Part III: Advanced Computation

The remainder of this book delves into more sophisticated models. Before we begin this enterprise, however, we detour to describe methods for computing posterior distributions in hierarchical models. Toward the end of Chapter 5, the algebra required for analytic derivation of posterior distributions became less and less attractive, and that was with a relatively simple model constructed entirely from normal distributions. If we try to solve more complicated problems analytically, the algebra starts to overwhelm the statistical science almost entirely, making the full Bayesian analysis of realistic probability models too cumbersome for most practical applications. Fortunately, a battery of powerful methods has been developed over the past few decades for approximating and simulating from probability distributions. In the next four chapters, we survey some useful simulation methods that we apply in later chapters in the context of specific models. Some of the simpler simulation methods we present here have already been introduced in examples in earlier chapters.

Because the focus of this book is on data analysis rather than computation, we move through the material of Part III briskly, with the intent that it be used as a reference when applying the models discussed in Parts IV and V. We have also attempted to place a variety of useful techniques in the context of a systematic general approach to Bayesian computation. Our general philosophy in computation, as in modeling, is pluralistic, developing approximations using a variety of techniques.

Chapter 10

Introduction to Bayesian computation

Bayesian computation revolves around two steps: computation of the posterior distribution, $p(\theta|y)$, and computation of the posterior predictive distribution, $p(\tilde{y}|y)$. So far we have considered examples where these could be computed analytically in closed form, with simulations performed directly using a combination of preprogrammed routines for standard distributions (normal, gamma, beta, Poisson, and so forth) and numerical computation on grids. For complicated or unusual models or in high dimensions, however, more elaborate algorithms are required to approximate the posterior distribution. Often the most efficient computation can be achieved by combining different algorithms. We discuss these algorithms in Chapters 11–13. This chapter provides a brief summary of statistical procedures to approximately evaluate integrals. The bibliographic note at the end of this chapter suggests other sources.

Normalized and unnormalized densities

We refer to the (multivariate) distribution to be simulated as the *target distribution* and call it $p(\theta|y)$. Unless otherwise noted (in Section 13.10), we assume that $p(\theta|y)$ can be easily computed for any value θ, up to a factor involving only the data y; that is, we assume there is some easily computable function $q(\theta|y)$, an *unnormalized density*, for which $q(\theta|y)/p(\theta|y)$ is a constant that depends only on y. For example, in the usual use of Bayes' theorem, we work with the product $p(\theta)p(y|\theta)$, which is proportional to the posterior density.

Log densities

To avoid computational overflows and underflows, one should compute with the logarithms of posterior densities whenever possible. Exponentiation should be performed only when necessary and as late as possible; for example, in the Metropolis algorithm, the required ratio of two densities (11.1) should be computed as the exponential of the difference of the log-densities.

10.1 Numerical integration

Numerical integration, also called 'quadrature,' refers to methods in which the integral over continuous function is evaluated by computing the value of the function at finite number of points. By increasing the number of points where the function is evaluated, desired accuracy can be obtained. Numerical integration methods can be divided to simulation (stochastic) methods, such as Monte Carlo, and deterministic methods such as many quadrature rule methods.

The posterior expectation of any function $h(\theta)$ is defined as $\mathrm{E}(h(\theta)|y) = \int h(\theta)p(\theta|y)d\theta$, where the integral has as many dimensions as θ. Conversely, we can express any integral over the space of θ as a posterior expectation by defining $h(\theta)$ appropriately. If we have posterior

draws θ^s from $p(\theta|y)$, we can estimate the integral by the sample average, $\frac{1}{S}\sum_{s=1}^{S} h(\theta^s)$. For any finite number of simulation draws, the accuracy of this estimate can be roughly gauged by the standard deviation of the $h(\theta^s)$ values (we discuss this in more detail in Section 10.5). If it is not easy to draw from the posterior distribution, or if the $h(\theta^s)$ values are too variable (so that the sample average is too variable an estimate to be useful), more sampling methods are necessary.

Simulation methods

Simulation (stochastic) methods are based on obtaining random samples θ^s from the desired distribution $p(\theta)$ and estimating the expectation of any function $h(\theta)$,

$$\mathrm{E}(h(\theta)|y) = \int h(\theta)p(\theta|y)d\theta \approx \frac{1}{S}\sum_{s=1}^{S} h(\theta^s). \tag{10.1}$$

The estimate is stochastic depending on generated random numbers, but the accuracy of the simulation can be improved by obtaining more samples. Basic Monte Carlo methods which produce independent samples are discussed in Sections 10.3–10.4 and Markov chain Monte Carlo methods which can better adapt to high-dimensional complex distributions, but produce dependent samples, are discussed in Chapters 11–12. Markov chain Monte Carlo methods have been important in making Bayesian inference practical for generic hierarchical models. Simulation methods can be used for high-dimensional distributions, and there are general algorithms which work for a wide variety of models; where necessary, more efficient computation can be obtained by combining these general ideas with tailored simulation methods, deterministic methods, and distributional approximations.

Deterministic methods

Deterministic numerical integration methods are based on evaluating the integrand $h(\theta)p(\theta|y)$ at selected points θ^s, based on a weighted version of (10.1):

$$\mathrm{E}(h(\theta)|y) = \int h(\theta)p(\theta|y)d\theta \approx \frac{1}{S}\sum_{s=1}^{S} w_s h(\theta^s)p(\theta^s|y),$$

with weight w_s corresponding to the volume of space represented by the point θ^s. More elaborate rules, such as Simpson's, use local polynomials for improved accuracy. Deterministic numerical integration rules typically have lower variance than simulation methods, but selection of locations gets difficult in high dimensions.

The simplest deterministic method is to evaluate the integrand in a grid with equal weights. Grid methods can be made adaptive starting the grid formation from the posterior mode. For an integrand where one part has some specific form as Gaussian, there are specific quadrature rules that can give more accurate estimates with fewer integrand evaluations. Quadrature rules exist for both bounded and unbounded regions.

10.2 Distributional approximations

Distributional (analytic) approximations approximate the posterior with some simpler parametric distribution, from which integrals can be computed directly or by using the approximation as a starting point for simulation-based methods. We have already discussed the normal approximation in Chapter 4, and we consider more advanced approximation methods in Chapter 13.

Crude estimation by ignoring some information

Before developing elaborate approximations or complicated methods for sampling from the target distribution, it is almost always useful to obtain a rough estimate of the location of the target distribution—that is, a point estimate of the parameters in the model—using some simple, noniterative technique. The method for creating this first estimate will vary from problem to problem but typically will involve discarding parts of the model and data to create a simple problem for which convenient parameter estimates can be found.

In a hierarchical model, one can sometimes roughly estimate the main parameters γ by first estimating the hyperparameters ϕ crudely, then using the conditional posterior distribution of $\gamma|\phi, y$. We applied this approach to the rat tumor example in Section 5.1, where crude estimates of the hyperparameters (α, β) were used to obtain initial estimates of the other parameters, θ_j.

For another example, in the educational testing analysis in Section 5.5, the school effects θ_j can be crudely estimated by the data y_j from the individual experiments, and the between-school standard deviation τ can then be estimated crudely by the standard deviation of the eight y_j-values or, to be slightly more sophisticated, the estimate (5.22), restricted to be nonnegative.

When some data are missing, a good way to get started is by simplistically imputing the missing values based on available data. (Ultimately, inferences for the missing data should be included as part of the model; see Chapter 18.)

In addition to creating a starting point for a more exact analysis, crude inferences are useful for comparison with later results—if the rough estimate differs greatly from the results of the full analysis, the latter may well have errors in programming or modeling. Crude estimates are often convenient and reliable because they can be computed using available computer programs.

10.3 Direct simulation and rejection sampling

In simple nonhierarchical Bayesian models, it is often easy to draw from the posterior distribution directly, especially if conjugate prior distributions have been assumed. For more complicated problems, it can help to factor the distribution analytically and simulate it in parts, first sampling from the marginal posterior distribution of the hyperparameters, then drawing the other parameters conditional on the data and the simulated hyperparameters. It is sometimes possible to perform direct simulations and analytic integrations for parts of the larger problem, as was done in the examples of Chapter 5.

Frequently, draws from standard distributions or low-dimensional non-standard distributions are required, either as direct draws from the posterior distribution of the estimand in an easy problem, or as an intermediate step in a more complex problem. Appendix A is a relatively detailed source of advice, algorithms, and procedures specifically relating to a variety of commonly used distributions. In this section, we describe methods of drawing a random sample of size 1, with the understanding that the methods can be repeated to draw larger samples. When obtaining more than one sample, it is often possible to reduce computation time by saving intermediate results such as the Cholesky factor for a fixed multivariate normal distribution.

Direct approximation by calculating at a grid of points

For the simplest discrete approximation, compute the target density, $p(\theta|y)$, at a set of evenly spaced values $\theta_1, \ldots, \theta_N$, that cover a broad range of the parameter space for θ, then approximate the continuous $p(\theta|y)$ by the discrete density at $\theta_1, \ldots, \theta_N$, with probabilities $p(\theta_i|y)/\sum_{j=1}^{N} p(\theta_j|y)$. Because the approximate density must be normalized anyway, this

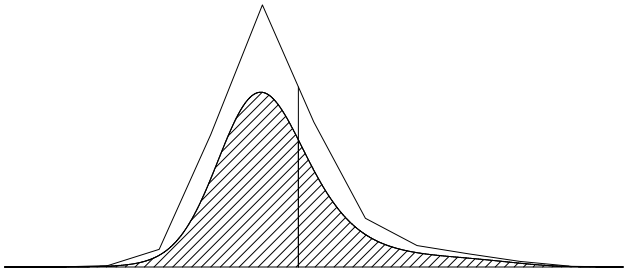

Figure 10.1 *Illustration of rejection sampling. The top curve is an approximation function, $Mg(\theta)$, and the bottom curve is the target density, $p(\theta|y)$. As required, $Mg(\theta) \geq p(\theta|y)$ for all θ. The vertical line indicates a single random draw θ from the density proportional to g. The probability that a sampled draw θ is accepted is the ratio of the height of the lower curve to the height of the higher curve at the value θ.*

method will work just as well using an unnormalized density function, $q(\theta|y)$, in place of $p(\theta|y)$.

Once the grid of density values is computed, a random draw from $p(\theta|y)$ is obtained by drawing a random sample U from the uniform distribution on $[0, 1]$, then transforming by the inverse cdf method (see Section 1.9) to obtain a sample from the discrete approximation. When the points θ_i are spaced closely enough and miss nothing important beyond their boundaries, this method works well. The discrete approximation is more difficult to use in higher-dimensional multivariate problems, where computing at every point in a dense multidimensional grid becomes prohibitively expensive.

Simulating from predictive distributions

Once we have a sample from the posterior distribution, $p(\theta|y)$, it is typically easy to draw from the predictive distribution of unobserved or future data, $\tilde{y}$. For each draw of θ from the posterior distribution, just draw one $\tilde{y}$ from the predictive distribution, $p(\tilde{y}|\theta)$. The set of simulated $\tilde{y}$'s from all the θ's characterizes the posterior predictive distribution. Posterior predictive distributions are crucial to the model-checking approach described in Chapter 6.

Rejection sampling

Suppose we want to obtain a single random draw from a density $p(\theta|y)$, or perhaps an unnormalized density $q(\theta|y)$ (with $p(\theta|y) = q(\theta|y)/\int q(\theta|y)d\theta$). In the following description we use p to represent the target distribution, but we could just as well work with the unnormalized form q instead. To perform *rejection sampling* we require a positive function $g(\theta)$ defined for all θ for which $p(\theta|y) > 0$ that has the following properties:

- We can draw from the probability density proportional to g. It is *not* required that $g(\theta)$ integrate to 1, but $g(\theta)$ must have a finite integral.
- The *importance ratio* $\frac{p(\theta|y)}{g(\theta)}$ must have a known bound; that is, there must be some known constant M for which $\frac{p(\theta|y)}{g(\theta)} \leq M$ for all θ.

The rejection sampling algorithm proceeds in two steps:

1. Sample θ at random from the probability density proportional to $g(\theta)$.
2. With probability $\frac{p(\theta|y)}{Mg(\theta)}$, *accept* θ as a draw from p. If the drawn θ is rejected, return to step 1.

Figure 10.1 illustrates rejection sampling. An accepted θ has the correct distribution, $p(\theta|y)$; that is, the distribution of drawn θ, conditional on it being accepted, is $p(\theta|y)$ (see Exercise

10.4). The boundedness condition is necessary so that the probability in step 2 is not greater than 1.

A good approximate density $g(\theta)$ for rejection sampling should be roughly proportional to $p(\theta|y)$ (considered as a function of θ). The ideal situation is $g \propto p$, in which case, with a suitable value of M, we can accept every draw with probability 1. When g is not nearly proportional to p, the bound M must be set so large that almost all draws obtained in step 1 will be rejected in step 2. A virtue of rejection sampling is that it is self-monitoring—if the method is not working efficiently, few simulated draws will be accepted.

The function $g(\theta)$ is chosen to approximate $p(\theta|y)$ and so in general will depend on y. We do not use the notation $g(\theta, y)$ or $g(\theta|y)$, however, because in practice we will be considering approximations to one posterior distribution at a time, and the functional dependence of g on y is not of interest.

Rejection sampling is used in some fast methods for sampling from standard univariate distributions. It is also often used for generic truncated multivariate distributions, if the proportion of the density mass in the truncated part is not close to 1.

10.4 Importance sampling

Importance sampling is a method, related to rejection sampling and a precursor to the Metropolis algorithm (discussed in the next chapter), that is used for computing expectations using a random sample drawn from an approximation to the target distribution. Suppose we are interested in $\mathrm{E}(h(\theta)|y)$, but we cannot generate random draws of θ from $p(\theta|y)$ and thus cannot evaluate the integral by a simple average of simulated values.

If $g(\theta)$ is a probability density from which we can generate random draws, then we can write,

$$\mathrm{E}(h(\theta|y)) = \frac{\int h(\theta)q(\theta|y)d\theta}{\int q(\theta|y)d\theta} = \frac{\int [h(\theta)q(\theta|y)/g(\theta)]\, g(\theta)d\theta}{\int [q(\theta|y)/g(\theta)]\, g(\theta)d\theta}, \tag{10.2}$$

which can be estimated using S draws $\theta^1, \ldots, \theta^S$ from $g(\theta)$ by the expression,

$$\frac{\frac{1}{S}\sum_{s=1}^{S} h(\theta^s)w(\theta^s)}{\frac{1}{S}\sum_{s=1}^{S} w(\theta^s)}, \tag{10.3}$$

where the factors

$$w(\theta^s) = \frac{q(\theta^s|y)}{g(\theta^s)}$$

are called *importance ratios* or *importance weights*. Recall that q is our general notation for unnormalized densities; that is, $q(\theta|y)$ equals $p(\theta|y)$ times some factor that does not depend on θ.

It is generally advisable to use the same set of random draws for both the numerator and denominator of (10.3) in order to reduce the sampling error in the estimate.

If $g(\theta)$ can be chosen such that $\frac{hq}{g}$ is roughly constant, then fairly precise estimates of the integral can be obtained. Importance sampling is not a useful method if the importance ratios vary substantially. The worst possible scenario occurs when the importance ratios are small with high probability but with a low probability are huge, which happens, for example, if hq has wide tails compared to g, as a function of θ.

Accuracy and efficiency of importance sampling estimates

In general, without some form of mathematical analysis of the exact and approximate densities, there is always the realistic possibility that we have missed some extremely large but rare importance weights. However, it may help to examine the distribution of sampled

importance weights to discover possible problems. It can help to examine a histogram of the logarithms of the largest importance ratios: estimates will often be poor if the largest ratios are too large relative to the average. In contrast, we do not have to worry about the behavior of small importance ratios, because they have little influence on equation (10.2). If the variance of the weights is finite, the effective sample size can be estimated using an approximation,

$$S_{\text{eff}} = \frac{1}{\sum_{s=1}^{S}(\tilde{w}(\theta^s))^2}, \tag{10.4}$$

where $\tilde{w}(\theta^s)$ are normalized weights; that is, $\tilde{w}(\theta^s) = w(\theta^s)S/\sum_{s'=1}^{S} w(\theta^{s'})$. The effective sample size S_{eff} is small if there are few extremely high weights which would unduly influence the distribution. If the distribution has occasional very large weights, however, this estimate is itself noisy; it can thus be taken as no more than a rough guide.

Importance resampling

To obtain independent samples with equal weights, it is possible to use *importance resampling* (also called sampling-importance resampling or SIR).

Once S draws, $\theta^1, \ldots, \theta^S$, from the approximate distribution g have been sampled, a sample of $k < S$ draws can be simulated as follows.

1. Sample a value θ from the set $\{\theta^1, \ldots, \theta^S\}$, where the probability of sampling each θ^s is proportional to the weight, $w(\theta^s) = \frac{q(\theta^s|y)}{g(\theta^s)}$.
2. Sample a second value using the same procedure, but excluding the already sampled value from the set.
3. Repeatedly sample without replacement $k-2$ more times.

Why sample without replacement? If the importance weights are moderate, sampling with and without replacement gives similar results. Now consider a bad case, with a few large weights and many small weights. Sampling with replacement will pick the same few values of θ repeatedly; in contrast, sampling without replacement yields a more desirable intermediate approximation somewhere between the starting and target densities. For other purposes, sampling with replacement could be superior.

Uses of importance sampling in Bayesian computation

Importance sampling can be used to improve analytic posterior approximations as described in Chapter 13. If importance sampling does not yield an accurate approximation, then importance resampling can still be helpful for obtaining starting points for an iterative simulation of the posterior distribution, as described in Chapter 11.

Importance (re)sampling can also be useful when considering mild changes in the posterior distribution, for example replacing the normal distribution by a t in the 8 schools model or when computing leave-one-out cross-validation. The idea in this case is to treat the original posterior distribution as an approximation to the modified posterior distribution.

A good way to develop an understanding of importance sampling is to program simulations for simple examples, such as using a t_3 distribution as an approximation to the normal (good practice) or vice versa (bad practice); see Exercises 10.6 and 10.7. The approximating distribution g in importance sampling should cover all the important regions of the target distribution.

10.5 How many simulation draws are needed?

Bayesian inferences are usually most conveniently summarized by random draws from the posterior distribution of the model parameters. Percentiles of the posterior distribution of univariate estimands can be reported to convey the shape of the distribution. For example, reporting the 2.5%, 25%, 50%, 75%, and 97.5% points of the sampled distribution of an estimand provides a 50% and a 95% posterior interval and also conveys skewness in its marginal posterior density. Scatterplots of simulations, contour plots of density functions, or more sophisticated graphical techniques can also be used to examine the posterior distribution in two or three dimensions. Quantities of interest can be defined in terms of the parameters (for example, LD50 in the bioassay example in Section 3.7) or of parameters and data.

We also use posterior simulations to make inferences about predictive quantities. Given each draw θ^s, we can sample any predictive quantity, $\tilde{y}^s \sim p(\tilde{y}|\theta^s)$ or, for a regression model, $\tilde{y}^s \sim p(\tilde{y}|\tilde{X}, \theta^s)$. Posterior inferences and probability calculations can then be performed for each predictive quantity using the S simulations (for example, the predicted probability of Bill Clinton winning each state in 1992, as displayed in Figure 6.1 on page 143).

Finally, given each simulation θ^s, we can simulate a replicated dataset $y^{\text{rep}\,s}$. As described in Chapter 6, we can then check the model by comparing the data to these posterior predictive replications.

Our goal in Bayesian computation is to obtain a set of independent draws θ^s, $s = 1, \ldots, S$, from the posterior distribution, with enough draws S so that quantities of interest can be estimated with reasonable accuracy. For most examples, $S = 100$ independent draws are enough for reasonable posterior summaries. We can see this by considering a scalar parameter θ with an approximately normal posterior distribution (see Chapter 4) with mean μ_θ and standard deviation σ_θ. We assume these cannot be calculated analytically and instead are estimated from the mean $\bar{\theta}$ and standard deviation s_θ of the S simulation draws. The posterior mean is then estimated to an accuracy of approximately $s_\theta/\sqrt{S}$. The total standard deviation of the computational parameter estimate (including *Monte Carlo error*, the uncertainty contributed by having only a finite number of simulation draws) is then $s_\theta\sqrt{1+1/S}$. For $S = 100$, the factor $\sqrt{1+1/S}$ is 1.005, implying that Monte Carlo error adds almost nothing to the uncertainty coming from actual posterior variance. However, it can be convenient to have more than 100 simulations just so that the numerical summaries are more stable, even if this stability typically confers no important practical advantage.

For some posterior inferences, more simulation draws are needed to obtain desired precisions. For example, posterior probabilities are estimated to a standard deviation of $\sqrt{p(1-p)/S}$, so that $S = 100$ simulations allow estimation of a probability near 0.5 to an accuracy of 5%. $S = 2500$ simulations are needed to estimate to an accuracy of 1%. Even more simulation draws are needed to compute the posterior probability of rare events, unless analytic methods are used to assist the computations.

Example. Educational testing experiments

We illustrate with the hierarchical model fitted to the data from the 8 schools as described in Section 5.5. First consider inference for a particular parameter, for example θ_1, the estimated effect of coaching in school A. Table 5.3 shows that from 200 simulation draws, our posterior median estimate was 10, with a 50% interval of $[7, 16]$ and a 95% interval of $[-2, 31]$. Repeating the computation, another 200 draws gave a posterior median of 9, with a 50% interval of $[6, 14]$ and a 95% interval of $[-4, 32]$. These intervals differ slightly but convey the same general information about θ_1. From $S = 10{,}000$ simulation draws, the median is 10, the 50% interval is $[6, 15]$, and the 95% interval is $[-2, 31]$. In practice, these are no different from either of the summaries obtained from 200 draws.

We now consider some posterior probability statements. Our original 200 simulations gave us an estimate of 0.73 for the posterior probability $\Pr(\theta_1 > \theta_3|y)$, the probability that the effect is larger in school A than in school C (see the end of Section 5.5). This probability is estimated to an accuracy of $\sqrt{0.73(1-0.73)/200} = 0.03$, which is good enough in this example.

How about a rarer event, such as the probability that the effect in School A is greater than 50 points? None of our 200 simulations θ_1^s exceeds 50, so the simple estimate of the probability is that it is zero (or less than 1/200). When we simulate $S =$ 10,000 draws, we find 3 of the draws to have $\theta_1 > 50$, which yields a crude estimated probability of 0.0003.

An alternative way to compute this probability is semi-analytically. Given μ and τ, the effect in school A has a normal posterior distribution, $p(\theta_1|\mu,\tau,y) = \mathrm{N}(\hat{\theta}_1, V_1)$, where this mean and variance depend on y_1, μ, and τ (see (5.17) on page 116). The conditional probability that θ_1 exceeds 50 is then $\Pr(\theta_1 > 50|\mu,\tau,y) = \Phi((\hat{\theta}_1 - 50)/\sqrt{V_1})$, and we can estimate the unconditional posterior probability $\Pr(\theta_1 > 50|y)$ as the average of these normal probabilities as computed for each simulation draw (μ^s, τ^s). Using this approach, $S = 200$ draws are sufficient for a reasonably accurate estimate.

In general, fewer simulations are needed to estimate posterior medians of parameters, probabilities near 0.5, and low-dimensional summaries than extreme quantiles, posterior means, probabilities of rare events, and higher-dimensional summaries. In most of the examples in this book, we use a moderate number of simulation draws (typically 100 to 2000) as a way of emphasizing that applied inferences do not typically require a high level of simulation accuracy.

10.6 Computing environments

Programs exist for full Bayesian inference for commonly used models such as hierarchical linear and logistic regression and some nonparametric models. These implementations use various combinations of the Bayesian computation algorithms discussed in the following chapters.

We see (at least) four reasons for wanting an automatic and general program for fitting Bayesian models. First, many applied statisticians and subject-matter researchers would like to fit Bayesian models but do not have the mathematical, statistical, and computer skills to program the inferential steps themselves. Economists and political scientists can run regressions with a single line of code or a click on a menu, epidemiologists can do logistic regression, sociologists can fit structural equation models, psychologists can fit analysis of variance, and education researchers can fit hierarchical linear models. We would like all these people to be able to fit Bayesian models (which include all those previously mentioned as special cases but also allow for generalizations such as robust error models, mixture distributions, and various arbitrary functional forms, not to mention a framework for including prior information).

A second use for a general Bayesian package is for teaching. Students can first learn to do inference automatically, focusing on the structure of their models rather than on computation, learning the algebra and computing later. A deeper understanding *is* useful—if we did not believe so, we would not have written this book. Ultimately it is helpful to learn what lies behind the inference and computation because one way we understand a model is by comparing to similar models that are slightly simpler or more complicated, and one way we understand the process of model fitting is by seeing it as a map from data and assumptions to inferences. Even when using a black box or 'inference engine,' we often want

to go back and see where the substantively important features of our posterior distribution came from.

A third motivation for writing an automatic model-fitting package is as a programming environment for implementing new models, and for practitioners who could program their own models to be able to focus on more important statistical issues.

Finally, a fourth potential benefit of a general Bayesian program is that it can be faster than custom code. There is an economy of scale. Because the automatic program will be used so many times, it can make sense to optimize it in various ways, implement it in parallel, and include algorithms that require more coding effort but are faster in hard problems.

That said, no program can be truly general. Any such software should be open and accessible, with places where the (sophisticated) user can alter the program or 'hold its hand' to ensure that it does what it is supposed to do.

The Bugs family of programs

During the 1990s and early 2000s, a group of statisticians and programmers developed Bugs (Bayesian inference using Gibbs sampling), a general-purpose program in which a user could supply data and specify a statistical model using a convenient language not much different from the mathematical notation used in probability theory, and then the program used a combination of Gibbs sampling, the Metropolis algorithm, and slice sampling (algorithms which are described in the following chapters) to obtain samples from the posterior distribution. When run for a sufficiently long time, Bugs could provide inference for an essentially unlimited variety of models. Instead of models being chosen from a list or menu of preprogrammed options, Bugs models could be put together in arbitrary ways using a large set of probability distributions, much in the way that we construct models in this book.

The most important limitations of Bugs have been computational. The program excels with complicated models for small datasets but can be slow with large datasets and multivariate structures. Bugs works by updating one scalar parameter at a time (following the ideas of Gibbs sampling, as discussed in the following chapter), which results in slow convergence when parameters are strongly dependent, as can occur in hierarchical models.

Stan

We have recently developed an open-source program, Stan (named after Stanislaw Ulam, a mathematician who was one of the inventors of the Monte Carlo method) that has similar functionality as Bugs but uses a more complicated simulation algorithm, Hamiltonian Monte Carlo (see Section 12.4). Stan is written in C++ and has been designed to be easily extendable, both in allowing improvements to the updating algorithm and in being open to the development of new models. We now develop and fit our Bayesian models in Stan, using its Bugs-like modeling language and improving it as necessary to fit more complicated models and larger problems. Stan is intended to serve both as an automatic program for general users and as a programming environment for the development of new simulation methods. We discuss Stan further in Section 12.6 and Appendix C.

Other Bayesian software

Following the success of Bugs, many research groups have developed general tools for fitting Bayesian models. These include mcsim (a C program that implements Gibbs and Metropolis for differential equation systems such as the toxicology model described in Section 19.2), PyMC (a suite of routines in the open-source language Python), HBC (developed for discrete-parameter models in computational linguistics). These and other programs have

been developed by individuals or communities of users who have wanted to fit particular models and have found it most effective to do this by writing general implementations. In addition various commercial programs are under development that fit Bayesian models with various degrees of generality.

10.7 Debugging Bayesian computing

Debugging using fake data

Our usual approach for building confidence in our posterior inferences is to fit different versions of the desired model, noticing when the inferences change unexpectedly. Section 10.2 discusses crude inferences from simplified models that typically ignore some structure in the data.

Within the computation of any particular model, we check convergence by running parallel simulations from different starting points, checking that they mix and converge to the same estimated posterior distribution (see Section 11.4). This can be seen as a form of debugging of the individual simulated sequences.

When a model is particularly complicated, or its inferences are unexpected enough to be not necessarily believable, one can perform more elaborate debugging using fake data. The basic approach is:

1. Pick a reasonable value for the 'true' parameter vector θ. Strictly speaking, this value should be a random draw from the prior distribution, but if the prior distribution is noninformative, then any reasonable value of θ should work.
2. If the model is hierarchical (as it generally will be), then perform the above step by picking reasonable values for the hyperparameters, then drawing the other parameters from the prior distribution conditional on the specified hyperparameters.
3. Simulate a large fake dataset y^{fake} from the data distribution $p(y|\theta)$.
4. Perform posterior inference about θ from $p(\theta|y^{\text{fake}})$.
5. Compare the posterior inferences to the 'true' θ from step 1 or 2. For instance, for any element of θ, there should be a 50% probability that its 50% posterior interval contains the truth.

Formally, this procedure requires that the model has proper prior distributions and that the frequency evaluations be averaged over many values of the 'true' θ, drawn independently from the prior distribution in step 1 above. In practice, however, the debugging procedure can be useful with just a single reasonable choice of θ in the first step. If the model does not produce reasonable inferences with θ set to a reasonable value, then there is probably something wrong, either in the computation or in the model itself.

Inference from a single fake dataset can be revealing for debugging purposes, if the true value of θ is far outside the computed posterior distribution. If the dimensionality of θ is large (as can easily happen with hierarchical models), we can go further and compute debugging checks such as the proportion of the 50% intervals that contain the true value.

To check that inferences are correct on average, one can create a 'residual plot' as follows. For each scalar parameter θ_j, define the predicted value as the average of the posterior simulations of θ_j, and the error as the true θ_j (as specified or simulated in step 1 or 2 above) minus the predicted value. If the model is computed correctly, the errors should have zero mean, and we can diagnose problems by plotting errors vs. predicted values, with one dot per parameter.

For models with only a few parameters, one can get the same effect by performing many fake-data simulations, resampling a new 'true' vector θ and a new fake dataset y^{fake} each time, and then checking that the errors have zero mean and the correct interval coverage, on average.

Model checking and convergence checking as debugging

Finally, the techniques for model checking and comparison described in Chapters 6 and 7, and the techniques for checking for poor convergence of iterative simulations, which we describe in Section 11.4, can also be interpreted as methods for debugging.

In practice, when a model grossly misfits the data, or when a histogram or scatterplot or other display of replicated data looks weird, it is often because of a computing error. These errors can be as simple as forgetting to recode discrete responses (for example, $1 =$ Yes, $0 =$ No, $-9 =$ Don't Know) or misspelling a regression predictor, or as subtle as a miscomputed probability ratio in a Metropolis updating step (see Section 11.2), but typically they show up as predictions that do not make sense or do not fit the data. Similarly, poor convergence of an iterative simulation algorithm can sometimes occur from programming errors or conceptual errors in the model.

When posterior inferences from a fitted model seem wrong, it is sometimes unclear if there is a bug in the program or a fundamental problem with the model itself. At this point, a useful conceptual and computational strategy is to simplify—to remove parameters from the model, or to give them fixed values or highly informative prior distributions, or to separately analyze data from different sources (that is, to un-link a hierarchical model). These computations can be performed in steps, for example first removing a parameter from the model, then setting it equal to a null value (for example, zero) just to check that adding it into the program has no effect, then fixing it at a reasonable nonzero value, then assigning it a precise prior distribution, then allowing it to be estimated more fully from the data. Model building is a gradual process, and we often find ourselves going back and forth between simpler and more complicated models, both for conceptual and computational reasons.

10.8 Bibliographic note

Excellent general books on simulation from a statistical perspective are Ripley (1987), and Gentle (2003), which cover two topics that we do not address in this chapter: creating uniformly distributed (pseudo)random numbers and simulating from standard distributions (on the latter, see our Appendix A for more details). Hammersley and Handscomb (1964) is a classic reference on simulation. Thisted (1988) is a general book on statistical computation that discusses many optimization and simulation techniques. Robert and Casella (2004) cover simulation algorithms from a variety of statistical perspectives.

For further information on numerical integration techniques, see Press et al. (1986); a review of the application of these techniques to Bayesian inference is provided by Smith et al. (1985), and O'Hagan and Forster (2004).

Bayesian quadrature methods using Gaussian process priors have been proposed by O'Hagan (1991) and Rasmussen and Ghahramani (2003). Adaptive grid sampling has been presented, for example, by Rue, Martino, and Chopin (2009).

Importance sampling is a relatively old idea in numerical computation; for some early references, see Hammersley and Handscomb (1964). Geweke (1989) is a pre-Gibbs sampler discussion in the context of Bayesian computation; also see Wakefield, Gelfand, and Smith (1991). Chapters 2–4 of Liu (2001) discuss importance sampling in the context of Markov chain simulation algorithms. Gelfand, Dey, and Chang (1992) proposed importance sampling for fast leave-one-out cross-validation. Kong, Liu, and Wong (1996) propose a method for estimating the reliability of importance sampling using approximation of the variance of importance weights. Skare, Bolviken, and Holden (2003) propose improved importance sampling using modified weights which reduce the bias of the estimate.

Importance resampling was introduced by Rubin (1987b), and an accessible exposition is given by Smith and Gelfand (1992). Skare, Bolviken, and Holden (2003) discuss why it

is best to draw importance resamples ($k < S$) without replacement and they propose an improved algorithm that uses modified weights. When $k = S$ draws are required, Kitagawa (1996) presents stratified and deterministic resampling, and Liu (2001) presents residual resampling; these methods all have smaller variance than simple random resampling.

Kass et al. (1998) discuss many practical issues in Bayesian simulation. Gelman and Hill (2007, chapter 8) and Cook, Gelman, and Rubin (2006) show how to check Bayesian computations using fake-data simulation. Kerman and Gelman (2006, 2007) discuss some ways in which R can be modified to allow more direct manipulation of random variable objects and Bayesian inferences.

Information about Bugs appears at Spiegelhalter et al. (1994, 2003) and Plummer (2003), respectively. The article by Lunn et al. (2009) and ensuing discussion give a sense of the scope of the Bugs project. Stan is discussed further in Section 12.6 and Appendix C of this book, as well as at Stan Development Team (2012). Several other efforts have been undertaken to develop Bayesian inference tools for particular classes of model, for example Daume (2008).

10.9 Exercises

The exercises in Part III focus on computational details. Data analysis exercises using the methods described in this part of the book appear in the appropriate chapters in Parts IV and V.

1. Number of simulation draws: Suppose the scalar variable θ is approximately normally distributed in a posterior distribution that is summarized by n independent simulation draws. How large does n have to be so that the 2.5% and 97.5% quantiles of θ are specified to an accuracy of $0.1\,\text{sd}(\theta|y)$?
   (a) Figure this out mathematically, without using simulation.
   (b) Check your answer using simulation and show your results.
2. Number of simulation draws: suppose you are interested in inference for the parameter θ_1 in a multivariate posterior distribution, $p(\theta|y)$. You draw 100 independent values θ from the posterior distribution of θ and find that the posterior density for θ_1 is approximately normal with mean of about 8 and standard deviation of about 4.
   (a) Using the average of the 100 draws of θ_1 to estimate the posterior mean, $\text{E}(\theta_1|y)$, what is the approximate standard deviation due to simulation variability?
   (b) About how many simulation draws would you need to reduce the simulation standard deviation of the posterior mean to 0.1 (thus justifying the presentation of results to one decimal place)?
   (c) A more usual summary of the posterior distribution of θ_1 is a 95% central posterior interval. Based on the data from 100 draws, what are the approximate simulation standard deviations of the estimated 2.5% and 97.5% quantiles of the posterior distribution? (Recall that the posterior density is approximately normal.)
   (d) About how many simulation draws would you need to reduce the simulation standard deviations of the 2.5% and 97.5% quantiles to 0.1?
   (e) In the eight-schools example of Section 5.5, we simulated 200 posterior draws. What are the approximate simulation standard deviations of the 2.5% and 97.5% quantiles for school A in Table 5.3?
   (f) Why was it not necessary, in practice, to simulate more than 200 draws for the SAT coaching example?
3. Posterior computations for the binomial model: suppose $y_1 \sim \text{Bin}(n_1, p_1)$ is the number of successfully treated patients under an experimental new drug, and $y_2 \sim \text{Bin}(n_2, p_2)$

is the number of successfully treated patients under the standard treatment. Assume that y_1 and y_2 are independent and assume independent beta prior densities for the two probabilities of success. Let $n_1 = 10, y_1 = 6$, and $n_2 = 20, y_2 = 10$. Repeat the following for several different beta prior specifications.

(a) Use simulation to find a 95% posterior interval for $p_1 - p_2$ and the posterior probability that $p_1 > p_2$.

(b) Numerically integrate to estimate the posterior probability that $p_1 > p_2$.

4. Rejection sampling:

(a) Prove that rejection sampling gives draws from $p(\theta|y)$.

(b) Why is the boundedness condition on $p(\theta|y)/q(\theta)$ necessary for rejection sampling?

5. Rejection sampling and importance sampling: Consider the model, $y_j \sim \text{Binomial}(n_j, \theta_j)$, where $\theta_j = \text{logit}^{-1}(\alpha + \beta x_j)$, for $j = 1, \ldots, J$, and with independent prior distributions, $\alpha \sim t_4(0, 2^2)$ and $\beta \sim t_4(0, 1)$. Suppose $J = 10$, the x_j values are randomly drawn from a $\text{U}(0, 1)$ distribution, and $n_j \sim \text{Poisson}^+(5)$, where Poisson^+ is the Poisson distribution restricted to positive values.

(a) Sample a dataset at random from the model.

(b) Use rejection sampling to get 1000 independent posterior draws from (α, β).

(c) Approximate the posterior density for (α, β) by a normal centered at the posterior mode with covariance matrix fit to the curvature at the mode.

(d) Take 1000 draws from the two-dimensional t_4 distribution with that center and scale matrix and use importance sampling to estimate $\text{E}(\alpha|y)$ and $\text{E}(\beta|y)$

(e) Compute an estimate of effective sample size for importance sampling using (10.4) on page 266.

6. Importance sampling when the importance weights are well behaved: consider a univariate posterior distribution, $p(\theta|y)$, which we wish to approximate and then calculate moments of, using importance sampling from an unnormalized density, $g(\theta)$. Suppose the posterior distribution is normal, and the approximation is t_3 with mode and curvature matched to the posterior density.

(a) Draw a sample of size $S = 100$ from the approximate density and compute the importance ratios. Plot a histogram of the log importance ratios.

(b) Estimate $\text{E}(\theta|y)$ and $\text{var}(\theta|y)$ using importance sampling. Compare to the true values.

(c) Repeat (a) and (b) for $S = 10{,}000$.

(d) Using the sample obtained in (c), compute an estimate of effective sample size using (10.4) on page 266.

7. Importance sampling when the importance weights are too variable: repeat the previous exercise, but with a t_3 posterior distribution and a normal approximation. Explain why the estimates of $\text{var}(\theta|y)$ are systematically too low.

8. Importance resampling with and without replacement:

(a) Consider the bioassay example introduced in Section 3.7. Use importance resampling to approximate draws from the posterior distribution of the parameters (α, β), using the normal approximation of Section 4.1 as the starting distribution. Sample $S = 10{,}000$ from the approximate distribution, and resample without replacement $k = 1000$ samples. Compare your simulations of (α, β) to Figure 3.3b and discuss any discrepancies.

(b) Comment on the distribution of the simulated importance ratios.

(c) Repeat part (a) using importance sampling with replacement. Discuss how the results differ.

Chapter 11

Basics of Markov chain simulation

Many clever methods have been devised for constructing and sampling from arbitrary posterior distributions. Markov chain simulation (also called *Markov chain Monte Carlo*, or MCMC) is a general method based on drawing values of θ from approximate distributions and then correcting those draws to better approximate the target posterior distribution, $p(\theta|y)$. The sampling is done sequentially, with the distribution of the sampled draws depending on the last value drawn; hence, the draws form a Markov chain. (As defined in probability theory, a *Markov chain* is a sequence of random variables $\theta^1, \theta^2, \ldots$, for which, for any t, the distribution of θ^t given all previous θ's depends only on the most recent value, θ^{t-1}.) The key to the method's success, however, is not the Markov property but rather that the approximate distributions are improved at each step in the simulation, in the sense of converging to the target distribution. As we shall see in Section 11.2, the Markov property is helpful in proving this convergence.

Figure 11.1 illustrates a simple example of a Markov chain simulation—in this case, a Metropolis algorithm (see Section 11.2) in which θ is a vector with only two components, with a bivariate unit normal posterior distribution, $\theta \sim \mathrm{N}(0, I)$. First consider Figure 11.1a, which portrays the early stages of the simulation. The space of the figure represents the range of possible values of the multivariate parameter, θ, and each of the five jagged lines represents the early path of a random walk starting near the center or the extremes of the target distribution and jumping through the distribution according to an appropriate sequence of random iterations. Figure 11.1b represents the mature stage of the same Markov chain simulation, in which the simulated random walks have each traced a path throughout the space of θ, with a common stationary distribution that is equal to the target distribution. We can then perform inferences about θ using points from the second halves of the Markov chains we have simulated, as displayed in Figure 11.1c.

In our applications of Markov chain simulation, we create several independent sequences; each sequence, $\theta^1, \theta^2, \theta^3, \ldots$, is produced by starting at some point θ^0 and then, for each t, drawing θ^t from a *transition distribution*, $T_t(\theta^t|\theta^{t-1})$ that depends on the previous draw, θ^{t-1}. As we shall see in the discussion of combining the Gibbs sampler and Metropolis sampling in Section 11.3, it is often convenient to allow the transition distribution to depend on the iteration number t; hence the notation T_t. The transition probability distributions must be constructed so that the Markov chain converges to a unique stationary distribution that is the posterior distribution, $p(\theta|y)$.

Markov chain simulation is used when it is not possible (or not computationally efficient) to sample θ directly from $p(\theta|y)$; instead we sample *iteratively* in such a way that at each step of the process we expect to draw from a distribution that becomes closer to $p(\theta|y)$. For a wide class of problems (including posterior distributions for many hierarchical models), this appears to be the easiest way to get reliable results. In addition, Markov chain and other iterative simulation methods have many applications outside Bayesian statistics, such as optimization, that we do not discuss here.

The key to Markov chain simulation is to create a Markov process whose stationary dis-

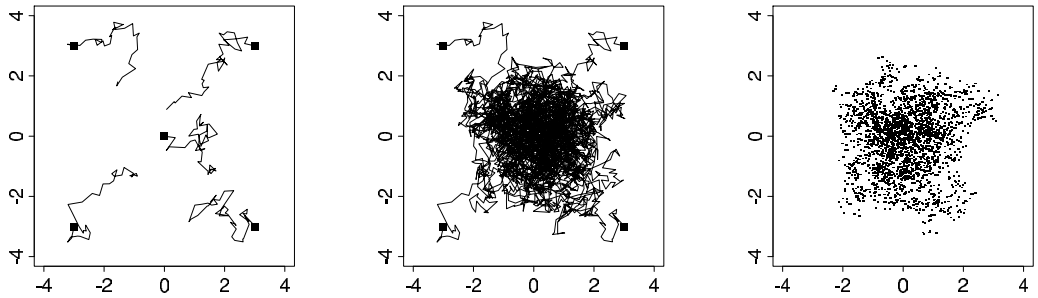

Figure 11.1 *Five independent sequences of a Markov chain simulation for the bivariate unit normal distribution, with overdispersed starting points indicated by solid squares. (a) After 50 iterations, the sequences are still far from convergence. (b) After 1000 iterations, the sequences are nearer to convergence. Figure (c) shows the iterates from the second halves of the sequences; these represent a set of (correlated) draws from the target distribution. The points in Figure (c) have been jittered so that steps in which the random walks stood still are not hidden. The simulation is a Metropolis algorithm described in the example on page 278, with a jumping rule that has purposely been chosen to be inefficient so that the chains will move slowly and their random-walk-like aspect will be apparent.*

tribution is the specified $p(\theta|y)$ and to run the simulation long enough that the distribution of the current draws is close enough to this stationary distribution. For any specific $p(\theta|y)$, or unnormalized density $q(\theta|y)$, a variety of Markov chains with the desired property can be constructed, as we demonstrate in Sections 11.1–11.3.

Once the simulation algorithm has been implemented and the simulations drawn, it is absolutely necessary to check the convergence of the simulated sequences; for example, the simulations of Figure 11.1a are far from convergence and are not close to the target distribution. We discuss how to check convergence in Section 11.4, and in Section 11.5 we construct an expression for the effective number of simulation draws for a correlated sample. If convergence is painfully slow, the algorithm should be altered, as discussed in the next chapter.

This chapter introduces the basic Markov chain simulation methods—the Gibbs sampler and the Metropolis-Hastings algorithm—in the context of our general computing approach based on successive approximation. We sketch a proof of the convergence of Markov chain simulation algorithms and present a method for monitoring the convergence in practice. We illustrate these methods in Section 11.6 for a hierarchical normal model. For most of this chapter we consider simple and familiar (even trivial) examples in order to focus on the principles of iterative simulation methods as they are used for posterior simulation. Many examples of these methods appear in the recent statistical literature and also in the later parts this book. Appendix C shows the details of implementation in the computer languages R and Stan for the educational testing example from Chapter 5.

11.1 Gibbs sampler

A particular Markov chain algorithm that has been found useful in many multidimensional problems is the *Gibbs sampler*, also called alternating conditional sampling, which is defined in terms of subvectors of θ. Suppose the parameter vector θ has been divided into d components or subvectors, $\theta = (\theta_1, \ldots, \theta_d)$. Each iteration of the Gibbs sampler cycles through the subvectors of θ, drawing each subset conditional on the value of all the others. There are thus d steps in iteration t. At each iteration t, an ordering of the d subvectors of θ is chosen and, in turn, each θ_j^t is sampled from the conditional distribution given all the

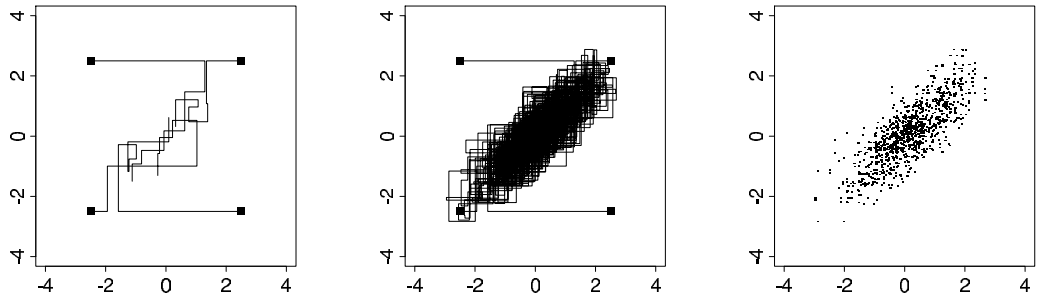

Figure 11.2 *Four independent sequences of the Gibbs sampler for a bivariate normal distribution with correlation $\rho = 0.8$, with overdispersed starting points indicated by solid squares. (a) First 10 iterations, showing the componentwise updating of the Gibbs iterations. (b) After 500 iterations, the sequences have reached approximate convergence. Figure (c) shows the points from the second halves of the sequences, representing a set of correlated draws from the target distribution.*

other components of θ:

$$p(\theta_j|\theta_{-j}^{t-1}, y),$$

where θ_{-j}^{t-1} represents all the components of θ, except for θ_j, at their current values:

$$\theta_{-j}^{t-1} = (\theta_1^t, \ldots, \theta_{j-1}^t, \theta_{j+1}^{t-1}, \ldots, \theta_d^{t-1}).$$

Thus, each subvector θ_j is updated conditional on the latest values of the other components of θ, which are the iteration t values for the components already updated and the iteration $t-1$ values for the others.

For many problems involving standard statistical models, it is possible to sample directly from most or all of the conditional posterior distributions of the parameters. We typically construct models using a sequence of conditional probability distributions, as in the hierarchical models of Chapter 5. It is often the case that the conditional distributions in such models are conjugate distributions that provide for easy simulation. We present an example for the hierarchical normal model at the end of this chapter and another detailed example for a normal-mixture model in Section 22.2. Here, we illustrate the workings of the Gibbs sampler with a simple example.

Example. Bivariate normal distribution

Consider a single observation (y_1, y_2) from a bivariate normally distributed population with unknown mean $\theta = (\theta_1, \theta_2)$ and known covariance matrix $\left(\begin{smallmatrix} 1 & \rho \\ \rho & 1 \end{smallmatrix}\right)$. With a uniform prior distribution on θ, the posterior distribution is

$$\begin{pmatrix} \theta_1 \\ \theta_2 \end{pmatrix} \Bigg| \, y \sim \mathrm{N}\left(\begin{pmatrix} y_1 \\ y_2 \end{pmatrix}, \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix} \right).$$

Although it is simple to draw directly from the joint posterior distribution of (θ_1, θ_2), for the purpose of exposition we demonstrate the Gibbs sampler here. We need the conditional posterior distributions, which, from the properties of the multivariate normal distribution (either equation (A.1) or (A.2) on page 580), are

$$\begin{aligned} \theta_1|\theta_2, y &\sim \mathrm{N}(y_1 + \rho(\theta_2 - y_2),\, 1 - \rho^2) \\ \theta_2|\theta_1, y &\sim \mathrm{N}(y_2 + \rho(\theta_1 - y_1),\, 1 - \rho^2). \end{aligned}$$

The Gibbs sampler proceeds by alternately sampling from these two normal distributions. In general, we would say that a natural way to start the iterations would be with random draws from a normal approximation to the posterior distribution; such

draws would eliminate the need for iterative simulation in this trivial example. Figure 11.2 illustrates for the case $\rho = 0.8$, data $(y_1, y_2) = (0, 0)$, and four independent sequences started at $(\pm 2.5, \pm 2.5)$.

11.2 Metropolis and Metropolis-Hastings algorithms

The *Metropolis-Hastings algorithm* is a general term for a family of Markov chain simulation methods that are useful for sampling from Bayesian posterior distributions. We have already seen the Gibbs sampler in the previous section; it can be viewed as a special case of Metropolis-Hastings (as described in Section 11.3). Here we present the basic Metropolis algorithm and its generalization to the Metropolis-Hastings algorithm.

The Metropolis algorithm

The Metropolis algorithm is an adaptation of a random walk with an acceptance/rejection rule to converge to the specified target distribution. The algorithm proceeds as follows.

1. Draw a starting point θ^0, for which $p(\theta^0|y) > 0$, from a *starting distribution* $p_0(\theta)$. The starting distribution might, for example, be based on an approximation as described in Section 13.3. Or we may simply choose starting values dispersed around a crude approximate estimate of the sort discussed in Chapter 10.
2. For $t = 1, 2, \ldots$:
   (a) Sample a *proposal* θ^* from a *jumping distribution* (or *proposal distribution*) at time t, $J_t(\theta^*|\theta^{t-1})$. For the Metropolis algorithm (but not the Metropolis-Hastings algorithm, as discussed later in this section), the jumping distribution must be *symmetric*, satisfying the condition $J_t(\theta_a|\theta_b) = J_t(\theta_b|\theta_a)$ for all θ_a, θ_b, and t.
   (b) Calculate the ratio of the densities,
   $$r = \frac{p(\theta^*|y)}{p(\theta^{t-1}|y)}. \tag{11.1}$$
   (c) Set
   $$\theta^t = \begin{cases} \theta^* & \text{with probability } \min(r, 1) \\ \theta^{t-1} & \text{otherwise.} \end{cases}$$

Given the current value θ^{t-1}, the transition distribution $T_t(\theta^t|\theta^{t-1})$ of the Markov chain is thus a mixture of a point mass at $\theta^t = \theta^{t-1}$, and a weighted version of the jumping distribution, $J_t(\theta^t|\theta^{t-1})$, that adjusts for the acceptance rate.

The algorithm requires the ability to calculate the ratio r in (11.1) for all (θ, θ^*), and to draw θ from the jumping distribution $J_t(\theta^*|\theta)$ for all θ and t. In addition, step (c) above requires the generation of a uniform random number.

When $\theta^t = \theta^{t-1}$—that is, if the jump is not accepted—this still counts as an iteration in the algorithm.

Example. Bivariate unit normal density with normal jumping kernel
For simplicity, we illustrate the Metropolis algorithm with the simple example of the bivariate unit normal distribution. The target density is the bivariate unit normal, $p(\theta|y) = \mathrm{N}(\theta|0, I)$, where I is the 2×2 identity matrix. The jumping distribution is also bivariate normal, centered at the current iteration and scaled to 1/5 the size: $J_t(\theta^*|\theta^{t-1}) = \mathrm{N}(\theta^*|\theta^{t-1}, 0.2^2 I)$. At each step, it is easy to calculate the density ratio $r = \mathrm{N}(\theta^*|0, I)/\mathrm{N}(\theta^{t-1}|0, I)$. It is clear from the form of the normal distribution that the jumping rule is symmetric. Figure 11.1 on page 276 displays five simulation runs starting from different points. We have purposely set the scale of this jumping

algorithm to be too small, relative to the target distribution, so that the algorithm will run inefficiently and its random-walk aspect will be obvious in the figure. In Section 12.2 we discuss how to set the jumping scale to optimize the efficiency of the Metropolis algorithm.

Relation to optimization

The acceptance/rejection rule of the Metropolis algorithm can be stated as follows: (a) if the jump increases the posterior density, set $\theta^t = \theta^*$; (b) if the jump decreases the posterior density, set $\theta^t = \theta^*$ with probability equal to the density ratio, r, and set $\theta^t = \theta^{t-1}$ otherwise. The Metropolis algorithm can thus be viewed as a stochastic version of a stepwise mode-finding algorithm, always accepting steps that increase the density but only sometimes accepting downward steps.

Why does the Metropolis algorithm work?

The proof that the sequence of iterations $\theta^1, \theta^2, \ldots$ converges to the target distribution has two steps: first, it is shown that the simulated sequence is a Markov chain with a unique stationary distribution, and second, it is shown that the stationary distribution equals this target distribution. The first step of the proof holds if the Markov chain is irreducible, aperiodic, and not transient. Except for trivial exceptions, the latter two conditions hold for a random walk on any proper distribution, and irreducibility holds as long as the random walk has a positive probability of eventually reaching any state from any other state; that is, the jumping distributions J_t must eventually be able to jump to all states with positive probability.

To see that the target distribution is the stationary distribution of the Markov chain generated by the Metropolis algorithm, consider starting the algorithm at time $t-1$ with a draw θ^{t-1} from the target distribution $p(\theta|y)$. Now consider any two such points θ_a and θ_b, drawn from $p(\theta|y)$ and labeled so that $p(\theta_b|y) \geq p(\theta_a|y)$. The unconditional probability density of a transition from θ_a to θ_b is

$$p(\theta^{t-1}=\theta_a,\, \theta^t=\theta_b) = p(\theta_a|y)J_t(\theta_b|\theta_a),$$

where the acceptance probability is 1 because of our labeling of a and b, and the unconditional probability density of a transition from θ_b to θ_a is, from (11.1),

$$\begin{aligned} p(\theta^t=\theta_a,\, \theta^{t-1}=\theta_b) &= p(\theta_b|y)J_t(\theta_a|\theta_b)\left(\frac{p(\theta_a|y)}{p(\theta_b|y)}\right) \\ &= p(\theta_a|y)J_t(\theta_a|\theta_b), \end{aligned}$$

which is the same as the probability of a transition from θ_a to θ_b, since we have required that $J_t(\cdot|\cdot)$ be symmetric. Since their joint distribution is symmetric, θ^t and θ^{t-1} have the same marginal distributions, and so $p(\theta|y)$ is the stationary distribution of the Markov chain of θ. For more detailed theoretical concerns, see the bibliographic note at the end of this chapter.

The Metropolis-Hastings algorithm

The Metropolis-Hastings algorithm generalizes the basic Metropolis algorithm presented above in two ways. First, the jumping rules J_t need no longer be symmetric; that is, there is no requirement that $J_t(\theta_a|\theta_b) \equiv J_t(\theta_b|\theta_a)$. Second, to correct for the asymmetry in the jumping rule, the ratio r in (11.1) is replaced by a ratio of ratios:

$$r = \frac{p(\theta^*|y)/J_t(\theta^*|\theta^{t-1})}{p(\theta^{t-1}|y)/J_t(\theta^{t-1}|\theta^*)}. \tag{11.2}$$

(The ratio r is always defined, because a jump from θ^{t-1} to θ^* can only occur if both $p(\theta^{t-1}|y)$ and $J_t(\theta^*|\theta^{t-1})$ are nonzero.)

Allowing asymmetric jumping rules can be useful in increasing the speed of the random walk. Convergence to the target distribution is proved in the same way as for the Metropolis algorithm. The proof of convergence to a unique stationary distribution is identical. To prove that the stationary distribution is the target distribution, $p(\theta|y)$, consider any two points θ_a and θ_b with posterior densities labeled so that $p(\theta_b|y)J_t(\theta_a|\theta_b) \geq p(\theta_a|y)J_t(\theta_b|\theta_a)$. If θ^{t-1} follows the target distribution, then it is easy to show that the unconditional probability density of a transition from θ_a to θ_b is the same as the reverse transition.

Relation between the jumping rule and efficiency of simulations

The ideal Metropolis-Hastings jumping rule is simply to sample the proposal, θ^*, from the target distribution; that is, $J(\theta^*|\theta) \equiv p(\theta^*|y)$ for all θ. Then the ratio r in (11.2) is always exactly 1, and the iterates θ^t are a sequence of independent draws from $p(\theta|y)$. In general, however, iterative simulation is applied to problems for which direct sampling is not possible.

A good jumping distribution has the following properties:

- For any θ, it is easy to sample from $J(\theta^*|\theta)$.
- It is easy to compute the ratio r.
- Each jump goes a reasonable distance in the parameter space (otherwise the random walk moves too slowly).
- The jumps are not rejected too frequently (otherwise the random walk wastes too much time standing still).

We return to the topic of constructing efficient simulation algorithms in the next chapter.

11.3 Using Gibbs and Metropolis as building blocks

The Gibbs sampler and the Metropolis algorithm can be used in various combinations to sample from complicated distributions. The Gibbs sampler is the simplest of the Markov chain simulation algorithms, and it is our first choice for conditionally conjugate models, where we can directly sample from each conditional posterior distribution. For example, we could use the Gibbs sampler for the normal-normal hierarchical models in Chapter 5.

The Metropolis algorithm can be used for models that are not conditionally conjugate, for example, the two-parameter logistic regression for the bioassay experiment in Section 3.7. In this example, the Metropolis algorithm could be performed in vector form—jumping in the two-dimensional space of (α, β)—or embedded within a Gibbs sampler structure, by alternately updating α and β using one-dimensional Metropolis jumps. In either case, the Metropolis algorithm will probably have to be tuned to get a good acceptance rate, as discussed in Section 12.2.

If some of the conditional posterior distributions in a model can be sampled from directly and some cannot, then the parameters can be updated one at a time, with the Gibbs sampler used where possible and one-dimensional Metropolis updating used otherwise. More generally, the parameters can be updated in blocks, where each block is altered using the Gibbs sampler or a Metropolis jump of the parameters within the block.

A general problem with conditional sampling algorithms is that they can be slow when parameters are highly correlated in the target distribution (for example, see Figure 11.2 on page 277). This can be fixed in simple problems using reparameterization (see Section 12.1) or more generally using the more advanced algorithms mentioned in Chapter 12.

Interpretation of the Gibbs sampler as a special case of the Metropolis-Hastings algorithm

Gibbs sampling can be viewed as a special case of the Metropolis-Hastings algorithm in the following way. We first define iteration t to consist of a series of d steps, with step j of iteration t corresponding to an update of the subvector θ_j conditional on all the other elements of θ. Then the jumping distribution, $J_{j,t}(\cdot|\cdot)$, at step j of iteration t only jumps along the jth subvector, and does so with the conditional posterior density of θ_j given θ_{-j}^{t-1}:

$$J_{j,t}^{\text{Gibbs}}(\theta^*|\theta^{t-1}) = \begin{cases} p(\theta_j^*|\theta_{-j}^{t-1}, y) & \text{if } \theta_{-j}^* = \theta_{-j}^{t-1} \\ 0 & \text{otherwise.} \end{cases}$$

The only possible jumps are to parameter vectors θ^* that match θ^{t-1} on all components other than the jth. Under this jumping distribution, the ratio (11.2) at the jth step of iteration t is

$$\begin{aligned} r &= \frac{p(\theta^*|y)/J_{j,t}^{\text{Gibbs}}(\theta^*|\theta^{t-1})}{p(\theta^{t-1}|y)/J_{j,t}^{\text{Gibbs}}(\theta^{t-1}|\theta^*)} \\ &= \frac{p(\theta^*|y)/p(\theta_j^*|\theta_{-j}^{t-1}, y)}{p(\theta^{t-1}|y)/p(\theta_j^{t-1}|\theta_{-j}^{t-1}, y)} \\ &= \frac{p(\theta_{-j}^{t-1}|y)}{p(\theta_{-j}^{t-1}|y)} \\ &\equiv 1, \end{aligned}$$

and thus every jump is accepted. The second line above follows from the first because, under this jumping rule, θ^* differs from θ^{t-1} only in the jth component. The third line follows from the second by applying the rules of conditional probability to $\theta = (\theta_j, \theta_{-j})$ and noting that $\theta_{-j}^* = \theta_{-j}^{t-1}$.

Usually, one iteration of the Gibbs sampler is defined as we do, to include all d steps corresponding to the d components of θ, thereby updating all of θ at each iteration. It is possible, however, to define Gibbs sampling without the restriction that each component be updated in each iteration, as long as each component is updated periodically.

Gibbs sampler with approximations

For some problems, sampling from some, or all, of the conditional distributions $p(\theta_j|\theta_{-j}, y)$ is impossible, but one can construct approximations, which we label $g(\theta_j|\theta_{-j})$, from which sampling is possible. The general form of the Metropolis-Hastings algorithm can be used to compensate for the approximation. As in the Gibbs sampler, we choose an order for altering the d elements of θ; the jumping function at the jth Metropolis step at iteration t is then

$$J_{j,t}(\theta^*|\theta^{t-1}) = \begin{cases} g(\theta_j^*|\theta_{-j}^{t-1}) & \text{if } \theta_{-j}^* = \theta_{-j}^{t-1} \\ 0 & \text{otherwise,} \end{cases}$$

and the ratio r in (11.2) must be computed and the acceptance or rejection of θ^* decided.

11.4 Inference and assessing convergence

The basic method of inference from iterative simulation is the same as for Bayesian simulation in general: use the collection of all the simulated draws from $p(\theta|y)$ to summarize the posterior density and to compute quantiles, moments, and other summaries of interest as needed. Posterior predictive simulations of unobserved outcomes $\tilde{y}$ can be obtained by simulation conditional on the drawn values of θ. Inference using the iterative simulation draws requires some care, however, as we discuss in this section.

Difficulties of inference from iterative simulation

Iterative simulation adds two challenges to simulation inference. First, if the iterations have not proceeded long enough, as in Figure 11.1a, the simulations may be grossly unrepresentative of the target distribution. Even when simulations have reached approximate convergence, early iterations still reflect the starting approximation rather than the target distribution; for example, consider the early iterations of Figures 11.1b and 11.2b.

The second problem with iterative simulation draws is their within-sequence correlation; aside from any convergence issues, simulation inference from correlated draws is generally less precise than from the same number of independent draws. Serial correlation in the simulations is not necessarily a problem because, at convergence, the draws are identically distributed as $p(\theta|y)$, and so when performing inferences, we ignore the order of the simulation draws in any case. But such correlation can cause inefficiencies in simulations. Consider Figure 11.1c, which displays 500 successive iterations from each of five simulated sequences of the Metropolis algorithm: the patchy appearance of the scatterplot would not be likely to appear from 2500 independent draws from the normal distribution but is rather a result of the slow movement of the simulation algorithm. In some sense, the 'effective' number of simulation draws here is far fewer than 2500. We calculate effective sample size using formula (11.8) on page 287.

We handle the special problems of iterative simulation in three ways. First, we attempt to design the simulation runs to allow effective monitoring of convergence, in particular by simulating multiple sequences with starting points dispersed throughout parameter space, as in Figure 11.1a. Second, we monitor the convergence of all quantities of interest by comparing variation between and within simulated sequences until 'within' variation roughly equals 'between' variation, as in Figure 11.1b. Only when the distribution of each simulated sequence is close to the distribution of all the sequences mixed together can they all be approximating the target distribution. Third, if the simulation efficiency is unacceptably low (in the sense of requiring too much real time on the computer to obtain approximate convergence of posterior inferences for quantities of interest), the algorithm can be altered, as we discuss in Sections 12.1 and 12.2.

Discarding early iterations of the simulation runs

To diminish the influence of the starting values, we generally discard the first half of each sequence and focus attention on the second half. Our inferences will be based on the assumption that the distributions of the simulated values θ^t, for large enough t, are close to the target distribution, $p(\theta|y)$. We refer to the practice of discarding early iterations in Markov chain simulation as *warm-up*; depending on the context, different warm-up fractions can be appropriate. For example, in the Gibbs sampler displayed in Figure 11.2, it would be necessary to discard only a few initial iterations.[1] We adopt the general practice of discarding the first half as a conservative choice. For example, we might run 200 iterations and discard the first half. If approximate convergence has not yet been reached, we might then run another 200 iterations, now discarding all of the initial 200 iterations.

Dependence of the iterations in each sequence

Another issue that sometimes arises, once approximate convergence has been reached, is whether to *thin* the sequences by keeping every kth simulation draw from each sequence

[1] In the simulation literature (including earlier editions of this book), the warm-up period is called *burn-in*, a term we now avoid because we feel it draws a misleading analogy to industrial processes in which products are stressed in order to reveal defects. We prefer the term 'warm-up' to describe the early phase of the simulations in which the sequences get closer to the mass of the distribution.

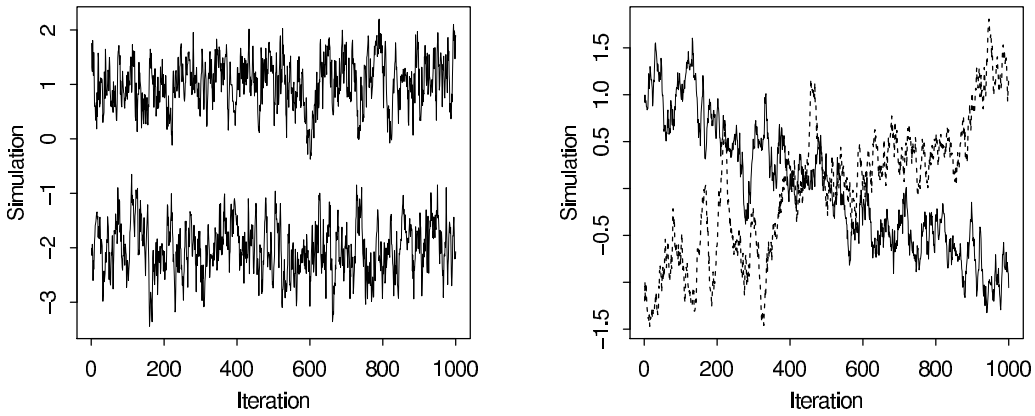

Figure 11.3 *Examples of two challenges in assessing convergence of iterative simulations. (a) In the left plot, either sequence alone looks stable, but the juxtaposition makes it clear that they have not converged to a common distribution. (b) In the right plot, the twosequences happen to cover a common distribution but neither sequence appears stationary. These graphs demonstrate the need to use between-sequence and also within-sequence information when assessing convergence.*

and discarding the rest. In our applications, we have found it useful to skip iterations in problems with large numbers of parameters where computer storage is a problem, perhaps setting k so that the total number of iterations saved is no more than 1000.

Whether or not the sequences are thinned, if the sequences have reached approximate convergence, they can be directly used for inferences about the parameters θ and any other quantities of interest.

Multiple sequences with overdispersed starting points

Our recommended approach to assessing convergence of iterative simulation is based on comparing different simulated sequences, as illustrated in Figure 11.1 on page 276, which shows five parallel simulations before and after approximate convergence. In Figure 11.1a, the multiple sequences clearly have not converged; the variance within each sequence is much less than the variance between sequences. Later, in Figure 11.1b, the sequences have mixed, and the two variance components are essentially equal.

To see such disparities, we clearly need more than one independent sequence. Thus our plan is to simulate independently at least two sequences, with starting points drawn from an overdispersed distribution (either from a crude estimate such as discussed in Section 10.2 or a more elaborate approximation as discussed in the next chapter).

Monitoring scalar estimands

We monitor each scalar estimand or other scalar quantities of interest separately. Estimands include all the parameters of interest in the model and any other quantities of interest (for example, the ratio of two parameters or the value of a predicted future observation). It is often useful also to monitor the value of the logarithm of the posterior density, which has probably already been computed if we are using a version of the Metropolis algorithm.

Challenges of monitoring convergence: mixing and stationarity

Figure 11.3 illustrates two of the challenges of monitoring convergence of iterative simulations. The first graph shows two sequences, each of which looks fine on its own (and,

indeed, when looked at separately would satisfy any reasonable convergence criterion), but when looked at together reveal a clear lack of convergence. Figure 11.3a illustrates that, to achieve convergence, the sequences must together have *mixed*.

The second graph in Figure 11.3 shows two chains that have mixed, in the sense that they have traced out a common distribution, but they do not appear to have converged. Figure 11.3b illustrates that, to achieve convergence, each individual sequence must reach *stationarity*.

Splitting each saved sequence into two parts

We diagnose convergence (as noted above, separately for each scalar quantity of interest) by checking mixing and stationarity. There are various ways to do this; we apply a fairly simple approach in which we split each chain in half and check that all the resulting half-sequences have mixed. This simultaneously tests mixing (if all the chains have mixed well, the separate parts of the different chains should also mix) and stationarity (at stationarity, the first and second half of each sequence should be traversing the same distribution).

We start with some number of simulated sequences in which the warm-up period (which by default we set to the first half of the simulations) has already been discarded. We then take each of these chains and split into the first and second half (this is all *after* discarding the warm-up iterations). Let m be the number of chains (after splitting) and n be the length of each chain. We always simulate at least two sequences so that we can observe mixing; see Figure 11.3a; thus m is always at least 4.

For example, suppose we simulate 5 chains, each of length 1000, and then discard the first half of each as warm-up. We are then left with 5 chains, each of length 500, and we split each into two parts: iterations 1–250 (originally iterations 501–750) and iterations 251–500 (originally iterations 751–1000). We now have $m = 10$ chains, each of length $n = 250$.

Assessing mixing using between- and within-sequence variances

For each scalar estimand ψ, we label the simulations as ψ_{ij} $(i=1,\ldots,n; j=1,\ldots,m)$, and we compute B and W, the between- and within-sequence variances:

$$B = \frac{n}{m-1}\sum_{j=1}^{m}(\overline{\psi}_{.j} - \overline{\psi}_{..})^2, \quad \text{where} \quad \overline{\psi}_{.j} = \frac{1}{n}\sum_{i=1}^{n}\psi_{ij}, \quad \overline{\psi}_{..} = \frac{1}{m}\sum_{j=1}^{m}\overline{\psi}_{.j}$$

$$W = \frac{1}{m}\sum_{j=1}^{m}s_j^2, \quad \text{where} \quad s_j^2 = \frac{1}{n-1}\sum_{i=1}^{n}(\psi_{ij} - \overline{\psi}_{.j})^2.$$

The between-sequence variance, B, contains a factor of n because it is based on the variance of the within-sequence means, $\overline{\psi}_{.j}$, each of which is an average of n values ψ_{ij}.

We can estimate $\text{var}(\psi|y)$, the marginal posterior variance of the estimand, by a weighted average of W and B, namely

$$\widehat{\text{var}}^{+}(\psi|y) = \frac{n-1}{n}W + \frac{1}{n}B. \tag{11.3}$$

This quantity *overestimates* the marginal posterior variance assuming the starting distribution is appropriately overdispersed, but is *unbiased* under stationarity (that is, if the starting distribution equals the target distribution), or in the limit $n \to \infty$ (see Exercise 11.5). This is analogous to the classical variance estimate with cluster sampling.

Meanwhile, for any finite n, the 'within' variance W should be an *underestimate* of $\text{var}(\psi|y)$ because the individual sequences have not had time to range over all of the target

Number of iterations	95% intervals and $\widehat{R}$ for ... θ_1	θ_2	$\log p(\theta_1, \theta_2 \mid y)$
50	$[-2.14, 3.74]$, 12.3	$[-1.83, 2.70]$, 6.1	$[-8.71, -0.17]$, 6.1
500	$[-3.17, 1.74]$, 1.3	$[-2.17, 2.09]$, 1.7	$[-5.23, -0.07]$, 1.3
2000	$[-1.83, 2.24]$, 1.2	$[-1.74, 2.09]$, 1.03	$[-4.07, -0.03]$, 1.10
5000	$[-2.09, 1.98]$, 1.02	$[-1.90, 1.95]$, 1.03	$[-3.70, -0.03]$, 1.00
∞	$[-1.96, 1.96]$, 1	$[-1.96, 1.96]$, 1	$[-3.69, -0.03]$, 1

Table 11.1 *95% central intervals and estimated potential scale reduction factors for three scalar summaries of the bivariate normal distribution simulated using a Metropolis algorithm. (For demonstration purposes, the jumping scale of the Metropolis algorithm was purposely set to be inefficient; see Figure 11.1.) Displayed are inferences from the second halves of five parallel sequences, stopping after 50, 500, 2000, and 5000 iterations. The intervals for ∞ are taken from the known normal and $\chi^2_2/2$ marginal distributions for these summaries in the target distribution.*

distribution and, as a result, will have less variability; in the limit as $n \to \infty$, the expectation of W approaches $\text{var}(\psi|y)$.

We monitor convergence of the iterative simulation by estimating the factor by which the scale of the current distribution for ψ might be reduced if the simulations were continued in the limit $n \to \infty$. This potential scale reduction is estimated by[2]

$$\widehat{R} = \sqrt{\frac{\widehat{\text{var}}^+(\psi|y)}{W}}, \tag{11.4}$$

which declines to 1 as $n \to \infty$. If the potential scale reduction is high, then we have reason to believe that proceeding with further simulations may improve our inference about the target distribution of the associated scalar estimand.

Example. Bivariate unit normal density with bivariate normal jumping kernel (continued)

We illustrate the multiple sequence method using the Metropolis simulations of the bivariate normal distribution illustrated in Figure 11.1. Table 11.1 displays posterior inference for the two parameters of the distribution as well as the log posterior density (relative to the density at the mode). After 50 iterations, the variance between the five sequences is much greater than the variance within, for all three univariate summaries considered. However, the five simulated sequences have converged adequately after 2000 or certainly 5000 iterations for the quantities of interest. The comparison with the true target distribution shows how some variability remains in the posterior inferences even after the Markov chains have converged. (This must be so, considering that even if the simulation draws were independent, so that the Markov chains would converge in a single iteration, it would still require hundreds or thousands of draws to obtain precise estimates of extreme posterior quantiles.)

The method of monitoring convergence presented here has the key advantage of not requiring the user to examine time series graphs of simulated sequences. Inspection of such plots is a notoriously unreliable method of assessing convergence and in addition is unwieldy when monitoring a large number of quantities of interest, such as can arise in complicated hierarchical models. Because it is based on means and variances, the simple

[2]In the first edition of this book, $\widehat{R}$ was defined as $\widehat{\text{var}}^+(\psi|y)/W$. We have switched to the square-root definition for notational convenience. We have also made one major change since the second edition of the book. Our current $\widehat{R}$ has the same formula as before, but we now compute it on the split chains, whereas previously we applied it to the entire chains unsplit. The unsplit $\widehat{R}$ from the earlier editions of this book would not correctly diagnose the poor convergence in Figure 11.3b.

method presented here is most effective for quantities whose marginal posterior distributions are approximately normal. When performing inference for extreme quantiles, or for parameters with multimodal marginal posterior distributions, one should monitor also extreme quantiles of the 'between' and 'within' sequences.

11.5 Effective number of simulation draws

Once the simulated sequences have mixed, we can compute an approximate 'effective number of independent simulation draws' for any estimand of interest ψ. We start with the observation that if the n simulation draws within each sequence were truly independent, then the between-sequence variance B would be an unbiased estimate of the posterior variance, $\text{var}(\psi|y)$, and we would have a total of mn independent simulations from the m sequences. In general, however, the simulations of ψ within each sequence will be autocorrelated, and B will be larger than $\text{var}(\psi|y)$, in expectation.

One way to define effective sample size for correlated simulation draws is to consider the statistical efficiency of the average of the simulations $\bar{\psi}_{\cdot\cdot}$, as an estimate of the posterior mean, $\text{E}(\psi|y)$. This can be a reasonable baseline even though is not the only possible summary and might be inappropriate, for example, if there is particular interest in accurate representation of low-probability events in the tails of the distribution.

Continuing with this definition, it is usual to compute effective sample size using the following asymptotic formula for the variance of the average of a correlated sequence:

$$\lim_{n\to\infty} mn\,\text{var}(\bar{\psi}_{\cdot\cdot}) = \left(1 + 2\sum_{t=1}^{\infty}\rho_t\right)\text{var}(\psi|y), \tag{11.5}$$

where ρ_t is the autocorrelation of the sequence ψ at lag t. If the n simulation draws from each of the m chains were independent, then $\text{var}(\bar{\psi}_{\cdot\cdot})$ would simply be $\frac{1}{mn}\text{var}(\psi|y)$ and the sample size would be mn. In the presence of correlation we then define the *effective sample size* as

$$n_{\text{eff}} = \frac{mn}{1 + 2\sum_{t=1}^{\infty}\rho_t}. \tag{11.6}$$

The asymptotic nature of (11.5)–(11.6) might seem disturbing given that in reality we will only have a finite simulation, but this should not be a problem given that we already want to run the simulations long enough for approximate convergence to the (asymptotic) target distribution.

To compute the effective sample size we need an estimate of the sum of the correlations ρ, for which we use information within and between sequences. We start by computing the total variance using the estimate $\widehat{\text{var}}^+$ from (11.3); we then estimate the correlations by first computing the *variogram* V_t at each lag t:

$$V_t = \frac{1}{m(n-t)}\sum_{j=1}^{m}\sum_{i=t+1}^{n}(\psi_{i,j} - \psi_{i-t,j})^2.$$

We then estimate the correlations by inverting the formula, $\text{E}(\psi_i - \psi_{i-t})^2 = 2(1-\rho_t)\text{var}(\psi)$:

$$\widehat{\rho}_t = 1 - \frac{V_t}{2\widehat{\text{var}}^+}. \tag{11.7}$$

Unfortunately we cannot simply sum all of these to estimate n_{eff} in (11.6); the difficulty is that for large values of t the sample correlation is too noisy. Instead we compute a partial sum, starting from lag 0 and continuing until the sum of autocorrelation estimates for two

successive lags $\hat{\rho}_{2t'} + \hat{\rho}_{2t'+1}$ is negative. We use this positive partial sum as our estimate of $\sum_{t=1}^{\infty} \rho_t$ in (11.6). Putting this all together yields the estimate,

$$\hat{n}_{\text{eff}} = \frac{mn}{1 + 2\sum_{t=1}^{T} \widehat{\rho}_t}, \tag{11.8}$$

where the estimated autocorrelations $\widehat{\rho}_t$ are computed from formula (11.7) and T is the first odd positive integer for which $\widehat{\rho}_{T+1} + \widehat{\rho}_{T+2}$ is negative.

All these calculations should be performed using only the saved iterations, after discarding the warm-up period. For example, suppose we simulate 4 chains, each of length 1000, and then discard the first half of each as warm-up. Then $m = 8$, $n = 250$, and we compute variograms and correlations only for the saved iterations (thus, up to a maximum lag t of 249, although in practice the stopping point T in (11.8) will be much lower).

Bounded or long-tailed distributions

The above convergence diagnostics are based on means and variances, and they will not work so well for parameters or scalar summaries for which the posterior distribution, $p(\phi|y)$, is far from Gaussian. (As discussed in Chapter 4, asymptotically the posterior distribution should typically be normally distributed as the data sample size approaches infinity, but (a) we are never actually at the asymptotic limit (in fact we are often interested in learning from small samples), and (b) it is common to have only a small amount of data on individual parameters that are part of a hierarchical model.)

For summaries ϕ whose distributions are constrained or otherwise far from normal, we can preprocess simulations using transformations before computing the potential scale reduction factor $\widehat{R}$ and the effective sample size $\hat{n}_{\text{eff}}$. We can take the logarithm of all-positive quantities, the logit of quantities that are constrained to fall in $(0, 1)$, and use the rank transformation for long-tailed distributions. Transforming the simulations to have well-behaved distributions should allow mean and variance-based convergence diagnostics to work better.

Stopping the simulations

We monitor convergence for the entire multivariate distribution, $p(\theta|y)$, by computing the potential scale reduction factor (11.4) and the effective sample size (11.8) for each scalar summary of interest. (Recall that we are using θ to denote the vector of unknowns in the posterior distribution, and ϕ to represent scalar summaries, considered one at a time.)

We recommend computing the potential scale reduction for all scalar estimands of interest; if $\widehat{R}$ is not near 1 for all of them, continue the simulation runs (perhaps altering the simulation algorithm itself to make the simulations more efficient, as described in the next section). The condition of $\widehat{R}$ being 'near' 1 depends on the problem at hand, but we generally have been satisfied with setting 1.1 as a threshold.

We can use effective sample size $\hat{n}_{\text{eff}}$ to give us a sense of the precision obtained from our simulations. As we have discussed in Section 10.5, for many purposes it should suffice to have 100 or even 10 independent simulation draws. (If $n_{\text{eff}} = 10$, the simulation standard error is increased by $\sqrt{1 + 1/10} = 1.05$). As a default rule, we suggest running the simulation until $\hat{n}_{\text{eff}}$ is at least $5m$, that is, until there are the equivalent of at least 10 independent draws per sequence (recall that m is twice the number of sequences, as we have split each sequence into two parts so that $\widehat{R}$ can assess stationarity as well as mixing). Having an effective sample size of 10 per sequence should typically correspond to stability of all the simulated sequences. For some purposes, more precision will be desired, and then a higher effective sample size threshold can be used.

Diet	Measurements
A	62, 60, 63, 59
B	63, 67, 71, 64, 65, 66
C	68, 66, 71, 67, 68, 68
D	56, 62, 60, 61, 63, 64, 63, 59

Table 11.2 *Coagulation time in seconds for blood drawn from 24 animals randomly allocated to four different diets. Different treatments have different numbers of observations because the randomization was unrestricted. From Box, Hunter, and Hunter (1978), who adjusted the data so that the averages are integers, a complication we ignore in our analysis.*

Once $\widehat{R}$ is near 1 and $\hat{n}_{\text{eff}}$ is more than 10 per chain for all scalar estimands of interest, just collect the mn simulations (with warm-up iterations already excluded, as noted before) and treat them as a sample from the target distribution.

Even if an iterative simulation appears to converge and has passed all tests of convergence, it still may actually be far from convergence if important areas of the target distribution were not captured by the starting distribution and are not easily reachable by the simulation algorithm. When we declare approximate convergence, we are actually concluding that each individual sequence appears stationary and that the observed sequences have mixed well with each other. These checks are not hypothesis tests. There is no p-value and no statistical significance. We assess discrepancy from convergence via practical significance (or some conventional version thereof, such as $\widehat{R} > 1.1$).

11.6 Example: hierarchical normal model

We illustrate the simulation algorithms with a hierarchical normal model, extending the problem discussed in Section 5.4 by allowing an unknown data variance, σ^2. The example is continued in Section 13.6 to illustrate mode-based computation. We demonstrate with the normal model because it is simple enough that the key computational ideas do not get lost in the details.

Data from a small experiment

We demonstrate the computations on a small experimental dataset, displayed in Table 11.2, that has been used previously as an example in the statistical literature. Our purpose here is solely to illustrate computational methods, not to perform a full Bayesian data analysis (which includes model construction and model checking), and so we do not discuss the applied context.

The model

Under the hierarchical normal model (restated here, for convenience), data y_{ij}, $i = 1, \ldots, n_j$, $j = 1, \ldots, J$, are independently normally distributed within each of J groups, with means θ_j and common variance σ^2. The total number of observations is $n = \sum_{j=1}^{J} n_j$. The group means are assumed to follow a normal distribution with unknown mean μ and variance τ^2, and a uniform prior distribution is assumed for $(\mu, \log\sigma, \tau)$, with $\sigma > 0$ and $\tau > 0$; equivalently, $p(\mu, \log\sigma, \log\tau) \propto \tau$. If we were to assign a uniform prior distribution to $\log\tau$, the posterior distribution would be improper, as discussed in Chapter 5.

The joint posterior density of all the parameters is

$$p(\theta, \mu, \log\sigma, \log\tau|y) \propto \tau \prod_{j=1}^{J} \text{N}(\theta_j|\mu, \tau^2) \prod_{j=1}^{J} \prod_{i=1}^{n_j} \text{N}(y_{ij}|\theta_j, \sigma^2).$$

Starting points

In this example, we can choose overdispersed starting points for each parameter θ_j by simply taking random points from the data y_{ij} from group j. We obtain 10 starting points for the simulations by drawing θ_j independently in this way for each group. We also need starting points for μ, which can be taken as the average of the starting θ_j values. No starting values are needed for τ or σ as they can be drawn as the first steps in the Gibbs sampler.

Section 13.6 presents a more elaborate procedure for constructing a starting distribution for the iterative simulations using the posterior mode and a normal approximation.

Gibbs sampler

The conditional distributions for this model all have simple conjugate forms:

1. *Conditional posterior distribution of each θ_j.* The factors in the joint posterior density that involve θ_j are the $\mathrm{N}(\mu, \tau^2)$ prior distribution and the normal likelihood from the data in the jth group, $y_{ij}, i = 1, \ldots, n_j$. The conditional posterior distribution of each θ_j given the other parameters in the model is

$$\theta_j | \mu, \sigma, \tau, y \sim \mathrm{N}(\hat{\theta}_j, V_{\theta_j}), \tag{11.9}$$

   where the parameters of the conditional posterior distribution depend on μ, σ, and τ as well as y:

$$\hat{\theta}_j = \frac{\frac{1}{\tau^2}\mu + \frac{n_j}{\sigma^2}\overline{y}_{.j}}{\frac{1}{\tau^2} + \frac{n_j}{\sigma^2}} \tag{11.10}$$

$$V_{\theta_j} = \frac{1}{\frac{1}{\tau^2} + \frac{n_j}{\sigma^2}}. \tag{11.11}$$

   These conditional distributions are independent; thus drawing the θ_j's one at a time is equivalent to drawing the vector θ all at once from its conditional posterior distribution.

2. *Conditional posterior distribution of μ.* Conditional on y and the other parameters in the model, μ has a normal distribution determined by the θ_j's:

$$\mu | \theta, \sigma, \tau, y \sim \mathrm{N}(\hat{\mu}, \tau^2/J), \tag{11.12}$$

   where

$$\hat{\mu} = \frac{1}{J}\sum_{j=1}^{J}\theta_j. \tag{11.13}$$

3. *Conditional posterior distribution of σ^2.* The conditional posterior density for σ^2 has the form corresponding to a normal variance with known mean; there are n observations y_{ij} with means θ_j. The conditional posterior distribution is

$$\sigma^2 | \theta, \mu, \tau, y \sim \text{Inv-}\chi^2(n, \hat{\sigma}^2), \tag{11.14}$$

   where

$$\hat{\sigma}^2 = \frac{1}{n}\sum_{j=1}^{J}\sum_{i=1}^{n_j}(y_{ij} - \theta_j)^2. \tag{11.15}$$

4. *Conditional posterior distribution of τ^2.* Conditional on the data and the other parameters in the model, τ^2 has a scaled inverse-χ^2 distribution, with parameters depending only on μ and θ (as can be seen by examining the joint posterior density):

$$\tau^2 | \theta, \mu, \sigma, y \sim \text{Inv-}\chi^2(J - 1, \hat{\tau}^2), \tag{11.16}$$

Estimand	Posterior quantiles					$\widehat{R}$
	2.5%	25%	median	75%	97.5%	
θ_1	58.9	60.6	61.3	62.1	63.5	1.01
θ_2	63.9	65.3	65.9	66.6	67.7	1.01
θ_3	66.0	67.1	67.8	68.5	69.5	1.01
θ_4	59.5	60.6	61.1	61.7	62.8	1.01
μ	56.9	62.2	63.9	65.5	73.4	1.04
σ	1.8	2.2	2.4	2.6	3.3	1.00
τ	2.1	3.6	4.9	7.6	26.6	1.05
$\log p(\mu, \log\sigma, \log\tau\|y)$	−67.6	−64.3	−63.4	−62.6	−62.0	1.02
$\log p(\theta, \mu, \log\sigma, \log\tau\|y)$	−70.6	−66.5	−65.1	−64.0	−62.4	1.01

Table 11.3 *Summary of inference for the coagulation example. Posterior quantiles and estimated potential scale reductions are computed from the second halves of ten Gibbs sampler sequences, each of length 100. Potential scale reductions for σ and τ are computed on the log scale. The hierarchical standard deviation, τ, is estimated less precisely than the unit-level standard deviation, σ, as is typical in hierarchical modeling with a small number of batches.*

with

$$\hat{\tau}^2 = \frac{1}{J-1}\sum_{j=1}^{J}(\theta_j - \mu)^2. \tag{11.17}$$

The expressions for τ^2 have $(J-1)$ degrees of freedom instead of J because $p(\tau) \propto 1$ rather than τ^{-1}.

Numerical results with the coagulation data

We illustrate the Gibbs sampler with the coagulation data of Table 11.2. Inference from ten parallel Gibbs sampler sequences appears in Table 11.3; 100 iterations were sufficient for approximate convergence.

The Metropolis algorithm

We also describe how the Metropolis algorithm can be used for this problem. It would be possible to apply the algorithm to the entire joint distribution, $p(\theta, \mu, \sigma, \tau|y)$, but we can work more efficiently in a lower-dimensional space by taking advantage of the conjugacy of the problem that allows us to compute the function $p(\mu, \log\sigma, \log\tau|y)$, as we discuss in Section 13.6. We use the Metropolis algorithm to jump through the marginal posterior distribution of $(\mu, \log\sigma, \log\tau)$ and then draw simulations of the vector θ from its normal conditional posterior distribution (11.9). Following a principle of efficient Metropolis jumping that we shall discuss in Section 12.2, we jump through the space of $(\mu, \log\sigma, \log\tau)$ using a multivariate normal jumping kernel centered at the current value of the parameters and variance matrix equal to that of a normal approximation (see Section 13.6), multiplied by $2.4^2/d$, where d is the dimension of the Metropolis jumping distribution. In this case, $d = 3$.

Metropolis results with the coagulation data

We ran ten parallel sequences of Metropolis algorithm simulations. In this case 500 iterations were sufficient for approximate convergence ($\widehat{R} < 1.1$ for all parameters); at that point we obtained similar results to those obtained using Gibbs sampling. The acceptance rate for the Metropolis simulations was 0.35, which is close to the expected result for the normal distribution with $d = 3$ using a jumping distribution scaled by $2.4/\sqrt{d}$ (see Section 12.1).

11.7 Bibliographic note

Gilks, Richardson, and Spiegelhalter (1996) is a book full of examples and applications of Markov chain simulation methods. Further references on Bayesian computation appear in the books by Tanner (1993), Chen, Shao, and Ibrahim (2000), and Robert, and Casella (2004). Many other applications of Markov chain simulation appear in the recent applied statistical literature.

Metropolis and Ulam (1949) and Metropolis et al. (1953) apparently were the first to describe Markov chain simulation of probability distributions (that is, the 'Metropolis algorithm'). Their algorithm was generalized by Hastings (1970); see Chib and Greenberg (1995) for an elementary introduction and Tierney (1998) for a theoretical perspective. The conditions for Markov chain convergence appear in probability texts such as Feller (1968), and more recent work such as Rosenthal (1995) has evaluated the rates of convergence of Markov chain algorithms for statistical models. The Gibbs sampler was first so named by Geman and Geman (1984) in a discussion of applications to image processing. Tanner and Wong (1987) introduced the idea of iterative simulation to many statisticians, using the special case of 'data augmentation' to emphasize the analogy to the EM algorithm (see Section 13.4). Gelfand and Smith (1990) showed how the Gibbs sampler could be used for Bayesian inference for a variety of important statistical models. The Metropolis-approximate Gibbs algorithm introduced at the end of Section 11.3 appears in Gelman (1992b) and is used by Gilks, Best, and Tan (1995).

Gelfand et al. (1990) applied Gibbs sampling to a variety of statistical problems, and many other applications of Gibbs sampler algorithms have appeared since; for example, Clayton (1991) and Carlin and Polson (1991). Besag and Green (1993), Gilks et al. (1993), and Smith and Roberts (1993) discuss Markov simulation algorithms for Bayesian computation. `Bugs` (Spiegelhalter et al., 1994, 2003) is a general-purpose computer program for Bayesian inference using the Gibbs sampler; see Appendix C for details.

Inference and monitoring convergence from iterative simulation are reviewed by Gelman and Rubin (1992b) and Brooks and Gelman (1998), who provide a theoretical justification of the method presented in Section 11.4 and discuss more elaborate versions of the method; see also Brooks and Giudici (2000) and Gelman and Shirley (2011). Other views on assessing convergence appear in the ensuing discussion of Gelman and Rubin (1992b) and Geyer (1992) and in Cowles and Carlin (1996) and Brooks and Roberts (1998). Gelman and Rubin (1992a,b) and Glickman (1993) present examples of iterative simulation in which lack of convergence is impossible to detect from single sequences but is obvious from multiple sequences. The rule for summing autocorrelations and stopping after the sum of two is negative comes from Geyer (1992).

Venna, Kaski, and Peltonen (2003) and Peltonen, Venna, and Kaski (2009) discuss graphical diagnostics for convergence of iterative simulations.

11.8 Exercises

1. Metropolis-Hastings algorithm: Show that the stationary distribution for the Metropolis-Hastings algorithm is, in fact, the target distribution, $p(\theta|y)$.
2. Metropolis algorithm: Replicate the computations for the bioassay example of Section 3.7 using the Metropolis algorithm. Be sure to define your starting points and your jumping rule. Compute with log-densities (see page 261). Run the simulations long enough for approximate convergence.
3. Gibbs sampling: Table 11.4 contains quality control measurements from 6 machines in a factory. Quality control measurements are expensive and time-consuming, so only 5 measurements were done for each machine. In addition to the existing machines, we are interested in the quality of another machine (the seventh machine). Implement a

Machine	Measurements
1	83, 92 ,92, 46, 67
2	117, 109, 114, 104, 87
3	101, 93, 92, 86, 67
4	105, 119, 116, 102, 116
5	79, 97, 103, 79, 92
6	57, 92, 104, 77, 100

Table 11.4: *Quality control measurements from 6 machines in a factory.*

separate, a pooled and hierarchical Gaussian model with common variance described in Section 11.6. Run the simulations long enough for approximate convergence. Using each of three models—separate, pooled, and hierarchical—report: (i) the posterior distribution of the mean of the quality measurements of the sixth machine, (ii) the predictive distribution for another quality measurement of the sixth machine, and (iii) the posterior distribution of the mean of the quality measurements of the seventh machine.

4. Gibbs sampling: Extend the model in Exercise 11.3 by adding a hierarchical model for the variances of the machine quality measurements. Use an Inv-χ^2 prior distribution for variances with unknown scale σ_0^2 and fixed degrees of freedom. (The data do not contain enough information for determining the degrees of freedom, so inference for that hyperparameter would depend very strongly on its prior distribution in any case). The conditional distribution of σ_0^2 is not of simple form, but you can sample from its distribution, for example, using grid sampling.
5. Monitoring convergence:
   (a) Prove that $\widehat{\text{var}}^+(\psi|y)$ as defined in (11.3) is an unbiased estimate of the marginal posterior variance of ϕ, if the starting distribution for the Markov chain simulation algorithm is the same as the target distribution, and if the m parallel sequences are computed independently. (Hint: show that $\widehat{\text{var}}^+(\psi|y)$ can be expressed as the average of the halved squared differences between simulations ϕ from different sequences, and that each of these has expectation equal to the posterior variance.)
   (b) Determine the conditions under which $\widehat{\text{var}}^+(\psi|y)$ approaches the marginal posterior variance of ϕ in the limit as the lengths n of the simulated chains approach ∞.
6. Effective sample size:
   (a) Derive the asymptotic formula (11.5) for the variance of the average of correlated simulations.
   (b) Implement a Markov chain simulation for some example and plot $\widehat{n}_{\text{eff}}$ from (11.8) over time. Is $\widehat{n}_{\text{eff}}$ stable? Does it gradually increase as a function of number of iterations, as one would hope?
7. Analysis of survey data: Section 8.3 presents an analysis of a stratified sample survey using a hierarchical model on the stratum probabilities.
   (a) Perform the computations for the simple nonhierarchical model described in the example.
   (b) Using the Metropolis algorithm, perform the computations for the hierarchical model, using the results from part (a) as a starting distribution. Check by comparing your simulations to the results in Figure 8.1b.

Chapter 12

Computationally efficient Markov chain simulation

The basic Gibbs sampler and Metropolis algorithm can be seen as building blocks for more advanced Markov chain simulation algorithms that can work well for a wide range of problems. In Sections 12.1 and 12.2, we discuss reparameterizations and settings of tuning parameters to make Gibbs and Metropolis more efficient. Section 12.4 presents Hamiltonian Monte Carlo, a generalization of the Metropolis algorithm that includes 'momentum' variables so that each iteration can move farther in parameter space, thus allowing faster mixing, especially in high dimensions. We follow up in Sections 12.5 and 12.6 with an application to a hierarchical model and a discussion of our program Stan, which implements HMC for general models.

12.1 Efficient Gibbs samplers

Transformations and reparameterization

The Gibbs sampler is most efficient when parameterized in terms of independent components; Figure 11.2 shows an example with highly dependent components that create slow convergence. The simplest way to reparameterize is by a linear transformation of the parameters, but posterior distributions that are not approximately normal may require special methods.

The same arguments apply to Metropolis jumps. In a normal or approximately normal setting, the jumping kernel should ideally have the same covariance structure as the target distribution, which can be approximately estimated based on the normal approximation at the mode (as we discussed in Chapter 13). Markov chain simulation of a distribution with multiple modes can be greatly improved by allowing jumps between modes.

Auxiliary variables

Gibbs sampler computations can often be simplified or convergence accelerated by adding auxiliary variables, for example indicators for mixture distributions, as described in Chapter 22. The idea of adding variables is also called *data augmentation* and is often a useful conceptual and computational tool, both for the Gibbs sampler and for the EM algorithm (see Section 13.4).

Example. Modeling the t distribution as a mixture of normals
A simple but important example of auxiliary variables arises with the t distribution, which can be expressed as a mixture of normal distributions, as noted in Chapter 3 and discussed in more detail in Chapter 17. We illustrate with the example of inference for the parameters μ, σ^2 given n independent data points from the $t_\nu(\mu, \sigma^2)$ distribution, where for simplicity we assume ν is known. We also assume a uniform

prior distribution on $\mu, \log\sigma$. The t likelihood for each data point is equivalent to the model,

$$\begin{aligned} y_i &\sim \text{N}(\mu, V_i) \\ V_i &\sim \text{Inv-}\chi^2(\nu, \sigma^2), \end{aligned} \tag{12.1}$$

where the V_i's are auxiliary variables that cannot be directly observed. If we perform inference using the joint posterior distribution, $p(\mu, \sigma^2, V|y)$, and then just consider the simulations for μ, σ, these will represent the posterior distribution under the original t model.

There is no direct way to sample the parameters μ, σ^2 in the t model, but it is straightforward to perform the Gibbs sampler on V, μ, σ^2 in the augmented model:

1. *Conditional posterior distribution of each V_i.* Conditional on the data y and the other parameters of the model, each V_i is a normal variance parameter with a scaled inverse-χ^2 prior distribution, and so its posterior distribution is also inverse-χ^2 (see Section 2.6):

$$V_i|\mu, \sigma^2, \nu, y \sim \text{Inv-}\chi^2\left(\nu + 1, \frac{\nu\sigma^2 + (y_i - \mu)^2}{\nu + 1}\right).$$

   The n parameters V_i are independent in their conditional posterior distribution, and we can directly apply the Gibbs sampler by sampling from their scaled inverse-χ^2 distributions.

2. *Conditional posterior distribution of μ.* Conditional on the data y and the other parameters of the model, information about μ is supplied by the n data points y_i, each with its own variance. Combining with the uniform prior distribution on μ yields,

$$\mu|\sigma^2, V, \nu, y \sim \text{N}\left(\frac{\sum_{i=1}^n \frac{1}{V_i} y_i}{\sum_{i=1}^n \frac{1}{V_i}}, \frac{1}{\sum_{i=1}^n \frac{1}{V_i}}\right).$$

3. *Conditional posterior distribution of σ^2.* Conditional on the data y and the other parameters of the model, all the information about σ comes from the variances V_i. The conditional posterior distribution is,

$$\begin{aligned} p(\sigma^2|\mu, V, \nu, y) &\propto \sigma^{-2} \prod_{i=1}^n \sigma^\nu e^{-\nu\sigma^2/(2V_i)} \\ &= (\sigma^2)^{n\nu/2-1} \exp\left(-\frac{\nu}{2}\sum_{i=1}^n \frac{1}{V_i}\sigma^2\right) \\ &\propto \text{Gamma}\left(\sigma^2 \left| \frac{n\nu}{2}, \frac{\nu}{2}\sum_{i=1}^n \frac{1}{V_i}\right.\right), \end{aligned}$$

   from which we can sample directly.

Parameter expansion

For some problems, the Gibbs sampler can be slow to converge because of posterior dependence among parameters that cannot simply be resolved with a linear transformation. Paradoxically, adding an additional parameter—thus performing the random walk in a larger space—can improve the convergence of the Markov chain simulation. We illustrate with the t example above.

Example. Fitting the t model (continued)
In the latent-parameter form (12.1) of the t model, convergence will be slow if a simulation draw of σ is close to zero, because the conditional distributions will then cause the V_i's to be sampled with values near zero, and then the conditional distribution of σ will be near zero, and so on. Eventually the simulations will get unstuck but it can be slow for some problems. We can fix things by adding a new parameter whose only role is to allow the Gibbs sampler to move in more directions and thus avoid getting stuck. The expanded model is,

$$\begin{aligned} y_i &\sim \text{N}(\mu, \alpha^2 U_i) \\ U_i &\sim \text{Inv-}\chi^2(\nu, \tau^2), \end{aligned}$$

where $\alpha > 0$ can be viewed as an additional scale parameter. In this new model, $\alpha^2 U_i$ plays the role of V_i in (12.1) and $\alpha\tau$ plays the role of σ. The parameter α has no meaning on its own and we can assign it a noninformative uniform prior distribution on the logarithmic scale.
The Gibbs sampler on this expanded model now has four steps:

1. For each i, U_i is updated much as V_i was before:

$$U_i|\alpha, \mu, \tau^2, \nu, y \sim \text{Inv-}\chi^2\left(\nu + 1, \frac{\nu\tau^2 + ((y_i - \mu)/\alpha)^2}{\nu + 1}\right).$$

2. The mean, μ, is updated as before:

$$\mu|\alpha, \tau^2, U, \nu, y \sim \text{N}\left(\frac{\sum_{i=1}^n \frac{1}{\alpha^2 U_i} y_i}{\sum_{i=1}^n \frac{1}{\alpha^2 U_i}}, \frac{1}{\sum_{i=1}^n \frac{1}{\alpha^2 U_i}}\right).$$

3. The variance parameter τ^2, is updated much as σ^2 was before:

$$\tau^2|\alpha, \mu, U, \nu, y \sim \text{Gamma}\left(\frac{n\nu}{2}, \frac{\nu}{2}\sum_{i=1}^n \frac{1}{U_i}\right).$$

4. Finally, we must update α^2, which is easy since conditional on all the other parameters in the model it is simply a normal variance parameter:

$$\alpha^2|\mu, \tau^2, U, \nu, y \sim \text{Inv-}\chi^2\left(n, \frac{1}{n}\sum_{i=1}^n \frac{(y_i - \mu)^2}{U_i}\right).$$

The parameters α^2, U, τ in this expanded model are not identified in that the data do not supply enough information to estimate each of them. However, the model as a whole is identified as long as we monitor convergence of the summaries μ, $\sigma = \alpha\tau$, and $V_i = \alpha^2 U_i$ for $i = 1, \ldots, n$. (Or, if the only goal is inference for the original t model, we can simply save μ and σ from the simulations.)
The Gibbs sampler under the expanded parameterizations converges more reliably because the new parameter α breaks the dependence between τ and the V_i's.

We discuss parameter expansion for hierarchical models in Section 15.5 and illustrate in Appendix C.

12.2 Efficient Metropolis jumping rules

For any given posterior distribution, the Metropolis-Hastings algorithm can be implemented in an infinite number of ways. Even after reparameterizing, there are still endless choices in

the jumping rules, J_t. In many situations with conjugate families, the posterior simulation can be performed entirely or in part using the Gibbs sampler, which is not always efficient but generally is easy to program, as we illustrated with the hierarchical normal model in Section 11.6. For nonconjugate models we must rely on Metropolis-Hastings algorithms (either within a Gibbs sampler or directly on the multivariate posterior distribution). The choice of jumping rule then arises.

There are two main classes of simple jumping rules. The first are essentially random walks around the parameter space. These jumping rules are often normal jumping kernels with mean equal to the current value of the parameter and variance set to obtain efficient algorithms. The second approach uses proposal distributions that are constructed to closely approximate the target distribution (either the conditional distribution of a subset in a Gibbs sampler or the joint posterior distribution). In the second case the goal is to accept as many draws as possible with the Metropolis-Hastings acceptance step being used primarily to correct the approximation. There is no natural advantage to altering one parameter at a time except for potential computational savings in evaluating only part of the posterior density at each step.

It is hard to give general advice on efficient jumping rules, but some results have been obtained for random walk jumping distributions that have been useful in many problems. Suppose there are d parameters, and the posterior distribution of $\theta = (\theta_1, \ldots, \theta_d)$, after appropriate transformation, is multivariate normal with known variance matrix Σ. Further suppose that we will take draws using the Metropolis algorithm with a normal jumping kernel centered on the current point and with the same shape as the target distribution: that is, $J(\theta^*|\theta^{t-1}) = \mathrm{N}(\theta^*|\theta^{t-1}, c^2\Sigma)$. Among this class of jumping rules, the most efficient has scale $c \approx 2.4/\sqrt{d}$, where efficiency is defined relative to independent sampling from the posterior distribution. The efficiency of this optimal Metropolis jumping rule for the d-dimensional normal distribution can be shown to be about $0.3/d$ (by comparison, if the d parameters were independent in their posterior distribution, the Gibbs sampler would have efficiency $1/d$, because after every d iterations, a new independent draw of θ would be created). Which algorithm is best for any particular problem also depends on the computation time for each iteration, which in turn depends on the conditional independence and conjugacy properties of the posterior density.

A Metropolis algorithm can also be characterized by the proportion of jumps that are accepted. For the multivariate normal random walk jumping distribution with jumping kernel the same shape as the target distribution, the optimal jumping rule has acceptance rate around 0.44 in one dimension, declining to about 0.23 in high dimensions (roughly $d > 5$). This result suggests an *adaptive* simulation algorithm:

1. Start the parallel simulations with a fixed algorithm, such as a version of the Gibbs sampler, or the Metropolis algorithm with a normal random walk jumping rule shaped like an estimate of the target distribution (using the covariance matrix computed at the joint or marginal posterior mode scaled by the factor $2.4/\sqrt{d}$).
2. After some number of simulations, update the Metropolis jumping rule as follows.
   (a) Adjust the covariance of the jumping distribution to be proportional to the posterior covariance matrix estimated from the simulations.
   (b) Increase or decrease the scale of the jumping distribution if the acceptance rate of the simulations is much too high or low, respectively. The goal is to bring the jumping rule toward the approximate optimal value of 0.44 (in one dimension) or 0.23 (when many parameters are being updated at once using vector jumping).

This algorithm can be improved in various ways, but even in its simple form, we have found it useful for drawing posterior simulations for some problems with d ranging from 1 to 50.

Adaptive algorithms

When an iterative simulation algorithm is 'tuned'—that is, modified while it is running—care must be taken to avoid converging to the wrong distribution. If the updating rule depends on previous simulation steps, then the transition probabilities are more complicated than as stated in the Metropolis-Hastings algorithm, and the iterations will not in general converge to the target distribution. To see the consequences, consider an adaptation that moves the algorithm more quickly through flat areas of the distribution and moves more slowly when the posterior density is changing rapidly. This would make sense as a way of exploring the target distribution, but the resulting simulations would spend disproportionately less time in the flat parts of the distribution and more time in variable parts; the resulting simulation draws would not match the target distribution unless some sort of correction is applied.

To be safe, we typically run any adaptive algorithm in two phases: first, the adaptive phase, where the parameters of the algorithm can be tuned as often as desired to increase the simulation efficiency, and second, a fixed phase, where the adapted algorithm is run long enough for approximate convergence. Only simulations from the fixed phase are used in the final inferences.

12.3 Further extensions to Gibbs and Metropolis

Slice sampling

A random sample of θ from the d-dimensional target distribution, $p(\theta|y)$, is equivalent to a random sample from the area under the distribution (for example, the shaded area under the curve in the illustration of rejection sampling in Figure 10.1 on page 264). Formally, sampling is performed from the d+1-dimensional distribution of (θ, u), where, for any θ, $p(\theta, u|y) \propto 1$ for $u \in [0, p(\theta|y)]$ and 0 otherwise. *Slice sampling* refers to the application of iterative simulation algorithms on this uniform distribution. The details of implementing an effective slice sampling procedure can be complicated, but the method can be applied in great generality and can be especially useful for sampling one-dimensional conditional distributions in a Gibbs sampling structure.

Reversible jump sampling for moving between spaces of differing dimensions

In a number of settings it is desirable to carry out a *trans-dimensional* Markov chain simulation, in which the dimension of the parameter space can change from one iteration to the next. One example where this occurs is in model averaging where a single Markov chain simulation is constructed that includes moves among a number of plausible models (perhaps regression models with different sets of predictors). The 'parameter space' for such a Markov chain simulation includes the traditional parameters along with an indication of the current model. A second example includes finite mixture models (see Chapter 22) in which the number of mixture components is allowed to vary.

It is still possible to perform the Metropolis algorithm in such settings, using the method of *reversible jump* sampling. We use notation corresponding to the case where a Markov chain moves among a number of candidate models. Let $M_k, k = 1, \ldots, K$, denote the candidate models and θ_k the parameter vector for model k with dimension d_k. A key aspect of the reversible jump approach is the introduction of additional random variables that enable the matching of parameter space dimensions across models. Specifically if a move from k to k^* is being considered, then an auxiliary random variable u with jumping distribution $J(u|k, k^*, \theta_k)$ is generated. A series of one-to-one deterministic functions are defined that do the dimension-matching with $(\theta_{k^*}, u^*) = g_{k,k^*}(\theta_k, u)$ and $d_k + \dim(u) =$

$d_{k^*} + \dim(u^*)$. The dimension matching ensures that the balance condition needed to prove the convergence of the Metropolis-Hastings algorithm in Chapter 11 continues to hold here.

We present the reversible jump algorithm in general terms followed by an example. For the general description, let π_k denote the prior probability on model k, $p(\theta_k|M_k)$ the prior distribution for the parameters in model k, and $p(y|\theta_k, M_k)$ the sampling distribution under model k. Reversible jump Markov chain simulation generates samples from $p(k, \theta_k|y)$ using the following three steps at each iteration:

1. Starting in state (k, θ_k) (that is, model M_k with parameter vector θ_k), propose a new model M_{k^*} with probability J_{k,k^*} and generate an augmenting random variable u from proposal density $J(u|k, k^*, \theta_k)$.
2. Determine the proposed model's parameters, $(\theta_{k^*}, u^*) = g_{k,k^*}(\theta_k, u)$.
3. Define the ratio

$$r = \frac{p(y|\theta_{k^*}, M_{k^*})p(\theta_{k^*}|M_{k^*})\pi_{k^*}}{p(y|\theta_k, M_k)p(\theta_k|M_k)\pi_k} \frac{J_{k^*,k}J(u^*|k^*, k, \theta_{k^*})}{J_{k,k^*}J(u|k, k^*, \theta_k)} \left|\frac{\nabla g_{k,k^*}(\theta_k, u)}{\nabla(\theta_k, u)}\right| \tag{12.2}$$

   and accept the new model with probability $\min(r, 1)$.

The resulting posterior draws provide inference about the posterior probability for each model as well as the parameters under that model.

Example. Testing a variance component in a logistic regression

The application of reversible jump sampling, especially the use of the auxiliary random variables u, is seen most easily through an example.

Consider a probit regression for survival of turtles in a natural selection experiment. Let y_{ij} denote the binary response for turtle i in family j with $\Pr(y_{ij} = 1) = p_{ij}$ for $i = 1, \ldots, n_j$ and $j = 1, \ldots, J$. The weight x_{ij} of the turtle is known to affect survival probability, and it is likely that familial factors also play a role. This suggests the model, $p_{ij} = \Phi(\alpha_0 + \alpha_1 x_{ij} + b_j)$. It is natural to model the b_j's as exchangeable family effects, $b_j \sim N(0, \tau^2)$. The prior distribution $p(\alpha_0, \alpha_1, \tau)$ is not central to this discussion so we do not discuss it further here.

Suppose for the purpose of this example that we seek to test whether the variance component τ is needed by running a Markov chain that considers the model with and without the varying intercepts, b_j. As emphasized in Chapters 6–7, we much prefer to fit the model with the variance parameter and assess its importance by examining its posterior distribution. However, it might be of interest to consider the model that allows $\tau = 0$ as a discrete possibility, and we choose this example to illustrate the reversible jump algorithm.

Let M_0 denote the model with $\tau = 0$ (no variance component) and M_1 denote the model including the variance component. We use numerical integration to compute the marginal likelihood $p(y|\alpha_0, \alpha_1, \tau)$ for model M_1. Thus the b_j's are not part of the iterative simulation under model M_1. The reversible jump algorithm takes $\pi_0 = \pi_1 = 0.5$ and $J_{0,0} = J_{0,1} = J_{1,0} = J_{1,1} = 0.5$. At each step we either take a Metropolis step within the current model (with probability 0.5) or propose a jump to the other model. If we are in model 0 and are proposing a jump to model 1, then the auxiliary random variable is $u \sim J(u)$ (scaled inverse-χ^2 in this case) and we define the parameter vector for model 1 by setting $\tau^2 = u$ and leaving α_0 and α_1 as they were in the previous iteration. The ratio (12.2) is then

$$r = \frac{p(y|\alpha_0, \alpha_1, \tau^2, M_1)p(\tau^2)}{p(y|\alpha_0, \alpha_1, M_0)J(\tau^2)},$$

because the prior distributions on α and the models cancel, and the Jacobian of the transformation is 1. The candidate model is accepted with probability $\min(r, 1)$.

There is no auxiliary random variable for going from model 1 to model 0. In that case we merely set $\tau = 0$, and the acceptance probability is the reciprocal of the above. In the example we chose $J(\tau^2)$ based on a pilot analysis of model M_1 (an inverse-χ^2 distribution matching the posterior mean and variance).

Simulated tempering and parallel tempering

Multimodal distributions can pose special problems for Markov chain simulation. The goal is to sample from the entire posterior distribution and this requires sampling from each of the modes with significant posterior probability. Unfortunately it is easy for Markov chain simulations to remain in the neighborhood of a single mode for a long period of time. This occurs primarily when two (or more) modes are separated by regions of extremely low posterior density. Then it is difficult to move from one mode to the other because, for example, Metropolis jumps to the region between the two modes are rejected.

Simulated tempering is one strategy for improving Markov chain simulation performance in this case. As usual, we take $p(\theta|y)$ to be the target density. The algorithm works with a set of $K+1$ distributions $p_k(\theta|y), k = 0, 1, \ldots, K$, where $p_0(\theta|u) = p(\theta|y)$, and $p_1, \ldots, p_K$ are distributions with the same basic shape but with improved opportunities for mixing across the modes, and each of these distributions comes with its own sampler (which might, for example, be a separately tuned Metropolis or HMC algorithm). As usual, the distributions p_k need not be fully specified; it is only necessary that the user can compute unnormalized density functions q_k, where $q_k(\theta) = p_k(\theta|y)$ multiplied by a constant which can depend on y and k but not on the parameters θ. (We write $q_k(\theta)$, but with the understanding that, since the q_k's are built for a particular posterior distribution $p(\theta|y)$, they can in general depend on y.)

One choice for the ladder of unnormalized densities q_k is

$$q_k(\theta) = p(\theta)^{1/T_k},$$

for a set of 'temperature' parameters $T_k > 0$. Setting $T_k = 1$ reduces to the original density, and large values of T_k produce less highly peaked modes. (That is, 'high temperatures' add 'thermal noise' to the system.) A single composite Markov chain simulation is then developed that randomly moves across the $K+1$ distributions, with T_0 set to 1 so that $q_0(\theta) \propto p(\theta|y)$. The state of the composite Markov chain at iteration t is represented by the pair (θ^t, s^t), where s^t is an integer identifying the distribution used at iteration t. Each iteration of the composite Markov chain simulation consists of two steps:

1. A new value θ^{t+1} is selected using the Markov chain simulation with stationary distribution q_{s^t}.
2. A jump from the current sampler s^t to an alternative sampler j is proposed with probability $J_{s^t,j}$. We accept the move with probability $\min(r, 1)$, where

$$r = \frac{c_j q_j(\theta^{t+1}) J_{j,s^t}}{c_{s^t} q_{s^t}(\theta^{t+1}) J_{s^t,j}}.$$

The constants c_k for $k = 0, 1, \ldots, K$ are set adaptively (that is, assigned initial values and then altered after the simulation has run a while) to approximate the inverses of the normalizing constants for the distributions defined by the unnormalized densities q_k. The chain will then spend an approximately equal amount of time in each sampler.

At the end of the Markov chain simulation, only those values of θ simulated from the target distribution (q_0) are used to obtain posterior inferences.

Parallel tempering is a variant of the above algorithm in which $K + 1$ parallel chains

are simulated, one for each density q_k in the ladder. Each chain moves on its own but with occasional flipping of states between chains, with a Metropolis accept-reject rule similar to that in simulated tempering. At convergence, the simulations from chain 0 represent draws from the target distribution.

Other auxiliary variable methods have been developed that are tailored to particular structures of multivariate distributions. For example, highly correlated variables such as arise in spatial statistics can be simulated using *multigrid sampling*, in which computations are done alternately on the original scale and on coarser scales that do not capture the local details of the target distribution but allow faster movement between states.

Particle filtering, weighting, and genetic algorithms

Particle filtering describes a class of simulation algorithms involving parallel chains, in which existing chains are periodically tested and allowed to die, live, or split, with the rule set up so that chains in lower-probability areas of the posterior distribution are more likely to die and those in higher-probability areas are more likely to split. The idea is that a large number of chains can explore the parameter space, with the birth/death/splitting steps allowing the ensemble of chains to more rapidly converge to the target distribution. The probabilities of the different steps are set up so that the stationary distribution of the entire process is the posterior distribution of interest.

A related idea is *weighting*, in which a simulation is performed that converges to a specified but wrong distribution, $g(\theta)$, and then the final draws are weighted by $p(\theta|y)/g(\theta)$. In more sophisticated implementations, this reweighting can be done throughout the simulation process. It can sometimes be difficult or expensive to sample from $p(\theta|y)$ and faster to work with a good approximation g if available. Weighting can be combined with particle filtering by using the weights in the die/live/split probabilities.

Genetic algorithms are similar to particle filtering in having multiple chains that can live or die, but with the elaboration that the updating algorithms themselves can change ('mutate') and combine ('sexual reproduction'). Many of these ideas are borrowed from the numerical analysis literature on optimization but can also be effective in a posterior simulation setting in which the goal is to converge to a distribution rather than to a single best value.

12.4 Hamiltonian Monte Carlo

An inherent inefficiency in the Gibbs sampler and Metropolis algorithm is their random walk behavior—as illustrated in Figures 11.1 and 11.2 on pages 276 and 277, the simulations can take a long time zigging and zagging while moving through the target distribution. Reparameterization and efficient jumping rules can improve the situation (see Sections 12.1 and 12.2), but for complicated models this local random walk behavior remains, especially for high-dimensional target distributions.

Hamiltonian Monte Carlo (HMC) borrows an idea from physics to suppress the local random walk behavior in the Metropolis algorithm, thus allowing it to move much more rapidly through the target distribution. For each component θ_j in the target space, Hamiltonian Monte Carlo adds a 'momentum' variable ϕ_j. Both θ and ϕ are then updated together in a new Metropolis algorithm, in which the jumping distribution for θ is determined largely by ϕ. Each iteration of HMC proceeds via several steps, during which the position and momentum evolve based on rules imitating the behavior of position the steps can move rapidly where possible through the space of θ and even can turn corners in parameter space to preserve the total 'energy' of the trajectory. Hamiltonian Monte Carlo is also called *hybrid Monte Carlo* because it combines MCMC and deterministic simulation methods.

In HMC, the posterior density $p(\theta|y)$ (which, as usual, needs only be computed up

to a multiplicative constant) is augmented by an independent distribution $p(\phi)$ on the momenta, thus defining a joint distribution, $p(\theta, \phi|y) = p(\phi)p(\theta|y)$. We simulate from the joint distribution but we are only interested in the simulations of θ; the vector ϕ is thus an auxiliary variable, introduced only to enable the algorithm to move faster through the parameter space.

In addition to the posterior density (which, as usual, needs to be computed only up to a multiplicative constant), HMC also requires the gradient of the log-posterior density. In practice the gradient must be computed analytically; numerical differentiation requires too many function evaluations to be computationally effective. If θ has d dimensions, this gradient is $\frac{d \log p(\theta|y)}{d\theta} = \left(\frac{d \log p(\theta|y)}{d\theta_1}, \ldots, \frac{d \log p(\theta|y)}{d\theta_d} \right)$. For most of the models we consider in this book, this vector is easy to determine analytically and then program. When writing and debugging the program, we recommend also programming the gradient numerically (using finite differences of the log-posterior density) as a check on the programming of the analytic gradients. If the two subroutines do not return identical results to several decimal places, there is likely a mistake somewhere.

The momentum distribution, $p(\phi)$

It is usual to give ϕ a multivariate normal distribution (recall that ϕ has the same dimension as θ) with mean 0 and covariance set to a prespecified 'mass matrix' M (so called by analogy to the physical model of Hamiltonian dynamics). To keep it simple, we commonly use a diagonal mass matrix, M. If so, the components of ϕ are independent, with $\phi_j \sim \text{N}(0, M_{jj})$ for each dimension $j = 1, \ldots, d$. It can be useful for M to roughly scale with the inverse covariance matrix of the posterior distribution, $(\text{var}(\theta|y))^{-1}$, but the algorithm works in any case; better scaling of M will merely make HMC more efficient.

The three steps of an HMC iteration

HMC proceeds by a series of iterations (as in any Metropolis algorithm), with each iteration having three parts:

1. The iteration begins by updating ϕ with a random draw from its posterior distribution—which, as specified, is the same as its prior distribution, $\phi \sim \text{N}(0, M)$.
2. The main part of the Hamiltonian Monte Carlo iteration is a simultaneous update of (θ, ϕ), conducted in an elaborate but effective fashion via a discrete mimicking of physical dynamics. This update involves L 'leapfrog steps' (to be defined in a moment), each scaled by a factor ϵ. In a leapfrog step, both θ and ϕ are changed, each in relation to the other. The L leapfrog steps proceed as follows:
   Repeat the following steps L times:
   (a) Use the gradient (the vector derivative) of the log-posterior density of θ to make a half-step of ϕ:
   $$\phi \leftarrow \phi + \frac{1}{2}\epsilon \frac{d \log p(\theta|y)}{d\theta}.$$
   (b) Use the 'momentum' vector ϕ to update the 'position' vector θ:
   $$\theta \leftarrow \theta + \epsilon M^{-1}\phi.$$
   Again, M is the mass matrix, the covariance of the momentum distribution $p(\phi)$. If M is diagonal, the above step amounts to scaling each dimension of the θ update. (It might seem redundant to include ϵ in the above expression: why not simply absorb it into M, which can itself be set by the user? The reason is that it can be convenient in tuning the algorithm to alter ϵ while keeping M fixed.)

(c) Again use the gradient of θ to half-update ϕ:

$$\phi \leftarrow \phi + \frac{1}{2}\epsilon \frac{d \log p(\theta|y)}{d\theta}.$$

Except at the first and last step, updates (c) and (a) above can be performed together. The stepping thus starts with a half-step of ϕ, then alternates $L-1$ full steps of the parameter vector θ and the momentum vector ϕ, and concludes with a half-step of ϕ. This algorithm (called a 'leapfrog' because of the splitting of the momentum updates into half steps) is a discrete approximation to physical Hamiltonian dynamics in which both position and momentum evolve in continuous time.

In the limit of ϵ near zero, the leapfrog algorithm preserves the joint density $p(\theta, \phi|y)$. We will not give the proof, but here is some intuition. Suppose the current value of θ is at a flat area of the posterior. Then $\frac{d \log p(\theta|y)}{d\theta}$ will be zero, and in step 2 above, the momentum will remain constant. Thus the leapfrog steps will skate along in θ-space with constant velocity. Now suppose the algorithm moves toward an area of low posterior density. Then $\frac{d \log p(\theta|y)}{d\theta}$ will be negative in this direction, thus in step 2 inducing a decrease in the momentum in the direction of movement. As the leapfrog steps continue to move into an area of lower density in θ-space, the momentum continues to decrease. The decrease in $\log p(\theta|y)$ is matched (in the limit $\epsilon \to 0$, exactly so) by a decrease in the 'kinetic energy,' $\log p(\phi)$. And if iterations continue to move in the direction of decreasing density, the leapfrog steps will slow to zero and then back down or curve around the dip. Now consider the algorithm heading in a direction in which the posterior density is increasing. Then $\frac{d \log p(\theta|y)}{d\theta}$ will be positive in that direction, leading in step 2 to an increase in momentum in that direction. As $\log p(\theta|y)$ increases, $\log p(\phi)$ increases correspondingly until the trajectory eventually moves past or around the mode and then starts to slow down.

For finite ϵ, the joint density $p(\theta, \phi|y)$ does not remain entirely constant during the leapfrog steps but it will vary only slowly if ϵ is small. For reasons we do not discuss here, the leapfrog integrator has the pleasant property that combining L steps of error δ does not produce $L\delta$ error, because the dynamics of the algorithm tend to send the errors weaving back and forth around the exact value that would be obtained by a continuous integration. Keeping the discretization error low is important because of the next part of the HMC algorithm, the accept/reject step.

3. Label θ^{t-1}, ϕ^{t-1} as the value of the parameter and momentum vectors at the start of the leapfrog process and θ^*, ϕ^* as the value after the L steps. In the accept-reject step, we compute

$$r = \frac{p(\theta^*|y)p(\phi^*)}{p(\theta^{t-1}|y)p(\phi^{t-1})}. \tag{12.3}$$

4. Set

$$\theta^t = \begin{cases} \theta^* & \text{with probability } \min(r, 1) \\ \theta^{t-1} & \text{otherwise.} \end{cases}$$

Strictly speaking it would be necessary to set ϕ^t as well, but since we do not care about ϕ in itself, and it gets immediately updated at the beginning of the next iteration (see step 1 above), so there is no need to keep track of it after the accept/reject step.

As with any other MCMC algorithm, we repeat these iterations until approximate convergence, as assessed by $\hat{R}$ being near 1 and the effective sample size being large enough for all quantities of interest; see Section 11.4.

Restricted parameters and areas of zero posterior density

HMC is designed to work with all-positive target densities. If at any point during an iteration the algorithm reaches a point of zero posterior density (for example, if the steps go below zero when updating a parameter that is restricted to be positive), we stop the stepping and give up, spending another iteration at the previous value of θ. The resulting algorithm preserves detailed balance and stays in the positive zone.

An alternative is 'bouncing,' where again the algorithm checks that the density is positive after each step and, if not, changes the sign of the momentum to return to the direction in which it came. This again preserves detailed balance and is typically more efficient than simply rejecting the iteration, for example with a hard boundary for a parameter that is restricted to be positive.

Another way to handle bounded parameters is via transformation, for example taking the logarithm of a parameter constrained to be positive or the logit for a parameter constrained to fall beween 0 and 1, or more complicated joint transformations for sets of parameters that are constrained (for example, if $\theta_1 < \theta_2 < \theta_3$ or if $\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 = 1$). One must then work out the Jacobian of the transformation and use it to determine the log posterior density and its gradient in the new space.

Setting the tuning parameters

HMC can be tuned in three places: (i) the probability distribution for the momentum variables ϕ (which, in our implementation requires specifying the diagonal elements of a covariance matrix, that is, a scale parameter for each of the d dimensions of the parameter vector), (ii) the scaling factor ϵ of the leapfrog steps, and (iii) the number of leapfrog steps L per iteration.

As with the Metropolis algorithm in general, these tuning parameters can be set ahead of time, or they can be altered completely at random (a strategy which can sometimes be helpful in keeping an algorithm from getting stuck), but one has to take care when altering them given information from previous iterations. Except in some special cases, adaptive updating of the tuning parameters alters the algorithm so that it no longer converges to the target distribution. So when we set the tuning parameters, we do so during the warm-up period: that is, we start with some initial settings, then run HMC for a while, then reset the tuning parameters based on the iterations so far, then discard the early iterations that were used for warm-up. This procedure can be repeated if necessary, as long as the saved iterations use only simulations after the last setting of the tuning parameters.

How, then, to set the parameters that govern HMC? We start by setting the scale parameters for the momentum variables to some crude estimate of the scale of the target distribution. (One can also incorporate covariance information but here we will assume a diagonal covariance matrix so that all that is required is the vector of scales.) By default we could simply use the identity matrix.

We then set the product ϵL to 1. This roughly calibrates the HMC algorithm to the 'radius' of the target distribution; that is, L steps, each of length ϵ times the already-chosen scale of ϕ, should roughly take you from one side of the distribution to the other. A default starting point could be $\epsilon = 0.1, L = 10$.

Finally, theory suggests that HMC is optimally efficient when its acceptance rate is approximately 65% (based on an analysis similar to that which finds an optimal 23% acceptance rate for the multidimensional Metropolis algorithm). The theory is based on all sorts of assumptions but seems like a reasonable guideline for optimization in practice. For now we recommend a simple adaptation in which HMC is with its initial settings and then adapted if the average acceptance probability (as computed from the simulations so far) is not close to 65%. If the average acceptance probability is lower, then the leapfrog jumps

are too ambitious and you should lower ϵ and correspondingly increase L (so their product remains 1). Conversely, if the average acceptance probability is much higher than 65%, then the steps are too cautious and we recommend raising ϵ and lowering L (not forgetting that L must be an integer). These rules do not solve all problems, and it should be possible to develop diagnostics to assess the efficiency of HMC to allow for more effective adaptation of the tuning parameters.

Varying the tuning parameters during the run

As with MCMC tuning more generally, any adaptation can go on during the warm-up period, but adaptation performed later on, during the simulations that will be used for inference, can cause the algorithm to converge to the wrong distribution. For example, suppose we were to increase the step size ϵ after high-probability jumps and decrease ϵ when the acceptance probability is low. Such an adaptation seems appealing but would destroy the detailed balance (that is, the property of the algorithm that the flow of probability mass from point A to B is the same as from B to A, for any points A and B in the posterior distribution) that is used to prove that the posterior distribution of interest is the stationary distribution of the Markov chain.

Completely random variation of ϵ and L, however, causes no problems with convergence and can be useful. If we randomly vary the tuning parameters (within specified ranges) from iteration to iteration while the simulation is running, the algorithm has a chance to take long tours through the posterior distribution when possible and make short movements where the iterations are stuck in a cramped part of the space. The price for this variation is some potential loss of optimality, as the algorithm will also take short steps where long tours would be feasible and try for long steps where the space is too cramped for such jumps to be accepted.

Locally adaptive HMC

For difficult HMC problems, it would be desirable for the tuning parameters to vary as the algorithm moves through the posterior distribution, with the mass matrix M scaling to the local curvature of the log density, the step size ϵ getting smaller in areas where the curvature is high, and the number of steps L being large enough for the trajectory to move far through the posterior distribution without being so large that the algorithm circles around and around. To this end, researchers have developed extensions of HMC that adapt without losing detailed balance. These algorithms are more complicated and can require more computations per iteration but can converge more effectively for complicated distributions. We describe two such algorithms here but without giving the details.

The no-U-turn sampler. In the no-U-turn sampler, the number of steps is determined adaptively at each iteration. Instead of running for a fixed number of steps, L, the trajectory in each iteration continues until it turns around (more specifically, until we reach a negative value of the dot product between the momentum variable ϕ and the distance traveled from the position θ at the start of the iteration). This rule essentially sends the trajectory as far as it can go during that iteration. If such a rule is applied alone, the simulations will not converge to the desired target distribution. The full no-U-turn sampler is more complicated, going backward and forward along the trajectory in a way that satisfies detailed balance. Along with this algorithm comes a procedure for adaptively setting the mass matrix M and step size ϵ; these parameters are tuned during the warm-up phase and then held fixed during the later iterations which are kept for the purpose of posterior inference.

Riemannian adaptation. Another approach to optimization is Riemannian adaptation, in which the mass matrix M is set to conform with the local curvature of the log posterior

density at each step. Again, the local adaptation allows the sampler to move much more effectively but the steps of the algorithm need to become more complicated to maintain detailed balance. Riemannian adaptation can be combined with the no-U-turn sampler.

Neither of the above extensions solves all the problems with HMC. The no-U-turn sampler is self-tuning and computationally efficient but, like ordinary Hamiltonian Monte Carlo, has difficulties with very short-tailed and long-tailed distributions, in both cases having difficulties transitioning from the center to the tails, even in one dimension. Riemannian adaptation handles varying curvature and non-exponentially tailed distributions but is impractical in high dimensions.

Combining HMC with Gibbs sampling

There are two ways in which ideas of the Gibbs sampler fit into Hamiltonian Monte Carlo. First, it can make sense to partition variables into blocks, either to simplify computation or to speed convergence. Consider a hierarchical model with J groups, with parameter vector $\theta = (\eta^{(1)}, \eta^{(2)}, \ldots, \eta^{(J)}, \phi)$, where each of the $\eta^{(j)}$'s is itself a vector of parameters corresponding to the model for group j and ϕ is a vector of hyperparameters, and for which the posterior distribution can be factored as, $p(\theta|y) \propto p(\phi) \prod_{j=1}^{J} p(\eta^{(j)}|\phi) p(y^{(j)}|\eta^{(j)})$. In this case, even if it is possible to update the entire vector θ at once using HMC, it may be more effective—in computation speed or convergence—to cycle through $J+1$ updating steps, altering each $\eta^{(j)}$ and then ϕ during each cycle. This way we only have to work with at most one of the likelihood factors, $p(y^{(j)}|\eta^{(j)})$, at each step. Parameter expansion can be used to facilitate quicker mixing through the joint distribution.

The second way in which Gibbs sampler principles can enter HMC is through the updating of discrete variables. Hamiltonian dynamics are only defined on continuous distributions. If some of the parameters in a model are defined on discrete spaces (for example, latent indicators for mixture components, or a parameter that follows a continuous distribution but has a positive probability of being exactly zero), they can be updated using Gibbs steps or, more generally, one-dimensional updates such as Metropolis or slice sampling (see Section 12.3). The simplest approach is to partition the space into discrete and continuous parameters, then alternate HMC updates on the continuous subspace and Gibbs, Metropolis, or slice updates on the discrete components.

12.5 Hamiltonian dynamics for a simple hierarchical model

We illustrate the tuning of Hamiltonian Monte Carlo with the model for the educational testing experiments described in Chapter 5. HMC is not necessary in this problem—the Gibbs sampler works just fine, especially after the parameter expansion which allows more efficient movement of the hierarchical variance parameter (see Section 12.1)—but it is helpful to understand the new algorithm in a simple example. Here we go through all the steps of the algorithm. The code appears in Section C.4, starting on page 601.

In order not to overload our notation, we label the eight school effects (defined as θ_j in Chapter 5) as α_j; the full vector of parameters θ then has $d = 10$ dimensions, corresponding to $\alpha_1, \ldots, \alpha_8, \mu, \tau$.

Gradients of the log posterior density. For HMC we need the gradients of the log posterior density for each of the ten parameters, a set of operations that are easily performed with the normal distributions of this model:

$$\frac{d \log p(\theta|y)}{d \alpha_j} = -\frac{\alpha_j - y_j}{\sigma_j^2} - \frac{\alpha_j - \mu}{\tau^2}, \text{ for } j = 1, \ldots, 8,$$

$$\frac{d \log p(\theta|y)}{d\,\mu} = -\sum_{j=1}^{J} \frac{\mu - \alpha_j}{\tau^2},$$

$$\frac{d \log p(\theta|y)}{d\,\tau} = -\frac{J}{\tau} + \sum_{j=1}^{J} \frac{(\mu - \alpha_j)^2}{\tau^3}.$$

As a debugging step we also compute the gradients numerically using finite differences of ± 0.0001 on each component of θ. Once we have checked that the two gradient routines yield identical results, we use the analytic gradient in the algorithm as it is faster to compute.

The mass matrix for the momentum distribution. As noted above, we want to scale the mass matrix to roughly match the posterior distribution. That said, we typically only have a vague idea of the posterior scale before beginning our computation; thus this scaling is primarily intended to forestall the problems that would arise if there are gross disparities in the scaling of different dimensions. In this case, after looking at the data in Table 5.2 we assign a rough scale of 15 for each of the parameters in the model and crudely set the mass matrix to $\text{Diag}(15, \ldots, 15)$.

Starting values. We run 4 chains of HMC with starting values drawn at random to crudely match the scale of the parameter space, in this case following the idea above and drawing the ten parameters in the model from independent $\text{N}(0, 15^2)$ distributions.

Tuning ϵ and L. To give the algorithm more flexibility, we do not set ϵ and L to fixed values. Instead we choose central values ϵ_0, L_0 and then at each step draw ϵ and L independently from uniform distributions on $(0, 2\epsilon_0)$ and $[1, 2L_0]$, respectively (with the distribution for L being discrete uniform, as L must be an integer). We have no reason to think this particular jittering is ideal; it is just a simple way to vary the tuning parameters in a way that does not interfere with convergence of the algorithm. Following the general advice given above, we start by setting $\epsilon_0 L_0 = 1$ and $L_0 = 10$. We simulate 4 chains for 20 iterations just to check that the program runs without crashing.

We then do some experimentation. We first run 4 chains for 100 iterations and see that the inferences are reasonable (no extreme values, as can sometimes happen when there is poor convergence or a bug in the program) but not yet close to convergence, with several values of $\widehat{R}$ that are more than 2. The average acceptance probabilities of the 4 chains are 0.23, 0.59, 0.02, and 0.57, well below 65%, so we suspect the step size is too large.

We decrease ϵ_0 to 0.05, increase L_0 to 20 (thus keeping $\epsilon_0 L_0$ constant), and rerun the 4 chains for 100 iterations, now getting acceptance rates of 0.72,, 0.87, 0.33, and 0.55, with chains still far from mixing. At this point we increase the number of simulations to 1000. The simulations now are close to convergence, with $\widehat{R}$ less than 1.2 for all parameters, and average acceptance probabilities are more stable, at 0.52, 0.68, 0.75, and 0.51. We then run 4 chains at 10,000 simulations at these tuning parameters and achieve approximate convergence, with $\widehat{R}$ less than 1.1 for all parameters.

In this particular example, HMC is unnecessary, as the Gibbs sampler works fine on an appropriately transformed scale. In larger and more difficult problems, however, Gibbs and Metropolis can be too slow, while HMC can move effectively efficiently move through high-dimensional parameter spaces.

Transforming to $\log \tau$

When running HMC on a model with constrained parameters, the algorithm can go outside the boundary, thus wasting some iterations. One remedy is to transform the space to be unconstrained. In this case, the simplest way to handle the constraint $\tau > 0$ is to transform to $\log \tau$. We then must alter the algorithm in the following ways:

1. We redefine θ as $(\alpha_1, \ldots, \alpha_8, \mu, \log\tau)$ and do all jumping on this new space.
2. The (unnormalized) posterior density $p(\theta|y)$ is multiplied by the Jacobian, τ, so we add $\log\tau$ to the log posterior density used in the calculations.
3. The gradient of the log posterior density changes in two ways: first, we need to account for the new term added just above; second, the derivative for the last component of the gradient is now with respect to $\log\tau$ rather than τ and so must be multiplied by the Jacobian, τ:
$$\frac{d\,\log\,p(\theta|y)}{d\,\log\tau} = -\,(J-1) + \sum_{j=1}^{J}\frac{(\mu-\alpha_j)^2}{\tau^2}.$$
4. We change the mass matrix to account for the transformation. We keep $\alpha_1, \ldots, \alpha_8, \mu$ with masses of 15 (roughly corresponding to a posterior distribution with a scale of 15 in each of these dimensions) but set the mass of $\log\tau$ to 1.
5. We correspondingly change the initial values by drawing the first nine parameters from independent $\mathrm{N}(0, 15^2)$ distributions and $\log\tau$ from $\mathrm{N}(0, 1)$.

HMC runs as before. Again, we start with $\epsilon = 0.1$ and $L = 10$ and then adjust to get a reasonable acceptance rate.

12.6 Stan: developing a computing environment

Hamiltonian Monte Carlo takes a bit of effort to program and tune. In more complicated settings, though, we have found HMC to be faster and more reliable than basic Markov chain simulation algorithms.

To mitigate the challenges of programming and tuning, we have developed a computer program, Stan (Sampling through adaptive neighborhoods) to automatically apply HMC given a Bayesian model. The key steps of the algorithm are data and model input, computation of the log posterior density (up to an arbitrary constant that cannot depend on the parameters in the model) and its gradients, a warm-up phase in which the tuning parameters are set, an implementation of the no-U-turn sampler to move through the parameter space, and convergence monitoring and inferential summaries at the end.

We briefly describe how each of these steps is done in Stan. Instructions and examples for running the program appear in Appendix C.

Entering the data and model

Each line of a Stan model goes into defining the log probability density of the data and parameters, with code for looping, conditioning, computation of intermediate quantities, and specification of terms of the log joint density. Standard distributions such as the normal, gamma, binomial, Poisson, and so forth, are preprogrammed, and arbitrary distributions can be entered by directly programming the log density. Algebraic manipulations and functions such as exp and logit can also be included in the specification; it is all just sent into C++.

To compute gradients, Stan uses automatic analytic differentiation, using an algorithm that parses arbitrary C++ expressions and then applies basic rules of differential calculus to construct a C++ program for the gradient. For computational efficiency, we have preprogrammed the gradients for various standard statistical expressions to make up some of this difference. We use special scalar variable classes that evaluate the function and at the same time construct the full expression tree used to generate the log probability. Then the reverse pass walks backward down the expression tree (visiting every dependent node before any node it depends on), propagating partial derivatives by the chain rule. The walk over the expression tree implicitly employs dynamic programming to minimize the number of

calculations. The resulting autodifferentiation is typically much faster than computing the gradient numerically via finite differences.

In addition to the data, parameters, and model statements, a Stan call also needs the number of chains, the number of iterations per chain, and various control parameters that can be set by default. Starting values can be supplied or else they are generated from preset default random variables.

Setting tuning parameters in the warm-up phase

As noted above, it can be tricky to tune Hamiltonian Monte Carlo for any particular example. The no-U-turn sampler helps with this, as it eliminates the need to assign the number of steps L, but we still need to set the mass matrix M and step size ϵ. During a prespecified warm-up phase of the simulation, Stan adaptively alters M and ϵ using ideas from stochastic optimization in numerical analysis. This adaptation will not always work—for distributions with varying curvature, there will not in general be any single good set of tuning parameters—and if the simulation is having difficulty converging, it can make sense to look at the values of M and ϵ chosen for different chains to better understand what is happening. Convergence can sometimes be improved by reparameterization. More generally, it could make sense to have different tuning parameters for different areas of the distribution—this is related to ideas such as Riemannian adaptation, which at the time of this writing we are incorporating into Stan.

No-U-turn sampler

Stan runs HMC using the no-U-turn sampler, preprocessing where possible by transforming bounded variables to put them on an unconstrained scale. For complicated constraints this cannot always be done automatically and then it can make sense for the user to reparameterize in writing the model. While running, Stan keeps track of acceptance probabilities (as well as the simulations themselves), which can be helpful in getting inside the algorithm if there are problems with mixing of the chains.

Inferences and postprocessing

Stan produces multiple sequences of simulations. For our posterior inferences we discard the iterations from the warm-up period (but we save them as possibly of diagnostic use if the algorithm is not mixing well) and compute $\widehat{R}$ and n_{eff} as described in Section 11.4.

12.7 Bibliographic note

For the relatively simple ways of improving simulation algorithms mentioned in Sections 12.1 and 12.2, Tanner and Wong (1987) discuss data augmentation and auxiliary variables, and Hills and Smith (1992) and Roberts and Sahu (1997) discuss different parameterizations for the Gibbs sampler. Higdon (1998) discusses some more complicated auxiliary variable methods, and Liu and Wu (1999), van Dyk and Meng (2001), and Liu (2003) present different approaches to parameter expansion. The results on acceptance rates for efficient Metropolis jumping rules appear in Gelman, Roberts, and Gilks (1995); more general results for Metropolis-Hastings algorithms appear in Roberts and Rosenthal (2001) and Brooks, Giudici, and Roberts (2003).

Gelfand and Sahu (1994) discuss the difficulties of maintaining convergence to the target distribution when adapting Markov chain simulations, as discussed at the end of Section 12.2. Andrieu and Robert (2001) and Andrieu and Thoms (2008) consider adaptive Markov chain Monte Carlo algorithms.

Slice sampling is discussed by Neal (2003), and simulated tempering is discussed by Geyer and Thompson (1993) and Neal (1996b). Besag et al. (1995) and Higdon (1998) review several ideas based on auxiliary variables that have been useful in high-dimensional problems arising in genetics and spatial models.

Reversible jump MCMC was introduced by Green (1995); see also Richardson and Green (1997) and Brooks, Giudici, and Roberts (2003) for more on trans-dimensional MCMC.

Mykland, Tierney, and Yu (1994) discuss an approach to MCMC in which the algorithm has regeneration points, or subspaces of θ, so that if a finite sequence starts and ends at a regeneration point, it can be considered as an exact (although dependent) sample from the target distribution. Propp and Wilson (1996) and Fill (1998) introduce a class of MCMC algorithms called perfect simulation in which, after a certain number of iterations, the simulations are known to have exactly converged to the target distribution.

The book by Liu (2001) covers a wide range of advanced simulation algorithms including those discussed in this chapter. The monograph by Neal (1993) also overviews many of these methods. Hamiltonian Monte Carlo was introduced by Duane et al. (1987) in the physics literature and Neal (1994) for statistics problems. Neal (2011) reviews HMC, Hoffman and Gelman (2013) introduce the no-U-turn sampler, and Girolami and Calderhead (2011) introduce Riemannian updating; see also Betancourt and Stein (2011) and Betancourt (2012, 2013). Romeel (2011) explains how leapfrog steps tend to reduce discretization error in HMC. Leimkuhler and Reich (2004) discuss the mathematics in more detail. Griewank and Walther (2008) is a standard reference on algorithmic differentiation.

12.8 Exercises

1. Efficient Metropolis jumping rules: Repeat the computation for Exercise 11.2 using the adaptive algorithm given in Section 12.2.
2. Simulated tempering: Consider the Cauchy model, $y_i \sim \text{Cauchy}(\theta, 1), i = 1, \ldots, n$, with two data points, $y_1 = 1.3, y_2 = 15.0$.
   (a) Graph the posterior density.
   (b) Program the Metropolis algorithm for this problem using a symmetric Cauchy jumping distribution. Tune the scale parameter of the jumping distribution appropriately.
   (c) Program simulated tempering with a ladder of 10 inverse-temperatures, 0.1, ..., 1.
   (d) Compare your answers in (b) and (c) to the graph in (a).
3. Hamiltonian Monte Carlo: Program HMC in R for the bioassay logistic regression example from Chapter 3.
   (a) Code the gradients analytically and numerically and check that the two programs give the same result.
   (b) Pick reasonable starting values for the mass matrix, step size, and number of steps.
   (c) Tune the algorithm to an approximate 65% acceptance rate.
   (d) Run 4 chains long enough so that each has an effective sample size of at least 100. How many iterations did you need?
   (e) Check that your inferences are consistent with those from the direct approach in Chapter 3.
4. Coverage of intervals and rejection sampling: Consider the following model: $y_j \sim \text{Binomial}(n_j, \theta_j)$, where $\theta_j = \text{logit}^{-1}(\alpha + \beta x_j)$, for $j = 1, \ldots, J$, and with independent prior distributions, $\alpha \sim t_4(0, 2^2)$ and $\beta \sim t_4(0, 1)$. Assume $J = 10$, the x_j values are randomly drawn from a $\text{U}(1, 1)$ distribution, and $n_j \sim \text{Poisson}^+(5)$, where Poisson^+ is the Poisson distribution restricted to positive values.

(a) Sample a dataset at random from the model, estimate α and β using Stan, and make a graph simultaneously displaying the data, the fitted model, and uncertainty in the fit (shown via a set of inverse logit curves that are thin and gray (in R, `lwd=.5, col="gray"`)).

(b) Did Stan's posterior 50% interval for α contain its true value? How about β?

(c) Use rejection sampling to get 1000 independent posterior draws from (α, β).