14

Bayesian Analysis of Spatially Autocorrelated Data

14.1 Introduction

Bayesian statistics is based on *Bayes' rule*, which is named after the Rev. Thomas Bayes, who discussed it in a paper published posthumously in 1763. We can introduce the concept as follows. Let A and B be two events. For example, suppose we are sampling leaves in a wheat field to determine whether the mean nitrogen level of the plants in the field exceeds the recommended minimum for an adequate supply of nitrogen. Let A be the event that the mean N level of a particular leaf sample exceeds the recommended minimum, and let B be the event that mean N level over all of the plants in the field exceeds the recommended minimum. The *joint probability*, denoted $P\{A, B\}$ is the probability that both events A and B occur. The *conditional probability* that A occurs given that B occurs is denoted $P\{A \mid B\}$, and one can similarly define the conditional probability $P\{B \mid A\}$ that B occurs given that A occurs. The laws of probability state that (Larsen and Marx, 86, p. 42)

$$P\{A, B\} = P\{B \mid A\}P\{A\} = P\{A \mid B\}P\{B\}, \tag{14.1}$$

and these can be use to formally define the conditional probability. Bayes' rule is obtained by dividing the second equation in (Equation 14.1) by $P\{A\}$ (Koop, 2003, p. 1):

$$P\{B \mid A\} = \frac{P\{A \mid B\}P\{B\}}{P\{A\}}. \tag{14.2}$$

If we have observed the event A, that the mean N level of the sample exceeds the minimum, then Bayes' rule gives us a means to compute the probability that B occurs, that is, that the field mean N level exceeds the minimum.

Let us now see how Bayesian statistics is applied to data rather than events. Instead of expressing the model in terms of events A and B as in the previous paragraph, we will employ as data a vector (which may have only one component) Y of observed values of a random variable or variables, and a vector θ of parameters describing the distribution of these random variables. Rather than considering the probability of events, we consider probability *densities*, which we denote with a lower case p. For example, let Y again be mean N level of a sample of plants in the field (in this case the vector Y does have only one component), and let θ be the mean plant leaf nitrogen content for the entire population of plants. Then Bayes' rule becomes

$$p(\theta \mid Y) = \frac{p(Y \mid \theta)p(\theta)}{p(Y)}. \tag{14.3}$$

The key distinction between Bayesian statistics and classical statistics is in how the parameters contained in the parameter vector θ are treated. In the classical, or *frequentist*, interpretation of statistics, the vector θ is not a random variable but rather a parameter or set of parameters fixed by nature, and thus it makes no sense to speak of its probability distribution. In the *Bayesian* interpretation of statistics, the vector θ *is* a random variable, and its distribution represents the state of our certainty about its value (in this case, the mean plant N content in the field). This interpretation is called *subjective probability*. The density $p(\theta)$ is called the *prior probability density*, or just the *prior*. It represents the state of our knowledge about the value of θ before making an observation. The density $p(Y \mid \theta)$ is called the *likelihood*, and indeed it has the same form as the likelihood described in Appendix A.5 and in our earlier discussions about maximum likelihood, although its interpretation is different in a Bayesian context. In the context of Appendix A.5, the likelihood is considered as a function $L(\theta \mid Y)$, and the objective is to determine a value of Y to maximize θ. In the Bayesian context, $p(Y \mid \theta)$ describes the probability density of observations Y for each value of θ, or, roughly speaking, the probability of observing values of Y given values of θ. The density $p(Y)$ is called the *marginal density* and in principle can be computed as

$$p(Y) = \int p(Y \mid \theta)p(\theta)d\theta. \tag{14.4}$$

The density $p(\theta \mid Y)$ in Equation 14.3 is called the *posterior density* and represents the updated belief about the value of θ taking into account the observation Y. Some measure of central tendency such as the mean or median of these densities may be taken to be a good point estimate of θ given the available knowledge, and a measure of spread such as the variance or quantile may be taken as representative of the degree of uncertainty about that value.

The following illustration of how Bayes' rule works in this context is taken from Box and Tiao (1973, p. 15). Suppose there is a parameter θ of a variable Y that is to be estimated. We continue with the example that Y is the concentration of nitrogen in the tissue of a sample of wheat plants in a field, and θ is the mean value over the whole field. Suppose further that one scientist, called Scientist A, is fairly sure that the value of θ lies between 880 and 920. We interpret "fairly sure" to mean that one standard deviation of Scientist A's belief has the value 20. Therefore, we can represent this belief using a normal probability density as

$$p_A(\theta) \sim N(900, 20^2). \tag{14.5}$$

This is the subjective probability of Scientist A prior to collecting data. Suppose that another scientist, Scientist B, is a little less sure of the value of θ but thinks it is lower, specifically between 720 and 880. Scientist B's prior can be represented as

$$p_B(\theta) \sim N(800, 80^2). \tag{14.6}$$

The curves in Figure 14.1 show the prior distributions $p_A(\theta)$ and $p_B(\theta)$. Suppose now that a single observation Y_1 of the variable Y is made, with mean $Y_1 = 850$. Let σ_1 be the standard deviation of the measurement, representing the magnitude of uncertainty in the

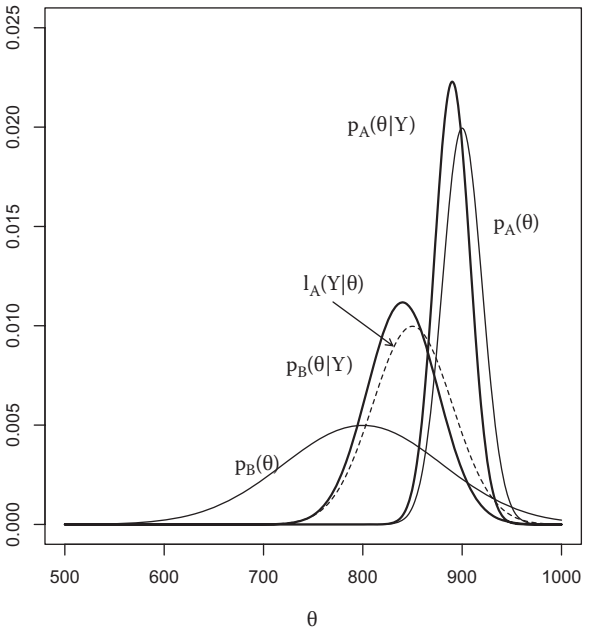

FIGURE 14.1
Prior, likelihood, and posterior densities of the beliefs of Scientist A and Scientist B regarding the value of the parameter θ.

measurement, and suppose $\sigma_1 = 40$. This measurement is interpreted as the likelihood function, so that we can write

$$l(Y \mid \theta) = l(850 \mid \theta) \sim N(850, 40^2). \tag{14.7}$$

With this information, we can compute the posterior densities $p_A(\theta \mid Y)$ and $p_B(\theta \mid Y)$. The formula for $p_A(\theta \mid Y)$ is

$$\begin{aligned} p_A(\theta \mid Y) &= \frac{l(Y \mid \theta) p_A(\theta)}{\int_{-\infty}^{\infty} l(Y \mid \theta) p_A(\theta) d\theta} \\ &= \frac{f_N(\theta; 850, 40^2) f_N(\theta, 900, 20^2)}{\int_{-\infty}^{\infty} f_N(\theta; 850, 40^2) f_N(\theta, 900, 20^2) d\theta}, \end{aligned} \tag{14.8}$$

where $f_N(\theta, \mu, \sigma^2)$ represents the normal density function. It turns out that this computation can be done analytically. The usual symbols for the parameters of the prior and posterior distribution involve bars above and below the character. However, since we use the bars to denote sample means, we will use the very nonstandard notation $\hat{\theta}$ and $\hat{\sigma}^2$ to denote parameters of the prior distribution and $\breve{\theta}$ and $\breve{\sigma}^2$ to denote parameters of the posterior distribution. Given a normal prior $p(\theta) \sim N(\hat{\theta}, \hat{\sigma}^2)$ and a likelihood $l(\theta \mid Y) \sim N(Y, \sigma_1^2)$, Box and Tiao (1973, p. 74) show that $p(\theta \mid Y) \sim N(\breve{\theta}, \breve{\sigma}^2)$ where

$$\breve{\theta} = \frac{(\hat{\tau}\hat{\theta} + \tau_1 Y)}{\hat{\tau} + \tau_1}, \quad \frac{1}{\breve{\sigma}^2} = (\hat{\tau} + \tau_1)$$
$$\hat{\tau} = \frac{1}{\hat{\sigma}^2}, \quad \tau_1 = \frac{1}{\sigma_1^2}. \tag{14.9}$$

The quantities $\hat{\tau}$ and τ_1, which are the inverse of the variances $\hat{\sigma}^2$ and σ_1^2, are called the *precision* of the prior and the data, respectively, and are commonly used instead of the variance in formulas in Bayesian statistics. Plugging in the numbers for Scientist A, we have $\hat{\theta} = 900$, $\hat{\sigma}^2 = 20^2 = 400$, $Y_1 = 850$, and $\sigma_1^2 = 40^2 = 1600$. The result is that $p_A(\theta \mid Y)$ is a normal density with mean 890 and standard deviation 17.9. Similarly, $p_B(\theta \mid Y)$ is a normal density with mean 840 and standard deviation 35.7. Figure 14.1 shows these results. The measurement $Y_1 = 850$ is less than the mean value of Scientist A's prior and greater than that of Scientist B, so Scientist A's posterior is reduced a bit and Scientist B's posterior is increased. The standard deviations of both scientists' posteriors are reduced, reflecting greater certainty as a result of new information. The effect both on the mean and the standard deviation is greater for Scientist B, who was less certain in the first place.

This simple example illustrates two very important aspects of Bayesian analysis. The first is that in order to compute the posterior density one must compute the marginal density by evaluating an integral. In the case of the normal density, this can be done analytically, but for more general densities it would have to be done numerically. The second aspect is that in this example the posterior density has the same form as the prior; both are normal. This is a very special and convenient property of the normal density. It means that if a second measurement is made, the posterior just computed can serve as the prior for a second calculation. For other densities besides the normal, the likelihood must be carefully chosen, if it exists at all, in order for the posterior density to have the same form as the prior. A prior that, given the appropriate likelihood function, produces a posterior whose density has the same form as the prior is called a *conjugate prior*. These two factors, the need to evaluate an integral and the need for a conjugate prior and corresponding likelihood function, were the primary reasons why, before the advent of reasonably powerful computers, Bayesian calculations had to be carried out using a very restricted set of prior and likelihood functions (e.g., Plant and Wilson, 1985). The great advance in Bayesian statistics in recent years has been the development of powerful numerical techniques that remove this restriction.

When, as in this example, the prior and posterior form a conjugate pair, the posterior $p(\theta \mid Y)$ can be used as the prior for a second measurement, and the process may be repeated. To continue the example of Box and Tiao (1973, p. 18), suppose that n independent observations of Y are drawn from a normal population with mean θ and variance σ^2. Then the likelihood is normally distributed with mean $\bar{Y}$ variance σ^2/n. Therefore, from Equation 14.9, the posterior is normally distributed with mean and variance (or precision) given by

$$\breve{\theta} = \frac{(\hat{\tau}\hat{\theta} + \tau_1 \bar{Y})}{\hat{\tau} + \tau_n}, \quad \frac{1}{\breve{\sigma}^2} = (\hat{\tau} + \tau_n)$$
$$\hat{\tau} = \frac{1}{\hat{\sigma}^2}, \quad \tau_n = \frac{n}{\sigma_1^2}. \tag{14.10}$$

In our example, with 100 observations drawn from a population with a mean of 870 and a variance of 40^2, the posterior has a distribution $p_A(\theta \mid Y) \sim N(871.2, 3.9^2)$.

An important feature of Bayes' rule as formulated in Equation 14.3 is that the denominator does not depend on θ, and thus for a given set of data Y, the denominator can be regarded as a constant. Thus we can write

$$p(\theta \mid Y) \propto p(Y \mid \theta)p(\theta), \tag{14.11}$$

where the symbol $\propto$ means "is proportional to." The verbal statement of Expression 14.11, "the posterior is proportional to the likelihood times the prior," is something of a Bayesian mantra. It means that, although the exact value of the posterior density $p(\theta \mid Y)$ cannot be determined, one can determine its shape from Expression 14.11 without having to compute the integral in Equation 14.4.

The last concept in Bayesian statistics to discuss in this section is the *noninformative prior* (Box and Tiao, 1973, p. 32; Koop, 2003, p. 6). Informally speaking, a prior density is noninformative if the posterior density $p(\theta \mid Y)$ is unaffected, or almost unaffected, by the parameters of the prior. To continue the example above, if the prior has mean 890 and variance 10^6 and again the likelihood has mean 850 and variance 40^2, then the posterior has mean 850.06 and variance 1597.44. Thus, the high value of the prior variance causes the prior to provide little information about the values of the posterior.

The availability of inexpensive, powerful computers beginning in the 1980s motivated the development of numerical methods that eliminate the need to restrict Bayesian analysis to likelihood functions that are associated with conjugate priors. In the next section, we discuss the most commonly used of these methods, and their application to linear and generalized linear regression models.

14.2 Markov Chain Monte Carlo Methods

Following Koop (2003), the first application we consider from a Bayesian perspective is simple linear regression. To further simplify the discussion, we initially assume that the explanatory variable X and the response variable Y have been centered, so that there is no intercept term. The model is then

$$Y_i = \beta X_i + \varepsilon_i,\ i = 1, \ldots, n, \tag{14.12}$$

where the $\varepsilon_i \sim N(0, 1/\tau)$ (from now on we will express the normal distribution in terms of τ rather than σ^2) are independent and identically distributed and the X_i are either a set of values of a mathematical variable or a set of values of a random variable independent of the ε_i. For now we will treat the values of the explanatory variable X as fixed (the mathematical variable case). The *gamma distribution* plays a major role in Bayesian linear regression analysis. This distribution is characterized by two parameters that we will denote μ and ν. The formula for the gamma density function can be written in several ways; the formula we will use can be specified in R as

$$f_G(Y \mid \mu, \nu) = \begin{cases} \dfrac{\nu^{-\mu} Y^{\mu-1} e^{-Y\nu}}{\Gamma(\mu)}, & \text{if } 0 < Y < \infty \\ 0 & \text{otherwise}, \end{cases} \tag{14.13}$$

where $\Gamma(\mu)$ is the gamma function (Abramowitz and Stegun, 1964). The gamma density has mean μ/ν and variance μ/ν^2. The parameter μ is called the *shape* parameter, and ν is called the *rate* parameter. We will use the notation $G(\mu,\nu)$ to denote a gamma density with parameters μ and ν as in Equation 14.13.

The unknown parameters in the linear regression model (Equation 14.12) are the regression coefficient β and the error precision $\tau = 1/\sigma^2$. Remember that in the Bayesian formulation, these are random variables, as was the case in Section 14.1. We begin the analysis with a prior probability density for the vector $\theta = (\beta,\tau)'$ that represents our subjective belief in the values of the parameters. The subjective formulation requires a prior probability density $p(\beta,\tau)$ that represents our belief prior to data collection, and a likelihood $p(Y \mid \beta,\tau)$ that represents the probability of observing the data that we collect given the values of β and τ. If we don't know anything about these values, we use a noninformative prior.

The conjugate prior for the normal regression model is a probability density called the *normal-gamma*. The mathematics of this density are discussed by Koop (2003, Ch 2–3). The advantage of using a conjugate prior, as we did in Section 14.1, is that we can obtain an analytical expression for the posterior. The problem with using the normal-gamma density as a prior is that one cannot independently specify prior probabilities for the regression coefficient β and the error precision τ. Instead, one must first specify the prior for the error precision τ and then specify the prior for β conditional on τ. This is often inconvenient and motivates the consideration of numerical methods for dealing with non-conjugate priors. This is the approach we will take.

It is very common, and convenient, in the numerical Bayesian analysis of linear regression models to specify a normal prior for β and a gamma prior for the precision τ. We will write the priors as

$$p(\beta) \sim N(\hat{\beta},\hat{\tau}_\beta),\ p(\tau) \sim G(\hat{\mu}_\tau,\hat{\nu}_\tau), \tag{14.14}$$

to indicate that the prior for β is a normal density with mean $\hat{\beta}$ and precision $\hat{\tau}_\beta$, and the prior for τ is a gamma density with shape and rate parameters $\hat{\mu}_\tau$ and $\hat{\nu}_\tau$. Remember that the hats are used to denote the parameter values of the prior.

Since β and τ are independent, the joint prior $p(\beta,\tau)$ is the product of the two priors in Equation 14.14. The likelihood is the same as the likelihood function developed for linear regression in Appendix A.5 and given in Equation A.50, except of course that β_0 is not present in Equation 14.12 because in the present case it has the value 0. From Equation A.50, the likelihood is given by

$$L(\beta,\sigma^2 \mid Y) = \frac{\tau}{(2\pi)^{n/2}} \exp(-\tau \sum_{i=1}^{n} (Y_i - \beta X_i)^2 / 2). \tag{14.15}$$

Using the relationship in Expression 14.11, that the posterior is proportional to the likelihood times the prior, gives the following

$$p(\beta,\tau \mid Y) \propto p(Y \mid \beta,\tau)p(\beta)p(\tau). \tag{14.16}$$

Here we have used the fact that β and τ are independent, so that $p(\beta,\tau) = p(\beta)p(\tau)$, where $p(\beta)$ is the normal prior of β and $p(\tau)$ is the gamma prior of the precision τ. Plugging in the thee densities (likelihood plus two priors) and substituting gives

$$p(\beta,\tau \mid Y) \propto \tau^{n/2} \exp\left(-\frac{\tau}{2}\left[\sum_{i=1}^{n} (Y_i - \beta X_i)^2 \right] \right) \exp\left(-\frac{\widehat{\tau}_\beta}{2}(\beta - \widehat{\beta})^2 \right) \times \tau^{\widehat{\mu}-1} \exp\left(-\tau \widehat{\nu}\right). \tag{14.17}$$

Note that the first part of the right-hand side (the first exponential function), which comes from the likelihood, contains the parameters τ and β, while the second part, which comes from the prior, contains their values in the prior $\widehat{\beta}$ and $\widehat{\tau}_\beta$. Koop (2003, p. 61) shows that given these densities, the posterior is proportional to the product of a normal density and a gamma density,

$$p(\beta,\tau \mid Y) \propto N(\breve{\beta},\breve{\sigma}_\beta^2) G(\breve{\mu}_\tau,\breve{\nu}_\tau), \tag{14.18}$$

where the symbols N and G again represent normal and gamma densities, respectively. Because of Expression 14.18, if τ is fixed, then the density of β conditional on τ is a normal density, and if β is fixed, then the density of τ conditional on β is a gamma density. The parameters in Expression 14.18 are, in our notation,

$$\begin{aligned}
\breve{\tau}_\beta &= \widehat{\tau}_\beta + \tau \sum_{i=1}^{n} X_i^2 \\
\breve{\beta} &= \frac{1}{\breve{\tau}_\beta}\left(\frac{\widehat{\beta}}{\widehat{\sigma}_\beta^2} + \tau \sum_{i=1}^{n} X_i Y_i \right) \\
\breve{\mu}_\tau &= n + \widehat{\mu}_\tau \\
\breve{\nu}_\tau &= \frac{\widehat{\nu}_\tau + \sum_{i=1}^{n} (Y_i - \beta X_i)^2}{2}
\end{aligned} \tag{14.19}$$

These equations are a bit complicated, so let's sort through them before moving on. Before we do, however, note that the first two equations define the parameters of the conditional probability density $p(\beta \mid \tau, Y)$, and the second two equations define the parameters of the gamma density $G(\mu,\nu)$. Keep this in mind, because it will be important when we discuss the Gibbs sampler below.

The quantities we are trying to estimate are the regression coefficient β and the precision τ, which is the inverse of the error variance. These appear themselves in the right-hand sides of three of the four equations (Equation 14.19) because the first two equations define the parameters of the posterior density of β conditional on τ, and the second two equations define the parameters of the posterior density of τ conditional on β. The parameters of the prior densities, with the hats (i.e., $\widehat{\beta}$ and so forth), also appear on the right-hand sides, and the posterior parameter $\breve{\tau}_\beta$ also appears on the right-hand side of the second equation, because the value of β is conditional on the value of τ. On the left-hand sides are all of the parameters of the posterior density, with the inverted hats. This means, however, that the equations that we use to estimate β and τ have β and τ themselves on their right-hand sides. Since we need β and τ to estimate β and τ, we are going to have to use some sort of iterative procedure.

The posterior density defined in Expression 14.18 does not have a convenient form that would allow us to express its mean and variance analytically, the way we can with a normal or a gamma density. Therefore, to compute these posterior parameters, we must use a numerical algorithm. The algorithm we will describe can be used to calculate a wide range of parameters, but we will focus on $\breve{\beta}$, the posterior mean, the estimated value of the regression coefficient β, and $\breve{\tau}$, the estimated precision (inverse of the variance) of the error, both after taking into account the observed values of the data.

To simplify things, let us return for the moment to the notation of Section 14.1, where we were dealing with only a single parameter θ. Consider the numerical calculation of the posterior mean,

$$\breve{\theta} = E\{\theta \mid Y\} = \int_{-\infty}^{\infty} \theta p(\theta \mid Y) d\theta. \tag{14.20}$$

A theorem of probability theory called the *weak law of large numbers* states (Larsen and Marx, 1986, p. 191) that if we draw a sequence of random numbers $\{\theta^{(0)}, \theta^{(1)}, ..., \theta^{(n)}\}$ from the probability distribution $p(\theta \mid Y)$, then as n increases the sample mean $\Sigma\theta^{(i)}/n$ converges to the expected value $\breve{\theta} = E\{\theta \mid Y\}$ of this distribution. This use of a sequence of random numbers to compute an expected value is called *Monte Carlo integration*. We mention in passing that this can also be extended to state that if we draw a sequence $\{g(\theta_1), g(\theta_2), ..., g(\theta_n)\}$ from the distribution, then the sample mean of this sequence converges to $E\{g(\theta)\} = \int_{-\infty}^{\infty} g(\theta) p(\theta \mid Y) d\theta$. However, for now we will stick to the case $g(\theta) = \theta$.

Thus, we would like to draw a sequence of random values $\theta^{(i)}$ from the posterior distribution $p\{\theta \mid Y\}$, because this sequence would enable us to compute a quantity that converges to $\breve{\theta}$. Although we don't know $p\{\theta \mid Y\}$ (if we did, we could compute $E\{\theta \mid Y\}$ directly), we do know its shape from Expression 14.18 ("the posterior is proportional to the likelihood times the prior"), and we can use this in the numerical calculation of the parameters in Equation 14.19 by employing Monte Carlo methods.

One very common algorithm for the numerical calculations described in the previous paragraph is called the *Gibbs sampler* (German and German, 1984). In order to minimize the notational complexity, we will describe the Gibbs sampler for the specific problem of Equation 14.12, $Y_i = \beta X_i + \varepsilon_i$, $i = 1, ..., n$, $\varepsilon_i \sim N(0, 1/\tau)$, in which we are interested in two quantities, the regression coefficient β and the precision τ.

From Expression 14.18, the posterior density $p(\beta, \tau \mid Y)$ is proportional to the product of a normal density and a gamma density. The conditional probability density $p(\beta \mid \tau, Y)$ is obtained by holding τ to a fixed value, and therefore it must be the case that this conditional density is normal, that is, $\beta \mid \tau, Y \sim N(\breve{\beta}, \breve{\tau}_\beta)$. Similarly, $p(\tau/\beta, Y)$ must be a gamma density, that is, $\tau \mid \beta, Y \sim G(\breve{\mu}, \breve{\nu})$. Consider the case of calculating $\breve{\beta}$. We would like to use Monte Carlo integration by drawing random numbers from the posterior density $p(\beta, \tau \mid Y)$ so that we could apply the weak law of large numbers to obtain a sequence $\beta^{(0)}, \beta^{(1)}, \beta^{(2)}, ...$ that converges to the expected value in Equation 14.20, which in this case is $\breve{\beta}$. We can't draw from $p(\beta, \tau \mid Y)$ because we don't know it. We can, however, draw from the conditional densities $p(\beta \mid \tau, Y)$ and $p(\tau \mid \beta, Y)$ since they are a normal and a gamma, respectively. The Gibbs sampler is based on drawing from these densities.

The Gibbs sampler works like this. Suppose we could find an initial value $\tau^{(0)}$ drawn from $p(\tau \mid \beta, Y)$. We can't, because we don't know the parameters of the distribution, but suppose we could. Step 1 of the Gibbs sampler algorithm is to substitute this value for τ into

the first two equations of Equation 14.19, which define the parameters of the conditional density $p(\beta \mid \tau, Y)$. Step 2 is to draw a value of $\beta^{(0)}$ from this density and substitute it for β in the second two equations of Equation 14.19 to produce a gamma density. From this density, we draw second value $\tau^{(1)}$, which we then plug into the first two equations to produce a density from which to draw $\beta^{(1)}$. We continue alternating in this way and generate random sequences of $\beta^{(i)}$ and $\tau^{(i)}$ that we can use in Monte Carlo integration.

The difficulty with this algorithm is that we do not have a density from which to draw the $\tau^{(0)}$ to start with. However, it turns out that the starting values of the sequences often do not matter, and that ultimately the alternating $\beta^{(i)}$ and $\tau^{(i)}$ do come from the correct densities even if the initial values of the sequence do not. Since we compute the expected value of a parameter by averaging all of the values of the sequence, we don't want to include the initial values that do not come from the correct density. Therefore, the usual procedure in employing the Gibbs sampler is to initialize the process by first generating sequences of length B of values $\beta^{(0)},...,\beta^{(B-1)}$ and $\tau^{(0)},...,\tau^{(B-1)}$, called the *burn-in values*, and then to discard these. One continues with the sequence and uses the values $\beta^{(B)},...,\beta^{(n)}$ and $\tau^{(B)},...,\tau^{(n)}$ in Monte Carlo integration to estimate $\breve{\beta}$ and $\breve{\tau}$.

The Gibbs sampler is one of a class of methods called *Markov Chain Monte Carlo* (or *MCMC*) methods. We have already seen enough Monte Carlo methods in earlier chapters to know where this term comes from, but what about the Markov Chain part? A *Markov Chain* is a stochastic sequence in which the value of each member of the sequence depends explicitly only on the value in the sequence that immediately precedes it (as well as random noise). We have already seen an example of a Markov Chain: the autoregressive time series $Y_i - \mu = \lambda(Y_{i-1} - \mu) + \varepsilon_i$, $i = 1,2,...$ of Equation 3.13. Note that each value Y_i in this sequence depends explicitly only on Y_{i-1} and not on earlier values of Y. This characterizes a Markov Chain. In the same way, the values $\{\beta^{(i)}, \tau^{(i)}\}$ of the Gibbs sampler depend explicitly only on $\{\beta^{(i-1)}, \tau^{(i-1)}\}$, and therefore this sequence forms a Markov Chain. For this reason, the sequence is often referred to as a *chain*.

We will illustrate the Gibbs sampler using artificial data. The code is based on a similar example given by Ntzoufras (2009).

```
> beta.true <- 1
> set.seed(123)
> n <- 20
> print(X <- rnorm(n))
 [1] -0.56047565 -0.23017749  1.55870831  0.07050839  0.12928774
 [6]  1.71506499  0.46091621 -1.26506123 -0.68685285 -0.44566197
[11]  1.22408180  0.35981383  0.40077145  0.11068272 -0.55584113
[16]  1.78691314  0.49785048 -1.96661716  0.70135590 -0.47279141
> print(Y <- beta.true * X + rnorm(n))
 [1] -1.62829935 -0.44815240  0.53270387 -0.65838284 -0.49575153
 [6]  0.02837168  1.29870325 -1.11168812 -1.82498979  0.80815295
[11]  1.65054602  0.06474234  1.29589711  0.98881620  0.26573995
[16]  2.47555339  1.05176813 -2.02852887  0.39539324 -0.85326241
> summary(lm(Y ~ X))
Call:
lm(formula = Y ~ X)
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -0.04017    0.19197  -0.209 0.836588
X            0.92174    0.20027   4.603 0.000221 ***
```

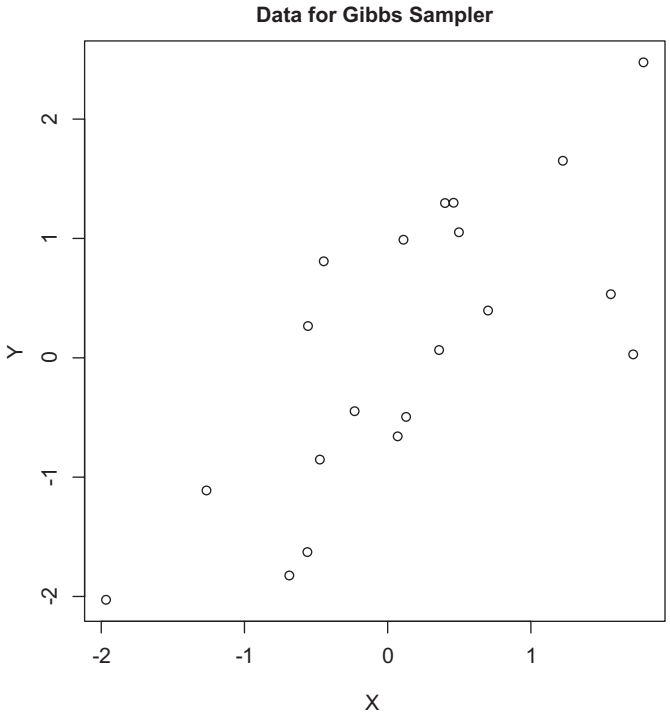

FIGURE 14.2
Scatterplot of Y versus X for the artificial data set used to illustrate the Gibbs sampler.

Figure 14.2 shows a scatterplot of X and Y. Next, we choose parameters $\widehat{\beta}$ and $\widehat{\tau}_\beta = 1/\widehat{\sigma}_\beta^2$ for the prior $p(\beta)$ and $\widehat{\mu}_\tau$ and $\widehat{\nu}_\tau$ for the prior $p(\tau)$. Recall that a noninformative prior is one that has a large variance, reflecting little or no knowledge of the values. We will use noninformative priors, with the priors for both β and τ having a variance of 100. The mean of $p(\beta)$ is set at 0 and the mean of $p(\tau)$ at 1. Recall that the gamma density has mean μ/ν and variance μ/ν^2. Therefore, if $\widehat{\mu}_\tau$ and $\widehat{\nu}_\tau$ are each assigned the value 0.01, then the prior density $p(\tau)$ has mean 1 and variance 100, which means that it is basically noninformative.

```
> beta.pri <- 0
> s2.pri <- 100
> mu.pri <- 0.01
> nu.pri <- 0.01
```

The next step is to pick initial value $\tau^{(0)}$. We want it to be small to reflect a low precision.

```
> cur.tau <- 0.01
```

Now we generate an array to hold our chain of $\beta^{(i)}$ values and carry out the Gibbs sampler algorithm. We generate 5000 iterations using a `for` loop. This could also be done using the `replicate()` function, but we use a `for` loop in order to compare this code with WinBUGS code in the next section.

```
> MCMC.data <- numeric(5000)
> for (i in 1:5000){
+ # Use  current tau to generate conditional posterior of beta
+     sb2.post <- 1/(1/s2.pri + cur.tau*sum(X^2))
+     beta.post <- sb2.post*(beta.pri/s2.pri+ cur.tau*sum(X*Y))
+ # Draw beta from the current conditional posterior
+     cur.beta <- rnorm(1, beta.post, sqrt(sb2.post))
+ # Use current beta to generate conditional posterior of tau
+     mu.post <- n + mu.pri
+     nu.post <- (nu.pri + sum((Y-cur.beta*X)^2)) / 2
+ # Draw tau from the current conditional posterior
+     cur.tau <- rgamma(1, shape = mu.post, rate = nu.post)
+     MCMC.data[i] <- c(cur.beta)
+ }
```

Now we get rid of the first 1000 iterations as the burn-in values and compute the mean of the remaining ones. It is reasonably close to the value generated by least squares regression above.

```
> burn.in <- 1000
> beta.MCMC <- MCMC.data[-(1:burn.in)]
> mean(beta.MCMC)
[1] 0.9153175
```

Figure 14.3 shows the values of $\beta^{(i)}$ for the first 200 iterations of the burn-in sequence. The values of $\beta^{(i)}$ appear to have in this example already settled into a steady sequence of random movement about the mean.

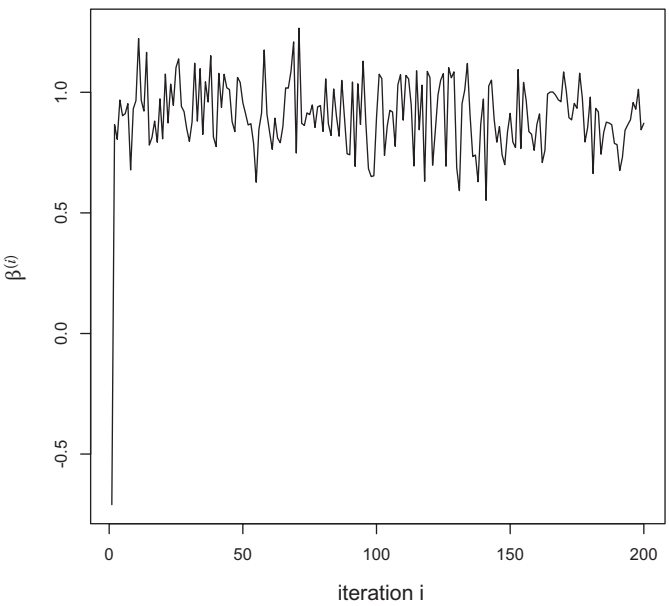

FIGURE 14.3
Plot of the first 200 iterations of the Gibbs sampler applied to the artificial data set of Figure 14.2.

Markov Chain Monte Carlo methods are not always so well behaved, and the most important phase of the analysis when using them is the diagnostic phase. This is a circumstance where most people who practice Bayesian analysis turn to a program specialized for this process. The most commonly used programs are descendents of the BUGS (for Bayesian inference Using Gibbs Sampling) program developed by statisticians at the Medical Research Council in Cambridge, England. Currently the two most powerful versions are WinBUGS (Lunn et al., 2000), which, as its name suggests, works under Microsoft Windows, and OpenBUGS (http://www.openbugs.net/w/FrontPage), developed at the University of Helsinki by a group led by Andrew Thomas. We will use the former. The next section introduces WinBUGS and describes how to access it from R.

14.3 Introduction to WinBUGS

14.3.1 WinBUGS Basics

As of the writing of this book, the most recently developed version of WinBUGS is Version 1.4.3. This will probably remain the most recent version for the foreseeable future, since, according to the website, future development efforts will all be devoted to OpenBUGS. Both programs are associated with R packages that can generate scripts from within R. WinBUGS is associated with `R2WinBUGS` (Sturtz et al., 2005) and OpenBUGS is associated with `BRugs` (Thomas et al., 2006), although OpenBUGS can also be run from `R2WinBUGS`. At present, WinBUGS is more widely used and arguably more stable, and so it is the one we will choose. WinBUGS Version 1.4.3, along with a user manual and associated data and scripts, can be downloaded from the BUGS website (https://www.mrc-bsu.cam.ac.uk/software/bugs/the-bugs-project-winbugs/). The program is free to use, and a permanent license key is provided.

Although graphical methods exist for developing WinBUGS models, we will focus on the WinBUGS programming language, which is quite similar in many respects to the R language. We will not make any attempt to provide a comprehensive introduction to WinBUGS but rather will provide just enough information to allow the reader to create, execute, and carry out diagnostic analysis of models developed in this chapter. Several sources of further information are given in Section 14.7. We begin by reanalyzing the simple linear regression model of the previous section. The initial description of the use of WinBUGS is taken from McCarthy (2007, p. 249).

In this section we will work in WinBUGS itself, and in the next section we will return to R. Once you have installed WinBUGS, open it in the usual manner by clicking on the WinBUGS icon. You will be presented with an application window. The first step is to open a new code window using *File->New*. Copy the following code from the file 14.3.1.txt in the R code for this chapter into the window.

```
model
{
beta ~ dnorm(0, 0.01)
tau ~ dgamma(0.01, 0.01)
for (i in 1:20)
  {
  Y.hat[i] <- beta * X[i]
```

```
  Y[i] ~ dnorm(Y.hat[i], tau)
  }
}
```

This code specifies the Bayesian regression model. The first two lines after the initial curly bracket specify that the parameters of the model are β and τ, that the prior density for β is normal with mean 0 and precision 0.01 (i.e., variance 100), and that the prior for τ is a gamma density with mean 1 and variance 100 (i.e., scale parameter 0.01 and rate parameter 0.01). Note that in WinBUGS one specifies in `dnorm()` the precision (the inverse of the variance) of the normal distribution, and in R one specifies in `rnorm()` the standard deviation (the square root of the variance). The statement `Y[i] ~ dnorm(Y.hat[i], tau)` says that the variable Y is normally distributed, but this statement does not actually generate variables, as would an R statement of the form `Y <- rnorm(Y.hat, sigma2)`. The rest of the `model` statement specifies the likelihood, which is the part that describes the regression model. This is specified using a `for` loop, where the first line specifies the deterministic part of the model, βX_i and the second line specifies that Y_i is drawn from a normal distribution with mean βX_i and precision τ.

Next, we input the data, which are the values of the X_i and corresponding Y_i specified in the previous section. This is done with a `list` statement.

```
list(
X=c(-0.56047565, -0.23017749,  1.55870831,  0.07050839,  0.12928774,
1.71506499,  0.46091621, -1.26506123, -0.68685285,
 -0.44566197,  1.22408180,  0.35981383,  0.40077145,  0.11068272,
-0.55584113,  1.78691314,  0.49785048, -1.96661716,
0.70135590, -0.47279141),
Y=c(-1.62829935, -0.44815240,  0.53270387, -0.65838284, -0.49575153,
0.02837168,  1.29870325, -1.11168812, -1.82498979,  0.80815295,
1.65054602,  0.06474234,  1.29589711,  0.98881620,  0.26573995,
2.47555339, 1.05176813, -2.02852887,  0.39539324,
 -0.85326241))
)
```

Notice that the `model` statement uses curly brackets and the `list` statement uses parentheses. The concatenate function `c()` is used in the same manner as in R, and the data were just copied in from the R output in Section 14.2. Finally, we must specify the values of $\beta^{(0)}$ and $\tau^{(0)}$.

```
list(beta=0, tau = 0.01)
```

This completes the WinBUGS program. There are many similarities between the WinBUGS and R programming languages, and these may serve to make the subtle differences more confusing. Let's go through the two code sequences and compare them. The first two lines in the `model` statement of the WinBUGS code are

```
beta ~ dnorm(0,0.01)
tau ~ dgamma(0.01, 0.01)
```

These establish the prior densities of β and τ. These statements involve the use of a tilde. In R, the tilde is used, for example, in functions such as `lm()` to express the form of a linear model for example with a statement like `lm(Y ~ X, data = field.data)`. In WinBUGS, the tilde is used to establish the distribution to which a random variable

belongs (cf. expressions like [Equation 14.5]), so that for example the statement `beta ~ dnorm(0,0.01)` indicates that the prior distribution of β is normal with mean $\hat{\beta} = 0$ and precision $\hat{\tau} = 1/\hat{\sigma}^2 = 0.01$. In the R code of Section 14.2, we did not explicitly specify the form of the priors; rather we simply set their parameter values.

```
> s2.pri <- 100
> beta.pri <- 0
> mu.pri <- 0.01
> nu.pri <- 0.01
```

The next set of code in the WinBUGS `model` statement above is a `for` loop.

```
for (i in 1:20)
  {
  Y.hat[i] <- beta * X[i]
  Y[i] ~ dnorm(Y.hat[i], tau)
  }
```

This establishes the relationship between X and Y. WinBUGS does not have the facility to vectorize assignment statements. In R, which does have this ability, the first assignment would be done with a simple statement like `Y.hat <- beta * X`. Note that the second statement in the WinBUGS code establishes for each Y_i the regression relationship $Y \sim N(\beta X_i, 1/\tau)$, which is equivalent to $Y_i = \beta X_i + \varepsilon_i$, $\varepsilon_i \sim N(0, 1/\tau)$. Note also that in the WinBUGS code the values of the `X[i]` have not yet been set. In R, this would generate an error, since R executes code line by line. WinBUGS code is compiled, however, so it is only executed when all statements have been loaded. That completes the WinBUGS `model` statement. In R we had to explicitly specify the form of the likelihood and the Gibbs sampler algorithm, but in WinBUGS this is handled for us.

To execute the model, do the following.

1. Open the Specification window by selecting *Model->Specification*.
2. Highlight the word `model` in the model description window (the window in which you did the typing) by double clicking on it.
3. Click on *check model* in the Specification window. The phrase "model is syntactically correct" should appear in the lower left corner of the WinBUGS widow, and the *load data* and *compile* buttons in the Specification window should no longer be grayed out.
4. Highlight the first word `list` and click on the *load data* button. The phrase *data loaded* should appear in the lower left corner of the WinBUGS window.
5. Click on the *compile* button in the Model Specification window. The phrase *model compiled* should appear in the lower left corner of the WinBUGS window and the two *inits* buttons should no longer be grayed out.
6. Highlight the second word `list` and click on the *load inits* button. The phrase *model initialized* should appear in the lower left corner of the WinBUGS window.

Now we are ready to run the model.

7. Select *Model->Update* and *Inference->Samples*. These two windows will pop up right on top of the Model Specification window, so drag them out of each other's way.

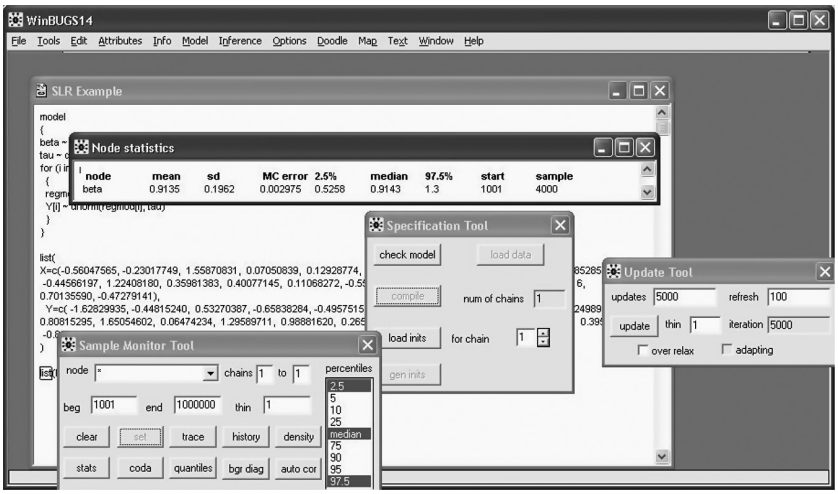

FIGURE 14.4
WinBUGS window after completion of the fitting of the regression model to the artificial data set.

8. In the Update window, change *updates* from 1000 to 5000. In the Sample Monitor window, change *beg* from 1 to 1001. This establishes the first 1000 iterations as the burn-in.
9. In the *node* box of the Sample Monitor window, type *beta* and click on *set*.
10. In the Update window, click on *update*. This runs the model.
11. In the *node* box of the Sample Monitor window, type a "*" and click on *stats*. Your screen should now resemble Figure 14.4. The estimated value of β is 0.9135, about the same as the values we have obtained previously.

In this case things look pretty good, but unfortunately in Bayesian inference things can sometimes look pretty good and not be so good at all. The most important phase of a Bayesian analysis is the diagnostic phase, which is the topic of the next subsection.

14.3.2 WinBUGS Diagnostics

Our principal use of WinBUGS diagnostics will be to determine that the MCMC method has produced a convergent sequence like that of Figure 14.3 so that the integral in Equation 14.20 is accurately estimated. The initial checks are graphical and are accomplished with the *Sample Monitor* window. Assuming that the WinBUGS run from the previous section is still available, from this window click on the *history*, *density*, *quantiles*, and *auto cor* buttons. You should see something like Figure 14.5, in which the other windows have been closed to make room (don't close your windows yet). The history window, labeled *Time series*, shows the values of $\beta^{(i)}$ from the end of the burn-in to the end of the chain. The sequence should look like patternless noise around a constant mean, as it does in Figure 14.5. In particular, there should be no trend and no pattern in the variation. The density window, labeled *Kernel density*, shows a histogram of the $\beta^{(0)}$ values. It should look smooth and should resemble the appropriate density, if the form of this is known. In the present case, it looks like a normal density (more or less). The quantiles window, labeled *Running quantiles*, shows an approximation of the moving quantiles of the sequence. The quantiles should be fairly constant. The *auto cor* window, labeled *Autocorrelation function*, shows the

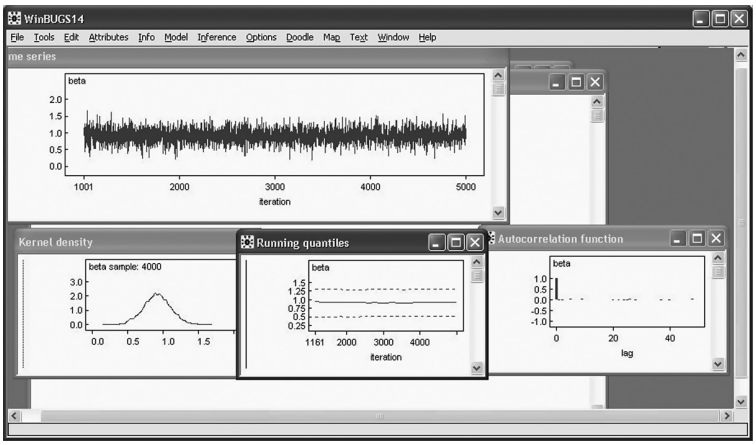

FIGURE 14.5
WinBUGS diagnostic windows.

autocorrelation value as a function of the lag. Since the sequence $\beta^{(i)}$ is an autoregressive time series, values of $\beta^{(i)}$ are correlated with values of $\beta^{(i-1)}$, $\beta^{(i-2)}$, and so forth. In the case of the model shown in Figure 14.5, this autocorrelation dies off quickly, but sometimes it does not. It is desirable to use an uncorrelated sequence of values in calculations such as Equation 14.20. If the *auto cor* window indicates a substantial correlation at a particular lag, the appropriate procedure is to run the model with the *thinning* parameter, which represents the number of values of the time series to skip in assembling the sequence used in the calculations, to a value just greater than the autocorrelation lag. This is set using the *thin* button in the *update* window. While the chain is still in memory, click the *coda* button. This will open two windows, an *index* window and a *chain* window. Copy the data from these windows into a text file and save the two files. We will use them shortly.

The button denoted *brg diag* in the Sample Monitor window allows the user to carry out a convergence check using the Gelman-Rubin diagnostic (Gelman and Rudin, 1992). This basically carries out an analysis of variance on the data (Ntzoufras, 2009, p. 143), and in order to employ it one must generate several chains simultaneously. Start WinBUGS and again go through the steps 1 through 11 given above, with the following changes.

1. Prior to clicking on *check model* in the Specification window, change the number in the *num of chains* window to 5.
2. Click on the *gen inits* button. This generates a random set of initial values for each of the five chains.

After the run, click the *brg diag* button in the Sample Monitor window. A plot similar to that in the upper left corner of Figure 14.6 should appear. This plot shows the variance ratio and convergence statistics for the run, which should both converge to 1. The actual numerical values displayed graphically can be obtained as follows. First double click on the graph window, and then hold down the *Ctrl* key and left click again. This brings up the window shown on the right in Figure 14.6. There are other diagnostics that we can run, but we will defer a discussion of these until the next section. This completes our introduction of WinBUGS. We now turn to a discussion of the package `R2WinBUGS`, which permits the user to execute WinBUGS from within R.

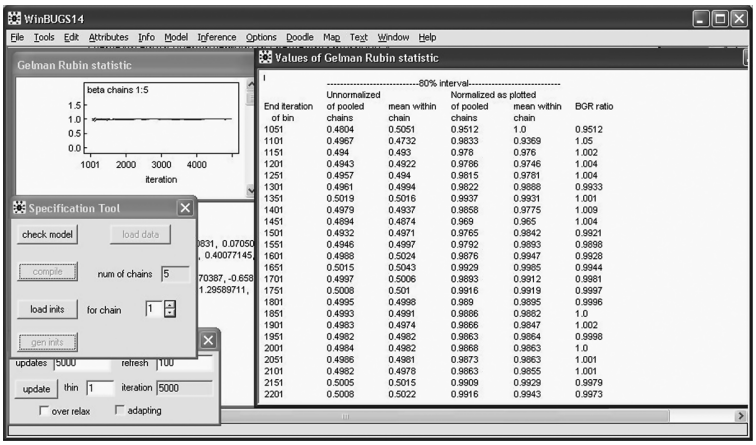

FIGURE 14.6
WinBUGS convergence statistics.

14.3.3 Introduction to R2WinBUGS

The `R2WinBUGS` package does not eliminate the need to know how to construct a WinBUGS model, but it does make it very easy to execute the model from within R. We will demonstrate the use of the package by continuing with our simple example. The first part of the program is identical to the R program of Section 14.2.

```
> library(R2WinBUGS)
> beta.true <- 1
> set.seed(123)
> n <- 20
> X <- rnorm(n)
> Y <- beta.true * X + rnorm(n)
```

The next step is to specify the model. This is done by writing an R function that contains within it the WinBUGS code.

```
> demo.model <- function(){
+ beta ~ dnorm(0, 0.01)
+ tau ~ dgamma(0.01, 0.01)
+ for (i in 1:n)
+   {
+   Y.hat[i] <- beta * X[i]
+   Y[i] ~ dnorm(Y.hat[i], tau)
+   }
+ }
```

Notice that this is the same code (literally) as is in the WinBUGS program of Section 14.3.2. It is *not* R code. Next, we use the function `write.model()` to write this model to disk. We will use the character string `mybugsdir` to contain the directory where this file is written. You need to establish this directory on your disk. Suppose you have set up the folders (i.e., directories), as described in Section 2.1.2, and you wanted to use a subfolder of your folder *C:\rdata\SDA2* called *WinBUGS* to house your WinBUGS files. First, you would use the Windows explorer to create the folder, and then you would enter the following statement.

```
> mybugsdir <- "c:\\rdata\\SDA2\\WinBUGS\\"
```

Now we are ready to write the model file.

```
> write.model(demo.model,
+     paste(mybugsdir,"demomodel.bug", sep = ""))
```

The first diversion from the R program of Section 14.2 is the manner in which the data and the inits are specified.

```
> XY.data <- list(X = X, Y = Y, n = n)
> XY.inits <- function(){
+     list(beta0 = rnorm(1), beta1 = rnorm(1), tau = exp(rnorm(1)))
+}
```

The data are specified by creating a list that contains the names of the data vectors in the model on the left side of the equals sign and the source of the data on the right side. The inits are specified by a function that tells how to generate the initial values. We want the values of `tau` to be positive, so we make its distribution lognormal. The use of a function to generate the inits corresponds roughly to the use of the *gen inits* button in WinBUGS as discussed in Section 14.3.2, and is necessary if there is to be more than one chain. There is some discussion in the literature of the merits of one long chain versus several short ones, but there are some important advantages to the use of more than one chain (the advantages to be described below), and we will always use more than one.

Now we call the function `bugs()`, which actually runs the model. The first time we run it, we do so as a debugging test by setting the argument `debug = TRUE` and by specifying a very small number of iterations with the value of the argument `n.iter`. The argument `debug` has the default value `FALSE`. When `debug` is `FALSE`, if there is a problem in WinBUGS, the error messages will flash by on the screen too quickly for you to read, and then the program will close. When `debug` is set to `TRUE`, WinBUGS does not close, and therefore you can read the error messages. Just make sure that once the program is running correctly, you remove the assignment `demo = TRUE`. Otherwise, when the program is finished, you and the computer will patiently wait for each other to make the next move, and the computer is much more patient than you are.

```
> demo.test <- bugs(data = XY.data, inits = XY.inits,
+     model.file = paste(mybugsdir,"demomodel.bug", sep = ""),
+     parameters = c("beta0", "beta1", "tau"), n.chains = 1,
+     n.iter = 10, n.burnin = 2, debug = TRUE,
+     bugs.directory=mybugsdir)
```

Assuming this runs in WinBUGS, hit the ESC key while in the RStudio Console to return to R.

The next step is a full run including diagnostics. Diagnostic evaluation in R is carried out using the `coda` (Plummer et al., 2011) or `BOA` (Smith, 2007) package. We will use `coda`. Use the assignment `codaPkg = TRUE` to be able to view the coda diagnostics. We can remove the `debug = TRUE` assignment and specify `nchains = 5`, `n.iter = 5000`, and `n.burnin = 1000`.

```
> library(coda)
> demo.coda <- bugs(data = XY.data, inits = XY.inits,
+     model.file = paste(mybugsdir,"demomodel.bug", sep = ""),
+     parameters = c("beta0", "beta1", "tau"), n.chains = 1,
```

```
+     n.iter = 5000, n.burnin = 1000, codaPkg = TRUE,
+     bugs.directory=mybugsdir)
```

This time you won't see anything in the WinBUGS screen because it is written to a file. Use `read.bugs()` to read the coda file and then type `codamenu()`. Your screen should look like this.

```
> library(coda)
> demo.mcmc <- read.bugs(coda.file)
> codamenu()
CODA startup menu

1: Read BUGS output files
2: Use an mcmc object
3: Quit

Selection:
```

The `coda` package is menu driven. The first step is to load the output from the WinBUGS files. Type 2. Then, when prompted for the name of the saved object, type it in as shown.

```
Enter name of saved object (or type "exit" to quit)
1:demo.mcmc
Checking effective sample size...OK
CODA Main Menu
1: Output Analysis
2: Diagnostics
3: List/Change Options
4: Quit
Selection:
```

Type 1 to access the output analysis.

```
CODA Output Analysis menu
1: Plots
2: Statistics
3: List/Change Options
4: Return to Main Menu
Selection: 1
```

To access the plots, type 1. This produces the plots shown in Figure 14.7. Now type 4 to return to the main menu, and select choice 2 from the menu. This leads to the following.

```
CODA Diagnostics Menu

1: Geweke
2: Gelman and Rubin
3: Raftery and Lewis
4: Heidelberger and Welch
5: Autocorrelations
6: Cross-Correlations
7: List/Change Options
8: Return to Main Menu

Selection:
```

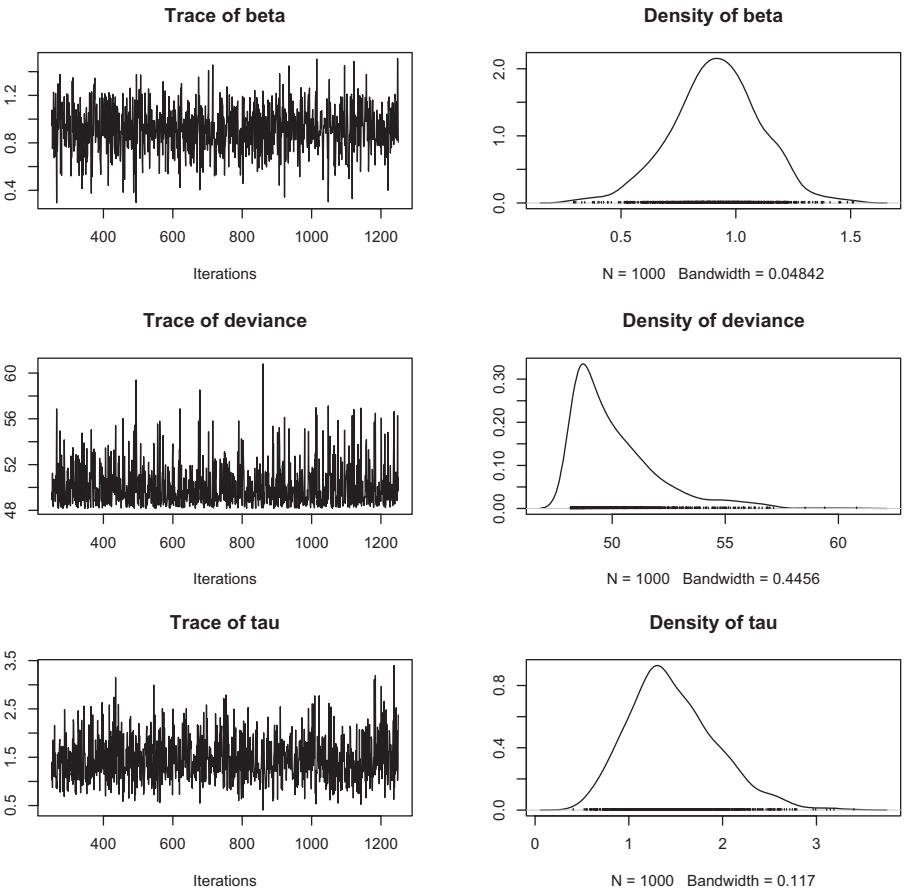

FIGURE 14.7
Diagnostic plots generated by `coda` for the artificial data set.

We have already discussed the Gelman-Rubin diagnostic as implemented in WinBUGS, so here we will briefly describe the diagnostics of Geweke (1992), Raftery and Lewis (1992), and Heidelberger and Welch (1992). The year 1992 was obviously a very good one for MCMC diagnostics. The Gewecke test generates a set of z scores based on subsamples of the elements of the chain; these z scores should fall between −2 and 2. The Raftery and Lewis diagnostic generates a statistic denoted I that measures the relative effect on the effective sample size due to autocorrelation. The value of I may be taken as an estimate of the appropriate thinning interval. The diagnostic test of Heidelberger and Welch checks subsets of the MCMC chain for stationarity. These various tests each focus on a different aspect of the chain, and a failure of any one of them should be taken as an indication of non-convergence. Further information can be found in Ntzoufras (2009) as well as other sources listed in Section 14.7.

The chains all appear stable, and the diagnostics are within the acceptable range, so we can exit the diagnostics. Next, run the simulation by removing the assignment `codaPkg = TRUE`. First, we will run the model with one chain.

```
> demo.sim <- bugs(data = XY.data, inits = XY.inits,
+     model.file = paste(mybugsdir,"demomodel.bug", sep = ""),
+     parameters = c("beta0", "beta1", "tau"), n.chains = 1,
```

```
+    n.iter = 5000, n.burnin = 1000,
+    bugs.directory = mybugsdir)
```

This runs the model, as you will see on your screen. The results can be displayed by the functions `print()` and `plot()`. We will only display the former.

```
> print(demo.sim, digits = 3)
      *       *       *
n.sims = 1000 iterations saved
           mean    sd   2.5%    25%    50%    75%  97.5%
beta      0.915 0.192  0.528  0.796  0.920  1.039  1.263
tau       1.469 0.464  0.698  1.156  1.407  1.745  2.524
deviance 50.102 1.902 48.200 48.690 49.530 50.932 55.453

DIC info (using the rule, pD = Dbar-Dhat)
pD = 1.9 and DIC = 52.0
DIC is an estimate of expected predictive error (lower deviance is
better).
```

For a discussion of the DIC (deviance information criterion), see Gelman and Hill (2007, p. 525). Now we will run the model with five chains.

```
> demo.sim <- bugs(data = XY.data, inits = XY.inits,
+ model.file = paste(mybugsdir,"demomodel.bug", sep = ""),
+ parameters = c("beta", "tau"), n.chains = 5,
+ n.iter = 5000, n.burnin = 1000,
+ bugs.directory=mybugsdir)
> print(demo.sim, digits = 3)
      *       *       *
           mean    sd   2.5%    25%    50%    75%  97.5%  Rhat n.eff
beta      0.920 0.198  0.520  0.789  0.920  1.043  1.305 1.002  1000
tau       1.450 0.466  0.696  1.107  1.391  1.741  2.497 1.002  1000
deviance 50.177 1.994 48.210 48.720 49.645 51.045 55.346 1.006   440

For each parameter, n.eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor (at convergence,
Rhat=1).
DIC info (using the rule, pD = Dbar-Dhat)
pD = 2.0 and DIC = 52.2
DIC is an estimate of expected predictive error (lower deviance is
better).
```

Note that the output from the second run includes two diagnostic quantities, `Rhat` and `n.eff`. These can only be computed when there is more than one chain. The former is a convergence diagnostic and is approximately equal to the square root of the variance of the mixture of all of the chains, divided by the average within-chain variance (Gelman and Hill, 2007, p. 358). As it says in the output, the fact that `Rhat` is approximately 1 indicates convergence. The second diagnostic quantity, `n.eff`, is the "effective number of simulation draws." This represents the extent to which autocorrelation in the Markov Chain has affected the assumption of independent draws. Gelman and Hill (2007, p. 358) recommend that `n.eff` be at least 100.

The extension of WinBUGS analysis to multiple linear regression is straightforward. We will demonstrate it using a simplified linear regression model for yield as a function of only soil clay content and soil potassium content for Field 1 of Data Set 4. First, we normalize all of the

data. For example, we use a normalized yield *YieldN* = *Yield* / max(*Yield*), and both *Clay* and *SoilK* are similarly normalized. There are two reasons for this. The first is to keep numerical problems to a minimum by avoiding very large or very small numbers. The second is to avoid inadvertently specifying priors that are not noninformative (Gelman and Hill, 2007, p. 355). Since the values of *Yield* are measured in the thousands and the values of *Clay* and *SoilK* are in one or two digits, a regression model involving *Yield*, *Clay*, and *SoilK* could have coefficients in the tens or hundreds. Other models might have coefficients with even larger magnitude. It would be easy to specify a prior with a mean of 0 and an insufficiently large variance and inadvertently (and inappropriately) pull the value of the coefficients toward 0.

```
> data.Set4.1$YieldN <- with(yield.pts, Yield / max(Yield))
> data.Set4.1$ClayN <- with(data.Set4.1, Clay / max(Clay))
> data.Set4.1$SoilKN <- with(data.Set4.1, SoilK / max(SoilK))
```

Debugging a WinBUGS model can be a maddening experience, so it is always best to move in the smallest possible steps from a model that works to the model that one is trying to construct. In this case, I started with the simple `demo.model` from this section. The first step was to delete the line

```
> n <- 20
```

and change the line

```
> XY.data <- list(X = X, Y = Y, n = n)
```

to

```
> XY.data <- list(X = X, Y = Y, n = nrow(data.Set4.1))
```

The next step was to change the line

```
> X <- rnorm(n)
> Y <- X + rnorm(n)
```

to

```
> X <- data.Set4.1$ClayN
> Y <- data.Set4.1$YieldN
```

In this way, one baby step at a time, I first created a model of *YieldN* regressed only on *ClayN* and then expanded it to include *SoilKN*. Here is the full resulting WinBUGS program.

```
> Set4.1.model <- function(){
+ beta0 ~ dnorm(0, 0.01)
+ beta1 ~ dnorm(0, 0.01)
+ beta2 ~ dnorm(0, 0.01)
+ tau ~ dgamma(0.01, 0.01)
+ for (i in 1:n)
+   {
+   Y.hat[i] <- beta0 + beta1 * Clay[i] + beta2 * SoilK[i]
+   Yield[i] ~ dnorm(Y.hat[i], tau)
+   }
```

```
+ }
>
> write.model(Set4.1.model,
+    paste(mybugsdir,"\\Set4.1model.bug", sep = ""))
>
> XY.data <- list(Clay = data.Set4.1$ClayN, SoilK = data.Set4.1$SoilKN,
+    Yield = data.Set4.1$YieldN, n = nrow(data.Set4.1))
> XY.inits <- function(){
+    list(beta0 = rnorm(1), beta1 = rnorm(1), beta2 = rnorm(1),
+    tau = exp(rnorm(1)))
+ }
```

Next, I went through the test and coda steps. Reading the coda file gives us something we have not seen before.

```
> codamenu()
CODA startup menu
1: Read BUGS output files
2: Use an mcmc object
3: Quit
Selection: 2
Enter name of saved object (or type "exit" to quit)
1:Set4.1.mcmc
Checking effective sample size...
****************************************
WARNING !!!
Some variables have an effective sample size of less than 200 in at least
one chain. This is too small, and may cause errors in the diagnostic
tests. HINT: Look at plots first to identify variables with slow mixing.
(Choose menu Output Analysis then Plots). Re-run your chain with a larger
sample size and thinning interval. If possible, reparameterize your model
to improve mixing
****************************************
CODA Main Menu
```

When it loads the file, the function `codamenu()` runs a test of the Gelman and Rubin statistic described in the previous section and flags values below an acceptable threshold. The term *mixing* refers to the selection of random samples from the full posterior distribution by the MCMC algorithm (Congdon, 2010, p. 21). The warning suggests that we increase the sample size and the thinning interval, so we will try that. After the increase, we still get the same message, but the plots of the chains look pretty good. Let's try the simulation steps. Here is the output of the simulation.

```
> Set4.1.sim <- bugs(data = XY.data, inits = XY.inits,
+    model.file = paste(mybugsdir,"\\Set4.1model.bug", sep = ""),
+    parameters = c("beta0", "beta1", "beta2", "tau"), n.chains = 5,
+    n.iter = 10000, n.burnin = 2000,
+    bugs.directory = mybugsdir)
> print(Set4.1.sim, digits = 3)
Inference for Bugs model at "c:\aardata\book\Winbugs\\Set4.1model.bug",
fit using WinBUGS,
 5 chains, each with 10000 iterations (first 1000 discarded), n.thin = 5
 n.sims = 9000 iterations saved
```

```
         mean     sd    2.5%   25%     50%    75%  97.5%  Rhat n.eff
beta0    1.362  0.092   1.181  1.300  1.362  1.422  1.543 1.002  3700
beta1   -1.216  0.157  -1.533 -1.319 -1.215 -1.115 -0.905 1.006   670
beta2    0.036  0.143  -0.241 -0.060  0.036  0.131  0.318 1.004   910
tau     59.022 8.959  42.610 52.850 58.610 64.890 77.591 1.001  9000
```

Your values may be slightly different. The values of the diagnostics `Rhat` and `n.eff` look pretty good. We can compare the output with the model computed using `lm()`.

```
> Set4.1.lm <- lm(YieldN ~ ClayN + SoilKN, data = data.Set4.1)
> coef(Set4.1.lm)
(Intercept)          ClayN         SoilKN
 1.364794826 -1.199936005  0.005542428
>
> t(Set4.1.sim$mean)
     beta0    beta1    beta2       tau      deviance
[1,] 1.364603 -1.19636 0.002188471 59.15018 -107.0198
```

Remember that `tau` is the inverse of the error variance. Here is a computation of the inverse of the error variance from `lm()`.

```
> 1 / (sum(residuals(Set4.1.lm)^2) / (nrow(data.Set4.1) - 3))
[1] 60.03328
```

All of the values are reasonably close. My own experience indicates that, although it is useful, the warning message produced by `codamenu()` often seems quite conservative. I generally put more weight in the chain plots and the statistics `Rhat` and `n.eff` and, when possible, a comparison of the results with another method. So far, however, we have not seen any particular advantage that the Bayesian approach provides over the classical approach. The greatest advantages occur in the analysis of more complex models, so we will not pursue ordinary linear regression any further. In the next section, we continue to build our Bayesian toolbox by considering generalized linear models.

14.3.4 Generalized Linear Models in WinBUGS

Once you know how to analyze a multiple linear regression model using WinBUGS, the extension to generalized linear models is in most cases fairly straightforward. We will describe it by continuing our analysis of the logistic regression model for presence/absence of blue oaks (Data Set 2). Recall that the logistic regression model is specified by (Section 8.3.1)

$$E\{Y \mid X_i\} = \pi_i,$$
$$\ln\left(\frac{\pi_i}{1-\pi_i}\right) = \beta_0 + \sum_{j=1}^{p-1} \beta_j X_{ij}, \tag{14.21}$$

where Y is a binomially distributed random variable taking on values 0 and 1.

In Section 8.4, we developed a logistic regression model for the relationship between blue oak presence/absence and the explanatory variables in the Sierra Nevada subset of Data Set 2. We will use the `sf` object `data.Set2S` created in that section and develop a Bayesian model for the data. As in the analysis of Section 14.3.2, we will normalize the

variable *Elevation*. The data are loaded into R as spatial features objects and can be used in this form.

```
> data.Set2S$ElevN <- data.Set2S$Elevation / max(data.Set2S$Elevation)
```

The logistic regression model and the coefficients are as follows:

```
> glm.demo <- glm(QUDO ~ ElevN,
+   data = data.Set2S, family = binomial)
> coef(glm.demo)
(Intercept)       ElevN
   3.329145  -11.463550
```

Now let's set up the model to pass to WinBUGS. The logistic regression form of the generalized linear model uses a normal density prior for the values of the coefficients β_i (Ntzoufras, 2009, p. 255). With binomial values of Y_i (i.e., $Y_i = 1$ implies presence and $Y_i = 0$ implies absence), the variance of the Y_i is the binomial variance, which is equal to $np(1-p)$, so there is no precision term τ as there is in the linear model. The code for the `model` statement is the following:

```
> library(R2WinBUGS)
> demo.model <- function(){
+ beta0 ~ dnorm(0, 0.01)
+ beta1 ~ dnorm(0, 0.01)
+ for (i in 1:n)
+   {
+   logit(p.QUDO[i]) <- beta0 + beta1 * ElevN[i]
+   QUDO[i] ~ dbin(p.bound[i], 1)
+   p.bound[i] <- max(0, min(1, p.QUDO[i]))
+   }
+ }
```

The first line inside the `for` loop sets up the model using the WinBUGS function `logit()` to create a logistic regression model. The last two lines express the fact that `QUDO` is a binomially distributed quantity. The variable `p.bound` is used instead of `p.QUDO` in order to avoid numerical problems in case a computed value of `p.QUDO` is less than 0 or greater than 1. Remember that the second line in the `for` loop is not an assignment statement but rather a statement of the relationship between the variable `p.bound` in the model and the data values in the vector `QUDO`.

The next step is to provide WinBUGS with the data values and to initialize the model.

```
> XY.data <- list(QUDO = data.Set2S$QUDO,
+   ElevN = data.Set2S$ElevN, n = nrow(data.Set2S))
> XY.inits <- list(list(beta0 = 0, beta1 = 0))
```

Next, we write a file providing information to WinBUGS.

```
> write.model(demo.model,
+    paste(mybugsdir,"\\demoglm.bug", sep = ""))
```

Before we continue, we must change the `XY.inits`. Since there will be five chains, we must have five initial sets of values. This is similar to the change made in the second run of WinBUGS in Section 14.3.1.

```
> XY.inits <- function(){
+    list(beta0 = rnorm(1), beta1 = rnorm(1))
+ }
```

We then go through the usual `debug` and `coda` steps, with `codamenu()` giving us the same warning as in the last section. Next, then we execute the model.

```
> demo.sim <- bugs(data = XY.data, inits = XY.inits,
+    model.file = paste(mybugsdir,"\\demoglm.bug", sep = ""),
+    parameters = c("beta0", "beta1"), n.chains = 5,
+    n.iter = 5000, n.burnin = 1000,
+    bugs.directory = mybugsdir)
```

Let's compare the Bayesian results with those of the function `glm()`.

```
> t(demo.sim$mean)
     beta0    beta1     deviance
[1,] 3.321083 -11.44447 1368.549
```

The values are quite close to those displayed above. In Exercise 14.4, you are asked to develop a WinBUGS version of the model selected in Section 8.4.2 to represent the candidates for the Sierra Nevada subset. My results of the `glm()` model and the WinBUGS model are the following (yours may vary slightly).

```
> coef(glm. Set2S)
  (Intercept)         Precip            MAT         SolRad
-6.3122558455 -0.0063894184  0.6026153179  0.0006361065
       AWCAvg        Permeab
-0.0066932919 -0.3157297346
> t(Set2S.sim$mean)
     beta0     betaPre      betaMAT  betaSol
[1,] -4.029703 -0.006347872 0.537417 0.0004254458
     betaAWC      betaPerm   deviance
[1,] -0.006252348 -0.3136346 1071.546
```

Again, the values of the β_i are reasonably close, with the greatest difference from the `glm()` output being between the coefficients for *MAT* and *SolRad*.

We are finally ready to tackle the models that take greatest advantage of what the Bayesian approach has to offer. These are hierarchical models, which are described in the next section.

14.4 Hierarchical Models

The mixed-model theory discussed in Chapter 12 deals with *grouped data*. These are data in which each data record is associated with a grouping factor, and data records associated with the same grouping factor are considered more closely related to each other than to data records associated with a different grouping factor. In the case of spatial data, the grouping factor is generally defined by geography. For example, in the case of Data Set 3 the grouping factor is *Field*. The grouping factor is considered to be a random effect. The mixed model contains a mixture of fixed and random effects, where a fixed effect is one in which

only the values of the effect actually present in the data are considered, and there is no attempt to extend the results of the analysis to other values. A random effect is…well, that is a problem. Statistics is a discipline that insists on precise definitions, but the definition of a random effect is remarkably vague and inconsistent in the literature (Gelman and Hill, 2007, p. 245). Some authors focus on the idea of a random effect as being defined by a probability distribution. This is, however, often not very realistic. Other authors emphasize the idea that a random effect is one for which the study is interested in values beyond those included in the observed data. The difficulty is that this implies that if the investigators change their interest, then the nature of the variable could change. It is not clear that this is desirable.

Another problem is that the one thing upon which almost all authors agree is that the values of a random effect should be randomly selected. In the case of ecological or agronomic data, however, this is usually not even remotely close to the truth. For example, consider the fields in Data Set 3. These should have been selected at random from the population of fields in northeastern Uruguay. In fact, as is true of virtually every such study, the selection of the fields was based on a combination of personal relationships, convenience, and availability. Researchers tend to work with cooperating growers with whom they have a relationship, the fields must be accessible and situated in a way that permits all of them to be visited in the allotted time, and the grower, on whose cooperation the study depends, may have some reason to prefer the use of one field over another. As with all such nonrandom selection, this opens up the possibility of bias, but more generally it calls into question the appropriateness of the assumption of randomness.

In Bayesian theory, the parameters as well as the data are considered random variables, and the probability distribution represents subjective belief about their value rather than some distribution occurring in nature. This allows one to avoid many of the definitional problems associated with random effects. For this reason, Bayesian theory does not generally use the term *mixed model*; instead, the term *hierarchical model* is used. We will introduce the concept of the hierarchical model using the same example as was used in Chapter 12 to introduce the mixed model: the relationship between irrigation effectiveness *Irrig* and *Yield* in Data Set 3.

We will start with the pooled data from all the fields. We found in Chapter 12 that all of the data records in Field 2 have a value of *Irrig* equal to 5 (the highest level), so that a regression of *Yield* on *Irrig* involving only this field would not work. We therefore remove Field 2 from the data set (however, see Exercise 14.5).

```
> data.Set3a <- data.Set3[-which(data.Set3$Field == 2),]
```

Once again, in order to avoid potential numerical problems we normalize *Yield*.

```
> data.Set3a$YieldN <- data.Set3a$Yield / max(data.Set3a$Yield)
```

Since *Irrig* ranges from one to five anyway, we will not normalize it. Here is the pooled model WinBUGS code for use with R2WinBUGS. It is derived by modification of the code in Section 14.3.4.

```
> Set3.model <- function(){
+ beta0 ~ dnorm(0, 0.01)
+ beta1 ~ dnorm(0, 0.01)
+ tau ~ dgamma(0.01, 0.01)
+ for (i in 1:n)
+   {
+   Y.hat[i] <- beta0 + beta1*Irrig[i]
```

```
+   Yield[i] ~ dnorm(Y.hat[i], tau)
+   }
+ }
> write.model(Set3.model,
+    paste(mybugsdir,"\\Set3model.bug", sep = ""))
> XY.data <- list(Irrig = data.Set3a$Irrig, Yield = data.Set3a$YieldN,
+    n = nrow(data.Set3a))
> XY.inits <- function(){
+    list(beta0 = rnorm(1), beta1 = rnorm(1), tau = exp(rnorm(1)))}
```

We now run through the usual debug, coda, simulation sequence. The coda analysis gives us the same warning message is in the last section, which we will ignore. Here are the results of the simulation.

```
> Set3.sim <- bugs(data = XY.data, inits = XY.inits,
+    model.file = paste(mybugsdir,"\\Set3model.bug", sep = ""),
+    parameters = c("beta0", "beta1", "tau"), n.chains = 5,
+    n.iter = 5000, n.burnin = 500,
+    bugs.directory = mybugsdir)
> print(Set3.sim, digits = 2)
Inference for Bugs model at "c:\aardata\book\Winbugs\\Set3model.bug", fit
using WinBUGS,
 5 chains, each with 5000 iterations (first 500 discarded), n.thin = 22
 n.sims = 1025 iterations saved
           mean    sd    2.5%     25%     50%     75%   97.5% Rhat n.eff
beta0     0.203 0.030   0.145   0.183   0.203   0.221   0.263 1.002  980
beta1     0.114 0.008   0.098   0.109   0.114   0.119   0.130 1.001 1000
tau      79.680 6.288 67.898  75.410  79.240  83.900  91.572 0.999 1000
deviance -482.2 2.40 -484.90 -484.00 -482.90 -481.30 -475.86 0.999 1000
```

As a check, we will print the coefficients of the ordinary least square (OLS) model.

```
> print(coef(lm(YieldN ~ Irrig, data = data.Set3a)),
+    digits = 3)
(Intercept)       Irrig
      0.204       0.114
```

Everything looks reasonably good: the model converges and the number of effective simulation draws is good (sometimes by random chance I get results that are not quite this good). We will therefore construct our first hierarchical model. The process is so effortless (except for the debugging) that you may find yourself asking whether this is indeed all there is to it. The answer is basically yes, at least at the operational level. At the level of interpretation, the issues are a bit subtler and will be discussed in a small way in Section 14.6.

In Section 12.3, we began by introducing a term α_i that represented the random overall effect of the field. We will do the same thing here. There is one little complication that must be addressed. We eliminated Field 2 earlier, so we now have values of the variable *Field* consisting of 1 and 3 through 16. In Chapter 12, when we were dealing with mixed models, the variable *Field* was considered a factor (i.e., a nominal quantity), so the missing value of 2 was not important. In the WinBUGS model, the easiest way to proceed is to use *Field* as an index quantity, but we will get an error message if we ignore the missing value of 2 (try it!). Therefore, we change the *Field* value of Field 16 from 16 to 2.

```
> data.Set3a$Field[which(data.Set3a$Field == 16)] <- 2
```

Now we are ready to proceed. To create a hierarchical model analogous to the mixed model of Equation 12.6 in which the intercept is allowed to depend on the group, we simply specify that the intercept β_0 be allowed to depend on the group (i.e., the field). We introduce a `for` loop defining the prior distribution of the β_{0i} and introduce two new parameters μ_0 and τ_0 that characterize their priors. In a hierarchical model, the parameters without subscripts, which are defined over all groups, are called *hyperparameters* (Gelman and Hill, 2007, p. 258), although this term is also sometimes used to characterize all the parameters associated with the distributions.

```
> Set3.model2 <- function(){
# For loop defining the beta0
+ for (i in 1:n.F){
+    beta0[i] ~ dnorm(mu.0, tau.0)
+ }
# Parameters for the priors
+ mu.0 ~ dnorm(0, 0.001)
+ tau.0 ~ dgamma(0.01, 0.01)
+ beta1 ~ dnorm(0, 0.001)
+ tau ~ dgamma(0.01, 0.01)
+ for (i in 1:n)
+   {
+   Y.hat[i] <- beta0[Field[i]] + beta1*Irrig[i]
+   Yield[i] ~ dnorm(Y.hat[i], tau)
+   }
+ }
```

The `write.model()` statement has the same form as before and is not shown. The modifications to the data and inits statements are straightforward.

```
> XY.data2 <- list(Irrig = data.Set3a$Irrig, Yield = data.Set3a$YieldN,
+    n = nrow(data.Set3a), n.F = length(unique(data.Set3a$Field)),
+    Field = data.Set3a$Field)
> n.F <- length(unique(data.Set3a$Field))
> XY.inits2 <- function(){
+    list(beta0 = rnorm(n.F), beta1 = rnorm(1), tau = exp(rnorm(1)),
+    mu.0 = rnorm(1), tau.0 = exp(rnorm(1)))}
```

We again go through the debug, coda, and simulate steps. Here are the results of the simulation step.

```
> Set3.sim2 <- bugs(data = XY.data2, inits = XY.inits2,
+    model.file = paste(mybugsdir,"\\Set3model.bug", sep = ""),
+    parameters = c("beta0", "beta1", "tau", "mu.0", "tau.0"),
+    n.chains = 5,
+    n.iter = 10000, n.burnin = 2000, n.thin = 20,
+    bugs.directory = mybugsdir)
> print(Set3.sim, digits = 2)
Inference for Bugs model at "c:\aardata\book\Winbugs\\Set3model.bug", fit
using WinBUGS,
 5 chains, each with 10000 iterations (first 2000 discarded), n.thin = 10
 n.sims = 4000 iterations saved
          mean    sd    2.5%     25%     50%     75%   97.5% Rhat n.eff
beta0[1]  0.48  0.03    0.41    0.46    0.48    0.51    0.55 1.01   430
```

```
beta0[2]   0.41  0.03    0.36    0.39    0.41    0.43    0.46 1.01   560
beta0[3]   0.50  0.03    0.45    0.48    0.50    0.52    0.55 1.01   360
   *    *    *   DELETED
beta0[15]  0.53  0.03    0.47    0.51    0.53    0.55    0.59 1.01   420
beta1      0.06  0.01    0.05    0.05    0.06    0.06    0.07 1.01   380
tau      213.97 17.52  181.50  201.90  213.50  225.60  250.10 1.00  2800
mu.0       0.38  0.04    0.30    0.35    0.37    0.40    0.45 1.00  2000
tau.0     78.26 30.56   31.23   55.83   74.17   95.75  147.50 1.00  4000
```

Now we move on to a model with both β_0 and β_1 varying with field, analogous to Equation 12.7.

```
> Set3.model3 <- function(){
+ for (i in 1:n.F){
+    beta0[i] ~ dnorm(mu.0, tau.0)
+    beta1[i] ~ dnorm(mu.1, tau.1)
+ }
+ mu.0 ~ dnorm(0, 0.001)
+ tau.0 ~ dgamma(0.01, 0.01)
+ mu.1 ~ dnorm(0, 0.01)
+ tau.1 ~ dgamma(0.01, 0.001)
+ tau ~ dgamma(0.01, 0.01)
+ for (i in 1:n)
+   {
+   Y.hat[i] <- beta0[Field[i]] + beta1[Field[i]]*Irrig[i]
+   Yield[i] ~ dnorm(Y.hat[i], tau)
+   }
+ }
```

The rest of the setup and execution is analogous to the model with only β_0 dependent on group. Here are the results of the simulation step.

```
> Set3.sim3 <- bugs(data = XY.data3, inits = XY.inits3,
+    model.file = paste(mybugsdir,"\\Set3model.bug", sep = ""),
+    parameters = c("beta0", "beta1", "tau", "mu.0", "tau.0", "mu.1",
"tau.1"), n.chains = 5,
+    n.iter = 10000, n.burnin = 2000, n.thin = 20,
+    bugs.directory = mybugsdir)
> print(Set3.sim, digits = 2)
Inference for Bugs model at "c:\aardata\book\Winbugs\\Set3model.bug", fit
using WinBUGS,
5 chains, each with 10000 iterations (first 2000 discarded), n.thin = 20
n.sims = 2000 iterations saved
5 chains, each with 10000 iterations (first 2000 discarded), n.thin = 20
n.sims = 2000 iterations saved
           mean     sd    2.5%     25%     50%     75%   97.5% Rhat
n.eff
beta0[1]   0.47   0.08    0.32    0.42    0.47    0.51    0.61 1.01   540
beta0[2]   0.51   0.07    0.38    0.46    0.51    0.55    0.65 1.00   730
beta0[3]   0.43   0.06    0.31    0.39    0.43    0.47    0.54 1.00  1100
*    *    *   DELETED   *    *    *
beta1[13]  0.05   0.02    0.01    0.04    0.05    0.07    0.10 1.00  2000
beta1[14]  0.05   0.02    0.01    0.03    0.05    0.06    0.09 1.00  1200
beta1[15]  0.01   0.02   -0.02    0.00    0.01    0.03    0.05 1.01   230
```

```
tau     236.19 20.06  198.19  222.50  235.60  249.52  276.11 1.00  2000
mu.0      0.39  0.05    0.30    0.35    0.39    0.42    0.49 1.00  1400
tau.0    49.33 23.38   16.15   32.37   45.78   60.76  104.91 1.00  1000
mu.1      0.06  0.01    0.03    0.05    0.06    0.06    0.08 1.00  2000
tau.1  1081.64 601.61 319.97  652.62  956.20 1342.25 2608.22 1.01   480
devian -825.72  9.25 -842.10 -832.30 -826.20 -819.70 -806.30 1.00  1000
```

We can use this result to get some insight into the question of the importance of the field versus the farmer in influencing yield. The fields in the central region tend to have a higher silt content than those in the other regions. Soil texture has a strong influence on soil water-holding capacity and drainage, and it is possible that improvements in irrigation effectiveness would not improve yield as much in the central region as in the other two regions. Let us compare the relationship between irrigation effectiveness and yield in the central region with that in the combined northern and southern regions. That is, we will compare the slopes b_1 of the regression lines. This analysis could also be carried out using the mixed model results of Chapter 12.

First, we compute the mean values of b_1 for the two regions (noting that Fields 1 through 4 and 15 are in the north or south of `data.Set3a`).

```
> param.est <- t(unlist(Set3.sim3$mean))
> b1 <- param.est[16:30]
> print(b1NS <- b1[c(1:4, 15)], digits = 2)
[1] 0.062 0.031 0.080 0.090 0.015
> print(b1C <- b1[5:14], digits = 2)
 [1] 0.100 0.031 0.052 0.074 0.060 0.032 0.058 0.058 0.054 0.049
> mean(b1NS)
[1] 0.05563988
> mean(b1C)
[1] 0.05674888
> mean(b1NS) - mean(b1C)
[1] -0.001108998
```

The mean value of b_1 is actually higher in the central region, which would indicate that on average, an increase in irrigation effectiveness would have a greater effect in this region. There is a lot of variability, however. Let's carry out a permutation test (Section 4.2.3) to determine whether the difference is significant. Recall that we concatenate the two vectors of b_1, randomly rearrange their elements many times, and determine where our observed difference sits among the rearrangements.

```
> b <- c(b1NS, b1C)
> print(d0 <- mean(b[1:5] - b[6:length(b)]))
[1] -0.001108998
> u <- numeric(1000)
> sample.d <- function(b){
+     b.samp <- sample(b)
+     mean(b.samp[1:5] - b.samp[6:length(b)])
+ }
> set.seed(123)
> u <- replicate(999, sample.d(b))
> u[1000] <- d0
> print(p <- length(which(u > d0)) / length(u))
[1] 0.537
```

The observed difference is not statistically significant. The results of this little analysis support the conclusion that improvements in irrigation effectiveness would have the same average impact on yield in the central region as they would in the northern or southern region.

The hierarchical analysis carried out in this section made implicit use of the spatial relationships among the data records through the grouping structure of the fields. None of the analyses completed so far have made explicit use of the spatial relationships of the data, that is of the geographical coordinates of the data records. We will take up explicit spatial analyses in the next section.

14.5 Incorporation of Spatial Effects

14.5.1 Spatial Effects in the Linear Model

The Bayesian approach to regression modeling of spatial data was largely initiated in a seminal paper by Besag (1974) in the context of the conditional autoregressive (CAR) model. The CAR model was introduced in Section 13.6. The equation for a linear CAR model with constant variance, given as Equation 13.42, is

$$E\{Y_i \mid Y_{-i}\} = X\beta + \sum_{j \neq i} c_{ij}(Y_j - X\beta)$$
$$\text{var}\{Y_i \mid Y_{-i}\} = \sigma^2, \tag{14.22}$$

where Y_{-i} is the vector of all of the Y values except Y_i. As discussed in Section 13.6, the weights matrix C is not the same as the spatial weights matrix W of the spatial autoregressive (SAR) model, although they are related. Moreover, in the model of Equation 14.22, we must have $c_{ij} = c_{ji}$. Our discussion of the issues associated with the CAR model is simplified a bit. For example, we assume that the variance term σ^2 is constant. Readers who wish to see the more general discussion may find it in the references, beginning with Waller and Gotway (2004, p. 270) and continuing with other sources listed in Section 14.7.

One of the issues associated with the conditional formulation is that in many cases the adoption of a standard form for the spatial relationships suitable for the CAR model results in a density function for the joint distribution of the Y values that cannot be integrated (Besag et al., 1991; Banerjee et al., 2004). Such a formulation can be used only to describe the prior densities, and not the model itself. Such a set of densities is called an *improper prior*. It is possible to formulate proper densities, but these have disadvantages that, for us, outweigh their advantages. Therefore, we will use improper priors in our analysis.

We will introduce the Bayesian approach to spatial modeling through the analysis of a simple model using artificial data, similar to the models used to introduce the spatial autoregression in Chapter 13. We will generate artificial data relating Y to X on a ten by ten square grid. First, we set up the model.

```
> lambda <- 0.4
> nlist <- cell2nb(10,10)
> IrWinv <- invIrM(nlist, lambda)
> set.seed(123)
> X <- rnorm(100)
> eps <- as.numeric(IrWinv %*% rnorm(100))
```

```
> sig2 <- 0.1
> Y <- X + sig2*eps
```

Now we will display SAR and CAR analyses using the function `spautolm()` described in Chapter 13. To keep the output simple, we will only display the regression coefficients.

```
> W <- nb2listw(nlist, style = "B")
> Y.SAR <- spautolm(Y ~ X, listw = W)
> print(coef(Y.SAR), digits = 3)
(Intercept)           X      lambda
    -0.0204      0.9975      0.1161
> Y.CAR <- spautolm(Y ~ X, listw = W, family = "CAR")
> print(coef(Y.CAR), digits = 3)
(Intercept)           X      lambda
    -0.0208      0.9967      0.1845
```

The estimates of β_0 and β_1 are close to their true values of 0 and 1, but the estimates of λ differ substantially from the value used to create the data set. This may be due to random variation, or it may be partly because no effort was made to take boundary effects into account.

Now we will use `R2WinBUGS` to analyze the CAR model. We will go through this first example in some detail. First, we will set up the independent errors WinBUGS model, and then we will modify the necessary lines to analyze the full CAR model. We will also anticipate the use of this code as the template for a subsequent model involving real data. Here is the initial WinBUGS formulation.

```
> library(R2WinBUGS)
> # WinBUGS formulation
> demo.model <- function(){
+ beta0 ~ dnorm(0, 0.01)
+ beta1 ~ dnorm(0, 0.01)
+ tau ~ dgamma(0.01, 0.01)
+ for (i in 1:100)
+   {
+   Y.hat[i] <- beta0 + beta1*X[i]
+   Y[i] ~ dnorm(Y.hat[i], tau)
+   }
+ }
> write.model(demo.model,
+    paste(mybugsdir,"\\demomodel.bug", sep = ""))
> data.XY <- data.frame(X = X, Y = Y)
> XY.data <- list(X = data.XY$X, Y = data.XY$Y)
> XY.inits <- function(){
+    list(beta0 = rnorm(1), beta1 = rnorm(1), tau = exp(rnorm(1)))}
```

The data frame `data.XY` is created to facilitate later transfer to the real data model. We go through the debug, coda, and simulate sequence to produce this WinBUGS simulation.

```
> demo.sim <- bugs(data = XY.data, inits = XY.inits,
+    model.file = paste(mybugsdir,"\\demomodel.bug", sep = ""),
+    parameters = c("beta0", "beta1", "tau"), n.chains = 5,
+    n.iter = 10000, n.burnin = 1000, n.thin = 10,
+    bugs.directory = mybugsdir)
```

```
> print(demo.sim, digits = 2)
Inference for Bugs model at "c:\aardata\book\Winbugs\\demomodel.bug",
fit using WinBUGS,
 5 chains, each with 10000 iterations (first 1000 discarded), n.thin = 10
 n.sims = 4500 iterations saved
         mean    sd    2.5%     25%     50%     75%   97.5% Rhat n.eff
beta0   -0.02  0.01   -0.04   -0.02   -0.02   -0.01    0.00    1  4500
beta1    0.99  0.01    0.97    0.99    0.99    1.00    1.02    1  4500
tau     90.12 12.87   66.60   80.97   89.52   98.44  117.10    1  4500
```

Now we will modify the code for the CAR analysis. The program WinBUGS contains a built-in program called GeoBUGS (Thomas et al., 2004) that facilitates the construction of CAR models. In particular, GeoBUGS provides a function called `car.normal()` that can be used to specify an improper prior Gaussian distribution. This function takes four arguments: `adj`, `num`, `weights`, and `tau`. The vector `adj` contains the adjacency values, that is, the data records spatially adjacent to data rerecord *i*. The vector `weights` contains the values of the spatial weights matrix. The vector `num` contains the number of neighbors for each data record. The scalar `tau` contains the inverse variance of the normal distribution.

The `spdep` package contains a function `nb2WB()` that creates the arguments for the function `car.normal()` from a neighbor list.

```
> W.WB <- nb2WB(nlist)
```

The CAR model is implemented by first adding a term `v[i]` in the model statement as shown below. The first modified line establishes the vector `v[i]` as describing the autocorrelation effect, and the second modified line inserts the vector in the linear model.

```
> CARlm.model <- function(){
+ beta0 ~ dnorm(0, 0.01)
+ beta1 ~ dnorm(0, 0.01)
+ tau ~ dgamma(0.01, 0.01)
+ # This line is modified.
+ v[1:100] ~ car.normal(adj[], weights[], num[], tau)
+   for (i in 1:100)
+   {
+ # This line is modified.
+   Y.hat[i] <- beta0 + beta1*X[i] + v[i]
+   Y[i] ~ dnorm(Y.hat[i], tau)
+   }
+ }
```

The `write.model()` statement is unchanged and is not shown. The `xy.data()` and `xy.inits()` statements incorporate the objects created by the function `nb2WB()`.

```
> XY.data <- list(X = data.XY$X, Y = data.XY$Y,
+   adj = W.WB$adj, weights = W.WB$weights,
+   num = W.WB$num, n = nrow(data.XY))
> XY.inits <- list(list(beta0 = 0, beta1 = 0, tau = 0.01,
+    v = rep(0.01,100)))
```

Once again, we run the debug, coda, and simulate sequence. Here is the result of the WinBUGS simulation.

```
> demo.sim <- bugs(data = XY.data, inits = XY.inits,
+    model.file = paste(mybugsdir,"\\demomodel.bug", sep = ""),
```

```
+    parameters = c("beta0", "beta1", "tau", "v"), n.chains = 5,
+    n.iter = 10000, n.burnin = 1000, n.thin = 10,
+    bugs.directory = mybugsdir)
> print(demo.sim, digits = 2)
Inference for Bugs model at "c:\aardata\book\Winbugs\\demomodel.bug", fit
using WinBUGS,
 5 chains, each with 10000 iterations (first 1000 discarded), n.thin = 10
 n.sims = 4500 iterations saved
         mean    sd    2.5%     25%     50%     75%   97.5% Rhat n.eff
beta0   -0.02  0.01   -0.03   -0.02   -0.02   -0.01    0.00    1  4500
beta1    0.99  0.01    0.97    0.99    0.99    1.00    1.02    1  4500
tau    132.81 19.14   98.38  119.30  131.80  145.22  172.40    1  4500
v[1]    -0.03  0.06   -0.14   -0.07   -0.03    0.01    0.08    1  4500
v[2]    -0.01  0.05   -0.11   -0.04   -0.01    0.02    0.09    1  3700
   *    *   *   DELETED   *   *   *
v[100]  -0.06  0.06   -0.18   -0.10   -0.06   -0.03    0.05    1  4500
```

The results indicate that the process has converged and that the effective number of simulation draws is acceptable. The coefficients are comparable with the results from the other methods. As a check on the CAR model, you are asked in Exercise 14.6 to run this model with lambda set at zero and compare the results of the CAR model with those of the model that does not include spatial autocorrelation.

It all looks very easy, right? Well, maybe not always.

14.5.2 Application to Data Set 3

As a demonstration of the application of the Bayesian approach to a real data set, we will return to the relationship between *Yield* and *Irrig* analyzed in Section 14.4. We will start with the pooled data model and then move to the hierarchical model. Once again, the easiest way to proceed in coding the model is to modify an existing model that works. In this case, I will use the simple demonstration model from Section 14.5.1. To emphasize the benefits of moving one small step at a time, I will change the code of Section 14.5.1 as little as possible to get a working model for the pooled data of Data Set 3. The model function does not need to be changed at all. The statement defining the spatial weights matrix `W.WB` is changed.

```
> nlist <- dnearneigh(data.Set3a, d1 = 0, d2 = 600)
> W.WB <- nb2WB(nlist)
```

Next, the assignment statement defining `data.XY` is changed, and the names of the data records `Irrig` and `Yield` are substituted for `X` and `Y` in the statement defining the data.

```
> data.XY <- data.Set3a
> XY.data <- list(X = data.XY$Irrig, Y = data.XY$YieldN,
+   adj = W.WB$adj, weights = W.WB$weights, num = W.WB$num,
+   n = nrow(data.XY))
```

Everything else remains the same. The coda analysis looks reasonable, so we run the simulation. Here is the result.

```
> Set3.sim <- bugs(data = XY.data, inits = XY.inits,
+    model.file = paste(mybugsdir,"\\Set3model.bug", sep = ""),
+    parameters = c("beta0", "beta1", "tau", "v"), n.chains = 5,
```

```
+    n.iter = 5000, n.burnin = 1000, n.thin = 10,
+    bugs.directory = mybugsdir)
>
> print(Set3.sim, digits = 2)
Inference for Bugs model at "c:\aardata\book\Winbugs\demomodel.bug", fit
using WinBUGS,
 5 chains, each with 5000 iterations (first 1000 discarded), n.thin = 10
 n.sims = 2000 iterations saved
           mean    sd   2.5%      25%    50%    75%  97.5%   Rhat n.eff
beta0       0.4   0.0    0.3      0.4    0.4    0.4    0.4   1.00   600
beta1       0.1   0.0    0.1      0.1    0.1    0.1    0.1   1.00   540
tau       223.2  18.1  187.3    211.0  223.1  234.8  260.3   1.00  2000
v[1]       -0.1   0.0   -0.2     -0.1   -0.1   -0.1   -0.1   1.00  2000
v[2]       -0.1   0.0   -0.2     -0.1   -0.1   -0.1   -0.1   1.00  1600
   *   *   *   DELETED   *   *   *
v[313]      0.15  0.02    0.11    0.1    0.1    0.2    0.2  1.00 2000
deviance 821.75 13.66 -847.61 -830.8 -822.1  -812.4  -795.2  1.00 2000
```

Let's compare the results of four different models.

```
> Y.lm <- lm(YieldN ~ Irrig, data = data.Set3a)
> W.B <- nb2listw(nlist, style = "B")
> Y.SAR <- spautolm(YieldN ~ Irrig,
+    data = data.Set3a, listw = W.B)
> Y.CAR <- spautolm(YieldN ~ Irrig,
+    data = data.Set3a, family = "CAR", listw = W.B)
> print(coef(Y.lm), digits = 3)
(Intercept)       Irrig
      0.204       0.114
> print(coef(Y.SAR), digits = 3)
(Intercept)       Irrig      lambda
     0.3112      0.0868      0.0522
> print(coef(Y.CAR), digits = 3)
(Intercept)       Irrig      lambda
     0.2781      0.0961      0.0528
> print(t(Set3.sim$mean[1:3]), digits = 3)
     beta0 beta1  tau
[1,] 0.389 0.0633 223
```

The results of the Bayesian analysis are a bit different from the others, which is disconcerting. We will keep this in mind but move on to the multi-intercept hierarchical model.

Once again, we build the more complex model starting from a simpler one, this time the pooled data model that we have just discussed. The symbols are adjusted for consistency with the model in Section 14.4. Here is the code.

```
> data.Set3a$Field[which(data.Set3a$Field == 16)] <- 2
>
> Set3.model <- function(){
+ # Priors for beta0
+ for (i in 1:n.F){
+    beta0[i] ~ dnorm(mu.0, tau.0)
+ }
+ # Parameters for the priors
+ mu.0 ~ dnorm(0, 0.001)
```

```
+ tau.0 ~ dgamma(0.01, 0.01)
+ beta1 ~ dnorm(0, 0.001)
+ tau ~ dgamma(0.01, 0.01)
+ # This line is modified.
+ v[1:n] ~ car.normal(adj[], weights[], num[], tau)
+ for (i in 1:n)
+   {
+   Y.hat[i] <- beta0[Field[i]] + beta1*Irrig[i] + v[i]
+   Yield[i] ~ dnorm(Y.hat[i], tau)
+   }
+ }
```

The `XY.data` and `XY.inits` statements are set appropriately.

```
> XY.data <- list(Irrig = data.Set3a$Irrig, Yield = data.Set3a$YieldN,
+    n = nrow(data.Set3a), n.F = length(unique(data.Set3a$Field)),
+    adj = W.WB$adj, weights = W.WB$weights,
+    num = W.WB$num, n = nrow(data.Set3a), Field = data.Set3a$Field)
> n.F <- length(unique(data.Set3a$Field))
> XY.inits <- function(){
+    list(beta0 = rnorm(n.F), beta1 = rnorm(1), tau = exp(rnorm(1)),
+    mu.0 = rnorm(1), tau.0 = exp(rnorm(1)),
+    v = rep(0.01,nrow(data.Set3a)))}
```

As usual, we run the test and coda steps. Here is the code for the coda step.

```
> Set3.coda <- bugs(data = XY.data, inits = XY.inits,
+    model.file = paste(mybugsdir,"\\Set3model.bug", sep = ""),
+    parameters = c("beta0", "beta1", "tau", "mu.0", "tau.0", "v"),
+    n.chains = 5, n.iter = 10000, n.burnin = 2000, n.thin = 10,
+    codaPkg = TRUE, bugs.directory = mybugsdir)
```

The resulting graphs provide a good example of what chains should not look like (Figure 14.8). Rather than looking like noise, several of the chains have a distinct pattern, indicating that they have not converged. Unfortunately, this occurrence seems to happen more often than not in models of more than the smallest level of complexity. Gelman and Hill (2007) devote a full chapter to dealing with failed WinBUGS programs, but for our particular case, probably the best (certainly the easiest) thing to do, assuming we have the patience to wait for the computer, is to increase the number of iterations, burn-in interval, and thinning interval.

```
> Set3.sim <- bugs(data = XY.data, inits = XY.inits,
+    model.file = paste(mybugsdir,"\\Set3model.bug", sep = ""),
+    parameters = c("beta0", "beta1", "tau", "mu.0", "tau.0", "v"),
+    n.chains = 5, n.iter = 30000, n.burnin = 20000, n.thin = 25,
+    bugs.directory = mybugsdir)
> print(Set3.sim)
Inference for Bugs model at "c:\aardata\book\Winbugs\Set3model.bug", fit
using WinBUGS,
5 chains, each with 30000 iterations (first 20000 discarded), n.thin = 25
 n.sims = 2000 iterations saved
            mean    sd   2.5%    25%    50%    75%  97.5% Rhat n.eff
beta0[1]     0.4   0.1    0.2    0.3    0.4    0.5    0.6  1.0   200
beta0[2]     0.4   0.1    0.2    0.3    0.4    0.5    0.6  1.0   260
```

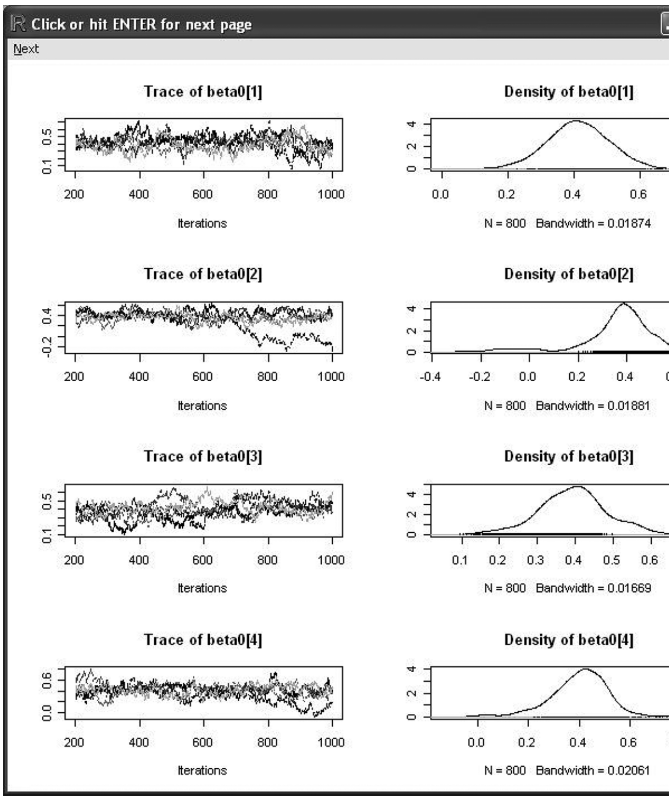

FIGURE 14.8
Diagnostic plots generated by coda for the model of *Yield* versus *Irrig* for Data Set 3. Note that several of the chains appear to wander, unlike those of Figure 14.7.

```
beta0[3]     0.4    0.1     0.3     0.4     0.4     0.5     0.6   1.1     29
beta0[4]     0.4    0.1     0.2     0.4     0.4     0.5     0.6   1.1     41
beta0[5]     0.4    0.1     0.3     0.4     0.4     0.5     0.5   1.0    150
                     *     *     *
```

Even at very large values of `n.iter`, `n.burnin`, and `n.thin`, the results are discouraging. A comparison of the first five values of the intercept β_0 with those obtained in the mixed model analysis of Section 12.3 is even more discouraging.

```
> library(nlme)
> print(t(coef(lme(YieldN ~ Irrig, data = data.Set3a,
+    random = ~ 1 | Field)))[1,1:5], digits = 3)
    1     2     3     4     5
0.480 0.411 0.501 0.558 0.366
> print(t(unlist(Set3.sim$mean))[1:5], digits = 3)
[1] 0.398 0.399 0.413 0.404 0.416
```

The purpose of this section has been to show that the incorporation of spatial autocorrelation into a Bayesian model may not work, at least not without a lot of effort. In such a case, to carry out a Bayesian analysis one would have to fall back on the model that does not include spatial autocorrelation. In the present case, for example, the mixed model analysis of Chapter 12 indicated that spatial autocorrelation does not have a strong effect on the model.

14.5.3 The spBayes Package

Bayesian analysis in general and Bayesian analysis of spatial data in particular is one of the most active areas of research and of the development of new R packages. One of the most popular and useful is `spBayes` (Finley et al. 2015) and we will use this as an example of the capabilities of these packages. The package `spBayes` implements a variety of models using the MCMC methods that vary in two important ways from those described previously in this chapter. The first difference is the probability model itself. We will illustrate this difference by reexamining the Sierra Nevada oak model that we studied in Section 14.3.4 using the WinBUGS. Recall from Equation 14.21 in that section that the model is specified by the expected value of the binary distributed random variable Y, $E\{Y \mid X_i\} = \pi_i$ where π_i is specified via the link function as $\ln(\pi_i / (1-\pi_i)) = \beta_0 + \Sigma\beta_j X_j$. This same formulation is used in the CAR model described in Equation14.22. In `spBayes` the error is treated as a geostatistical random variable that is described by a variogram. For this reason the model is not expressed in terms of the expected value. Rather it is given as

$$\begin{aligned} Y_i &\sim Binomial(\pi_i), \\ \ln\left(\frac{\pi_i}{1-\pi_i}\right) &= \beta_0 + \sum_{i=1}^{n} \beta_i X_i + w_i. \end{aligned} \tag{14.23}$$

Here w_i describes the site-specific Gaussian random field (cf Sections 3.1 and 3.2). If we let W be the vector whose components are the w_i, then W is a multivariate normal vector with mean 0 and variance-covariance matrix Σ. The value of the components σ_{ij} depends via a variogram model on the distance h_{ij} that separates them (cf. Section 4.6.1). We will use the exponential model, so that

$$\sigma_{ij} = \sigma^2 \exp(-\varphi h_{ij}). \tag{14.24}$$

The second way in which the algorithms in `spBayes` are different is that, although they use the MCMC method, they do not use the Gibbs sampler described in Section 14.2. Instead, they use a more sophisticated method called the Metropolis-Hastings algorithm (Metropolis et al., 1953; Hastings, 1970). With this algorithm, the prior distribution of the parameters is specified as before, but instead of iterating directly on this distribution a second probability density called the *proposal distribution* is used. Very roughly speaking, at each iteration, instead of drawing from the existing distribution to generate the updated one, as does the Gibbs sampler, distributional parameter values are drawn from the proposal distribution. If the resulting probability density represents an improvement over the previous iteration, it is accepted. If not, it is rejected. The parameters of the proposal distribution are called the *tuning parameters.*

Turning to the Sierra Nevada oak versus elevation generalized linear model (GLM) from Section 14.3.4, after reading in the data, we first normalize the elevation data and scale the coordinates to avoid numerical problems.

```
> data.Set2S.sf$ElevN <-
+   data.Set2S.sf$Elevation / max(data.Set2S.sf$Elevation)
> data.Set2S.sf$x <- scale(data.Set2S.sf$Longitude)
> data.Set2S.sf$y <- scale(data.Set2S.sf$Latitude)
> glm.demo <- glm(QUDO ~ ElevN, data = data.Set2S.sf, family =
```

```
+    binomial)
> coef(glm.demo)
(Intercept)       ElevN
   3.329145  -11.463550
```

The full Sierra Nevada data set has almost 2000 records. To simplify things for this demonstration, we will use the modulo operator to reduce that number by a factor of ten.

```
> ID <- 1:nrow(data.Set2S.sf)
> mod.fac <- 10
> data.Set2Sr.sf <- data.Set2S.sf[which(ID %% mod.fac == 0),]
> nrow(data.Set2Sr.sf)
[1] 176
```

Recall that `data.Set2Sr.sf` is a spatial features object. Just to make sure this data reduction has not affected the analysis too much, we can rerun the GLM.

```
> glm.small <- glm(QUDO ~ ElevN, data = data.Set2Sr.sf, family =
+   binomial)
> coef(glm.small)
(Intercept)       ElevN
   4.213447  -13.457494
```

Again, to reduce the possibility of numerical problems, we rescale the coordinate values.

```
> Set2Sr.coords <- as.matrix(data.Set2Sr.sf[, c("x","y")])[,1:2]
```

Next, we need to set the values of the starting and tuning parameters β_i.

```
> beta.start <- as.numeric(coef(glm.small))
> beta.tuning <- t(chol(vcov(glm.small)))
```

The starting values are those of the GLM. The tuning parameters require some explanation. The `spGLM` documentation recommends using for the tuning parameters the lower triangular Cholesky decomposition of the variance-covariance matrix of the desired parameters of the proposal distribution. The function `vcov()` computes the variance-covariance matrix of this model. The function `chol()` computes the *Cholesky decomposition* of the variance covariance matrix. As used by `spBayes`, the Cholesky decomposition of a matrix A, sometimes called the "square root" of A (Press et al., 1986, p. 311), is a lower triangular matrix (i.e., one with zeroes above the diagonal) denoted L such that $LL' = A$, (where the prime denotes transpose). The function `chol()` returns it in upper triangular form so the transpose function `t()` converts this to lower triangular form.

The function call to the `spBayes` function `spGLM()` is a long one, so we will give the code first and then explain it.

```
> n.samples <- 100000
> model.glm.bayes <- spGLM(QUDO ~ ElevN, data = data.Set2Sr.sf, family +
+       = "binomial", coords = Set2Sr.coords,
+   starting = list("beta" = beta.start, "phi"=0.05,"sigma.sq"=0.1,"w"
+       = 1),
```

```
+      tuning = list("beta" = beta.tuning, "phi"=0.05, "sigma.sq"=0.1,
+        "w"=0.01),
+      priors=list("beta.Flat", "phi.Unif"=c(0.01, 0.50), "sigma.sq.IG"
+        =c (2, 1)),
+      cov.model = "exponential", verbose=FALSE, n.samples = n.samples)
```

The number of MCMC samples is set at 100,000, which should give us a good result if one is possible. The first four arguments are straightforward. The argument `starting` is a list of starting values for each parameter with each tag corresponding to a parameter name. These correspond to the parameters in Equations 14.23 and 14.24. Because only one value is given for the vector W in Equation 14.23, that value is used for all of its components. The same thing applies to the values of the argument `tuning` of the proposal density and the parameters of the prior density. The last line specifies that we use the exponential model as in Equation 14.24, that intermediate output is suppressed, and gives the number of samples.

The function `spGLM()` returns a lot of information. We can see what we get by first looking at the names of all of its elements.

```
> names(model.glm.bayes)
 [1] "p.beta.theta.samples" "acceptance"           "p.w.samples"
 [4] "weights"              "family"               "Y"
 [7] "X"                    "coords"               "is.pp"
[10] "cov.model"            "run.time"
```

According to the `spGLM()` documentation, the variable `p.beta.theta.samples` is "a `coda` object of posterior samples for the defined parameters." Since it is a `coda` object we can plot it as we did with WinBUGS. We will set a burn in period of the first half of the samples, and use the R function `window()` to extract the second half. First, we plot the results.

```
> burn.in <- 0.5 * n.samples
> plot(window(model.glm.bayes$p.beta.theta.samples, start = burn.in))
```

Figure 14.9 shows the results. These don't look as pretty as the ones in the textbooks, but mine never seem to. Next, we plot the results.

```
> summary(window(model.glm.bayes$p.beta.theta.samples, start =
+   burn.in))

Iterations = 50000:1e+05
Thinning interval = 1
Number of chains = 1
Sample size per chain = 50001

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

                Mean      SD  Naive SE Time-series SE
(Intercept)   3.6827 0.74469 0.0033303        0.11844
ElevN       -14.8351 2.30977 0.0103295        0.37997
sigma.sq      1.8162 0.57684 0.0025797        0.15159
phi           0.3851 0.08042 0.0003596        0.04585
```

```
2. Quantiles for each variable:

                2.5%      25%      50%      75%    97.5%
(Intercept)   2.3027   3.0314   3.7452   4.1623   5.4000
ElevN       -20.4809 -16.1254 -15.0191 -13.0516 -10.7612
sigma.sq      1.2118   1.3616   1.6269   2.1382   3.2950
phi           0.2127   0.3447   0.4207   0.4519   0.4784
```

Because some of the distributions are far from normal, the median values are probably more appropriate to use as the reported output. The values are reasonably close to those generated by `glm()`, which is a good sign. In the next section, we will summarize the results of the various analyses we have carried out in this chapter and how they apply to the issued of frequentist versus subjective probability in spatial data analysis.

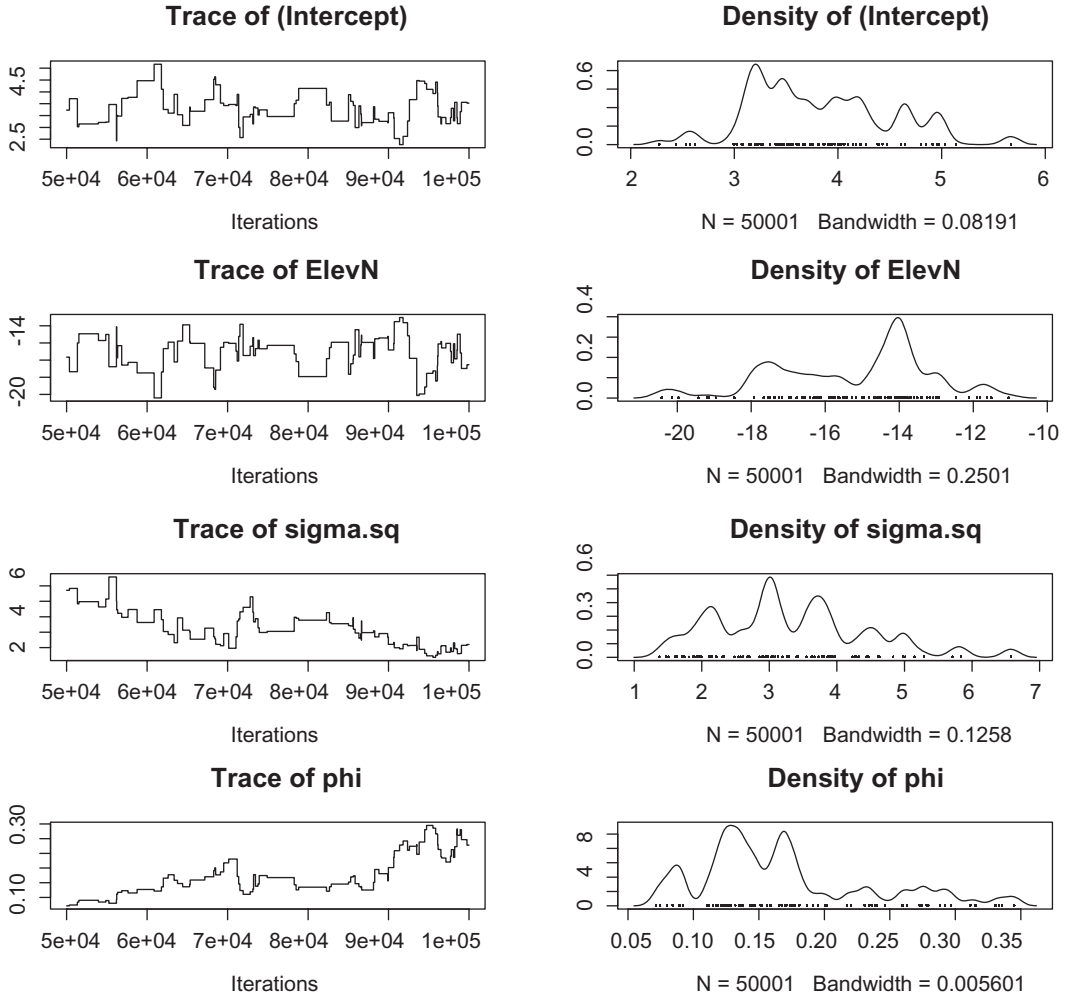

FIGURE 14.9
Diagnostic information from `spBayes()` model of *QUDO* versus *ElevN*.

14.6 Comparison of the Methods

First of all, it is important to emphasize that Bayes' theorem is exactly that, a mathematical theorem, and the results of a correct application of that theorem are mathematically correct. Bayes' theorem has been used to derive many highly useful probabilistic results and tools, such as the Kalman filter (Gelb, 1974), that do not necessarily invoke subjective probability in the manner that it is presented in this chapter. Moreover, some operations, such as search and rescue, that do involve using the repeated incorporation of new data to improve an estimate, fit very naturally into a subjective probability framework. Indeed, we will have occasion to apply these methods when we discuss spatiotemporal data in Chapter 15. Finally, the power of the MCMC method is increasingly being used to attack so-called "Big Data" problems, where the size of the data set precludes analysis by more traditional methods (Miroshnikov and Conlon, 2014). In this context, it is probably unfortunate that the term *Bayesian analysis* has come to be the standard term to characterize this approach, as opposed to "subjective probability," which more appropriately emphasizes the distinction between it and the other, more "traditional" methods that are generally called the "frequentist" approach.

At a theoretical level, subjective probability does remove some of the imprecision in the interpretation of parameters and the probabilities associated with them. At a more practical level, it is often easier to formulate multilevel models in a Bayesian rather than a mixed-model form. Often the precise statement of a complex mixed effects model involving several interacting variables will leave even the mixed-model experts tied in knots. Both mixed models and Bayesian models require numerical solution using methods that are often subject to instability. The numerical solutions associated with Bayesian computation, however, often seem even more intractable than those associated with the mixed model. The WinBUGS manual (Spiegelhalter et al., 2003) has the statement on the first page, in red letters, "Beware: MCMC sampling can be dangerous!" For the practitioner who does not want to devote a career to working out statistical intricacies, this is not very reassuring.

At a very practical level, both WinBUGS and `spBayes` programs can be excruciatingly difficult to debug (it is probably for this reason that Gelman and Hill (2007) devote an entire chapter to the subject, one that should be read by anyone venturing into this territory). Using a terminology that is somewhat dated, we can divide the computer programs we construct in this book into two classes, interpreted and compiled. R programs are interpreted; that is, they are executed one line at a time. WinBUGS programs are compiled; that is, the entire program is translated into a machine readable form and then this is executed. Interpreted programs are much easier to construct and debug because you can see the results of each statement directly. Indeed, most modern compilers include a debugging module that allows the programmer to step through the code line by line on the screen as it executes. In this context, the `spBayes` package may appear to offer some advantage since it runs directly in R, but this advantage is sometimes illusory because actually most of the action takes place with a function such as `spGLM()` itself, behind the locked doors of the compiled C++ program. It is a poor carpenter who blames his or her tools, but nevertheless the efficiency of use of one's time is something that one must take into account in carrying out a data analysis project. To repeat something I have said already, in my own experience the most effective way to construct a new Bayesian program is to start with an existing program that works and move toward the new one in a series of baby steps.

There are, however, a number of examples of the successful application of Bayesian analysis to difficult problems. Several examples are given in the references in Section 14.7.

Anyone who is even remotely aware of the subject is also aware of the endless amounts of ink that have been spilled on one side or another of the debate between the "frequentist" and the "subjective" approaches. There is no point in adding to this, and as a practitioner I am not qualified to do so anyway. My own feeling is that the best approach is that presented by Bolker et al. (2009). One should be knowledgeable about both approaches and able to use both effectively and appropriately.

I want to conclude with one point that I consider especially important. In Section 12.1 it was stated that in a mixed-model analysis involving grouped variables, at least six or so levels of the grouping factor are needed to do a valid analysis. This is because a reasonably sized sample of values is needed to get a good estimate of the range of variation of the effect. Thus, for example, we can use the fields in Data Set 3 as a grouping factor because there are 16 of them, but we cannot use the fields in Data Set 4 in this way because there are only two. Some authors claim that Bayesian analysis eliminates this requirement for a minimum number of levels of a grouping factor because if its use of subjective probability. In my opinion, this is absolutely false. Critics of the subjective probability approach used to say that Bayesian analysis made it possible to carry out data analysis without actually having any data. One cannot make something out of nothing, however, and ultimately no method will ever permit you to draw justifiable conclusions without having the data to back them up. The amount of data needed to draw a valid conclusion is no different whether the method used is Bayesian analysis or mixed model analysis.

14.7 Further Reading

If you don't know anything about Bayesian analysis, McCarthy (2007) provides a nice introduction. Korner-Nievergelt et al. (2015) provide an excellent introduction to the use of Bayesian methods in the analysis of ecological data. Koop (2003) is an excellent introduction to Bayesian linear regression analysis and Markov Chain Monte Carlo methods. Gelman and Hill (2007) provide very complete coverage of both classical and Bayesian regression that is comprehensive, highly readable, and oriented to the practitioner. At a more intermediate level, Ntzoufras (2009) provides many good examples. Congdon (2003) and Gelman et al. (2014) provide a complete description of modern Bayesian methodology at a more advanced level. Cressie and Wilkes (2011) develop a sort of middle ground between the "frequentist" and the "subjective" approaches that focuses on conditional probability. This may gain traction in the future.

The term *Gibbs sampler* was coined by Geman and Geman (1984), who developed the method. They used this term because their data followed a Gibbs distribution, named for the physicist J. Willard Gibbs (1839–1903), who is often considered the founder of modern statistical thermodynamics. The distributions we work with are generally not Gibbs distributions, but the Gibbs sampler has since been shown to have far more general applicability. Clark and Gelfand (2006) and Gilks et al. (2010) contain collections of papers on Bayesian modeling. Jiang et al. (2009) carry out a Bayesian analysis of spatial data in an agricultural context. There are a number of R packages devoted to Bayesian analysis and the Markov Chain Monte Carlo method. For example, Hadfield (2010) provides in the `MCMCglmm` package a very nice implementation of the MCMC method for the solution of generalized linear mixed models.

Finally, those contemplating a Bayesian analysis may find it interesting to read Efron (1986) and the discussant papers that follow it.

Exercises

14.1 Develop and run in R2WinBUGS a model using artificial data for the regression relation $Y_i = \beta_0 + \beta_1 X_i + \varepsilon_i$. In other words, the same model as in Section 14.3.3, but including an intercept.

14.2 In Section 9.3.2 we analyzed two linear models, model A, in which the two explanatory variables are independent, and model B, in which the two explanatory variables are highly collinear. (a) Develop a Bayesian model analysis of model A. (b) Using the same values for `n.iter` and `n.burnin` as in part (a), compute the solution for model B. What do you observe?

14.3 Carry out a Bayesian analysis using R2WinBUGS of `model.5` from Section 8.3 and compare the results with those obtained in that section.

14.4 In Section 8.4.2 a GLM was developed for the Sierra Nevada subset that includes *Precip*, *MAT*, *SolRad*, *AWCAvg*, and *Permeab*. Analyze this model using R2WinBUGS.

14.5 In our mixed model and hierarchical analyses of *Yield* versus *Irrig*, we have always removed Field 2 because all of the values of *Irrig* are the same in that field. However, although this makes it impossible to carry out a regression analysis on data from this field in isolation, a mixed model or hierarchical analysis can still be carried out that incorporates the data. Repeat the analysis of Section 14.4 with data from Field 2 included, and compare the results with those obtained in that section.

14.6 The CAR model discussed in Section 14.5.1 should give the similar results to the ordinary WinBUGS model when $\lambda = 0$. Check to see whether it does.

14.7 (a) Read about the function `spLM()` in the package `spBayes`. Load the data for Field 4.1 and construct a model using `spLM()` for the linear regression of *Yield* on *Clay*. (b) Read about the values returned by `spLM()` and display a plot of the Bayesian parameters. Then read about the function `spRecover()` and use it to display the quantiles of the Bayesian estimates of β_0 and β_1. (c) Plot *Yield* versus *Clay* and plot the curve fits of both the linear regression computed using `lm()` and that computed using `spLM()`.