1

Introduction to Markov Chain Monte Carlo

Charles J. Geyer

1.1 History

Despite a few notable uses of simulation of random processes in the pre-computer era (Hammersley and Handscomb, 1964, Section 1.2; Stigler, 2002, Chapter 7), practical widespread use of simulation had to await the invention of computers. Almost as soon as computers were invented, they were used for simulation (Hammersley and Handscomb, 1964, Section 1.2). The name "Monte Carlo" started as cuteness—gambling was then (around 1950) illegal in most places, and the casino at Monte Carlo was the most famous in the world—but it soon became a colorless technical term for simulation of random processes.

Markov chain Monte Carlo (MCMC) was invented soon after ordinary Monte Carlo at Los Alamos, one of the few places where computers were available at the time. Metropolis et al. (1953)* simulated a liquid in equilibrium with its gas phase. The obvious way to find out about the thermodynamic equilibrium is to simulate the dynamics of the system, and let it run until it reaches equilibrium. The *tour de force* was their realization that they did not need to simulate the exact dynamics; they only needed to simulate some Markov chain having the same equilibrium distribution. Simulations following the scheme of Metropolis et al. (1953) are said to use the *Metropolis algorithm*. As computers became more widely available, the Metropolis algorithm was widely used by chemists and physicists, but it did not become widely known among statisticians until after 1990. Hastings (1970) generalized the Metropolis algorithm, and simulations following his scheme are said to use the *Metropolis–Hastings* algorithm. A special case of the Metropolis–Hastings algorithm was introduced by Geman and Geman (1984), apparently without knowledge of earlier work. Simulations following their scheme are said to use the *Gibbs sampler*. Much of Geman and Geman (1984) discusses optimization to find the posterior mode rather than simulation, and it took some time for it to be understood in the spatial statistics community that the Gibbs sampler simulated the posterior distribution, thus enabling full Bayesian inference of all kinds. A methodology that was later seen to be very similar to the Gibbs sampler was introduced by Tanner and Wong (1987), again apparently without knowledge of earlier work. To this day, some refer to the Gibbs sampler as "data augmentation" following these authors. Gelfand and Smith (1990) made the wider Bayesian community aware of the Gibbs sampler, which up to that time had been known only in the spatial statistics community. Then it took off; as of this writing, a search for Gelfand and Smith (1990) on Google Scholar yields 4003 links to other works. It was rapidly realized that most Bayesian inference could

* The fifth author was Edward Teller, the "father of the hydrogen bomb."

be done by MCMC, whereas very little could be done without MCMC. It took a while for researchers to properly understand the theory of MCMC (Geyer, 1992; Tierney, 1994) and that all of the aforementioned work was a special case of the notion of MCMC. Green (1995) generalized the Metropolis–Hastings algorithm, as much as it can be generalized. Although this terminology is not widely used, we say that simulations following his scheme use the *Metropolis–Hastings–Green* algorithm. MCMC is not used only for Bayesian inference. Likelihood inference in cases where the likelihood cannot be calculated explicitly due to missing data or complex dependence can also use MCMC (Geyer, 1994, 1999; Geyer and Thompson, 1992, 1995, and references cited therein).

1.2 Markov Chains

A sequence $X_1, X_2, \ldots$ of random elements of some set is a *Markov chain* if the conditional distribution of X_{n+1} given $X_1, \ldots, X_n$ depends on X_n only. The set in which the X_i take values is called the *state space* of the Markov chain.

A Markov chain has *stationary transition probabilities* if the conditional distribution of X_{n+1} given X_n does not depend on n. This is the main kind of Markov chain of interest in MCMC. Some kinds of adaptive MCMC (Chapter 4, this volume) have nonstationary transition probabilities. In this chapter we always assume stationary transition probabilities.

The joint distribution of a Markov chain is determined by

- The marginal distribution of X_1, called the *initial distribution*
- The conditional distribution of X_{n+1} given X_n, called the *transition probability distribution* (because of the assumption of stationary transition probabilities, this does not depend on n)

People introduced to Markov chains through a typical course on stochastic processes have usually only seen examples where the state space is finite or countable. If the state space is finite, written $\{x_1, \ldots, x_n\}$, then the initial distribution can be associated with a vector $\lambda = (\lambda_1, \ldots, \lambda_n)$ defined by

$$\Pr(X_1 = x_i) = \lambda_i, \quad i = 1, \ldots, n,$$

and the transition probabilities can be associated with a matrix P having elements p_{ij} defined by

$$\Pr(X_{n+1} = x_j \mid X_n = x_i) = p_{ij}, \quad i = 1, \ldots, n \quad \text{and} \quad j = 1, \ldots, n.$$

When the state space is countably infinite, we can think of an infinite vector and matrix. But most Markov chains of interest in MCMC have uncountable state space, and then we cannot think of the initial distribution as a vector or the transition probability distribution as a matrix. We must think of them as an unconditional probability distribution and a conditional probability distribution.

1.3 Computer Programs and Markov Chains

Suppose you have a computer program

```
Initialize x
repeat {
    Generate pseudorandom change to x
    Output x
}
```

If x is the entire state of the computer program exclusive of random number generator seeds (which we ignore, pretending pseudorandom is random), this is MCMC. It is important that x must be the entire state of the program. Otherwise the resulting stochastic process need not be Markov.

There is not much structure here. Most simulations can be fit into this format. Thus most simulations can be thought of as MCMC if the entire state of the computer program is considered the state of the Markov chain. Hence, MCMC is a very general simulation methodology.

1.4 Stationarity

A sequence $X_1, X_2, \ldots$ of random elements of some set is called a *stochastic process* (Markov chains are a special case). A stochastic process is *stationary* if for every positive integer k the distribution of the k-tuple

$$(X_{n+1}, \ldots, X_{n+k})$$

does not depend on n. A Markov chain is stationary if it is a stationary stochastic process. In a Markov chain, the conditional distribution of $(X_{n+2}, \ldots, X_{n+k})$ given X_{n+1} does not depend on n. It follows that a Markov chain is stationary if and only if the marginal distribution of X_n does not depend on n.

An initial distribution is said to be *stationary* or *invariant* or *equilibrium* for some transition probability distribution if the Markov chain specified by this initial distribution and transition probability distribution is stationary. We also indicate this by saying that the transition probability distribution *preserves* the initial distribution.

Stationarity implies stationary transition probabilities, but not vice versa. Consider an initial distribution concentrated at one point. The Markov chain can be stationary if and only if all iterates are concentrated at the same point, that is, $X_1 = X_2 = \ldots$, so the chain goes nowhere and does nothing. Conversely, any transition probability distribution can be combined with any initial distribution, including those concentrated at one point. Such a chain is usually not stationary (even though the transition probabilities are stationary).

Having an equilibrium distribution is an important property of a Markov chain transition probability. In Section 1.8 below, we shall see that MCMC samples the equilibrium distribution, whether the chain is stationary or not. Not all Markov chains have equilibrium distributions, but all Markov chains used in MCMC do. The Metropolis–Hastings–Green (MHG) algorithm (Sections 1.12.2, 1.17.3.2, and 1.17.4.1 below) constructs transition probability mechanisms that preserve a specified equilibrium distribution.

1.5 Reversibility

A transition probability distribution is *reversible* with respect to an initial distribution if, for the Markov chain $X_1, X_2, \ldots$ they specify, the distribution of pairs (X_i, X_{i+1}) is exchangeable.

A Markov chain is *reversible* if its transition probability is reversible with respect to its initial distribution. Reversibility implies stationarity, but not vice versa. A reversible Markov chain has the same laws running forward or backward in time, that is, for any i and k the distributions of $(X_{i+1}, \ldots, X_{i+k})$ and $(X_{i+k}, \ldots, X_{i+1})$ are the same. Hence the name.

Reversibility plays two roles in Markov chain theory. All known methods for constructing transition probability mechanisms that preserve a specified equilibrium distribution in non-toy problems are special cases of the MHG algorithm, and all of the elementary updates constructed by the MHG algorithm are reversible (which accounts for its other name, the "reversible jump" algorithm). Combining elementary updates by composition (Section 1.12.7 below) may produce a combined update mechanism that is not reversible, but this does not diminish the key role played by reversibility in constructing transition probability mechanisms for MCMC. The other role of reversibility is to simplify the Markov chain central limit theorem (CLT) and asymptotic variance estimation. In the presence of reversibility the Markov chain CLT (Kipnis and Varadhan, 1986; Roberts and Rosenthal, 1997) is much sharper and the conditions are much simpler than without reversibility. Some methods of asymptotic variance estimation (Section 1.10.2 below) only work for reversible Markov chains but are much simpler and more reliable than analogous methods for nonreversible chains.

1.6 Functionals

If $X_1, X_2, \ldots$ is a stochastic process and g is a real-valued function on its state space, then the stochastic process $g(X_1), g(X_2), \ldots$ having state space $\mathbb{R}$ is said to be a *functional* of $X_1, X_2, \ldots$.

If $X_1, X_2, \ldots$ is a Markov chain, then a functional $g(X_1), g(X_2), \ldots$ is usually not a Markov chain. The conditional distribution of X_{n+1} given $X_1, \ldots, X_n$ depends only on X_n, but this does not, in general, imply that the conditional distribution of $g(X_{n+1})$ given $g(X_1), \ldots, g(X_n)$ depends only on $g(X_n)$. Nevertheless, functionals of Markov chains have important properties not shared by other stochastic processes.

1.7 The Theory of Ordinary Monte Carlo

Ordinary Monte Carlo (OMC), also called "independent and identically distributed (i.i.d.) Monte Carlo" or "good old-fashioned Monte Carlo," is the special case of MCMC in which $X_1, X_2, \ldots$ are independent and identically distributed, in which case the Markov chain is stationary and reversible.

Suppose you wish to calculate an expectation

$$\mu = \mathrm{E}\{g(X)\}, \tag{1.1}$$

where g is a real-valued function on the state space, but you cannot do it by exact methods (integration or summation using pencil and paper, a computer algebra system, or exact numerical methods). Suppose you can simulate $X_1, X_2, \ldots$ i.i.d. having the same distribution as X. Define

$$\hat{\mu}_n = \frac{1}{n} \sum_{i=1}^{n} g(X_i). \tag{1.2}$$

If we introduce the notation $Y_i = g(X_i)$, then the Y_i are i.i.d. with mean μ and variance

$$\sigma^2 = \operatorname{var}\{g(X)\}, \tag{1.3}$$

$\hat{\mu}_n$ is the sample mean of the Y_i, and the CLT says that

$$\hat{\mu}_n \approx N\left(\mu, \frac{\sigma^2}{n}\right). \tag{1.4}$$

The variance in the CLT can be estimated by

$$\hat{\sigma}_n^2 = \frac{1}{n} \sum_{i=1}^{n} \bigl(g(X_i) - \hat{\mu}_n\bigr)^2, \tag{1.5}$$

which is the empirical variance of the Y_i. Using the terminology of Section 1.6, we can also say that $\hat{\mu}_n$ is the sample mean of the functional $g(X_1), g(X_2), \ldots$ of $X_1, X_2, \ldots$.

The theory of OMC is just elementary statistics. For example, $\hat{\mu}_n \pm 1.96 \cdot \hat{\sigma}_n / \sqrt{n}$ is an asymptotic 95% confidence interval for μ. Note that OMC obeys what an elementary statistics text (Freedman et al., 2007) calls the *square root law:* statistical accuracy is inversely proportional to the square root of the sample size. Consequently, the accuracy of Monte Carlo methods is limited. Each additional significant figure, a tenfold increase in accuracy, requires a hundredfold increase in the sample size.

The only tricky issue is that the randomness involved is the pseudorandomness of computer simulation, rather than randomness of real-world phenomena. Thus it is a good idea to use terminology that emphasizes the difference. We call Equation 1.2 the *Monte Carlo approximation* or *Monte Carlo calculation* of μ, rather than the "point estimate" or "point estimator" of μ, as we would if not doing Monte Carlo. We call n the *Monte Carlo sample size*, rather than just the "sample size." We call $\hat{\sigma}_n / \sqrt{n}$ the *Monte Carlo standard error* (MCSE), rather than just the "standard error." We also do not refer to Equation 1.1 as an unknown parameter, even though we do not know its value. It is simply the expectation we are trying to calculate, known in principle, although unknown in practice, since we do not know how to calculate it other than by Monte Carlo approximation.

It is especially important to use this terminology when applying Monte Carlo to statistics. When the expectation (Equation 1.1) arises in a statistical application, there may already be a sample size in this application, which is unrelated to the Monte Carlo sample size, and there may already be standard errors unrelated to MCSEs. It can be hopelessly confusing if these are not carefully distinguished.

1.8 The Theory of MCMC

The theory of MCMC is just like the theory of OMC, except that stochastic dependence in the Markov chain changes the standard error. We start as in OMC with an expectation (Equation 1.1) that we cannot do other than by Monte Carlo. To begin the discussion, suppose that $X_1, X_2, \ldots$ is a stationary Markov chain having initial distribution the same as the distribution of X. We assume that the Markov chain CLT (Equation 1.4) holds, where now

$$\sigma^2 = \operatorname{var}\{g(X_i)\} + 2\sum_{k=1}^{\infty} \operatorname{cov}\{g(X_i), g(X_{i+k})\} \tag{1.6}$$

(this formula is correct only for stationary Markov chains; see below for nonstationary chains). Since the asymptotic variance (Equation 1.6) is more complicated than the i.i.d. case (Equation 1.3), it cannot be estimated by Equation 1.5. It can, however, be estimated in several ways discussed below (Section 1.10). Conditions for the Markov chain CLT to hold (Chan and Geyer, 1994; Jones, 2004; Roberts and Rosenthal, 1997, 2004; Tierney, 1994) are beyond the scope of this chapter.

Now we come to a somewhat confusing issue. We never use stationary Markov chains in MCMC, because if we could simulate X_1 so that it has the invariant distribution, then we could also simulate $X_2, X_3, \ldots$ in the same way and do OMC. It is a theorem, however, that, under a condition (Harris recurrence) that is easier to verify than the CLT (Chan and Geyer, 1994; Tierney, 1994), if the CLT holds for one initial distribution and transition probability, then it holds for all initial distributions and that same transition probability (Meyn and Tweedie, 1993, Proposition 17.1.6), and the asymptotic variance is the same for all initial distributions. Although the theoretical asymptotic variance formula (Equation 1.6) contains variances and covariances for the stationary Markov chain, it also gives the asymptotic variance for nonstationary Markov chains having the same transition probability distribution (but different initial distributions). In practice, this does not matter, because we can never calculate (Equation 1.6) exactly except in toy problems and must estimate it from our simulations.

1.8.1 Multivariate Theory

Suppose that we wish to approximate by Monte Carlo (Equation 1.1) where we change notation so that μ is a vector with components μ_r and $g(x)$ is a vector with components $g_r(x)$. Our Monte Carlo estimator is still given by Equation 1.2, which is now also a vector equation because each $g(X_i)$ is a vector. Then the multivariate Markov chain CLT says that

$$\hat{\mu}_n \approx N(\mu, n^{-1}\Sigma),$$

where

$$\Sigma = \operatorname{var}\{g(X_i)\} + 2\sum_{k=1}^{\infty} \operatorname{cov}\{g(X_i), g(X_{i+k})\}, \tag{1.7}$$

and where, although the right-hand sides of Equations 1.6 and 1.7 are the same, they mean different things: in Equation 1.7 $\operatorname{var}\{g(X_i)\}$ denotes the matrix with components $\operatorname{cov}\{g_r(X_i), g_s(X_i)\}$ and $\operatorname{cov}\{g(X_i), g(X_{i+k})\}$ denotes the matrix with components $\operatorname{cov}\{g_r(X_i), g_s(X_{i+k})\}$.

Conditions for the multivariate CLT to hold are essentially the same as for the univariate CLT. By the Cramér–Wold theorem, the multivariate convergence in distribution $Z_n \xrightarrow{\mathcal{D}} Z$ holds if and only if the univariate convergence in distribution $t'Z_n \xrightarrow{\mathcal{D}} t'Z$ holds for every nonrandom vector t. Thus the multivariate CLT essentially follows from the univariate CLT, and is often not discussed. It is important, however, for users to understand that the multivariate CLT does hold and can be used when needed.

1.8.2 The Autocovariance Function

We introduce terminology for the covariances that appear in Equation 1.6:

$$\gamma_k = \operatorname{cov}\{g(X_i), g(X_{i+k})\} \tag{1.8}$$

is called the *lag-k autocovariance* of the functional $g(X_1), g(X_2), \ldots$. Recall that in Equation 1.8 as in Equation 1.6 the covariances refer to the stationary chain with the same transition probability distribution as the chain being used. The variance that appears in Equation 1.6 is then γ_0. Hence, (Equation 1.6) can be rewritten

$$\sigma^2 = \gamma_0 + 2\sum_{k=1}^{\infty} \gamma_k. \tag{1.9}$$

The function $k \mapsto \gamma_k$ is called the *autocovariance function* of the functional $g(X_1), g(X_2), \ldots$, and the function $k \mapsto \gamma_k/\gamma_0$ is called the *autocorrelation function* of this functional.

The natural estimator of the autocovariance function is

$$\hat{\gamma}_k = \frac{1}{n}\sum_{i=1}^{n-k}[g(X_i) - \hat{\mu}_n][g(X_{i+k}) - \hat{\mu}_n] \tag{1.10}$$

It might be thought that one should divide by $n-k$ instead of n, but the large k terms are already very noisy so dividing by $n-k$ only makes a bad situation worse. The function $k \mapsto \hat{\gamma}_k$ is called the *empirical autocovariance function* of the functional $g(X_1), g(X_2), \ldots$, and the function $k \mapsto \hat{\gamma}_k/\hat{\gamma}_0$ is called the *empirical autocorrelation function* of this functional.

1.9 AR(1) Example

We now look at a toy problem for which exact calculation is possible. An AR(1) process (AR stands for autoregressive) is defined recursively by

$$X_{n+1} = \rho X_n + Y_n, \tag{1.11}$$

where Y_n are i.i.d. $N(0, \tau^2)$ and X_1 may have any distribution with finite variance. From Equation 1.11 we get

$$\operatorname{cov}(X_{n+k}, X_n) = \rho \operatorname{cov}(X_{n+k-1}, X_n) = \ldots = \rho^{k-1} \operatorname{cov}(X_{n-1}, X_n) = \rho^k \operatorname{var}(X_n). \tag{1.12}$$

If the process is stationary, then

$$\operatorname{var}(X_n) = \operatorname{var}(X_{n+1}) = \rho^2 \operatorname{var}(X_n) + \operatorname{var}(Y_n)$$

so

$$\operatorname{var}(X_n) = \frac{\tau^2}{1 - \rho^2} \tag{1.13}$$

and since variances are nonnegative, we must have $\rho^2 < 1$. Since a linear combination of independent normal random variables is normal, we see that the normal distribution with mean zero and variance (Equation 1.13) is invariant. Define υ^2 to be another notation for the right-hand side of Equation 1.13 so the invariant distribution is $N(0, \upsilon^2)$.

It can be shown that this is the unique invariant distribution and this Markov chain obeys the CLT. The variance in the CLT is

$$\begin{aligned}
\sigma^2 &= \operatorname{var}(X_i) + 2 \sum_{k=1}^{\infty} \operatorname{cov}(X_i, X_{i+k}) \\
&= \frac{\tau^2}{1 - \rho^2} \left(1 + 2 \sum_{k=1}^{\infty} \rho^k \right) \\
&= \frac{\tau^2}{1 - \rho^2} \left(1 + \frac{2\rho}{1 - \rho} \right) \\
&= \frac{\tau^2}{1 - \rho^2} \cdot \frac{1 + \rho}{1 - \rho} \\
&= \upsilon^2 \cdot \frac{1 + \rho}{1 - \rho}.
\end{aligned} \tag{1.14}$$

1.9.1 A Digression on Toy Problems

It is hard to know what lessons to learn from a toy problem. Unless great care is taken to point out which features of the toy problem are like real applications and which unlike, readers may draw conclusions that do not apply to real-world problems.

Here we are supposed to pretend that we do not know the invariant distribution, and hence we do not know that the expectation we are trying to estimate, $\mu = \mathrm{E}(X)$, where X has the invariant distribution, is zero.

We cannot be interested in any functional of the Markov chain other than the one induced by the identity function, because we cannot do the analog of Equation 1.14 for any function g other than the identity function, and thus would not have a closed-form expression for the variance in the Markov chain CLT, which is the whole point of this toy problem.

Observe that Equation 1.14 goes to infinity as $\rho \to 1$. Thus in order to obtain a specified accuracy for $\hat{\mu}_n$ as an approximation to μ, say $\sigma/\sqrt{n} = \varepsilon$, we may need a very large Monte Carlo sample size n. How large n must be depends on how close ρ is to one. When we pretend that we do not know the asymptotic variance (Equation 1.14), which we should do because the asymptotic variance is never known in real applications, all we can conclude is that we may need the Monte Carlo sample size to be very large and have no idea how large.

We reach the same conclusion if we are only interested in approximation error relative to the standard deviation υ of the invariant distribution, because

$$\frac{\sigma^2}{\upsilon^2} = \frac{1+\rho}{1-\rho} \tag{1.15}$$

also goes to infinity as $\rho \to 1$.

1.9.2 Supporting Technical Report

In order to avoid including laborious details of examples while still making all examples fully reproducible, those details are relegated to a technical report (Geyer, 2010a) or the vignettes for the R package `mcmc` (Geyer, 2010b). All calculations in this technical report or those package vignettes are done using the R function `Sweave`, so all results in them are actually produced by the code shown therein and hence are fully reproducible by anyone who has R. Moreover, anyone can download the `Sweave` source for the technical report from the URL given in the references at the end of this chapter or find the `Sweave` source for the package vignettes in the `doc` directory of any installation of the `mcmc` package, separate the R from the LaTeX using the `Stangle` function, and play with it to see how the examples work.

1.9.3 The Example

For our example, we choose $\rho = 0.99$ and Monte Carlo sample size $n = 10^4$. This makes the MCSE about 14% of the standard deviation of the invariant distribution, which is a pretty sloppy approximation. To get the relative MCSE down to 10%, we would need $n = 2 \times 10^4$. To get the relative MCSE down to 1%, we would need $n = 2 \times 10^6$.

Figure 1.1 shows a time series plot of one MCMC run for this AR(1) process. From this plot we can see that the series seems stationary—there is no obvious trend or change in spread. We can also get a rough idea of how much dependence there is in the chain by

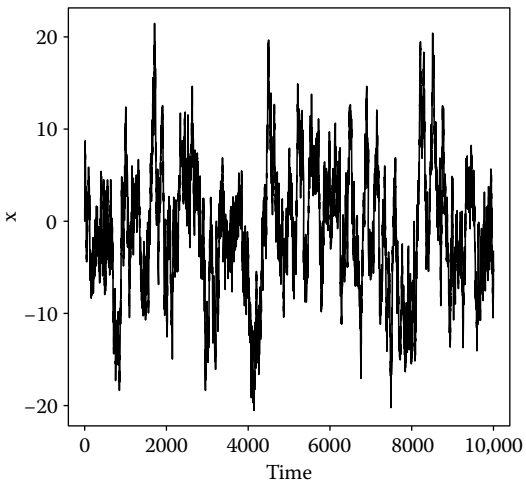

FIGURE 1.1
Time series plot for AR(1) example.

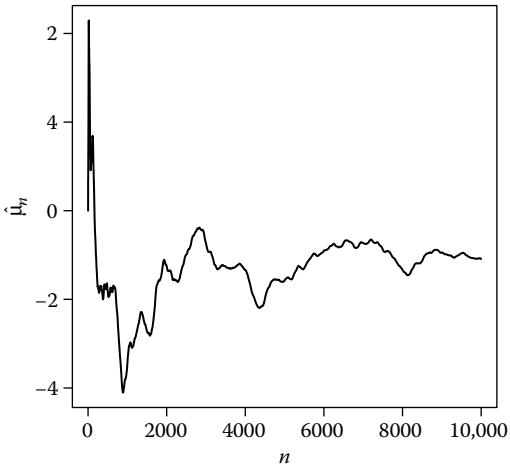

FIGURE 1.2
Running averages plot for AR(1) example.

counting large wiggles. The ratio of the variance in the CLT to the variance of the invariant distribution (Equation 1.15) is 199 for this example. Hence, this MCMC sample is about as useful as an i.i.d. sample with the same marginal distribution of sample size $10^4/199 \approx 50$.

Figure 1.2 shows a running averages plot for the same run shown in Figure 1.1. For some reason, these running averages plots seem popular among MCMC users although they provide no useful information. We know that MCMC, like OMC, obeys the square root law. A plot like Figure 1.2 does illustrate that $1/\sqrt{n}$ is a decreasing function of n, but not much else. Elementary statistics texts (Freedman et al., 2007, p. 276) often include one (and only one) figure like our Figure 1.2 to illustrate to naive students how the law of averages works. We have included Figure 1.2 only as an example of what not to do. In particular, such running averages plots should never be used to illustrate talks, since they tell the audience nothing they do not already know. Show a time series plot, like Figure 1.1, instead.

Figure 1.3 shows an autocorrelation plot for the same run shown in Figure 1.1. The black bars show the empirical autocorrelation function (ACF) defined in Section 1.8.2. We could let the domain of the ACF be zero to $n - 1$, but the R function `acf` cuts the plot at the argument `lag.max`. The `acf` function automatically adds the horizontal dashed lines, which the documentation for `plot.acf` says are 95% confidence intervals assuming white noise input. The dotted curve is the simulation truth autocorrelation function ρ^k derived from Equation 1.12. In the spirit of this toy problem, we are supposed to pretend we do not know the dotted curve, since we would not have its analog in any real application. We can see, however, how well (not very) the empirical ACF matches the theoretical ACF.

It should come as no surprise that the empirical ACF estimates the theoretical ACF less well than $\hat{\mu}_n$ estimates μ. Even in i.i.d. sampling, the mean is always much better estimated than the variance.

The ACF is well enough estimated, however, to give some idea how far significant autocorrelation extends in our Markov chain. Of course, the theoretical autocorrelation is nonzero for all lags, no matter how large, but we know (although we pretend we do not) that they decrease exponentially fast. They are not practically significantly different from zero past lag 500.

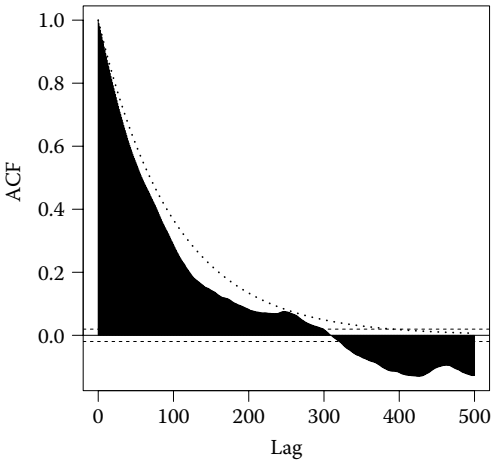

FIGURE 1.3
Autocorrelation plot for AR(1) Example. Dashed lines: 95% confidence intervals assuming white noise input. Dotted curve: simulation truth autocorrelation function.

1.10 Variance Estimation

Many methods of variance estimation have been proposed. Most come from the time series literature and are applicable to arbitrary stationary stochastic processes, not just to Markov chains. We will cover only a few very simple, but very effective, methods.

1.10.1 Nonoverlapping Batch Means

A *batch* is simply a subsequence of consecutive iterates of the Markov chain $X_{k+1}, \ldots, X_{k+b}$. The number b is called the *batch length*. If we assume the Markov chain is stationary, then all batches of the same length have the same joint distribution, and the CLT applies to each batch. The batch mean

$$\frac{1}{b} \sum_{j=1}^{b} g(X_{k+j})$$

is a Monte Carlo approximation of the expectation (Equation 1.1) we are trying to calculate, and its distribution is approximately $N(\mu, \sigma^2/b)$, where, as before, σ^2 is given by Equation 1.6. A batch of length b is just like the entire run of length n, except for length. The sample mean of a batch of length b is just like the sample mean of the entire run of length n, except that the asymptotic variance is σ^2/b instead of σ^2/n.

Suppose b divides n evenly. Divide the whole run into m nonoverlapping batches of length b. Average these batches:

$$\hat{\mu}_{b,k} = \frac{1}{b} \sum_{i=b(k-1)+1}^{bk} g(X_i). \tag{1.16}$$

Then

$$\frac{1}{m}\sum_{k=1}^{m}(\hat{\mu}_{b,k} - \hat{\mu}_n)^2 \tag{1.17}$$

estimates σ^2/b.

It is important to understand that the stochastic process $\hat{\mu}_{b,1}, \hat{\mu}_{b,2}, \ldots$ is also a functional of a Markov chain, not the original Markov chain but a different one. If S is the state space of the original Markov chain $X_1, X_2, \ldots$, then the batches

$$(X_{b(k-1)+1}, \ldots, X_{kb}), \quad k = 1, 2, \ldots$$

also form a Markov chain with state space S^b, because the conditional distribution of one batch $(X_{b(k-1)+1}, \ldots, X_{kb})$ given the past history actually depends only on $X_{b(k-1)}$, which is a component of the immediately preceding batch. The batch means are a functional of this Markov chain of batches.

Figure 1.4 shows a batch mean plot for the same run shown in Figure 1.1. The batch length is 500, the run length is 10^4, so the number of batches is 20. Like the running averages plot (Figure 1.2), we do not recommend this kind of plot for general use, because it does not show anything a sophisticated MCMC user should not already know. It is useful to show such a plot (once) in a class introducing MCMC, to illustrate the point that the stochastic process shown is a functional of a Markov chain. It is not useful for talks about MCMC.

Figure 1.5 shows the autocorrelation plot of the batch mean stochastic process for the same run shown in Figure 1.1, which shows the batches are not significantly correlated, because all of the bars except the one for lag 0 are inside the dashed lines. In this case, a confidence interval for the unknown expectation (Equation 1.1) is easily done using the R function `t.test`:

```
> t.test(batch)
        One Sample t-test

data:  batch
t = -1.177, df = 19, p-value = 0.2537
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 -2.5184770  0.7054673
sample estimates:
 mean of x
-0.9065049
```

Here, `batch` is the vector of batch means which is plotted in Figure 1.4.

If this plot had shown the batches to be significantly correlated, then the method of batch means should not have been used because it would have a significant downward bias. However, the time series of batches can still be used, as explained in Section 1.10.2 below.

How does one choose the batch length? The method of batch means will work well only if the batch length b is large enough so that the infinite sum in Equation 1.9 is well approximated by the partial sum of the first b terms. Hence, when the method of batch means is used blindly with no knowledge of the ACF, b should be as large as possible. The only restriction on the length of batches is that the number of batches should be enough to

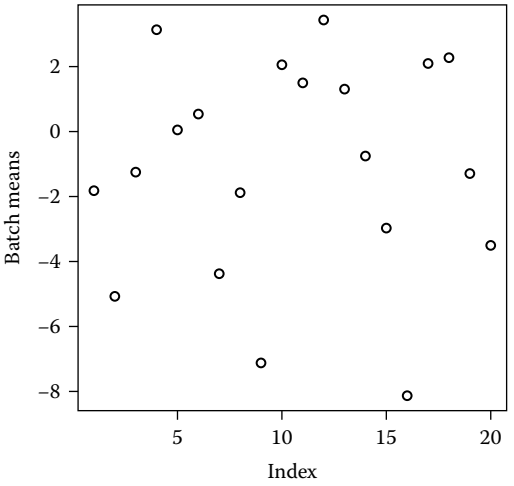

FIGURE 1.4
Batch mean plot for AR(1) example. Batch length 500.

get a reasonable estimate of variance. If one uses a t test, as shown above, then the t critical value corrects for the number of batches being small (Geyer, 1992; Schmeiser, 1982), but there is no point in the number of batches being so small that that the variance estimate is extremely unstable: 20–30 batches is a reasonable recommendation. One sometimes sees assumptions that the number of batches "goes to infinity" in theorems, but this is not necessary for simple MCSE calculation (Geyer, 1992, Section 3.2). If one is using estimated variance in a sequential stopping rule (Glynn and Whitt, 1991, 1992), then one does need the number of batches to go to infinity.

Meketon and Schmeiser (1984) pointed out that the batch means estimator of variance (Equation 1.17) is still valid if the batches are allowed to overlap, and a slight gain in efficiency is thereby achieved. For reasons explained in the following section, we do not

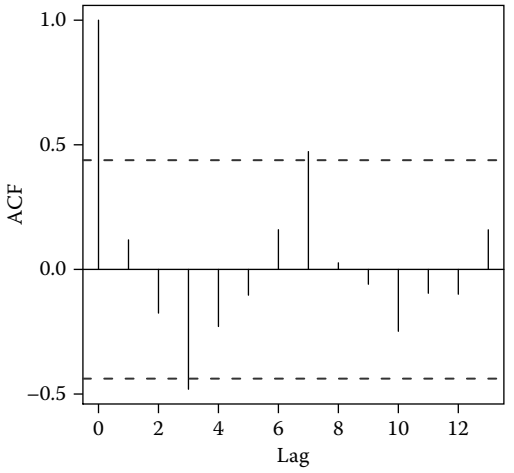

FIGURE 1.5
Autocorrelation plot of batch means for AR(1) example. Batch length 500.

recommend overlapping batch means, not because there is anything wrong with it, but because it does not fit together well with other methods we recommend.

1.10.2 Initial Sequence Methods

Another approach to variance estimation is to work directly with the representation (Equation 1.9) of the asymptotic variance. One cannot simply plug the empirical estimates (Equation 1.10) into Equation 1.9 because the variance of the high-lag terms does not decrease with lag, so as n goes to infinity an infinite amount of noise swamps the finite signal. Many solutions for this problem have been proposed in the time series literature (Geyer, 1992, Section 3.1 and references cited therein). But reversible Markov chains permit much simpler methods. Define

$$\Gamma_k = \gamma_{2k} + \gamma_{2k+1}. \tag{1.18}$$

Geyer (1992, Theorem 3.1) showed that the function $k \mapsto \Gamma_k$ is strictly positive, strictly decreasing, and strictly convex, and proposed three estimators of the asymptotic variance (Equation 1.9) that use these three properties, called the *initial positive sequence*, *initial monotone sequence*, and *initial convex sequence* estimators. Each is a consistent overestimate of the asymptotic variance (meaning the probability of underestimation by any fixed amount goes to zero as the Monte Carlo sample size goes to infinity) under no regularity conditions whatsoever (Geyer, 1992, Theorem 3.2). The initial convex sequence estimator is the best, because the smallest and still an asymptotic overestimate, but is a bit difficult to calculate. Fortunately, the R contributed package `mcmc` now has a function `initseq` that calculates all three estimators. We will only discuss the last. It forms

$$\widehat{\Gamma}_k = \hat{\gamma}_{2k} + \hat{\gamma}_{2k+1},$$

where $\hat{\gamma}_k$ is given by Equation 1.10, then finds the largest index m such that

$$\widehat{\Gamma}_k > 0, \quad k = 0, \ldots, m,$$

then defines $\widehat{\Gamma}_{m+1} = 0$, and then defines $k \mapsto \widetilde{\Gamma}_k$ to be the greatest convex minorant of $k \mapsto \widehat{\Gamma}_k$ over the range $0, \ldots, m+1$. Finally, it estimates

$$\hat{\sigma}^2_{\text{conv}} = -\hat{\gamma}_0 + 2 \sum_{k=0}^{m} \widetilde{\Gamma}_k. \tag{1.19}$$

Figure 1.6 shows a plot of the function $k \mapsto \widetilde{\Gamma}_k$ for the same run shown in Figure 1.1 compared to its theoretical value. When comparing this plot to Figure 1.3, remember that each index value in Figure 1.6 corresponds to two index values in Figure 1.3 because of the way Equation 1.18 is defined. Thus Figure 1.6 indicates significant autocorrelation out to about lag 300 (not 150).

The estimator of asymptotic variance (Equation 1.19) is calculated very simply in R:

```
> initseq(out)$var.con
[1] 7467.781
```

assuming the `mcmc` contributed package has already been loaded and `out` is the functional of the Markov chain for which the variance estimate is desired.

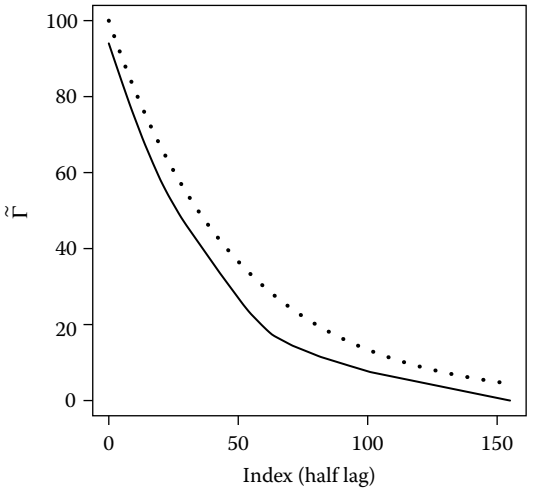

FIGURE 1.6
Plot of $\widetilde{\Gamma}$ for AR(1) example. Solid line: initial convex sequence estimator of Equation 1.18. Dotted line: theoretical value.

1.10.3 Initial Sequence Methods and Batch Means

When the original Markov chain is reversible, so is the chain of batches. Hence, initial sequence methods can be applied to a sequence of nonoverlapping batch means derived from a reversible Markov chain.

This means that the method of nonoverlapping batch means can be used without testing whether the batches are large enough. Simply process them with an initial sequence method, and the result is valid regardless of the batch length.

Here is how that works. Suppose we use a batch length of 50, which is too short.

```
> blen * var(batch)
[1] 2028.515
> blen * initseq(batch)$var.con
[1] 7575.506
```

The naive batch means estimator is terrible, less than a third of the size of the initial convex sequence estimator applied to the batch means (7575.506), but this is about the same as the initial convex sequence estimator applied to the original output (7467.781). So nothing is lost when only nonoverlapping batch means are output, regardless of the batch length used.

Partly for this reason, and partly because nonoverlapping batch means are useful for reducing the size of the output, whereas overlapping batch means are not, we do not recommend overlapping batch means and will henceforth always use the term *batch means* to mean nonoverlapping batch means.

1.11 The Practice of MCMC

The practice of MCMC is simple. Set up a Markov chain having the required invariant distribution, and run it on a computer. The folklore of simulation makes this seem more

complicated than it really is. None of this folklore is justified by theory and none of it actually helps users do good simulations, but, like other kinds of folklore, it persists despite its lack of validity.

1.11.1 Black Box MCMC

There is a great deal of theory about convergence of Markov chains. Unfortunately, none of it can be applied to get useful convergence information for most MCMC applications. Thus most users find themselves in the following situation we call *black box MCMC*:

1. You have a Markov chain having the required invariant distribution.
2. You know nothing other than that. The Markov chain is a "black box" that you cannot see inside. When run, it produces output. That is all you know. You know nothing about the transition probabilities of the Markov chain, nor anything else about its dynamics.
3. You know nothing about the invariant distribution except what you may learn from running the Markov chain.

Point 2 may seem extreme. You may know a lot about the particular Markov chain being used—for example, you may know that it is a Gibbs sampler—but if whatever you know is of no help in determining any convergence information about the Markov chain, then whatever knowledge you have is useless. Point 3 may seem extreme. Many examples in the MCMC literature use small problems that can be done by OMC or even by pencil and paper and for which a lot of information about the invariant distribution is available, but in complicated applications point 3 is often simply true.

1.11.2 Pseudo-Convergence

A Markov chain can appear to have converged to its equilibrium distribution when it has not. This happens when parts of the state space are poorly connected by the Markov chain dynamics: it takes many iterations to get from one part to another. When the time it takes to transition between these parts is much longer than the length of simulated Markov chain, then the Markov chain can appear to have converged but the distribution it appears to have converged to is the equilibrium distribution conditioned on the part in which the chain was started. We call this phenomenon *pseudo-convergence*.

This phenomenon has also been called "multimodality" since it may occur when the equilibrium distribution is multimodal. But multimodality does not cause pseudo-convergence when the troughs between modes are not severe. Nor does pseudo-convergence only happen when there is multimodality. Some of the most alarming cases of pseudo-convergence occur when the state space of the Markov chain is discrete and "modes" are not well defined (Geyer and Thompson, 1995). Hence pseudo-convergence is a better term.

1.11.3 One Long Run versus Many Short Runs

When you are in the black box situation, you have no idea how long runs need to be to get good mixing (convergence rather than pseudo-convergence). If you have a run that is already long enough, then an autocovariance plot like Figure 1.6 gives good information about mixing, and you know that you need to run a large multiple of the time it takes the

autocovariances to decay to nearly zero. But if all the runs you have done so far are nowhere near long enough, then they provide no information about how long is long enough.

The phenomenon of pseudo-convergence has led many people to the idea of comparing multiple runs of the sampler started at different points. If the multiple runs appear to converge to the same distribution, then—according to the multistart heuristic—all is well. But this assumes that you can arrange to have at least one starting point in each part of the state space to which the sampler can pseudo-converge. If you cannot do that—and in the black box situation you never can—then the multistart heuristic is worse than useless: it can give you confidence that all is well when in fact your results are completely erroneous.

Worse, addiction to many short runs can keep one from running the sampler long enough to detect pseudo-convergence or other problems, such as bugs in the code. People who have used MCMC in complicated problems can tell stories about samplers that appeared to be converging until, after weeks of running, they discovered a new part of the state space and the distribution changed radically. If those people had thought it necessary to make hundreds of runs, none of them could have been several weeks long.

Your humble author has a dictum that the least one can do is to make an overnight run. What better way for your computer to spend its time? In many problems that are not too complicated, this is millions or billions of iterations. If you do not make runs like that, you are simply not serious about MCMC. Your humble author has another dictum (only slightly facetious) that one should start a run when the paper is submitted and keep running until the referees' reports arrive. This cannot delay the paper, and may detect pseudo-convergence.

1.11.4 Burn-In

Burn in is used to start a good starting point for MCMC

Burn-in is a colloquial term that describes the practice of throwing away some iterations at the beginning of an MCMC run. This notion says that you start somewhere, say at x, then you run the Markov chain for n steps (the burn-in period) during which you throw away all the data (no output). After the burn-in you run normally, using each iterate in your MCMC calculations.

The name "burn-in" comes from electronics. Many electronics components fail quickly. Those that do not are a more reliable subset. So a burn-in is done at the factory to eliminate the worst ones.

Markov chains do not work the same way. Markov chain "failure" (nonconvergence or pseudo-convergence) is different from electronic component failure. Running longer may cure the first, but a dead transistor is dead forever. Thus "burn-in" is a bad term in MCMC, but there is more wrong than just the word, there is something fishy about the whole concept.

Figure 1.7 illustrates the issue that burn-in addresses. It shows an AR(1) time series with all parameters except starting position the same as Figure 1.1 so the equilibrium distribution, normal with mean zero and variance (Equation 1.13), is the same for both. In Figure 1.7 the starting position is far out in the tail of the equilibrium distribution, 10 standard deviations from the mean. In Figure 1.1 the starting position is the mean (zero). It takes several hundred iterations before the sample path in Figure 1.7 gets into the region containing the whole sample path in Figure 1.1.

The naive idea behind burn-in is that if we throw away several hundred iterations from Figure 1.7 it will be just as good as Figure 1.1. Overgeneralizing examples like Figure 1.7 leads to the idea that every MCMC run should have burn-in. Examples like Figure 1.1 show that this is not so. A Markov chain started anywhere near the center of the equilibrium distribution needs no burn-in.

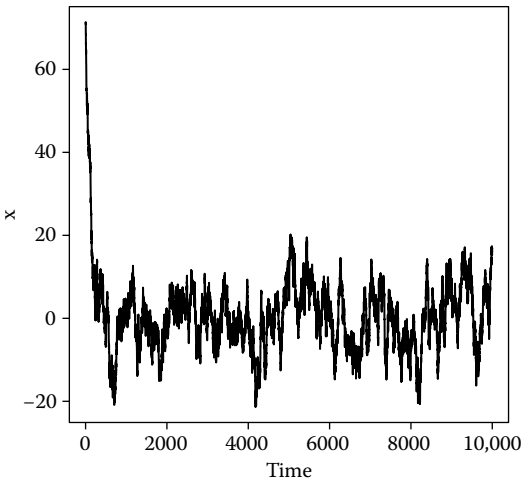

FIGURE 1.7
Time series plot for AR(1) example. Differs from Figure 1.1 only in the starting position.

> Burn-in is only one method, and not a particularly good method, of finding a good starting point.

There are several methods other than burn-in for finding a good starting point. One rule that is unarguable is

> Any point you don't mind having in a sample is a good starting point.

In a typical application, one has no theoretical analysis of the Markov chain dynamics that tells where the good starting points are (nor how much burn-in is required to get to a good starting point). All decisions about starting points are based on the output of some preliminary runs that appear to have "converged." Any point of the parts of these preliminary runs one believes to be representative of the equilibrium distribution is as good a starting point as any other.

So a good rule to follow is to start the next run where the last run ended. This is the rule most authorities recommend for random number generator seeds and the one used by R. It is also used by functions in the R package `mcmc` as discussed in Section 1.13 below.

Another method is to start at a mode of the equilibrium distribution (which can sometimes be found by optimization before doing MCMC) if it is known to be in a region of appreciable probability.

None of the examples in this chapter use burn-in. All use an alternative method of finding starting points. Burn-in is mostly harmless, which is perhaps why the practice persists. But everyone should understand that it is unnecessary, and those who do not use it are not thereby making an error.

Burn-in has a pernicious interaction with the multistart heuristic. If one believes in multistart, then one feels the need to start at many widely dispersed, and hence bad, starting points. Thus all of these short runs need be shortened some more by burn-in. Thus an erroneous belief in the virtues of multistart leads to an erroneous belief in the necessity of burn-in.

Another erroneous argument for burn-in is unbiasedness. If one could start with a realization from the equilibrium distribution, then the Markov chain would be stationary

and the Monte Carlo approximation (Equation 1.2) would be an unbiased estimator of what it estimates (Equation 1.1). Burn-in does not produce a realization from the equilibrium distribution, hence does not produce unbiasedness. At best it produces a small bias, but the alternative methods also do that. Moreover, the bias is of order n^{-1}, where n is the Monte Carlo sample size, whereas the MCSE is of order $n^{-1/2}$, so bias is negligible in sufficiently long runs.

1.11.5 Diagnostics

Many MCMC diagnostics have been proposed in the literature. Some work with one run of a Markov chain, but tell little that cannot be seen at a glance at a time series plot like Figure 1.1 or an autocorrelation plot like Figure 1.3. Others with multiple runs of a Markov chain started at different points, what we called the multistart heuristic above. Many of these come with theorems, but the theorems never prove the property you really want a diagnostic to have. These theorems say that if the chain converges, then the diagnostic will probably say that the chain converged, but they do not say that if the chain pseudo-converges, then the diagnostic will probably say that the chain did not converge. Theorems that claim to reliably diagnose pseudo-convergence have unverifiable conditions that make them useless. For example, as we said above, it is clear that a diagnostic based on the multistart heuristic will reliably diagnose pseudo-convergence if there is at least one starting point in each part of the state space to which the sampler can pseudo-converge, but in practical applications one has no way of arranging that.

There is only one perfect MCMC diagnostic: perfect sampling (Propp and Wilson, 1996; Kendall and Møller, 2000; see also Chapter 8, this volume). This is best understood as not a method of MCMC but rather a method of Markov-chain-assisted i.i.d. sampling. Since it is guaranteed to produce an i.i.d. sample from the equilibrium distribution of the Markov chain, a sufficiently large sample is guaranteed to not miss any parts of the state space having appreciable probability. Perfect sampling is not effective as a sampling scheme. If it works, then simply running the underlying Markov chain in MCMC mode will produce more accurate results in the same amount of computer time. Thus, paradoxically, perfect sampling is most useful when it fails to produce an i.i.d. sample of the requested size in the time one is willing to wait. This shows that the underlying Markov chain is useless for sampling, MCMC or perfect.

Perfect sampling does not work on black box MCMC (Section 1.11.1 above), because it requires complicated theoretical conditions on the Markov chain dynamics. No other diagnostic ever proposed works on black box MCMC, because if you know nothing about the Markov chain dynamics or equilibrium distribution except what you learn from output of the sampler, you can always be fooled by pseudo-convergence.

> There are known knowns. These are things we know that we know. There are known unknowns. That is to say, there are things that we now know we don't know. But there are also unknown unknowns. These are things we do not know we don't know.
>
> **Donald Rumsfeld**
> *US Secretary of Defense*

Diagnostics can find the known unknowns. They cannot find the unknown unknowns. They cannot find out what a black box MCMC sampler will do eventually. Only sufficiently long runs can do that.

1.12 Elementary Theory of MCMC

We say that a bit of computer code that makes a pseudorandom change to its state is an *update mechanism*. We are interested in update mechanisms that preserve a specified distribution, that is, if the state has the specified distribution before the update, then it has the same distribution after the update. From them we can construct Markov chains to sample that distribution.

We say that an update mechanism is *elementary* if it is not made up of parts that are themselves update mechanisms preserving the specified distribution.

1.12.1 The Metropolis–Hastings Update

Suppose that the specified distribution (the desired stationary distribution of the MCMC sampler we are constructing) has *unnormalized density* h. This means that h is a positive constant times a probability density. Thus h is a nonnegative-valued function that integrates (for continuous state) or sums (for discrete state) to a value that is finite and nonzero. The *Metropolis–Hastings update* does the following:

- When the current state is x, propose a move to y, having conditional probability density given x denoted $q(x, \cdot)$.
- Calculate the *Hastings ratio*

$$r(x, y) = \frac{h(y)q(y, x)}{h(x)q(x, y)}. \tag{1.20}$$

- Accept the proposed move y with probability

$$a(x, y) = \min(1, r(x, y)), \tag{1.21}$$

  that is, the state after the update is y with probability $a(x, y)$, and the state after the update is x with probability $1 - a(x, y)$.

The last step is often called *Metropolis rejection*. The name is supposed to remind one of "rejection sampling" in OMC, but this is a misleading analogy because in OMC rejection sampling is done repeatedly until some proposal is accepted (so it always produces a new value of the state). In contrast, one Metropolis–Hastings update makes one proposal y, which is the new state with probability $a(x, y)$, but otherwise the new state the same as the old state x. Any attempt to make Metropolis rejection like OMC rejection, destroys the property that this update preserves the distribution with density h.

The Hastings ratio (Equation 1.20) is undefined if $h(x) = 0$, thus we must always arrange that $h(x) > 0$ in the initial state. There is no problem if $h(y) = 0$. All that happens is that $r(x, y) = 0$ and the proposal y is accepted with probability zero. Thus the Metropolis–Hastings update can never move to a new state x having $h(x) = 0$. Note that the proposal y must satisfy $q(x, y) > 0$ with probability one because $q(x, \cdot)$ is the conditional density of y given x. Hence, still assuming $h(x) > 0$, the denominator of the Hastings ratio is nonzero with probability one, and the Hastings ratio is well defined. Note that either term of the numerator of the Hastings ratio can be zero, so the proposal is almost surely rejected if

either $h(y) = 0$ or $q(y, x) = 0$, that is, if y is an impossible value of the desired equilibrium distribution or if x is an impossible proposal when y is the current state.

We stress that nothing bad happens if the proposal y is an impossible value of the desired equilibrium distribution. The Metropolis–Hastings update automatically does the right thing, almost surely rejecting such proposals. Hence, it is not necessary to arrange that proposals are always possible values of the desired equilibrium distribution; it is only necessary to assure that one's implementation of the unnormalized density function h works when given any possible proposal as an argument and gives $h(y) = 0$ when y is impossible.

If `unifrand` is a function with no arguments that produces one $U(0, 1)$ random variate and the Hastings ratio has already been calculated and stored in a variable `r`, then the following computer code does the Metropolis rejection step:

```
if (unifrand() < r) {
    x = y
}
```

The variable `x`, which is considered the state of the Markov chain, is set to `y` (the proposal) when a uniform random variate is less than the Hastings ratio `r` and left alone otherwise.

The following computer code works with the log Hastings ratio `logr` to avoid overflow:

```
if (logr >= 0 || unifrand() < exp(logr)) {
    x = y
}
```

It uses the "short circuit" property of the `||` operator in the R or C language. Its second operand `unifrand() < exp(logr)` is only evaluated when its first operand `logr >= 0` evaluates to `FALSE`. Thus `exp(logr)` can never overflow.

1.12.2 The Metropolis–Hastings Theorem

We now prove that the Metropolis–Hastings update is reversible with respect to h, meaning that the transition probability that describes the update is reversible with respect to the distribution having unnormalized density h.

If X_n is the current state and Y_n is the proposal, we have $X_n = X_{n+1}$ whenever the proposal is rejected. Clearly, the distribution of (X_n, X_{n+1}) given rejection is exchangeable.

Hence, it only remains to be shown that (X_n, Y_n) is exchangeable given acceptance. We need to show that

$$\mathrm{E}\{f(X_n, Y_n)a(X_n, Y_n)\} = \mathrm{E}\{f(Y_n, X_n)a(X_n, Y_n)\}$$

for any function f that has expectation (assuming X_n has desired stationary distribution). That is, we must show we can interchange arguments of f in

$$\iint f(x, y)h(x)a(x, y)q(x, y)\,dx\,dy \tag{1.22}$$

(with integrals replaced by sums if the state is discrete), and that follows if we can interchange x and y in

$$h(x)a(x, y)q(x, y) \tag{1.23}$$

because we can exchange x and y in Equation 1.22, x and y being dummy variables. Clearly only the set of x and y such that $h(x) > 0$, $q(x, y) > 0$, and $a(x, y) > 0$ contributes to the integral or (in the discrete case) sum (Equation 1.22), and these inequalities further imply that $h(y) > 0$ and $q(y, x) > 0$. Thus we may assume these inequalities, in which case we have

$$r(y, x) = \frac{1}{r(x, y)}$$

for all such x and y.

Suppose that $r(x, y) \le 1$, so $r(x, y) = a(x, y)$ and $a(y, x) = 1$. Then

$$\begin{aligned} h(x)a(x, y)q(x, y) &= h(x)r(x, y)q(x, y) \\ &= h(y)q(y, x) \\ &= h(y)q(y, x)a(y, x). \end{aligned}$$

Conversely, suppose that $r(x, y) > 1$, so $a(x, y) = 1$ and $a(y, x) = r(y, x)$. Then

$$\begin{aligned} h(x)a(x, y)q(x, y) &= h(x)q(x, y) \\ &= h(y)r(y, x)q(y, x) \\ &= h(y)a(y, x)q(y, x). \end{aligned}$$

In either case we can exchange x and y in Equation 1.23, and the proof is done.

1.12.3 The Metropolis Update

The special case of the Metropolis–Hastings update when $q(x, y) = q(y, x)$ for all x and y is called the *Metropolis update*. Then the Hastings ratio (Equation 1.20) simplifies to

$$r(x, y) = \frac{h(y)}{h(x)} \tag{1.24}$$

and is called the Metropolis ratio or the odds ratio. Thus Metropolis updates save a little time in calculating $r(x, y)$ but otherwise have no advantages over Metropolis–Hastings updates.

One obvious way to arrange the symmetry property is to make proposals of the form $y = x + e$, where e is stochastically independent of x and symmetrically distributed about zero. Then $q(x, y) = f(y - x)$, where f is the density of e. Widely used proposals of this type have e normally distributed with mean zero or e uniformly distributed on a ball or a hypercube centered at zero (see Section 1.12.10 below for more on such updates).

1.12.4 The Gibbs Update

In a *Gibbs update* the proposal is from a conditional distribution of the desired equilibrium distribution. It is always accepted.

The proof of the theorem that this update is reversible with respect to the desired equilibrium distribution is trivial. Suppose that X_n has the desired stationary distribution. Suppose that the conditional distribution of X_{n+1} given $f(X_n)$ is same as the conditional distribution of X_n given $f(X_n)$. Then the pair (X_n, X_{n+1}) is conditionally exchangeable given $f(X_n)$, hence unconditionally exchangeable.

In common parlance, a Gibbs update uses the conditional distribution of one component of the state vector given the rest of the components, that is, the special case of the update described above where $f(X_n)$ is X_n with one component omitted. Conditional distributions of this form are called "full conditionals." There is no reason other than tradition why such conditional distributions should be preferred.

In fact other conditionals have been considered in the literature. If $f(X_n)$ is X_n with several components omitted, this is called "block Gibbs." Again, there is no reason other than tradition why such conditional distributions should be preferred.

If one insists that Gibbs update only apply to full conditionals, then one could call the updates described here "generalized Gibbs." But the "generalized" here is not much of a generalization. Simply do a change of variable so that $f(X_n)$ is a group of components of the new state vector and "generalized Gibbs" is "block Gibbs." Also the argument for all these updates is exactly the same.

Gibbs updates have one curious property not shared by other Metropolis–Hastings updates: they are *idempotent*, meaning the effect of multiple updates is the same as the effect of just one. This is because the update never changes $f(X_n)$, hence the result of many repetitions of the same Gibbs update results in X_{n+1} having the conditional distribution given $f(X_n)$ just like the result of a single update. In order for Gibbs elementary updates to be useful, they must be combined somehow with other updates.

1.12.5 Variable-at-a-Time Metropolis–Hastings

Gibbs updates alter only part of the state vector; when using "full conditionals" the part is a single component. Metropolis–Hastings updates can be modified to do the same.

Divide the state vector into two parts, $x = (u, v)$. Let the proposal alter u but not v. Hence, the proposal density has the form $q(x, u)$ instead of the $q(x, y)$ we had in Section 1.12.1. Again let $h(x) = h(u, v)$ be the unnormalized density of the desired equilibrium distribution. The variable-at-a-time Metropolis–Hastings update does the following:

- When the current state is $x = (u, v)$, propose a move to $y = (u^*, v)$, where u^* has conditional probability density given x denoted $q(x, \cdot) = q(u, v, \cdot)$.
- Calculate the *Hastings ratio*

$$r(x, y) = \frac{h(u^*, v)q(u^*, v, u)}{h(u, v)q(u, v, u^*)}.$$

- Accept the proposed move y with probability (Equation 1.21), that is, the state after the update is y with probability $a(x, y)$, and the state after the update is x with probability $1 - a(x, y)$.

We shall not give a proof of the validity of variable-at-a-time Metropolis–Hastings, which would look very similar to the proof in Section 1.12.2.

The term "variable-at-a-time Metropolis–Hastings" is something of a misnomer. The sampler run in Metropolis et al. (1953) was a "variable-at-a-time" sampler. For historical accuracy, the name "Metropolis algorithm" should include the updates described in Section 1.12.1 and in this section. Current usage, however, seems otherwise, naming the samplers as we have done here.

1.12.6 Gibbs Is a Special Case of Metropolis–Hastings

To see that Gibbs is a special case of Metropolis–Hastings, do a change of variable so that the new state vector can be split $x = (u, v)$ as we did in the preceding section, and v is the part of the state on which the Gibbs update conditions. Thus we are doing block Gibbs updating u from its conditional distribution given v. Factor the unnormalized density $h(u, v) = g(v)q(v, u)$, where $g(v)$ is an unnormalized marginal of v and $q(v, u)$ is the (properly normalized) conditional of u given v. Now do a Metropolis–Hastings update with q as the proposal distribution. The proposal is $y = (u^*, v)$, where u^* has the distribution $q(v, \cdot)$. The Hastings ratio is

$$r(x, y) = \frac{h(u^*, v)q(u, v)}{h(u, v)q(v, u^*)} = \frac{g(v)q(v, u^*)q(u, v)}{g(v)q(v, u)q(v, u^*)} = 1.$$

Hence the proposal is always accepted.

1.12.7 Combining Updates

1.12.7.1 Composition

Let $P_1, \ldots, P_k$ be update mechanisms (computer code) and let $P_1 P_2 \cdots P_k$ denote the composite update that consists of these updates done in that order with P_1 first and P_k last. If each P_i preserves a distribution, then obviously so does $P_1 P_2 \ldots P_k$.

If $P_1, \ldots, P_k$ are the Gibbs updates for the "full conditionals" of the desired equilibrium distribution, then the composition update is often called a *fixed scan Gibbs sampler*.

As a simple example, suppose that the desired equilibrium distribution is exchangeable and multivariate normal. Then the conditional distribution of one component of the state vector given the rest is univariate normal with mean that is a symmetric linear function of the rest of the components and constant variance. In the special case where there are just two components, the fixed scan Gibbs sampler is just consecutive pairs of an AR(1) process (Section 1.9 above).

1.12.7.2 Palindromic Composition

Note that $P_1 P_2 \ldots P_k$ is not reversible with respect to the distribution it preserves unless the transition probabilities associated with $P_1 P_2 \ldots P_k$ and $P_k P_{k-1} \ldots P_1$ are the same.

The most obvious way to arrange reversibility is to make $P_i = P_{k-i}$, for $i = 1, \ldots, k$. Then we call this composite update *palindromic*. Palindromic compositions are reversible, nonpalindromic ones need not be.

1.12.8 State-Independent Mixing

Let P_y be update mechanisms (computer code) and let $\mathrm{E}(P_Y)$ denote the update that consists of doing a random one of these updates: generate Y from some distribution and do P_Y.

If Y is independent of the current state and each P_y preserves the same distribution, then so does $\mathrm{E}(P_Y)$. If X_n has the desired equilibrium distribution, then it also has this distribution conditional on Y, and X_{n+1} also has this distribution conditional on Y. Since the conditional distribution of X_{n+1} does not depend on Y, these variables are independent, and X_{n+1} has the desired equilibrium distribution unconditionally.

Furthermore, the Markov chain with update $\mathrm{E}(P_Y)$ is reversible if each P_y is reversible.

"Mixture" is used here in the sense of mixture models. The update $\mathrm{E}(P_Y)$ is the mixture of updates P_y.

The most widely used mixtures use a finite set of y values. For example, one popular way to combine the "full conditional" Gibbs updates, one for each component of the state vector, is by state-independent mixing using the uniform distribution on the set of full conditionals as the mixing distribution. This is often called a *random scan Gibbs sampler*. The choice of the uniform distribution is arbitrary. It has no optimality properties. It does, however, make a simple default choice.

Mixing and composition can be combined. Suppose we have elementary update mechanisms $P_1, \ldots, P_k$, and let $\mathcal{Y}$ be a set of functions from $\{1, \ldots, m\}$ to $\{1, \ldots, k\}$. For $y \in \mathcal{Y}$, let Q_y denote the composition $P_{y(1)} P_{y(2)} \ldots P_{y(m)}$. Now consider the update $\mathrm{E}(Q_Y)$, where Y is a random element of $\mathcal{Y}$ independent of the state of the Markov chain.

If $m = k$ and the P_i are the "full conditional" Gibbs updates and Y has the uniform distribution on $\mathcal{Y}$, which consists of all permutations of $1, \ldots, k$, then this mixture of compositions sampler is often called a *random sequence scan Gibbs sampler*.

We are not fond of this "scan" terminology, because it is too limiting. It focuses attention on a very few special cases of combination by composition and mixing, special cases that have no optimality properties and no reason other than tradition for their prominence.

State-independent mixing with the mixing distribution having an infinite sample space has also been used. Bélisle et al. (1993) and Chen and Schmeiser (1993) investigate the "hit and run algorithm" which uses elementary updates P_y where the state space of the Markov chain is Euclidean and y is a direction in the state space. Do a change of coordinates so that y is a coordinate direction, and do a Gibbs or other variable-at-a-time Metropolis–Hastings update of the coordinate in the y direction. The mixture update $\mathrm{E}(P_Y)$ is called a "hit and run sampler" when Y has the uniform distribution on directions.

Again there is no particular reason to use a "hit and run" sampler. It is merely one of an infinite variety of samplers using composition and state-independent mixing.

State-dependent mixing is possible, but the argument is very different (Section 1.17.1 below).

1.12.9 Subsampling

Another topic that is not usually discussed in terms of composition and mixing, although it is another special case of them, is subsampling of Markov chains.

If P is an update mechanism, we write P^k to denote the k-fold composition of P with itself. If $X_1, X_2, \ldots$ is a Markov chain with update mechanism P, then $X_1, X_{k+1}, X_{2k+1}, \ldots$ is a Markov chain with update mechanism P^k.

The process that takes every kth element of a Markov chain $X_1, X_2, \ldots$ forming a new Markov chain $X_1, X_{k+1}, X_{2k+1}, \ldots$ is called *subsampling* the original Markov chain at *spacing* k. As we just said, the result is another Markov chain. Hence, a subsampled Markov chain is just like any other Markov chain.

According to Elizabeth Thompson, "You don't get a better answer by throwing away data." This was proved as a theorem about Markov chains by Geyer (1992) for reversible Markov chains and by MacEachern and Berliner (1994) for nonreversible Markov chains. Subsampling cannot improve the accuracy of MCMC approximation; it must make things worse.

The original motivation for subsampling appears to have been to reduce autocorrelation in the subsampled chain to a negligible level. Before 1994 the Markov chain CLT was not well understood by statisticians, so appeal was made to a non-theorem: the central limit

almost-but-not-quite theorem for almost-but-not-quite i.i.d. data. Now that the Markov chain CLT is well understood, this cannot be a justification for subsampling.

Subsampling may appear to be necessary just to reduce the amount of output of a Markov chain sampler to manageable levels. Billions of iterations may be needed for convergence, but billions of iterations of output may be too much to handle, especially when using R, which chokes on very large objects. But nonoverlapping batch means (Section 1.10.1) can reduce the size of the output with no loss of accuracy of estimation. Moreover, one does not need to know the batch length necessary to make the empirical variance of the batch means a good estimate of the asymptotic variance in the Markov chain CLT in order to use batches to reduce the size of output. The method of Section 1.10.3 allows one to use batches that are too short and still obtain accurate estimates of the asymptotic variance in the Markov chain CLT. Hence, if the objective is to reduce the size of output, batching is better than subsampling.

Hence, the only reason to use subsampling is to reduce the size of output when one cannot use batching. Good MCMC code, for example the functions `metrop` and `temper` in the R contributed package `mcmc` (Geyer, 2010b), allow an arbitrary function g supplied by the user as an R function to be used in calculation of the batch means in Equation 1.16. Other MCMC code that does not allow this may not output batch means for required functionals of the Markov chain. In this case the only way to reduce the size of output and still calculate the required functionals is subsampling. Another case where one cannot use the batch means is when the required functionals are not known when the sampling is done. This occurs, for example, in Monte Carlo likelihood approximation (Geyer and Thompson, 1992).

Geyer (1992) gave another justification of subsampling based on the cost of calculating the function g in a functional (Section 1.6 above). If the cost in computing time of calculating $g(X_i)$ is much more than the cost of sampling (producing X_i given X_{i-1}), then subsampling may be justified. This is rarely the case, but it does happen.

1.12.10 Gibbs and Metropolis Revisited

Our terminology of "elementary updates" combined by "composition" or "mixing" or both is not widespread. The usual terminology for a much more limited class of samplers is the following:

- A *Gibbs sampler* is an MCMC sampler in which all of the elementary updates are Gibbs, combined either by composition (fixed scan), by mixing (random scan), or both (random sequence scan), the "scan" terminology being explained in Section 1.12.8 above.
- A *Metropolis algorithm* is an MCMC sampler in which all of the elementary updates are Metropolis, combined either by composition, mixing, or both (and the same "scan" terminology is used).
- A *Metropolis–Hastings algorithm* is an MCMC sampler in which all of the elementary updates are Metropolis–Hastings, combined either by composition, mixing, or both (and the same "scan" terminology is used).
- A *Metropolis-within-Gibbs sampler* is the same as the preceding item. This name makes no sense at all since Gibbs is a special case of Metropolis–Hastings (Section 1.12.6 above), but it is widely used.

- An *independence Metropolis–Hastings algorithm* (named by Tierney, 1994) is a special case of the Metropolis–Hastings algorithm in which the proposal distribution does not depend on the current state: $q(x, \cdot)$ does not depend on x.
- A *random-walk Metropolis–Hastings algorithm* (named by Tierney, 1994) is a special case of the Metropolis–Hastings algorithm in which the proposal has the form $x + e$, where e is stochastically independent of the current state x, so $q(x, y)$ has the form $f(y - x)$.

The Gibbs sampler became very popular after the paper of Gelfand and Smith (1990) appeared. The term MCMC had not been coined (Geyer, 1992). It was not long, however, before the limitations of the Gibbs sampler were recognized. Peter Clifford (1993), discussing Smith and Roberts (1993), Besag and Green (1993), and Gilks et al. (1993), said:

> Currently, there are many statisticians trying to reverse out of this historical *cul-de-sac*. To use the Gibbs sampler, we have to be good at manipulating conditional distributions . . . this rather brings back the mystique of the statisticians.

The American translation of "reverse out of this *cul-de-sac*" is "back out of this blind alley." Despite this, many naive users still have a preference for Gibbs updates that is entirely unwarranted. If I had a nickel for every time someone had asked for help with slowly converging MCMC and the answer had been to stop using Gibbs, I would be rich. Use Gibbs updates only if the resulting sampler works well. If not, use something else.

One reason sometimes given for the use of Gibbs updates is that they are "automatic." If one chooses to use a Gibbs sampler, no other choices need be made, whereas if one uses the Metropolis–Hastings algorithm, one must choose the proposal distribution, and even if one's choice of Metropolis–Hastings algorithm is more restricted, say to normal random-walk Metropolis–Hastings, there is still the choice of the variance matrix of the normal proposal distribution. This "automaticity" of the Gibbs sampler is illusory, because even if one only knows about "scans" one still must choose between fixed and random scan. Moreover, one should consider "block Gibbs" or even the more general Gibbs updates described in Section 1.12.4 above.

Nevertheless, Gibbs does seem more automatic than Metropolis–Hastings to many users. The question is whether this lack of options is a good thing or a bad thing. It is good if it works well and bad otherwise.

1.13 A Metropolis Example

We now turn to a realistic example of MCMC, taken from the package vignette of the `mcmc` contributed R package (Geyer, 2010b). The function `metrop` in this package runs a normal random-walk Metropolis sampler in the terminology of Section 1.12.10 having equilibrium distribution for a continuous random vector specified by a user-written R function that calculates its log unnormalized density. A major design goal of this package is that there be very little opportunity for user mistakes to make the simulation incorrect. For the `metrop` function, if the user codes the log unnormalized density function correctly, then the function will run a Markov chain having the correct stationary distribution (specified by this user-written function). There is nothing other than incorrectly writing the log unnormalized

density function that the user can do to make the Markov chain have the wrong stationary distribution.

It may seem that this is a very weak correctness property. There is no guarantee that the Markov chain mixes rapidly and so produces useful results in a reasonable amount of time. But nothing currently known can guarantee that for arbitrary problems. Methods of proving rapid mixing, although they are applicable in principle to arbitrary problems, are so difficult that they have actually been applied only to a few simple examples. Moreover, they are entirely pencil-and-paper proofs. There is nothing the computer can do to assure rapid mixing of Markov chains for arbitrary user-specified equilibrium distributions. Thus this weak correctness property (having the correct equilibrium distribution) is the most one can expect a computer program to assure.

Thus this "weak" correctness property is the strongest property one can reasonably assert for an MCMC program. All MCMC programs should guarantee it, but how many do? The functions in the `mcmc` package have been exhaustively tested using the methodology explained in Section 1.16 below and further described in the package vignette `debug.pdf` that comes with every installation of the package. All of the tests are in the `tests` directory of the source code of the package, which is available from CRAN (http://www.cran.r-project.org/).

In addition to an R function that specifies the log unnormalized density of the equilibrium distribution, the user may also provide an R function that specifies an arbitrary functional of the Markov chain to be output. If the Markov chain is $X_1, X_2, \ldots$ and this user-supplied R function codes the mathematical function g, then $g(X_1), g(X_2), \ldots$ is output. Alternatively, batch means of $g(X_1), g(X_2), \ldots$ are output.

Finally, the user must specify the variance matrix of the multivariate normal distribution used in the "random-walk" proposal. There is nothing else the user can do to affect the Markov chain simulated by the `metrop` function.

Let us see how it works. We use the example from the package vignette `demo.pdf` that comes with every installation of the package. This is a Bayesian logistic regression problem that uses the data set `logit` in the package. There are five variables in this data frame, the response `y` and four quantitative predictor variables `x1`, `x2`, `x3`, and `x4`.

A frequentist analysis of these data is done by the following R statements:

```
library(mcmc)
data(logit)
out <- glm(y ~ x1 + x2 + x3 + x4, data = logit,
    family = binomial(), x = TRUE)
summary(out)
```

We wish to do a Bayesian analysis where the prior distribution for the five regression coefficients (one for each predictor and an intercept) makes them i.i.d. normal with mean 0 and standard deviation 2.

The log unnormalized posterior (log likelihood plus log prior) density for this model is calculated by the R function `lupost` defined as follows:

```
x <- out$x
y <- out$y

lupost <- function(beta, x, y, ...) {
    eta <- as.numeric(x %*% beta)
```

```
    logp <- ifelse(eta < 0, eta - log1p(exp(eta)),- log1p
     (exp(- eta)))
    logq <- ifelse(eta < 0, - log1p(exp(eta)), - eta - log1p
     (exp(- eta)))
    logl <- sum(logp[y == 1]) + sum(logq[y == 0])
    return(logl - sum(beta^2) / 8)
}
```

This assumes that `out` is the result of the call to `glm` shown above, so `y` is the response vector and `x` is the model matrix for this logistic regression.

The tricky calculation of the log likelihood avoids overflow and catastrophic cancelation in calculation of $\log(p)$ and $\log(q)$, where

$$p = \frac{\exp(\eta)}{1 + \exp(\eta)} = \frac{1}{1 + \exp(-\eta)},$$
$$q = \frac{1}{1 + \exp(\eta)} = \frac{\exp(-\eta)}{1 + \exp(-\eta)},$$

so taking logs gives

$$\log(p) = \eta - \log(1 + \exp(\eta)) = -\log(1 + \exp(-\eta)),$$
$$\log(q) = -\log(1 + \exp(\eta)) = -\eta - \log(1 + \exp(-\eta)).$$

To avoid overflow, we always chose the case where the argument of exp is negative. We have also avoided catastrophic cancelation when $|\eta|$ is large. If η is large and positive, then

$$p \approx 1,$$
$$q \approx 0,$$
$$\log(p) \approx -\exp(-\eta),$$
$$\log(q) \approx -\eta - \exp(-\eta),$$

and our use of the R function `log1p`, which calculates the function $x \mapsto \log(1 + x)$ correctly for small x, avoids problems with calculating $\log(1 + \exp(-\eta))$ here. The case where η is large and negative is similar. The above definitions having been made, the following statements do an MCMC run:

```
beta.init <- as.numeric(coefficients(out))
out <- metrop(lupost, beta.init, 1e3, x = x, y = y)
```

where `beta.init` is the initial state of the Markov chain (it would be more natural to a Bayesian to use the posterior mode rather than the maximum likelihood estimate, but the starting position makes no difference so long as it is not too far out in the tails of the equilibrium distribution) and where `1e3` is the MCMC sample size. The default batch length is one, so there is no batching here. The component `out$accept` of the result gives the acceptance rate (the fraction of Metropolis updates in which the proposal is accepted) and the component `out$batch` gives the output of the Markov chain as an $n \times p$ matrix, where n is the number of iterations here where there is no batching (although in general

n is the number of batches) and where p is the dimension of the state space here where no functional of the Markov chain is specified and the default is the identity functional (although in general p is the dimension of the result of the user-supplied output function).

The functions in the `mcmc` package are designed so that if given the output of a preceding run as their first argument, they continue the run of the Markov chain where the other run left off. For example, if we were to say

```
out2 <- metrop(out, x = x, y = y)
```

here, then `rbind(out$batch, out2$batch)` would be a run of the Markov chain. The second invocation of the `metrop` function starts with the seed of R's random number generator (RNG) and the state of the Markov chain set to what they were when the first invocation finished. Thus there is no difference between `rbind(out$batch, out2$batch)` and the result of one invocation starting at the same RNG seed and initial state and running for twice as many iterations as the two shown here did.

This "restart" property obviates any need for burn-in. If the first run "converged" in the sense that any part of the run was in a high-probability part of the state space, then the second run starts in a good place and needs no burn-in. Since the first run started at the maximum likelihood estimate, which is in a high-probability part of the state space, the first run needed no burn-in either.

Using this function is not quite this simple. We need to adjust the normal proposal to achieve a reasonable acceptance rate. It is generally accepted (Gelman et al., 1996) that an acceptance rate of about 20% is right, although this recommendation is based on the asymptotic analysis of a toy problem (simulating a multivariate normal distribution) for which one would never use MCMC and is very unrepresentative of difficult MCMC applications. Geyer and Thompson (1995) came to a similar conclusion, that a 20% acceptance rate is about right, in a very different situation. But they also warned that a 20% acceptance rate could be very wrong, and produced an example where a 20% acceptance rate was impossible and attempting to reduce the acceptance rate below 70% would keep the sampler from ever visiting part of the state space. So the 20% magic number must be considered like other rules of thumb we teach in introductory courses (such as $n > 30$ means the normal approximation is valid). We know these rules of thumb can fail. There are examples in the literature where they do fail. We keep repeating them because we want something simple to tell beginners, and they are all right for some problems.

The `scale` argument to the `metrop` function specifies the variance matrix for the proposal. The default is the identity matrix. This results in too low an acceptance rate in this problem (0.008). A little bit of trial and error (shown in the vignette) shows that

```
out <- metrop(out, scale = 0.4, x = x, y = y)
```

gives about 20% acceptance rate, so this scaling, which specifies proposal variance matrix 0.4 times the identity matrix, is what we use. More complicated specification of the proposal variance is possible; see the help for the `metrop` function for details.

Now we do a longer run

```
out <- metrop(out, nbatch = 1e4, x = x, y = y)
```

and look at time series plots and autocorrelation plots (shown in the vignette), which show that the Markov chain seems to mix well and that autocorrelations are negligible after

lag 25. We use batch length 100 to be safe. We are interested here in calculating both posterior means and posterior variances. Variances are not functionals of the Markov chain, but squares are, and we can use the identity $\mathrm{var}(Z) = \mathrm{E}(Z^2) - \mathrm{E}(Z)^2$ to calculate variances from means and means of squares. Thus we run the following:

```
out <- metrop(out, nbatch = 1e2, blen = 100,
    outfun = function(z, ...) c(z, z^2), x = x, y = y)
```

Here the user-specified output function (argument `outfun` of the `metrop` function) maps the state `z`, a vector of length 5, to `c(z, z^2)`, a vector of length 10. So now `out$batch` is a 100×10 matrix, 100 being the number of batches (argument `nbatch`) and 10 being the length of the result of `outfun`).

Now

```
foo <- apply(out$batch, 2, mean)
foo.mcse <- apply(out$batch, 2, sd) / sqrt(out$nbatch)
```

are estimates of the posterior means of the components of the vector returned by `outfun` (the regression coefficients and their squares) and the MCSE of these estimates, respectively. The first five components are useful directly:

```
mu <- foo[1:5]
mu.mcse <- foo.mcse[1:5]
```

These are estimates of the posterior means of the regression coefficients and their MCSE (see the vignette for actual numbers).

Monte Carlo estimates of the posterior variances are found using $\mathrm{var}(Z) = \mathrm{E}(Z^2) - \mathrm{E}(Z)^2$,

```
sigmasq <- foo[6:10] - foo[1:5]^2
```

but to calculate the MCSE we need the delta method. Let u_i denote the sequence of batch means for one parameter and $\bar{u}$ the grand mean of this sequence (the estimate of the posterior mean of that parameter), let v_i denote the sequence of batch means for the squares of the same parameter and $\bar{v}$ the grand mean of that sequence (the estimate of the posterior second absolute moment of that parameter), and let $\mu = \mathrm{E}(\bar{u})$ and $\nu = \mathrm{E}(\bar{v})$. Then the delta method linearizes the nonlinear function

$$g(\mu, \nu) = \nu - \mu^2$$

as

$$\Delta g(\mu, \nu) = \Delta\nu - 2\mu\,\Delta\mu,$$

saying that

$$g(\bar{u}, \bar{v}) - g(\mu, \nu)$$

has the same asymptotic normal distribution as

$$(\bar{v} - \nu) - 2\mu(\bar{u} - \mu)$$

which, of course, has variance `1 / out$nbatch` times that of

$$(v_i - \nu) - 2\mu(u_i - \mu),$$

and this variance is estimated by

$$\frac{1}{n_{\text{batch}}} \sum_{i=1}^{n_{\text{batch}}} \bigl[(v_i - \bar{v}) - 2\bar{u}(u_i - \bar{u})\bigr]^2.$$

So

```
u <- out$batch[ , 1:5]
v <- out$batch[ , 6:10]
ubar <- apply(u, 2, mean)
vbar <- apply(v, 2, mean)
deltau <- sweep(u, 2, ubar)
deltav <- sweep(v, 2, vbar)
foo <- sweep(deltau, 2, ubar, "*")
sigmasq.mcse <- sqrt(apply((deltav - 2 * foo)^2,
2, mean) / out$nbatch)
```

does the MCSE for the posterior variance (see the vignette for actual numbers).

Another application of the delta method gives MCSE for posterior standard deviations (see the vignette for details).

1.14 Checkpointing

The "restart" property of the `metrop` and `temper` functions is also useful for checkpointing. If one wants to do very long runs, they need not be done with one function invocation. Suppose that `out` is the result of an invocation of `metrop` and that the log unnormalized density function and output function (if present) do not take additional arguments, getting any additional data from the R global environment, and suppose that any such additional data has been set up. Let `ncheck` be the number of repetitions of `out` we want to make. Then

```
for (icheck in 1:ncheck) {
    out <- metrop(out)
    save(out, file = sprintf("check%03d.rda", icheck))
}
```

does them and saves them on disk, unless the computer crashes for some reason. After a crash, only the work not done and saved is left to do. Set up any required global variables and `ncheck` as before, and restart with

```
files <- system("ls check*.rda", intern = TRUE)
kcheck <- length(files)
```

```
load(file = files[kcheck])
if (kcheck < ncheck) {
    for (icheck in (kcheck + 1):ncheck) {
        out <- metrop(out)
        save(out, file = sprintf("check%03d.rda", icheck))
    }
}
```

(this is for UNIX, e.g., Linux or MAC OS X, and would have to be modified for Microsoft Windows). When finished collect the results with

```
files <- system("ls check*.rda", intern = TRUE)
ncheck <- length(files)
batch <- NULL
for (icheck in 1:ncheck) {
    load(file = files[icheck])
    batch <- rbind(batch, out$batch, deparse.level = 0)
}
```

and batch is the same as `out$batch` from one long run. This idiom allows very long runs even with unreliable computers.

1.15 Designing MCMC Code

Nothing is easier than designing MCMC algorithms. Hundreds have been introduced into the literature under various names. All that are useful in non-toy problems are special cases of the Metropolis–Hastings–Green algorithm.

When one invents a new sampler, how does one argue that it is correct? One proves a theorem: the new sampler is a special case of the MHG algorithm. The proof is usually not difficult but does require tight reasoning, like all proofs. One common error is sloppiness about what is the state of the Markov chain. Many have made the mistake of having proposals depend on some variables in the computer program that are not considered part of the state in calculating the Hastings ratio, that is, the state space is considered one thing in one part of the argument and another thing in another part—a clear error if one thinks about it.

One does not have to call this theorem a theorem, but one does need the care in proving it that any theorem requires. A few hours of careful thought about what is and what is not a special case of the MHG algorithm can save weeks or months of wasted work on a mistake. This notion that you have to prove a theorem every time you invent an MCMC algorithm came to your humble author from the experience of humbling mistakes committed by himself and others. If you think you have to prove a theorem, you will (hopefully) exercise appropriately careful argument. If you think you can use your intuition, many sad stories could be told about failure of intuition. The MHG algorithm is not difficult but is also not very intuitive.

Before one can prove a theorem, one must state the theorem, and here too care is required. The theorem must state precisely how one's MCMC algorithm works, with no vagueness.

This is very important. One cannot correctly implement an MCMC algorithm in computer code when one has to guess what the algorithm actually is. Most erroneous MCMC algorithms (just like most erroneous attempts at theorems) result from vagueness.

These general remarks having been made, we now turn to some desirable features of MCMC code that few computer packages have but the `mcmc` package has shown to be very useful.

The first is the "restart" property discussed in Sections 1.13 and 1.14 above and possessed by both the `metrop` and `temper` functions. This is the property that the R object output by a function doing MCMC (or the equivalent object for computer languages other than R) should contain the RNG seeds and the final state of the Markov chain, so the next run can simply continue this run. A sampler with the "restart" property needs no burn-in (Section 1.11.4 above) and is easily checkpointed (Section 1.14).

The second is the property of outputting batch means for batches of a possibly subsampled chain, also possessed by both the `metrop` and `temper` functions, specified by the arguments `blen` and `nspac`. This property allows very long runs without overly voluminous output. If `nspac = 1` (the default, meaning no subsampling) is used, then no information is lost by the batching. The batches can be used for valid inference—regardless of whether the batch length is long enough for the ordinary method of batch means to work—as described in Section 1.10.3 above.

The third is the property of outputting batch means (for batches of a possibly subsampled chain) for an arbitrary functional of the Markov chain. The `mcmc` and `temper` functions do this via a user-specified function supplied as their `outfun` argument. This allows users to make the inferences they want without rewriting the R package. This makes statistical computer languages in which functions are not first-class objects (like they are in R) unsuitable for MCMC.

1.16 Validating and Debugging MCMC Code

Along with "black box" MCMC (Section 1.11.1) above we introduce the notion of "black box" testing of MCMC code. Black box testing is widely used terminology in software testing. It refers to tests that do not look inside the code, using only its ordinary input and output. Not looking at the code means it cannot use knowledge of the structure of the program or the values any of its internal variables. For MCMC code black box testing means you run the sampler and test that the output has the expected probability distribution.

Since goodness-of-fit testing for complicated multivariate probability distributions is very difficult, black box testing of MCMC code is highly problematic. It is even more so when the sampler is itself black box, so nothing is known about the expected equilibrium distribution except what we may learn from the sampler itself. Thus your humble author has been driven to the conclusion that black box testing of MCMC code is pointless.

Instead testing of the functions `metrop` and `temper` in the `mcmc` package uses a "white box" approach that exposes all important internal variables of the program when the optional argument `debug = TRUE` is specified. In particular, all uniform or normal random variates obtained from R's RNG system are output. This means that, assuming we can trust R's normal and uniform RNG, we can test whether `metrop` and `temper` behave properly as deterministic functions of those pseudorandom numbers obtained from R.

Testing whether a program correctly implements a deterministic function is much easier than testing whether it correctly simulates a specified probability distribution. In addition, when `debug = TRUE` these programs also output proposals, log Hastings ratios, and decisions in the Metropolis rejection step, making it easy to check whether these are correct and hence whether the Metropolis–Hastings algorithm is implemented correctly.

It must be admitted that, although this "white box" testing methodology it much superior to anything your humble author has previously used, it is not guaranteed to find conceptual problems. That is why a clearly written specification (what we called the "theorem" in the preceding section) is so important. During the writing of this chapter just such a conceptual bug was discovered in the `temper` function in versions of the `mcmc` package before 0.8. The terms $q(i, j)$ and $q(j, i)$ in the Hastings ratio for serial tempering (Equation 11.11 in Chapter 11, this volume) were omitted from the code, and the tests of whether the Hastings ratio was calculated correctly were implemented by looking at the code rather than the design document (the file `temper.pdf` in the `doc` directory of every installation of the `mcmc` package), which was correct.

Ideally, the tests should be implemented by someone other than the programmer of the code, a well-recognized principle in software testing. We know of no statistics code that conforms to this practice, perhaps because there is no tradition of refereeing computer code as opposed to papers. The most we can claim is that the "white box" testing methodology used for the `mcmc` would at least make such referring possible.

1.17 The Metropolis–Hastings–Green Algorithm

There are so many ideas in Green (1995) it is hard to know where to start. They include the following:

- State-dependent mixing of updates
- Measure-theoretic Metropolis–Hastings using Radon–Nikodym derivatives
- Per-update augmentation of the state space
- Metropolis–Hastings with Jacobians

any one of which would have been a major contribution by itself.

We have deferred discussion of the MHG algorithm till now because we wanted to avoid measure theory as long as we could. The MHG algorithm cannot easily be discussed without using measure-theoretic terminology and notation.

A kernel $K(x, A)$ is a generalization of regular conditional probability. For a fixed point x in the state space, $K(x, \cdot)$ is a countably-additive real signed measure on the state space. For a fixed measurable set A in the state space, $K(\cdot, A)$ is a measurable real-valued function on the state space. If

$$K(x, A) \geq 0, \quad \text{for all } x \text{ and } A,$$

then we say that K is *nonnegative*. If K is nonnegative and

$$K(x, A) \leq 1, \quad \text{for all } x \text{ and } A,$$

then we say that K is *sub-Markov*. If K is sub-Markov and

$$K(x, S) = 1, \quad \text{for all } x,$$

where S is the state space, then we say that K is *Markov*. A Markov kernel is a regular conditional probability and can be used to describe an elementary update mechanism for a Markov chain or a combined update. In widely used sloppy notation, we write

$$K(x, A) = \Pr(X_{t+1} \in A \mid X_t = x)$$

to describe the combined update (the sloppiness is the conditioning on an event of measure zero).

A kernel K is *reversible* with respect to a signed measure m if

$$\iint g(x)h(y)m(dx)K(x, dy) = \iint h(x)g(y)m(dx)K(x, dy)$$

for all measurable functions g and h such that the expectations exist. A Markov kernel P *preserves* a probability measure π if

$$\iint g(y)\pi(dx)P(x, dy) = \int g(x)\pi(dx)$$

for every bounded function g. Reversibility with respect to π implies preservation of π.

1.17.1 State-Dependent Mixing

Suppose we have a family of updates represented by Markov kernels P_i, $i \in I$. Choose one at random with probability $c_i(x)$ that depends on the current state x, and use it to update the state. The kernel that describes this combined update is

$$P(x, A) = \sum_{i \in I} c_i(x)P_i(x, A).$$

It is *not a theorem* that if each P_i preserves π, then P preserves π. The argument in Section 1.12.8 above does not work.

Define

$$K_i(x, A) = c_i(x)P_i(x, A).$$

If each K_i is reversible with respect to π, then the mixture kernel

$$P(x, A) = \sum_{i \in I} c_i(x)P_i(x, A) = \sum_{i \in I} K_i(x, A)$$

is reversible with respect to π and hence preserves π. This is how state-dependent mixing works.

It is often convenient to allow the identity kernel defined by

$$I(x, A) = \begin{cases} 1, & x \in A, \\ 0, & x \notin A, \end{cases}$$

to be among the P_i. The identity kernel is a Markov kernel that describes a do-nothing update (the state is the same before and after).

Sometimes state-dependent mixing involving the identity kernel is described differently. We insist that

$$c_i(x) \geq 0, \quad \text{for all } i \text{ and } x,$$

and

$$\sum_{i \in I} c_i(x) \leq 1, \quad \text{for all } x.$$

Then when x is the current state the mixture update chooses the ith update with probability $c_i(x)$ and performs the update described by P_i. With the remaining probability

$$1 - \sum_{i \in I} c_i(x)$$

the mixture update does nothing (which is the same as doing the update described by the identity kernel).

1.17.2 Radon–Nikodym Derivatives

Suppose that m is a finite signed measure and n a sigma-finite positive measure defined on the same space. We say that m is *dominated* by n or that m is *absolutely continuous with respect to* n if

$$n(A) = 0 \text{ implies } m(A) = 0, \quad \text{for all events } A.$$

We say that m is *concentrated* on a set C if

$$m(A) = m(A \cap C), \quad \text{for all events } A.$$

We say that measures m_1 and m_2 are *mutually singular* if they are concentrated on disjoint sets.

The Lebesgue–Radon–Nikodym theorem (Rudin, 1987, Theorem 6.10) says the following about m and n as defined above. Firstly, there exist unique finite signed measures m_a and m_s such that m_s and n are mutually singular, m_a is dominated by n, and $m = m_a + m_s$ (this is called the *Lebesgue decomposition*). Secondly, there exists a real-valued function f, which is unique up to redefinition on a set of n measure zero, such that

$$m_a(A) = \int_A f(x) n(dx), \quad \text{for all events } A. \tag{1.25}$$

We say that f is the *density* or *Radon–Nikodym derivative* of m with respect to n and write

$$f = \frac{dm}{dn}.$$

If n is Lebesgue measure and m is dominated by n, then f is an ordinary probability density function. If n is counting measure and m is dominated by n, then f is an ordinary probability

mass function. Hence, the Radon–Nikodym derivative generalizes these concepts. When m is not dominated by n, we have

$$\frac{dm}{dn} = \frac{dm_a}{dn}$$

so the Radon–Nikodym derivative only determines the part of m that is absolutely continuous with respect to n and says nothing about the part of m that is singular with respect to n, but that is enough for many applications.

That the Radon–Nikodym derivative f is unique only up to redefinition on a set of n measure zero would cause a problem if we made a different choice of f every time we used it, but it causes no problem if we fix one choice of f and use it always. (The same issue arises with ordinary probability density functions.)

Radon–Nikodym derivatives are often calculated using ratios. Suppose that m and n are as above and that λ is a measure that dominates both, for example, $\lambda = m + n$. Then we have

$$\frac{dm}{dn} = \frac{dm/d\lambda}{dn/d\lambda}, \tag{1.26}$$

where the right-hand side is interpreted as ordinary division when the denominator is nonzero and an arbitrary choice when the denominator is zero.

To see this, let $f_m = dm/d\lambda$ and $f_n = dn/d\lambda$, let $C = \{ x : f_n(x) \neq 0 \}$, let h be an arbitrary measurable real-valued function, and define

$$f(x) = \begin{cases} f_m(x)/f_n(x), & x \in C, \\ h(x), & x \notin C. \end{cases}$$

By the Lebesgue–Radon–Nikodym theorem, n is concentrated on C. Define a measure m_s by

$$m_s(A) = m(A \setminus C), \quad \text{for all events } A,$$

and let $m_a = m - m_s$. It remains to be shown that m_a is dominated by n and $f = dm_a/dn$. Both are shown by verifying (Equation 1.25) as follows. For any event A,

$$m_a(A) = m(A \cap C) = \int_C f_m \, d\lambda = \int_C f \cdot f_n \, d\lambda = \int_C f \, dn = \int f \, dn$$

(the last equality being that n is concentrated on C).

1.17.3 Measure-Theoretic Metropolis–Hastings

1.17.3.1 Metropolis–Hastings–Green Elementary Update

We now describe the MHG elementary update with state-dependent mixing. For $i \in I$ we have proposal mechanisms described by kernels Q_i. When the current state is x, we choose the ith proposal mechanism with probability $c_i(x)$, generating a proposal y having distribution $Q_i(x, \cdot)$.

The unnormalized measure to preserve is η (the analog of the unnormalized density h in the ordinary Metropolis–Hastings algorithm). Define measures m and m_{rev} by

$$m(B) = \iint 1_B(x,y)\eta(dx)c_i(x)Q_i(x,dy), \tag{1.27a}$$

$$m_{\text{rev}}(B) = \iint 1_B(y,x)\eta(dx)c_i(x)Q_i(x,dy), \tag{1.27b}$$

where $1_B(x,y)$ is equal to one if $(x,y) \in B$ and zero otherwise, so m and m_{rev} are measures on the Cartesian product of the sample space with itself and each B is a measurable subset of that Cartesian product. Define

$$r = \frac{dm_{\text{rev}}}{dm}. \tag{1.27c}$$

Then accept the proposal with probability $\min(1, r(x,y))$.

Note the similarity of this MHG update to the Metropolis–Hastings update (Section 1.12.1 above). It differs in the incorporation of state-dependent mixing so that $c_i(x)$ appears. It also differs in that the *Green ratio* (Equation 1.27c) is actually a Radon–Nikodym derivative rather than a simple ratio like the Hastings ratio (Equation 1.20). The "Metropolis rejection" step—accept the proposal with probability $\min(1, r)$—is the same as in the Metropolis and Metropolis–Hastings algorithms.

As we saw in Equation 1.26, a Radon–Nikodym derivative is often calculated as a ratio, so the terminology "Green ratio" for Equation 1.27c is not so strange. But our main reason for introducing this terminology is the analogy between the Metropolis ratio (Equation 1.24), the Hastings ratio (Equation 1.20), and the Green ratio (Equation 1.27c). People often write things like

$$r(x,y) = \frac{c_i(y)\eta(dy)Q_i(y,dx)}{c_i(x)\eta(dx)Q_i(x,dy)} \tag{1.28}$$

as a sloppy shorthand for actual definition via Equations 1.27a through 1.27c, but Equation 1.28 has no mathematical content other than as a mnemonic for the actual definition.

Green (1995) described a specific recipe for calculating the Green ratio (Equation 1.27c) using the ratio method (Equation 1.26) in the particular case where λ is symmetric in the sense that

$$\iint 1_B(x,y)\lambda(dx,dy) = \iint 1_B(y,x)\lambda(dx,dy) \tag{1.29}$$

for any measurable set B in the Cartesian product of the state space with itself. Such λ always exist. For example, $\lambda = m + m_{\text{rev}}$ works. Then if $f = dm/d\lambda$ and

$$C = \{\, (x,y) : f(x,y) \neq 0 \,\} \tag{1.30}$$

we have

$$r(x,y) = \begin{cases} f(y,x)/f(x,y), & x \in C, \\ 0, & x \notin C. \end{cases} \tag{1.31}$$

It does not matter whether or not we use Green's recipe for calculating (Equation 1.27c). Radon–Nikodym derivatives are unique up to redefinition on sets of measure zero, hence are the same no matter how we calculate them.

Note that the proposal distributions can be anything, described by arbitrary kernels Q_i. Thus the MHG algorithm generalizes the Metropolis–Hastings algorithm about as far as it can go. The only way your humble author can think to generalize this would be to allow state-dependent mixing over a continuum rather than a countable set of Q_i (the way state-independent mixing works; Section 1.12.8 above).

Ordinary Metropolis–Hastings samplers avoid forever the set of x such that $h(x) = 0$, where h is the unnormalized density of the equilibrium distribution (Section 1.12.1 above). Now thinking measure-theoretically, we are reminded that we may redefine h arbitrarily on sets of measure zero under the equilibrium distribution, so the set avoided depends on our choice of h. The MHG algorithm has a similar property. Suppose there is a set N that must be avoided, and $\eta(N) = 0$. Then $m_{\text{rev}}(A \times N) = 0$ for any set A, and we may choose a version of the Green ratio such that $r(x, y) = 0$ for $y \in N$. Then no proposal in N can be accepted, and the chain forever avoids N.

All MCMC ideas discussed above are special cases of the MHG algorithm. Variable-at-a-time Metropolis–Hastings updates are special cases where proposals only change one coordinate. Gibbs updates are special cases where the MHG ratio is always one and the proposal is always accepted.

1.17.3.2 The MHG Theorem

Define

$$\begin{aligned} a(x, y) &= \min\bigl(1, r(x, y)\bigr), \\ b(x) &= 1 - \int a(x, y) Q_i(x, dy). \end{aligned}$$

The kernel describing the MHG elementary update is

$$P_i(x, A) = b(x) I(x, A) + \int_A a(x, y) Q_i(x, dy),$$

and the kernel that we must verify is reversible with respect to η is

$$K_i(x, A) = c_i(x) P_i(x, A),$$

that is, we must verify that

$$\iint g(x) h(y) \eta(dx) c_i(x) P_i(x, dy)$$

is unchanged when g and h are swapped. Since

$$\begin{aligned} \iint g(x) h(y) c_i(x) \eta(dx) P_i(x, dy) = {} & \int g(x) h(x) b(x) c_i(x) \eta(dx) \\ & + \iint g(x) h(y) a(x, y) c_i(x) \eta(dx) Q_i(x, dy), \end{aligned}$$

it clearly is enough to show last term is unchanged when g and h are swapped.

Suppose we have calculated the Green ratio (Equation 1.27c) using Green's recipe (Equation 1.31) with $f = dm/d\lambda$ and λ satisfying Equation 1.29. Then

$$\begin{aligned}
\iint g(x)h(y)a(x,y)c_i(x)\eta(dx)Q_i(x,dy) &= \iint g(y)h(x)a(y,x)c_i(y)\eta(dy)Q_i(y,dx) \\
&= \iint g(y)h(x)a(y,x)m_{\text{rev}}(dx,dy) \\
&= \iint_C g(y)h(x)a(y,x)m_{\text{rev}}(dx,dy) \\
&= \iint_C g(y)h(x)a(y,x)r(x,y)m(dx,dy) \\
&= \iint g(y)h(x)a(y,x)r(x,y)m(dx,dy) \\
&= \iint g(y)h(x)a(y,x)r(x,y)c_i(x)\eta(dx)Q_i(x,dy),
\end{aligned}$$

where C is defined by Equation 1.30, the first equality being the interchange of the dummy variables x and y, the second and sixth equalities being the definitions of m and m_{rev}, the third and fifth equalities being $a(y,x) = 0$ when $(x,y) \in C$, and the fourth equality being $r = dm_{\text{rev}}/dm$ and the fact that the part of m_{rev} that is dominated by m is concentrated on C, as we saw in our discussion of Equation 1.26.

Comparing the expressions at the ends of this chain of equalities, we see that it is enough to show that

$$a(y,x)r(x,y) = a(x,y), \quad \text{whenever } (x,y) \in C, \tag{1.32}$$

because the integrals are the same whether or not they are restricted to C. If $(x,y) \in C$ and $r(x,y) \le 1$, then $a(x,y) = r(x,y)$ and $a(y,x) = 1$, in which case (1.32) holds. If $(x,y) \in C$ and $1 < r(x,y)$, then $a(x,y) = 1$ and

$$a(y,x) = r(y,x) = \frac{1}{r(x,y)}$$

by Equation 1.31, in which case (Equation 1.32) holds again.

Example: Spatial Point Processes

All of this is very abstract. That is the point! But Radon–Nikodym derivatives are nothing to be frightened of. We look at some simple examples to show how the MHG algorithm works in practice.

One only needs the MHG algorithm when proposals are singular with respect to the equilibrium distribution of the Markov chain (otherwise Metropolis–Hastings would do). This often happens when the state space is the union of sets of different dimension. One example of this is spatial point processes. Geyer and Møller (1994) proposed the sampler described here independently of Green (1995), but in hindsight it is a special case of the MHG algorithm.

A spatial point process is a random pattern of points in a region A having finite measure (length, area, volume, ...), both the number of points and the positions of the points being random. A homogeneous Poisson process has a Poisson distributed number of points and the locations of the

points are independent and identically and uniformly distributed conditional on the number. We consider processes having unnormalized densities h_θ with respect to the Poisson processes.

The state space of the Poisson process is

$$\mathcal{A} = \bigcup_{n=0}^{\infty} A^n,$$

where A^0 denotes a set consisting of one point, representing the spatial pattern with no points. The probability measure of the Poisson process is defined by

$$P(B) = \sum_{n=0}^{\infty} \frac{\mu^n e^{-\mu}}{n!} \cdot \frac{\lambda^n(B \cap A^n)}{\lambda(A)^n}, \quad \text{for measurable } B \subset \mathcal{A},$$

where λ is Lebesgue measure on A and μ is an adjustable parameter (the mean number of points). To say that h_θ is an unnormalized density with respect to P means that the probability measure of the non-Poisson process is defined by

$$\begin{aligned} Q_\theta(B) &= \frac{1}{c(\theta)} \int_B h_\theta(x) P(dx) \\ &= \frac{1}{c(\theta)} \sum_{n=0}^{\infty} \frac{\mu^n e^{-\mu}}{n!} \cdot \frac{1}{\lambda(A)^n} \int_{B \cap A^n} h_\theta(x) \lambda^n(dx) \end{aligned}$$

for measurable $B \subset \mathcal{A}$, where

$$c(\theta) = \sum_{n=0}^{\infty} \frac{\mu^n e^{-\mu}}{n!} \cdot \frac{1}{\lambda(A)^n} \int h_\theta(x) \lambda^n(dx).$$

Note that the dimension of x, which is n, is different in different terms of these sums.

Let $n(x)$ denote the number of points in x. We use state-dependent mixing over a set of updates, one for each nonnegative integer i. The ith update is only valid when $n(x) = i$, in which case we propose to add one point uniformly distributed in A to the pattern, or when $n(x) = i + 1$, in which case we propose to delete a point from the pattern. (For definiteness, suppose we add or delete the last point.) The state-dependent mixing probabilities are

$$c_i(x) = \begin{cases} 1/2, & n(x) = i, \\ 1/2, & n(x) = i + 1, \\ 0, & \text{otherwise.} \end{cases}$$

For fixed x have $\sum_i c_i(x) = 1$ except when $n(x) = 0$. In that case, we do nothing (perform the identity update) with probability $1 - \sum_i c_i(x) = 1/2$ following the convention explained at the end of Section 1.17.1.

In order to apply Green's recipe for calculating Radon–Nikodym derivatives for the ith update, we need a symmetric measure on

$$(A^i \times A^{i+1}) \cup (A^{i+1} \times A^i) \tag{1.33}$$

that dominates the joint distribution m of the current state x and the proposal y or its reverse m_{rev}. This symmetric measure cannot be Lebesgue measure on Equation 1.33, because m and m_{rev} are degenerate, their first i coordinates being equal. Thus we choose the symmetric measure Λ that is the image of λ^{i+1} onto the subset of Equation 1.33 where the first i coordinates of the two parts are equal.

On the part of Equation 1.33 where $x \in A^i$ and $y \in A^{i+1}$, we have

$$f(x, y) = \frac{dm}{d\Lambda}(x, y) = \frac{\mu^i e^{-\mu} h_\theta(x)}{i! \lambda(A)^i} \cdot \frac{1}{\lambda(A)},$$

the first part on the right-hand side being the unnormalized density of the equilibrium distribution, unnormalized because we left out $c(\theta)$, which we do not know how to calculate, and the second part being the proposal density. On the part of Equation 1.33 where $x \in A^{i+1}$ and $y \in A^i$, we have

$$f(x, y) = \frac{dm}{d\Lambda}(x, y) = \frac{\mu^{i+1} e^{-\mu} h_\theta(x)}{(i+1)! \lambda(A)^{i+1}} \cdot 1,$$

the first part on the right-hand side being the unnormalized density of the equilibrium distribution, and the second part being the proposal density (which is one because deleting the last point involves no randomness). Thus the Green ratio is

$$r(x, y) = \begin{cases} \dfrac{\mu}{i+1} \cdot \dfrac{h_\theta(y)}{h_\theta(x)}, & x \in A^i \quad \text{and} \quad y \in A^{i+1}, \\ \dfrac{i+1}{\mu} \cdot \dfrac{h_\theta(y)}{h_\theta(x)}, & x \in A^{i+1} \quad \text{and} \quad y \in A^i. \end{cases}$$

We hope readers feel they could have worked this out themselves.

Since point patterns are usually considered as unordered, it is traditional to use $h_\theta(x)$ that is symmetric under exchange of points in pattern. In this case, the update that reorders the points randomly also preserves the stationary distribution. The composition of this random reordering with the update specified above (which deletes the last point) is equivalent to picking a random point to delete.

Example: Bayesian Model Selection

We consider an example done by other means in Chapter 11 of this volume. If we use MHG, there is no need for "padding" parameter vectors. We can just use the parameterization from the problem statement. If, like the ST/US sampler in Section 11.3, we only make jumps between models whose dimensions differ by one, then a very simple MHG proposal simply deletes a component of the parameter vector when moving down in dimension and adds a component distributed normally with mean zero and variance τ^2 independently of the current state when moving up in dimension. If $h(\theta)$ denotes the unnormalized posterior, then a move up in dimension from current state θ to proposed state ψ, which adds a component z to the current state, has Green ratio

$$r(\theta, \psi) = \frac{c_i(\psi) h(\psi)}{c_i(\theta) h(\theta) \phi(z/\tau)/\tau}, \tag{1.34}$$

where ϕ is the probability density function of the standard normal distribution, and a move down in dimension from current state ψ to proposed state θ, which deletes a component z from the current state, has Green ratio that is the reciprocal of the right-hand side of Equation 1.34.

1.17.4 MHG with Jacobians and Augmented State Space

Green (1995) also proposed what is in some respects a special case of MHG and in other respects an extension. We call it Metropolis–Hastings–Green with Jacobians (MHGJ). This

version is so widely used that many users think that MHGJ is the general version. This form of elementary update moves between parts of the state space that are Euclidean spaces of different dimension, hence it is often called "dimension jumping"—although that name applies to other examples, such as the preceding one, that do not involve Jacobians.

Suppose that the state space is a disjoint union

$$S = \bigcup_{m \in M} S_m,$$

where S_m is a Euclidean space of dimension d_m. We assume that the equilibrium distribution of the Markov chain is specified by an unnormalized density $h(x)$ with respect to Lebesgue measure on S. MHGJ elementary updates move from one S_m to another. Say the ith elementary update moves between $S_{m(i)}$ and $S_{n(i)}$. Thus it only makes sense to have $c_i(x) > 0$ when $x \in S_{m(i)} \cup S_{n(i)}$.

Let $U_{m(i)}$ and $U_{n(i)}$ be Euclidean spaces such that $S_{m(i)} \times U_{m(i)}$ is the same dimension as $S_{n(i)} \times U_{n(i)}$. We specify a proposal density $q_i(x, \cdot)$, which describes the conditional distribution of the proposal u given the current state x such that $u \in U_{m(i)}$ when $x \in S_{m(i)}$ and $u \in U_{n(i)}$ when $x \in S_{n(i)}$. We also specify a function g_i that maps points in $S_{m(i)} \times U_{m(i)}$ to points in $S_{n(i)} \times U_{n(i)}$ and vice versa and which is its own inverse.

The MHGJ proposal is a combination of two steps. First generate a random u from the distribution $q_i(x, \cdot)$. Then propose $g_i(x, u) = (y, v)$. The MHG ratio is

$$r(x, u, y, v) = \frac{c_i(y)h(y)q_i(y, v)}{c_i(x)h(x)q_i(x, u)} \cdot \det\bigl(\nabla g_i(x, u)\bigr),$$

the last factor being the Jacobian of the mapping g_i. This is followed by the usual Metropolis rejection: accept the proposal with probability $\min(1, r)$.

For examples of the MHGJ algorithm, see Chapter 3 (this volume).

1.17.4.1 The MHGJ Theorem

The MHGJ algorithm, because of its per-update augmentation of $U_{m(i)}$ and $U_{n(i)}$, does not exactly fit in the pattern of the MHG algorithm described above. Thus we give a separate proof.

The proof starts just like the one in Section 1.17.3.2. We see that we can deal with one arbitrary elementary update, and consequently only one pair of state augmentations. Whenever one augments the state, there are two issues: what is the equilibrium distribution on the augmented state space, and how does it relate to the distribution of interest on the original state? Here the augmented state is (x, u), the equilibrium distribution on the augmented state space has unnormalized density with respect to Lebesgue measure $h(x)q_i(x, u)$. The original state is x and the distribution of interest with unnormalized density $h(x)$ is a marginal of it. The proposal $(y, v) = g(x, u)$ is deterministic.

We now determine the Radon–Nikodym derivative of the distribution of (y, v) with respect to (x, u). We use the ratio method, determining first the Radon–Nikodym derivatives of each with respect to Lebesgue measure λ on the space where (x, u) lives. We have

$$\frac{dm}{d\lambda} = c_i(x) \cdot h(x)q_i(x, u),$$

$$\frac{dm_{\text{rev}}}{d\lambda} = c_i(y) \cdot h(y)q_i(y, v) \cdot \det\bigl(\nabla g_i(x, u)\bigr),$$

where in the latter the Jacobian arises from the multivariate change-of-variable theorem, because we are differentiating with respect to (x, u) rather than (y, v).

Acknowledgments

This chapter benefited from detailed comments by Christina Knudson, Leif Johnson, Galin Jones, and Brian Shea.

References

Bélisle, C. J. P., Romeijn, H. E., and Smith, R. L. 1993. Hit-and-run algorithms for generating multivariate distributions. *Mathematics of Operations Research*, 18:255–266.

Besag, J. and Green, P. J. 1993. Spatial statistics and Bayesian computation (with discussion). *Journal of the Royal Statistical Society, Series B*, 55:25–37.

Chan, K. S. and Geyer, C. J. 1994. Discussion of the paper by Tierney (1994). *Annals of Statistics*, 22:1747–1758.

Chen, M.-H. and Schmeiser, B. 1993. Performance of the Gibbs, hit-and-run, and Metropolis samplers. *Journal of Computational and Graphical Statistics*, 2:251–272.

Clifford, P. 1993. Discussion of Smith and Roberts (1993), Besag and Green (1993), and Gilks et al. (1993). *Journal of the Royal Statistical Society, Series B*, 55:53–54.

Freedman, D., Pisani, R., and Purves, R. 2007. *Statistics*, 4th edn. W. W. Norton, New York.

Gelfand, A. E. and Smith, A. F. M. 1990. Sampling-based approaches to calculating marginal densities. *Journal of the American Statistical Association*, 85:398–409.

Gelman, A., Roberts, G. O., and Gilks, W. R. 1996. Efficient Metropolis jumping rules. In J. M. Bernardo, J. O. Berger, A. P. Dawid, and A. F. M. Smith (eds), *Bayesian Statistics 5: Proceedings of the Fifth Valencia International Meeting*, pp. 599–607. Oxford University Press, Oxford.

Geman, S. and Geman, D. 1984. Stochastic relaxation, Gibbs distribution, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6:721–741.

Geyer, C. J. 1992. Practical Markov chain Monte Carlo (with discussion). *Statistical Science*, 7:473–511.

Geyer, C. J. 1994. On the convergence of Monte Carlo maximum likelihood calculations. *Journal of the Royal Statistical Society, Series B*, 56:261–274.

Geyer, C. J. 1999. Likelihood inference for spatial point processes. In O. E. Barndorff-Nielsen, W. S. Kendall, and M. N. M. van Lieshout (eds), *Stochastic Geometry: Likelihood and Computation*, pp. 79–140. Chapman & Hall/CRC, Boca Raton, FL.

Geyer, C. J. 2010a. Computation for the Introduction to MCMC chapter of *Handbook of Markov Chain Monte Carlo*. Technical Report 679, School of Statistics, University of Minnesota. http://purl.umn.edu/92549.

Geyer, C. J. 2010b. *mcmc: Markov Chain Monte Carlo*. R package version 0.8, available from CRAN.

Geyer, C. J. and Møller, J. 1994. Simulation and likelihood inference for spatial point processes. *Scandinavian Journal of Statistics*, 21:359–373.

Geyer, C. J. and Thompson, E. A. 1992. Constrained Monte Carlo maximum likelihood for dependent data (with discussion). *Journal of the Royal Statistical Society, Series B*, 54:657–699.

Geyer, C. J. and Thompson, E. A. 1995. Annealing Markov chain Monte Carlo with applications to ancestral inference. *Journal of the American Statistical Association*, 90:909–920.

Gilks, W. R., Clayton, D. G., Spiegelhalter, D. J., Best, N. G., and McNeil, A. J. 1993. Modelling complexity: Applications of Gibbs sampling in medicine (with discussion). *Journal of the Royal Statistical Society, Series B*, 55:39–52.

Glynn, P. W. and Whitt, W. 1991. Estimating the asymptotic variance with batch means. *Operations Research Letters*, 10:431–435.

Glynn, P. W. and Whitt, W. 1992. The asymptotic validity of sequential stopping rules for stochastic simulations. *Annals of Applied Probability*, 2:180–198.

Green, P. J. 1995. Reversible jump Markov chain Monte Carlo computation and Bayesian model determination. *Biometrika*, 82:711–732.

Hammersley, J. M. and Handscomb, D. C. 1964. *Monte Carlo Methods*. Methuen, London.

Hastings, W. K. 1970. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57:97–109.

Jones, G. L. 2004. On the Markov chain central limit theorem. *Probability Surveys*, 1:299–320.

Kendall, W. S. and Møller, J. 2000. Perfect simulation using dominating processes on ordered spaces, with application to locally stable point processes. *Advances in Applied Probability*, 32:844–865.

Kipnis, C. and Varadhan, S. R. S. 1986. Central limit theorem for additive functionals of reversible Markov processes and applications to simple exclusions. *Communications in Mathematical Physics*, 104:1–19.

MacEachern, S. N. and Berliner, L. M. 1994. Subsampling the Gibbs sampler. *American Statistician*, 48:188–190.

Meketon, M. S. and Schmeiser, B. W. 1984. Overlapping batch means: Something for nothing? In S. Sheppard, U. Pooch, and D. Pegden (eds), *Proceedings of the 1984 Winter Simulation Conference*, pp. 227–230. IEEE Press Piscataway, NJ.

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E. 1953. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21:1087–1092.

Meyn, S. P. and Tweedie, R. L. 1993. *Markov Chains and Stochastic Stability*. Springer, London.

Propp, J. G. and Wilson, D. B. 1996. Exact sampling with coupled Markov chains and applications to statistical mechanics. *Random Structures and Algorithms*, 9:223–252.

Roberts, G. O. and Rosenthal, J. S. 1997. Geometric ergodicity and hybrid Markov chains. *Electronic Communications in Probability*, 2:13–25.

Roberts, G. O. and Rosenthal, J. S. 2004. General state space Markov chains and MCMC algorithms. *Probability Surveys*, 1:20–71.

Rudin, W. 1987. *Real and Complex Analysis*, 3rd edn. McGraw-Hill, New York.

Schmeiser, B. 1982. Batch size effects in the analysis of simulation output. *Operations Research*, 30:556–568.

Smith, A. F. M. and Roberts, G. O. 1993. Bayesian computation via the Gibbs sampler and related Markov chain Monte Carlo methods (with discussion). *Journal of the Royal Statistical Society, Series B*, 55:3–23.

Stigler, S. M. 2002. *Statistics on the Table: The History of Statistical Concepts and Methods*. Harvard University Press, Cambridge, MA.

Tanner, M. A. and Wong, W. H. 1987. The calculation of posterior distributions by data augmentation (with discussion). *Journal of the American Statistical Association*, 82:528–550.

Tierney, L. 1994. Markov chains for exploring posterior distributions (with discussion). *Annals of Statistics*, 22:1701–1762.