6

Markov Chain Monte Carlo Methods

6.1 Introduction

In Chapter 5, we introduced the use of simulation in Bayesian inference. Rejection sampling is a general method for simulating from an arbitrary posterior distribution, but it can be difficult to set up since it requires the construction of a suitable proposal density. Importance sampling and SIR algorithms are also general-purpose algorithms, but they also require proposal densities that may be difficult to find for high-dimensional problems. In this chapter, we illustrate the use of Markov chain Monte Carlo (MCMC) algorithms in summarizing posterior distributions. Markov chains are introduced in the discrete state space situation in Section 6.2. Through a simple random walk example, we illustrate some of the important properties of a special Markov chain, and we use R to simulate from the chain and move toward the stationary distribution. In Section 6.3, we describe two variants of the popular Metropolis-Hastings algorithms in setting up Markov chains, and in Section 6.4, we describe Gibbs sampling where the Markov chain is set up through the conditional distributions of the posterior. We describe one strategy for summarizing a posterior distribution and illustrate it for three problems. MCMC algorithms are very attractive in that they are easy to set up and program and require relatively little prior input from the user. R is a convenient language for programming these algorithm and is also very suitable for performing output analysis, where one does several graphical and numerical computations to check if the algorithm is indeed producing draws from the target posterior distribution.

6.2 Introduction to Discrete Markov Chains

Suppose a person takes a random walk on a number line on the values 1, 2, 3, 4, 5, 6. If the person is currently at an interior value (2, 3, 4, or 5), in the next second she is equally likely to remain at that number or move to an adjacent number. If she does move, she is equally likely to move left or right. If the person is currently at one of the end values (1 or 6), in the next second she is equally likely to stay or move to the adjacent location.

This is a simple example of a discrete Markov chain. A Markov chain describes probabilistic movement between a number of states. Here there are six possible states, 1 through 6, corresponding to the possible location of the walker. Given that the person is at a current location, she moves to other locations with specified probabilities. The probability she moves to another location depends only on her current location and not on previous locations visited. We describe movement between states in terms of transition probabilities – they describe the likelihoods of moving between all possible states in a single step in a Markov chain. We summarize the transition probabilities by means of a transition matrix T:

$$T = \begin{bmatrix} .50 & .50 & 0 & 0 & 0 & 0 \\ .25 & .50 & .25 & 0 & 0 & 0 \\ 0 & .25 & .50 & .25 & 0 & 0 \\ 0 & 0 & .25 & .50 & .25 & 0 \\ 0 & 0 & 0 & .25 & .50 & .25 \\ 0 & 0 & 0 & 0 & .50 & .50 \end{bmatrix}$$

The first row in T gives the probabilities of moving to all states 1 through 6 in a single step from location 1, the second row gives the transition probabilities in a single step from location 2, and so on.

There are several important properties of this particular Markov chain. It is possible to go from every state to every state in one or more steps – a Markov chain with this property is said to be *irreducible*. Given that the person is in a particular state, if the person can only return to this state at regular intervals, then the Markov chain is said to be *periodic*. This example is *aperiodic* since it is not a periodic Markov chain.

We can represent one's current location as a probability row vector of the form

$$p = (p_1, p_2, p_3, p_4, p_5, p_6),$$

where p_i represents the probability the person is currently in state i. If p^j represents the location of the traveler at step j, then the location of the traveler at the $j + 1$ step is given by the matrix product

$$p^{j+1} = p^j T.$$

Suppose we can find a probability vector w such that $wT = w$. Then w is said to be the *stationary* distribution. If a Markov chain is irreducible and aperiodic, then it has a unique stationary distribution. Moreover, the limiting distribution of this Markov chain, as the number of steps approaches infinity, will be equal to this stationary distribution.

We can empirically demonstrate the existence of the stationary distribution of our Markov chain by running a simulation experiment. We start our random walk at a particular state, say location 3, and then simulate many steps of the Markov chain using the transition matrix T. The relative frequencies of our traveler in the six locations after many steps will eventually approach the stationary distribution w.

We start our simulation in R by reading in the transition matrix `T` and setting up a storage vector `s` for the locations of our traveler in the random walk.

```
> T=matrix(c(.5,.5,0,0,0,0,.25,.5,.25,0,0,0,0,.25,.5,.25,0,0,
+           0,0,.25,.5,.25,0,0,0,0,.25,.5,.25,0,0,0,0,.5,.5),
+ nrow=6,ncol=6,byrow=TRUE)
> T

     [,1] [,2] [,3] [,4] [,5] [,6]
[1,] 0.50 0.50 0.00 0.00 0.00 0.00
[2,] 0.25 0.50 0.25 0.00 0.00 0.00
[3,] 0.00 0.25 0.50 0.25 0.00 0.00
[4,] 0.00 0.00 0.25 0.50 0.25 0.00
[5,] 0.00 0.00 0.00 0.25 0.50 0.25
[6,] 0.00 0.00 0.00 0.00 0.50 0.50

> s=array(0,c(50000,1))
```

We indicate that the starting location for our traveler is state 3 and perform a loop to simulate 50,000 draws from the Markov chain. We use the `sample` function to simulate one step – the arguments to this function indicate that we are sampling a single value from the set $\{1, 2, 3, 4, 5, 6\}$ with probabilities given by the s^{j-1} row of the transition matrix T, where s^{j-1} is the current location of our traveler.

```
> s[1]=3
> for (j in 2:50000)
+   s[j]=sample(1:6,size=1,prob=T[s[j-1],])
```

We summarize the frequencies of visits to the six states after 500, 2000, 8000, and 50,000 steps of the chain by use of the `table` command; we convert the counts to relative frequencies by dividing by the number of steps.

```
> m=c(500,2000,8000,50000)
> for (i in 1:4)
+   print(table(s[1:m[i]])/m[i])


    1     2     3     4     5     6
0.164 0.252 0.174 0.130 0.174 0.106

     1      2      3      4      5      6
0.1205 0.1965 0.1730 0.1735 0.2170 0.1195
```

```
       1        2        3        4        5        6
0.109250 0.188000 0.183875 0.194625 0.212000 0.112250

      1       2       3       4       5       6
0.10970 0.20770 0.19450 0.19342 0.19628 0.09840
```

It appears from the output that the relative frequencies of the states are converging to the stationary distribution $w = (0.1, 0.2, 0.2, 0.2, 0.2, 0.1)$. We can confirm that w is indeed the stationary distribution of this chain by multiplying w by the transition matrix T:

```
> w=matrix(c(.1,.2,.2,.2,.2,.1),nrow=1,ncol=6)
> w%*%T

     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]  0.1  0.2  0.2  0.2  0.2  0.1
```

6.3 Metropolis-Hasting Algorithms

A popular way of simulating from a general posterior distribution is by Markov chain Monte Carlo (MCMC) methods. This essentially is a continuous-valued generalization of the discrete Markov chain setup described in the previous section. The MCMC sampling strategy sets up an irreducible, aperiodic Markov chain for which the stationary distribution equals the posterior distribution of interest. A general way of constructing a Markov chain is by a Metropolis-Hastings algorithm. In this section, we focus on two particular variants of Metropolis-Hastings algorithms, the independence chain and the random walk chain, that are applicable to a wide variety of Bayesian inference problems.

Suppose we wish to simulate from a posterior density $g(\theta|y)$. In the following, to simplify notation, we write the density simply as $g(\theta)$. A Metropolis-Hastings algorithm begins with an initial value θ^0 and specifies a rule for simulating the tth value in the sequence θ^t given the $(t-1)$st value in the sequence θ^{t-1}. This rule consists of a *proposal density* which simulates a candidate value θ^*, and the computation of an *acceptance probability* P that indicates the probability the candidate value will be accepted to be the next value in the sequence. Specifically, this algorithm can be described as follows:

- Simulate a candidate value θ^* from a proposal density $p(\theta^*|\theta^{t-1})$.
- Compute the ratio
$$R = \frac{g(\theta^*)p(\theta^{t-1}|\theta^*)}{g(\theta^{t-1})p(\theta^*|\theta^{t-1})}.$$
- Compute the acceptance probability $P = \min\{R, 1\}$.
- Sample a value θ^t such that $\theta^t = \theta^*$ with probability P; otherwise $\theta^t = \theta^{t-1}$.

Under some easily satisfied regularity conditions on the proposal density $p(\theta^*|\theta^{t-1})$, the sequence of simulated draws $\theta^1, \theta^2, ...$ will converge to a random variable that is distributed according to the posterior distribution $g(\theta)$.

Different Metropolis-Hastings algorithms are constructed by the choice of proposal density. If the proposal density is independent of the current value in the sequence, that is,

$$p(\theta^*|\theta^{t-1}) = p(\theta^*),$$

then the resulting algorithm is called an *independence* chain. Other proposal densities can be defined by letting the density have the form

$$p(\theta^*|\theta^{t-1}) = h(\theta^* - \theta^{t-1}),$$

where h is a symmetric density about the origin. In this type of *random walk* chain, the ratio R has the simple form

$$R = \frac{g(\theta^*)}{g(\theta^{t-1})}.$$

The R functions `rwmetrop` and `indepmetrop` in the LearnBayes package implement, respectively, the random-walk and independence Metropolis-Hasting algorithms for special choices of proposal densities. For the function `rwmetrop`, the proposal density has the form

$$\theta^* = \theta^{t-1} + \text{scale}\, Z,$$

where Z is multivariate normal with mean vector 0 and variance-covariance matrix V and *scale* is a positive scale parameter. For the function `indepmetrop`, the proposal density for θ^* is multivariate normal with mean vector μ and covariance matrix V.

To use a Metropolis-Hastings algorithm, one first decides on the proposal density and then obtains a simulated sample of draws $\{\theta^t, t = 1, ...m\}$ by use of the R functions `rwmetrop` or `indepmetrop`. The output of each of these functions has two components: `par` is a matrix of simulated draws where each row corresponds to a value of θ, and `accept` gives the acceptance rate of the algorithm.

Desirable features of the proposal density in an algorithm depend on the MCMC algorithm employed. For an independence chain, we desire that the proposal density p approximates the posterior density g, suggesting a high acceptance rate. But, as in rejection sampling, it is important that the ratio g/p is bounded, especially in the tail portion of the posterior density. This means that one may choose a proposal p that is more diffuse than the posterior, resulting in a lower acceptance rate. For random walk chains with normal proposal densities, it has been suggested that acceptance rates between 25% and 45% are good. The "best" choice of acceptance rate ranges from 45% for one and two parameters to 25% for problems with more parameters. This advice also applies when one monitors the Metropolis within Gibbs algorithm described in Section 6.4.

6.4 Gibbs Sampling

One of the attractive methods of setting up an MCMC algorithm is Gibbs sampling. Suppose that the parameter vector of interest is $\theta = (\theta_1, ..., \theta_p)$. The joint posterior distribution of θ, which we denote by $[\theta|\text{data}]$, may be of high dimension and difficult to summarize. Suppose we define the set of conditional distributions

$$[\theta_1|\theta_2, ..., \theta_p, \text{data}],$$
$$[\theta_2|\theta_1, \theta_3, ..., \theta_p, \text{data}],$$
$$...$$
$$[\theta_p|\theta_1, ..., \theta_{p-1}, \text{data}],$$

where $[X|Y, Z]$ represents the distribution of X conditional on values of the random variables Y and Z. The idea behind Gibbs sampling is that we can set up a Markov chain simulation algorithm from the joint posterior distribution by successfully simulating individual parameters from the set of p conditional distributions. Simulating in turn one value of each individual parameter from these distributions is called one cycle of Gibbs sampling. Under general conditions, draws from this simulation algorithm will converge to the target distribution (the joint posterior of θ) of interest.

In situations where it is not convenient to sample directly from the conditional distributions, one can use a Metropolis algorithm such as the random walk type to simulate from each distribution. A “Metropolis within Gibbs” algorithm of this type is programmed in the function `gibbs` in the LearnBayes package. Suppose that θ_i^t represents the current value of θ_i in the simulation and let $g(\theta_i)$ represent the conditional distribution where we have suppressed the dependence of this distribution on values of the remaining components of θ. Then a candidate value for θ_i is given by

$$\theta_i^* = \theta_i^t + c_i Z,$$

where Z is a standard normal variate and c_i is a fixed scale parameter. The next simulated value of θ_i, θ_i^{t+1}, will be equal to the candidate value with probability $P = \min\{1, g(\theta_i^*)/g(\theta_i^t)\}$; otherwise the value $\theta_i^{t+1} = \theta_i^t$. To use the function `gibbs`, one inputs the function defining the log posterior, the starting value of the simulation, the number of Gibbs cycles, and a vector of scale parameters containing $c_1, ..., c_p$. The output of `gibbs` is a list; the component `par` is a matrix of simulated draws and `accept` is a vector of acceptance rates for the individual Metropolis steps.

6.5 MCMC Output Analysis

For the MCMC algorithms described in this book, the distribution of the simulated value at the jth iterate, θ^j, will converge to a draw from the posterior distribution as j approaches infinity. Unfortunately, this theoretical result provides no practical guidance on how to decide if the simulated sample provides a reasonable approximation to the posterior density $g(\theta|\text{data})$.

In typical practice, one monitors the performance of an MCMC algorithm by inspecting the value of the acceptance rate, constructing graphs, and computing diagnostic statistics on the stream of simulated draws. We call this investigation an *MCMC output analysis*. By means of this exploratory analysis, one decides if the chain has sufficiently explored the entire posterior distribution (there is good *mixing*) and the sequence of draws has approximately converged. If one has a sample from the posterior distribution, then one wishes to obtain a sufficient number of draws so that one can accurately estimate any particular summary of the posterior of interest.

In this section we briefly describe some of the important issues in interpreting MCMC output and describe a few graphical and numerical diagnostics for assessing convergence. One issue in understanding MCMC output is detecting the size of the burn-in period. The simulated values of θ obtained at the beginning of an MCMC run are not distributed from the posterior distribution. However, after some number of iterations have been performed (the burn-in period), the effect of the initial values wears off and the distribution of the new iterates approaches the true posterior distribution. One way of estimating the length of the burn-in period is to examine *trace plots* of simulated values of a component or particular function of θ against the iteration number. Trace plots are especially important when MCMC algorithms are initialized with parameter values that are far from the center of the posterior distribution.

A second concern in analyzing output from MCMC algorithms is the degree of autocorrelation in the sampled values. In both the Metropolis and Gibbs sampling algorithms, the simulated value of θ at the $(j+1)$st iteration is dependent on the simulated value at the jth iteration. If there is strong correlation between successive values in the chain, then two consecutive values provide only marginally more information about the posterior distribution than a single simulated draw. Also, a strong correlation between successive iterates may prevent the algorithm from exploring the entire region of the parameter space. A standard statistic for measuring the degree of dependence between successive draws in the chain is the autocorrelation that measures the correlation between the sets $\{\theta^j\}$ and $\{\theta^{j+L}\}$, where L is the lag or number of iterates separating the two sets of values. A standard graph is to plot the values of the autocorrelation against the log L. If the chain is mixing adequately, the values of the autocorrelation will decrease to zero as the lag value is increased.

Another issue that arises in output analysis is the choice of the simulated sample size and the resulting accuracy of calculated posterior summaries. Since iterates in an MCMC algorithm are not independent, one cannot use standard "independent sample" methods to compute estimated standard errors. One simple method of computing standard errors for correlated output is the method of batch means. Suppose we estimate the posterior mean of θ_i with the summary sample mean

$$\bar{\theta}_i = \frac{\sum_{j=1}^{m} \theta_i^j}{m}.$$

What is the simulation standard error of this estimate? In the batch means method, the stream of simulated draws $\{\theta_i^j\}$ is subdivided into b batches, each batch of size v, where $m = bv$. In each batch, we compute a sample mean; call the set of sample means $\bar{\theta}_i^1, ..., \bar{\theta}_i^b$. If the lag one autocorrelation in the sequence in the batch means is small, then we can approximate the standard error of the estimate $\bar{\theta}_i$ by the standard deviation of the batch means divided by the square root of the number of batches.

6.6 A Strategy in Bayesian Computing

For a particular Bayesian inference problem, we assume that one has defined the log posterior density by an R function. Following the recommendation of Gelman et al (2003), Chapter 11, a good approach for summarizing this density is to set up a Markov chain simulation algorithm. The Metropolis-Hastings random walk and independence chains and the Gibbs sampling algorithm are attractive Markov chains since they are easy to program and require relatively little prior input. But these algorithms do require some initial guesses at the location and spread of the parameter vector θ. These initial guesses can be found by non-Bayesian methods such as the method of moments or maximum likelihood. Alternatively, one can obtain an approximation to the posterior distribution by finding the mode by use of some optimization algorithm. For example, Newton's method gives the posterior mode and an approximation to the variance-covariance matrix that can be used in specifying the proposal densities in the Metropolis-Hastings algorithms.

In our examples, we illustrate the use of the function `laplace` to locate the posterior density. We can check the accuracy of the normal approximation in the two-parameter case by the construction of a contour graph of the joint posterior. These examples show that there can be some errors in the normal approximation. But the `laplace` function is still helpful in that the values of $\hat{\theta}$ and V can be used to construct efficient Metropolis-Hastings algorithms for simulating from the exact joint posterior distribution. Once one has decided that the simulated stream of values represents an approximate sample from the posterior, then one can summarize this sample in different ways to perform inferences about θ.

6.7 Learning About a Normal Population from Grouped Data

As a first example, suppose a random sample is taken from a normal population with mean μ and standard deviation σ. But one only observes the data in "grouped" form, where the frequencies of the data in bins are recorded. For example, suppose one is interested in learning about the mean and standard deviation of the heights (in inches) of men from a local college. One is given the summary frequency data shown in Table 6.1. One sees that 14 men were shorter than 66 inches, 30 men had heights between 66 and 68 inches, and so on.

Table 6.1. Grouped frequency data for heights of male students at a college.

Height Interval (in.)	Frequency
less than 66	14
between 66 and 68	30
between 68 and 70	49
between 70 and 72	70
between 72 and 74	33
over 74	15

We are observing multinomial data with unknown bin probabilities $p_1, ..., p_6$ where the probabilities are functions of the unknown parameters of the normal population. For example, the probability that a student's height is between 66 and 68 inches is given by $p_2 = \Phi(68, \mu, \sigma) - \Phi(66, \mu, \sigma)$, where $\Phi(; \mu, \sigma)$ is the cdf of a normal(μ, σ) random variable. It is straightforward to show that the likelihood of the normal parameters given this grouped data is given by

$$\begin{aligned} L(\mu, \sigma) = {} & \Phi(66, \mu, \sigma)^{14}(\Phi(68, \mu, \sigma) - \Phi(66, \mu, \sigma))^{30} \\ & \times (\Phi(70, \mu, \sigma) - \Phi(68, \mu, \sigma))^{49}(\Phi(72, \mu, \sigma) - \Phi(70, \mu, \sigma))^{70} \\ & \times (\Phi(74, \mu, \sigma) - \Phi(72, \mu, \sigma))^{33}(1 - \Phi(74, \mu, \sigma))^{15}. \end{aligned}$$

Suppose (μ, σ) are assigned the usual noninformative prior proportional to $1/\sigma$. Then the posterior density of the parameters is proportional to

$$g(\mu, \sigma|\text{data}) \propto \frac{1}{\sigma} L(\mu, \sigma).$$

Following our general strategy, we transform the positive standard deviation by $\lambda = \log(\sigma)$ and the posterior density of (μ, λ) is given by

$$g(\mu, \lambda|\text{data}) \propto L(\mu, \exp(\lambda)).$$

We begin by writing a short function `groupeddatapost` that computes the logarithm of the posterior density of (μ, λ). There are two arguments to this function: a matrix `theta`, where each row corresponds to a value of (μ, λ), and a list `data`. The list has two components: `data$b` is a vector of cutpoints for the bins and `data$f` is a vector of bin frequencies.

```
groupeddatapost=function(theta,data)
{
cpoints=data$b
freq=data$f
nbins=length(cpoints)
m=theta[,1]; logsigma=theta[,2]
z=0*m; s=exp(logsigma)
z=freq[1]*log(pnorm(cpoints[1],m,s))
for (j in 1:(nbins-1))
  z=z+freq[j+1]*log(pnorm(cpoints[j+1],m,s)-
    pnorm(cpoints[j],m,s))
z=z+freq[nbins+1]*log(1-pnorm(cpoints[nbins],m,s))
return(z)
}
```

We begin by defining the grouped data by the list `d`.

```
> d=list(b=seq(66,74,by=2),f=c(14,30,49,70,33,15))
```

To use the function `laplace`, one requires a good guess at the location of the posterior mode. To estimate the mode of $(\mu, \log \sigma)$, we first create an artificial continuous dataset by replacing each grouped observation by its bin midpoint. Then we approximate the posterior mode by computing the sample mean and the logarithm of the standard deviation of these artificial observations.

```
> y=c(rep(65,14),rep(67,30),rep(69,49),rep(71,70),rep(73,33),
+  rep(75,15))
> mean(y)

[1] 70.16588

> log(sd(y))

[1] 0.9504117
```

Based on this computation, we believe that the posterior of the vector $(\mu, \log \sigma)$ is approximately (70, 1). We use the `laplace` function, where the log posterior is defined in the function `groupeddatapost`, `start` is set equal to this starting value, 10 iterations of Newton's method are run, and the grouped data are contained in the list `d`.

```
> start=array(c(70,1),c(1,2))
> fit=laplace(groupeddatapost,start,10,d)
> fit

$mode
         [,1]      [,2]
[1,] 70.17025 0.9736653
```

```
$var
             [,1]         [,2]
[1,] 3.534866e-02 3.747464e-05
[2,] 3.747464e-05 3.146590e-03

$int
[1] -350.6305
```

From the output, the posterior mode of $(\mu, \log \sigma)$ is found to be $(70.17, 0.97)$. The associated posterior standard deviations of the parameters can be estimated by computing the square roots of the diagonal elements of the variance-covariance matrix.

```
> modal.sds=sqrt(diag(fit$var))
```

We use the output from the function `laplace` to design a Metropolis random walk algorithm to simulate from the joint posterior. For the proposal density we use the variance-covariance matrix obtained from `laplace` and we set the scale parameter equal to 2. We run 10,000 iterations of the random walk algorithm starting at the value `start`. The output `fit2` is a list with two components: `par` is a matrix of simulated values where each row corresponds to a single draw of the parameter vector, and `accept` gives the acceptance rate of the random walk chain.

```
> proposal=list(var=fit$var,scale=2)
> fit2=rwmetrop(groupeddatapost,proposal,start,10000,d)
```

We monitor the algorithm by displaying the acceptance rate; here the value is .2937 which is close to the desired acceptance rate for this Metropolis random walk algorithm.

```
> fit2$accept

[1] 0.2826
```

We can summarize the parameters μ and $\log \sigma$ by computation of the posterior means and posterior standard deviations.

```
> post.means=apply(fit2$par,2,mean)
> post.sds=apply(fit2$par,2,sd)
```

One can assess the accuracy of the model approximation to the posterior by comparing the means and standard deviations from the function `laplace` with the values computed from the simulated output from the MCMC algorithm.

```
> cbind(c(fit$mode),modal.sds)

                modal.sds
[1,] 70.1702518 0.18801241
[2,]  0.9736653 0.05609447
```

```
> cbind(post.means,post.sds)

      post.means   post.sds
[1,] 70.1631820 0.18160482
[2,]  0.9778666 0.05404014
```

For this model, there is close agreement in the two sets of posterior moments which indicates that the modal approximation to the posterior distribution is reasonably accurate.

We confirm this statement by using the function `mycontour` to draw a contour plot of the joint posterior of μ and $\log \sigma$. The last 5000 simulated draws from the random walk Metropolis algorithm are drawn on top in Fig. 6.1. Note that the contour lines have an elliptical shape that confirms the accuracy of the normal approximation in this example.

```
> mycontour(groupeddatapost,c(69,71,.6,1.3),d)
> points(fit2$par[5001:10000,1],fit2$par[5001:10000,2])
> title(xlab="mu",ylab="log sigma")
```

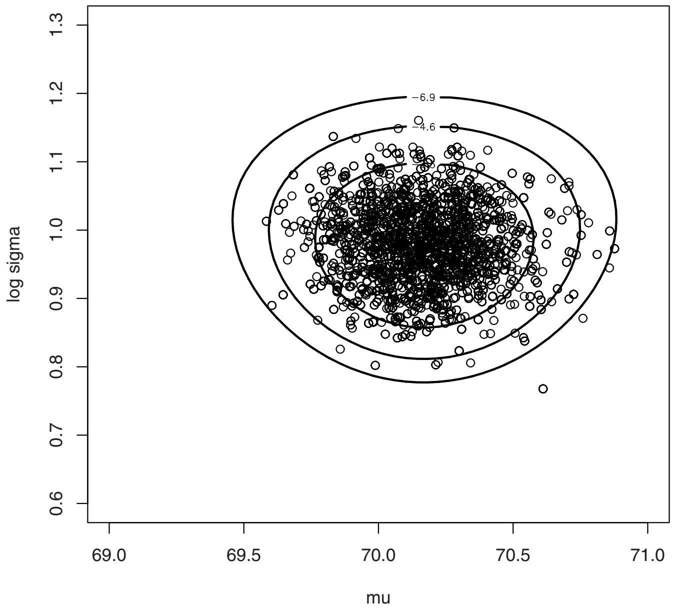

Fig. 6.1. Contour plot of posterior of μ and $\log \sigma$ for grouped data example. A simulated sample of 5000 draws of the posterior is also shown.

6.8 Example of Output Analysis

We illustrate the use of MCMC output analysis by use of the R package `boa` that will be described in Chapter 11. Suppose we rerun the Metropolis random walk algorithm for the grouped data posterior with poor choices of starting value and proposal density. As a starting value, we choose $(\mu, \log \sigma) = (65, 1)$ (the choice of μ is too small) and we select the small scale factor of 0.2 (instead of 2):

```
> start=array(c(65,1),c(1,2))
> proposal=list(var=fit$var,scale=0.2)
```

We then rerun the Metropolis function `rwmetrop`:

```
> bayesfit=rwmetrop(groupeddatapost,proposal,start,10000,d)
```

We find that the acceptance rate of this modified algorithm is 0.89 which is much larger than the 0.29 rate that we found using the scale factor 2.

Fig. 6.2 displays a trace plot of the simulated draws of μ from this Metropolis algorithm. Note that there is a significant burn-in period, approximately 600 iterations, before the simulated draws reach the main support of the posterior of μ. Also note the irregularity of the simulated sequence; for example, the iterates will explore the region where $\mu > 71$ for a while before returning to the center of the distribution.

One can observe the strong correlation structure of the sequence by the use of an autocorrelation plot shown in Fig. 6.3. The autocorrelations are close to one for lag one and reduce very slowly as a function of the lag.

The following summary output of the simulated draws of μ confirm the behavior of the MCMC run seen in Fig. 6.2 and Fig. 6.3. The estimate at the posterior mean of μ is 70.04. If we assume naively that this simulated sample represented independent draws, then the standard error of this estimate is .0069. However, a more accurate estimate at the standard error is the `Batch SE` given by .0483. The lag one autocorrelation of the batch means (using batches of size 50) is .842.

```
SUMMARY STATISTICS:
===================
Bin size for calculating Batch SE and (Lag 1) ACF = 50

Chain: mu
```

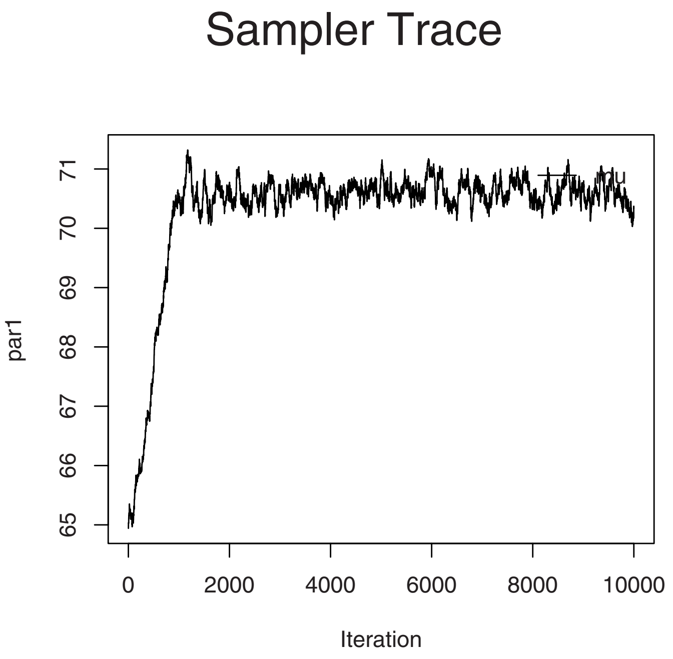

Fig. 6.2. Trace plot of simulated draws of μ for an MCMC chain with poor choices for starting value and scale factor.

```
---------
          Mean        SD    Naive SE  MC Error   Batch SE Batch ACF
par1 70.03787 0.6879849 0.006879849 0.0889096 0.04828488 0.8423536
        0.025      0.5    0.975 MinIter MaxIter Sample
par1 67.50639 70.16366 70.53974       1   10000  10000
```

It is instructive to compare these diagnostic graphs with the graphs using the better starting value and choice of proposal density used in Section 6.7. Fig. 6.4 and Fig. 6.5 display a trace plot and autocorrelation graph of the simulated draws of μ using the starting value $(\mu, \log \sigma) = (70, 1)$ and scale factor equal to 2. The trace plot of the simulated stream of μ looks more like random noise. The lag one autocorrelation is high, but the autocorrelation values dissipate rapidly as a function of the lag.

As before, we can compute summary statistics for this stream of MCMC output.

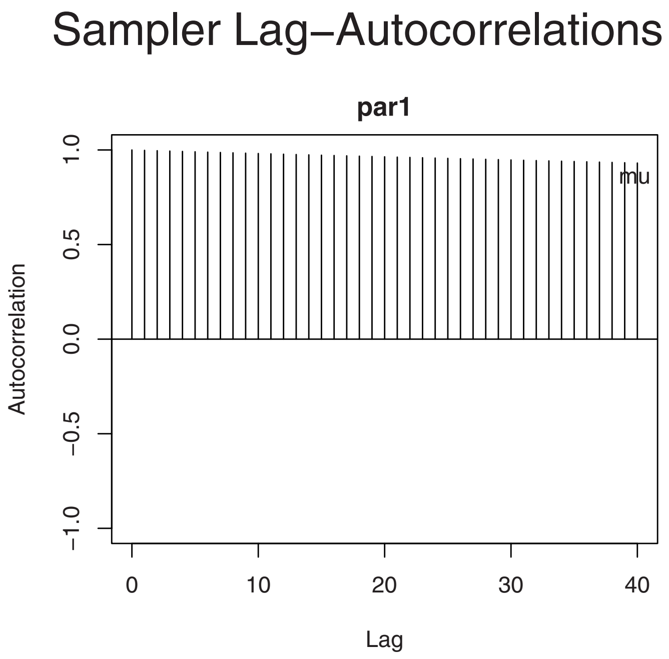

Fig. 6.3. Autocorrelation plot of simulated draws of μ for an MCMC chain with poor choices for starting value and scale factor.

```
SUMMARY STATISTICS:
===================
Bin size for calculating Batch SE and (Lag 1) ACF = 50

Chain: mu
---------
         Mean        SD    Naive SE   MC Error   Batch SE
par1 70.16019 0.2583128 0.002583128 0.01156611 0.01110407
         Batch ACF    0.025      0.5    0.975 MinIter MaxIter
par1    0.01962624 69.77265 70.17123 70.54962       1   10000
       Sample
par1   10000
```

Here the estimate of the posterior mean of μ is 70.16 with a batch standard error of .011. The autocorrelation between batch means of size 50 is the small value 0.0196. The graphs and the summary statistics confirm the better performance of the MCMC chain with a starting value $(\mu, \log \sigma) = (70, 1)$ and scale factor of 2.

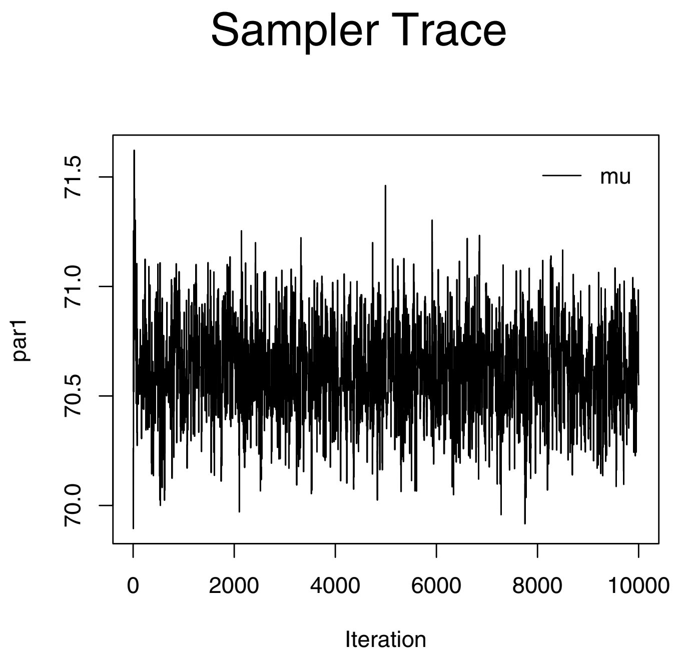

Fig. 6.4. Trace plot of simulated draws of μ for MCMC chain with good choices for starting value and scale factor.

6.9 Modeling Data with Cauchy Errors

For a second example, suppose that we are interested in modeling data where outliers may be presented. Suppose $y_1, ..., y_n$ are a random sample from a Cauchy density with location parameter μ and scale parameter σ:

$$f(y|\mu, \sigma) = \frac{1}{\pi\sigma(1+z^2)},$$

where $z = (y - \mu)/\sigma$. Suppose that we assign the usual noninformative prior to (μ, σ):

$$g(\mu, \sigma) = \frac{1}{\sigma}.$$

The posterior density of μ and σ is given, up to a proportionality constant, by

$$\begin{aligned} g(\mu, \sigma|\text{data}) &\propto \frac{1}{\sigma}\prod_{i=1}^{n} f(y_i|\mu, \sigma). \\ &= \frac{1}{\sigma}\prod_{i=1}^{n}\left[\frac{1}{\sigma}\Big(1 + (y_i - \mu)^2/\sigma^2\Big)^{-1}\right]. \end{aligned}$$

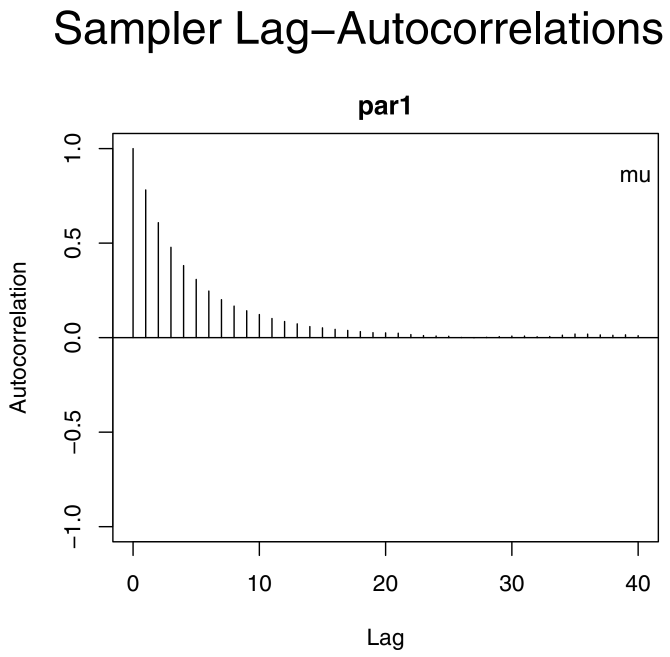

Fig. 6.5. Autocorrelation plot of simulated draws of μ for an MCMC chain with good choices for starting value and scale factor.

Again we first transform the positive parameter σ to the real line by the reexpression $\lambda = \log \sigma$, leading to the posterior density of (μ, λ):

$$g(\mu, \lambda|\text{data}) \propto \prod_{i=1}^{n} \left[\exp(-\lambda) \left(1 + \exp(-2\lambda)(y_i - \mu)^2 \right)^{-1} \right].$$

The logarithm of the density is then given, up to an additive constant, by

$$\log g(\mu, \lambda|\text{data}) = \sum_{i=1}^{n} \left[-\lambda - \log \left(1 + \exp(-2\lambda)(y_i - \mu)^2 \right) \right].$$

We write the following R function `cauchyerrorpost` to compute the logarithm of the posterior density. There are two arguments to the function: `theta`, a matrix where each row corresponds to a value of the pair (μ, λ), and the vector of observations `y`. Since the parameters are vectors, we use a loop in the function where the individual terms of the log likelihood are summed over the values of $y_1, ..., y_n$. To simplify the code, we use the R function `dt`, which computes the density of the t random variable. (The Cauchy density is the t density with a single degree of freedom.)

```
cauchyerrorpost=function(theta,y)
{
mu=theta[,1]; lambda=theta[,2]
sigma=exp(lambda)
val=0*mu
for (i in 1:length(y))
    {val=val+log(dt((y[i]-mu)/sigma,df=1)/sigma)}
return(val)
}
```

We apply this model to Darwin's famous dataset concerning 15 differences of the heights of cross- and self-fertilized plants quoted by Fisher (1960). This dataset can be found in the LearnBayes library with the name `darwin`. We read in the dataset and attach the dataframe so we can access the variable `difference`. We initially compute the mean and logarithm of the standard deviation of the data to get some initial estimates at the locations of the posterior distributions of μ and $\lambda = \log(\sigma)$.

```
> data(darwin)
> attach(darwin)
> mean(difference)

[1] 21.66667

> log(sd(difference))

[1] 3.65253
```

To find the posterior mode, we use the function `laplace`. The arguments are the name of the function `cauchyerrorpost` defining the log posterior density, an array of initial estimates at the parameters, the number of iterations of the Newton-Raphson algorithm, and the data used in the log posterior function. For initial estimates, we use the values $\mu = 21.6, \lambda = 3.6$ found earlier, and we use 10 iterations of the algorithm.

```
> laplace(cauchyerrorpost,array(c(21.6,3.6),c(1,2)),10,difference)

$mode
         [,1]     [,2]
[1,] 24.70160 2.772829

$var
           [,1]      [,2]
[1,] 34.9647321 0.3672069
[2,]  0.3672069 0.1378207

$int
[1] -73.24035
```

The posterior mode is given by $(\mu, \lambda) = (24.7, 2.77)$. The output also gives the associated variance-covariance matrix and an estimate at the log integral.

We can use these estimates of center and spread to construct a rectangle that covers essentially all of the posterior probability of the parameters. As an initial guess at this rectangle, we take for each parameter the posterior mode plus and minus four standard deviations, where the standard deviations are obtainable from the diagonal elements of the variance-covariance matrix.

```
> c(24.7-4*sqrt(34.96),24.7+4*sqrt(34.96))

[1]  1.049207 48.350793

> c(2.77-4*sqrt(.138),2.77+4*sqrt(.138))

[1] 1.284066 4.255934
```

After some trial and error, we use the rectangle $\mu \in (-10, 60), \lambda \in (1, 4.5)$ as the bounding rectangle for the function `mycontour`. Fig. 6.6 displays the contour graph of the exact posterior distribution.

```
> mycontour(cauchyerrorpost,c(-10,60,1,4.5),difference)
> title(xlab="mu",ylab="log sigma")
```

The contours of the exact posterior distribution have an interesting shape and one may wonder how these contours compare to those for a bivariate normal approximation. In the R code, we rerun the `laplace` function to obtain the posterior mode `t$mode` and associated variance-covariance matrix `t$var`. Using these values as inputs, we draw contours of a bivariate normal density in Fig. 6.7 where the log bivariate normal density is programmed in the function `lbinorm`. The eliptical shape of these normal contours seems significantly different from the shape of the exact posterior contours, which indicates that the normal approximation may be inadequate.

```
> fitlaplace=laplace(cauchyerrorpost,array(c(21.6,3.6),c(1,2)),
+   10,difference)
> mycontour(lbinorm,c(-10,60,1,4.5),list(m=fitlaplace$mode,
+   v=fitlaplace$var))
> title(xlab="mu",ylab="log sigma")
```

Although the normal approximation may not be the best summary of the posterior distribution, the estimated variance-covariance matrix is helpful in setting up a Metropolis random walk chain. We initially define a list `proposal` that contains the estimated variance-covariance matrix and a scale factor. We define the starting value of the chain in the array `start`. The simulation algorithm is run using the function `rwmetrop`. The inputs are the function defining the log posterior, the list `proposal`, the starting value, the number of simulations, and the data vector.

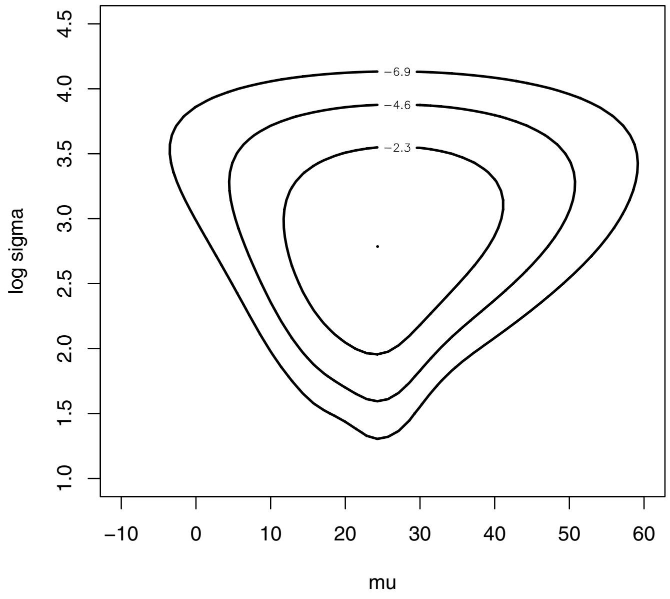

Fig. 6.6. Contour plot of posterior of μ and $\log\sigma$ for Cauchy error model problem.

```
> proposal=list(var=fitlaplace$var,scale=2.5)
> start=array(c(20,3),c(1,2))
> m=1000
> s=rwmetrop(cauchyerrorpost,proposal,start,m,difference)
> mycontour(cauchyerrorpost,c(-10,60,1,4.5),difference)
> title(xlab="mu",ylab="log sigma")
> points(s$par[,1],s$par[,2])
```

In Fig. 6.8 we display simulated draws from `rwmetrop` on top of the contour graph.

Fig. 6.9 and Fig. 6.10 show the "exact" marginal posterior densities of μ and $\log\sigma$ found from a density estimate from 50,000 simulated draws from the random walk algorithm. Also each figure shows the approximate normal approximation from the `laplace` output. These figures demonstrate the non-normal shape of these marginal posteriors.

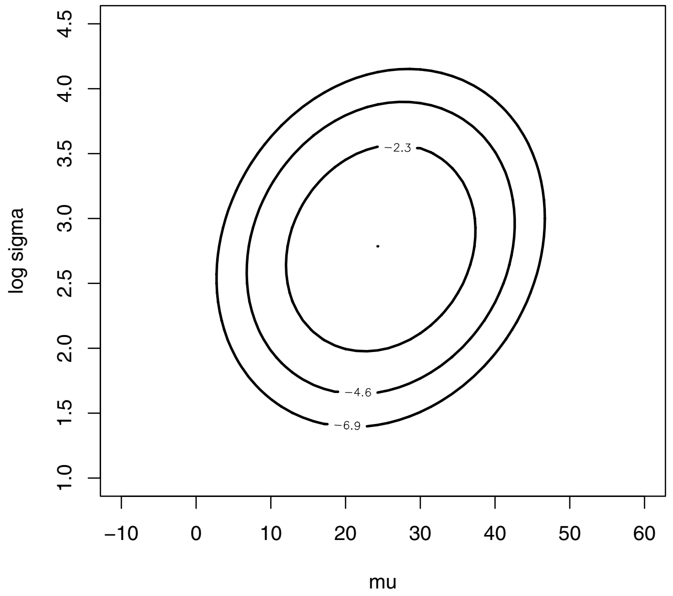

Fig. 6.7. Contour plot of normal approximation to posterior of μ and $\log \sigma$ for Cauchy error model problem.

It is instructive to illustrate "brute-force" and other Metropolis-Hastings algorithms for this problem. The brute-force algorithm is based on simulating draws of $(\mu, \log \sigma)$ from the grid using the function `simcontour`. One can use a Metropolis-Hastings independence chain, where the proposal density is multivariate normal with mean and variance given by the normal approximation. Alternatively, one can apply a Gibbs sampling algorithm with a vector of scale parameters equal to (12, .75); these values are approximately equal to twice the estimated posterior standard deviations of the two parameters. All the simulation algorithms were run with a simulation sample size of 50,000. The R code for the implementation of the four simulation algorithms follows.

```
> fitgrid=simcontour(cauchyerrorpost,c(-10,60,1,4.5),difference,
+  50000)
> proposal=list(var=fitlaplace$var,scale=2.5)
> start=array(c(20,3),c(1,2))
> fitrw=rwmetrop(cauchyerrorpost,proposal,start,50000,
+  difference)
> proposal2=list(var=fitlaplace$var,mu=t(fitlaplace$mode))
> fitindep=indepmetrop(cauchyerrorpost,proposal2,start,50000,
```

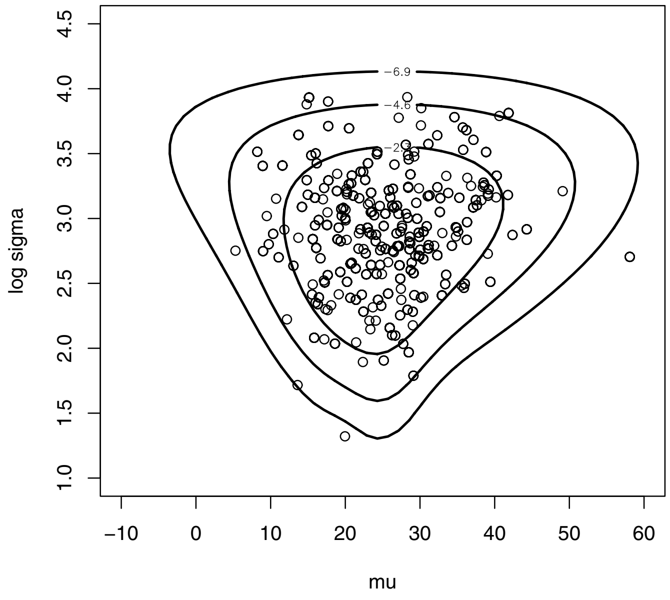

Fig. 6.8. Contour plot of posterior of μ and $\log\sigma$ with simulated sample for Cauchy error model problem.

```
+  difference)
> fitgibbs=gibbs(cauchyerrorpost,start,50000,c(12,.75),
+  difference)
```

The simulated draws for a parameter can be summarized by the computation of the 5th, 50th, and 95th percentiles. For example, one can find the summaries of μ and $\log\sigma$ from the random walk simulation by the command

```
> apply(fitrw$par,2,mean)

[1] 25.562859  2.843484

> apply(fitrw$par,2,sd)

[1] 7.175004 0.372534
```

Table 6.2 displays the estimated posterior quantiles for all of the algorithms described in this chapter. In addition, the acceptance rates for the Metropolis-Hastings random walk and independence chains and the Gibbs sampler are shown. Generally there is agreement among the simulation-based methods and these "exact" posterior summaries are different from the quantiles found using

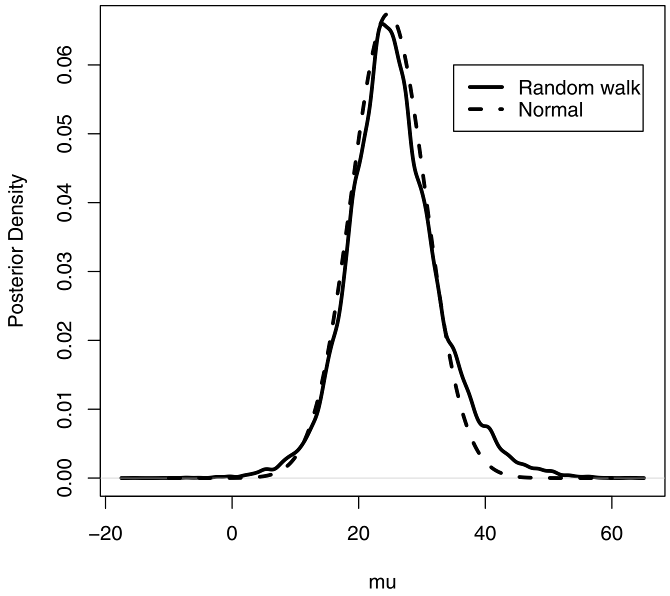

Fig. 6.9. Posterior density of μ using normal approximation and simulated draws from the Metropolis random walk chain.

the Laplace normal approximation. The exact marginal posterior distribution of μ has heavier tails than suggested by the normal approximation; also there is some skewness in the marginal posterior distribution of $\log \sigma$.

Table 6.2. Summaries of the marginal posterior densities of μ and $\log \sigma$ using five computational methods.

Method	Acceptance Rate	μ	$\log \sigma$
Normal approx.		(15.0, 24.7, 34.4)	(2.16, 2.77, 3.38)
Brute force		(14.5, 25.1, 37.7)	(2.22, 2.85, 3.45)
Random walk	.231	(14.8, 25.1, 38.0)	(2.23, 2.85, 3.45)
Independence	.849	(14.4, 25.0, 37.1)	(2.22, 2.85, 3.44)
Gibbs	(.318, .314)	(14.5, 25.2, 38.0)	(2.20, 2.86, 3.45)

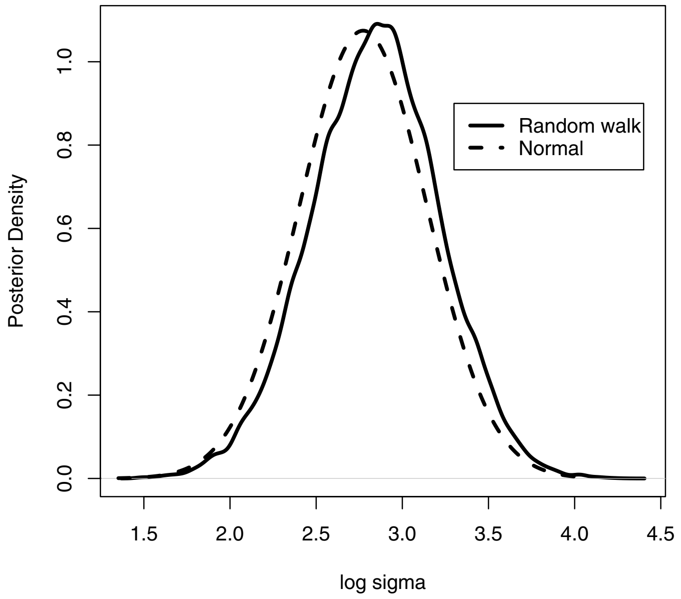

Fig. 6.10. Posterior density of $\log \sigma$ using normal approximation and simulated draws from the Metropolis random walk chain.

6.10 Analysis of the Stanford Heart Transplant Data

Turnbull et al (1974) describe a number of approaches for analyzing heart transplant data from the Stanford Heart Transplanation Program. One of the inferential goals is to decide if heart transplantation extends a patient's life. One of their models, the Pareto model, assumes individual patients in the nontransplant group have exponential lifetime distributions with mean $1/\theta$, where θ is assumed to vary between patients and is drawn from a gamma distribution with density

$$f(\theta) = \frac{\lambda^p}{\Gamma(p)} \theta^{p-1} \exp(-\lambda\theta).$$

Patients in the transplant group have a similar exponential lifetime distribution where the mean is $1/(\theta\tau)$. This model assumes that the patient's risk of death changes by an unknown constant factor $\tau > 0$. If $\tau = 1$, then there is no increased risk by having a transplant operation.

Suppose the survival times $\{x_i\}$ are observed for N nontransplant patients. For n of these patients, x_i represents the actual survival time (in days); the remaining $N-n$ patients were still alive at the end of the study, so x_i represents the censoring time. For the M patients that have a heart transplant, let y_j and z_j denote the time to transplant and survival time; m of these patients died during the study. The unknown parameter vector is (τ, λ, p) with likelihood function given by

$$L(\tau, \lambda, p) = \prod_{i=1}^{n} \frac{p\lambda^p}{(\lambda + x_i)^{p+1}} \prod_{i=n+1}^{N} \left(\frac{\lambda}{\lambda + x_i}\right)^p$$
$$\times \prod_{j=1}^{m} \frac{\tau p\lambda^p}{(\lambda + y_j + \tau z_j)^{p+1}} \prod_{j=m+1}^{M} \left(\frac{\lambda}{\lambda + y_j + \tau z_j}\right)^p,$$

where all the parameters are positive. Suppose we place a uniform prior on (τ, λ, p), and so the posterior density is proportional to the likelihood.

Following our summarization strategy, we transform the parameters by logs:

$$\theta_1 = \log \tau, \; \theta_2 = \log \lambda, \; \theta_3 = \log p.$$

The posterior density of $\theta = (\theta_1, \theta_2, \theta_3)$ is given by

$$g(\theta|\text{data}) \propto L(\exp(\theta_1), \exp(\theta_2), \exp(\theta_3)) \prod_{i=1}^{3} \exp(\theta_i).$$

The dataset `stanfordheart` in the LearnBayes package contains the data for 82 patients; for each patient, there are four variables: `survtime`, the survival time; `transplant`, a variable that is 1 or 0 if the patient had a transplant or not; `timetotransplant`, the time a transplant patient waits for the operation; and `state`, a variable that indicates if the survival time was censored (0 if the patient died and 1 if he was still alive). We load this datafile into R.

```
> data(stanfordheart)
```

We write a function `transplantpost` that computes a value of the log posterior. In the following code, we generally follow the earlier notation. The numbers of nontransplant and transplant patients are denoted by `N` and `M`. We divide the data into two groups by the transplant indicator variable `t`. For the nontransplant patients, the survival times and censoring indicators are denoted by `xnt` and `dnt`, and for the transplant patients, the waiting times, survival times, and censoring indicators are denoted by `y`, `z`, and `dt`.

```
transplantpost=function(theta,data)
{
x=data[,1]    #  survival time
y=data[,3]    #  time to transplant
t=data[,2]    #  transplant indicator
d=data[,4]    #  censoring indicator (d = 0 if died)
tau=exp(theta[,1])
lambda=exp(theta[,2])
p=exp(theta[,3])
val=0*tau
xnt=x[t==0]; dnt=d[t==0]
z=x[t==1]; y=y[t==1]; dt=d[t==1]
N=length(xnt)
M=length(z)
for (i in 1:N)
  val=val+(dnt[i]==0)*(p*log(lambda)+log(p)-
          (p+1)*log(lambda+xnt[i]))+
          (dnt[i]==1)*p*log(lambda/(lambda+xnt[i]))
for (i in 1:M)
  val=val+(dt[i]==0)*(p*log(lambda)+log(p*tau)-
          (p+1)*log(lambda+y[i]+tau*z[i]))+
          (dt[i]==1)*p*log(lambda/(lambda+y[i]+tau*z[i]))
val=val+theta[,1]+theta[,2]+theta[,3]
return(val)
}
```

To get an initial idea about the location of the posterior, we run the function `laplace`. Our initial estimate at the posterior mode is $\theta = (0, 3, -1)$ and we run 10 Newton steps. The algorithm converges and we obtain the posterior mode and an estimate at the variance-covariance matrix.

```
> start=array(c(0,3,-1),c(1,3))
> laplacefit=laplace(transplantpost,start,10,stanfordheart)
> laplacefit

$mode
           [,1]    [,2]      [,3]
[1,] -0.0924209 3.38503 -0.722881

$var
            [,1]        [,2]        [,3]
[1,]  0.17275867 -0.00925073 -0.04994602
[2,] -0.00925073  0.21467648  0.09300626
[3,] -0.04994602  0.09300626  0.06893108

$int
[1] -376.2505
```

We use a Metropolis random walk algorithm (implemented in the function `rwmetrop`) to simulate from the posterior. We use a proposal variance of $2V$, where V is the estimated variance-covariance matrix from the Laplace fit. We run the simulation for 10,000 iterations and as the output indicates, the acceptance rate was equal to 19%.

```
> proposal=list(var=laplacefit$var,scale=2)
> s=rwmetrop(transplantpost,proposal,start,10000,stanfordheart)
> s$accept
[1] 0.1893
```

One primary inference in this problem is to learn about the three parameters τ, λ, and p. Fig. 6.11 displays density estimates of the simulated draws from the marginal posterior densities of each parameter. These are simply obtained by exponentiating the simulated draws of θ that are output from the function `rwmetrop`. For example, the first plot in Fig. 6.11 is constructed by first computing the simulated draws of τ and then using the `plot(density())` command.

```
> tau=exp(s$par[,1])
> plot(density(tau),main="TAU")
```

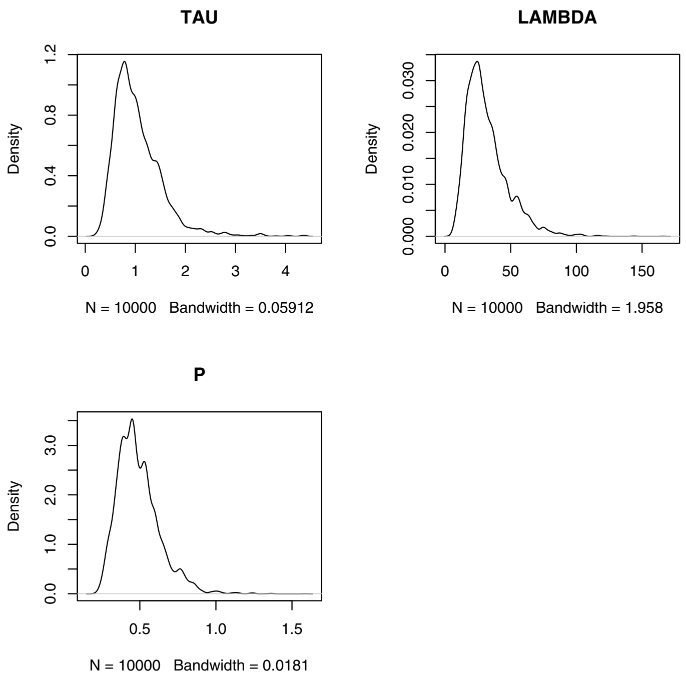

Fig. 6.11. Posterior densities of parameters τ, λ, and p in Pareto survival model.

We can summarize the parameters τ, λ, and p by computing the 5th, 50th, and 95th percentiles of the simulated draws by the `apply` command.

```
> apply(exp(s$par),2,quantile,c(.05,.5,.95))
```

```
         [,1]     [,2]      [,3]
5%  0.4720614 13.35309 0.3133939
50% 0.9562069 29.01064 0.4746410
95% 2.0703049 63.54526 0.7623879
```

From Fig. 6.11 and these summaries, we see that the value $\tau = 1$ is in the center of the posterior distribution and so there is insufficient evidence to conclude from this data that $\tau \neq 1$. This means that there is insufficient evidence to conclude that the risk of death is higher (or lower) with a transplant operation.

In this problem, one is typically interested in estimating a patient's survival curve. For a nontransplant patient, the survival function is equal to

$$S(t) = \frac{\lambda^p}{(\lambda + t)^p}, \; t > 0.$$

For a given value of the time t_0, one can compute a sample from the posterior distribution of $S(t_0)$ by computing the function $\lambda^p/(\lambda+t_0)^p$ from the simulated values from the joint posterior distribution of λ and p. In the following code, we assume that simulated samples from the marginal posterior distributions of λ and p are stored in the vectors `lambda` and `p`, respectively. Then we (1) set up a grid of values of t and storage vectors `p5`, `p50`, and `p95`; (2) simulate a sample of values of $S(t)$ for each value of t on the grid; and (3) summarize the posterior sample by the computation of the 5th, 50th, and 95th percentiles. These percentiles are stored in the variables `p5`, `p50`, and `p95`. In Fig. 6.12, we graph these percentiles as a function of the time variable t. Since there is little evidence that $\tau \neq 1$, this survival curve represents the risk for both transplant and nontransplant patients.

```
> p=exp(s$par[,3])
> lambda=exp(s$par[,2])
> t=seq(1,240)
> p5=0*t; p50=0*t; p95=0*t
> for (j in 1:240)
+ { S=(lambda/(lambda+t[j]))^p
+   q=quantile(S,c(.05,.5,.95))
+   p5[j]=q[1]; p50[j]=q[2]; p95[j]=q[3]}
> plot(t,p50,type="l",ylim=c(0,1),ylab="Prob(Survival)",
+   xlab="time")
> lines(t,p5,lty=2)
> lines(t,p95,lty=2)
```

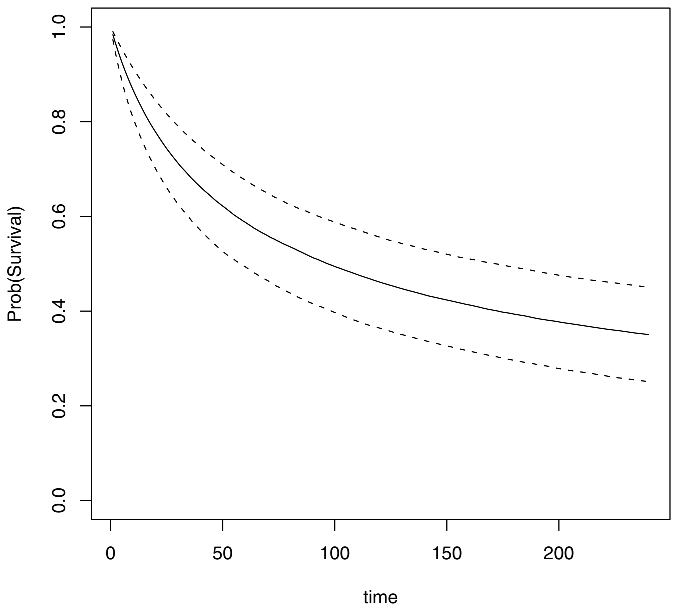

Fig. 6.12. Posterior distribution of probability of survival $S(t)$ for heart transplant patients. Lines correspond to the 5th, 50th, and 95th percentiles of the posterior of $S(t)$ for each time t.

6.11 Further Reading

A good overview of discrete Markov chains is contained in Kemeny and Snell (1976). Since MCMC algorithms currently play a central role in applied Bayesian inference, most modern textbooks devote significant content to these methods. Chapter 11 of Gelman et al (2003) and chapter 5 of Carlin and Louis (2000) provide good introductions to MCMC methods and their application in Bayesian methods. Robert and Casella (2004) and Givens and Hoeting (2005) give more detailed descriptions of MCMC algorithms within the context of computational statistical methods. Introductory discussions of Metropolis and Gibbs sampling are provided, respectively, in Chib and Greenberg (1995) and Casella and George (1992).

6.12 Summary of R Functions

`cauchyerrorpost` – computes the log posterior density of (M,log S) when a sample is taken from a Cauchy density with location M and scale S and a uniform prior distribution is taken on (M, log S)
Usage: `cauchyerrorpost(theta, data)`
Arguments: `theta`, matrix of parameter values where each row represents a value of (M, log S); `data`, vector containing sample of observations
Value: vector of values of the log posterior where each value corresponds to each row of the parameters in theta

`gibbs` – implements a Metropolis within Gibbs algorithm for an arbitrary real-valued posterior density defined by the user
Usage: `gibbs(logpost,start,m,scale,data)`
Arguments: `logpost`, function defining the log posterior density; `start`, array with a single row that gives the starting value of the parameter vector; `m`, the number of iterations of the Gibbs sampling algorithm; `scale`, vector of scale parameters for the random walk Metropolis steps; `data`, data used in the function logpost
Value: `par`, a matrix of simulated values where each row corresponds to a value of the vector parameter; `accept`, vector of acceptance rates of the Metropolis steps of the algorithm

`groupeddatapost` – computes the log posterior for (M, log S), when sampling from a normal density and the data are recorded in grouped format
Usage: `groupeddatapost=function(theta,data)`
Arguments: `theta`, matrix of parameter values where each row represents a value of (M, log S); `data`, list with components `b`, a vector of midpoints, and `f`, the corresponding bin frequencies
Value: vector of values of the log posterior where each value corresponds to each row of the parameters in theta

`indepmetrop` – simulates iterates of a Metropolis independence chain for an arbitrary real-valued posterior density defined by the user
Usage: `indepmetrop(logpost,proposal,start,m,data)`
Arguments: `logpost`, function defining the log posterior density; `proposal`, a list containing `mu`, an estimated mean and `var`, an estimated variance-covariance matrix of the normal proposal density; `start`, array with a single row that gives the starting value of the parameter vector; `m`, the number of iterations of the chain `data`, data used in the function logpost
Value: `par`, a matrix of simulated values where each row corresponds to a value of the vector parameter; `accept`, the acceptance rate of the algorithm.

`lbinorm` – computes the logarithm of a bivariate normal density
Usage: `lbinorm(xy,par)`
Arguments: `xy`, matrix of values where each row corresponds to a value of (x, y); `par`, list containing `m`, a vector of means, and `v`, a variance-covariance matrix

Value: vector of values of the kernel of the log density function

`rwmetrop` – simulates iterates of a random walk Metropolis chain for an arbitrary real-valued posterior density defined by the user
Usage: `rwmetrop(logpost,proposal,start,m,par)`
Arguments: `logpost`, function defining the log posterior density; `proposal`, a list containing `var`, an estimated variance-covariance matrix, and `scale`, the Metropolis scale factor; `start`, array with a single row that gives the starting value of the parameter vector; `m`, the number of iterations of the chain; `par`, data used in the function logpost
Value: `par`, a matrix of simulated values where each row corresponds to a value of the vector parameter; `accept`, the acceptance rate of the algorithm

`transplantpost` – computes the log posterior for (log tau, log lambda, log p) for a Pareto model for survival data
Usage: `transplantpost=function(theta,data)`
Arguments: `theta`, matrix of parameter values where each row represents a value of (log tau, log lambda, log p); `data`, data matrix where columns are survival time, time to transplant, transplant indicator, and censoring indicator
Value: vector of values of the log posterior where each value corresponds to each row of the parameters in theta

6.13 Exercises

1. **A random walk**
   The following matrix represents the transition matrix for a random walk on the integers {1, 2, 3, 4, 5}.

$$T = \begin{bmatrix} .2 & .8 & 0 & 0 & 0 \\ .2 & .2 & .6 & 0 & 0 \\ 0 & .4 & .2 & .4 & 0 \\ 0 & 0 & .6 & .2 & .2 \\ 0 & 0 & 0 & .8 & .2 \end{bmatrix}$$

   a) Suppose one starts at the location 1. By use of the `sample` command, simulate 1000 steps of the Markov chain using the probabilities given in the transition matrix. Store the locations of the walk in a vector.
   b) Compute the relative frequencies of the walker in the five states from the simulation output. Guess at the value of the stationary distribution vector w.
   c) Confirm that your guess is indeed the stationary distribution by the matrix computation `w %*% T`.
2. **Estimating a log-odds with a normal prior**
   In Exercise 1 of Chapter 5, we considered the estimation of a log-odds parameter when y is binomial(n, p) and the log-odds $\theta = \log(p/(1-p))$

is distributed $N(\mu, \sigma)$ with $\mu = 0$ and $\sigma = .25$. The coin was tossed $n = 5$ times and $y = 5$ heads were observed.

Use a Metropolis-Hastings random walk algorithm to simulate from the posterior density. In the algorithm, let s be equal to twice the approximate posterior standard deviation found in the normal approximation. Use the simulation output to approximate the posterior mean and standard deviation of θ, and the posterior probability that θ is positive. Compare your answers with those obtained by the normal approximation in Exercise 1 of Chapter 5.

3. **Genetic linkage model from Rao (2002)**
   In Exercise 2 of Chapter 5, we considered the estimation of a parameter θ in a genetic linkage model. The posterior density was expressed in terms of the real-valued logit $\eta = \log(\theta/(1-\theta))$.
   a) Use a Metropolis-Hastings random walk algorithm to simulate from the posterior density of η. (Choose the scale parameter s to be twice the approximate posterior standard deviation of η found in a normal approximation.) Compare the histogram of the simulated output of η with the normal approximation. From the simulation output, find a 95% interval estimate for the parameter of interest θ.
   b) Use a Metropolis-Hastings independence algorithm to simulate from the posterior density of η. Use a normal proposal density. Again compare the histogram of the simulated output with the normal approximation and find a 95% probability interval for the parameter of interest θ.
4. **Modeling data with Cauchy errors**
   As in Section 6.8, suppose we observe $y_1, ..., y_n$ from a Cauchy density with location μ and scale σ and a noninformative prior is placed on (μ, σ). Consider the following hypothetical test scores from a class that is a mixture of good and poor students.

```
36  13  23   6  20  12  23  93
98  91  89 100  90  95  90  87
```

   The function `cauchyerrorpost` computes the log of the posterior density. A contour plot of the posterior $(\mu, \log\sigma)$ for this data is shown in Fig. 6.13.
   a) Use the `laplace` function to find the posterior mode. Check that you have indeed found the posterior mode by trying several starting values in Newton's algorithm.
   b) Use the Metropolis random walk algorithm (using the function `rwmetrop`) to simulate 1000 draws from the posterior density. Compute the posterior mean and standard deviation of μ and $\log\sigma$.

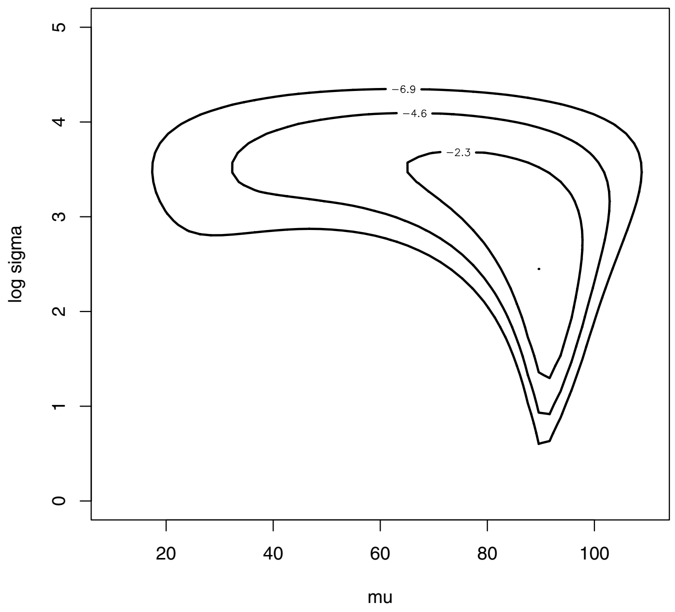

Fig. 6.13. Posterior distribution of μ and $\log \sigma$ for the Cauchy sampling exercise.

5. **Estimation for the two-parameter exponential distribution**
   Exercise 3 of Chapter 5 considered the "type I/time-truncated" life testing experiment. We are interested in the posterior density of $\theta = (\theta_1, \theta_2)$, where $\theta_1 = \log \beta, \theta_2 = \log(t_1 - \mu)$.
   a) Using the posterior mode and variance-covariance matrix from `laplace`, simulate 1000 values from the posterior distribution by the Metropolis random walk algorithm (function `rwmetrop`).
   b) Suppose one is interested in estimating the reliability at time t_0 defined by
   $$R(t_0) = e^{-(t_0 - \mu)/\beta}.$$
   Using your simulated values from the posterior, find the posterior mean and posterior standard deviation of $R(t_0)$ when $t_0 = 10^6$ cycles.
6. **Poisson regression**
   Exercise 4 of Chapter 5 describes an experiment from Haberman (1978) involving subjects reporting one stressful event. The number of events recalled i months before an interview y_i is distributed Poisson with mean λ_i, where the $\{\lambda_i\}$ satisfy the loglinear regression model
   $$\log \lambda_i = \beta_0 + \beta_1 i.$$

One is interested in learning about the posterior density of the regression coefficients (β_0, β_1).

a) Using the output of `laplace`, construct a Metropolis random walk algorithm for simulating from the posterior density. Use the function `rwmetrop` to simulate 1000 iterates and compute the posterior mean and standard deviation of β_1.
b) Construct a Metropolis independence algorithm and use the function `rwindep` to simulate 1000 iterates from the posterior. Compute the posterior mean and standard deviation of β_1.
c) Use a table such as Table 6.2 to compare the posterior estimates using the three computational methods.

7. **Generalized logit model**
Carlin and Louis (2000) describe the use of a generalized logit model to fit dose-mortality data from Bliss (1935). Table 6.3 records the number of adult flour beetles killed after five hours of exposure to various levels of gaseous carbon disulphide. The number of insects killed y_i under dose w_i is assumed binomial(n_i, p_i), where the probability p_i of death is given by

$$p_i = \left(\frac{\exp(x_i)}{1+\exp(x_i)}\right)^{m_1},$$

where $x_i = (w_i - \mu)/\sigma$. The prior distributions for μ, σ, m_1 are assumed independent, where μ is assigned a uniform prior, σ is assigned a prior proportional to $1/\sigma$, and m_1 is gamma with parameters a_0 and b_0. In the example, the prior hyperparameters of $a_0 = .25$ and $b_0 = 4$ were used. If one transforms to the real-valued parameters $(\theta_1, \theta_2, \theta_3) = (\mu, \log\sigma, \log m_1)$, then Carlin and Louis (2000) show the posterior density is given by

$$g(\theta|\text{data}) \propto \prod_{i=1}^{8} \left[\, p_i^{y_i}(1-p_i)^{n_i-y_i} \,\right] \exp(a_0\theta_3 - e^{\theta_3}/b_0).$$

Table 6.3. Flour beetle mortality data

Dosage w_i	Number Killed y_i	Number Exposed n_i
1.6907	6	59
1.7242	13	60
1.7552	18	62
1.7842	28	56
1.8113	52	63
1.8369	53	59
1.8610	61	62
1.8839	60	60

a) Write an R function that defines the log posterior of $(\theta_1, \theta_2, \theta_3)$.
b) Carlin and Louis (2000) suggest running a Metropolis random walk chain with a multivariate normal proposal density where the variance-covariance matrix is diagonal with elements 0.00012, 0.033, and 0.10. Use the function `rwmetrop` to run this chain for 10,000 iterations. Compute the acceptance rate and the 5th and 95th percentiles for each parameter.
c) Run the function `laplace` to get a non-diagonal estimate of the variance-covariance matrix. Use this estimate in the proposal density of `rwmetrop` and run the chain for 10,000 iterations. Compute the acceptance rate and the 5th and 95th percentiles for each parameter.
d) Compare your answers in parts (b) and (c).