the correspondence of the upper row and middle row, a point in each scatter plot is marked by a square, and the corresponding values in the upper row are framed by a rectangle. Specifically, the value of the chain at about index 50 is nearly 16. At a lag of 10 (about index 60), the value of the chain is about 13. The square in the middle-right panel surrounds the point plotted at about $\langle 13, 16 \rangle$, and the rectangle in the upper-right panel surrounds those values at about index 50 in the original chain.

The autocorrelation can be computed for any lag of interest. The *autocorrelation function* is the autocorrelation across a spectrum of candidate lags. For our applications, we are typically interested in lags from around 1 to 20 or so. The lower panel of Figure 7.12 shows the autocorrelation function for the chain. Because lags are discrete, the autocorrelations are plotted as bars at each integer lag. The autocorrelation at lag k is denoted ACF(k), and you can see the correspondence of the ACF annotating the three scatterplots and the bar heights. The plot of the ACF reveals that this chain is highly autocorrelated, which means that it changes only gradually from step to step. In other words, this chain is fairly clumpy.

ACFs were also displayed in Figures 7.10 and 7.11, in their upper-right panels. There are multiple chains being plotted, with a distinct ACF for each chain. Instead of using bar graphs to plot the ACFs, the figures use points connected by lines because it is easier to

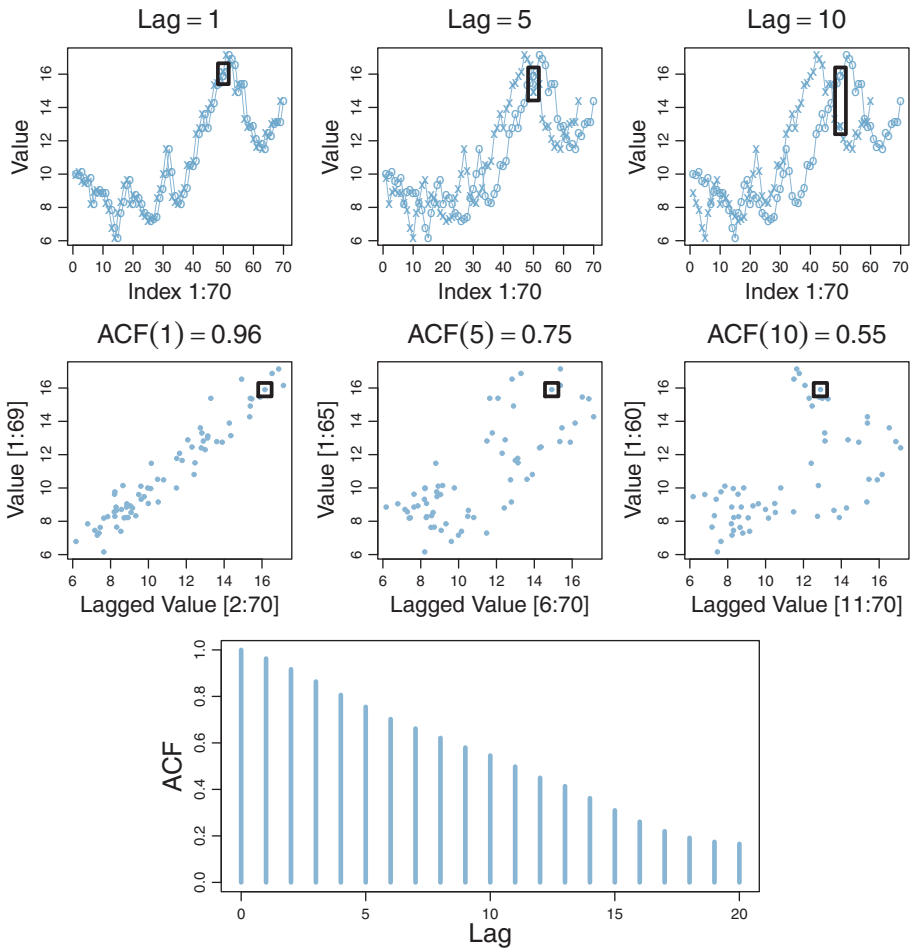

Figure 7.12 Autocorrelation of a chain. Upper panels show examples of lagged chains. Middle panels show scatter plots of chain values against lagged chain values, with their correlation annotated. Lowest panel shows the autocorrelation function (ACF).

see the superimposed ACFs. You can see that the chains are highly autocorrelated, insofar as the autocorrelations remain well above zero for large lags. Therefore, the parameter values at successive steps in the chain are not providing independent information about the posterior distribution, because each successive step is partially redundant with the previous step. The upshot is that it takes a lot of steps of these chains to generate independent information about the posterior distribution.

We would like some measure of how much independent information there is in autocorrelated chains. In particular, we can ask, what would be the sample size of a completely *non*-autocorrelated chain that yielded the same information? An answer to

this question is provided by a measure called the *effective sample size* (proposed by Radford Neal in Kass, Carlin, Gelman, & Neal, 1998, p. 99), which divides the actual sample size by the amount of autocorrelation. Formally, denote the actual number of steps in the chain as N. The ESS is

$$\text{ESS} = N \Big/ \left(1 + 2\sum_{k=1}^{\infty} \text{ACF}(k)\right) \tag{7.11}$$

where ACF(k) is the autocorrelation of the chain at lag k. For practical computation, the infinite sum in the definition of ESS may be stopped when ACF(k) < 0.05 (because, typically, ACF($k+1$) $<$ ACF(k)). In R, the ESS can be computed by the `effectiveSize` function in the `coda` package (which uses a different algorithm than Equation 7.11).

The upper-right panels of Figures 7.10 and 7.11 annotate the ACFs with the *total* ESS across the multiple chains. For example, in Figure 7.11, there are three chains, each with $N = 9{,}500$ (i.e., each chain goes from "iteration" 500–10,000), yielding 28,500 steps overall. But the autocorrelation is so high that the ESS is only 605.8. The first decimal value is included in the display of ESS merely as a reminder that the ESS is an estimated continuous number, not to be confused with an actual sample.

How big should the ESS be for an accurate and stable picture of the posterior distribution? The answer depends on which detail of the posterior distribution you want a handle on. For aspects of the distribution that are strongly influenced by dense regions, such as the median in unimodal distributions, the ESS does not need to be huge. But for aspects of the distribution that are strongly influenced by sparse regions, such as the limits of the 95% HDI, the ESS needs to be relatively large. The reason is that sparse regions of the distribution are relatively rarely sampled in the chain, and therefore, a long chain is required to generate a high-resolution picture of sparse regions such as 95% HDI limits. One simple guideline is this: For reasonably accurate and stable estimates of the limits of the 95% HDI, an ESS of 10,000 is recommended. This is merely a heuristic based on experience with practical applications, it is not a requirement. If accuracy of the HDI limits is not crucial for your application, then a smaller ESS may be sufficient.

To get an intuition for the (in-)stability of the estimates of the 95% HDI limits, we will repeatedly generate MCMC chains from a known distribution, which has precisely known true 95% HDI limits. In particular, a standardized normal distribution has 95% HDI limits at very nearly -1.96 and $+1.96$. We will repeatedly generate MCMC chains from the normal distribution, each time using an ESS of 10,000. For each MCMC sample, we will estimate the 95% HDI. Then we will look at the estimates and assess how much they depart from the known true values.

Figure 7.13 shows the results of 50,000 repetitions (each using an ESS of 10,000). The upper panels show that the estimate of each HDI limit, where it can be seen that

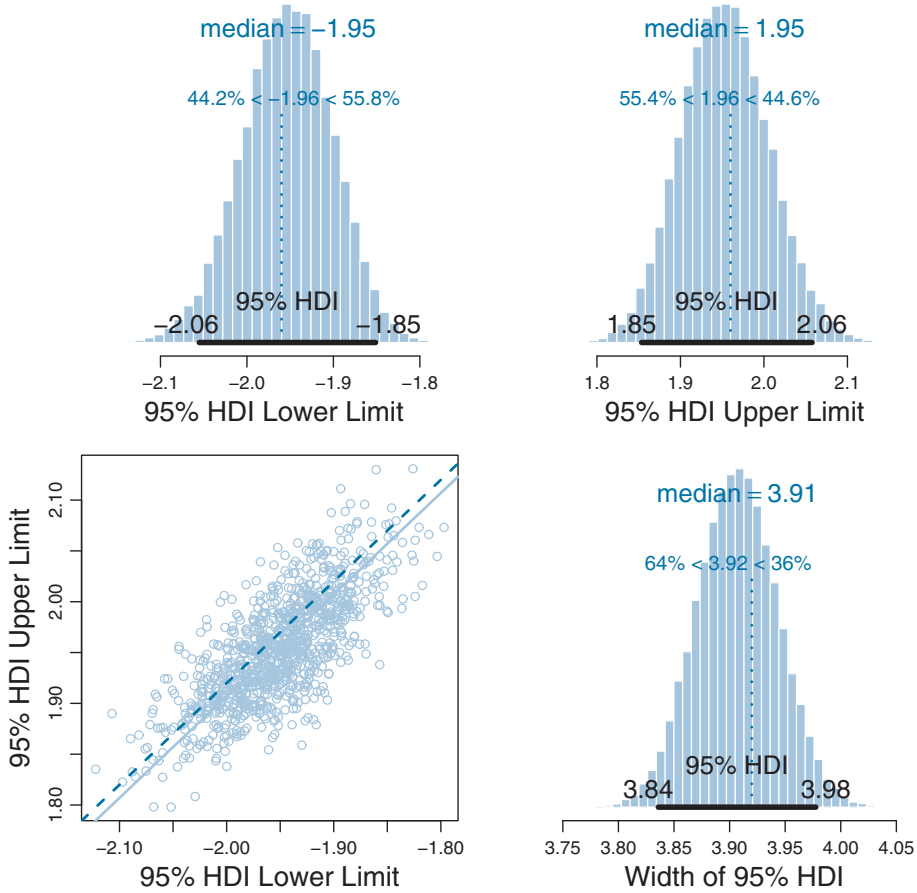

Figure 7.13 Estimated 95% HDI limits for random samples from a standardized normal distribution that have an *ESS* of *10,000*. Repeated runs yield a distribution of estimates as shown here; there were 50,000 repetitions. Upper panels show estimate of HDI limits. Lower panels show estimate of HDI width. True values are indicated by dashed lines.

the median estimate of ± 1.95 is a little less extreme than the true value of ± 1.96. The distribution of the estimated HDI limit has a SD of about 0.053. The sample of parameter values should have a SD of 1.0 because it comes from a standardized normal. Hence, the SD of the estimate of the 95% HDI limit, when using an ESS of 10,000, is roughly 5% of the SD of the MCMC chain.

The width of the HDI is slightly underestimated, but not by as much as might be suggested by the marginal distributions of the HDI limits. The estimates of the HDI limits are strongly correlated, as shown in the lower-left panel of Figure 7.13. (See Section 12.1.2.1, p. 340, for more discussion.) The distribution of the estimated widths is shown in the lower right panel, where it can be seen that the median estimated width

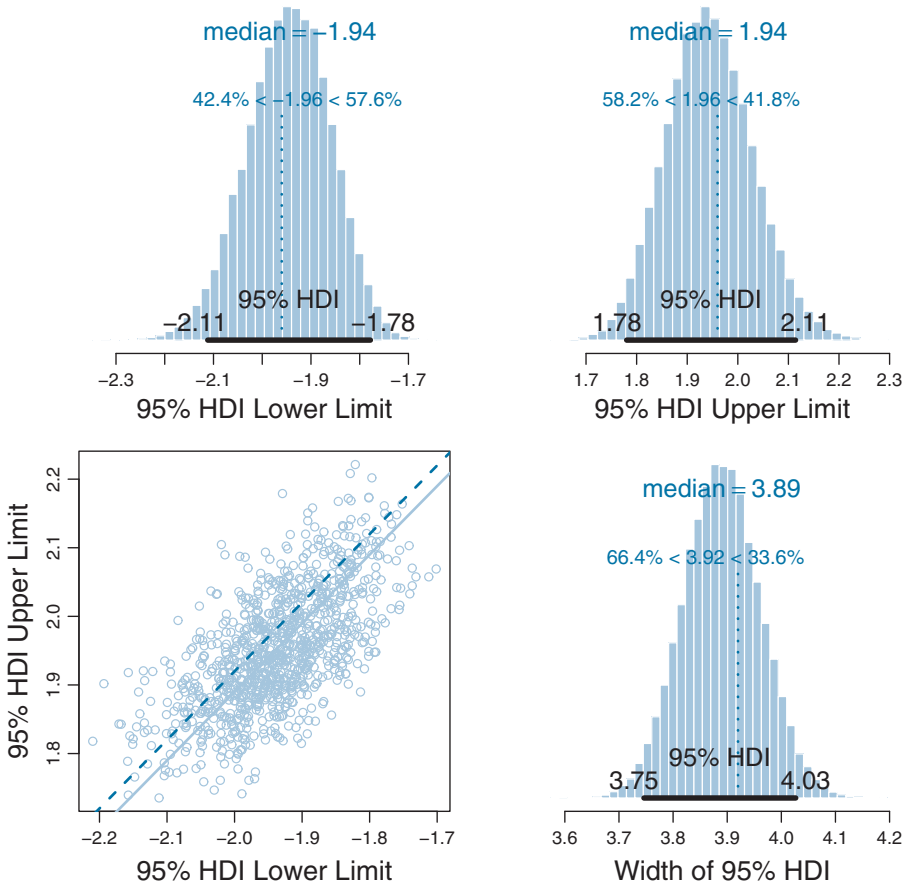

Figure 7.14 Estimated 95% HDI limits for random samples from a standardized normal distribution that have an *ESS of only 2,500*. Repeated runs yield a distribution of estimates as shown here; there were 50,000 repetitions. Upper panels show estimate of HDI limits. Lower panels show estimate of HDI width. True values are indicated by dashed lines.

is only slightly less than the true width. The true width of the 95% HDI is 3.920, and the median of the estimated widths is 3.907, which indicates that the estimated width is about 99.7% of the true width in this case.

The SD of the HDI estimate gets bigger for skewed tails of distributions, and for smaller ESS. For instance, when the ESS is only 2,500 instead of 10,000, the estimated HDI is more unstable across repeated Monte Carlo runs, as shown in Figure 7.14. When compared with Figure 7.13, it can be seen that the smaller ESS produces, on average, slightly worse underestimates of the HDI limits and width, and less stable estimates.

Another useful measure of the effective accuracy of the chain is the Monte Carlo standard error (MCSE). First, some background concepts and terminology. Consider

randomly sampling values $x_1, \ldots, x_N$ from a normal distribution, and computing the mean of the sample, $\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$. The sample mean will be a noisy indicator of the underlying mean μ of the normal generating distribution. If we sample N values many times, sometimes the sample mean $\bar{x}$ will be greater than μ and sometimes less than μ. The SD of the sample mean, across repetitions, is called the *standard error* of the sample mean, and its estimated value is simply $\text{SE} = \text{SD}/\sqrt{N}$, where SD is the standard deviation of the sample. Thus, as the sample size increases, the standard error decreases. In other words, the bigger the sample, the less noise there is in the estimate of the underlying mean, and the standard error itself provides a quantitative suggestion of how big the estimation noise is. The notion of standard error for independent values is extended to Markov chains merely by substituting the ESS for the actual sample size. Thus, a simple formulation of the MCSE is

$$\text{MCSE} = \text{SD}/\sqrt{\text{ESS}} \tag{7.12}$$

where SD is the standard deviation of the chain and ESS is as defined in Equation 7.11. This is a simple version of MCSE; for more elaborate considerations see Flegal, Haran, and Jones (2008).

Examples of the MCSE are displayed in the lower-right panels of Figures 7.10 and 7.11. The MCSE indicates the estimated SD of the sample mean in the chain, on the scale of the parameter value. In Figure 7.11, for example, despite the small ESS, the mean of the posterior appears to be estimated very stably.

Here is a recapitulation regarding accuracy and stability of MCMC results. Visual inspection of the trace plots and density plots, and the Gelman-Rubin statistic, can suggest whether the burn-in period has been suitably passed. Second, those indicators can also suggest whether or not the chains are well mixed and representative of the posterior. Remember, the diagnostics logically can only probabilistically indicate violations of representativeness and cannot guarantee representativeness. Next, the measures of ESS and MCSE suggest how stable and accurate the chain is. As a heuristic, if you want reasonable stability in the estimates of the limits of the 95% HDI, an ESS of (at least) 10,000 is desirable. If you want a particular accuracy in the estimate of the posterior mean, consult the MCSE, which is interpreted on the scale of the parameter.

7.5.3. MCMC efficiency

The third of our main goals for MCMC is efficiency, so we do not exceed our patience and computing power. It is often the case in realistic applications that there is strong autocorrelation for some parameters, and therefore, an extremely long chain is required to achieve an adequate ESS or MCSE. There are various ways to (attempt to) improve the efficiency of the MCMC process.

- Run chains on parallel hardware. Most new computers have four or more processors, and each can be recruited to run a chain simultaneously, in parallel. Running parallel chains does not improve the amount of information per step in each chain, but it does improve the amount of information per increment of real time. The package `runjags` is nice for running parallel chains, as we will see in Chapter 8.
- Adjust the sampling method; for example, use a Gibbs sampler instead of a Metropolis sampler, as illustrated in the improvement in ESS from Figures 7.6 to 7.8. This approach requires great knowledge of various samplers and what types of model structures and substructures work best with which samplers. We will not explore these nuances in this book and will instead let sophisticated software tackle this issue for us. One sampling method that can be relatively efficient is Hamiltonian Monte Carlo, which is implemented in the Stan software introduced in Chapter 14.
- Change the parameterization of the model. In some cases, the various parameters of the model can be algebraically re-expressed in equivalent different forms but MCMC operates more efficiently in one form than another. For example, two parameters α and β might be re-expressed as $\mu = (\alpha + \beta)/2$ and $\delta = (\alpha - \beta)/2$, hence $\alpha = \mu + \delta$ and $\beta = \mu - \delta$. (This particular reparameterization might have little effect on MCMC sampling but is mentioned to illustrate the idea.) One application of this idea is mean-centering of data in regression analysis, as we will see in Chapter 17. We will see few other examples of this approach because applications can require a nuanced understanding of technicalities specific to individual models, which goes beyond the intended purpose of this book.

A method for reducing autocorrelation that does not improve efficiency is *thinning* the chain. In thinning, only every *k*th step in the chain is stored in the computer's memory. For example, when thinning to every 10th step, only steps 1, 11, 21, 31, and so on, are retained. The resulting thinned chain is less autocorrelated than the original complete chain. But the thinned chain also has less information than the original chain, and estimates from a thinned chain are (on average) less stable and accurate than from the original chain (Link & Eaton, 2012). After all, each step in the original chain did provide some new information. And, of course, the thinned chain takes just as many steps as the original chain to generate, so there is no savings in time during the generation of the chain. The only reason to thin a chain is if storing the full original chain would take too much computer memory, or if subsequent processing of the full original chain would take too much time.

7.6. SUMMARY

Let's regain perspective on the forest of Bayesian inference after focusing on the trees of MCMC. Recall that the overarching goal of Bayesian analysis is identifying the credibility of parameter values in a descriptive model of data. Bayes' rule provides an exact

mathematical formulation for the posterior distribution on the parameter values. But the exact form requires evaluation of an integral that might be intractable for realistically complex models. Therefore, we approximate the posterior distribution, to arbitrarily high accuracy, using MCMC methods. Because of recent developments in MCMC algorithms, software that cleverly applies them in complex models, and hardware that runs them incredibly quickly, we can now use MCMC methods to analyze realistically complex models that would have been impossible only a few decades ago.

This chapter focused on explaining the concepts of MCMC, applied to simple examples that illustrate the concepts. (Software for realistic applications is introduced in the next chapter.) This chapter introduced a simple case of the Metropolis algorithm in the guise of a politician hopping across adjacent islands. Some basic mathematical intuitions were established in that case of discrete parameter values. A more general form of the Metropolis algorithm was then applied to estimating a continuous parameter, namely, the bias in a coin. After that, Gibbs sampling was introduced and applied to the estimation of the difference of biases from two coins. The Metropolis and Gibbs methods are two types of MCMC samplers. There are many others that are not discussed in this chapter. All the MCMC methods converge to an accurate representation of the posterior distribution in the infinitely long run, but they differ in how efficiently they generate representative samples for different models in finite runs.

The chapter concluded, in Section 7.5, with a discussion of diagnostics and heuristics for assessing whether an MCMC chain is representative of the posterior distribution and is sufficiently accurate. Figures 7.10 and 7.11 showed visual and numerical representations of MCMC chains, including trace plot, density plot, autocorrelation plot, and plot of the Gelman-Rubin statistic, along with two numerical indicators, the ESS and MCSE.

In subsequent chapters, we will use software that *automatically* sets up MCMC samplers for complex models. We will not need to manually tune proposal distributions in Metropolis samplers. We will not need to derive conditional distributions for Gibbs samplers. We will not need to figure out which of various sampling algorithms should be applied. But we will need to evaluate the output of the MCMC samplers and decide whether it is sufficiently representative and accurate. Thus, it is crucial to retain from this chapter the concepts of what MCMC does, and the details of how to assess it.

7.7. EXERCISES

Look for more exercises at https://sites.google.com/site/doingbayesiandataanalysis/

Exercise 7.1. [Purpose: Experiment with the Metropolis algorithm as displayed in Figure 7.4.] Open the program named `BernMetrop.R` from the files

that accompany this book. The script implements a Metropolis algorithm for Figure 7.4 Midway through the script, you will find a line that specifies the SD of the proposal distribution:

```
proposalSD = c(0.02,0.2,2.0)[2]
```

The line may look strange but it's merely a vector of constants with an index at the end to specify which component should be used. Thus, it's a simple way of specifying three options and then selecting one option. Run the script three times, once with each option (i.e., once with `[1]`, once with `[2]`, and once with `[3]`). *There is also a line that specifies the seed for the random number generator; comment it out so that you get a different trajectory than the ones shown in Figure 7.4.* Notice at the end of the script that you can specify the format of the graphic file for saving the resulting graph. Include the graphs in your write-up and describe whether they show similar behavior as the corresponding trajectories in Figure 7.4. Be sure to discuss the ESS.

Exercise 7.2. [Purpose: To explore the autocorrelation function in Figure 7.12.] At the end of the script `BernMetrop.R`, add these lines:

```
openGraph(height=7,width=3.5)
layout(matrix(1:2,nrow=2))
acf( acceptedTraj , lag.max=30 , col="skyblue" , lwd=3 )
Len = length( acceptedTraj )
Lag = 10
trajHead = acceptedTraj[  1      : (Len-Lag) ]
trajTail = acceptedTraj[ (1+Lag) :  Len      ]
plot( trajHead , trajTail , pch="." , col="skyblue" ,
      main=bquote( list( "Prpsl.SD" == .(proposalSD) ,
                         lag == .(Lag) ,
                         cor == .(round(cor(trajHead,trajTail),3)))) )
```

(A) Before each line, add a comment that explains what the line does. Include the commented code in your write-up.

(B) Repeat the previous exercise, with the lines above appended to the script. Include the resulting new graphs in your write-up. For each run, verify that the height of the ACF bar at the specified lag matches the correlation in the scatterplot.

(C) When the proposal distribution has SD=2, why does the scatter plot have a dense line of points on the diagonal? (*Hint:* Look at the trajectory.)

Exercise 7.3. [Purpose: Using a multimodal prior with the Metropolis algorithm, and seeing how chains can transition across modes or get stuck within them.] In this exercise, you will see that the Metropolis algorithm operates with multimodal distributions.

(A) Consider a prior distribution on coin bias that puts most credibility at 0.0, 0.5, and 1.0, which we will formulate as $p(\theta) = (\cos(4\pi\theta) + 1)^2/1.5$.

(B) Make a plot of the prior. *Hint:* `theta = seq(0,1,length=501); plot (theta , (cos(4*pi*theta)+1)^2/1.5 )`

(C) In the script `BernMetrop.R`, find the function definition that specifies the prior distribution. Inside that function definition, comment out the line that assigns a beta density to `pTheta`, and instead put in a trimodal prior like this:

```
#pTheta = dbeta( theta , 1 , 1 )
pTheta = (cos(4*pi*theta)+1)^2/1.5
```

To have the Metropolis algorithm explore the prior, we give it empty data. Find the line in the script that specifies the data and set `myData = c()`. Run the script, using a proposal SD=0.2. Include the graphical output in your write-up. Does the histogram of the trajectory look like the graph of the previous part of the exercise?

(D) Repeat the previous part but now with `myData = c(0,1,1)`. Include the graphical output in your write-up. Does the posterior distribution make sense? Explain why.

(E) Repeat the previous part but now with proposal SD=0.02. Include the graphical output in your write-up. Does the posterior distribution make sense? Explain why *not*; what has gone wrong? If we did not know from the previous part that this output was unrepresentative of the true posterior, how could we try to check? *Hint:* See next part.

(F) Repeat the previous part but now with the initial position at 0.99: `trajectory[1] = 0.99`. In conjunction with the previous part, what does this result tell us?