6

Metropolis–Hastings Algorithms

"How absurdly simple!", I cried.
"Quite so!", said he, a little nettled. "Every problem becomes very childish when once it is explained to you."

Arthur Conan Doyle
The Adventure of the Dancing Men

Reader's guide

This chapter is the first of a series of two on simulation methods based on *Markov chains*. Although the Metropolis–Hastings algorithm can be seen as one of the most general Markov chain Monte Carlo (MCMC) algorithms, it is also one of the simplest both to understand and explain, making it an ideal algorithm to start with.

This chapter begins with a quick refresher on Markov chains, just the basics needed to understand the algorithms. Then we define the Metropolis–Hastings algorithm, focusing on the most common versions of the algorithm. We end up discussing the calibration of the algorithm via its acceptance rate in Section 6.5.

C.P. Robert, G. Casella, *Introducing Monte Carlo Methods with R*, Use R,
DOI 10.1007/978-1-4419-1576-4_6, © Springer Science+Business Media, LLC 2010

6.1 Introduction

For reasons that will become clearer as we proceed, we now make a fundamental shift in the choice of our simulation strategy. Up to now we have typically generated *iid* variables directly from the density of interest f or indirectly in the case of importance sampling. The Metropolis–Hastings algorithm introduced below instead generates *correlated* variables from a Markov chain. The reason why we opt for such a radical change is that Markov chains carry different convergence properties that can be exploited to provide easier proposals in cases where generic importance sampling does not readily apply. For one thing, the requirements on the target f are quite minimal, which allows for settings where very little is known about f. Another reason, as illustrated in the next chapter, is that this Markov perspective leads to efficient decompositions of high-dimensional problems in a sequence of smaller problems that are much easier to solve.

Thus, be warned that this is a pivotal chapter in that we now introduce a totally new perspective on the generation of random variables, one that has had a profound effect on research and has expanded the application of statistical methods to solve more difficult and more relevant problems in the last twenty years, even though the origins of those techniques are tied with those of the Monte Carlo method in the remote research center of Los Alamos during the Second World War. Nonetheless, despite the recourse to Markov chain principles that are briefly detailed in the next section, the implementation of these new methods is not harder than those of earlier chapters, and there is no need to delve any further into Markov chain theory, as you will soon discover. (Most of your time and energy will be spent in designing and assessing your MCMC algorithms, just as for the earlier chapters, not in establishing convergence theorems, so take it easy!)

6.2 A peek at Markov chain theory

ϟ This section is intended as a minimalist refresher on Markov chains in order to define the vocabulary of Markov chains, nothing more. In case you have doubts or want more details about these notions, you are strongly advised to check a more thorough treatment such as Robert and Casella (2004, Chapter 6) or Meyn and Tweedie (1993) since no theory of convergence is provided in the present book.

A Markov chain $\{X^{(t)}\}$ is a sequence of dependent random variables

$$X^{(0)}, X^{(1)}, X^{(2)}, \ldots, X^{(t)}, \ldots$$

such that the probability distribution of $X^{(t)}$ given the past variables depends only on $X^{(t-1)}$. This conditional probability distribution is called a *transition kernel* or a *Markov kernel* K; that is,

$$X^{(t+1)} \mid X^{(0)}, X^{(1)}, X^{(2)}, \ldots, X^{(t)} \sim K(X^{(t)}, X^{(t+1)})\,.$$

For example, a simple *random walk* Markov chain satisfies

$$X^{(t+1)} = X^{(t)} + \epsilon_t\,,$$

where $\epsilon_t \sim \mathcal{N}(0,1)$, independently of $X^{(t)}$; therefore, the Markov kernel $K(X^{(t)}, X^{(t+1)})$ corresponds to a $\mathcal{N}(X^{(t)}, 1)$ density.

For the most part, the Markov chains encountered in Markov chain Monte Carlo (MCMC) settings enjoy a very strong stability property. Indeed, a *stationary probability distribution* exists by construction for those chains; that is, there exists a probability distribution f such that if $X^{(t)} \sim f$, then $X^{(t+1)} \sim f$. Therefore, formally, the kernel and stationary distribution satisfy the equation

$$\int_{\mathcal{X}} K(x,y) f(x) \mathrm{d}x = f(y). \tag{6.1}$$

The existence of a stationary distribution (or *stationarity*) imposes a preliminary constraint on K called *irreducibility* in the theory of Markov chains, which is that the kernel K allows for free moves all over the stater-space, namely that, no matter the starting value $X^{(0)}$, the sequence $\{X^{(t)}\}$ has a positive probability of eventually reaching any region of the state-space. (A sufficient condition is that $K(x,\cdot) > 0$ everywhere.) The existence of a stationary distribution has major consequences on the behavior of the chain $\{X^{(t)}\}$, one of which being that most of the chains involved in MCMC algorithms are *recurrent*, that is, they will return to any arbitrary nonnegligible set an infinite number of times.

Exercise 6.1 Consider the Markov chain defined by $X^{(t+1)} = \varrho X^{(t)} + \epsilon_t$, where $\epsilon_t \sim \mathcal{N}(0,1)$. Simulating $X^{(0)} \sim \mathcal{N}(0,1)$, plot the histogram of a sample of $X^{(t)}$ for $t \leq 10^4$ and $\varrho = .9$. Check the potential fit of the stationary distribution $\mathcal{N}(0, 1/(1-\varrho^2))$.

In the case of recurrent chains, the stationary distribution is also a *limiting distribution* in the sense that the limiting distribution of $X^{(t)}$ is f for almost any initial value $X^{(0)}$. This property is also called *ergodicity*, and it obviously has major consequences from a simulation point of view in that, if a given kernel K produces an ergodic Markov chain with stationary distribution f, generating a chain from this kernel K will eventually produce simulations from f. In particular, for integrable functions h, the standard average

$$\frac{1}{T}\sum_{t=1}^{T} h(X^{(t)}) \longrightarrow \mathbb{E}_f[h(X)]\,, \tag{6.2}$$

which means that the Law of Large Numbers that lies at the basis of Monte Carlo methods (Section 3.2) can also be applied in MCMC settings. (It is then sometimes called the *Ergodic Theorem*.)

We won't dabble any further into the theory of convergence of MCMC algorithms, relying instead on the guarantee that standard versions of these algorithms such as the Metropolis–Hastings algorithm or the Gibbs sampler are almost always theoretically convergent. Indeed, the real issue with MCMC algorithms is that, despite those convergence guarantees, the practical implementation of those principles may imply a very lengthy convergence time or, worse, may give an impression of convergence while missing some important aspects of f, as discussed in Chapter 8.

There is, however, one case where convergence never occurs, namely when, in a Bayesian setting, the posterior distribution is not proper (Robert, 2001) since the chain cannot be recurrent. With the use of improper priors $f(x)$ being quite common in complex models, there is a possibility that the product likelihood $\times$ prior, $\ell(x) \times f(x)$, is not integrable and that this problem goes undetected because of the inherent complexity. In such cases, Markov chains can be simulated in conjunction with the target $\ell(x) \times f(x)$ but cannot converge. In the best cases, the resulting Markov chains will quickly exhibit divergent behavior, which signals there is a problem. Unfortunately, in the worst cases, these Markov chains present all the outer signs of stability and thus fail to indicate the difficulty. More details about this issue are discussed in Section 7.6.4 of the next chapter.

Exercise 6.2 Show that the random walk has no stationary distribution. Give the distribution of $X^{(t)}$ for $t = 10^4$ and $t = 10^6$ when $X^{(0)} = 0$, and deduce that $X^{(t)}$ has no limiting distribution.

6.3 Basic Metropolis–Hastings algorithms

The working principle of Markov chain Monte Carlo methods is quite straightforward to describe. Given a target density f, we build a Markov kernel K with stationary distribution f and then generate a Markov chain $(X^{(t)})$ using this kernel so that the limiting distribution of $(X^{(t)})$ is f and integrals can be approximated according to the Ergodic Theorem (6.2). The difficulty should thus be in constructing a kernel K that is associated with an arbitrary density f. But, quite miraculously, there exist methods for deriving such kernels that are universal in that they are theoretically valid for any density f!

The Metropolis–Hastings algorithm is an example of those methods. (Gibbs sampling, described in Chapter 7, is another example with equally universal potential.) Given the target density f, it is associated with a working conditional density $q(y|x)$ that, in practice, is easy to simulate. In addition, q can be almost arbitrary in that the only theoretical requirements are that the ratio $f(y)/q(y|x)$ is known up to a constant *independent* of x and that $q(\cdot|x)$ has enough dispersion to lead to an exploration of the entire support of f. Once

again, we stress the incredible feature of the Metropolis–Hastings algorithm that, for *every* given q, we can then construct a Metropolis–Hastings kernel such that f is its stationary distribution.

6.3.1 A generic Markov chain Monte Carlo algorithm

The Metropolis–Hastings algorithm associated with the objective (target) density f and the conditional density q produces a Markov chain $(X^{(t)})$ through the following transition kernel:

Algorithm 4 Metropolis–Hastings

Given $x^{(t)}$,

1. Generate $Y_t \sim q(y|x^{(t)})$.
2. Take

$$X^{(t+1)} = \begin{cases} Y_t & \text{with probability } \rho(x^{(t)}, Y_t), \\ x^{(t)} & \text{with probability } 1 - \rho(x^{(t)}, Y_t), \end{cases}$$

where

$$\rho(x, y) = \min\left\{ \frac{f(y)}{f(x)} \, \frac{q(x|y)}{q(y|x)} \, , 1 \right\} .$$

A generic `R` implementation is straightforward, assuming a generator for $q(y|x)$ is available as `geneq(x)`. If `x[t]` denotes the value of $X^{(t)}$,

```
> y=geneq(x[t])
> if (runif(1)<f(y)*q(y,x[t])/(f(x[t])*q(x[t],y))){
+    x[t+1]=y
+    }else{
+      x[t+1]=x[t]
+      }
```

since the value `y` is always accepted when the ratio is larger than one.

The distribution q is called the *instrumental* (or *proposal* or *candidate*) *distribution* and the probability $\rho(x, y)$ the *Metropolis–Hastings acceptance probability*. It is to be distinguished from the *acceptance rate*, which is the average of the acceptance probability over iterations,

$$\overline{\rho} = \lim_{T \to \infty} \frac{1}{T} \sum_{t=0}^{T} \rho(X^{(t)}, Y_t) = \int \rho(x, y) f(x) q(y|x) \, \mathrm{d}y \mathrm{d}x.$$

This quantity allows an evaluation of the performance of the algorithm, as discussed in Section 6.5.

While, at first glance, Algorithm 4 does not seem to differ from Algorithm 2, except for the notation, there are two fundamental differences between the two algorithms. The first difference is in their use since Algorithm 2 aims at maximizing a function $h(x)$, while the goal of Algorithm 4 is to explore the support of the density f according to its probability. The second difference is in their convergence properties. With the proper choice of a temperature schedule T_t in Algorithm 2, the simulated annealing algorithm converges to the maxima of the function h, while the Metropolis–Hastings algorithm is converging to the distribution f itself. Finally, modifying the proposal q along iterations may have drastic consequences on the convergence pattern of this algorithm, as discussed in Section 8.5.

Algorithm 4 satisfies the so-called *detailed balance condition*,

$$f(x)K(y|x) = f(y)K(x|y)\,, \tag{6.3}$$

from which we can deduce that f is the stationary distribution of the chain $\{X^{(t)}\}$ by integrating each side of the equality in x (see Exercise 6.8).

That Algorithm 4 is naturally associated with f as its stationary distribution thus comes quite easily as a consequence of the detailed balance condition for an arbitrary choice of the pair (f, q). In practice, the performance of the algorithm will obviously strongly depend on this choice of q, but consider first a straightforward example where Algorithm 4 can be compared with iid sampling.

Example 6.1. Recall Example 2.7, where we used an Accept–Reject algorithm to simulate a beta distribution. We can just as well use a Metropolis–Hastings algorithm, where the target density f is the $\mathcal{B}e(2.7, 6.3)$ density and the candidate q is uniform over $[0, 1]$, which means that it does not depend on the previous value of the chain. A Metropolis–Hastings sample is then generated with the following `R` code:

```
> a=2.7; b=6.3; c=2.669 # initial values
> Nsim=5000
> X=rep(runif(1),Nsim)  # initialize the chain
> for (i in 2:Nsim){
+   Y=runif(1)
+   rho=dbeta(Y,a,b)/dbeta(X[i-1],a,b)
+   X[i]=X[i-1] + (Y-X[i-1])*(runif(1)<rho)
+   }
```

A representation of the sequence $(X^{(t)})$ by `plot` does not produce any pattern in the simulation since the chain explores the same range at different periods. If we zoom in on the final period, for $4500 \le t \le 4800$, Figure 6.1 exhibits some characteristic features of Metropolis–Hastings sequences, namely that, for some intervals of time, the sequence $(X^{(t)})$ does not change because all corresponding

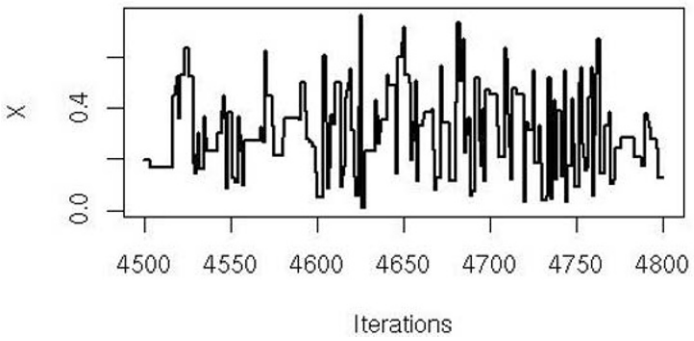

Fig. 6.1. Sequence $X^{(t)}$ for $t = 4500, \ldots, 4800$, when simulated from the Metropolis–Hastings algorithm with uniform proposal and $\mathcal{B}e(2.7, 6.3)$ target.

Y_t's are rejected. Note that those multiple occurrences of the same numerical value must be kept in the sample as such; otherwise, the validity of the approximation of f is lost! Indeed, when considering the entire chain as a sample, its histogram properly approximates the $\mathcal{B}e(2.7, 6.3)$ target. Figure 6.2 shows histograms and overlaid densities both for this Metropolis–Hastings sample and for an (exact) iid sample drawn using the `rbeta` command. The fits are quite similar, and this can be checked even further using a Kolmogorov–Smirnov test of equality between the two samples:

```
> ks.test(jitter(X),rbeta(5000,a,b))

        Two-sample Kolmogorov-Smirnov test

data:  jitter(X) and rbeta(5000,a,b)
D = 0.0202, p-value = 0.2594
alternative hypothesis: two-sided
```

which states that both samples are compatible with the same distribution. An additional (if mild) check of agreement is provided by the moments. For instance, since the mean and variance of a $\mathcal{B}e(a, b)$ distribution are $a/(a+b)$ and $ab/(a+b)^2(a+b+1)$, respectively, we can compare

$$\bar{X} = .301\,, \quad S^2 = .0205\,,$$

with the theoretical values of .3 for the mean and .021 for the variance. ◀

While the MCMC and exact sampling outcomes look identical in Figure 6.2, it is important to remember that the Markov chain Monte Carlo sample has correlation, while the iid sample does not. This means that the quality of the sample is necessarily degraded or, in other words, that we need more

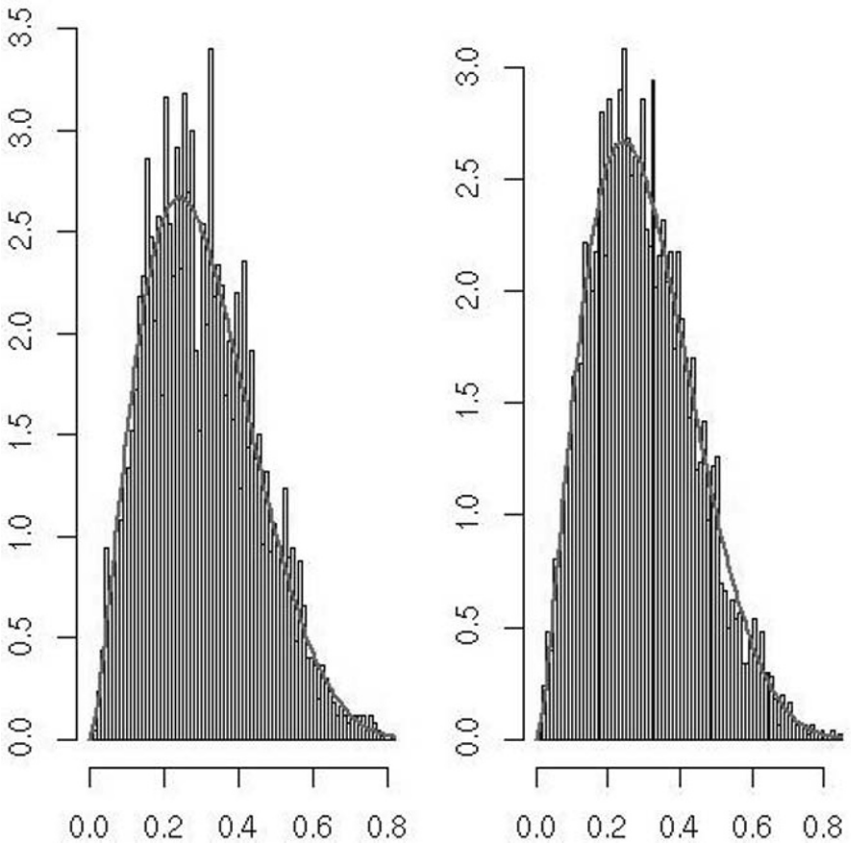

Fig. 6.2. Histograms of beta $\mathcal{B}e(2.7, 6.3)$ random variables with density function overlaid. In the left panel, the variables were generated from a Metropolis–Hastings algorithm with a uniform candidate, and in the right panel the random variables were directly generated using `rbeta(n,2.7,6.3)`.

simulations to achieve the same precision. This issue is formalized through the notion of *effective sample size* for Markov chains (Section 8.4.3).

In the symmetric case (that is, when $q(x|y) = q(y|x)$), the acceptance probability $\rho(x_t, y_t)$ is driven by the objective ratio $f(y_t)/f(x^{(t)})$ and thus even the acceptance probability is independent from q. (This special case is detailed in Section 6.4.1.) Again, Metropolis–Hastings algorithms share the same feature as the stochastic optimization Algorithm 2 (see Section 5.5), namely that they always accept values of y_t such that the ratio $f(y_t)/q(y_t|x^{(t)})$ is increased compared with the "previous" value $f(x^{(t)})/q(x^{(t)}|y_t)$. Some values y_t such that the ratio is decreased may also be accepted, depending on the ratio of the

ratios, but if the decrease is too sharp, the proposed value y_t will almost always be rejected. This property indicates how the choice of q can impact the performance of the Metropolis–Hastings algorithm. If the domain explored by q (its *support*) is too small, compared with the range of f, the Markov chain will have difficulties in exploring this range and thus will converge very slowly (if at all for practical purposes).

Another interesting property of the Metropolis–Hastings algorithm that adds to its appeal is that it only depends on the ratios

$$f(y_t)/f(x^{(t)}) \qquad \text{and} \qquad q(x^{(t)}|y_t)/q(y_t|x^{(t)})\,.$$

It is therefore independent of normalizing constants. Moreover, since all that matters is the ability to (a) simulate from q and (b) compute the ratio $f(y_t)/q(y_t|x^{(t)})$, q may be chosen in such a way that the intractable parts of f are eliminated in the ratio.

Since $q(y|x)$ is a conditional density, it integrates to one in y and, as such, involves a functional term that depends on both y and x as well as a normalizing term that depends on x, namely $q(y|x) = C(x)\tilde{q}(x, y)$. When noting above that the Metropolis–Hastings acceptance probability does not depend on normalizing constants, terms like $C(x)$ are obviously excluded from this remark since they must appear in the acceptance probability, lest it jeopardize the stationary distribution of the chain.

6.3.2 The independent Metropolis–Hastings algorithm

The Metropolis–Hastings algorithm of Section 6.3.1 allows a candidate distribution q that only depends on the present state of the chain. If we now require the candidate q to be *independent* of this present state of the chain (that is, $q(y|x) = g(y)$), we do get a special case of the original algorithm:

Algorithm 5 Independent Metropolis–Hastings

Given $x^{(t)}$

1. Generate $Y_t \sim g(y)$.
2. Take

$$X^{(t+1)} = \begin{cases} Y_t & \text{with probability } \min\left\{\dfrac{f(Y_t)\,g(x^{(t)})}{f(x^{(t)})\,g(Y_t)}\,, 1\right\} \\ x^{(t)} & \text{otherwise.} \end{cases}$$

This method then appears as a straightforward generalization of the Accept–Reject method in the sense that the instrumental distribution is the same density g as in the Accept–Reject method. Thus, the proposed values Y_t are the same, if not the accepted ones.

Metropolis–Hastings algorithms and Accept–Reject methods (Section 2.3), both being generic simulation methods, have similarities between them that allow comparison, even though it is rather rare to consider using a Metropolis–Hastings solution when an Accept–Reject algorithm is available. In particular, consider that

a. The Accept–Reject sample is iid, while the Metropolis–Hastings sample is not. Although the Y_t's are generated independently, the resulting sample is not iid, if only because the probability of acceptance of Y_t depends on $X^{(t)}$ (except in the trivial case when $f = g$).
b. The Metropolis–Hastings sample will involve repeated occurrences of the same value since rejection of Y_t leads to repetition of $X^{(t)}$ at time $t + 1$. This will have an impact on tests like `ks.test` that do not accept ties.
c. The Accept–Reject acceptance step requires the calculation of the upper bound $M \geq \sup_x f(x)/g(x)$, which is not required by the Metropolis–Hastings algorithm. This is an appealing feature of Metropolis–Hastings if computing M is time-consuming or if the existing M is inaccurate and thus induces a waste of simulations.

Exercise 6.3 Compute the acceptance probability $\rho(x, y)$ in the case $q(y|x) = g(y)$. Deduce that, for a given value $x^{(t)}$, the Metropolis–Hastings algorithm associated with the same pair (f, g) as an Accept–Reject algorithm accepts the proposed value Y_t more often than the Accept–Reject algorithm.

The following exercise gives a first comparison of Metropolis–Hastings with an Accept–Reject algorithm already used in Exercise 2.20 when both algorithms are based on the same candidate.

Exercise 6.4 Consider the target as the $\mathcal{G}(\alpha, \beta)$ distribution and the candidate as the gamma $\mathcal{G}([\alpha], b)$ distribution (where $[a]$ denotes the integer part of a).

a. Derive the corresponding Accept–Reject method and show that, when $\beta = 1$, the optimal choice of b is $b = [\alpha]/\alpha$.
b. Generate 5000 $\mathcal{G}(4, 4/4.85)$ random variables to derive a $\mathcal{G}(4.85, 1)$ sample (note that you will get less than 5000 random variables).
c. Use the same sample in the corresponding Metropolis–Hastings algorithm to generate 5000 $\mathcal{G}(4.85, 1)$ random variables.
d. Compare the algorithms using (i) their acceptance rates and (ii) the estimates of the mean and variance of the $\mathcal{G}(4.85, 1)$ along with their errors. (*Hint:* Examine the correlation in both samples.)

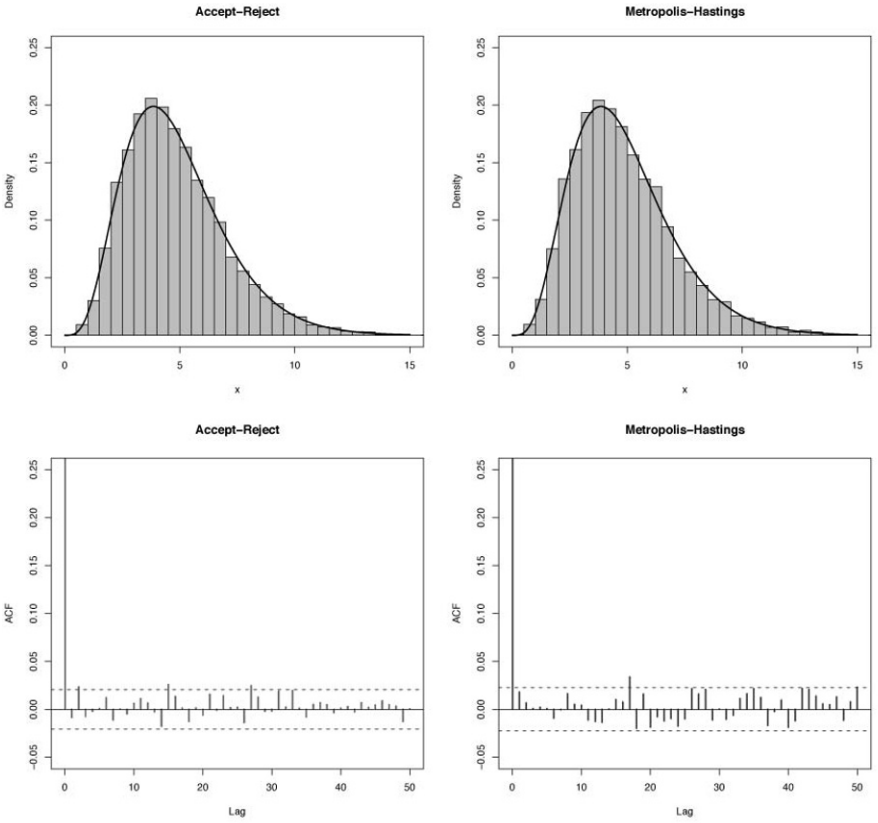

Fig. 6.3. Histograms and autocovariance functions from a gamma Accept–Reject algorithm (left panels) and a gamma Metropolis–Hastings algorithm (right panels). The target is a $\mathcal{G}(4.85, 1)$ distribution and the candidate is a $\mathcal{G}(4, 4/4.85)$ distribution. The autocovariance function is calculated with the `R` function `acf`.

Figure 6.3 illustrates Exercise 6.4 by comparing both Accept–Reject and Metropolis–Hastings samples. In this setting, operationally, the independent Metropolis–Hastings algorithm performs very similarly to the Accept–Reject algorithm, which in fact generates perfect and independent random variables.

Theoretically, it is also feasible to use a pair (f, g) such that a bound M on f/g does not exist and thus to use Metropolis–Hastings when Accept–Reject is not possible. However, as detailed in Robert and Casella (2004) and illustrated in the following formal example, the performance of the Metropolis–Hastings algorithm is then very poor, while it is very strong as long as $\sup f/g = M < \infty$.

Example 6.2. To generate a Cauchy random variable (that is, when f corresponds to a $\mathcal{C}(0,1)$ density), formally it is possible to use a $\mathcal{N}(0,1)$ candidate within a Metropolis–Hastings algorithm. The following R code will do it:

```
> Nsim=10^4
> X=c(rt(1,1))    # initialize the chain from the stationary
> for (t in 2:Nsim){
+    Y=rnorm(1)   # candidate normal
+    rho=dt(Y,1)*dnorm(X[t-1])/(dt(X[t-1],1)*dnorm(Y))
+    X[t]=X[t-1] + (Y-X[t-1])*(runif(1)<rho)
+    }
```

When executing this code, you may sometimes start with a large value for $X^{(0)}$, 12.788 say. In this case, `dnorm(X[t-1])` is equal to 0 because, while 12.788 can formally be a realization from a normal $\mathcal{N}(0,1)$, it induces computational underflow problems

```
> pnorm(12.78,log=T,low=F)/log(10)
[1] -36.97455
```

(meaning the probability of exceeding 12.78 is 10^{-37}) and the Markov chain remains constant for the 10^4 iterations! If the chain starts from a more central value, the outcome will resemble a normal sample much more than a Cauchy sample, as shown by Figure 6.4 *(center right)*. In addition, very large values of the sequence will be heavily weighted, resulting in long strings where the chain remains constant, as shown by Figure 6.4, the isolated peak in the histogram being representative of such an occurrence. If instead we use for the independent proposal g a Student's t distribution with $.5$ degrees of freedom (that is, if we replace `Y=rnorm(1)` with `Y=rt(1,.5)` in the code above), the behavior of the chain is quite different. Very large values of Y_t may occur from time to time (as shown in Figure 6.4 *(upper left)*), the histogram fit is quite good *(center left)*, and the sequence exhibits no visible correlation *(lower left)*. If we consider the approximation of a quantity like $\Pr(X < 3)$, for which the exact value is `pt(3,1)` (that is, 0.896), the difference between the two choices of g is crystal clear in Figure 6.5, obtained by

```
> plot(cumsum(X<3)/(1:Nsim),lwd=2,ty="l",ylim=c(.85,1)).
```

The chain based on the normal proposal is consistently off the true value, while the chain based on the t distribution with $.5$ degrees of freedom converges quite quickly to this value. Note that, from a theoretical point of view, the Metropolis–Hastings algorithm associated with the normal proposal still converges, but the convergence is so slow as to be useless. ◀

We now look at a somewhat more realistic statistical example that corresponds to the general setting when an independent proposal is derived from a preliminary estimation of the parameters of the model. For instance, when simulating from a posterior distribution $\pi(\theta|x) \propto \pi(\theta)f(x|\theta)$, this independent

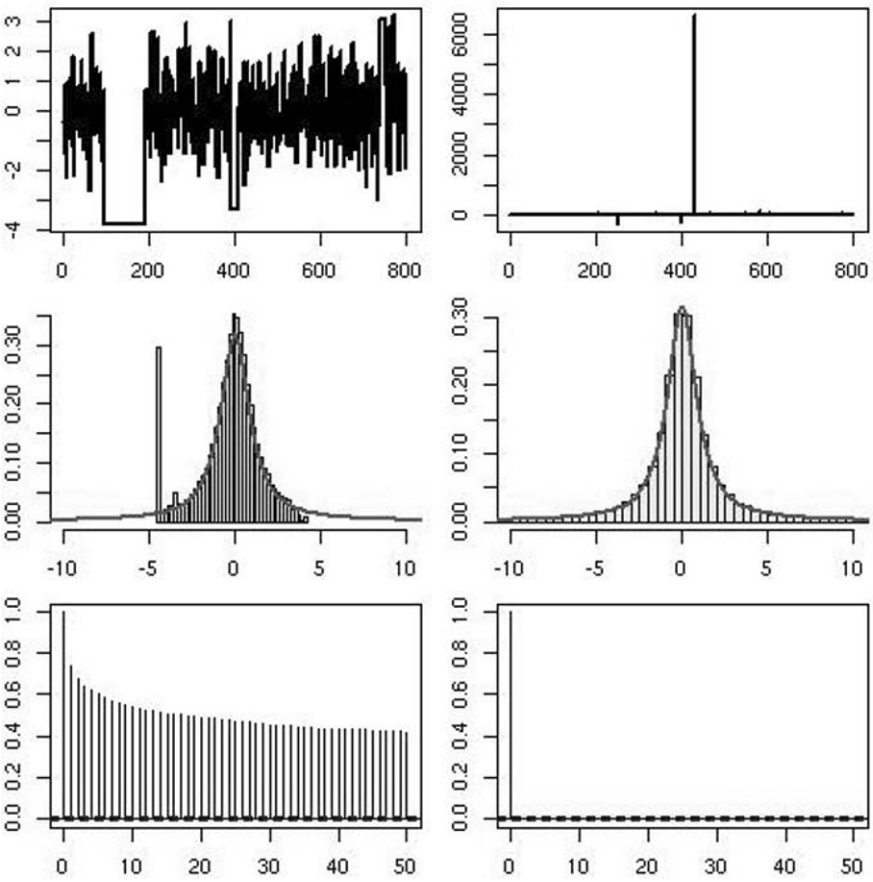

Fig. 6.4. Comparison of two Metropolis–Hastings schemes for a Cauchy target when generating *(left)* from a $\mathcal{N}(0,1)$ proposal and *(right)* from a $\mathcal{T}_{1/2}$ proposal based on 10^5 simulations. *(top)* Excerpt from the chains $(X^{(t)})$; *(center)* histograms of the samples; *(bottom)* autocorrelation graphs obtained by acf.

proposal could be a normal or a t distribution centered at the MLE $\hat{\theta}$ and with variance-covariance matrix equal to the inverse of Fisher's information matrix.

Example 6.3. The `cars` dataset relates braking distance (y) to speed (x) in a sample of cars. Figure 6.6 shows the data along with a fitted quadratic curve that is given by the `R` function `lm`. The model posited for this dataset is a quadratic model

$$y_{ij} = a + bx_i + cx_i^2 + \varepsilon_{ij}, \quad i = 1, \ldots, k, \quad j = 1, \ldots n_i,$$

where we assume that $\varepsilon_{ij} \sim N(0, \sigma^2)$ and independent. The likelihood function is then proportional to

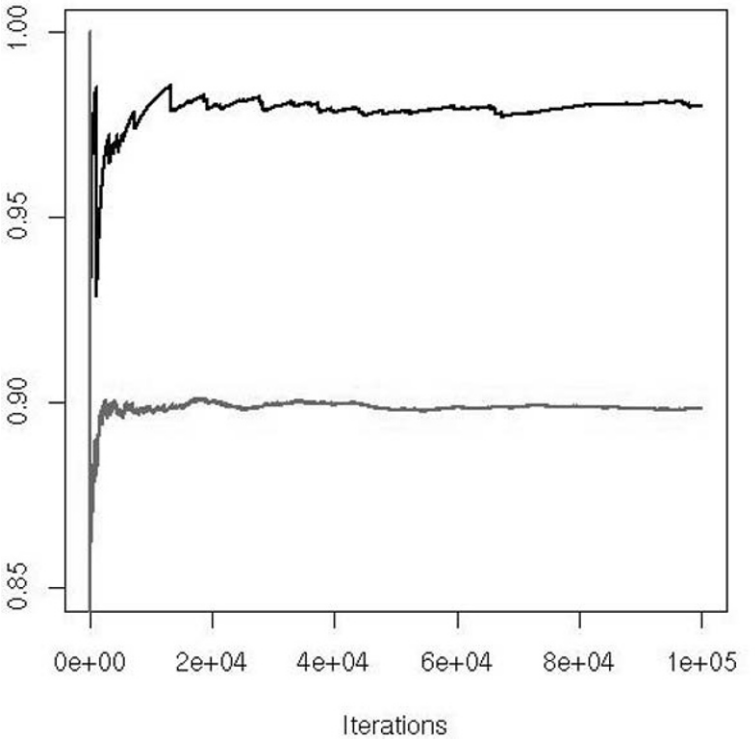

Fig. 6.5. Example 6.2: cumulative coverage plot of a Cauchy sequence generated by a Metropolis–Hastings algorithm based on a $\mathcal{N}(0,1)$ proposal (upper lines) and one generated by a Metropolis–Hastings algorithm based on a $\mathcal{T}_{1/2}$ proposal (lower lines). After 10^5 iterations, the Metropolis–Hastings algorithm based on the normal proposal has not yet converged.

$$\left(\frac{1}{\sigma^2}\right)^{N/2} \exp\left\{\frac{-1}{2\sigma^2}\sum_{ij}(y_{ij}-a-bx_i-cx_i^2)^2\right\},$$

where $N=\sum_i n_i$ is the total number of observations. We can view this likelihood function as a posterior distribution on a, b, c, and σ^2 (for instance based on a flat prior), and, as a toy problem, we can try to sample from this distribution with a Metropolis–Hastings algorithm (since this standard distribution can be simulated directly; see Exercise 6.12). To start with, we can get a candidate by generating coefficients according to their fitted sampling distribution. That is, we can use the `R` command

```
> x2=x^2
> summary(lm(y~x+x2))
```

to get the output

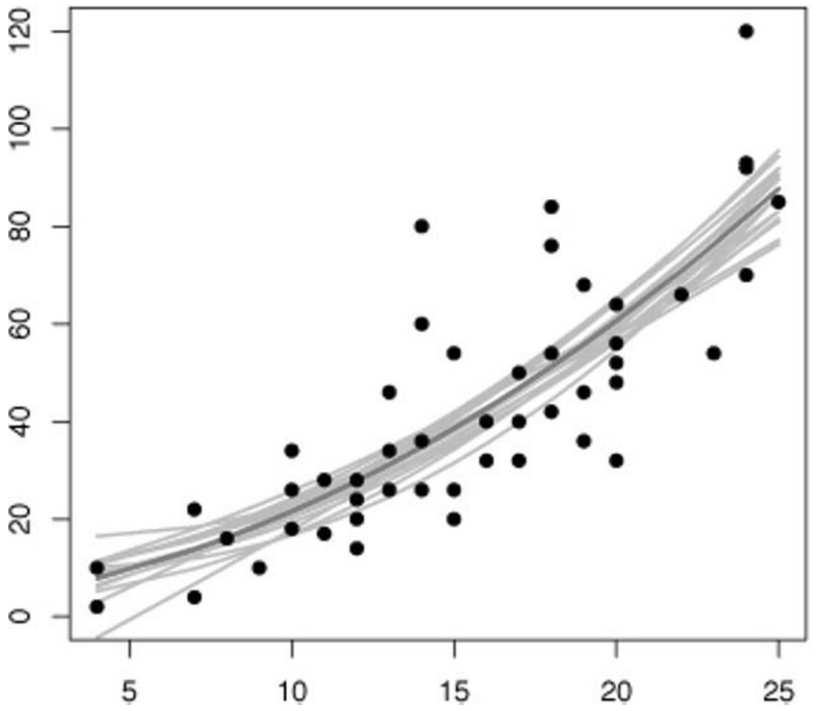

Fig. 6.6. Braking data with quadratic curve *(dark)* fitted with the least squares function lm. The grey curves represent the Monte Carlo sample $(a^{(i)}, b^{(i)}, c^{(i)})$ and show the variability in the fitted lines based on the last 500 iterations of 4000 simulations.

Coefficients:

	Estimate	Std. Error	t value	Pr($>\|t\|$)
(Intercept)	2.63328	14.80693	0.178	0.860
x	0.88770	2.03282	0.437	0.664
x2	0.10068	0.06592	1.527	0.133

Residual standard error: 15.17 on 47 degrees of freedom

As suggested above, we can use the candidate normal distribution centered at the MLEs,

$$a \sim \mathcal{N}(2.63, (14.8)^2), \quad b \sim \mathcal{N}(.887, (2.03)^2), \quad c \sim \mathcal{N}(.100, (0.065)^2),$$

$$\sigma^{-2} \sim \mathcal{G}(n/2, (n-3)(15.17)^2),$$

in a Metropolis–Hastings algorithm to generate samples $(a^{(i)}, b^{(i)}, c^{(i)})$. Figure 6.6 illustrates the variability of the curves associated with the outcome of this simulation. ◀

6.4 A selection of candidates

The study of independent Metropolis–Hastings algorithms is certainly interesting, but their practical implementation is more problematic in that they are delicate to use in complex settings because the construction of the proposal is complicated—if we are using simulation, it is often because deriving estimates like MLEs is difficult—and because the choice of the proposal is highly influential on the performance of the algorithm. Rather than building a proposal from scratch or suggesting a non-parametric approximation based on a preliminary run—because it is unlikely to work for moderate to high dimensions—it is therefore more realistic to gather information about the target stepwise, that is, by exploring the neighborhood of the current value of the chain. If the exploration mechanism has enough energy to reach as far as the boundaries of the support of the target f, the method will eventually uncover the complexity of the target. (This is fundamentally the same intuition at work in the simulated annealing algorithm of Section 5.3.3 and the stochastic gradient method of Section 5.3.2.)

6.4.1 Random walks

A more natural approach for the practical construction of a Metropolis–Hastings proposal is thus to take into account the value previously simulated to generate the following value; that is, to consider a *local* exploration of the neighborhood of the current value of the Markov chain.

The implementation of this idea is to simulate Y_t according to

$$Y_t = X^{(t)} + \varepsilon_t,$$

where ε_t is a random *perturbation* with distribution g independent of $X^{(t)}$, for instance a uniform distribution or a normal distribution, meaning that $Y_t \sim \mathcal{U}(X^{(t)} - \delta, X^{(t)} + \delta)$ or $Y_t \sim \mathcal{N}(X^{(t)}, \tau^2)$ in unidimensional settings. In terms of the general Metropolis–Hastings algorithm, the proposal density $q(y|x)$ is now of the form $g(y - x)$. The Markov chain associated with q is a *random walk* (as described in Section 6.2) when the density g is symmetric around zero; that is, satisfying $g(-t) = g(t)$. But, due to the additional Metropolis–Hastings acceptance step, the Metropolis–Hastings Markov chain $\{X^{(t)}\}$ is *not* a random walk. This approach leads to the following Metropolis–Hastings algorithm, which also happens to be the original one proposed by Metropolis et al. (1953).

Algorithm 6 Random walk Metropolis–Hastings
Given $x^{(t)}$,

1. Generate $Y_t \sim g(y - x^{(t)})$.

2. Take

$$X^{(t+1)} = \begin{cases} Y_t & \text{with probability } \min\left\{1, f(Y_t)/f(x^{(t)})\right\}, \\ x^{(t)} & \text{otherwise.} \end{cases}$$

As noted above, the acceptance probability does not depend on g. This means that, for a given pair $(x^{(t)}, y_t)$, the probability of acceptance is the same whether y_t is generated from a normal or from a Cauchy distribution. Obviously, changing g will result in different ranges of values for the Y_t's and a different acceptance rate, so this is not to say that the choice of g has no impact whatsoever on the behavior of the algorithm, but this invariance of the acceptance probability is worth noting. It is actually linked to the fact that, for *any* (symmetric) density g, the invariant measure associated with the random walk is the Lebesgue measure on the corresponding space (see Meyn and Tweedie, 1993).

Example 6.4. The historical example of Hastings (1970) considers the formal problem of generating the normal distribution $\mathcal{N}(0,1)$ based on a random walk proposal equal to the uniform distribution on $[-\delta, \delta]$. The probability of acceptance is then

$$\rho(x^{(t)}, y_t) = \exp\{(x^{(t)2} - y_t^2)/2\} \wedge 1.$$

Figure 6.7 describes three samples of 5000 points produced by this method for $\delta = 0.1, 1$, and 10 and clearly shows the difference in the produced chains: Too narrow or too wide a candidate (that is, a smaller or a larger value of δ) results in higher autocovariance and slower convergence. Note the distinct patterns for $\delta = 0.1$ and $\delta = 10$ in the upper graphs: In the former case, the Markov chain moves at each iteration but very slowly, while in the latter it remains constant over long periods of time. ◀

As noted in this formal example, calibrating the scale δ of the random walk is crucial to achieving a good approximation to the target distribution in a reasonable number of iterations. In more realistic situations, this calibration becomes a challenging issue, partly tackled in Section 6.5 and reconsidered in further detail in Chapter 8.

Example 6.5. The mixture example detailed in Example 5.2 from the perspective of a maximum likelihood estimation can also be considered from a Bayesian point of view using for instance a uniform prior $\mathcal{U}(-2, 5)$ on both μ_1 and μ_2. The posterior distribution we are interested in is then proportional to the likelihood. Implementing Algorithm 6 in this example is surprisingly easy in that we can recycle most of the implementation of the simulated annealing Algorithm 2, already

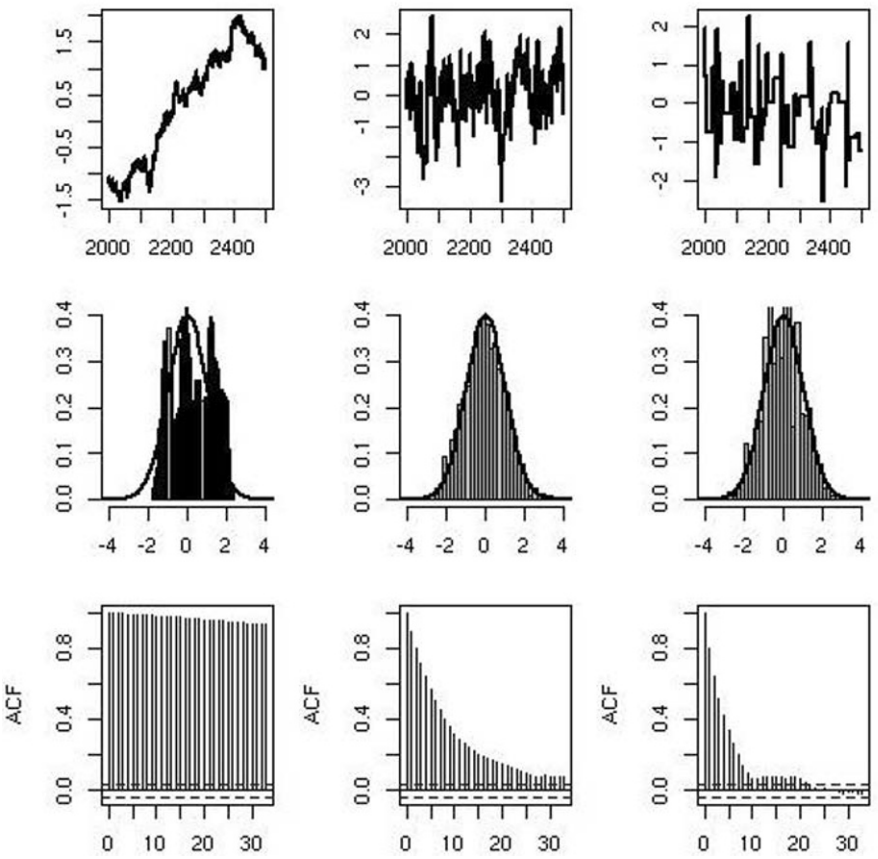

Fig. 6.7. Outcomes of random walk Metropolis–Hastings algorithms for Example 6.4. The left panel has a $\mathcal{U}(-.1, .1)$ candidate, the middle panel has $\mathcal{U}(-1, 1)$, and the right panel has $\mathcal{U}(-10, 10)$. The upper graphs represent the last 500 iterations of the chains, the middle graphs indicate how the histograms fit the target, and the lower graphs give the respective autocovariance functions.

programmed in Example 5.2. Indeed, the core of the R code is very similar except for the increase in temperature, which obviously is not necessary here:

```
> scale=1
> the=matrix(runif(2,-2,5),ncol=2)
> curlike=hval=like(x)
> Niter=10^4
> for (iter in (1:Niter)){
+   prop=the[iter,]+rnorm(2)*scale
+   if ((max(-prop)>2)||(max(prop)>5)||
+       (log(runif(1))>like(prop)-curlike)) prop=the[iter,]
```

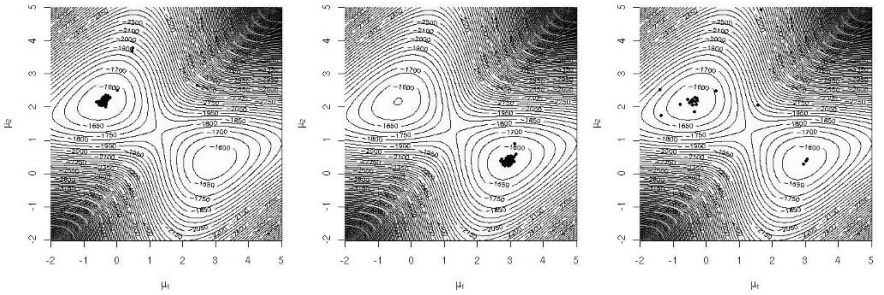

Fig. 6.8. Impact of the scale of the random walk on the exploration of the modes in the mixture model: representation of the Markov chain $(\mu_1^{(t)}, \mu_2^{(t)})$ on top of the log-posterior surface with *(left and center)* scale equal to 1 and *(right)* scale equal to 2 based on 10^4 simulations and 500 simulated observations.

```
+    curlike=like(prop)
+    hval=c(hval,curlike)
+    the=rbind(the,prop)}
```

Since the main problem of this target is the existence of two modes, one of which is smaller than the other, we can compare the impact of different choices of scale on the behavior of the chain in terms of exploration of both modes and the attraction therein. When the scale is 1, the modes are highly attractive and, out of 10^4 iterations, it is not uncommon to explore only one mode neighborhood, as shown in Figure 6.8 (left and center) for both modes. If the scale increases to 2, the proposal is diverse enough to reach both modes but at a cost. Out of 10^4 iterations, the chain only changes values 23 times! For the smaller scale 1, the number of changes is closer to 100, still a very low acceptance rate. ◀

An issue that often arises when using random walks on constrained domains is whether or not the random walk should be constrained as well. The answer to this question is no in that using constraints in the proposal modifies the function g and thus jeopardizes the validity of the ratio of the targets found in Algorithm 6. When values y_t outside the range of f are proposed (that is, when $f(y_t) = 0$), the proposed value is rejected and the current value $X^{(t)}$ is duplicated. Obviously, picking a random walk density that often ends up outside the domain of f is a poor idea in that the chain will be stuck most of the time! But it is formally correct.

6.4.2 Alternative candidates

While the independent Metropolis–Hastings algorithm only applies in specific situations, the random walk Metropolis–Hastings algorithm often appears as a generic Metropolis–Hastings algorithm that caters to most cases. Nonetheless,

the random walk solution is not necessarily the most efficient choice in that (a) it requires many iterations to overcome difficulties such as low-probability regions between modal regions of f and (b) because of its symmetric features, it spends roughly half the simulation time revisiting regions it has already explored. There exist alternatives that bypass the perfect symmetry in the random walk proposal to gain in efficiency, although they are not always easy to implement (see, for example, Robert and Casella, 2004).

One of those alternatives is the Langevin algorithm of Roberts and Rosenthal (1998) that tries to favor moves toward higher values of the target f by including a gradient in the proposal,

$$Y_t = X^{(t)} + \frac{\sigma^2}{2} \nabla \log f(X^{(t)}) + \sigma \epsilon_t \,, \qquad \epsilon_t \sim g(\epsilon) \,,$$

the parameter σ being the scale factor of the proposal. When Y_t is constructed this way, the Metropolis–Hastings acceptance probability is equal to

$$\rho(x,y) = \min \left\{ \frac{f(y)}{f(x)} \, \frac{g\left[(x-y)/\sigma - \sigma \nabla \log f(y)/2\right]}{g\left[(y-x)/\sigma - \sigma \nabla \log f(x)/2\right]} \,, 1 \right\} \,.$$

While this scheme may remind you of the stochastic gradient techniques of Section 5.3.2, it differs from those for two reasons. One is that the scale σ is fixed in the Langevin algorithm, as opposed to decreasing in the stochastic gradient method. Another is that the proposed move to Y_t is not necessarily accepted for the Langevin algorithm, ensuring the stationarity of f for the resulting chain.

Example 6.6. Based on the same probit model of the now well-known `Pima.tr` dataset as in Example 3.10, we can use the likelihood function `like` already defined on page 85 and compute the gradient in closed form as

```
grad=function(a,b){
  don=pnorm(q=a+outer(X=b,Y=da[,2],FUN="*"))
  x1=sum((dnorm(x=a+outer(X=b,Y=da[,2],FUN="*"))/don)*da[,1]-
          (dnorm(x=-a-outer(X=b,Y=da[,2],FUN="*"))/
                (1-don))*(1-da[,1]))
  x2=sum(da[,2]*(
    (dnorm(x=a+outer(X=b,Y=da[,2],FUN="*"))/don)*da[,1]-
    (dnorm(x=-a-outer(X=b,Y=da[,2],FUN="*"))/
          (1-don))*(1-da[,1])))
  return(c(x1,x2))
  }
```

When implementing the basic iteration of the Langevin algorithm

```
>  prop=curmean+scale*rnorm(2)
>  propmean=prop+0.5*scale^2*grad(prop[1],prop[2])
```

```
> if (log(runif(1))>like(prop[1],prop[2])-likecur-
+    sum(dnorm(prop,mean=curmean,sd=scale,lo=T))+
+    sum(dnorm(the[t-1,],mean=propmean,sd=scale,lo=T))){
+       prop=the[t-1,];propmean=curmean}
```

we need to select `scale` small enough because otherwise `grad(prop)` returns `NaN` given that `pnorm(q=a+outer(X=b,Y=da[,2],FUN="*"))` is then either 1 or 0. With a scale equal to 0.01, the chain correctly explores the posterior distribution, as shown in Figure 6.9, even though it moves very slowly. ◀

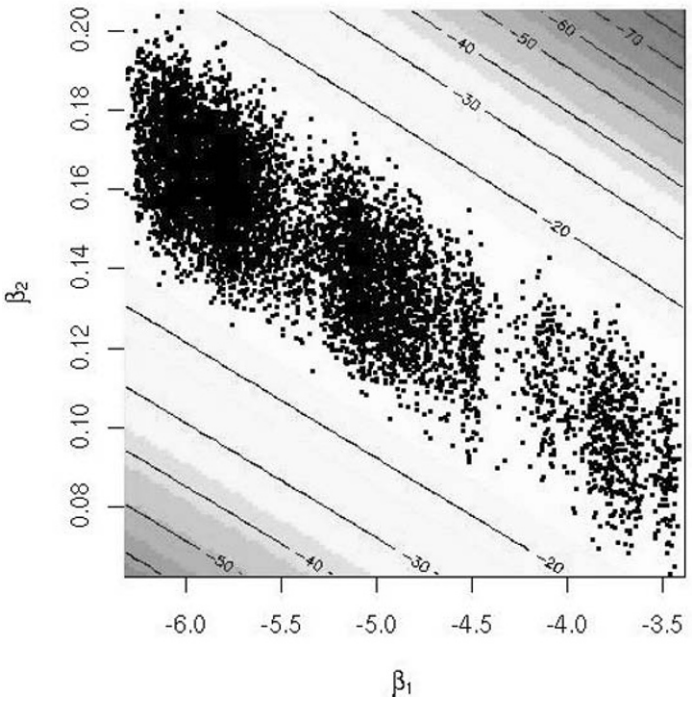

Fig. 6.9. Repartition of the Langevin sample corresponding to the probit posterior defined in Example 3.10 based on 20 observations from Pima.tr and 5×10^4 iterations.

The modification of the random walk proposal may, however, further hinder the mobility of the Markov chain by reinforcing the polarization around local modes. For instance, when the target is the posterior distribution of the mixture model studied in Example 6.5, the bimodal structure of the target is a hindrance for the implementation of the Langevin algorithm in that the local mode becomes even more attractive.

Example 6.7. (Continuation of Example 6.5) The modification of the random walk Metropolis–Hastings algorithm is straightforward in that we simply have to add the gradient drift in the R code. Defining the gradient function

```
gradlike=function(mu){
  deno=.2*dnorm(da-mu[1])+.8*dnorm(da-mu[2])
  gra=sum(.2*(da-mu[1])*dnorm(da-mu[1])/deno)
  grb=sum(.8*(da-mu[2])*dnorm(da-mu[2])/deno)
  return(c(gra,grb))
  }
```

the simulation of the Markov chain involves

```
> prop=curmean+rnorm(2)*scale
> meanprop=prop+.5*scale^2*gradlike(prop)
> if ((max(-prop)>2)||(max(prop)>5)||(log(runif(1))>like(prop)
+  -curlike-sum(dnorm(prop,curmean,lo=T))+
+  sum(dnorm(the[iter,],meanprop,lo=T)))){
+     prop=the[iter,]
+     meanprop=curmean
+     }
> curlike=like(prop)
> curmean=meanprop
```

When running this Langevin alternative on the same dataset as in Example 6.5, the scale needs to be reduced quite a lot for the chain to move. For instance, using `scale=.2` was not small enough for this purpose and we had to lower it to `scale=.1` to start seeing high enough acceptance rates. Figure 6.10 is representative of the impact of the starting point on the convergence of the chain since starting near the wrong mode leads to a sample concentrated on this very mode. The reason for this difficulty is that, with 500 observations, the likelihood is very peaked and so is the gradient. ◀

Both examples above show how delicate the tuning of the Langevin algorithm can be. This may explain why it is not widely implemented, even though it is an easy enough modification of the basic random walk code.

Random walk Metropolis–Hastings algorithms also apply to discrete support targets. While this sounds more like a combinatoric or an image-processing setting, since most statistical problems involve continuous parameter spaces, an exception is the case of model choice (see, for example, Robert, 2001, Chapter 7), where the index of the model to be selected is the "parameter" of interest.

Example 6.8. Given an ordinary linear regression with n observations,

$$\mathbf{y}|\beta, \sigma^2, X \sim \mathcal{N}_n(X\beta, \sigma^2 I_n)\,,$$

where X is a (n, p) matrix, the likelihood is

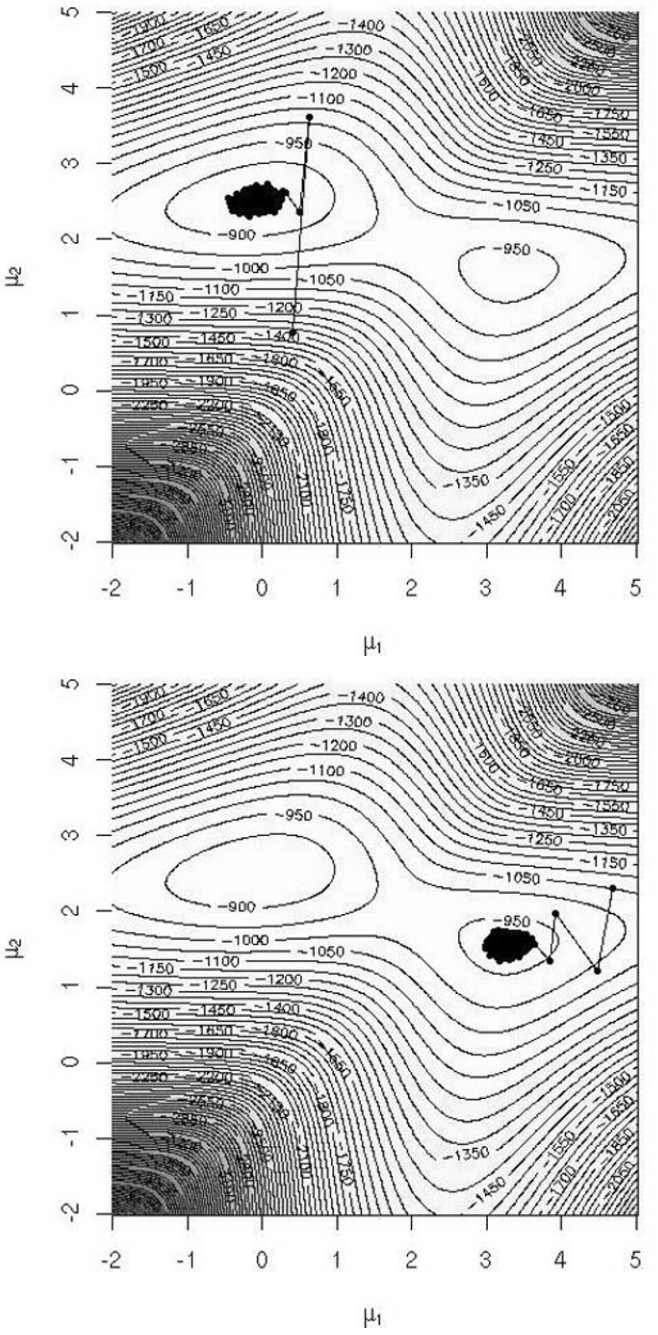

Fig. 6.10. Exploration of the modes in the mixture model by a Langevin algorithm: representation of two Markov chains $(\mu_1^{(t)}, \mu_2^{(t)})$ on top of the log-posterior surface with a scale equal to .1 based on 10^4 simulations and a simulated dataset of 500 observations.

$$\ell\left(\beta,\sigma^2|\mathbf{y},X\right)=\left(2\pi\sigma^2\right)^{-n/2}\exp\left[-\frac{1}{2\sigma^2}(\mathbf{y}-X\beta)^{\mathsf{T}}(\mathbf{y}-X\beta)\right]$$

and, under the so-called g-prior of Zellner (1986),

$$\beta|\sigma^2,X\sim\mathcal{N}_{k+1}(\tilde{\beta},n\sigma^2(X^{\mathsf{T}}X)^{-1})\quad\text{and}\quad\pi(\sigma^2|X)\propto\sigma^{-2}$$

(where the constant g is chosen equal to n), the marginal distribution of $\mathbf{y}$ is a multivariate t distribution,

$$m(\mathbf{y}|X)=(n+1)^{-(k+1)/2}\pi^{-n/2}\Gamma(n/2)\left[\mathbf{y}^{\mathsf{T}}\mathbf{y}-\frac{n}{n+1}\mathbf{y}^{\mathsf{T}}X(X^{\mathsf{T}}X)^{-1}X^{\mathsf{T}}\mathbf{y}\right.$$
$$\left.-\frac{1}{n+1}\tilde{\beta}^{\mathsf{T}}X^{\mathsf{T}}X\tilde{\beta}\right]^{-n/2}.$$

As an illustration, we consider the `swiss` dataset, where the logarithm of the fertility in 47 districts of Switzerland around 1888 is the variable y to be explained by some socioeconomic indicators,

```
> y=log(as.vector(swiss[,1]))
> X=as.matrix(swiss[,2:6])
```

The covariate matrix X involves five explanatory variables

```
> names(swiss)
[1] "Fertility"  "Agriculture"  "Examination"  "Education"
[5] "Catholic"   "Infant.Mortality"
```

(that are explained by `?swiss`) and we want to compare the 2^5 models corresponding to all possible subsets of covariates. (In this toy example, the number of models is small enough to allow for the computation of all marginals and therefore the true probabilities of all models under comparison.) Following Marin and Robert (2007), we index all models by vectors γ of binary indicators where $\gamma_i=0$ indicates that the corresponding column of X is used in the regression. (Note that, adopting Marin and Robert's, 2007, convention, we always include the intercept in a model.) Using the fast inverse matrix function

```
inv=function(X){
  EV=eigen(X)
  EV$vector%*%diag(1/EV$values)%*%t(EV$vector)
  }
```

we then compute the log marginal density corresponding to the model γ, now denoted as $m(\mathbf{y}|X,\gamma)$, as

```
lpostw=function(gam,y,X,beta){
  n=length(y)
  qgam=sum(gam)
  Xt1=cbind(rep(1,n),X[,which(gam==1)])
```

```
if (qgam!=0) P1=Xt1%*%inv(t(Xt1)%*%Xt1)%*%t(Xt1) else{
   P1=matrix(0,n,n)}
  -(qgam+1)/2*log(n+1)-n/2*log(t(y)%*%y-n/(n+1)*
  t(y)%*%P1%*%y-1/(n+1)*t(beta)%*%t(cbind(rep(1,n),
  X))%*%P1%*%cbind(rep(1,n),X)%*%beta)
}
```

The exploration of the space of models can result from a Metropolis–Hastings algorithm that moves around models by changing one model indicator at a time; that is, given the current indicator vector $\gamma^{(t)}$, the Metropolis–Hastings proposal picks one of the p coordinates, say i, and chooses between keeping $\gamma_i^{(t)}$ and switching to $1-\gamma_i^{(t)}$ with probabilities proportional to the associated marginals. The Metropolis–Hastings acceptance probability of the proposed model $\gamma^\star$ is then equal to

$$\min\left\{\frac{m(\mathbf{y}|X,\gamma^\star)}{m(\mathbf{y}|X,\gamma^{(t)})}\,\frac{m(\mathbf{y}|X,\gamma^{(t)})}{m(\mathbf{y}|X,\gamma^\star)},1\right\}=1$$

since the normalising constants cancel. This means that we do not have to consider rejecting the proposed model $\gamma^\star$ because it is always accepted at the Metropolis–Hastings step! Running the `R` function

```
gocho=function(niter,y,X){
  lga=dim(X)[2]
  beta=lm(y~X)$coeff
  gamma=matrix(0,nrow=niter,ncol=lga)
  gamma[1,]=sample(c(0,1),lga,rep=T)
  for (t in 1:(niter-1)){
    j=sample(1:lga,1)
    gam0=gam1=gamma[t,];gam1[j]=1-gam0[j]
    pr=lpostw(gam0,y,X,beta)
    pr=c(pr,lpostw(gam1,y,X,beta))
    pr=exp(pr-max(pr))
    gamma[t+1,]=gam0
    if (sample(c(0,1),1,prob=pr))
      gamma[t+1,]=gam1}
  gamma
  }
```

then produces a sample (approximately) distributed from the posterior distribution on the set of indicators; that is, on the collection of possible submodels. Based on the outcome

```
> out=gocho(10^5,y,X)
```

the most likely model corresponds to the exclusion of the `Agriculture` variable (that is, $\gamma=(1,0,1,1,1)$), with estimated probability 0.4995, while the true probability is 0.4997. (This model is also the one indicated by `lm(y~X)`.) Similarly,

the second most likely model is $\gamma = (0, 0, 1, 1, 1)$, with an estimated probability of 0.237 versus a true probability of 0.234. The probability that each variable is included within the model is also provided by

```
> apply(out,2,mean)
[1] 0.66592 0.17978 0.99993 0.91664 0.94499
```

which, again, indicates that the last three variables of `swiss` are the most significant in this analysis. ◀

The fact that the acceptance probability is always equal to 1 in Example 6.8 is due to the use of the true target probability on a subset of the possible values of the model indicator.

Exercise 6.5 Starting from the prior distribution

$$\beta|\sigma^2, X \sim \mathcal{N}_{k+1}(\tilde{\beta}, n\sigma^2(X^{\mathsf{T}}X)^{-1}) \ :$$

a. Show that
$$X\beta|\sigma^2, X \sim \mathcal{N}_n(X\tilde{\beta}, n\sigma^2 X(X^{\mathsf{T}}X)^{-1}X^{\mathsf{T}})$$
and that
$$\mathbf{y}|\sigma^2, X \sim \mathcal{N}_n(X\tilde{\beta}, \sigma^2(I_n + nX(X^{\mathsf{T}}X)^{-1}X^{\mathsf{T}}))\,.$$
b. Show that integrating in σ^2 with $\pi(\sigma^2) = 1/\sigma^2$ yields the marginal distribution of $\mathbf{y}$ above.
c. Compute the value of the marginal density of y for the `swiss` dataset.

6.5 Acceptance rates

There are infinite choices for the candidate distribution q in a Metropolis–Hastings algorithm, and here we discuss the possibility of achieving an "optimal" choice. Most obviously, this is not a well-defined concept in that the "optimal" choice of q is to take $q = f$, the target distribution, when reasoning in terms of speed of convergence. This is obviously a formal result that has no relevance in practice! Instead, we need to adopt a practical criterion that allows the comparison of proposal kernels in situations where (almost) nothing is known about f. One such criterion is the acceptance rate of the corresponding Metropolis–Hastings algorithm since it can be easily computed when running this algorithm via the empirical frequency of acceptance. In contrast to Chapter 2, where the calibration of an Accept–Reject algorithm was based on the maximum acceptance rate, merely optimizing the acceptance rate will not necessarily result in the best algorithm in terms of mixing and convergence.

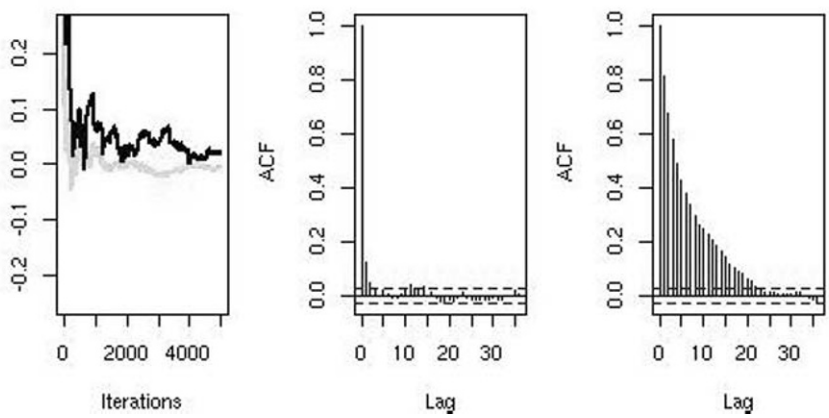

Fig. 6.11. Cumulative mean plot (left) from a Metropolis–Hastings algorithm used to generate a $\mathcal{N}(0,1)$ random variable from a double-exponential proposal distribution $\mathcal{L}(1)$ (lighter) and $\mathcal{L}(3)$ (black). The center and left panels show the autocovariance for the $\mathcal{L}(1)$ and $\mathcal{L}(3)$ proposals, respectively.

Example 6.9. In an Accept–Reject algorithm generating a $\mathcal{N}(0,1)$ sample from a double-exponential distribution $\mathcal{L}(\alpha)$ with density $g(x|\alpha) = (\alpha/2)\exp(-\alpha|x|)$, the choice $\alpha = 1$ optimizes the acceptance rate (Exercise 2.19). We can use this distribution as an independent candidate q in a Metropolis–Hastings algorithm. Figure 6.11 compares the behavior of this $\mathcal{L}(1)$ candidate along with an $\mathcal{L}(3)$ distribution, which, for this simulation, produces an inferior outcome in the sense that it has larger autocovariances and, as a result of this, slower convergence. Obviously, a deeper analysis would be necessary to validate this statement, but our point here is that the acceptance rate (estimated) for $\alpha = 1$ is twice as large, 0.83, as the acceptance rate (estimated) for $\alpha = 3$, 0.47. ◀

While independent Metropolis–Hastings algorithms can indeed be optimized or at least compared through their acceptance rate, because this reduces the number of replicas in the chain $\{X^{(t)}\}$ and thus the correlation level in the chain, this is not true for other types of Metropolis–Hastings algorithms, first and foremost the random walk version.

Exercise 6.6 The *inverse Gaussian distribution* has the density

$$f(z|\theta_1,\theta_2) \propto z^{-3/2} \exp\left\{-\theta_1 z - \frac{\theta_2}{z} + 2\sqrt{\theta_1\theta_2} + \log\sqrt{2\theta_2}\right\}$$

on $\mathbb{R}_+$ $(\theta_1 > 0, \theta_2 > 0)$.

a. A candidate for a Metropolis–Hastings algorithm targeting f is the $\mathcal{G}(\alpha,\beta)$ distribution. Show that

$$\frac{f(x)}{g(x)} \propto x^{-\alpha-1/2} \exp\left\{(\beta-\theta_1)x - \frac{\theta_2}{x}\right\}$$

is maximized in x at

$$x^*_\beta = \frac{(\alpha+1/2) - \sqrt{(\alpha+1/2)^2 + 4\theta_2(\theta_1-\beta)}}{2(\beta-\theta_1)}\,.$$

b. After maximizing in x, the goal would be to minimize the bound on f/g over (α,β) for fixed (θ_1,θ_2). This is impossible analytically, but for chosen values of (θ_1,θ_2) we can plot this function of (α,β). Do so using for instance `persp`. Do any patterns emerge?

c. The mean of the inverse Gaussian distribution is $\sqrt{\theta_2/\theta_1}$, so taking $\alpha = \beta\sqrt{\theta_2/\theta_1}$ will make the means of the candidate and target coincide. For $\theta_1 = \theta_2$, match means and find an "optimal" candidate in terms of the acceptance rate.

The *random walk* version of the Metropolis–Hastings algorithm, introduced in Section 6.4.1, does indeed require a different approach to acceptance rates, given the dependence of the candidate distribution on the current state of the chain. In fact, as already seen in Example 6.4, a high acceptance rate does not necessarily indicate that the algorithm is behaving satisfactorily since it may instead correspond to the fact that the chain is moving too slowly on the surface of f. When $x^{(t)}$ and y_t are close, in the sense that $f(x^{(t)})$ and $f(y_t)$ are approximately equal, the random walk Metropolis–Hastings algorithm leads to the acceptance of y_t with probability

$$\min\left(\frac{f(y_t)}{f(x^{(t)})}, 1\right) \simeq 1.$$

A high acceptance rate may therefore signal a poor convergence pattern as the moves on the support of f are more limited. Obviously, this is not always the case. For instance, when f is nearly flat, high acceptance rates are not indicative of any wrong behavior! But, unless f is completely flat (that is, it corresponds to a uniform target), there are parts of the domain to be explored where f takes smaller values and hence where the acceptance probabilities should be small. A high acceptance rate then indicates that those parts of the domain are not often (or not at all!) explored by the Metropolis–Hastings algorithm.

In contrast, if the average acceptance rate is low, the successive values of $f(y_t)$ often are small when compared with $f(x^{(t)})$, which corresponds to the scenario where the random walk moves quickly on the surface of f since it often reaches the "borders" of the support of f (or at least when the random walk explores regions with low probability under f). Again, a low acceptance rate does not mean that the chain explores the entire support of f. Even with

a small acceptance rate, it may miss an important but isolated mode of f. Nonetheless, a low acceptance rate is less of an issue, except from the computing time point of view, because it explicitly indicates that a larger number of simulations are necessary. Using the effective sample size as a convergence indicator (see Section 8.4.3) would clearly signal this requirement.

Example 6.10. (Continuation of Example 6.4) The three random walk Metropolis–Hastings algorithms of Figure 6.7 have acceptance rates equal to

```
[1] 0.9832
[1] 0.7952
[1] 0.1512
```

respectively. Looking at the histogram fit, we see that the medium acceptance rate does better but that the lowest acceptance rate still fares better than the highest one. ◀

The question is then to decide on a golden acceptance rate against which to calibrate random walk Metropolis–Hastings algorithms in order to avoid "too high" as well as "too low" acceptance rates. Roberts et al. (1997) recommend the use of instrumental distributions with *acceptance rates close to* $1/4$ *for models of high dimension and equal to* $1/2$ *for the models of dimension* 1 *or* 2. (This is the rule adopted in the adaptive `amcmc` package described in Section 8.5.2.) While this rule is not universal (in the sense that it was primarily designed for a Gaussian environment), we advocate it as a default calibration goal whenever it can be achieved (which is not always the case). For instance, if we consider the Metropolis–Hastings algorithm in Example 6.8, there is *no* acceptance rate since the acceptance probability is always equal to 1. However, since the proposal includes the current value in its support, the chain $\{\gamma^{(t)}\}$ has identical values in a row and thus an implicit acceptance (or renewal) rate. It is equal to 0.1805, much below the 0.25 goal, and the algorithm cannot be easily modified (for instance, by looking at more alternative moves around the current model) to reach this token acceptance rate.

6.6 Additional exercises

Exercise 6.7 Referring to Example 2.7, consider a $\mathcal{B}e(2.7, 6.3)$ target density.

a. Generate Metropolis–Hastings samples from this density using a range of independent beta candidates from a $\mathcal{B}e(1, 1)$ to a beta distribution with small variance. (*Note:* Recall that the variance is $ab/(a+b)^2(a+b+1)$.) Compare the acceptance rates of the algorithms.

b. Suppose that we want to generate a *truncated* beta $\mathcal{B}e(2.7, 6.3)$ restricted to the interval (c, d) with $c, d \in (0, 1)$. Compare the performance of a Metropolis–Hastings algorithm based on a $\mathcal{B}e(2, 6)$ proposal with one based on a $\mathcal{U}(c, d)$ proposal. Take $c = .1, .25$ and $d = .9, .75$.

Exercise 6.8 While q is a Markov kernel used in Algorithm 4, it is not the Markov kernel K of the algorithm.

1. Show that the probability that $X^{(t+1)} = x^{(t)}$ is
$$\underline{\rho}(x^{(t)}) = \int \left\{1 - \rho(x^{(t)}, y)\right\} q(y|x^{(t)})\, \mathrm{d}y\,.$$
2. Deduce that the kernel K can be written as
$$K(x^{(t)}, y) = \rho(x^{(t)}, y) q(y|x^{(t)}) + \underline{\rho}(x^{(t)}) \delta_{x^{(t)}}(y)\,.$$
3. Show that Algorithm 4 satisfies the *detailed balance condition* (6.3).

Exercise 6.9 Calculate the mean of a gamma $\mathcal{G}(4.3, 6.2)$ random variable using

a. Accept–Reject with a gamma $\mathcal{G}(4, 7)$ candidate;
b. Metropolis–Hastings with a gamma $\mathcal{G}(4, 7)$ candidate;
c. Metropolis–Hastings with a gamma $\mathcal{G}(5, 6)$ candidate.

In each case, monitor the convergence across iterations.

Exercise 6.10 Student's t density with ν degrees of freedom, $\mathcal{T}_\nu$, is given by
$$f(x|\nu) = \frac{\Gamma\left(\frac{\nu+1}{2}\right)}{\Gamma\left(\frac{\nu}{2}\right)} \frac{1}{\sqrt{\nu\pi}} \left(1 + x^2/\nu\right)^{-(\nu+1)/2}\,.$$
Calculate the mean of a t distribution with $\nu = 4$ degrees of freedom using a Metropolis–Hastings algorithm with candidate density

a. $\mathcal{N}(0, 1)$;
b. t with $\nu = 2$ degrees of freedom.

In each case monitor the convergence across iterations.

Exercise 6.11 Referring to Example 6.3:

1. Use the candidate given in this example to generate a sample $(a^{(i)}, b^{(i)}, c^{(i)})$, $i = 1, \ldots, 500$ with a Metropolis–Hastings algorithm. The data is from the dataset `cars`.
2. Monitor convergence and check autocorrelations for each parameter across iterations.
3. Make histograms of the posterior distributions of the coefficient estimates, and provide 95% confidence intervals.

Exercise 6.12 Still in connection with Example 6.3, show that the posterior distribution on (a, b, c, σ^{-2}) is a standard distribution made of a trivariate normal on (a, b, c) conditional on σ and the data and a gamma distribution on σ^{-2} given the data. (*Hint:* See Robert, 2001, or Marin and Robert, 2007, for details.)

Exercise 6.13 In 1986, the space shuttle Challenger exploded during takeoff, killing the seven astronauts aboard. The explosion was the result of an *O-ring* failure, a splitting of a ring of rubber that seals the parts of the ship together. The accident was believed to have been caused by the unusually cold weather (31^o F or 0^o C) at the time of launch, as there is reason to believe that the O-ring failure probabilities increase as temperature decreases. Data on previous space shuttle launches and O-ring failures is given in the dataset `challenger` provided with the `mcsm` package. The first column corresponds to the failure indicators y_i and the second column to the corresponding temperature x_i ($1 \le i \le 24$).

1. Fit this dataset with a logistic regression, where
$$P(Y_i = 1|x_i) = p(x_i) = \exp(\alpha + \beta x_i)/1 + \exp(\alpha + \beta x_i)\,,$$
using `R glm` function, as illustrated on page 21. Deduce the MLEs for α and β, along with standard errors.
2. Set up a Metropolis–Hastings algorithm with the likelihood as target using an exponential candidate for α and a Laplace (double-exponential) candidate for β. (*Hint:* Choose the parameters of the candidates based on the MLEs derived in a.)
3. Generate 5000 iterations of the Markov chain and construct a picture similar to Figure 6.6 to evaluate the variability of $p(x)$ minus the observation dots.
4. Derive from this sample an estimate of the probability of failure at 60^o, 50^o, and 40^o F along with a standard error.

Exercise 6.14 Referring to Example 6.4:

a. Reproduce the graphs in Figure 6.7 for difference values of δ. Explore both small and large δ's. Can you find an optimal choice in terms of autocovariance?
b. The random walk candidate can be based on other distributions. Consider generating a $\mathcal{N}(0,1)$ distribution using a random walk with a (i) Cauchy candidate, and a (ii) Laplace candidate. Construct these Metropolis–Hastings algorithms and compare them with each other and with the Metropolis–Hastings random walk with a uniform candidate.
c. For each of these three random walk candidates, examine whether or not the acceptance rate can be brought close to 0.25 for the proper choice of parameters.

Exercise 6.15 Referring to Example 6.9:

a. Write a Metropolis–Hastings algorithm to produce Figure 6.11. Note that n $\mathcal{L}(a)$ random variables can be generated at once with the `R` command
```
> ifelse(runif(n)>0.5, 1, -1) * rexp(n)/a
```
b. What is the acceptance rate for the Metropolis–Hastings algorithm with candidate $\mathcal{L}(3)$? Plot the curve of the acceptance rates for $\mathcal{L}(\alpha)$ candidates when α varies between 1 and 10. Comment.
c. Plot the curve of the acceptance rates for candidates $\mathcal{L}(0,\omega)$ when ω varies between $.01$ and 10. Compare it with those of the $\mathcal{L}(\alpha)$ candidates.
d. Plot the curve of the acceptance rates when the proposal is based on a random walk, $Y = X^{(t)} + \varepsilon$, where $\varepsilon \sim \mathcal{L}(\alpha)$. Once again, compare it with the earlier proposals.

Exercise 6.16 In connection with Example 6.8, compare the current implementation with an alternative where more values are considered at once according to the `R` code

```
> progam=matrix(gama[i,],ncol=lga,nrow=lga,byrow=T)
> probam=rep(0,lga)
> for (j in 1:lga){
+   progam[j,j]=1-gama[i,j]
+   probam[j]=lpostw(progam[j,],y,X,betatilde)}
> probam=exp(probam)
> sumam=sum(probam)
> probam=probam/sumam
> select=progam[sample(1:lga,1,prob=probam),]
```

a. Show that the acceptance probability is different from 1 and involves `sumam`.
b. Study the speed of convergence of the evaluation of the posterior probability of the most likely model in comparison with the implementation on page 191.

http://www.springer.com/978-1-4419-1575-7

Introducing Monte Carlo Methods with R
Robert, C.; Casella, G.
2010, XX, 284 p., Softcover
ISBN: 978-1-4419-1575-7