Other statistical tasks such as the hypothesis testing can also be accommodated.

This simple method was shown very efficient in a number of simulation examples we have tested on. We started with an 8×8 matrix of which we know the true answer. The cv^2 of our procedure, which approximates $\mathrm{perm}(A)$ accurately, was about 0.3. We then simulated a random matrix of 50×50. The cv^2 of our SIS procedure for this case only increased moderately to 0.5. It took about half a second to generate one permutation on a Sun Ultra 60 workstation. When we increased the matrix size to 100×100, the cv^2 was still impressively small, only about 0.7.

It should be noted that the methods of Kuznetsov (1996) and Beichl and Sullivan (1999) can all be viewed as SIS samplers. The computational and conceptual complexity of our approach is similar to that of Kuznetsov (1996). But our method tends to be more efficient. Compared with Beichl and Sullivan, our approach is much simpler, yet more efficient.

4.3 Counting 0-1 Tables with Fixed Margins

The problem of counting 0-1 tables with fixed margins was introduced in Section 3.4.2. It is a long-standing problem in the fields of applied mathematics and computer science and has been subject to active research for many years. In a recent work, Chakraborty, Chen, Diaconis, Holmes and Liu (2001) developed a number of techniques for solving the problem. These include a MCMC method, an exact counting method, and an SIS method. Here, we focus on their SIS approach.

Briefly, the SIS method begins by filling in columns (or rows) of the $m \times n$ table from left to right sequentially. After the first $t-1$ columns are filled, the row sums are updated and the tth column is filled in by sampling c_t of its m possible positions to put in 1's. These c_t positions are sampled according to a distribution that is related to the vector of the updated row sums. For example, the probability that positions $i_1, \ldots, i_{c_t}$ of the tth column are sampled can be proportional to

$$\left(\frac{r_{i_1}}{n - r_{i_1}} \times \cdots \times \frac{r_{i_{c_t}}}{n - r_{i_{c_t}}} \right)^{1+\delta}. \tag{4.8}$$

More choices of the sampling distribution and other mathematical details can be found in Chen (2001). This sequential sampling method gives us a rather accurate estimate for fairly large tables.

To test the method, they examined Darwin's finch data (Sanderson 2000), which, in the form of an "occurrence matrix," recorded the presence and absence for each of the 13 species of Galápagos finch in 17 islands of an east archipelago. This data is shown as in Table 4.3.

0	0	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	14
1	1	1	1	1	1	1	1	1	1	0	1	0	1	1	0	0	13
1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	0	0	14
0	0	1	1	1	0	0	1	0	1	0	1	1	0	1	1	1	10
1	1	1	0	1	1	1	1	1	1	0	1	0	1	1	0	0	12
0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	2
0	0	1	1	1	1	1	1	1	0	0	1	0	1	1	0	0	10
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1
0	0	1	1	1	1	1	1	1	1	0	1	0	0	1	0	0	10
0	0	1	1	1	1	1	1	1	1	0	1	0	1	1	0	0	11
0	0	1	1	1	0	1	1	0	1	0	0	0	0	0	0	0	6
0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	2
3	3	10	9	9	7	8	9	7	8	2	9	3	6	8	2	2	

TABLE 4.3. The finch dataset records the occurrences of 13 species(rows) of Galápagos finch in 17 islands (columns) of an east Pacific archipelago. Ecologists are interested in testing if the occurrence pattern is a random draw from the uniform distribution of all such patterns.

The problem of couting the number of 0-1 tables that have the same marginal sums as the finch data was first raised by Susan Holmes in Stanford. Based on 1000 sequentially simulated tables, the SIS method estimated that the total count of the 0-1 tables with the given margins is $(6.72 \pm .02) \times 10^{16}$. The computation took 12 minutes on a Pentium 400 machine. With more computing time, 10^8 tables were generated and the total number of the tables was estimated as 6.715×10^{16}. The coefficient of variation of the importance weights (defined as the sample variance divided by the square of the sample mean) was around 0.7. The "truth" they obtained by using an exact-counting algorithm is $6.71... \times 10^{16}$.

To push further for the algorithm, we simulated a 50×50 random table and recorded its marginal sums. The row sums are 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 5, 5, 6, 6, 7, 8, 8, 9, 10, 11, 12, 12, 12, 12, 12, 13, 14, 14, 15, 16, 18, 19, respectively, and the column sums are 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 4, 4, 4, 5, 5, 5, 5, 5, 6, 6, 6, 6, 6, 7, 7, 8, 8, 8, 8, 9, 9, 9, 10, 10, 11, 12, 13, 13, 13, 14, 14, 14, respectively. The column sums were ordered from the largest to the smallest and the columns were sampled sequentially according to (4.8) with $\delta = 0$. This gave us a cv^2 of around 0.2. Based on $1,000$ samples, which took about 5 minutes to generate, we estimated that the total number of 0-1 tables with these marginal sums is $(8.9 \pm 0.1) \times 10^{242}$. Based on $10,000$ samples, the estimate was improved to 8.78×10^{242} with standard error 0.05×10^{242}. This example shows that the SIS method is still very efficient even for large tables.

4.4 Bayesian Missing Data Problems

4.4.1 Murray's data

Let us revisit the Gaussian missing data problem described in Section 2.5.7. The previous approach was based on the plain importance sampling in which the trial density was chosen as the posterior distribution based on the first four complete observations. Here, we consider an SIS approach.

Recall that the covariance matrix is assigned the Jeffreys non-informative prior (2.13) The posterior distribution of Σ given complete data is

$$p(\mathcal{Z}|\text{complete data}) \;\propto\; \mathcal{Z}^{\frac{3}{2}-1} \;\; \exp\left\{-\frac{1}{2}\,\mathrm{tr}[\mathcal{Z}\cdot S]\right\},$$

where $S = (s_{ij})_{2\times 2}$ is the uncorrected sum of squares matrix and $\mathcal{Z} = \Sigma^{-1}$.

Let the complete data be $y_1, \ldots, y_{12}$, where $y_t = (y_{t,1}, y_{t,2})$ for $t = 1, \ldots, 12$. Thus, the $y_{t,2}$ are missing for observations 5 to 8 and the $y_{t,1}$ are missing for observations 9 to 12. The predictive distribution of y_{t+1} given $\mathbf{y}_t = (y_1, \ldots, y_t)$ is

$$y_{t+1}|\mathbf{y}_t \;\sim t_2\left(0,\; \frac{S_t}{t-1},\; t-1\right),$$

where $S_t = (s_{ij})$, $s_{ij} = \sum_{s=1}^{t} y_{s,i}y_{s,j}$, and t_2 is the bivariate t-distribution. It follows that, conditional on $\mathbf{y}_t = (y_1, \ldots, y_t)$, the marginal and conditional distributions are

$$\begin{aligned} y_{t+1,1} \mid \mathbf{y}_t \;&\sim\; t_1\left(0, \frac{s_{11}}{t-1}, t-1\right), \\ y_{t+1,2} \mid \mathbf{y}_t, y_{t+1,1} \;&\sim\; t_1\left[\frac{s_{12}}{s_{11}}y_{t+1,1},\; \frac{|S_t|}{ts_{11}}\left(1+\frac{y_{t+1,1}^2}{s_{11}}\right),\; t\right], \end{aligned}$$

respectively. Similar results can be obtained for conditional distributions $[y_{t+1,2} \mid \mathbf{y}_t]$ and $[y_{t+1,1} \mid \mathbf{y}_t, y_{t+1,2}]$. Based on these distributional results, Steps A and B of the sequential imputation (Section 3.2.2) can both be easily implemented. Figure 4.5 gives the exact posterior distribution of ρ and the approximated posterior distribution based on sequential imputation. The approximation is a weighted mixture of $m = 1000$ complete-data posterior distributions.

The complete-data posterior distribution of ρ is still difficult to compute since it has the form

$$p(\rho \mid \mathbf{y}) \propto \; (1-\rho^2)^{\frac{\nu-2}{2}} \int_0^\infty \omega^{-1}\left(\omega + \frac{1}{\omega} - 2\rho r\right)^{-(\nu+1)} d\omega, \qquad (4.9)$$

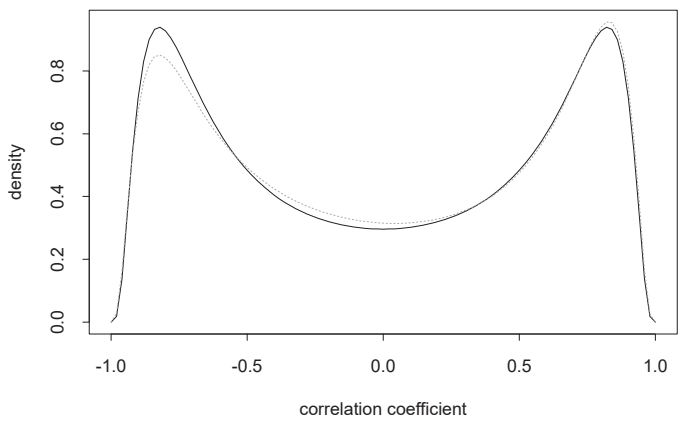

FIGURE 4.5. An approximation to the posterior distribution of the sample correlation coefficient ρ for Murray's data. Solid line: true density; dashed line: approximation.

where $\mathbf{y}$ is the completed data after the missing part being imputed, r is the sample correlation coefficient, and $\nu = n - 2 = 10$. To avoid numerical integration in (4.9), we can use the algorithm of Odell and Feiveson (1966) to generate observations from the inverse Wishart distribution, which gives rise to a sample from $p(\rho \mid \mathbf{y})$. Note that we do not have to draw from an inverse Wishart distribution *during* sequential imputation.

4.4.2 Nonparametric Bayes analysis of binomial data

Kong et al. (1994) and Liu (1996b) considered the following *hierarchical* binomial model:

$$\begin{aligned} y_t &\sim \text{Binomial}(l_t, \zeta_t) \\ \zeta_t &\overset{\text{i.i.d.}}{\sim} F, \quad 1 \le t \le n. \end{aligned}$$

Our interests are in drawing inference about both F and the ζ_t's based on the observed data y_t. As a concrete example, consider a dataset (Figure 4.6) of n thumbtacks randomly drawn from a certain population. Here, ζ_t can be interpreted as the inherent probability for the tth tack to point up when being flicked. The observed data for the tth tack is y_t, the number of times the tack landed point up out of a total of l_t flicks. The data in Figure 4.6 was generated by Beckett and Diaconis (1994), of whom each flicked 16 different thumbtacks on 10 different surfaces. For each person-tack-surface combination (a total of $2 \times 16 \times 10$ scenarios), the experiment was repeated 9 times. For simplicity, we treat the data as though they came from 320 different tacks and each being flicked 9 times independently. A histogram of the data (the y_i) is shown in Figure 4.6.

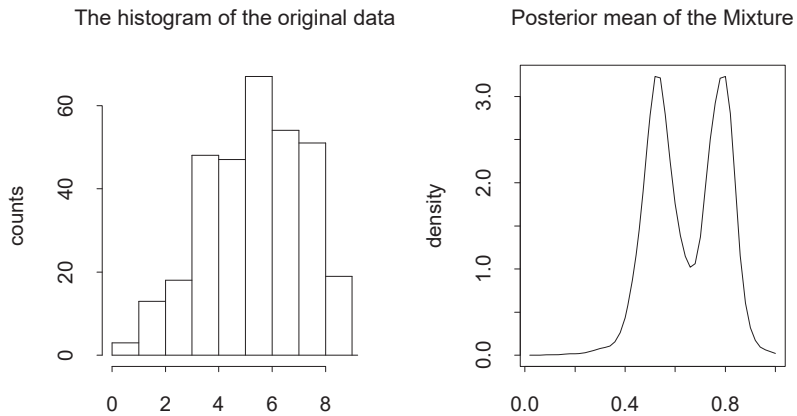

FIGURE 4.6. Left: The histogram of the tack data produced by Beckett and Diaconis (1994). Right: The predictive distribution of ζ for a future tack (which is also the posterior mean of F).

Different from the method outlined in Section 1.8, we take a *nonparametric* Bayes approach by assigning a *Dirichlet process* prior, $\mathcal{D}(\alpha)$, to the infinite-dimensional parameter F. The hyperparameter α is a finite positive measure on interval $[0,1]$. We define the norm $\|\alpha\|$ as the total measure of the support of F. Note that $\mathcal{D}(\alpha)$ is a probability measure on $\mathcal{P}$, the set of all probabilities on the Borel sets of $[0,1]$. Readers not familiar with this special class of Dirichlet process distribution and nonparametric Bayes inference are referred to Ferguson (1974). In the tacks example, we may be interested in estimating ζ_t for tack t, given its performance in 9 independent flips, or in obtaining the posterior mean of the unknown distribution F [i.e., $E(F \mid \zeta_1, \ldots, \zeta_n)$].

For simplicity, suppose α is the uniform measure on $[0,1]$ (hence, the norm $\|\alpha\| = 1$). Then, a key result (Ferguson 1974) is that conditional on $\zeta_1, \ldots, \zeta_{t-1}$, the posterior distribution of F is again a Dirichlet process, $\mathcal{D}(\alpha_{t-1})$ with $\alpha_{t-1} = \alpha + \sum_{i=1}^{t-1} \delta_{\zeta_i}$, where δ_{ζ_i} is a delta measure with mass 1 located at ζ_s. Thus, the predictive distribution for the next thumbtack, ζ_t, is

$$\zeta_t | \zeta_1, \ldots, \zeta_{t-1} \sim \frac{1}{t} \left(\alpha + \sum_{i=1}^{t-1} \delta_{\zeta_i} \right), \tag{4.10}$$

which should be interpreted as a probabilistic mixture of α and delta measures concentrated at the ζ_i's (Antoniak 1974). Note that the posterior mean of F, $E(F \mid \zeta_1, \ldots, \zeta_t)$, is the same as (4.10). It is easy to see from

(4.10) that

$$[\zeta_t|\zeta_1,\ldots,\zeta_{t-1},y_t] = \frac{1}{Z}\Bigg[B(y_t+1,l_t-y_t+1)\text{Beta}(y_t+1,l_t-y_t+1) + \sum_{i=1}^{t-1}\zeta_i^{y_t}(1-\zeta_i)^{l_t-y_t}\delta_{\zeta_i}\Bigg] \tag{4.11}$$

where

$$B(y_t+1,l_t-y_t+1) = \int_0^1 \zeta^{y_t}(1-\zeta)^{l_t-y_t}d\zeta = \frac{\Gamma(y_t+1)\Gamma(l_t-y_t+1)}{\Gamma(l_t+2)}$$

is the Beta function, $\text{Beta}(\cdot,\cdot)$ is the standard Beta distribution, and

$$Z = B(y_t+1,l_t-y_t+1) + \sum_{i=1}^{t-1}\zeta_i^{y_t}(1-\zeta_i)^{l_t-y_t}$$

is the normalizing constant. Note that (4.11) is a mixture of a Beta distribution and discrete point masses. From (4.10), we also get

$$p(y_t|\zeta_1,\ldots,\zeta_{t-1}) = \frac{1}{t}B(y_t+1,l_t-y_t+1) + \frac{1}{t}\sum_{i=1}^{t-1}\zeta_i^{y_t}(1-\zeta_i)^{l_t-y_t},$$

which is the term needed for updating the importance sampling weights. Hence, both Steps A and B of sequential imputation can be easily implemented (Liu 1996b).

Note that a direct application of the data augmentation algorithm (Section 6.4) is difficult because sampling F from

$$[F \mid \zeta_1,\ldots,\zeta_n] = \mathcal{D}\left(\alpha + \delta_{\zeta_1} + \cdots + \delta_{\zeta_n}\right)$$

is infeasible. [Some approximations exist; see Doss (1994)]. Escobar (1994) described a "collapsed" Gibbs sampling algorithm (Section 6.7) in which the sampling of F is not needed. His method also takes advantage of the simplicity of the predictive distributions (4.10) and (4.11). For a related problem where the $\zeta_t's$ are ordered, Gelfand and Kuo (1991) used a similar idea to avoid sampling the infinite-dimensional parameter F.

The sequential imputation procedure derived from using (4.10) and (4.11) was applied to the thumbtack data with α being uniform and $\|\alpha\| = 1$. Figure 4.6 displays the posterior mean of the unknown density function F, $E(F \mid \mathbf{y})$. Liu (1996b) studied the sensitivity of this result to the prior assumption and presented a method to estimate the norm of α.

A more sophisticated "second-generation" SIS procedure for nonparametric Bayes problems was proposed by MacEachern et al. (1999). They noticed that when sampling ζ_t from (4.11), either a new ζ is produced

from the Beta distribution or a previous ζ is drawn. As a consequence, the ζ_t forms natural clusters because of the Dirichlet process assumption on F. The plain SIS procedure tends to fix the location of these clusters at a relatively early stage, which is inefficient because an early cluster location may be invalidated by the subsequent new observations. The new SIS method introduces the *cluster* indicator variable I_t, which tells us whether ζ_t should be in one of the clusters formed by $\zeta_1, \ldots, \zeta_{t-1}$ or form a new cluster. Conditional on the indicators $I_1, \ldots, I_n$, one can integrate out all the ζ_t. The new sequential importance sampler is then constructed on the space of $(I_1, \ldots, I_n)$. When applied to the tack data under the same prior setting, the new method yields a substantial improvement over the plain SIS described earlier in this section (a cv^2 of 11 for the new method versus 43 for the old one).

4.5 Problems in Signal Processing

4.5.1 Target tracking in clutter and mixture Kalman filter

Tracking a target in clutter is of interest to engineers and computer scientists. The problem has received much attention recently since the proposal of the bootstrap filter (Avitzour 1995, Gordon et al. 1995). Here, we use the simple one-dimensional tracking problem in Avitzour (1995) to show how the SIS methods can be applied in this area.

Avitzour (1995) modeled the tracking problem as a state-space model with the state variable $x_t=(x_{t,1}, x_{t,2})$, where $x_{t,1}$ is the location of the target on a straight line and $x_{t,2}$ is the target velocity. The x_t evolve in the following way:

$$\begin{aligned} x_{t,1} &= x_{t-1,1} + x_{t-1,2} + \frac{1}{2} w_t, \\ x_{t,2} &= x_{t-1,2} + w_t, \end{aligned}$$

where the noise term w_t are i.i.d. and follow distribution $N(0, q^2)$. If we could identify the object (without confusing it with others) at all times, then our observation would have been the object's location $x_{t,1}$ plus a small random noise. In other words, we observe

$$z_t = x_{t,1} + v_t,$$

where the v_t are i.i.d. Gaussian noises with distribution $N(0, r^2)$. If all the z_t are directly observable, then the tracking problem can be solved satisfactorily by using a Kalman filter (Bar-Shalom and Fortmann 1988).

When confusing objects are present in the detection window, however, our observation at time t becomes y_t, which is a vector of length m_t, where m_t is the total number of observed objects in the window and each component of y_t represents an object's position. Among these m_t measured

locations, at most one corresponds to the true target we are interested in tracking (i.e., is equal to z_t). The occurrence of the confusing objects is assumed to follow a Poisson process with rate α. We further assume that there is only a probability $p_d < 1$ for the observation window to actually include the target's location (z_t) in y_t. Therefore, if the range of the detection window is Δ, the distribution of m_t is Bernoulli(p_d)+Poisson($\lambda\Delta$) and the false signals are uniformly distributed in the detection region. By introducing an indicator variable I_t,

$$I_t = \begin{cases} 0 & \text{if the target object is not in the detection range} \\ k & \text{if the } k\text{th object corresponds to the target,} \end{cases}$$

we can formulate this problem as a state-space model; that is, when $I_t = 0$,

$$p(y_t \mid x_t, I_t = 0) = \Delta^{-m_t} \frac{(\lambda\Delta)^{m_t}}{m_t!} e^{-\lambda\Delta} = \frac{\lambda^{m_t}}{m_t!} e^{-\lambda\Delta};$$

and when $I_t = k$,

$$p(y_t \mid x_t, I_t = k) = \frac{\lambda^{m_t - 1}}{(m_t - 1)!} e^{-\lambda\Delta} \frac{1}{\sqrt{2\pi} r} \exp\left\{ -\frac{(y_{t,k} - x_t)^2}{2r^2} \right\}.$$

Since *a priori* $P(I_t = 0) = 1 - p_d$ and $P(I_t = k) = p_d/m_t$, we have

$$f_t(y_t, I_t \mid x_t) \propto \begin{cases} (1 - p_d)\lambda & \text{if } I_t = 0; \\ p_d (2\pi r^2)^{-1/2} \exp\left[-\dfrac{(y_{t,k} - x_t)^2}{2r^2} \right] & \text{otherwise.} \end{cases}$$

Since I_t is not observable, we need to sum out the I_t to obtain the *observation distribution* $f_t(y_t \mid x_t)$.

Using $q(x_t \mid x_{t-1})$ to denote the state evolution relationship, we can obtain a sequence of auxiliary distributions, $\pi_t(x_1, \ldots, x_t)$, as in (3.8):

$$\pi_t(\mathbf{x}_t) \propto f_t(y_t \mid x_t) q(x_t \mid x_{t-1}) \pi_{t-1}(\mathbf{x}_{t-1})$$

and

$$\pi_t(x_t) \propto \int f_t(y_t \mid x_t) q(x_t \mid x_{t-1}) \pi_{t-1}(x_{t-1}) dx_{t-1}.$$

The second equation holds because of the Markovian structure among the x_t. With this formulation, the current position of the target can be estimated as $E_{\pi_t}(x_t)$.

Conditional on the current value of x_{t-1}, we can easily simulate x_t from the state equation $q(\ )$. Given a sampled value of x_t, the computation of its weight, $f_t(y_t \mid x_t)$, is also easy. Thus, the *bootstrap filter* (Section 3.3) can be easily applied to this problem. However, if we pay a little more attention to the model's special structure, we can come up with a much more efficient algorithm.

Let $\mathbf{\Lambda}_t = (I_1, \ldots, I_t)$ be called a *trajectory* up to time t. We note that an important feature of our tracking model is that if we know the values of the trajectory $\mathbf{\Lambda}_t$ of the target, the tracking system becomes linear and Gaussian and the computation of the Bayes estimator, $E(x_t \mid y_1, \ldots, y_t)$, can be achieved *exactly* by a standard Kalman filter. Therefore, conditional on $\mathbf{\Lambda}_t$, we can integrate out $\mathbf{x}_t$ exactly. This feature enables us to design a SIS system only on the reduced space of $\mathbf{\Lambda}_t$ (Liu and Chen 1998, Chen and Liu 2000a). This approach is a vivid demonstration of the power of *marginalization* technique (Section 3.4.6). Since this new algorithm takes the form of mixing over a number of Kalman filters, we call the method *mixture Kalman filter* (MKF). Chen and Liu (2000a) give more details. Briefly, a general MKF algorithm can be stated as follows.

Suppose at time $t-1$, we have sampled m trajectories $\{\mathbf{\Lambda}_{t-1}^{(1)}, \ldots, \mathbf{\Lambda}_{t-1}^{(m)}\}$ with weights $w_{t-1}^{(1)}, \ldots, w_{t-1}^{(m)}$. For each trajectory $\mathbf{\Lambda}_{t-1}^{(j)}$, we can compute via a Kalman filter the mean vector $\boldsymbol{\mu}_{t-1}^{(j)}$ and the covariance matrix $\Sigma_{t-1}^{(j)}$ for the target. These are the sufficient statistics because the system is linear and Gaussian given $\mathbf{\Lambda}_{t-1}$. We denote $KF_{t-1}^{(j)} = (\boldsymbol{\mu}_{t-1}^{(j)}, \Sigma_{t-1}^{(j)})$. At time t, we run the following MKF updates: For $j = 1, \ldots, m$,

- generate $I_t^{(j)}$ from a trial distribution $g(I_t \mid \mathbf{\Lambda}_{t-1}^{(j)}, KF_{t-1}^{(j)}, y_t)$;
- conditional on each $\{KF_{t-1}^{(j)}, y_t, I_t^{(j)}\}$, obtain $KF_{t+1}^{(j)}$ by a one-step Kalman filter (Chen and Liu 2000a);
- update the new weight as $w_t^{(j)} = w_{t-1}^{(j)} \times u_t^{(j)}$, where

$$u_t^{(j)} = \frac{p(\mathbf{\Lambda}_{t-1}^{(j)}, I_t^{(j)} \mid \mathbf{y}_t)}{p(\mathbf{\Lambda}_{t-1}^{(j)} \mid \mathbf{y}_{t-1}) g(I_t^{(j)} \mid \boldsymbol{\lambda}_{t-1}^{(j)}, KF_{t-1}^{(j)}, y_t)};$$

- if the coefficient of variation of the w_t exceeds a threshold value, we resample a new set of KF_t from $\{KF_t^{(1)}, \ldots, KF_t^{(m)}\}$ with probability proportional to the weights $w_t^{(j)}$.

Smith and Winter (1978) proposed a deterministic filtering method, called *split-track filter* (STF), which has a similar flavor to the MKF we just outlined. In STF, one always keeps m trajectories of the latent indicators. At a future time step, it evaluates the likelihoods of all possible propagations from the m trajectories kept at the previous step, then finds and keeps the m new trajectories with the highest likelihood values. In contrast, our MKF selects these trajectories randomly, according to the weights (which is the predictive likelihood value), and uses the associated weights to measure how good each trajectory is. The important step of resampling is naturally built into MKF, which can overcome some weaknesses of STF. More sophisticated sampling and estimation methods can also be incorporated.

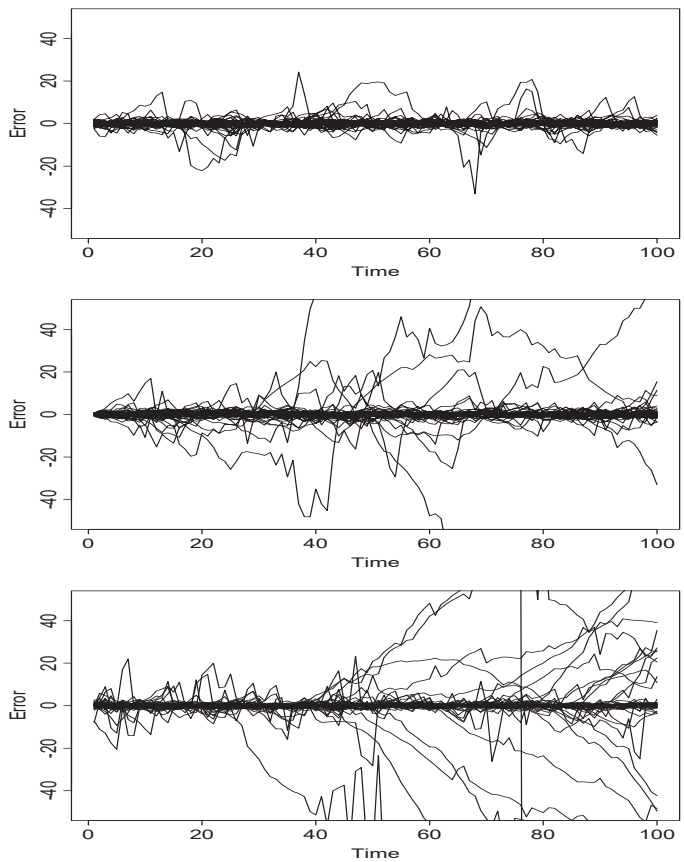

FIGURE 4.7. The tracking errors of 50 runs of the MKF (top), a sequential importance sampler (middle), and the split-track filter (bottom) for a simulated target moving system.

Figure 4.7 shows the plots of tracking errors (estimated location − true location) of 50 simulated runs, with $r^2 = 1.0$, $q^2 = 0.1$, $p_d = 0.9$, and $\lambda = 0.1$. These parameter combination is slightly different from that of Avitzour (1995), with smaller clutter density but larger state equation variance. With their configuration, the results are similar, but the differences between different procedures are smaller. Five hundred streams (m=500) were used, with resampling done at every step. The top part of Figure 4.7 resulted from using the MKF method, and the middle part shows the result from using a slightly improved bootstrap filter (Avitzour 1995). We also implemented the split-track filter for this problem, which, at each step, saves the 500 trajectories with the highest likelihood values (bottom part).

The generality of the MKF method is more thoroughly explored in Chen and Liu (2000a).

4.5.2 Digital signal extraction in fading channels

Many mobile communication channels can be modeled as Rayleigh flat-fading channels, which have the following form:

$$\begin{array}{rl} \text{State equations:} & \left\{ \begin{array}{rcl} \mathbf{x}_t & = & F\mathbf{x}_{t-1} + Ww_t \\ \alpha_t & = & G\mathbf{x}_t \\ s_t & \sim & p(\cdot \mid s_{t-1}), \end{array} \right. \\ \text{Observation equation:} & y_t = \alpha_t s_t + Vv_t, \end{array}$$

where s_t are the input digital signals (symbols), y_t are the received complex signals, and α_t are the unobserved (changing) fading coefficients. Both w_t and v_t are complex Gaussian with identity covariance matrices. It is important to note that this model has some similarity to the tracking problem; that is, given the input signals s_t, the system is linear in $\mathbf{x}_t$ and y_t. Therefore, we can design a MKF which focuses solely on the s_t (with $\mathbf{x}_t$ integrated out). The algorithmic detail is similar to the MKF updating steps described in the previous subsection and will be omitted here. Readers interested in more details and related applications are referred to Chen and Liu (2000a) and Chen, Wang and Liu (2000).

Consider a special example in which input signals are binary (i.e., $s_t = -1$ or $+1$). The fading coefficient takes complex values, with independent real and imaginary parts following the same state equation. Simulation were done with the following configurations:

$$F = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0.94 & -2.88 & 2.94 \end{pmatrix}; \quad G' = 10^{-4} \begin{pmatrix} 0.04 \\ 0.11 \\ 0.11 \\ 0.04 \end{pmatrix}; \quad W = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix};$$

and $V = r$. That is, both of the real and the imaginary parts of α_t follow an ARMA(3,3) process

$$\begin{aligned} & \alpha_t - 0.94\alpha_{t-1} + 2.88\alpha_{t-2} - 2.94\alpha_{t-3} \\ = \; & 0.04e_t + 0.11e_{t-1} + 0.11e_{t-2} + 0.04e_{t-3} \end{aligned}$$

where $e_t \sim N(0, 0.01^2)$. In the communication literature, this is called a (low-pass) Butterworth filter of order 3 with cutoff frequency 0.01. It is normalized to have a stationary variance 1.

We are interested in estimating the differential code $d_t = s_t s_{t-1}$. Figure 4.8 shows the bit error rate of different signal-to-noise ratios (SNR), using a form of the MKF, the differential detection $\hat{d}_t = \text{sgn}[\text{real}(y_t y_{t-1}^*)]$

and a lower bound. The lower bound is obtained using the true fading coefficients α_t and $\hat{d}_t = \text{sgn}[\text{real}(\alpha_t^* y_t y_{t-1}^* \alpha_{t-1})]$. The Monte Carlo sample size m was 100 for the MKF. We also include the result of a delayed estimation, in which s_t is estimated using the samples $s_t^{(j)}$ generated by MKF and the weight $w_{t+1}^{(j)}$ at time $t+1$ (Liu and Chen 1998). This delayed estimation is able to utilize the substantial information contained in the future information y_{t+1}, hence more accurate, due to the strong memory in the fading channel.

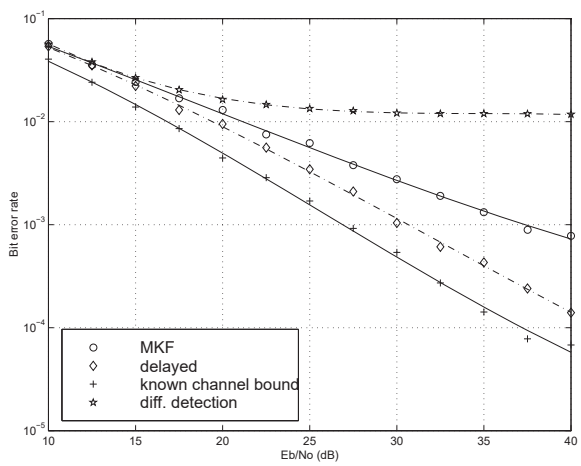

FIGURE 4.8. The bit error rate of extracting differential binary signals from a fading channel using MKF and differential detection. A lower bound that assumes the exact knowledge of the fading coefficients is also shown.

We can see that the simple differential detection works very well in low-SNR cases and no significant improvement can be expected. However, it has an apparent bit error rate floor for high-SNR cases. The MKF managed to break that floor, by using the structure of the fading coefficients.

4.6 Problems

1. Prove that the method described in Section 4.2 is valid. In particular, show that the weight updating strategy is correct. That is, the average of the weights indeed provide us an unbiased estimate on perm(A).

2. Describe how to use the SIS method of Section 4.2 to conduct the hypothesis test of independence for the astronomy data in Section 1.7.

3. Test to see whether you can find a better sampling distribution than (4.6) for approximating permanents. One possible choice is $P(\sigma(1) =$

$s) \propto (r_s - 1)^{-\alpha}$ for some $\alpha > 0$. But it might be possible to use a more complex function of all the r_j.

4. Study the "scanning future" method (Meirovitch 1982, Meirovitch 1985). Investigate its potential for the nonlinear filtering problem and the molecular structural optimization problem.

5. Investigate further the difference between the method of "split-track filter" and the mixture Kalman filter.

5
Metropolis Algorithm and Beyond

We have discussed in the previous chapters the important role of Monte Carlo methods in evaluating integrals and simulating stochastic systems. The most critical step in developing an efficient Monte Carlo algorithm is the simulation (sampling) from an appropriate probability distribution $\pi(\mathbf{x})$. When directly generating independent samples from $\pi(\mathbf{x})$ is not possible, we have to either opt for an *importance sampling* strategy, in which random samples are generated from a trial distribution *different* from (but close to) the target one and then weighted according to the importance ratio; or produce statistically *dependent* samples based on the idea of *Markov chain Monte Carlo* sampling. The importance sampling approach and its extensions have been discussed in Chapters 2–4. In this chapter, we introduce the cornerstone of all Markov chain-based Monte Carlo methods: the algorithm proposed in a very short paper (four pages) by Nicholas Metropolis, Arianna Rosenbluth, Marshall Rosenbluth, Augusta Teller, and Edward Teller in 1953.

Let $\pi(\mathbf{x}) = Z^{-1}\exp\{-h(\mathbf{x})\}$ be the target distribution under investigation (presumably all probability distribution functions can be written in this form), where the normalizing constant, or the *partition function*, Z, is often unknown to us. In principle, $Z = \int \exp\{-h(\mathbf{x})\}d\mathbf{x}$ is "knowable," but evaluating Z is no easier (and often harder) than the original problem of simulating from π. Motivated by computational problems in statistical physics, Metropolis et al. (1953) introduced the fundamental idea of evolving a Markov process to achieve the sampling of π. This idea, later known as the *Metropolis algorithm*, is of great simplicity and power — its variations and extensions have now been widely adopted by researchers in many

different scientific fields, including biology, chemistry, computer sciences, economics, engineering, material sciences, physics, statistics, and others.

The Metropolis algorithm can be used to generate random samples from virtually any target distribution $\pi(\mathbf{x})$ known up to a normalizing constant, regardless of its analytical complexity and its dimensionality. Although this claim is true in theory, a potential problem with these Markov-chain-based Monte Carlo methods is that the resulting samples are often highly correlated. Therefore, the estimates resulting from these samples tend to have greater (often much greater) variances than those resulting from independent samples. Various attempts have been made in many different fields (e.g., physics, chemistry, structural biology, and statistics) to overcome these limitations. Interesting research topics include the design of Markov-chain-based Monte Carlo algorithms that can generate less correlated samples, the finding of more efficient ways to use generated Monte Carlo samples, the assessment of statistical errors of the estimates, etc. Detailed discussions regarding these topics will be given in the later chapters.

5.1 The Metropolis Algorithm

The basic idea of the Metropolis algorithm is to simulate a Markov chain in the state space of $\mathbf{x}$ so that the limiting/stationary/equilibrium[1] distribution of this chain is the target distribution π. Note that in traditional Markov chain analysis, one is often given a *transition rule*[2] and is interested in knowing what the stationary distribution is (see Section 12.1 for an introduction to Markov chains), whereas in Markov chain Monte Carlo simulations, one *knows* the equilibrium distribution and is interested in prescribing an efficient transition rule so as to reach this equilibrium.

Starting with any configuration $\mathbf{x}^{(0)}$, the Metropolis algorithm proceeds by iterating the following two steps.

M1: Propose a random "unbiased perturbation" of the current state $\mathbf{x}^{(t)}$ so as to generate a new configuration $\mathbf{x}'$. Mathematically, $\mathbf{x}'$ can be seen as being generated from a *symmetric* probability transition function[3] (often called the *proposal function* or *trial proposal*) $T(\mathbf{x}^{(t)}, \mathbf{x}')$ [i.e., $T(\mathbf{x}, \mathbf{x}') = T(\mathbf{x}', \mathbf{x})$]; calculate the change $\Delta h = h(\mathbf{x}') - h(\mathbf{x}^{(t)})$.

[1] There are subtle differences among these three concepts: limiting, stationary, or equilibrium distributions. But for most practical examples, they are the same thing. See the Appendix and Karlin and Taylor (1998) for more details.

[2] A *transition rule* is a probabilistic law, or more precisely, a conditional distribution, that dictates the chances of moving from one point in the state space to another.

[3] A function $T(\mathbf{x}, \mathbf{y})$ is called a probability transition function if it is non-negative and satisfies $\sum_{\text{all } \mathbf{y}} T(\mathbf{x}, \mathbf{y}) = 1$, for all $\mathbf{x}$.

M2: Generate a random number $U \sim$ Uniform[0,1]. Let $\mathbf{x}^{(t+1)} = \mathbf{x}'$ if

$$U \leq \pi(\mathbf{x}')/\pi(\mathbf{x}^{(t)}) \equiv \exp(-\Delta h),$$

and let $\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)}$ otherwise.

A more casual but perhaps better known description of the Metropolis algorithm is as follows: At each iteration, (a) a small but random perturbation of the current configuration is made, (b) the "gain" in an objective function [i.e., $-h(\mathbf{x})$] resulting from this perturbation is computed, (c) a random number U is generated independently, and (d) the new configuration is accepted if $\log(U)$ is smaller than or equal to the "gain" and is rejected otherwise. Heuristically, the Metropolis algorithm is constructed based on a "trial-and-error" strategy.

Metropolis et al. (1953) restricted their choices of the "perturbation rule" to the symmetric ones. According to this perturbation rule, the chance of obtaining $\mathbf{x}'$ from perturbing $\mathbf{x}$ is always equal to that of obtaining $\mathbf{x}$ from perturbing $\mathbf{x}'$. Intuitively, this means that there is no "trend bias" at the proposal stage. Mathematically, this symmetry requirement can be expressed as

$$T(\mathbf{x}, \mathbf{x}') = T(\mathbf{x}', \mathbf{x}).$$

The Metropolis scheme has been extensively used in statistical physics over the past five decades and is the cornerstone of all Markov chain Monte Carlo (MCMC) techniques recently adopted and further developed in the statistics community.

As an illustration, we consider the simulation of a simple hard-shell ball model for gas. In this model, the positions of K nonoverlapping hard-shell balls, with equal diameters, are required to be uniformly distributed in the box $[0, A] \times [0, B]$. Let $(X, Y) = \{(x_i, y_i),\ i = 1, \ldots, K\}$ denote the positions of these balls. The target distribution of interest, $\pi(X, Y)$, is then *uniform* for all allowable configurations (i.e., nonoverlapping and within the box). The Metropolis algorithm for this simulation can be implemented as follows: (a) Pick a ball at random, say, the ball at position (x_i, y_i); (b) propose to move this ball to a new position $(x_i', y_i') = (x_i + \delta_1, y_i + \delta_2)$, where $\delta_j \sim N(0, \sigma_0^2)$; and (c) accept the proposed position (x_i', y_i') if it does not violate the constraints, otherwise stay put. With $K = 6$, $d = 0.8$, $A = B = 3.5$, and starting positions of the balls at regular grids, we adjusted σ_0^2 to 0.5, which gave us an acceptance rate of about 30%. Figure 5.1 shows two snapshots of this simulation: The first one was taken after 1000 iterations, and the second one taken after 2000 iterations.

Another example is the simulation of the Ising model. As described in Section 1.3, the Ising model takes a probabilistic form

$$\pi(\mathbf{x}) \propto \exp\{-U(\mathbf{x})/\beta T\},$$

where $\mathbf{x} = (x_s,\ s \in \mathcal{L})$ and $x_s = \pm 1$, $\mathcal{L}$ is a lattice space; and $U(\mathbf{x}) = -J \sum_{s \sim s'} \delta_{x_s = x_{s'}}$. To simulate from this model, one needs to prescribe a

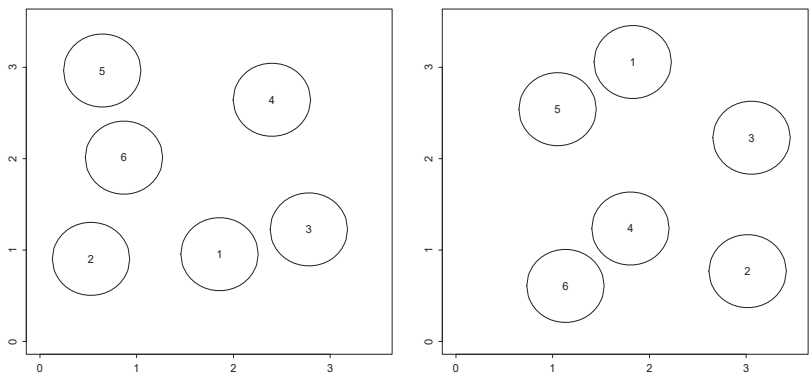

FIGURE 5.1. The simulation of a hard-shell ball model by the Metropolis algorithm. Left: after 1000 iterations; right: after 2000 iterations.

way to "perturb" the current configuration. A convenient proposal transition function is as follows: Pick a site, say, σ, at random, and negate its current value x_σ to $-x_\sigma$. Thus, the proposed new configuration $\mathbf{x}'$ differs from the initial one $\mathbf{x}$ only by a single site.

To be more concrete, consider the simulation of a one-dimensional Ising model in which $\mathbf{x} = (x_1, \ldots, x_d)$ and $U(\mathbf{x}) = -J \sum_{s=1}^{d-1} x_s x_{s+1}$. By letting $J = \mu\beta T$, we write the target distribution as

$$\pi(\mathbf{x}) = \frac{1}{Z} \exp\left\{ \mu \sum_{s=1}^{d-1} x_s x_{s+1} \right\}. \tag{5.1}$$

Suppose the current configuration is $\mathbf{x}^{(t)} = (x_1^{(t)}, \ldots, x_d^{(t)})$; then the next state $\mathbf{x}^{(t+1)}$ is produced by the Metropolis rule as follows:

- Choose a site, say, the jth site at random and set its current spin $x_j^{(t)}$ to the opposite. Thus, the newly proposed configuration is $\mathbf{x}' = (x_1^{(t)}, \ldots, -x_j^{(t)}, \ldots, x_d^{(t)})$.
- Compute the Metropolis ratio. In this case, it is easy to check that the random flipping process is completely symmetric; hence, $T(\mathbf{x}^{(t)}, \mathbf{x}') = T(\mathbf{x}', \mathbf{x}^{(t)})$, and, when $j \neq 1$ or d,

$$r = \pi(\mathbf{x}')/\pi(\mathbf{x}^{(t)}) = \exp\left\{ -2\mu x_j^{(t)} \left(x_{j-1}^{(t)} + x_{j+1}^{(t)} \right) \right\}.$$

- Simulate an independent uniform random variable U. Let $\mathbf{x}^{(t+1)} = \mathbf{x}'$ if $U \leq r$ and let $\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)}$ otherwise.

Since this distribution has many components, it is difficult to have an overall sense of how the chain moves in the space. In Figure 5.2(a), we plot the

traces of the total magnetization, defined as $M^{(t)} = \sum_{i=1}^{d} x_i^{(t)}$, for the first 2000 steps of a simulation. In this example, we took $\mu = 1$, $d = 50$, and the "all-up" starting configuration [i.e., $\mathbf{x}^{(0)} = (1, \ldots, 1)$].

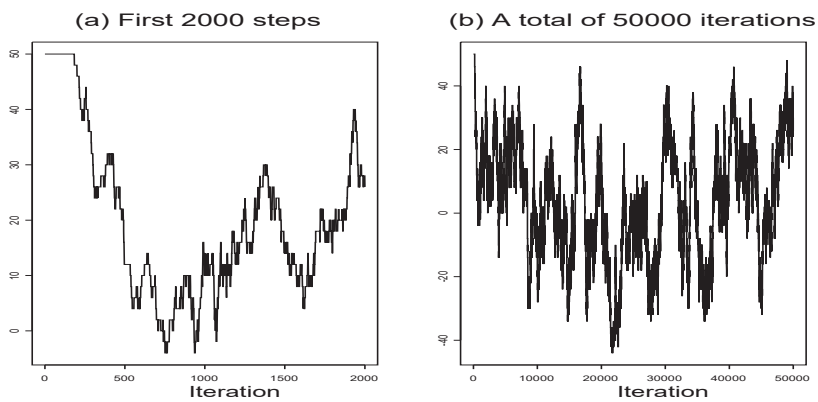

FIGURE 5.2. Simulation of the 1-D Ising model. The trace plots of (a) the first 2000 steps and (b) the total of 50,000 steps.

For this example, we can, in fact, compute the normalizing constant Z analytically and conduct an *exact* simulation (Section 2.4). Note that

$$\begin{aligned}\sum_{x_1} g(\mathbf{x}) &= (e^{\mu x_2} + e^{-\mu x_2}) \exp\left\{\mu \sum_{i=2}^{d-1} x_i x_{i+1}\right\} \\ &= \left(e^{-\mu} + e^{\mu}\right) \exp\left\{\mu \sum_{i=2}^{d-1} x_i x_{i+1}\right\}.\end{aligned}$$

We can recursively sum out x_2, x_3, etc., and obtain that

$$\sum_{x_1,\ldots,x_d} g(\mathbf{x}) = \sum_{x_d}\left[\cdots \sum_{x_2}\left\{\sum_{x_1} g(\mathbf{x})\right\}\cdots\right] = 2\left(e^{-\mu} + e^{\mu}\right)^{d-1},$$

for $d \geq 2$. Thus, the marginal distribution of x_d is $x_d = 1$ or -1 with equal probability (which is actually obvious without doing any computation). Conditional on x_d, the distribution of x_{d-1} is

$$\Pr(x_{d-1} = x_d) = e^{\mu}/(e^{\mu} + e^{-\mu}),$$

and $\Pr(x_{d-1} = -x_d) = e^{-\mu}/(e^{\mu} + e^{-\mu})$. Thus, we can recursively simulate $\mathbf{x}$ backward (Section 2.4).

We implemented both the exact simulation method and the Metropolis algorithm for the 1-D Ising model (5.1) with $\mu = 1$ and $\mu = 2$, respectively.

Figures 5.3(a) and 5.3(c) show the histograms of the total magnetization variable M from 20,000 exact samples, for model (5.1) with $\mu = 1$ and $\mu = 2$, respectively. Figures 5.3 (b) and 5.3(d) show the corresponding histograms from 20,000 Monte Carlo samples generated from 1,000,000 Metropolis sampling steps. The chosen samples were 1 in every 50 lags [i.e., $\mathbf{x}^{(50)}, \mathbf{x}^{(100)}, \ldots, \mathbf{x}^{(50k)}, \ldots$].

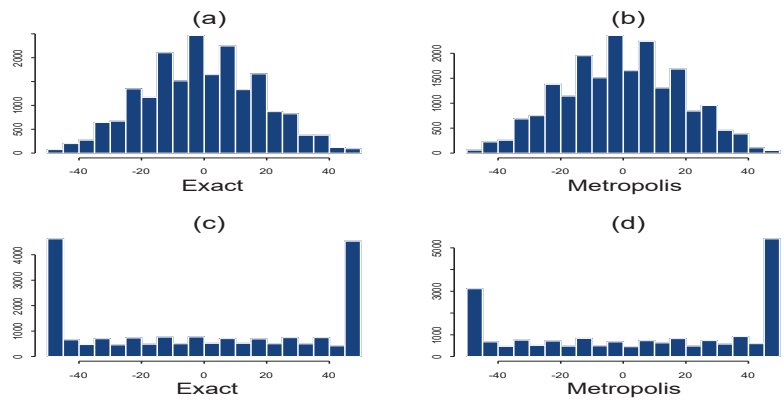

FIGURE 5.3. Histograms of the total magnetization M. (a) and (c): using 20,000 exact samples from the 1-D Ising models with $\mu = 1$ and $\mu = 2$; (b) and (d): using 20,000 samples chosen from 1 million Metropolis steps for each model.

From these simple graphs, we can make several interesting observations: (i) The computational effort for each Metropolis step is roughly 1/50th of that of exact simulation, so 1 million Metropolis steps took roughly the same CPU time as the production of 20,000 exact samples; (ii) when $\mu = 1$, the samples produced by the Metropolis algorithm were almost as good as those produced by independent sampling (at least to our eyes); (iii) when $\mu = 2$, the independent sampling showed an obvious advantage. These observations, in fact, reflect some important features of and deeper issues about the Metropolis algorithm: Each step of the Metropolis algorithm is usually very simple, but the Monte Carlo samples produced by a Metropolis sampler may become very "sticky"(e.g., getting stuck in local modes) in distributions with "low temperature" (i.e., high-energy barrier). In fact, the "stickiness" is already observable when $\mu = 1$ from Figure 5.2 (a): If one starts from an "all-up" configuration, it took about 1000 steps to get to the "ball park" of the interesting region and it took about 2000 steps to complete an "up-down" cycle. Quantitative measurement of this "stickiness" is often expressed as *autocorrelations*. A lag-k autocorrelation for a time series $M^{(1)}, M^{(2)}, \ldots$, is defined as

$$\rho_k = \mathrm{corr}(M^{(1)}, M^{(k+1)}),$$

under their stationary distribution. Higher autocorrelations imply that the produced samples are stickier. Figure 5.4 shows the autocorrelation plots (i.e., a plot for ρ_k versus k) for those Monte Carlo samples in Figures 5.3(b) and 5.3(d), respectively.

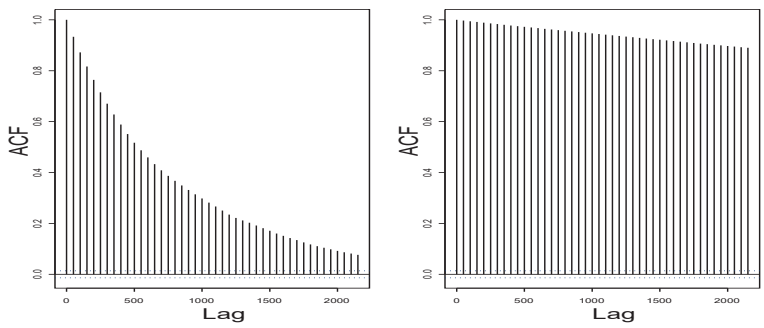

FIGURE 5.4. Autocorrelation plots of the time series $M^{(n)}$ produced by the Metropolis sampler for the 1-D Ising model with (a) $\mu = 1$ and (b) $\mu = 2$.

One observes that the 2000-lag autocorrelation of the $M^{(n)}$ series produced by the Metropolis sampler for $\mu = 2$ is still as high as about 0.9, implying that roughly one independent sample is as good as 20,000 Metropolis steps. More discussions on efficiency analysis of Markov chain Monte Carlo methods are discussed in Section 5.8 and Chapter 12.

5.2 Mathematical Formulation and Hastings's Generalization

The Metropolis algorithm prescribes a *transition rule* for a Markov chain. It uses a symmetric proposal function $T(\mathbf{x}, \mathbf{y})$ to suggest a possible move and then employs an acceptance-rejection rule to "thin it down." Hastings (1970) later extended the algorithm to the case when T is not necessarily symmetric. In Hastings' generalization, the only serious restriction on the proposal function is that $T(\mathbf{x}, \mathbf{y}) > 0$ if and only if $T(\mathbf{y}, \mathbf{x}) > 0$. With this transition function, one can implement the following iteration:

Metropolis-Hastings Algorithm. Given current state $\mathbf{x}^{(t)}$:

- Draw $\mathbf{y}$ from the proposal distribution $T(\mathbf{x}^{(t)}, \mathbf{y})$.
- Draw $U \sim$ Uniform[0,1] and update

$$\mathbf{x}^{(t+1)} = \begin{cases} \mathbf{y}, & \text{if } U \le r(\mathbf{x}^{(t)}, \mathbf{y}) \\ \mathbf{x}^{(t)} & \text{otherwise.} \end{cases} \tag{5.2}$$

where Metropolis et al. (1953) and Hastings (1970) suggested using

$$r(\mathbf{x},\mathbf{y}) = \min\left\{1,\ \frac{\pi(\mathbf{y})T(\mathbf{y},\mathbf{x})}{\pi(\mathbf{x})T(\mathbf{x},\mathbf{y})}\right\}.$$

Clearly, this algorithm is identical to the original Metropolis algorithm when $T(\mathbf{x},\mathbf{y}) = T(\mathbf{y},\mathbf{x})$.

Barker (1965) suggested another acceptance function:

$$r_B(\mathbf{x},\mathbf{y}) = \frac{\pi(\mathbf{y})T(\mathbf{y},\mathbf{x})}{\pi(\mathbf{y})T(\mathbf{y},\mathbf{x}) + \pi(\mathbf{x})T(\mathbf{x},\mathbf{y})}.$$

A more general formula for $r(\mathbf{x},\mathbf{y})$ is given by Charles Stein (personal communication):

$$r(\mathbf{x},\mathbf{y}) = \frac{\delta(\mathbf{x},\mathbf{y})}{\pi(\mathbf{x})T(\mathbf{x},\mathbf{y})}, \tag{5.3}$$

where $\delta(\mathbf{x},\mathbf{y})$ is any *symmetric function* in $\mathbf{x}$ and $\mathbf{y}$ that makes $r(\mathbf{x},\mathbf{y}) \leq 1$ for all $\mathbf{x},\mathbf{y}$. The intuition behind the ratio $T(\mathbf{y},\mathbf{x})/T(\mathbf{x},\mathbf{y})$ is that it compensates the "flow bias" of the proposal function.

If a rejection function of the form (5.3) is used, then for any $\mathbf{y} \neq \mathbf{x}$, the actual transition probability from $\mathbf{x}$ to $\mathbf{y}$ is

$$A(\mathbf{x},\mathbf{y}) = T(\mathbf{x},\mathbf{y})r(\mathbf{x},\mathbf{y}) = T(\mathbf{x},\mathbf{y})\frac{\delta(\mathbf{x},\mathbf{y})}{\pi(\mathbf{x})T(\mathbf{x},\mathbf{y})} = \pi(\mathbf{x})^{-1}\delta(\mathbf{x},\mathbf{y}). \tag{5.4}$$

Because $\delta(\mathbf{x},\mathbf{y}) = \delta(\mathbf{y},\mathbf{x})$, we have that $\pi(\mathbf{x})A(\mathbf{x},\mathbf{y}) = \pi(\mathbf{y})A(\mathbf{y},\mathbf{x})$. This implies that the Markov chain induced by the Metropolis-Hastings rule is *reversible* and has π as its invariant distribution (the next section).

For discrete state spaces, Peskun (1973) showed that the optimal choice of $r(\mathbf{x},\mathbf{y})$ in terms of statistical efficiency is the one in the original Metropolis algorithm (see Section 13.3.1 for more details). But the issue is less clear in terms of convergence rate of the induced Markov chain (Frigessi, Distefano, Hwang and Sheu 1993, Liu 1996c) As we will show in the next section, a main criterion used in the design of a Markov transition rule such as the Metropolis-Hastings algorithm is to ensure that the target distribution $\pi(\mathbf{x})$ is the invariant distribution of this chain.

5.3 Why Does the Metropolis Algorithm Work?

We first verify that the Metropolis-Hastings algorithm prescribes a transition rule with respect to which the target distribution $\pi(\mathbf{x})$ is invariant. Let $A(\mathbf{x},\mathbf{y})$ be the *actual* transition function of the algorithm. It differs

from the proposal function $T(\mathbf{x}, \mathbf{y})$ because an acceptance-rejection step is involved. We are required to show that

$$\int \pi(\mathbf{x}) A(\mathbf{x}, \mathbf{y}) d\mathbf{x} = \pi(\mathbf{y}). \tag{5.5}$$

Fortunately, there is an easier-to-check, but more restrictive, condition than (5.5), the *detailed balance*, which can be stated as

$$\pi(\mathbf{x}) A(\mathbf{x}, \mathbf{y}) = \pi(\mathbf{y}) A(\mathbf{y}, \mathbf{x}). \tag{5.6}$$

Clearly, if the detailed balance (5.6) holds, we have

$$\int \pi(\mathbf{x}) A(\mathbf{x}, \mathbf{y}) d\mathbf{x} = \int \pi(\mathbf{y}) A(\mathbf{y}, \mathbf{x}) d\mathbf{x} = \pi(\mathbf{y}) \int A(\mathbf{y}, \mathbf{x}) d\mathbf{x} = \pi(\mathbf{y}).$$

Thus, the detailed balance *ensures* invariance. The converse is not true. In Markov chain literature, chains that satisfy the detailed balance condition are called *reversible.*

We can write down $A(\mathbf{x}, \mathbf{y})$ explicitly for the Metropolis algorithm: For any $\mathbf{x} \neq \mathbf{y}$, the probability that we actually make the move from $\mathbf{x}$ to $\mathbf{y}$ is equal to the proposal probability, $T(\mathbf{x}, \mathbf{y})$, multiplied by the acceptance probability; that is,

$$A(\mathbf{x}, \mathbf{y}) = T(\mathbf{x}, \mathbf{y}) \min \left\{ 1, \frac{\pi(\mathbf{y}) T(\mathbf{y}, \mathbf{x})}{\pi(\mathbf{x}) T(\mathbf{x}, \mathbf{y})} \right\}, \tag{5.7}$$

for $\mathbf{x} \neq \mathbf{y}$. Hence,

$$\begin{aligned} \pi(\mathbf{x}) A(\mathbf{x}, \mathbf{y}) &= \pi(\mathbf{x}) T(\mathbf{x}, \mathbf{y}) \min \left\{ 1, \frac{\pi(\mathbf{y}) T(\mathbf{y}, \mathbf{x})}{\pi(\mathbf{x}) T(\mathbf{x}, \mathbf{y})} \right\} \\ &= \min\{\pi(\mathbf{x}) T(\mathbf{x}, \mathbf{y}), \pi(\mathbf{y}) T(\mathbf{y}, \mathbf{x})\}, \end{aligned}$$

which is a *symmetric function* in $\mathbf{x}$ and $\mathbf{y}$. Thus, the detailed balance condition is satisfied.

More generally, as long as the transition function $A(\mathbf{x}, \mathbf{y})$ is of the form

$$A(\mathbf{x}, \mathbf{y}) = \pi(\mathbf{y}) \delta(\mathbf{x}, \mathbf{y}),$$

where $\delta(\mathbf{x}, \mathbf{y})$ is a symmetric function in $\mathbf{x}, \mathbf{y}$, one can easily verify that the detailed balance condition has to be satisfied. The difficulty in the construction of A, however, is that the symmetric function $\delta(\mathbf{x}, \mathbf{y})$ has to be chosen properly so that the integral $\int A(\mathbf{x}, \mathbf{y}) d\mathbf{y} = 1$. One can easily check that the Metropolis transition can be written as (for $\mathbf{x} \neq \mathbf{y}$)

$$A(\mathbf{x}, \mathbf{y}) = \pi(\mathbf{y}) \min \left\{ \frac{T(\mathbf{x}, \mathbf{y})}{\pi(\mathbf{y})}, \frac{T(\mathbf{y}, \mathbf{x})}{\pi(\mathbf{x})} \right\}.$$

The acceptance rule proposed by Barker (1965) corresponds to a transition (for $\mathbf{x} \neq \mathbf{y}$)

$$A(\mathbf{x}, \mathbf{y}) = \pi(\mathbf{y}) \frac{T(\mathbf{x}, \mathbf{y}) T(\mathbf{y}, \mathbf{x})}{\pi(\mathbf{y}) T(\mathbf{y}, \mathbf{x}) + \pi(\mathbf{x}) T(\mathbf{x}, \mathbf{y})}.$$

It was not clear, then, which acceptance rule is better. Peskun (1973) later showed that the Metropolis rule generally works better in terms of statistical efficiency.

By the standard Markov chain theory, if the chain is irreducible[4], aperiodic[5] [this is almost surely true for the Metropolis algorithm (Tierney 1994)], and possesses an invariant distribution, then the chain will become stationary at its invariant distribution, π. Therefore, if we run this chain long enough (say, after a burn-in period of n_0 steps), the samples $\mathbf{x}_{n_0+1}, \mathbf{x}_{n_0+2}, \ldots$ produced by the chain can be regarded as approximately following the target distribution π. One then realizes the task of drawing random (but correlated) samples from a given distribution.

5.4 Some Special Algorithms

To illustrate how the Metropolis-Hastings rule is practiced, we describe a few special algorithms that have appeared frequently in the literature.

5.4.1 Random-walk Metropolis

Suppose the target distribution $\pi(\mathbf{x})$ is defined on the d-dimensional Euclidean space $\mathbb{R}^d$. A natural "perturbation" of the current configuration $\mathbf{x}^{(t)}$ is the addition of a random "error;" that is, the next candidate position is proposed as $\mathbf{x}' = \mathbf{x}^{(t)} + \boldsymbol{\epsilon}_t$, where $\boldsymbol{\epsilon}_t \sim g_\sigma(\cdot)$ is independent and identically distributed for different t. Here, σ represents the "range" of the proposal exploration and is controlled by the user. In problems where we do not have much information on the shape of the target distribution, we often end up letting $g_\sigma(\cdot)$ be a spherically symmetric distribution. Typical choices include the spherical Gaussian distribution $N(0, \sigma^2 I)$ or the uniform distribution in a ball of radius σ. Clearly, if one does not exercise the Metropolis rejection rule to this proposal, the resulting walk will drift away to infinity and never come back (when $d \geq 3$). It is thus worthwhile to point out that in a Metropolis algorithm, the proposal chain is not required to have any good "global properties" other than being irreducible

[4] A Markov chain is said to be *irreducible* if the chain has nonzero probability (density) to move from one position in the state space to any other position in a finite number of steps.

[5] A Markov chain is said *aperiodic* if the maximum common divider of the number of steps it takes for the chain to come back to the starting point (any) is equal to one.

(see footnote 4 on page 114). But we do require some local properties for $T(\mathbf{x}, \mathbf{y})$ [e.g., $T(\mathbf{x}, \mathbf{y}) > 0$ whenever $T(\mathbf{y}, \mathbf{x}) > 0$].

Given the current state $\mathbf{x}^{(t)}$, the random-walk Metropolis algorithm iterates the following steps:

- Draw $\boldsymbol{\epsilon} \sim g_\sigma$ and set $\mathbf{x}' = \mathbf{x}^{(t)} + \boldsymbol{\epsilon}$, where g_σ is a spherically symmetric distribution and σ can be controlled by the user.
- Simulate $u \sim$ Uniform[0,1] and update

$$\mathbf{x}^{(t+1)} = \begin{cases} \mathbf{y} & \text{if } \ u \leq \dfrac{\pi(\mathbf{x}')}{\pi(\mathbf{x}^{(t)})} \\ \mathbf{x}^{(t)} & \text{otherwise.} \end{cases}$$

In the example of simulating six hard-shell balls in a box, we used the random-walk method for each individual ball. In an interesting study, Gelman, Roberts and Gilks (1995) suggested that a rule of thumb in choosing σ in a random-walk Metropolis is to maintain a 25% to 35% acceptance rate. This rule is supported by a theoretical analysis for a Gaussian target density (more details in Section 5.8).

5.4.2 Metropolized independence sampler

A very special choice of the proposal transition function $T(\mathbf{x}, \mathbf{y})$ is an *independent* trial density $g(\mathbf{y})$; that is, the proposed move $\mathbf{y}$ is generated from $g(\cdot)$ independent of the previous state $\mathbf{x}^{(t)}$. This method, as first suggested in Hastings (1970), appears to be an alternative to the rejection sampling and importance sampling. Its convergence properties was studied in Liu (1996a), where all the eigenvalues and eigenfunctions of the actual transition function are derived (see Section 13.4).

The MIS Scheme: given the current state $\mathbf{x}^{(t)}$,

- Draw $\mathbf{y} \sim g(\mathbf{y})$.
- Simulate $u \sim$ Uniform[0,1] and let

$$\mathbf{x}^{(t+1)} = \begin{cases} \mathbf{y} & \text{if } \ u \leq \min\left\{1, \dfrac{w(\mathbf{y})}{w(\mathbf{x}^{(t)})}\right\} \\ x^{(t)} & \text{otherwise,} \end{cases}$$

where $w(\mathbf{x}) = \pi(\mathbf{x})/g(\mathbf{x})$ is the usual *importance sampling weight.*

As with the rejection method, the efficiency of MIS depends on how close the trial density $g(\mathbf{y})$ is to the target $\pi(\mathbf{y})$. To ensure robust performance, it is advisable to let $g(\cdot)$ be a relatively long-tailed distribution. Gelman and Rubin (1992) and Tierney (1994) suggested that one can insert a couple of

MIS steps into Gibbs iteration when correctly sampling from a conditional distribution is difficult. The idea is useful in many Bayesian computations in which each conditional density can be approximated reasonably well by a Gaussian distribution. To accommodate irregular tail behaviors, it is essential to use a long-tailed t-distribution as $g(\mathbf{x})$.

5.4.3 Configurational bias Monte Carlo

The configurational bias Monte Carlo (CBMC) algorithm can be viewed as an SIS-based Metropolized independence sampler. Suppose the argument of the target distribution, $\mathbf{x}$, can be decomposed as $\mathbf{x} = (x_1, \ldots, x_d)$. As in a sequential importance sampler (Section 2.6.3 and Chapters 3 and 4), we assume that there is a sequence of *auxiliary distributions*

$$\pi_1(x_1), \pi_2(x_1, x_2), \ldots, \pi_{d-1}(\mathbf{x}_{d-1}), \pi(\mathbf{x})$$

that can help us construct the trial sampling distribution.

Let the trial sampling distribution of $\mathbf{x}$ be

$$g(\mathbf{x}) = g_1(x_1) g_2(x_2 \mid x_1) \cdots g_d(x_d \mid \mathbf{x}_{d-1}).$$

To implement a CBMC algorithm (Siepmann and Frenkel 1992), we first draw $\mathbf{x}^{(0)}$ from $g(\mathbf{x})$ via a SIS strategy and compute its importance weight $w^{(0)}$ (up to a normalizing constant). Suppose that currently we have $\mathbf{x}^{(t)}$ with weight $w(\mathbf{x}^{(t)})$; then, at the next iteration, we do the following:

- Independently generate a trial configuration $\mathbf{y}$ from $g(\ )$ (using the sequential approach); compute its importance weight

  $$w(\mathbf{y}) = \pi(\mathbf{y})/g(\mathbf{y}),$$

  which can often be derived recursively as

  $$w(\mathbf{y}) = \frac{\pi_1(y_1)}{g_1(y_1)} \frac{\pi_2(y_1, y_2)}{g_2(y_2 \mid y_1)\pi_1(y_1)} \cdots \frac{\pi_d(y_1, \ldots, y_d)}{g_d(y_d \mid \mathbf{y}_{d-1})\pi_{d-1}(\mathbf{y}_{d-1})}.$$

  (This recursion is key to the SIS approach.)

- Accept $\mathbf{y}$; that is, let $\mathbf{x}^{(t+1)} = \mathbf{y}$, with probability $\min\left\{1, \frac{w(\mathbf{y})}{w(\mathbf{x}^{(t)})}\right\}$; and let $\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)}$ otherwise.

This procedure is exactly the same as the *Metropolized independence sampler* described in the previous subsection, except that the trial density is built up sequentially and the importance weight computed recursively.

A useful modification of the foregoing CBMC procedure is to incorporate a stage-wise rejection decision. Suppose all the previous $k-1$ steps in the

sequential simulation of the trial configuration $\mathbf{x}'$ have been accepted. Then, at the kth stage of SIS, we accept $\mathbf{x}'_k = (x'_1, \ldots, x'_k)$ with probability

$$p_k = \min\left\{1,\ \frac{\pi_k(\mathbf{x}'_k)\pi_{k-1}(\mathbf{x}_{k-1})g_k(x_k \mid \mathbf{x}_{k-1})}{\pi_k(\mathbf{x}_k)\pi_{k-1}(\mathbf{x}'_{k-1})g_k(x'_k \mid \mathbf{x}'_{k-1})}\right\} = \min\left\{1,\ \frac{u_k(\mathbf{x}'_k)}{u_k(\mathbf{x}_k)}\right\},$$

where u_k is the same as that in (3.10). In other words, the acceptance probability is equal to the ratio of the incremental importance weights between the trial and the current configurations. When rejected, we go back to the first stage to rebuild the whole configuration. It should be noted that one does not need to perform the acceptance-rejection decision at every stage and she/he has a complete control on when to conduct conduct the acceptance-rejection step.

This multistage method has been shown effective in simulating superfluid Helium 4 and other quantum mechanical systems (Ceperley 1995). Compared to the CBMC, this multistage sampler can force an early stop so as to save computing power. However, the chance of the final acceptance of a complete configuration in the multistage approach should be smaller than that in the CBMC because

$$\min(1, a_1) \times \cdots \times \min(1, a_d) \leq \min(1,\ a_1 \times \cdots \times a_d).$$

To show that the multistage modification of CBMC is proper, we can write down its actual transition function and prove that it satisfies the detailed balance. Here, we provide only a proof for the case when $d = 2$ [i.e. $\mathbf{x} = (x_1, x_2)$]. The general proof is left to the reader. Suppose the auxiliary distributions when $d = 2$ are $\pi_1(x_1)$ and $\pi_2(\mathbf{x}) \equiv \pi(\mathbf{x})$. Suppose the current state is $\mathbf{x}$. Then, the probability of accepting a new configuration $\mathbf{x}' = (x'_1 x'_2) \neq \mathbf{x}$ is

$$\begin{aligned} P(\mathbf{x} \to \mathbf{x}') &= g_1(x'_1)g_2(x'_2|x'_1)\min\left\{1, \frac{\pi_1(x'_1)g_1(x_1)}{\pi_1(x_1)g_1(x'_1)}\right\} \\ &\quad \times \min\left\{1, \frac{\pi(x'_1, x'_2)\pi_1(x_1)g_2(x_2 \mid x_1)}{\pi(x_1, x_2)\pi_1(x'_1)g_1(x'_2 \mid x'_1)}\right\} \\ &= \pi(\mathbf{x}')\min\{g_1(x'_1)\pi_1(x_1), \pi_1(x'_1)g_1(x_1)\} \\ &\quad \times \min\left\{\frac{\pi_1(x'_1)g_1(x'_2|x'_1)}{\pi(\mathbf{x}')}, \frac{\pi_1(x_1)g_1(x_2|x_1)}{\pi(\mathbf{x})}\right\}. \end{aligned}$$

Hence, this transition function is indeed of the form $\pi(\mathbf{x}')\delta(\mathbf{x}, \mathbf{x}')$, where δ is a symmetric function. The detailed balance condition is thus satisfied (see the argument in Section 5.3).

5.5 Multipoint Metropolis Methods

In principle, the Metropolis sampling method discussed in Section 5.2 can be applied to almost any target distribution. In practice, however, it is