23 *Monte Carlo inference*

23.1 Introduction

So far, we discussed various deterministic algorithms for posterior inference. These methods enjoy many of the benefits of the Bayesian approach, while still being about as fast as optimization-based point-estimation methods. The trouble with these methods is that they can be rather complicated to derive, and they are somewhat limited in their domain of applicability (e.g., they usually assume conjugate priors and exponential family likelihoods, although see (Wand et al. 2011) for some recent extensions of mean field to more complex distributions). Furthermore, although they are fast, their accuracy is often limited by the form of the approximation which we choose.

In this chapter, we discuss an alternative class of algorithms based on the idea of Monte Carlo approximation, which we first introduced in Section 2.7. The idea is very simple: generate some (unweighted) samples from the posterior, $\mathbf{x}^s \sim p(\mathbf{x}|\mathcal{D})$, and then use these to compute any quantity of interest, such as a posterior marginal, $p(x_1|\mathcal{D})$, or the posterior of the difference of two quantities, $p(x_1 - x_2|\mathcal{D})$, or the posterior predictive, $p(y|\mathcal{D})$, etc. All of these quantities can be approximated by $\mathbb{E}[f|\mathcal{D}] \approx \frac{1}{S}\sum_{s=1}^{S} f(\mathbf{x}^s)$ for some suitable function f.

By generating enough samples, we can achieve any desired level of accuracy we like. The main issue is: how do we efficiently generate samples from a probability distribution, particularly in high dimensions? In this chapter, we discuss non-iterative methods for generating independent samples. In the next chapter, we discuss an iterative method known as Markov Chain Monte Carlo, or MCMC for short, which produces dependent samples but which works well in high dimensions. Note that sampling is a large topic. The reader should consult other books, such as (Liu 2001; Robert and Casella 2004), for more information.

23.2 Sampling from standard distributions

We briefly discuss some ways to sample from 1 or 2 dimensional distributions of standard form. These methods are often used as subroutines by more complex methods.

23.2.1 Using the cdf

The simplest method for sampling from a univariate distribution is based on the **inverse probability transform**. Let F be a cdf of some distribution we want to sample from, and let F^{-1}

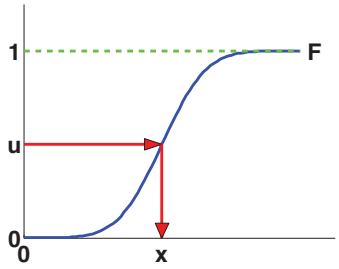


Figure 23.1 Sampling using an inverse CDF. Figure generated by `sampleCdf`.

be its inverse. Then we have the following result.

Theorem 23.2.1. *If $U \sim U(0,1)$ is a uniform rv, then $F^{-1}(U) \sim F$.*

Proof.

$$\Pr(F^{-1}(U) \leq x) = \Pr(U \leq F(x)) \quad \text{(applying } F \text{ to both sides)} \tag{23.1}$$

$$= F(x) \quad \text{(because } \Pr(U \leq y) = y \tag{23.2}$$

where the first line follows since F is a monotonic function, and the second line follows since U is uniform on the unit interval. □

Hence we can sample from any univariate distribution, for which we can evaluate its inverse cdf, as follows: generate a random number $u \sim U(0,1)$ using a **pseudo random number generator** (see e.g., (Press et al. 1988) for details). Let u represent the height up the y axis. Then "slide along" the x axis until you intersect the F curve, and then "drop down" and return the corresponding x value. This corresponds to computing $x = F^{-1}(u)$. See Figure 23.1 for an illustration.

For example, consider the exponential distribution

$$\text{Expon}(x|\lambda) \triangleq \lambda e^{-\lambda x} \, \mathbb{I}(x \geq 0) \tag{23.3}$$

The cdf is

$$F(x) = 1 - e^{-\lambda x} \, \mathbb{I}(x \geq 0) \tag{23.4}$$

whose inverse is the quantile function

$$F^{-1}(p) = -\frac{\ln(1-p)}{\lambda} \tag{23.5}$$

By the above theorem, if $U \sim \text{Unif}(0,1)$, we know that $F^{-1}(U) \sim \text{Expon}(\lambda)$. Furthermore, since $1 - U \sim \text{Unif}(0,1)$ as well, we can sample from the exponential distribution by first sampling from the uniform and then transforming the results using $-\ln(u)/\lambda$.

23.2.2 Sampling from a Gaussian (Box-Muller method)

We now describe a method to sample from a Gaussian. The idea is we sample uniformly from a unit radius circle, and then use the change of variables formula to derive samples from a spherical 2d Gaussian. This can be thought of as two samples from a 1d Gaussian.

In more detail, sample $z_1, z_2 \in (-1, 1)$ uniformly, and then discard pairs that do not satisfy $z_1^2 + z_2^2 \leq 1$. The result will be points uniformly distributed inside the unit circle, so $p(\mathbf{z}) = \frac{1}{\pi}\mathbb{I}(z \text{ inside circle})$. Now define

$$x_i = z_i \left(\frac{-2 \ln r^2}{r^2} \right)^{\frac{1}{2}} \tag{23.6}$$

for $i = 1 : 2$, where $r^2 = z_1^2 + z_2^2$. Using the multivariate change of variables formula, we have

$$p(x_1, x_2) = p(z_1, z_2) \left| \frac{\partial(z_1, z_2)}{\partial(x_1, x_2)} \right| = \left[\frac{1}{\sqrt{2\pi}} \exp(-\frac{1}{2} x_1^2) \right] \left[\frac{1}{\sqrt{2\pi}} \exp(-\frac{1}{2} x_2^2) \right] \tag{23.7}$$

Hence x_1 and x_2 are two independent samples from a univariate Gaussian. This is known as the **Box-Muller** method.

To sample from a multivariate Gaussian, we first compute the Cholesky decomposition of its covariance matrix, $\mathbf{\Sigma} = \mathbf{L}\mathbf{L}^T$, where $\mathbf{L}$ is lower triangular. Next we sample $\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ using the Box-Muller method. Finally we set $\mathbf{y} = \mathbf{L}\mathbf{x} + \boldsymbol{\mu}$. This is valid since

$$\text{cov}\left[\mathbf{y}\right] = \mathbf{L}\text{cov}\left[\mathbf{x}\right]\mathbf{L}^T = \mathbf{L}\,\mathbf{I}\,\mathbf{L}^T = \mathbf{\Sigma} \tag{23.8}$$

23.3 Rejection sampling

When the inverse cdf method cannot be used, one simple alternative is to use **rejection sampling**, which we now explain.

23.3.1 Basic idea

In rejection sampling, we create a **proposal distribution** $q(x)$ which satisifes $Mq(x) \geq \tilde{p}(x)$, for some constant M, where $\tilde{p}(x)$ is an unnormalized version of $p(x)$ (i.e., $p(x) = \tilde{p}(x)/Z_p$ for some possibly unknown constant Z_p). The function $Mq(x)$ provides an upper envelope for $\tilde{p}$. We then sample $x \sim q(x)$, which corresponds to picking a random x location, and then we sample $u \sim U(0, 1)$, which corresponds to picking a random height (y location) under the envelope. If $u > \frac{\tilde{p}(x)}{Mq(x)}$, we reject the sample, otherwise we accept it. See Figure 23.2(a). where the acceptance region is shown shaded, and the rejection region is the white region between the shaded zone and the upper envelope.

We now prove that this procedure is correct. Let

$$S = \{(x, u) : u \leq \tilde{p}(x)/Mq(x)\},\ S_0 = \{(x, u) : x \leq x_0, u \leq \tilde{p}(x)/Mq(x)\} \tag{23.9}$$

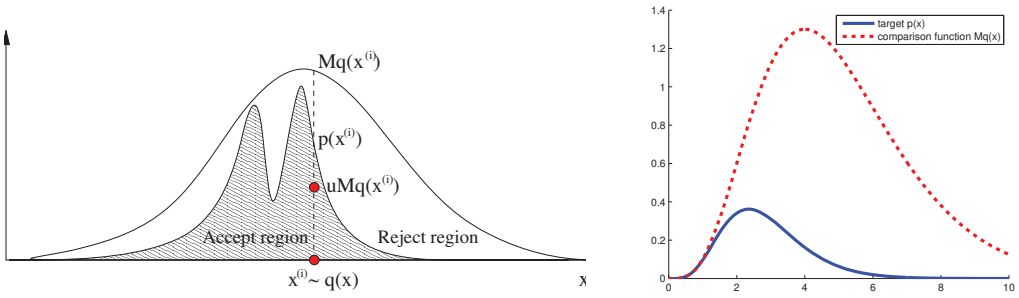


Figure 23.2 (a) Schematic illustration of rejection sampling. Source: Figure 2 of (Andrieu et al. 2003). Used with kind permission of Nando de Freitas. (b) Rejection sampling from a $\text{Ga}(\alpha = 5.7, \lambda = 2)$ distribution (solid blue) using a proposal of the form $M\text{Ga}(k, \lambda - 1)$ (dotted red), where $k = \lfloor 5.7 \rfloor = 5$. The curves touch at $\alpha - k = 0.7$. Figure generated by `rejectionSamplingDemo`.

Then the cdf of the accepted points is given by

$$P(x \leq x_0 | x \text{ accepted}) = \frac{P(x \leq x_0, x \text{ accepted})}{P(x \text{ accepted})} \tag{23.10}$$

$$= \frac{\int\int \mathbb{I}((x,u) \in S_0) q(x) du dx}{\int\int \mathbb{I}((x,u) \in S) q(x) du dx} = \frac{\int_{-\infty}^{x_0} \tilde{p}(x) dx}{\int_{-\infty}^{\infty} \tilde{p}(x) dx} \tag{23.11}$$

which is the cdf of $p(x)$, as desired.

How efficient is this method? Since we generate with probability $q(x)$ and accept with probability $\frac{\tilde{p}(x)}{Mq(x)}$, the probability of acceptance is

$$p(\text{accept}) = \int \frac{\tilde{p}(x)}{Mq(x)} q(x) dx = \frac{1}{M} \int \tilde{p}(x) dx \tag{23.12}$$

Hence we want to choose M as small as possible while still satisfying $Mq(x) \geq \tilde{p}(x)$.

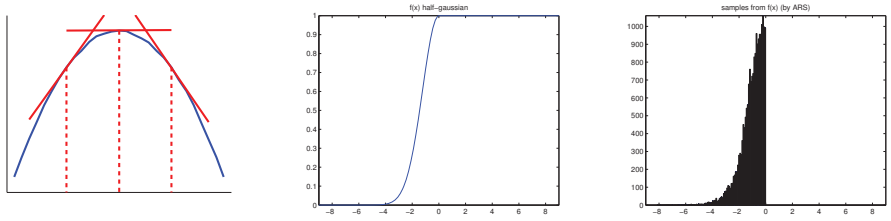
23.3.2 Example

For example, suppose we want to sample from a Gamma distribution:[1]

$$\text{Ga}(x|\alpha, \lambda) = \frac{1}{\Gamma(\alpha)} x^{\alpha-1} \lambda^{\alpha} \exp(-\lambda x) \tag{23.13}$$

One can show that if $X_i \overset{iid}{\sim} \text{Expon}(\lambda)$, and $Y = X_1 + \cdots + X_k$, then $Y \sim \text{Ga}(k, \lambda)$. For non-integer shape parameters, we cannot use this trick. However, we can use rejection sampling

1. This section is based on notes by Ioana A. Cosma, available at `http://users.aims.ac.za/~ioana/cp2.pdf`.



(a) (b) (c)

Figure 23.3 (a) Idea behind adaptive rejection sampling. We place piecewise linear upper (and lower) bounds on the log-concave density. Based on Figure 1 of (Gilks and Wild 1992). Figure generated by `arsEnvelope`. (b-c) Using ARS to sample from a half-Gaussian. Figure generated by `arsDemo`, written by Daniel Eaton.

using a $\text{Ga}(k, \lambda - 1)$ distribution as a proposal, where $k = \lfloor \alpha \rfloor$. The ratio has the form

$$\frac{p(x)}{q(x)} = \frac{\text{Ga}(x|\alpha, \lambda)}{\text{Ga}(x|k, \lambda - 1)} = \frac{x^{\alpha-1}\lambda^\alpha \exp(-\lambda x)/\Gamma(\alpha)}{x^{k-1}(\lambda-1)^k \exp(-(\lambda-1)x)/\Gamma(k)} \tag{23.14}$$

$$= \frac{\Gamma(k)\lambda^\alpha}{\Gamma(\alpha)(\lambda-1)^k} x^{\alpha-k} \exp(-x) \tag{23.15}$$

This ratio attains its maximum when $x = \alpha - k$. Hence

$$M = \frac{\text{Ga}(\alpha - k|\alpha, \lambda)}{\text{Ga}(\alpha - k|k, \lambda - 1)} \tag{23.16}$$

See Figure 23.2(b) for a plot. (Exercise 23.2 asks you to devise a better proposal distribution based on the Cauchy distribution.)

23.3.3 Application to Bayesian statistics

Suppose we want to draw (unweighted) samples from the posterior, $p(\boldsymbol{\theta}|\mathcal{D}) = p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})/p(\mathcal{D})$. We can use rejection sampling with $\tilde{p}(\boldsymbol{\theta}) = p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})$ as the target distribution, $q(\boldsymbol{\theta}) = p(\boldsymbol{\theta})$ as our proposal, and $M = p(\mathcal{D}|\hat{\boldsymbol{\theta}})$, where $\hat{\boldsymbol{\theta}} = \arg\max p(\mathcal{D}|\boldsymbol{\theta})$ is the MLE; this was first suggested in (Smith and Gelfand 1992). We accept points with probability

$$\frac{\tilde{p}(\boldsymbol{\theta})}{Mq(\boldsymbol{\theta})} = \frac{p(\mathcal{D}|\boldsymbol{\theta})}{p(\mathcal{D}|\hat{\boldsymbol{\theta}})} \tag{23.17}$$

Thus samples from the prior that have high likelihood are more likely to be retained in the posterior. Of course, if there is a big mismatch between prior and posterior (which will be the case if the prior is vague and the likelihood is informative), this procedure is very inefficient. We discuss better algorithms later.

23.3.4 Adaptive rejection sampling

We now describe a method that can automatically come up with a tight upper envelope $q(x)$ to any log concave density $p(x)$. The idea is to upper bound the log density with a piecewise

linear function, as illustrated in Figure 23.3(a). We choose the initial locations for the pieces based on a fixed grid over the support of the distribution. We then evaluate the gradient of the log density at these locations, and make the lines be tangent at these points.

Since the log of the envelope is piecewise linear, the envelope itself is piecewise exponential:

$$q(x) = M_i \lambda_i \exp(-\lambda_i(x - x_{i-1})), \quad x_{i-1} < x \leq x_i \tag{23.18}$$

where x_i are the grid points. It is relatively straightforward to sample from this distribution. If the sample x is rejected, we create a new grid point at x, and thereby refine the envelope. As the number of grid points is increased, the tightness of the envelope improves, and the rejection rate goes down. This is known as **adaptive rejection sampling** (ARS) (Gilks and Wild 1992). Figure 23.3(b-c) gives an example of the method in action. As with standard rejection sampling, it can be applied to unnormalized distributions.

23.3.5 Rejection sampling in high dimensions

It is clear that we want to make our proposal $q(x)$ as close as possible to the target distribution $p(x)$, while still being an upper bound. But this is quite hard to achieve, especially in high dimensions. To see this, consider sampling from $p(\mathbf{x}) = \mathcal{N}(\mathbf{0}, \sigma_p^2 \mathbf{I})$ using as a proposal $q(\mathbf{x}) = \mathcal{N}(\mathbf{0}, \sigma_q^2 \mathbf{I})$. Obviously we must have $\sigma_q^2 \geq \sigma_p^2$ in order to be an upper bound. In D dimensions, the optimum value is given by $M = (\sigma_q/\sigma_p)^D$. The acceptance rate is $1/M$ (since both p and q are normalized), which decreases exponentially fast with dimension. For example, if σ_q exceeds σ_p by just 1%, then in 1000 dimensions the acceptance ratio will be about 1/20,000. This is a fundamental weakness of rejection sampling.

In Chapter 24, we will describe MCMC sampling, which is a more efficient way to sample from high dimensional distributions. Sometimes this uses (adaptive) rejection sampling as a subroutine, which is known as **adaptive rejection Metropolis sampling** (Gilks et al. 1995).

23.4 Importance sampling

We now describe a Monte Carlo method known as **importance sampling** for approximating integrals of the form

$$I = \mathbb{E}[f] = \int f(\mathbf{x}) p(\mathbf{x}) d\mathbf{x} \tag{23.19}$$

23.4.1 Basic idea

The idea is to draw samples $\mathbf{x}$ in regions which have high probability, $p(\mathbf{x})$, but also where $|f(\mathbf{x})|$ is large. The result can be **super efficient**, meaning it needs less samples than if we were to sample from the exact distribution $p(\mathbf{x})$. The reason is that the samples are focussed on the important parts of space. For example, suppose we want to estimate the probability of a **rare event**. Define $f(\mathbf{x}) = \mathbb{I}(\mathbf{x} \in E)$, for some set E. Then it is better to sample from a proposal of the form $q(\mathbf{x}) \propto f(\mathbf{x})p(\mathbf{x})$ than to sample from $p(\mathbf{x})$ itself.

Importance sampling samples from any proposal, $q(\mathbf{x})$. It then uses these samples to estimate

the integral as follows:

$$\mathbb{E}[f] = \int f(\mathbf{x}) \frac{p(\mathbf{x})}{q(\mathbf{x})} q(\mathbf{x}) d\mathbf{x} \approx \frac{1}{S} \sum_{s=1}^{S} w_s f(\mathbf{x}^s) = \hat{I} \tag{23.20}$$

where $w_s \triangleq \frac{p(\mathbf{x}^s)}{q(\mathbf{x}^s)}$ are the **importance weights**. Note that, unlike rejection sampling, we use all the samples.

How should we choose the proposal? A natural criterion is to minimize the variance of the estimate $\hat{I} = \sum_s w_s f(\mathbf{x}^s)$. Now

$$\mathrm{var}_{q(\mathbf{x})}[f(\mathbf{x})w(\mathbf{x})] = \mathbb{E}_{q(\mathbf{x})}\left[f^2(\mathbf{x})w^2(\mathbf{x})\right] - I^2 \tag{23.21}$$

Since the last term is independent of q, we can ignore it. By Jensen's inequality, we have the following lower bound:

$$\mathbb{E}_{q(\mathbf{x})}\left[f^2(\mathbf{x})w^2(\mathbf{x})\right] \geq (\mathbb{E}_{q(\mathbf{x})}[|f(\mathbf{x})w(\mathbf{x})|])^2 = \left(\int |f(\mathbf{x})|p(\mathbf{x})d\mathbf{x}\right)^2 \tag{23.22}$$

The lower bound is obtained when we use the optimal importance distribution:

$$q^*(\mathbf{x}) = \frac{|f(\mathbf{x})|p(\mathbf{x})}{\int |f(\mathbf{x}')|p(\mathbf{x}')d\mathbf{x}'} \tag{23.23}$$

When we don't have a particular target function $f(\mathbf{x})$ in mind, we often just try to make $q(\mathbf{x})$ as close as possible to $p(\mathbf{x})$. In general, this is difficult, especially in high dimensions, but it is possible to adapt the proposal distribution to improve the approximation. This is known as **adaptive importance sampling** (Oh and Berger 1992).

23.4.2 Handling unnormalized distributions

It is frequently the case that we can evaluate the unnormalized target distribution, $\tilde{p}(\mathbf{x})$, but not its normalization constant, Z_p. We may also want to use an unnormalized proposal, $\tilde{q}(\mathbf{x})$, with possibly unknown normlization constant Z_q. We can do this as follows. First we evaluate

$$\mathbb{E}[f] = \frac{Z_q}{Z_p} \int f(\mathbf{x}) \frac{\tilde{p}(\mathbf{x})}{\tilde{q}(\mathbf{x})} q(\mathbf{x}) d\mathbf{x} \approx \frac{Z_q}{Z_p} \frac{1}{S} \sum_{s=1}^{S} \tilde{w}_s f(\mathbf{x}^s) \tag{23.24}$$

where $\tilde{w}_s \triangleq \frac{\tilde{p}(\mathbf{x}^s)}{\tilde{q}(\mathbf{x}^s)}$ is the unnormalized importance weight. We can use the same set of samples to evaluate the ratio Z_p/Z_q as follows:

$$\frac{Z_p}{Z_q} = \frac{1}{Z_q} \int \tilde{p}(\mathbf{x}) d\mathbf{x} = \int \frac{\tilde{p}(\mathbf{x})}{\tilde{q}(\mathbf{x})} q(\mathbf{x}) d\mathbf{x} \approx \frac{1}{S} \sum_{s=1}^{S} \tilde{w}_s \tag{23.25}$$

Hence

$$\hat{I} = \frac{\frac{1}{S}\sum_s \tilde{w}_s f(\mathbf{x}^s)}{\frac{1}{S}\sum_s \tilde{w}_s} = \sum_{s=1}^{S} w_s f(\mathbf{x}^s) \tag{23.26}$$

where

$$w_s \triangleq \frac{\tilde{w}_s}{\sum_{s'} \tilde{w}_{s'}} \tag{23.27}$$

are the normalized importance weights. The resulting estimate is a ratio of two estimates, and hence is biased. However, as $S \to \infty$, we have that $\hat{I} \to I$, under weak assumptions (see e.g., (Robert and Casella 2004) for details).

23.4.3 Importance sampling for a DGM: likelihood weighting

We now describe a way to use importance sampling to generate samples from a distribution which can be represented as a directed graphical model (Chapter 10).

If we have no evidence, we can sample from the unconditional joint distribution of a DGM $p(\mathbf{x})$ as follows: first sample the root nodes, then sample their children, then sample their children, etc. This is known as **ancestral sampling**. It works because, in a DAG, we can always topologically order the nodes so that parents preceed children. (Note that there is no equivalent easy method for sampling from an unconditional *undirected* graphical model.)

Now suppose we have some evidence, so some nodes are "clamped" to observed values, and we want to sample from the posterior $p(\mathbf{x}|\mathcal{D})$. If all the variables are discrete, we can use the following simple procedure: perform ancestral sampling, but as soon as we sample a value that is inconsistent with an observed value, reject the whole sample and start again. This is known as **logic sampling** (Henrion 1988).

Needless to say, logic sampling is very inefficient, and it cannot be applied when we have real-valued evidence. However, it can be modified as follows. Sample unobserved variables as before, conditional on their parents. But don't sample observed variables; instead we just use their observed values. This is equivalent to using a proposal of the form

$$q(\mathbf{x}) = \prod_{t \notin E} p(x_t|\mathbf{x}_{\mathrm{pa}(t)}) \prod_{t \in E} \delta_{x_t^*}(x_t) \tag{23.28}$$

where E is the set of observed nodes, and x_t^* is the observed value for node t. We should therefore give the overall sample an importance weight as follows:

$$w(\mathbf{x}) = \frac{p(\mathbf{x})}{q(\mathbf{x})} = \prod_{t \notin E} \frac{p(x_t|\mathbf{x}_{\mathrm{pa}(t)})}{p(x_t|\mathbf{x}_{\mathrm{pa}(t)})} \prod_{t \in E} \frac{p(x_t|\mathbf{x}_{\mathrm{pa}(t)})}{1} = \prod_{t \in E} p(x_t|\mathbf{x}_{\mathrm{pa}(t)}) \tag{23.29}$$

This technique is known as **likelihood weighting** (Fung and Chang 1989; Shachter and Peot 1989).

23.4.4 Sampling importance resampling (SIR)

We can draw unweighted samples from $p(x)$ by first using importance sampling (with proposal q) to generate a distribution of the form

$$p(\mathbf{x}) \approx \sum_s w_s \delta_{\mathbf{x}^s}(\mathbf{x}) \tag{23.30}$$

where w_s are the normalized importance weights. We then sample with replacement from Equation 23.30, where the probability that we pick $\mathbf{x}^s$ is w_s. Let this procedure induce a distribution denoted by $\hat{p}$. To see that this is valid, note that

$$\hat{p}(x \leq x_0) = \sum_s \mathbb{I}(x^s \leq x_0) w_s = \frac{\sum_s \mathbb{I}(x^s \leq x_0)\tilde{p}(x^s)/q(x^s)}{\sum_s \tilde{p}(x^s)/q(x^s)} \tag{23.31}$$

$$\rightarrow \frac{\int \mathbb{I}(x \leq x_0)\frac{\tilde{p}(x)}{q(x)} q(x) dx}{\int \frac{\tilde{p}(x)}{q(x)} q(x) dx} \tag{23.32}$$

$$= \frac{\int \mathbb{I}(x \leq x_0)\tilde{p}(x) dx}{\int \tilde{p}(x) dx} = \int \mathbb{I}(x \leq x_0) p(x) dx = p(x \leq x_0) \tag{23.33}$$

This is known as **sampling importance resampling** (SIR) (Rubin 1998). The result is an unweighted approximation of the form

$$p(\mathbf{x}) \approx \frac{1}{S'} \sum_{s=1}^{S'} \delta_{\mathbf{x}^s}(\mathbf{x}) \tag{23.34}$$

Note that we typically take $S' \ll S$.

This algorithm can be used to perform Bayesian inference in low-dimensional settings (Smith and Gelfand 1992). That is, suppose we want to draw (unweighted) samples from the posterior, $p(\boldsymbol{\theta}|\mathcal{D}) = p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})/p(\mathcal{D})$. We can use importance sampling with $\tilde{p}(\boldsymbol{\theta}) = p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})$ as the unnormalized posterior, and $q(\boldsymbol{\theta}) = p(\boldsymbol{\theta})$ as our proposal. The normalized weights have the form

$$w_s = \frac{\tilde{p}(\boldsymbol{\theta}_s)/q(\boldsymbol{\theta}_s)}{\sum_{s'} \tilde{p}(\boldsymbol{\theta}_{s'})/q(\boldsymbol{\theta}_{s'})} = \frac{p(\mathcal{D}|\boldsymbol{\theta}_s)}{\sum_{s'} p(\mathcal{D}|\boldsymbol{\theta}_{s'})} \tag{23.35}$$

We can then use SIR to sample from $p(\boldsymbol{\theta}|\mathcal{D})$.

Of course, if there is a big discrepancy between our proposal (the prior) and the target (the posterior), we will need a huge number of importance samples for this technique to work reliably, since otherwise the variance of the importance weights will be very large, implying that most samples carry no useful information. (This issue will come up again in Section 23.5, when we discuss particle filtering.)

23.5 Particle filtering

Particle filtering (PF) is a Monte Carlo, or **simulation based**, algorithm for recursive Bayesian inference. That is, it approximates the predict-update cycle described in Section 18.3.1. It is very widely used in many areas, including tracking, time-series forecasting, online parameter learning, etc. We explain the basic algorithm below. For a book-length treatment, see (Doucet et al. 2001); for a good tutorial, see (Arulampalam et al. 2002), or just read on.

23.5.1 Sequential importance sampling

The basic idea is to appproximate the belief state (of the entire state trajectory) using a weighted set of particles:

$$p(\mathbf{z}_{1:t}|\mathbf{y}_{1:t}) \approx \sum_{s=1}^{S} \hat{w}_t^s \delta_{\mathbf{z}_{1:t}^s}(\mathbf{z}_{1:t}) \tag{23.36}$$

where $\hat{w}_t^s$ is the normalized weight of sample s at time t. From this representation, we can easily compute the marginal distribution over the most recent state, $p(\mathbf{z}_t|\mathbf{y}_{1:t})$, by simply ignoring the previous parts of the trajectory, $\mathbf{z}_{1:t-1}$. (The fact that PF samples in the space of entire trajectories has various implications which we will discuss later.)

We update this belief state using importance sampling. If the proposal has the form $q(\mathbf{z}_{1:t}^s|\mathbf{y}_{1:t})$, then the importance weights are given by

$$w_t^s \propto \frac{p(\mathbf{z}_{1:t}^s|\mathbf{y}_{1:t})}{q(\mathbf{z}_{1:t}^s|\mathbf{y}_{1:t})} \tag{23.37}$$

which can be normalized as follows:

$$\hat{w}_t^s = \frac{w_t^s}{\sum_{s'} w_t^{s'}} \tag{23.38}$$

We can rewrite the numerator recursively as follows:

$$p(\mathbf{z}_{1:t}|\mathbf{y}_{1:t}) = \frac{p(\mathbf{y}_t|\mathbf{z}_{1:t}, \mathbf{y}_{1:t-1}) p(\mathbf{z}_{1:t}|\mathbf{y}_{1:t-1})}{p(\mathbf{y}_t|\mathbf{y}_{1:t-1})} \tag{23.39}$$

$$= \frac{p(\mathbf{y}_t|\mathbf{z}_t) p(\mathbf{z}_t|\mathbf{z}_{1:t-1}, \mathbf{y}_{1:t-1}) p(\mathbf{z}_{1:t-1}|\mathbf{y}_{1:t-1})}{p(\mathbf{y}_t|\mathbf{y}_{1:t-1})} \tag{23.40}$$

$$\propto p(\mathbf{y}_t|\mathbf{z}_t) p(\mathbf{z}_t|\mathbf{z}_{t-1}) p(\mathbf{z}_{1:t-1}|\mathbf{y}_{1:t-1}) \tag{23.41}$$

where we have made the usual Markov assumptions. We will restrict attention to proposal densities of the following form:

$$q(\mathbf{z}_{1:t}|\mathbf{y}_{1:t}) = q(\mathbf{z}_t|\mathbf{z}_{1:t-1}, \mathbf{y}_{1:t}) q(\mathbf{z}_{1:t-1}|\mathbf{y}_{1:t-1}) \tag{23.42}$$

so that we can "grow" the trajectory by adding the new state $\mathbf{z}_t$ to the end. In this case, the importance weights simplify to

$$w_t^s \propto \frac{p(\mathbf{y}_t|\mathbf{z}_t^s) p(\mathbf{z}_t^s|\mathbf{z}_{t-1}^s) p(\mathbf{z}_{1:t-1}^s|\mathbf{y}_{1:t-1})}{q(\mathbf{z}_t^s|\mathbf{z}_{1:t-1}^s, \mathbf{y}_{1:t}) q(\mathbf{z}_{1:t-1}^s|\mathbf{y}_{1:t-1})} \tag{23.43}$$

$$= w_{t-1}^s \frac{p(\mathbf{y}_t|\mathbf{z}_t^s) p(\mathbf{z}_t^s|\mathbf{z}_{t-1}^s)}{q(\mathbf{z}_t^s|\mathbf{z}_{1:t-1}^s, \mathbf{y}_{1:t})} \tag{23.44}$$

If we further assume that $q(\mathbf{z}_t|\mathbf{z}_{1:t-1}, \mathbf{y}_{1:t}) = q(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{y}_t)$, then we only need to keep the most recent part of the trajectory and observation sequence, rather than the whole history, in order to compute the new sample. In this case, the weight becomes

$$w_t^s \propto w_{t-1}^s \frac{p(\mathbf{y}_t|\mathbf{z}_t^s) p(\mathbf{z}_t^s|\mathbf{z}_{t-1}^s)}{q(\mathbf{z}_t^s|\mathbf{z}_{t-1}^s, \mathbf{y}_t)} \tag{23.45}$$

Hence we can approximate the posterior filtered density using

$$p(\mathbf{z}_t|\mathbf{y}_{1:t}) \approx \sum_{s=1}^{S} \hat{w}_t^s \delta_{\mathbf{z}_t^s}(\mathbf{z}_t) \tag{23.46}$$

As $S \to \infty$, one can show that this approaches the true posterior (Crisan et al. 1999).

The basic algorithm is now very simple: for each old sample s, propose an extension using $\mathbf{z}_t^s \sim q(\mathbf{z}_t|\mathbf{z}_{t-1}^s, \mathbf{y}_t)$, and give this new particle weight w_t^s using Equation 23.45. Unfortunately, this basic algorithm does not work very well, as we discuss below.

23.5.2 The degeneracy problem

The basic sequential importance sampling algorithm fails after a few steps because most of the particles will have negligible weight. This is called the **degeneracy problem**, and occurs because we are sampling in a high-dimensional space (in fact, the space is growing in size over time), using a myopic proposal distribution.

We can quantify the degree of degeneracy using the **effective sample size**, defined by

$$S_{\text{eff}} \triangleq \frac{S}{1 + \text{var}\left[w_t^{*s}\right]} \tag{23.47}$$

where $w_t^{*s} = p(\mathbf{z}_t^s|\mathbf{y}_{1:t})/q(\mathbf{z}_t^s|\mathbf{z}_{t-1}^s, \mathbf{y}_t)$ is the "true weight" of particle s. This quantity cannot be computed exactly, since we don't know the true posterior, but we can approximate it using

$$\hat{S}_{\text{eff}} = \frac{1}{\sum_{s=1}^{S}(w_t^s)^2} \tag{23.48}$$

If the variance of the weights is large, then we are wasting our resources updating particles with low weight, which do not contribute much to our posterior estimate.

There are two main solutions to the degeneracy problem: adding a resampling step, and using a good proposal distribution. We discuss both of these in turn.

23.5.3 The resampling step

The main improvement to the basic SIS algorithm is to monitor the effective sampling size, and whenever it drops below a threshold, to eliminate particles with low weight, and then to create replicates of the surviving particles. (Hence PF is sometimes called **survival of the fittest** (Kanazawa et al. 1995).) In particular, we generate a new set $\{\mathbf{z}_t^{s*}\}_{s=1}^S$ by sampling with replacement S times from the weighted distribution

$$p(\mathbf{z}_t|\mathbf{y}_{1:t}) \approx \sum_{s=1}^{S} \hat{w}_t^s \delta_{\mathbf{z}_t^s}(\mathbf{z}_t) \tag{23.49}$$

where the probability of choosing particle j for replication is w_t^j. (This is sometimes called **rejuvenation**.) The result is an iid *unweighted* sample from the discrete density Equation 23.49, so we set the new weights to $w_t^s = 1/S$. This scheme is illustrated in Figure 23.4.

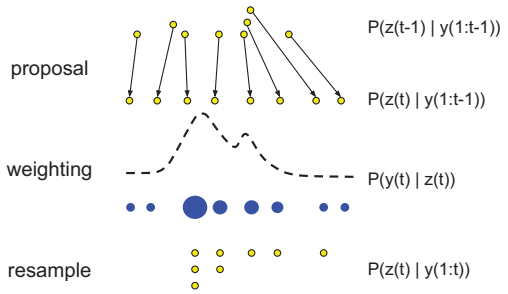


Figure 23.4 Illustration of particle filtering.

There are a variety of algorithms for peforming the resampling step. The simplest is **multinomial resampling**, which computes

$$(K_1, \ldots, K_S) \sim \text{Mu}(S, (w_t^1, \ldots, w_t^S)) \tag{23.50}$$

We then make K_s copies of $\mathbf{z}_t^s$. Various improvements exist, such as **systematic resampling residual resampling**, and **stratified sampling**, which can reduce the variance of the weights. All these methods take $O(S)$ time. See (Doucet et al. 2001) for details.

The overall particle filtering algorithm is summarized in Algorithm 6. (Note that if an estimate of the state is required, it should be computed before the resampling step, since this will result in lower variance.)

Algorithm 23.1: One step of a generic particle filter

for $s = 1 : S$ **do**
  Draw $\mathbf{z}_t^s \sim q(\mathbf{z}_t|\mathbf{z}_{t-1}^s, \mathbf{y}_t)$;
  Compute weight $w_t^s \propto w_{t-1}^s \frac{p(\mathbf{y}_t|\mathbf{z}_t^s)p(\mathbf{z}_t^s|\mathbf{z}_{t-1}^s)}{q(\mathbf{z}_t^s|\mathbf{z}_{t-1}^s,\mathbf{y}_t)}$;
Normalize weights: $w_t^s = \frac{w_t^s}{\sum_{s'} w_t^{s'}}$;
Compute $\hat{S}_{\text{eff}} = \frac{1}{\sum_{s=1}^S (w_t^s)^2}$;
if $\hat{S}_{eff} < S_{min}$ **then**
  Resample S indices $\boldsymbol{\pi} \sim \mathbf{w}_t$;
  $\mathbf{z}_t^{:} = \mathbf{z}_t^{\boldsymbol{\pi}}$;
  $w_t^s = 1/S$;

Although the resampling step helps with the degeneracy problem, it introduces problems of its own. In particular, since the particles with high weight will be selected many times, there is a loss of diversity amongst the population. This is known as **sample impoverishment**. In the

extreme case of no process noise (e.g., if we have static but unknown parameters as part of the state space), then all the particles will collapse to a single point within a few iterations.

To mitigate this problem, several solutions have been proposed. (1) Only resample when necessary, not at every time step. (The original **bootstrap filter** (Gordon 1993) resampled at every step, but this is suboptimal.) (2) After replicating old particles, sample new values using an MCMC step which leaves the posterior distribution invariant (see e.g., the **resample-move** algorithm in (Gilks and Berzuini 2001)). (3) Create a kernel density estimate on top of the particles,

$$p(\mathbf{z}_t|\mathbf{y}_{1:t}) \approx \sum_{s=1}^{S} w_t^s \kappa(\mathbf{z}_t - \mathbf{z}_t^s) \tag{23.51}$$

where κ is some smoothing kernel. We then sample from this smoothed distribution. This is known as a **regularized particle filter** (Musso et al. 2001). (4) When performing inference on static parameters, add some artificial process noise. (If this is undesirable, other algorithms must be used for online parameter estimation, e.g., (Andrieu et al. 2005)).

23.5.4 The proposal distribution

The simplest and most widely used proposal distribution is to sample from the prior:

$$q(\mathbf{z}_t|\mathbf{z}_{t-1}^s, \mathbf{y}_t) = p(\mathbf{z}_t|\mathbf{z}_{t-1}^s) \tag{23.52}$$

In this case, the weight update simplifies to

$$w_t^s \propto w_{t-1}^s p(\mathbf{y}_t|\mathbf{z}_t^s) \tag{23.53}$$

This can be thought of a "generate and test" approach: we sample values from the dynamic model, and then evaluate how good they are after we see the data (see Figure 23.4). This is the approach used in the **condensation** algorithm (which stands for "conditional density propagation") used for visual tracking (Isard and Blake 1998). However, if the likelihood is narrower than the dynamical prior (meaning the sensor is more informative than the motion model, which is often the case), this is a very inefficient approach, since most particles will be assigned very low weight.

It is much better to actually look at the data $\mathbf{y}_t$ when generating a proposal. In fact, the optimal proposal distribution has the following form:

$$q(\mathbf{z}_t|\mathbf{z}_{t-1}^s, \mathbf{y}_t) = p(\mathbf{z}_t|\mathbf{z}_{t-1}^s, \mathbf{y}_t) = \frac{p(\mathbf{y}_t|\mathbf{z}_t)p(\mathbf{z}_t|\mathbf{z}_{t-1}^s)}{p(\mathbf{y}_t|\mathbf{z}_{t-1}^s)} \tag{23.54}$$

If we use this proposal, the new weight is given by

$$w_t^s \propto w_{t-1}^s p(\mathbf{y}_t|\mathbf{z}_{t-1}^s) = w_{t-1}^s \int p(\mathbf{y}_t|\mathbf{z}_t')p(\mathbf{z}_t'|\mathbf{z}_{t-1}^s)d\mathbf{z}_t' \tag{23.55}$$

This proposal is optimal since, for any given $\mathbf{z}_{t-1}^s$, the new weight w_t^s takes the same value regardless of the value drawn for $\mathbf{z}_t^s$. Hence, conditional on the old values $\mathbf{z}_{t-1}^i$, the variance of true weights $\mathrm{var}\,[w_t^{*s}]$, is zero.

In general, it is intractable to sample from $p(\mathbf{z}_t|\mathbf{z}_{t-1}^s, \mathbf{y}_t)$ and to evaluate the integral needed to compute the predictive density $p(\mathbf{y}_t|\mathbf{z}_{t-1}^s)$. However, there are two cases when the optimal proposal distribution can be used. The first setting is when $\mathbf{z}_t$ is discrete, so the integral becomes a sum. Of course, if the entire state space is discrete, we can use an HMM filter instead, but in some cases, some parts of the state are discrete, and some continuous. The second setting is when $p(\mathbf{z}_t|\mathbf{z}_{t-1}^s, \mathbf{y}_t)$ is Gaussian. This occurs when the dynamics are nonlinear but the observations are linear. See Exercise 23.3 for the details.

In cases where the model is not linear-Gaussian, we may still compute a Gaussian approximation to $p(\mathbf{z}_t|\mathbf{z}_{t-1}^s, \mathbf{y}_t)$ using the unscented transform (Section 18.5.2) and use this as a proposal. This is known as the **unscented particle filter** (van der Merwe et al. 2000). In more general settings, we can use other kinds of **data-driven proposals**, perhaps based on discriminative models. Unlike MCMC, we do not need to worry about the proposals being reversible.

23.5.5 Application: robot localization

Consider a mobile robot wandering around an office environment. We will assume that it already has a map of the world, represented in the form of an **occupancy grid**, which just specifies whether each grid cell is empty space or occupied by an something solid like a wall. The goal is for the robot to estimate its location. This can be solved optimally using an HMM filter, since we are assuming the state space is discrete. However, since the number of states, K, is often very large, the $O(K^2)$ time complexity per update is prohibitive. We can use a particle filter as a sparse approximation to the belief state. This is known as **Monte Carlo localization**, and is described in detail in (Thrun et al. 2006).

Figure 23.5 gives an example of the method in action. The robot uses a sonar range finder, so it can only sense distance to obstacles. It starts out with a uniform prior, reflecting the fact that the owner of the robot may have turned it on in an arbitrary location. (Figuring out where you are, starting from a uniform prior, is called **global localization**.) After the first scan, which indicates two walls on either side, the belief state is shown in (b). The posterior is still fairly broad, since the robot could be in any location where the walls are fairly close by, such as a corridor or any of the narrow rooms. After moving to location 2, the robot is pretty sure it must be in the corridor, as shown in (c). After moving to location 3, the sensor is able to detect the end of the corridor. However, due to symmetry, it is not sure if it is in location I (the true location) or location II. (This is an example of **perceptual aliasing**, which refers to the fact that different things may look the same.) After moving to locations 4 and 5, it is finally able to figure out precisely where it is. The whole process is analogous to someone getting lost in an office building, and wandering the corridors until they see a sign they recognize.

In Section 23.6.3, we discuss how to estimate location and the map at the same time.

23.5.6 Application: visual object tracking

Our next example is concerned with tracking an object (in this case, a remote-controlled helicopter) in a video sequence. The method uses a simple linear motion model for the centroid of the object, and a color histogram for the likelihood model, using **Bhattacharya distance** to compare histograms. The proposal distribution is obtained by sampling from the likelihood. See (Nummiaro et al. 2003) for further details.

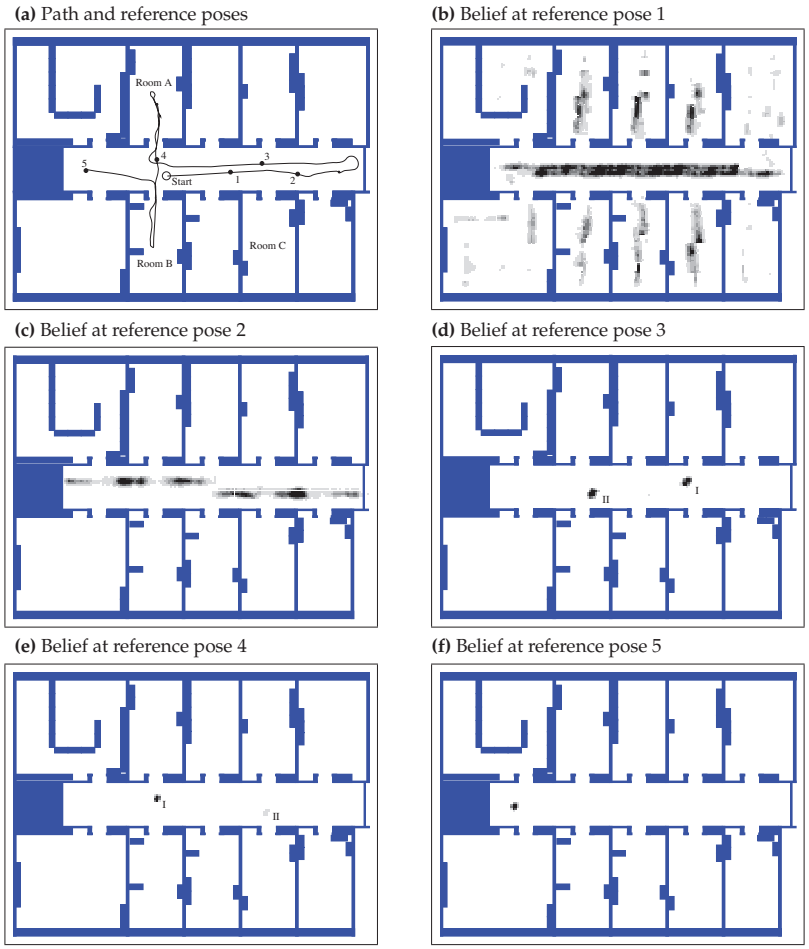


Figure 23.5 Illustration of Monte Carlo localization. Source: Figure 8.7 of (Thrun et al. 2006). Used with kind permission of Sebastian Thrun.

Figure 23.6 shows some example frames. The system uses $S = 250$ particles, with an effective sample size of $\hat{S}_{\text{eff}} = 134$. (a) shows the belief state at frame 1. The system has had to resample 5 times to keep the effective sample size above the threshold of 150; (b) shows the belief state at frame 251; the red lines show the estimated location of the center of the object over the last 250 frames. (c) shows that the system can handle visual clutter, as long as it does not have the same color as the target object. (d) shows that the system is confused between the grey of the helicopter and the grey of the building. The posterior is bimodal. The green ellipse, representing the posterior mean and covariance, is in between the two modes. (e) shows that the probability mass has shifted to the wrong mode: the system has lost track. (f) shows the particles spread out over the gray building; recovery of the object is very unlikely from this state using this

Figure 23.6 Example of particle filtering applied to visual object tracking, based on color histograms. (a-c) succesful tracking: green ellipse is on top of the helicopter. (d-f): tracker gets distracted by gray clutter in the background. See text for details. Figure generated by `pfColorTrackerDemo`, written by Sebastien Paris.

proposal.

We see that the method is able to keep track for a fairly long time, despite the presence of clutter. However, eventually it loses track of the object. Note that since the algorithm is stochastic, simply re-running the demo may fix the problem. But in the real world, this is not an option. The simplest way to improve performance is to use more particles. An alternative is to perform **tracking by detection**, by running an object detector over the image every few frames. See (Forsyth and Ponce 2002; Szeliski 2010; Prince 2012) for details.

23.5.7 Application: time series forecasting

In Section 18.2.4, we discussed how to use the Kalman filter to perform time series forecasting. This assumes that the model is a linear-Gaussian state-space model. There are many models which are either non-linear and/or non-Gaussian. For example, **stochastic volatility** models, which are widely used in finance, assume that the variance of the system and/or observation noise changes over time. Particle filtering is widely used in such settings. See e.g., (Doucet et al. 2001) and references therein for details.

23.6 Rao-Blackwellised particle filtering (RBPF)

In some models, we can partition the hidden variables into two kinds, $\mathbf{q}_t$ and $\mathbf{z}_t$, such that we can analytically integrate out $\mathbf{z}_t$ provided we know the values of $\mathbf{q}_{1:t}$. This means we only have sample $\mathbf{q}_{1:t}$, and can represent $p(\mathbf{z}_t|\mathbf{q}_{1:t})$ parametrically. Thus each particle s represents a value for $\mathbf{q}^s_{1:t}$ and a distribution of the form $p(\mathbf{z}_t|\mathbf{y}_{1:t}, \mathbf{q}^s_{1:t})$. These hybrid particles are are sometimes called **distributional particles** or **collapsed particles** (Koller and Friedman 2009, Sec 12.4).

The advantage of this approach is that we reduce the dimensionality of the space in which we are sampling, which reduces the variance of our estimate. Hence this technique is known as **Rao-Blackwellised particle filtering** or **RBPF** for short, named after Theorem 24.20. The method is best explained using a specific example.

23.6.1 RBPF for switching LG-SSMs

A canonical example for which RBPF can be applied is the switching linear dynamical system (SLDS) model discussed in Section 18.6 (Chen and Liu 2000; Doucet et al. 2001). We can represent $p(\mathbf{z}_t|\mathbf{y}_{1:t}, \mathbf{q}^s_{1:t})$ using a mean and covariance matrix for each particle s, where $q_t \in \{1, \ldots, K\}$.

If we propose from the prior, $q(q_t = k|q^s_{t-1})$, the weight update becomes

$$w^s_t \propto w^s_{t-1} p(\mathbf{y}_t|q_t = k, \mathbf{q}^s_{1:t-1}, \mathbf{y}_{1:t-1}) = w^s_{t-1} L^s_{t,k} \tag{23.56}$$

where

$$L^s_{tk} = \int p(\mathbf{y}_t|q_t = k, \mathbf{z}_t, \cancel{\mathbf{y}_{1:t-1}}, \cancel{\mathbf{q}^s_{1:t-1}}) p(\mathbf{z}_t|q_t = k, \mathbf{y}_{1:t-1}\mathbf{q}^s_{1:t-1},) d\mathbf{z}_t \tag{23.57}$$

The quantity L^s_{tk} is the predictive density for the new observation $\mathbf{y}_t$ conditioned on $q_t = k$ and the history $\mathbf{q}^s_{1:t-1}$. In the case of SLDS models, this can be computed using the normalization constant of the Kalman filter, Equation 18.41.

We give some pseudo-code in Algorithm 8. (The step marked "KFupdate" refers to the Kalman filter update equations in Section 18.3.1.) This is known as a **mixture of Kalman filters**.

If K is small, we can compute the optimal proposal distribution, which is

$$p(q_t = k|\mathbf{y}_{1:t}, \mathbf{q}^s_{1:t-1}) = \hat{p}^s_{t-1}(q_t = k|\mathbf{y}_t) \tag{23.58}$$

$$= \frac{\hat{p}^s_{t-1}(\mathbf{y}_t|q_t = k)\hat{p}^s_{t-1}(q_t = k)}{\hat{p}^s_{t-1}(\mathbf{y}_t)} \tag{23.59}$$

$$= \frac{L^s_{tk} p(q_t = k|q^s_{t-1})}{\sum_{k'} L^s_{tk'} p(q_t = k'|q^s_{t-1})} \tag{23.60}$$

Algorithm 23.2: One step of RBPF for SLDS using prior as proposal

for $s = 1 : S$ **do**
  $k \sim p(q_t|q_{t-1}^s)$;
  $q_t^s := k$;
  $(\boldsymbol{\mu}_t^s, \boldsymbol{\Sigma}_t^s, L_{tk}^s) = \text{KFupdate}(\boldsymbol{\mu}_{t-1}^s, \boldsymbol{\Sigma}_{t-1}^s, \mathbf{y}_t, \boldsymbol{\theta}_k)$;
  $w_t^s = w_{t-1}^s L_{ts}^k$;
Normalize weights: $w_t^s = \frac{w_t^s}{\sum_{s'} w_t^{s'}}$;
Compute $\hat{S}_{\text{eff}} = \frac{1}{\sum_{s=1}^{S}(w_t^s)^2}$;
if $\hat{S}_{eff} < S_{min}$ **then**
  Resample S indices $\boldsymbol{\pi} \sim \mathbf{w}_t$;
  $\mathbf{q}_t^{:} = \mathbf{q}_t^{\boldsymbol{\pi}}$, $\boldsymbol{\mu}_t^{:} = \boldsymbol{\mu}_t^{\boldsymbol{\pi}}$, $\boldsymbol{\Sigma}_t^{:} = \boldsymbol{\Sigma}_t^{\boldsymbol{\pi}}$, ;
  $w_t^s = 1/S$;

where we use the following shorthand:

$$\hat{p}_{t-1}^s(\cdot) = p(\cdot|\mathbf{y}_{1:t-1}, \mathbf{q}_{1:t-1}^s) \tag{23.61}$$

We then sample from $p(q_t|\mathbf{q}_{1:t-1}^s, \mathbf{y}_{1:t})$ and give the resulting particle weight

$$w_t^s \propto w_{t-1}^s p(\mathbf{y}_t|\mathbf{q}_{1:t-1}^s, \mathbf{y}_{1:t-1}) = w_{t-1}^s \sum_k \left[L_{tk}^s p(q_t = k|q_{t-1}^s)\right] \tag{23.62}$$

Since the weights of the particles in Equation 23.62 are independent of the new value that is actually sampled for q_t, we can compute these weights first, and use them to decide which particles to propagate. That is, we choose the fittest particles at time $t-1$ using information from time t. This is called **look-ahead RBPF** (de Freitas et al. 2004).

In more detail, the idea is this. We pass each sample in the prior through all K models to get K posteriors, one per sample. The normalization constants of this process allow us to compute the optimal weights in Equation 23.62. We then resample S indices. Finally, for each old particle s that is chosen, we sample one new state $q_t^s = k$, and use the corresponding posterior from the K possible alternative that we have already computed. The pseudo-code is shown in Algorithm 7. This method needs $O(KS)$ storage, but has the advantage that each particle is chosen using the latest information, $\mathbf{y}_t$.

A further improvement can be obtained by exploiting the fact that the state space is discrete. Hence we can use the resampling method of (Fearnhead 2004) which avoids duplicating particles.

23.6.2 Application: tracking a maneuvering target

One application of SLDS is to track moving objects that have piecewise linear dynamics. For example, suppose we want to track an airplane or missile; q_t can specify if the object is flying normally or is taking evasive action. This is called **maneuvering target tracking**.

Figure 23.7 gives an example of an object moving in 2d. The setup is essentially the same as in Section 18.2.1, except that we add a three-state discrete Markov chain which controls the

Algorithm 23.3: One step of look-ahead RBPF for SLDS using optimal proposal

for $s = 1 : S$ **do**
  for $k = 1 : K$ **do**
    $(\boldsymbol{\mu}_{tk}^s, \boldsymbol{\Sigma}_{tk}^s, L_{ts}^k) = \text{KFupdate}(\boldsymbol{\mu}_{t-1}^s, \boldsymbol{\Sigma}_{t-1}^s, \mathbf{y}_t, \boldsymbol{\theta}_k)$;
  $w_t^s = w_{t-1}^s [\sum_k L_{ts}^k p(q_t = k | q_{t-1}^s)]$;
Normalize weights: $w_t^s = \frac{w_t^s}{\sum_{s'} w_t^{s'}}$;
Resample S indices $\boldsymbol{\pi} \sim \mathbf{w}_t$;
for $s \in \boldsymbol{\pi}$ **do**
  Compute optimal proposal $p(k|\mathbf{q}_{1:t-1}^s, \mathbf{y}_{1:t}) = \frac{L_{tk}^s p(q_t=k|q_{t-1}^s)}{\sum_{k'} L_{tk}^s p(q_t=k|q_{t-1}^s)}$;
  Sample $k \sim p(k|\mathbf{q}_{1:t-1}^s, \mathbf{y}_{1:t})$;
  $q_t^s = k$, $\boldsymbol{\mu}_t^s = \boldsymbol{\mu}_{tk}^s$, $\boldsymbol{\Sigma}_t^s = \boldsymbol{\Sigma}_{tk}^s$;
  $w_t^s = 1/S$;

Method	misclassification rate	MSE	Time (seconds)
PF	0.440	21.051	6.086
RBPF	0.340	18.168	10.986

Table 23.1 Comparison of PF an RBPF on the maneuvering target problem in Figure 23.7.

input to the system. We define $\mathbf{u}_t = 1$ and set

$$\mathbf{B}_1 = (0, 0, 0, 0)^T, \mathbf{B}_2 = (-1.225, -0.35, 1.225, 0.35)^T, \mathbf{B}_3 = (1.225, 0.35, -1.225, -0.35)^T$$

so the system will turn in different directions depending on the discrete state.

Figure 23.7(a) shows the true state of the system from a sample run, starting at $(0, 0)$: the colored symbols denote the discrete state, and the location of the symbol denotes the (x, y) location. The small dots represent noisy observations. Figure 23.7(b) shows the estimate of the state computed using particle filtering with 500 particles, where the proposal is to sample from the prior. The colored symbols denote the MAP estimate of the state, and the location of the symbol denotes the MMSE (minimum mean square error) estimate of the location, which is given by the posterior mean. Figure 23.7(c) shows the estimate computing using RBPF with 500 particles, using the optimal proposal distribution. A more quantitative comparison is shown in Table 23.1. We see that RBPF has slightly better performance, although it is also slightly slower.

Figure 23.8 visualizes the belief state of the system. In (a) we show the distribution over the discrete states. We see that the particle filter estimate of the belief state (second column) is not as accurate as the RBPF estimate (third column) in the beginning, although after the first few observations performance is similar for both methods. In (b), we plot the posterior over the x locations. For simplicity, we use the PF estimate, which is a set of weighted samples, but we could also have used the RBPF estimate, which is a set of weighted Gaussians.

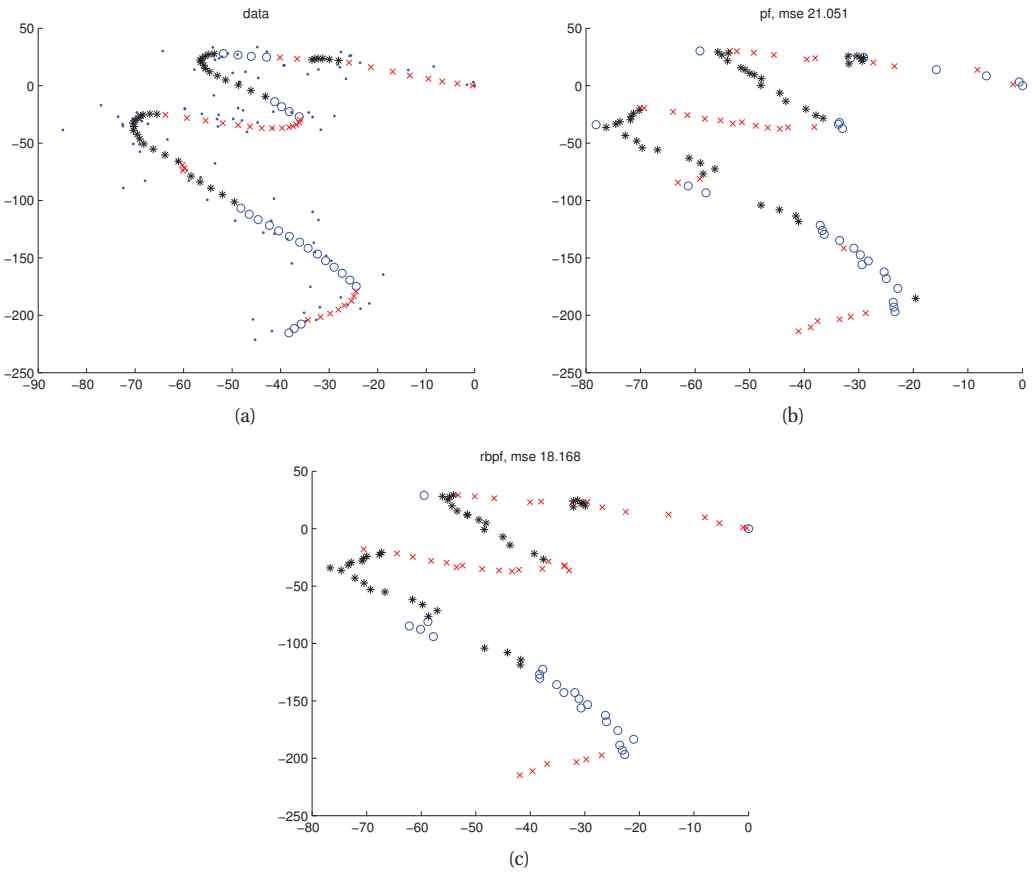
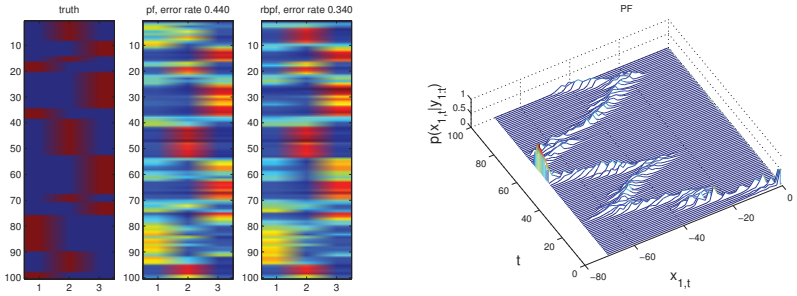


Figure 23.7 (a) A maneuvering target. The colored symbols represent the hidden discrete state. (b) Particle filter estimate. (c) RBPF estimate. Figure generated by `rbpfManeuverDemo`, based on code by Nando de Freitas.

23.6.3 Application: Fast SLAM

In Section 18.2.2, we introduced the problem of simultaneous localization and mapping or SLAM for mobile robotics. The main problem with the Kalman filter implementation is that it is cubic in the number of landmarks. However, by looking at the DGM in Figure 18.2, we see that, conditional on knowing the robot's path, $\mathbf{q}_{1:t}$, where $\mathbf{q}_t \in \mathbb{R}^2$, the landmark locations $\mathbf{z} \in \mathbb{R}^{2L}$ are independent. (We assume the landmarks don't move, so we drop the t subscript). That is, $p(\mathbf{z}|\mathbf{q}_{1:t}, \mathbf{y}_{1:t}) = \prod_{l=1}^{L} p(\mathbf{z}_l|\mathbf{q}_{1:t}, \mathbf{y}_{1:t})$. Consequently we can use RBPF, where we sample the robot's trajectory, $\mathbf{q}_{1:t}$, and we run L independent 2d Kalman filters inside each particle. This takes $O(L)$ time per particle. Fortunately, the number of particles needed for good performance is quite small (this partly depends on the control / exploration policy), so the algorithm is essentially linear in the number of particles. This technique has the additional advantage that



(a) (b)

Figure 23.8 Belief states corresponding to Figure 23.7. (a) Discrete state. The system starts in state 2 (red x in Figure 23.7), then moves to state 3 (black * in Figure 23.7), returns briefly to state 2, then switches to state 1 (blue circle in Figure 23.7), etc. (b) Horizontal location (PF estimate). Figure generated by `rbpfManeuverDemo`, based on code by Nando de Freitas.

it is easy to use sampling to handle the data association ambiguity, and that it allows for other representations of the map, such as occupancy grids. This idea was first suggested in (Murphy 2000), and was subsequently extended and made practical in (Thrun et al. 2004), who christened the technique **FastSLAM**. See `rbpfSlamDemo` for a simple demo in a discrete grid world.

Exercises

Exercise 23.1 Sampling from a Cauchy

Show how to use inverse probability transform to sample from a standard Cauchy, $\mathcal{T}(x|0,1,1)$.

Exercise 23.2 Rejection sampling from a Gamma using a Cauchy proposal

Show how to use a Cauchy proposal to perform rejection sampling from a Gamma distribution. Derive the optimal constant M, and plot the density and its upper envelope.

Exercise 23.3 Optimal proposal for particle filtering with linear-Gaussian measurement model

Consider a state-space model of the following form:

$$\mathbf{z}_t = f_t(\mathbf{z}_{t-1}) + \mathcal{N}(\mathbf{0}, \mathbf{Q}_{t-1}) \tag{23.63}$$

$$\mathbf{y}_t = \mathbf{H}_t \mathbf{z}_t + \mathcal{N}(\mathbf{0}, \mathbf{R}_t) \tag{23.64}$$

Derive expressions for $p(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{y}_t)$ and $p(\mathbf{y}_t|\mathbf{z}_{t-1})$, which are needed to compute the optimal (minimum variance) proposal distribution. Hint: use Bayes rule for Gaussians.

24 *Markov chain Monte Carlo (MCMC) inference*

24.1 Introduction

In Chapter 23, we introduced some simple Monte Carlo methods, including rejection sampling and importance sampling. The trouble with these methods is that they do not work well in high dimensional spaces. The most popular method for sampling from high-dimensional distributions is **Markov chain Monte Carlo** or **MCMC**. In a survey by *SIAM News*[1], MCMC was placed in the top 10 most important algorithms of the 20th century.

The basic idea behind MCMC is to construct a Markov chain (Section 17.2) on the state space $\mathcal{X}$ whose stationary distribution is the target density $p^*(\mathbf{x})$ of interest (this may be a prior or a posterior). That is, we perform a random walk on the state space, in such a way that the fraction of time we spend in each state $\mathbf{x}$ is proportional to $p^*(\mathbf{x})$. By drawing (correlated!) samples $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots,$ from the chain, we can perform Monte Carlo integration wrt p^*. We give the details below.

The MCMC algorithm has an interesting history. It was discovered by physicists working on the atomic bomb at Los Alamos during World War II, and was first published in the open literature in (Metropolis et al. 1953) in a chemistry journal. An extension was published in the statistics literature in (Hastings 1970), but was largely unnoticed. A special case (Gibbs sampling, Section 24.2) was independently invented in 1984 in the context of Ising models and was published in (Geman and Geman 1984). But it was not until (Gelfand and Smith 1990) that the algorithm became well-known to the wider statistical community. Since then it has become wildly popular in Bayesian statistics, and is becoming increasingly popular in machine learning.

It is worth briefly comparing MCMC to variational inference (Chapter 21). The advantages of variational inference are (1) for small to medium problems, it is usually faster; (2) it is deterministic; (3) is it easy to determine when to stop; (4) it often provides a lower bound on the log likelihood. The advantages of sampling are: (1) it is often easier to implement; (2) it is applicable to a broader range of models, such as models whose size or structure changes depending on the values of certain variables (e.g., as happens in matching problems), or models without nice conjugate priors; (3) sampling can be faster than variational methods when applied to really huge models or datasets.[2]

1. Source: `http://www.siam.org/pdf/news/637.pdf`.

2. The reason is that sampling passes specific values of variables (or sets of variables), whereas in variational inference, we pass around distributions. Thus sampling passes sparse messages, whereas variational inference passes dense messages For comparisons of the two approaches, see e.g., (Yoshida and West 2010) and articles in (Bekkerman et al.

24.2 Gibbs sampling

In this section, we present one of the most popular MCMC algorithms, known as **Gibbs sampling**.[3] (In physics, this method is known as **Glauber dynamics** or the **heat bath** method.) This is the MCMC analog of coordinate descent.

24.2.1 Basic idea

The idea behind Gibbs sampling is that we sample each variable in turn, conditioned on the values of all the other variables in the distribution. That is, given a joint sample $\mathbf{x}^s$ of all the variables, we generate a new sample $\mathbf{x}^{s+1}$ by sampling each component in turn, based on the most recent values of the other variables. For example, if we have $D = 3$ variables, we use

- $x_1^{s+1} \sim p(x_1|x_2^s, x_3^s)$
- $x_2^{s+1} \sim p(x_2|x_1^{s+1}, x_3^s)$
- $x_3^{s+1} \sim p(x_3|x_1^{s+1}, x_2^{s+1})$

This readily generalizes to D variables. If x_i is a visible variable, we do not sample it, since its value is already known.

The expression $p(x_i|\mathbf{x}_{-i})$ is called the **full conditional** for variable i. In general, x_i may only depend on some of the other variables. If we represent $p(\mathbf{x})$ as a graphical model, we can infer the dependencies by looking at i's Markov blanket, which are its neighbors in the graph. Thus to sample x_i, we only need to know the values of i's neighbors. In this sense, Gibbs sampling is a distributed algorithm. However, it is not a parallel algorithm, since the samples must be generated sequentially.

For reasons that we will explain in Section 24.4.1, it is necessary to discard some of the initial samples until the Markov chain has **burned in**, or entered its stationary distribution. We discuss how to estimate when burnin has occured in Section 24.4.1. In the examples below, we just discard the initial 25% of the samples, for simplicity.

24.2.2 Example: Gibbs sampling for the Ising model

In Section 21.3.2, we applied mean field to an Ising model. Here we apply Gibbs sampling.

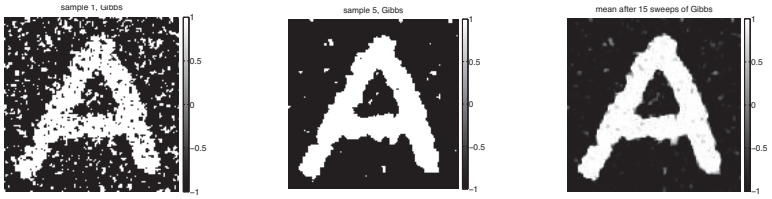
Gibbs sampling in pairwise MRF/CRF takes the form

$$p(x_t|\mathbf{x}_{-t}, \boldsymbol{\theta}) \propto \prod_{s \in \text{nbr}(t)} \psi_{st}(x_s, x_t) \tag{24.1}$$

In the case of an Ising model with edge potentials $\psi(x_s, x_t) = \exp(Jx_s x_t)$, where $x_t \in$

2011)

3. Josiah Willard Gibbs, 1839–1903, was an American physicist.



(a) (b) (c)

Figure 24.1 Example of image denoising. We use an Ising prior with $W_{ij} = J = 1$ and a Gaussian noise model with $\sigma = 2$. We use Gibbs sampling (Section 24.2) to perform approximate inference. (a) Sample from the posterior after one sweep over the image. (b) Sample after 5 sweeps. (c) Posterior mean, computed by averaging over 15 sweeps. Compare to Figure 21.3 which shows the results of using mean field inference. Figure generated by `isingImageDenoiseDemo`.

$\{-1, +1\}$, the full conditional becomes

$$p(x_t = +1|\mathbf{x}_{-t}, \boldsymbol{\theta}) = \frac{\prod_{s\in\text{nbr}(t)} \psi_{st}(x_t = +1, x_s)}{\prod_{s\in\text{nbr}(t)} \psi(s_t = +1, x_s) + \prod_{s\in\text{nbr}(t)} \psi(x_t = -1, x_s)} \tag{24.2}$$

$$= \frac{\exp[J \sum_{s\in\text{nbr}(t)} x_s]}{\exp[J \sum_{s\in\text{nbr}(t)} x_s] + \exp[-J \sum_{s\in\text{nbr}(t)} x_s]} \tag{24.3}$$

$$= \frac{\exp[J\eta_t]}{\exp[J\eta_t] + \exp[-J\eta_t]} = \text{sigm}(2J\eta_t) \tag{24.4}$$

where J is the coupling strength, $\eta_t \triangleq \sum_{s\in\text{nbr}(t)} x_t$ and $\text{sigm}(u) = 1/(1+e^{-u})$ is the sigmoid function. It is easy to see that $\eta_t = x_t(a_t - d_t)$, where a_t is the number of neighbors that agree with (have the same sign as) t, and d_t is the number of neighbors who disagree. If this number is equal, the "forces" on x_t cancel out, so the full conditional is uniform.

We can combine an Ising prior with a local evidence term ψ_t. For example, with a Gaussian observation model, we have $\psi_t(x_t) = \mathcal{N}(y_t|x_t, \sigma^2)$. The full conditional becomes

$$p(x_t = +1|\mathbf{x}_{-t}, \mathbf{y}, \boldsymbol{\theta}) = \frac{\exp[J\eta_t]\psi_t(+1)}{\exp[J\eta_t]\psi_t(+1) + \exp[-J\eta_t]\psi_t(-1)} \tag{24.5}$$

$$= \text{sigm}\left(2J\eta_t - \log\frac{\psi_t(+1)}{\psi_t(-1)}\right) \tag{24.6}$$

Now the probability of x_t entering each state is determined both by compatibility with its neighbors (the Ising prior) and compatibility with the data (the local likelihood term).

See Figure 24.1 for an example of this algorithm applied to a simple image denoising problem. The results are similar to mean field (Figure 21.3) except that the final estimate (based on averaging the samples) is somewhat "blurrier", due to the fact that mean field tends to be over-confident.

24.2.3 Example: Gibbs sampling for inferring the parameters of a GMM

It is straightforward to derive a Gibbs sampling algorithm to "fit" a mixture model, especially if we use conjugate priors. We will focus on the case of mixture of Gaussians, although the results are easily extended to other kinds of mixture models. (The derivation, which follows from the results of Section 4.6, is much easier than the corresponding variational Bayes algorithm in Section 21.6.1.)

Suppose we use a semi-conjugate prior. Then the full joint distribution is given by

$$p(\mathbf{x}, \mathbf{z}, \boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\pi}) = p(\mathbf{x}|\mathbf{z}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) p(\mathbf{z}|\boldsymbol{\pi}) p(\boldsymbol{\pi}) \prod_{k=1}^{K} p(\boldsymbol{\mu}_k) p(\boldsymbol{\Sigma}_k) \tag{24.7}$$

$$= \left(\prod_{i=1}^{N} \prod_{k=1}^{K} (\pi_k \mathcal{N}(\mathbf{x}_i|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k))^{\mathbb{I}(z_i=k)} \right) \times \tag{24.8}$$

$$\mathrm{Dir}(\boldsymbol{\pi}|\boldsymbol{\alpha}) \prod_{k=1}^{K} \mathcal{N}(\boldsymbol{\mu}_k|\mathbf{m}_0, \mathbf{V}_0) \mathrm{IW}(\boldsymbol{\Sigma}_k|\mathbf{S}_0, \nu_0) \tag{24.9}$$

We use the same prior for each mixture component. The full conditionals are as follows. For the discrete indicators, we have

$$p(z_i = k|\mathbf{x}_i, \boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\pi}) \propto \pi_k \mathcal{N}(\mathbf{x}_i|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \tag{24.10}$$

For the mixing weights, we have (using results from Section 3.4)

$$p(\boldsymbol{\pi}|\mathbf{z}) = \mathrm{Dir}(\{\alpha_k + \sum_{i=1}^{N} \mathbb{I}(z_i = k)\}_{k=1}^{K}) \tag{24.11}$$

For the means, we have (using results from Section 4.6.1)

$$p(\boldsymbol{\mu}_k|\boldsymbol{\Sigma}_k, \mathbf{z}, \mathbf{x}) = \mathcal{N}(\boldsymbol{\mu}_k|\mathbf{m}_k, \mathbf{V}_k) \tag{24.12}$$

$$\mathbf{V}_k^{-1} = \mathbf{V}_0^{-1} + N_k \boldsymbol{\Sigma}_k^{-1} \tag{24.13}$$

$$\mathbf{m}_k = \mathbf{V}_k(\boldsymbol{\Sigma}_k^{-1} N_k \bar{\mathbf{x}}_k + \mathbf{V}_0^{-1} \mathbf{m}_0) \tag{24.14}$$

$$N_k \triangleq \sum_{i=1}^{N} \mathbb{I}(z_i = k) \tag{24.15}$$

$$\bar{\mathbf{x}}_k \triangleq \frac{\sum_{i=1}^{N} \mathbb{I}(z_i = k)\mathbf{x}_i}{N_k} \tag{24.16}$$

For the covariances, we have (using results from Section 4.6.2)

$$p(\boldsymbol{\Sigma}_k|\boldsymbol{\mu}_k, \mathbf{z}, \mathbf{x}) = \mathrm{IW}(\boldsymbol{\Sigma}_k|\mathbf{S}_k, \nu_k) \tag{24.17}$$

$$\mathbf{S}_k = \mathbf{S}_0 + \sum_{i=1}^{N} \mathbb{I}(z_i = k)(\mathbf{x}_i - \boldsymbol{\mu}_k)(\mathbf{x}_i - \boldsymbol{\mu}_k)^T \tag{24.18}$$

$$\nu_k = \nu_0 + N_k \tag{24.19}$$

See `gaussMissingFitGibbs` for some Matlab code. (This code can also sample missing values for $\mathbf{x}$, if necessary.)

24.2.3.1 Label switching

Although it is simple to implement, Gibbs sampling for mixture models has a fundamental weakness. The problem is that the parameters of the model $\boldsymbol{\theta}$, and the indicator functions $\mathbf{z}$, are unidentifiable, since we can arbitrarily permute the hidden labels without affecting the likelihood (see Section 11.3.1). Consequently, we cannot just take a Monte Carlo average of the samples to compute posterior means, since what one sample considers the parameters for cluster 1 may be what another sample considers the parameters for cluster 2. Indeed, if we could average over all modes, we would find $\mathbb{E}[\boldsymbol{\mu}_k|\mathcal{D}]$ is the same for all k (assuming a symmetric prior). This is called the **label switching** problem.

This problem does not arise in EM or VBEM, which just "lock on" to a single mode. However, it arises in any method that visits multiple modes. In 1d problems, one can try to prevent this problem by introducing constraints on the parameters to ensure identifiability, e.g., $\mu_1 < \mu_2 < \mu_3$ (Richardson and Green 1997). However, this does not always work, since the likelihood might overwhelm the prior and cause label switching anyway. Furthermore, this technique does not scale to higher dimensions. Another approach is to post-process the samples by searching for a global label permutation to apply to each sample that minimizes some loss function (Stephens 2000); however, this can be slow.

Perhaps the best solution is simply to "not ask" questions that cannot be uniquely identified. For example, instead of asking for the probability that data point i belongs to cluster k, ask for the probability that data points i and j belong to the same cluster. The latter question is invariant to the labeling. Furthermore, it only refers to observable quantities (are i and j grouped together or not), rather than referring to unobservable quantities, such as latent clusters. This approach has the further advantage that it extends to infinite mixture models, discussed in Section 25.2, where K is unbounded; in such models, the notion of a hidden cluster is not well defined, but the notion of a partitioning of the data *is* well defined

24.2.4 Collapsed Gibbs sampling *

In some cases, we can analytically integrate out some of the unknown quantities, and just sample the rest. This is called a **collapsed Gibbs sampler**, and it tends to be much more efficient, since it is sampling in a lower dimensional space.

More precisely, suppose we sample $\mathbf{z}$ and integrate out $\boldsymbol{\theta}$. Thus the $\boldsymbol{\theta}$ parameters do not participate in the Markov chain; consequently we can draw conditionally independent samples $\boldsymbol{\theta}^s \sim p(\boldsymbol{\theta}|\mathbf{z}^s, \mathcal{D})$, which will have much lower variance than samples drawn from the joint state space (Liu et al. 1994). This process is called **Rao-Blackwellisation**, named after the following theorem:

Theorem 24.2.1 (Rao-Blackwell)**.** *Let* $\mathbf{z}$ *and* $\boldsymbol{\theta}$ *be dependent random variables, and* $f(\mathbf{z}, \boldsymbol{\theta})$ *be some scalar function. Then*

$$\mathrm{var}_{\mathbf{z},\boldsymbol{\theta}}[f(\mathbf{z}, \boldsymbol{\theta})] \geq \mathrm{var}_{\mathbf{z}}[\mathbb{E}_{\boldsymbol{\theta}}[f(\mathbf{z}, \boldsymbol{\theta})|\mathbf{z}]] \tag{24.20}$$

This theorem guarantees that the variance of the estimate created by analytically integrating out $\boldsymbol{\theta}$ will always be lower (or rather, will never be higher) than the variance of a direct MC estimate. In collapsed Gibbs, we sample $\mathbf{z}$ with $\boldsymbol{\theta}$ integrated out; the above Rao-Blackwell theorem still applies in this case (Liu et al. 1994).

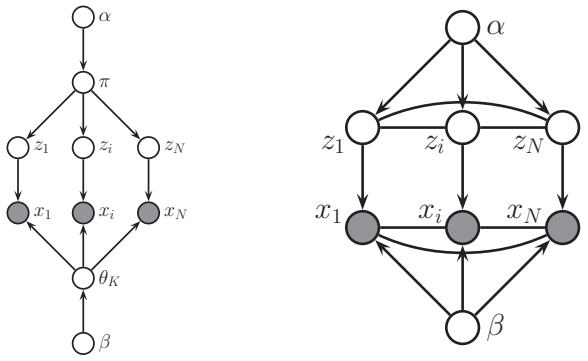


Figure 24.2 (a) A mixture model. (b) After integrating out the parameters.

We will encounter Rao-Blackwellisation again in Section 23.6. Although it can reduce statistical variance, it is only worth doing if the integrating out can be done quickly, otherwise we will not be able to produce as many samples per second as the naive method. We give an example of this below.

24.2.4.1 Example: collapsed Gibbs for fitting a GMM

Consider a GMM with a fully conjugate prior. In this case we can analytically integrate out the model parameters $\boldsymbol{\mu}_k$, $\boldsymbol{\Sigma}_k$ and $\boldsymbol{\pi}$, and just sample the indicators $\mathbf{z}$. Once we integrate out $\boldsymbol{\pi}$, all the z_i nodes become inter-dependent. Similarly, once we integrate out $\boldsymbol{\theta}_k$, all the $\mathbf{x}_i$ nodes become inter-dependent, as shown in Figure 24.2(b). Nevertheless, we can easily compute the full conditionals as follows:

$$
\begin{aligned}
p(z_i = k|\mathbf{z}_{-i}, \mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}) &\propto p(z_i = k|\mathbf{z}_{-i}, \boldsymbol{\alpha}, \cancel{\boldsymbol{\beta}}) p(\mathbf{x}|z_i = k, \mathbf{z}_{-i}, \cancel{\boldsymbol{\alpha}}, \boldsymbol{\beta}) && (24.21)\\
&\propto p(z_i = k|\mathbf{z}_{-i}, \boldsymbol{\alpha}) p(\mathbf{x}_i|\mathbf{x}_{-i}, z_i = k, \mathbf{z}_{-i}, \boldsymbol{\beta}) \\
&\quad p(\mathbf{x}_{-i}|\cancel{z_i = k}, \mathbf{z}_{-i}, \boldsymbol{\beta}) && (24.22)\\
&\propto p(z_i = k|\mathbf{z}_{-i}, \boldsymbol{\alpha}) p(\mathbf{x}_i|\mathbf{x}_{-i}, z_i = k, \mathbf{z}_{-i}, \boldsymbol{\beta}) && (24.23)
\end{aligned}
$$

where $\boldsymbol{\beta} = (\mathbf{m}_0, \mathbf{V}_0, \mathbf{S}_0, \nu_0)$ are the hyper-parameters for the class-conditional densities. The first term can be obtained by integrating out $\boldsymbol{\pi}$. Suppose we use a symmetric prior of the form $\boldsymbol{\pi} \sim \mathrm{Dir}(\boldsymbol{\alpha})$, where $\alpha_k = \alpha/K$. From Equation 5.26 we have

$$
p(z_1, \ldots, z_N|\alpha) = \frac{\Gamma(\alpha)}{\Gamma(N+\alpha)} \prod_{k=1}^{K} \frac{\Gamma(N_k + \alpha/K)}{\Gamma(\alpha/K)} \tag{24.24}
$$

Hence

$$p(z_i = k|\mathbf{z}_{-i}, \alpha) = \frac{p(\mathbf{z}_{1:N}|\alpha)}{p(\mathbf{z}_{-i}|\alpha)} = \frac{\frac{1}{\Gamma(N+\alpha)}}{\frac{1}{\Gamma(N+\alpha-1)}} \times \frac{\Gamma(N_k + \alpha/K)}{\Gamma(N_{k,-i} + \alpha/K)} \tag{24.25}$$

$$= \frac{\Gamma(N+\alpha-1)}{\Gamma(N+\alpha)} \frac{\Gamma(N_{k,-i} + 1 + \alpha/K)}{\Gamma(N_{k,-i} + \alpha/K)} = \frac{N_{k,-i} + \alpha/K}{N + \alpha - 1} \tag{24.26}$$

where $N_{k,-i} \triangleq \sum_{n \neq i} \mathbb{I}(z_n = k) = N_k - 1$, and where we exploited the fact that $\Gamma(x+1) = x\Gamma(x)$.

To obtain the second term in Equation 24.23, which is the posterior predictive distribution for $\mathbf{x}_i$ given all the other data and all the assignments, we use the fact that

$$p(\mathbf{x}_i|\mathbf{x}_{-i}, \mathbf{z}_{-i}, z_i = k, \boldsymbol{\beta}) = p(\mathbf{x}_i|\mathcal{D}_{-i,k}) \tag{24.27}$$

where $\mathcal{D}_{-i,k} = \{\mathbf{x}_j : z_j = k, j \neq i\}$ is all the data assigned to cluster k except for $\mathbf{x}_i$. If we use a conjugate prior for $\boldsymbol{\theta}_k$, we can compute $p(\mathbf{x}_i|\mathcal{D}_{-i,k})$ in closed form. Furthermore, we can efficiently update these predictive likelihoods by caching the sufficient statistics for each cluster. To compute the above expression, we remove $\mathbf{x}_i$'s statistics from its current cluster (namely z_i), and then evaluate $\mathbf{x}_i$ under each cluster's posterior predictive. Once we have picked a new cluster, we add $\mathbf{x}_i$'s statistics to this new cluster.

Some pseudo-code for one step of the algorithm is shown in Algorithm 1, based on (Sudderth 2006, p94). (We update the nodes in random order to improve the mixing time, as suggested in (Roberts and Sahu 1997).) We can initialize the sample by sequentially sampling from $p(z_i|\mathbf{z}_{1:i-1}, \mathbf{x}_{1:i})$. (See `fmGibbs` for some Matlab code, by Yee-Whye Teh.) In the case of GMMs, both the naive sampler and collapsed sampler take $O(NKD)$ time per step.

Algorithm 24.1: Collapsed Gibbs sampler for a mixture model

```
for each i = 1 : N in random order do
    Remove x_i's sufficient statistics from old cluster z_i ;
    for each k = 1 : K do
        Compute p_k(x_i) ≜ p(x_i|{x_j : z_j = k, j ≠ i}) ;
    Compute p(z_i = k|z_{-i}, D) ∝ (N_{k,-i} + α/K) p_k(x_i);
    Sample z_i ~ p(z_i|·) ;
    Add x_i's sufficient statistics to new cluster z_i
```

A comparison of this method with the standard Gibbs sampler is shown in Figure 24.3. The vertical axis is the data log probability at each iteration, computed using

$$\log p(\mathcal{D}|\mathbf{z}, \boldsymbol{\theta}) = \sum_{i=1}^{N} \log \left[\pi_{z_i} p(\mathbf{x}_i|\boldsymbol{\theta}_{z_i})\right] \tag{24.28}$$

To compute this quantity using the collapsed sampler, we have to sample $\boldsymbol{\theta} = (\boldsymbol{\pi}, \boldsymbol{\theta}_{1:K})$ given the data and the current assignment $\mathbf{z}$.

In Figure 24.3 we see that the collapsed sampler does indeed generally work better than the vanilla sampler. Occasionally, however, both methods can get stuck in poor local modes. (Note

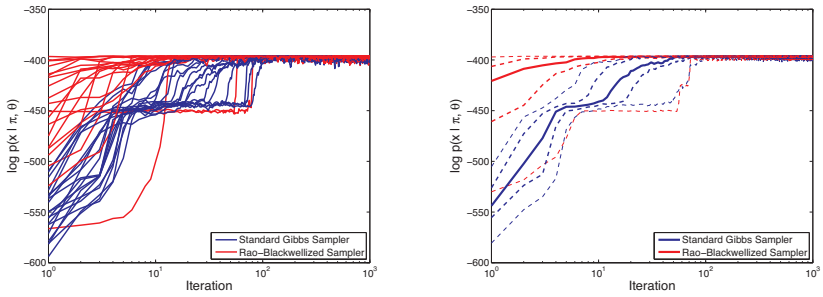


Figure 24.3 Comparison of collapsed (red) and vanilla (blue) Gibbs sampling for a mixture of $K = 4$ two-dimensional Gaussians applied to $N = 300$ data points (shown in Figure 25.7). We plot log probability of the data vs iteration. (a) 20 different random initializations. (b) logprob averaged over 100 different random initializations. Solid line is the median, thick dashed in the 0.25 and 0.75 quantiles, and thin dashed are the 0.05 and 0.95 quintiles. Source: Figure 2.20 of (Sudderth 2006). Used with kind permission of Erik Sudderth.

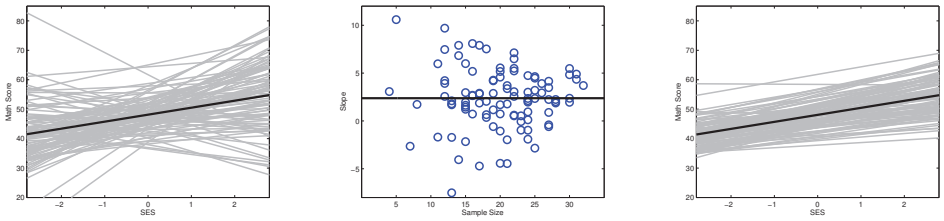


Figure 24.4 (a) Least squares regression lines for math scores vs socio-economic status for 100 schools. Population mean (pooled estimate) is in bold. (b) Plot of $\hat{w}_{2j}$ (the slope) vs N_j (sample size) for the 100 schools. The extreme slopes tend to correspond to schools with smaller sample sizes. (c) Predictions from the hierarchical model. Population mean is in bold. Based on Figure 11.1 of (Hoff 2009). Figure generated by `multilevelLinregDemo`, written by Emtiyaz Khan.

that the error bars in Figure 24.3(b) are averaged over starting values, whereas the theorem refers to MC samples in a single run.)

24.2.5 Gibbs sampling for hierarchical GLMs

Often we have data from multiple related sources. If some sources are more reliable and/or data-rich than others, it makes sense to model all the data simultaneously, so as to enable the borrowing of statistical strength. One of the most natural way to solve such problems is to use hierarchical Bayesian modeling, also called multi-level modeling. In Section 9.6, we discussed a way to perform approximate inference in such models using variational methods. Here we discuss how to use Gibbs sampling.

To explain the method, consider the following example. Suppose we have data on students

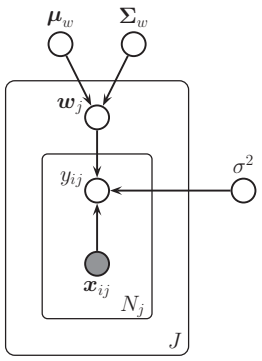


Figure 24.5 Multi-level model for linear regression.

in different schools. Such data is naturally modeled in a two-level hierarchy: we let y_{ij} be the response variable we want to predict for student i in school j. This prediction can be based on school and student specific covariates, $\mathbf{x}_{ij}$. Since the quality of schools varies, we want to use a separate parameter for each school. So our model becomes

$$y_{ij} = \mathbf{x}_{ij}^T \mathbf{w}_j + \epsilon_{ij} \tag{24.29}$$

We will illustrate this model below, using a dataset from (Hoff 2009, p197), where x_{ij} is the socio-economic status (SES) of student i in school y, and y_{ij} is their math score.

We could fit each $\mathbf{w}_j$ separately, but this can give poor results if the sample size of a given school is small. This is illustrated in Figure 24.4(a), which plots the least squares regression line estimated separately for each of the $J = 100$ schools. We see that most of the slopes are positive, but there are a few "errant" cases where the slope is negative. It turns out that the lines with extreme slopes tend to be in schools with small sample size, as shown in Figure 24.4(b). Thus we may not necessarily trust these fits.

We can get better results if we construct a hierarchical Bayesian model, in which the $\mathbf{w}_j$ are assumed to come from a common prior: $\mathbf{w}_j \sim \mathcal{N}(\boldsymbol{\mu}_w, \boldsymbol{\Sigma}_w)$. This is illustrated in Figure 24.5. In this model, the schools with small sample size borrow statistical strength from the schools with larger sample size, because the $\mathbf{w}_j$'s are correlated via the latent common parents $(\boldsymbol{\mu}_w, \boldsymbol{\Sigma}_w)$. (It is crucial that these hyper-parameters be inferrred from data; if they were fixed constants, the $\mathbf{w}_j$ would be conditionally independent, and there would be no information sharing between them.)

To complete the model specification, we must specify priors for the shared parameters. Following (Hoff 2009, p198), we will use the following semi-conjugate forms, for convenience:

$$\boldsymbol{\mu}_w \sim \mathcal{N}(\boldsymbol{\mu}_0, \mathbf{V}_0) \tag{24.30}$$

$$\boldsymbol{\Sigma}_w \sim \mathrm{IW}(\eta_0, \mathbf{S}_0^{-1}) \tag{24.31}$$

$$\sigma^2 \sim \mathrm{IG}(\nu_0/2, \nu_0\sigma_0^2/2) \tag{24.32}$$

Given this, it is simple to show that the full conditionals needed for Gibbs sampling have the

following forms. For the group-specific weights:

$$p(\mathbf{w}_j|\mathcal{D}_j,\boldsymbol{\theta}) = \mathcal{N}(\mathbf{w}_j|\boldsymbol{\mu}_j,\boldsymbol{\Sigma}_j) \tag{24.33}$$

$$\boldsymbol{\Sigma}_j^{-1} = \boldsymbol{\Sigma}^{-1} + \mathbf{X}_j^T\mathbf{X}_j/\sigma^2 \tag{24.34}$$

$$\boldsymbol{\mu}_j = \boldsymbol{\Sigma}_j(\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu} + \mathbf{X}_j^T\mathbf{y}_j/\sigma^2) \tag{24.35}$$

For the overall mean:

$$p(\boldsymbol{\mu}_w|\mathbf{w}_{1:J},\boldsymbol{\Sigma}_w) = \mathcal{N}(\boldsymbol{\mu}|\boldsymbol{\mu}_N,\boldsymbol{\Sigma}_N) \tag{24.36}$$

$$\boldsymbol{\Sigma}_N^{-1} = \mathbf{V}_0^{-1} + J\boldsymbol{\Sigma}^{-1} \tag{24.37}$$

$$\boldsymbol{\mu}_N = \boldsymbol{\Sigma}_N(\mathbf{V}_0^{-1}\boldsymbol{\mu}_0 + J\boldsymbol{\Sigma}^{-1}\overline{\mathbf{w}}) \tag{24.38}$$

where $\overline{\mathbf{w}} = \frac{1}{J}\sum_j \mathbf{w}_j$. For the overall covariance:

$$p(\boldsymbol{\Sigma}_w|\boldsymbol{\mu}_w,\mathbf{w}_{1:J}) = \mathrm{IW}((\mathbf{S}_0+\mathbf{S}_\mu)^{-1},\eta_0+J) \tag{24.39}$$

$$\mathbf{S}_\mu = \sum_j(\mathbf{w}_j-\boldsymbol{\mu}_w)(\mathbf{w}_j-\boldsymbol{\mu}_w)^T \tag{24.40}$$

For the noise variance:

$$p(\sigma^2|\mathcal{D},\mathbf{w}_{1:J}) = \mathrm{IG}([\nu_0+N]/2,[\nu_0\sigma_0^2+\mathrm{SSR}(\mathbf{w}_{1:J})]/2) \tag{24.41}$$

$$\mathrm{SSR}(\mathbf{w}_{1:J}) = \sum_{j=1}^{J}\sum_{i=1}^{N_j}(y_{ij}-\mathbf{w}_j^T\mathbf{x}_{ij})^2 \tag{24.42}$$

Applying Gibbs sampling to our hierarchical model, we get the results shown in Figure 24.4(c). The light gray lines plot the mean of the posterior predictive distribution for each school:

$$\mathbb{E}\left[y_j|\mathbf{x}_{ij}\right] = \mathbf{x}_{ij}^T\hat{\mathbf{w}}_j \tag{24.43}$$

where

$$\hat{\mathbf{w}}_j = \mathbb{E}\left[\mathbf{w}_j|\mathcal{D}\right] \approx \frac{1}{S}\sum_{s=1}^{S}\mathbf{w}_j^{(s)} \tag{24.44}$$

The dark gray line in the middle plots the prediction using the overall mean parameters, $\mathbf{x}_{ij}^T\hat{\boldsymbol{\mu}}_w$. We see that the method has regularized the fits quite nicely, without enforcing too much uniformity. (The amount of shrinkage is controlled by $\boldsymbol{\Sigma}_w$, which in turns depends on the hyper-parameters; in this example, we used vague values.)

24.2.6 BUGS and JAGS

One reason Gibbs sampling is so popular is that it is possible to design general purpose software that will work for almost any model. This software just needs a model specification, usually in the form a directed graphical model (specified in a file, or created with a graphical user interface), and a library of methods for sampling from different kinds of full conditionals. (This can often be done using adaptive rejection sampling, described in Section 23.3.4.) An example

of such a package is **BUGS** (Lunn et al. 2000), which stands for "Bayesian updating using Gibbs Sampling". BUGS is very widely used in biostatistics and social science. Another more recent, but very similar, package is **JAGS** (Plummer 2003), which stands for "Just Another Gibbs Sampler". This uses a similar model specification language to BUGS.

For example, we can describe the model in Figure 24.5 as follows:

```
model {
 for (i in 1:N) {
    for (j in 1:J) {
       y[i,j] ~ dnorm(y.hat[i,j], tau.y)
       y.hat[i,j] <- inprod(W[j, ], X[i, j, ])
    }
}
tau.y <- pow(sigma.y, -2)
sigma.y ~ dunif(0,100)

for (j in 1:J) {
  W[j,] ~ dmnorm(mu, SigmaInv)
}
SigmaInv  ~ dwish(S0[,], eta0)
mu ~ dmnorm(mu0, V0inv)
}
```

We can then just pass this model to BUGS or JAGS, which will generate samples for us. See the webpages for details.

Although this approach is appealing, unfortunately it can be much slower than using hand-written code, especially for complex models. There has been some work on automatically deriving model-specific optimized inference code (Fischer and Schumann 2003), but fast code still typically requires human expertise.

24.2.7 The Imputation Posterior (IP) algorithm

The **Imputation Posterior** or IP algorithm (Tanner and Wong 1987) is a special case of Gibbs sampling in which we group the variables into two classes: hidden variables $\mathbf{z}$ and parameters $\boldsymbol{\theta}$. This should sound familiar: it is basically an MCMC version of EM, where the E step gets replaced by the I step, and the M step gets replaced the P step. This is an example of a more general strategy called **data augmentation**, whereby we introduce auxiliary variables in order to simplify the posterior computations (here the computation of $p(\boldsymbol{\theta}|\mathcal{D})$). See (Tanner 1996; van Dyk and Meng 2001) for more information.

24.2.8 Blocking Gibbs sampling

Gibbs sampling can be quite slow, since it only updates one variable at a time (so-called **single site updating**). If the variables are highly correlated, it will take a long time to move away from the current state. This is illustrated in Figure 24.6, where we illustrate sampling from a 2d Gaussian (see Exercise 24.1 for the details). If the variables are highly correlated, the algorithm

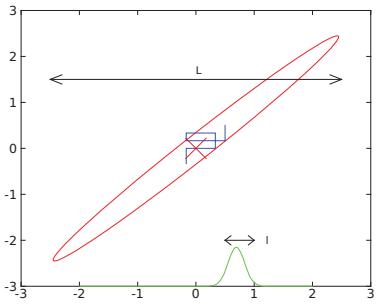


Figure 24.6 Illustration of potentially slow sampling when using Gibbs sampling for a skewed 2D Gaussian. Based on Figure 11.11 of (Bishop 2006b). Figure generated by `gibbsGaussDemo`.

will move very slowly through the state space. In particular, the size of the moves is controlled by the variance of the conditional distributions. If this is ℓ in the x_1 direction, and the support of the distribution is L along this dimension, then we need $O((L/\ell)^2)$ steps to obtain an independent sample.

In some cases we can efficiently sample groups of variables at a time. This is called **blocking Gibbs sampling** or **blocked Gibbs sampling** (Jensen et al. 1995; Wilkinson and Yeung 2002), and can make much bigger moves through the state space.

24.3 Metropolis Hastings algorithm

Although Gibbs sampling is simple, it is somewhat restricted in the set of models to which it can be applied. For example, it is not much help in computing $p(\mathbf{w}|\mathcal{D})$ for a logistic regression model, since the corresponding graphical model has no useful Markov structure. In addition, Gibbs sampling can be quite slow, as we mentioned above.

Fortunately, there is a more general algorithm that can be used, known as the **Metropolis Hastings** or **MH** algorithm, which we describe below.

24.3.1 Basic idea

The basic idea in MH is that at each step, we propose to move from the current state $\mathbf{x}$ to a new state $\mathbf{x}'$ with probability $q(\mathbf{x}'|\mathbf{x})$, where q is called the **proposal distribution** (also called the **kernel**). The user is free to use any kind of proposal they want, subject to some conditions which we explain below. This makes MH quite a flexible method. A commonly used proposal is a symmetric Gaussian distribution centered on the current state, $q(\mathbf{x}'|\mathbf{x}) = \mathcal{N}(\mathbf{x}'|\mathbf{x}, \boldsymbol{\Sigma})$; this is called a **random walk Metropolis algorithm**. We discuss how to choose $\boldsymbol{\Sigma}$ in Section 24.3.3. If we use a proposal of the form $q(\mathbf{x}'|\mathbf{x}) = q(\mathbf{x}')$, where the new state is independent of the old state, we get a method known as the **independence sampler**, which is similar to importance sampling (Section 23.4).

Having proposed a move to $\mathbf{x}'$, we then decide whether to **accept** this proposal or not according to some formula, which ensures that the fraction of time spent in each state is proportional to $p^*(\mathbf{x})$. If the proposal is accepted, the new state is $\mathbf{x}'$, otherwise the new state

is the same as the current state, $\mathbf{x}$ (i.e., we repeat the sample).

If the proposal is symmetric, so $q(\mathbf{x}'|\mathbf{x}) = q(\mathbf{x}|\mathbf{x}')$, the acceptance probability is given by the following formula:

$$r = \min(1, \frac{p^*(\mathbf{x}')}{p^*(\mathbf{x})}) \tag{24.45}$$

We see that if $\mathbf{x}'$ is more probable than $\mathbf{x}$, we definitely move there (since $\frac{p^*(\mathbf{x}')}{p^*(\mathbf{x})} > 1$), but if $\mathbf{x}'$ is less probable, we may still move there anyway, depending on the relative probabilities. So instead of greedily moving to only more probable states, we occasionally allow "downhill" moves to less probable states. In Section 24.3.6, we prove that this procedure ensures that the fraction of time we spend in each state $\mathbf{x}$ is proportional to $p^*(\mathbf{x})$.

If the proposal is asymmetric, so $q(\mathbf{x}'|\mathbf{x}) \neq q(\mathbf{x}|\mathbf{x}')$, we need the **Hastings correction**, given by the following:

$$r = \min(1, \alpha) \tag{24.46}$$

$$\alpha = \frac{p^*(\mathbf{x}')q(\mathbf{x}|\mathbf{x}')}{p^*(\mathbf{x})q(\mathbf{x}'|\mathbf{x})} = \frac{p^*(\mathbf{x}')/q(\mathbf{x}'|\mathbf{x})}{p^*(\mathbf{x})/q(\mathbf{x}|\mathbf{x}')} \tag{24.47}$$

This correction is needed to compensate for the fact that the proposal distribution itself (rather than just the target distribution) might favor certain states.

An important reason why MH is a useful algorithm is that, when evaluating α, we only need to know the target density up to a normalization constant. In particular, suppose $p^*(\mathbf{x}) = \frac{1}{Z}\tilde{p}(\mathbf{x})$, where $\tilde{p}(\mathbf{x})$ is an unnormalized distribution and Z is the normalization constant. Then

$$\alpha = \frac{(\tilde{p}(\mathbf{x}')/Z)\ q(\mathbf{x}|\mathbf{x}')}{(\tilde{p}(\mathbf{x})/Z)\ q(\mathbf{x}'|\mathbf{x})} \tag{24.48}$$

so the Z's cancel. Hence we can sample from p^* even if Z is unknown. In particular, all we have to do is evaluate $\tilde{p}$ pointwise, where $\tilde{p}(\mathbf{x}) = p^*(\mathbf{x})Z$.

The overall algorithm is summarized in Algorithm 2.

24.3.2 Gibbs sampling is a special case of MH

It turns out that Gibbs sampling, which we discussed in Section 24.2, is a special case of MH. In particular, it is equivalent to using MH with a sequence of proposals of the form

$$q(\mathbf{x}'|\mathbf{x}) = p(x_i'|\mathbf{x}_{-i})\mathbb{I}(\mathbf{x}'_{-i} = \mathbf{x}_{-i}) \tag{24.49}$$

That is, we move to a new state where x_i is sampled from its full conditional, but $\mathbf{x}_{-i}$ is left unchanged.

We now prove that the acceptance rate of each such proposal is 1, so the overall algorithm also has an acceptance rate of 100%. We have

$$\alpha = \frac{p(\mathbf{x}')q(\mathbf{x}|\mathbf{x}')}{p(\mathbf{x})q(\mathbf{x}'|\mathbf{x})} = \frac{p(x_i'|\mathbf{x}'_{-i})p(\mathbf{x}'_{-i})p(x_i|\mathbf{x}'_{-i})}{p(x_i|\mathbf{x}_{-i})p(\mathbf{x}_{-i})p(x_i'|\mathbf{x}_{-i})} \tag{24.50}$$

$$= \frac{p(x_i'|\mathbf{x}_{-i})p(\mathbf{x}_{-i})p(x_i|\mathbf{x}_{-i})}{p(x_i|\mathbf{x}_{-i})p(\mathbf{x}_{-i})p(x_i'|\mathbf{x}_{-i})} = 1 \tag{24.51}$$

Algorithm 24.2: Metropolis Hastings algorithm

Initialize x^0 ;
for $s = 0, 1, 2, \ldots$ **do**
  Define $x = x^s$;
  Sample $x' \sim q(x'|x)$;
  Compute acceptance probability

$$\alpha = \frac{\tilde{p}(x')q(x|x')}{\tilde{p}(x)q(x'|x)}$$

  Compute $r = \min(1, \alpha)$;
  Sample $u \sim U(0, 1)$;
  Set new sample to

$$x^{s+1} = \begin{cases} x' & \text{if } u < r \\ x^s & \text{if } u \geq r \end{cases}$$

where we exploited the fact that $\mathbf{x}'_{-i} = \mathbf{x}_{-i}$, and that $q(\mathbf{x}'|\mathbf{x}) = p(x'_i|\mathbf{x}_{-i})$.

The fact that the acceptance rate is 100% does not necessarily mean that Gibbs will converge rapidly, since it only updates one coordinate at a time (see Section 24.2.8). Fortunately, there are many other kinds of proposals we can use, as we discuss below.

24.3.3 Proposal distributions

For a given target distribution p^*, a proposal distribution q is valid or admissible if it gives a non-zero probability of moving to the states that have non-zero probability in the target. Formally, we can write this as

$$\text{supp}(p^*) \subseteq \cup_x \text{supp}(q(\cdot|x)) \tag{24.52}$$

For example, a Gaussian random walk proposal has non-zero probability density on the entire state space, and hence is a valid proposal for any continuous state space.

Of course, in practice, it is important that the proposal spread its probability mass in just the right way. Figure 24.7 shows an example where we use MH to sample from a mixture of two lD Gaussians using a random walk proposal, $q(x'|x) = \mathcal{N}(x'|x, v)$. This is a somewhat tricky target distribution, since it consists of two well separated modes. It is very important to set the variance of the proposal v correctly: If the variance is too low, the chain will only explore one of the modes, as shown in Figure 24.7(a), but if the variance is too large, most of the moves will be rejected, and the chain will be very **sticky**, i.e., it will stay in the same state for a long time. This is evident from the long stretches of repeated values in Figure 24.7(b). If we set the proposal's variance just right, we get the trace in Figure 24.7(c), where the samples clearly explore the support of the target distribution. We discuss how to tune the proposal below.

One big advantage of Gibbs sampling is that one does not need to choose the proposal

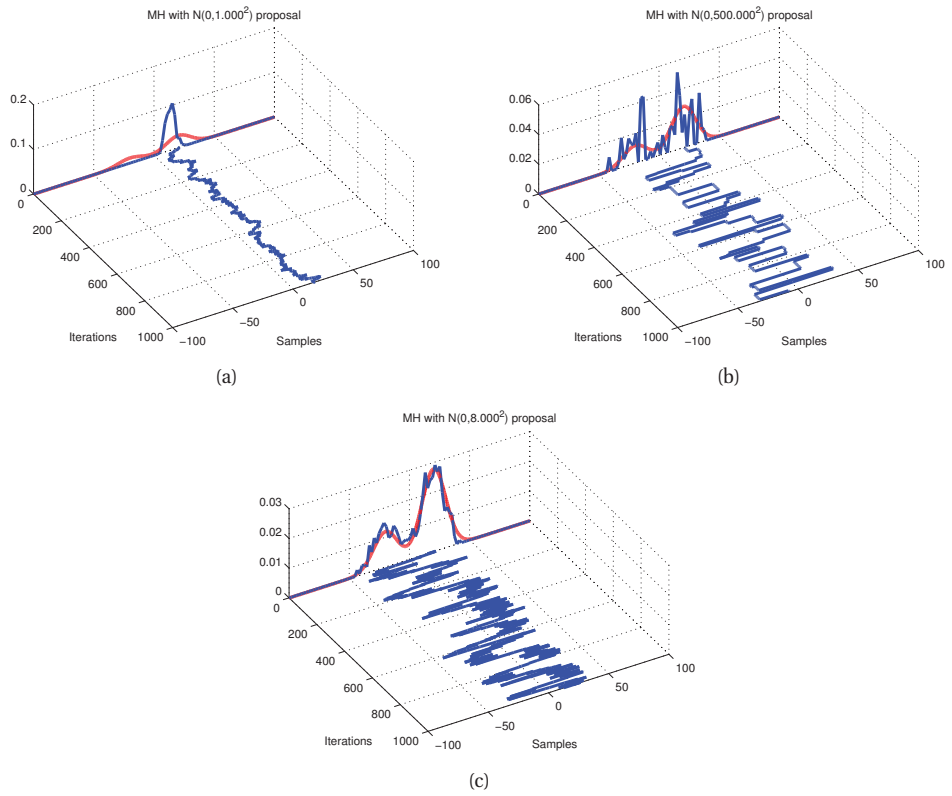
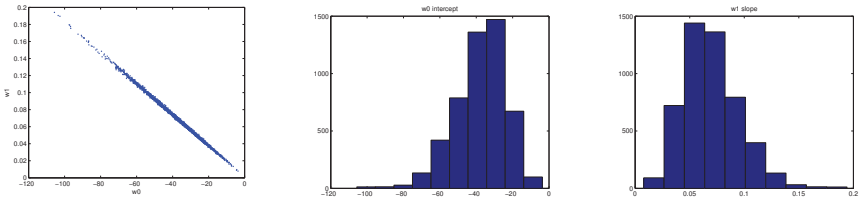


Figure 24.7 An example of the Metropolis Hastings algorithm for sampling from a mixture of two 1D Gaussians ($\boldsymbol{\mu} = (-20, 20)$, $\boldsymbol{\pi} = (0.3, 0.7)$, $\boldsymbol{\sigma} = (100, 100)$), using a Gaussian proposal with variances of $v \in \{1, 500, 8\}$. (a) When $v = 1$, the chain gets trapped near the starting state and fails to sample from the mode at $\mu = -20$. (b) When $v = 500$, the chain is very "sticky", so its effective sample size is low (as reflected by the rough histogram approximation at the end). (c) Using a variance of $v = 8$ is just right and leads to a good approximation of the true distribution (shown in red). Figure generated by `mcmcGmmDemo`. Based on code by Christophe Andrieu and Nando de Freitas.

distribution, and furthermore, the acceptance rate is 100%. Of course, a 100% acceptance can trivially be achieved by using a proposal with variance 0 (assuming we start at a mode), but this is obviously not exploring the posterior. So having a high acceptance is not the ultimate goal. We can increase the amount of exploration by increasing the variance of the Gaussian kernel. Often one experiments with different parameters until the acceptance rate is between 25% and 40%, which theory suggests is optimal, at least for Gaussian target distributions. These short initial runs, used to tune the proposal, are called **pilot runs**.



(a) (b) (c)

Figure 24.8 (a) Joint posterior of the parameters for 1d logistic regression when applied to some SAT data. (b) Marginal for the offset w_0. (c) Marginal for the slope w_1. We see that the marginals do not capture the fact that the parameters are highly correlated. Figure generated by `logregSatMhDemo`.

24.3.3.1 Gaussian proposals

If we have a continuous state space, the Hessian $\mathbf{H}$ at a local mode $\hat{\mathbf{w}}$ can be used to define the covariance of a Gaussian proposal distribution. This approach has the advantage that the Hessian models the local curvature and length scales of each dimension; this approach therefore avoids some of the slow mixing behavior of Gibbs sampling shown in Figure 24.6.

There are two obvious approaches: (1) an independence proposal, $q(\mathbf{w}'|\mathbf{w}) = \mathcal{N}(\mathbf{w}'|\hat{\mathbf{w}}, \mathbf{H}^{-1})$ or (2), a random walk proposal, $q(\mathbf{w}'|\mathbf{w}) = \mathcal{N}(\mathbf{w}'|\mathbf{w}, s^2\mathbf{H}^{-1})$, where s^2 is a scale factor chosen to facilitate rapid mixing. (Roberts and Rosenthal 2001) prove that, if the posterior is Gaussian, the asymptotically optimal value is to use $s^2 = 2.38^2/D$, where D is the dimensionality of $\mathbf{w}$; this results in an acceptance rate of 0.234.

For example, consider MH for binary logistic regression. From Equation 8.7, we have that the Hessian of the log-likelihood is $\mathbf{H}_l = \mathbf{X}^T\mathbf{D}\mathbf{X}$, where $\mathbf{D} = \text{diag}(\mu_i(1-\mu_i))$ and $\mu_i = \text{sigm}(\hat{\mathbf{w}}^T\mathbf{x}_i)$. If we assume a Gaussian prior, $p(\mathbf{w}) = \mathcal{N}(\mathbf{0}, \mathbf{V}_0)$, we have $\mathbf{H} = \mathbf{V}_0^{-1} + \mathbf{H}_l$, so the asymptotically optimal Gaussian proposal has the form

$$q(\mathbf{w}'|\mathbf{w}) = \mathcal{N}\left(\mathbf{w}, \frac{2.38^2}{D}\left(\mathbf{V}_0^{-1} + \mathbf{X}^T\mathbf{D}\mathbf{X}\right)^{-1}\right) \tag{24.53}$$

See (Gamerman 1997; Rossi et al. 2006; Fruhwirth-Schnatter and Fruhwirth 2010) for further details. The approach is illustrated in Figure 24.8, where we sample parameters from a 1d logistic regression model fit to some SAT data. We initialize the chain at the mode, computed using IRLS, and then use the above random walk Metropolis sampler.

If you cannot afford to compute the mode or its Hessian $\mathbf{X}\mathbf{D}\mathbf{X}$, an alternative approach, suggested in (Scott 2009), is to approximate the above proposal as follows:

$$q(\mathbf{w}'|\mathbf{w}) = \mathcal{N}\left(\mathbf{w}, \left(\mathbf{V}_0^{-1} + \frac{6}{\pi^2}\mathbf{X}^T\mathbf{X}\right)^{-1}\right) \tag{24.54}$$

24.3.3.2 Mixture proposals

If one doesn't know what kind of proposal to use, one can try a **mixture proposal**, which is a convex combination of base proposals:

$$q(\mathbf{x}'|\mathbf{x}) = \sum_{k=1}^{K} w_k q_k(\mathbf{x}'|\mathbf{x}) \tag{24.55}$$

where w_k are the mixing weights. As long as each q_k is individually valid, the overall proposal will also be valid.

24.3.3.3 Data-driven MCMC

The most efficient proposals depend not just on the previous hidden state, but also the visible data, i.e., they have the form $q(\mathbf{x}'|\mathbf{x}, \mathcal{D})$. This is called **data-driven MCMC** (see e.g., (Tu and Zhu 2002)). To create such proposals, one can sample $(\mathbf{x}, \mathcal{D})$ pairs from the forwards model and then train a discriminative classifier to predict $p(\mathbf{x}|f(\mathcal{D}))$, where $f(\mathcal{D})$ are some features extracted from the visible data.

Typically $\mathbf{x}$ is a high-dimensional vector (e.g., position and orientation of all the limbs of a person in a visual object detector), so it is hard to predict the entire state vector, $p(\mathbf{x}|f(\mathcal{D}))$. Instead we might train a discriminative detector to predict parts of the state-space, $p(x_k|f_k(\mathcal{D}))$, such as the location of just the face of a person. We can then use a proposal of the form

$$q(\mathbf{x}'|\mathbf{x}, \mathcal{D}) = \pi_0 q_0(\mathbf{x}'|\mathbf{x}) + \sum_k \pi_k q_k(x'_k|f_k(\mathcal{D})) \tag{24.56}$$

where q_0 is a standard data-independent proposal (e.g., random walk), and q_k updates the k'th component of the state space. For added efficiency, the discriminative proposals should suggest joint changes to multiple variables, but this is often hard to do.

The overall procedure is a form of **generate and test**: the discriminative proposals $q(\mathbf{x}'|\mathbf{x})$ generate new hypotheses, which are then "tested" by computing the posterior ratio $\frac{p(\mathbf{x}'|\mathcal{D})}{p(\mathbf{x}|\mathcal{D})}$, to see if the new hypothesis is better or worse. By adding an annealing step, one can modify the algorithm to find posterior modes; this is called **simulated annealing**, and is described in Section 24.6.1. One advantage of using the mode-seeking version of the algorithm is that we do not need to ensure the proposal distribution is reversible.

24.3.4 Adaptive MCMC

One can change the parameters of the proposal as the algorithm is running to increase efficiency. This is called **adaptive MCMC**. This allows one to start with a broad covariance (say), allowing large moves through the space until a mode is found, followed by a narrowing of the covariance to ensure careful exploration of the region around the mode.

However, one must be careful not to violate the Markov property; thus the parameters of the proposal should not depend on the entire history of the chain. It turns out that a sufficient condition to ensure this is that the adaption is "faded out" gradually over time. See e.g., (Andrieu and Thoms 2008) for details.

24.3.5 Initialization and mode hopping

It is necessary to start MCMC in an initial state that has non-zero probability. If the model has deterministic constraints, finding such a legal configuration may be a hard problem in itself. It is therefore common to initialize MCMC methods at a local mode, found using an optimizer.

In some domains (especially with discrete state spaces), it is a more effective use of computation time to perform multiple restarts of an optimizer, and to average over these modes, rather than exploring similar points around a local mode. However, in continuous state spaces, the mode contains negligible volume (Section 5.2.1.3), so it is necessary to locally explore around each mode, in order to visit enough posterior probability mass.

24.3.6 Why MH works *

To prove that the MH procedure generates samples from p^*, we have to use a bit of Markov chain theory, so be sure to read Section 17.2.3 first.

The MH algorithm defines a Markov chain with the following transition matrix:

$$p(\mathbf{x}'|\mathbf{x}) = \begin{cases} q(\mathbf{x}'|\mathbf{x})r(\mathbf{x}'|\mathbf{x}) & \text{if } \mathbf{x}' \neq \mathbf{x} \\ q(\mathbf{x}|\mathbf{x}) + \sum_{\mathbf{x}' \neq \mathbf{x}} q(\mathbf{x}'|\mathbf{x})(1 - r(\mathbf{x}'|\mathbf{x})) & \text{otherwise} \end{cases} \tag{24.57}$$

This follows from a case analysis: if you move to $\mathbf{x}'$ from $\mathbf{x}$, you must have proposed it (with probability $q(\mathbf{x}'|\mathbf{x})$) and it must have been accepted (with probability $r(\mathbf{x}'|\mathbf{x})$); otherwise you stay in state $\mathbf{x}$, either because that is what you proposed (with probability $q(\mathbf{x}|\mathbf{x})$), or because you proposed something else (with probability $q(\mathbf{x}'|\mathbf{x})$) but it was rejected (with probability $1 - r(\mathbf{x}'|\mathbf{x})$).

Let us analyse this Markov chain. Recall from Section 17.2.3.4 that a chain satisfies **detailed balance** if

$$p(\mathbf{x}'|\mathbf{x})p^*(\mathbf{x}) = p(\mathbf{x}|\mathbf{x}')p^*(\mathbf{x}') \tag{24.58}$$

We also showed that if a chain satisfies detailed balance, then p^* is its stationary distribution. Our goal is to show that the MH algorithm defines a transition function that satisfies detailed balance and hence that p^* is its stationary distribution. (If Equation 24.58 holds, we say that p^* is an **invariant** distribution wrt the Markov transition kernel q.)

Theorem 24.3.1. *If the transition matrix defined by the MH algorithm (given by Equation 24.57) is ergodic and irreducible, then p^* is its unique limiting distribution.*

Proof. Consider two states $\mathbf{x}$ and $\mathbf{x}'$. Either

$$p^*(\mathbf{x})q(\mathbf{x}'|\mathbf{x}) < p^*(\mathbf{x}')q(\mathbf{x}|\mathbf{x}') \tag{24.59}$$

or

$$p^*(\mathbf{x})q(\mathbf{x}'|\mathbf{x}) > p^*(\mathbf{x}')q(\mathbf{x}|\mathbf{x}') \tag{24.60}$$

We will ignore ties (which occur with probability zero for continuous distributions). Without loss of generality, assume that $p^*(\mathbf{x})q(\mathbf{x}'|\mathbf{x}) > p^*(\mathbf{x}')q(\mathbf{x}|\mathbf{x}')$. Hence

$$\alpha(\mathbf{x}'|\mathbf{x}) = \frac{p^*(\mathbf{x}')q(\mathbf{x}|\mathbf{x}')}{p^*(\mathbf{x})q(\mathbf{x}'|\mathbf{x})} < 1 \tag{24.61}$$

Hence we have $r(\mathbf{x}'|\mathbf{x}) = \alpha(\mathbf{x}'|\mathbf{x})$ and $r(\mathbf{x}|\mathbf{x}') = 1$.

Now to move from $\mathbf{x}$ to $\mathbf{x}'$ we must first propose $\mathbf{x}'$ and then accept it. Hence

$$p(\mathbf{x}'|\mathbf{x}) = q(\mathbf{x}'|\mathbf{x})r(\mathbf{x}'|\mathbf{x}) = q(\mathbf{x}'|\mathbf{x})\frac{p^*(\mathbf{x}')q(\mathbf{x}|\mathbf{x}')}{p^*(\mathbf{x})q(\mathbf{x}'|\mathbf{x})} = \frac{p^*(\mathbf{x}')}{p^*(\mathbf{x})}q(\mathbf{x}|\mathbf{x}') \tag{24.62}$$

Hence

$$p^*(\mathbf{x})p(\mathbf{x}'|\mathbf{x}) = p^*(\mathbf{x}')q(\mathbf{x}|\mathbf{x}') \tag{24.63}$$

The backwards probability is

$$p(\mathbf{x}|\mathbf{x}') = q(\mathbf{x}|\mathbf{x}')r(\mathbf{x}|\mathbf{x}') = q(\mathbf{x}|\mathbf{x}') \tag{24.64}$$

since $r(\mathbf{x}|\mathbf{x}') = 1$. Inserting this into Equation 24.63 we get

$$p^*(\mathbf{x})p(\mathbf{x}'|\mathbf{x}) = p^*(\mathbf{x}')p(\mathbf{x}|\mathbf{x}') \tag{24.65}$$

so detailed balance holds wrt p^*. Hence, from Theorem 17.2.3, p^* is a stationary distribution. Furthermore, from Theorem 17.2.2, this distribution is unique, since the chain is ergodic and irreducible. □

24.3.7 Reversible jump (trans-dimensional) MCMC *

Suppose we have a set of models with different numbers of parameters, e.g., mixture models in which the number of mixture components is unknown. Let the model be denoted by m, and let its unknowns (e.g., parameters) be denoted by $\mathbf{x}_m \in \mathcal{X}_m$ (e.g., $\mathcal{X}_m = \mathbb{R}^{n_m}$, where n_m is the dimensionality of model m). Sampling in spaces of differing dimensionality is called **trans-dimensional MCMC** (Green 2003). We could sample the model indicator $m \in \{1, \ldots, M\}$ and sample all the parameters from the product space $\prod_{m=1}^{M} \mathcal{X}_m$, but this is very inefficient. It is more parsimonious to sample in the union space $\mathcal{X} = \cup_{m=1}^{M}\{m\} \times \mathcal{X}_m$, where we only worry about parameters for the currently active model.

The difficulty with this approach arises when we move between models of different dimensionality. The trouble is that when we compute the MH acceptance ratio, we are comparing densities defined in different dimensionality spaces, which is meaningless. It is like trying to compare a sphere with a circle. The solution, proposed by (Green 1998) and known as **reversible jump MCMC** or **RJMCMC**, is to augment the low dimensional space with extra random variables so that the two spaces have a common measure.

Unfortunately, we do not have space to go into details here. Suffice it to say that the method can be made to work in theory, although it is a bit tricky in practice. If, however, the continuous parameters can be integrated out (resulting in a method called collapsed RJMCMC), much of the difficulty goes away, since we are just left with a discrete state space, where there is no need to worry about change of measure. For example, (Denison et al. 2002) includes many examples of applications of collapsed RJMCMC applied to Bayesian inference fro adaptive basis-function models. They sample basis functions from a fixed set of candidates (e.g., centered on the data points), and integrate out the other parameters analytically. This provides a Bayesian alternative to using RVMs or SVMs.

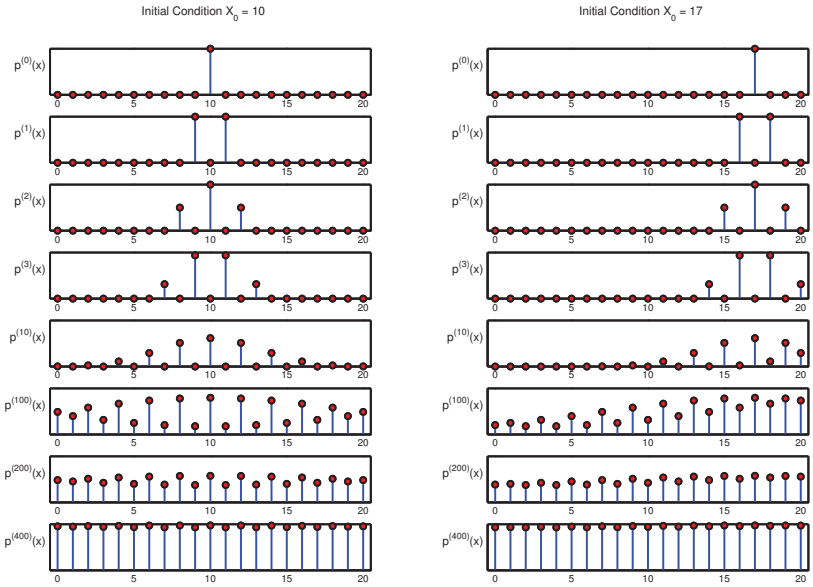


Figure 24.9 Illustration of convergence to the uniform distribution over $\{0, 1, \ldots, 20\}$ using a symmetric random walk starting from (left) state 10, and (right) state 17. Based on Figures 29.14 and 29.15 of (MacKay 2003). Figure generated by `randomWalk0to20Demo`.

24.4 Speed and accuracy of MCMC

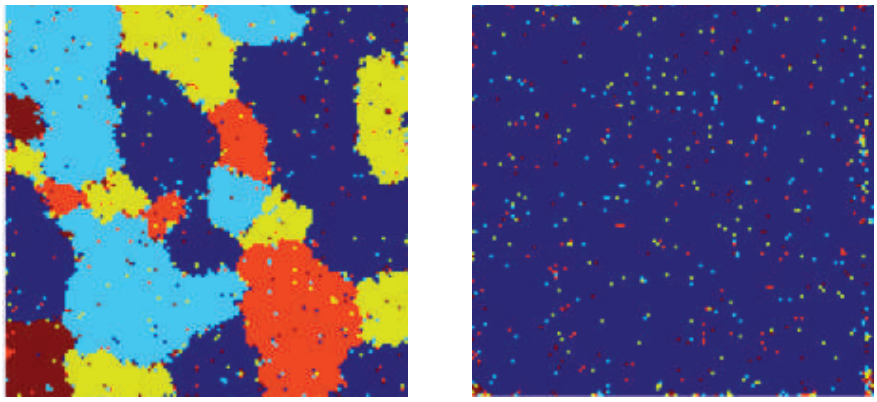
In this section, we discuss a number of important theoretical and practical issues to do with MCMC.

24.4.1 The burn-in phase

We start MCMC from an arbitrary initial state. As we explained in Section 17.2.3, only when the chain has "forgotten" where it started from will the samples be coming from the chain's stationary distribution. Samples collected before the chain has reached its stationary distribution do not come from p^*, and are usually thrown away. The initial period, whose samples will be ignored, is called the **burn-in phase**.

For example, consider a uniform distribution on the integers $\{0, 1, \ldots, 20\}$. Suppose we sample from this using a symmetric random walk. In Figure 24.9, we show two runs of the algorithm. On the left, we start in state 10; on the right, we start in state 17. Even in this small problem it takes over 100 steps until the chain has "forgotten" where it started from.

It is difficult to diagnose when the chain has burned in, an issue we discuss in more detail below. (This is one of the fundamental weaknesses of MCMC.) As an interesting example of what can happen if you start collecting samples too early, consider the Potts model. Figure 24.10(a), shows a sample after 500 iterations of Gibbs sampling. This suggests that the model likes



(a) (b)

Figure 24.10 Illustration of problems caused by poor mixing. (a) One sample from a 5-state Potts model on a 128×128 grid with 8 nearest neighbor connectivity and $J = 2/3$ (as in (Geman and Geman 1984)), after 200 iterations. (b) One sample from the same model after 10,000 iterations. Used with kind permission of Erik Sudderth.

medium-sized regions where the label is the same, implying the model would make a good prior for image segmentation. Indeed, this was suggested in the original Gibbs sampling paper (Geman and Geman 1984).

However, it turns out that if you run the chain long enough, you get isolated speckles, as in Figure 24.10(b). The results depend on the coupling strength, but in general, it is very hard to find a setting which produces nice medium-sized blobs: most parameters result in a few super-clusters, or lots of small fragments. In fact, there is a rapid phase transition between these two regimes. This led to a paper called "The Ising/Potts model is not well suited to segmentation tasks" (Morris et al. 1996). It is possible to create priors more suited to image segmentation (e.g., (Sudderth and Jordan 2008)), but the main point here is that sampling before reaching convergence can lead to erroneous conclusions.

24.4.2 Mixing rates of Markov chains *

The amount of time it takes for a Markov chain to converge to the stationary distribution, and forget its initial state, is called the **mixing time**. More formally, we say that the mixing time from state x_0 is the minimal time such that, for any constant $\epsilon > 0$, we have that

$$\tau_\epsilon(x_0) \triangleq \min\{t : ||\delta_{x_0}(x)T^t - p^*||_1 \leq \epsilon\} \tag{24.66}$$

where $\delta_{x_0}(x)$ is a distribution with all its mass in state x_0, T is the transition matrix of the chain (which depends on the target p^* and the proposal q), and $\delta_{x_0}(x)T^t$ is the distribution after t steps. The mixing time of the chain is defined as

$$\tau_\epsilon \triangleq \max_{x_0} \tau_\epsilon(x_0) \tag{24.67}$$

The mixing time is determined by the eigengap $\gamma = \lambda_1 - \lambda_2$, which is the difference of the

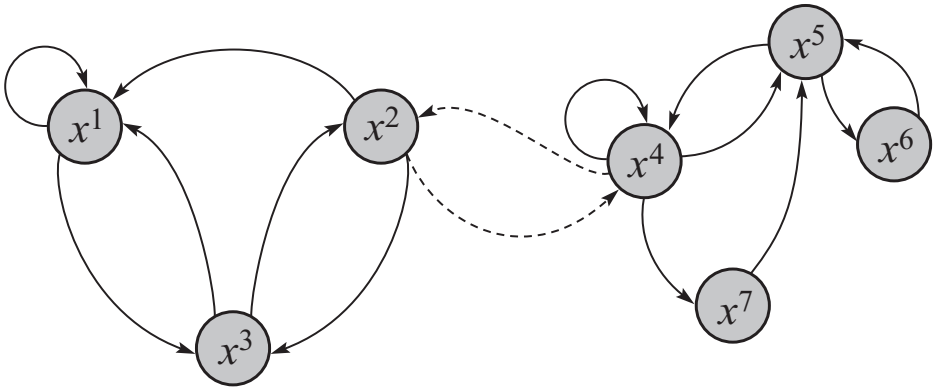


Figure 24.11 A Markov chain with low conductance. The dotted arcs represent transitions with very low probability. Source: Figure 12.6 of (Koller and Friedman 2009). Used with kind permission of Daphne Koller.

first and second eigenvalues of the transition matrix. In particular, one can show that

$$\tau_\epsilon \leq O(\frac{1}{\gamma} \log \frac{n}{\epsilon}) \tag{24.68}$$

where n is the number of states. Since computing the transition matrix can be hard to do, especially for high dimensional and/or continuous state spaces, it is useful to find other ways to estimate the mixing time.

An alternative approach is to examine the geometry of the state space. For example, consider the chain in Figure 24.11. We see that the state space consists of two "islands", each of which is connected via a narrow "bottleneck". (If they were completely disconnected, the chain would not be ergodic, and there would no longer be a unique stationary distribution.) We define the **conductance** ϕ of a chain as the minimum probability, over all subsets of states, of transitioning from that set to its complement:

$$\phi \triangleq \min_{S:0 \leq p^*(S) \leq 0.5} \frac{\sum_{x \in S, x' \in S^c} T(x \to x')}{p^*(S)}, \tag{24.69}$$

One can show that

$$\tau_\epsilon \leq O(\frac{1}{\phi^2} \log \frac{n}{\epsilon}) \tag{24.70}$$

Hence chains with low conductance have high mixing time. For example, distributions with well-separated modes usually have high mixing time. Simple MCMC methods often do not work well in such cases, and more advanced algorithms, such as parallel tempering, are necessary (see e.g., (Liu 2001)).

24.4.3 Practical convergence diagnostics

Computing the mixing time of a chain is in general quite difficult, since the transition matrix is usually very hard to compute. In practice various heuristics have been proposed to diagnose

convergence — see (Geyer 1992; Cowles and Carlin 1996; Brooks and Roberts 1998) for a review. Strictly speaking, these methods do not diagnose convergence, but rather non-convergence. That is, the method may claim the chain has converged when in fact it has not. This is a flaw common to all convergence diagnostics, since diagnosing convergence is computationally intractable in general (Bhatnagar et al. 2010).

One of the simplest approaches to assessing when the method has converged is to run multiple chains from very different **overdispersed** starting points, and to plot the samples of some variables of interest. This is called a **trace plot**. If the chain has mixed, it should have "forgotten" where it started from, so the trace plots should converge to the same distribution, and thus overlap with each other.

Figure 24.12 gives an example. We show the traceplot for x which was sampled from a mixture of two 1D Gaussians using four different methods: MH with a symmetric Gaussian proposal of variance $\sigma^2 \in \{1, 8, 500\}$, and Gibbs sampling. We see that $\sigma^2 = 1$ has not mixed, which is also evident from Figure 24.7(a), which shows that a single chain never leaves the area where it started. The results for the other methods indicate that the chains rapidly converge to the stationary distribution, no matter where they started. (The sticky nature of the $\sigma^2 = 500$ proposal is very evident. This reduces the computational efficiency, as we discuss below, but not the statistical validity.)

24.4.3.1 Estimated potential scale reduction (EPSR)

We can assess convergence more quantitatively as follows. The basic idea is to compare the variance of a quantity within each chain to its variance across chains. More precisely, suppose we collect S samples (after burn-in) from each of C chains of D variables, x_{isc}, $i = 1 : D$, $s = 1 : S$, $c = 1 : C$. Let y_{sc} be a scalar quantity of interest derived from $\mathbf{x}_{1:D,s,c}$ (e.g., $y_{sc} = x_{isc}$ for some chosen i). Define the within-sequence mean and overall mean as

$$\overline{y}_{\cdot c} \triangleq \frac{1}{S}\sum_{s=1}^{S} y_{sc}, \;\; \overline{y}_{\cdot\cdot} \triangleq \frac{1}{C}\sum_{c=1}^{C} \overline{y}_{\cdot c} \tag{24.71}$$

Define the between-sequence and within-sequence variance as

$$B \triangleq \frac{S}{C-1}\sum_{c=1}^{C}(\overline{y}_{\cdot c} - \overline{y}_{\cdot\cdot})^2, \;\; W \triangleq \frac{1}{C}\sum_{c=1}^{C}\left[\frac{1}{S-1}\sum_{s=1}^{S}(y_{sc} - \overline{y}_{\cdot c})^2\right] \tag{24.72}$$

We can now construct two estimates of the variance of y. The first estimate is W: this should underestimate $\mathrm{var}\,[y]$ if the chains have not ranged over the full posterior. The second estimate is

$$\hat{V} = \frac{S-1}{S}W + \frac{1}{S}B \tag{24.73}$$

This is an estimate of $\mathrm{var}\,[y]$ that is unbiased under stationarity, but is an overestimate if the starting points were overdispersed (Gelman and Rubin 1992). From this, we can define the following convergence diagnostic statistic, known as the **estimated potential scale reduction** or **EPSR**:

$$\hat{R} \triangleq \sqrt{\frac{\hat{V}}{W}} \tag{24.74}$$

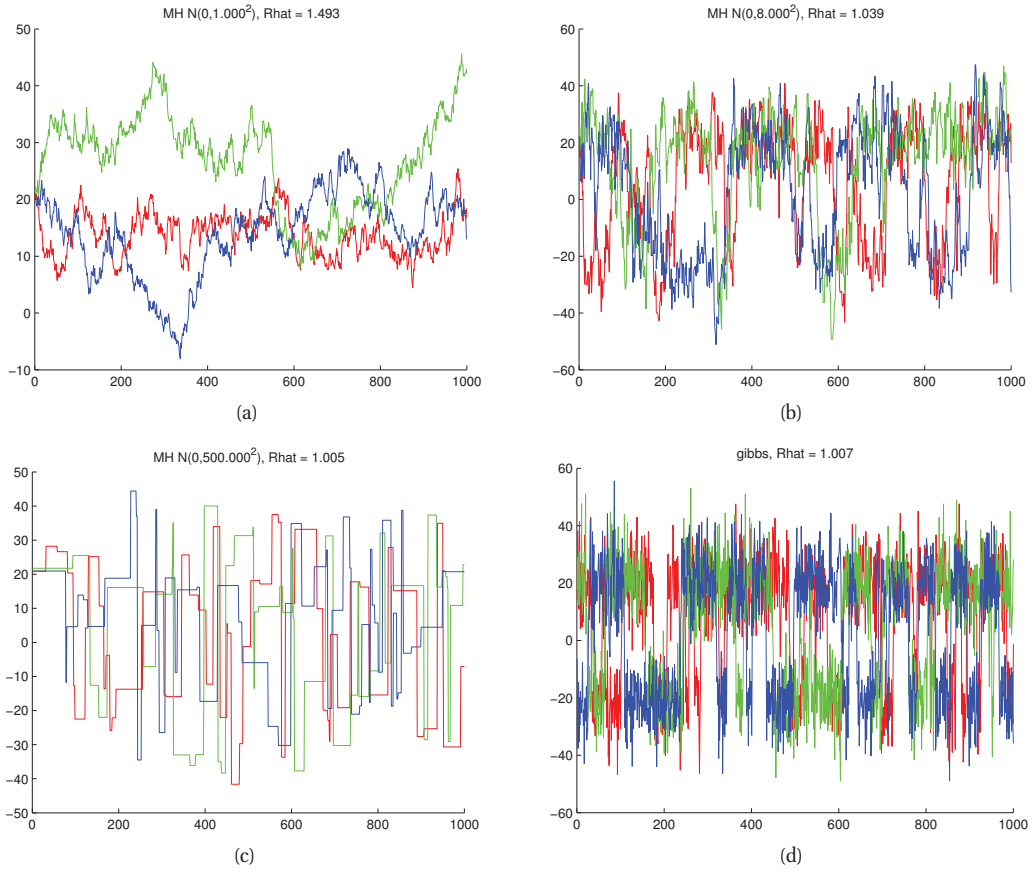


Figure 24.12 Traceplots for MCMC samplers. Each color represents the samples from a different starting point. (a-c) MH with proposal $\mathcal{N}(x'|x, \sigma^2)$ for $\sigma^2 \in \{1, 8, 500\}$, corresponding to Figure 24.7. (d) Gibbs sampling. Figure generated by `mcmcGmmDemo`.

This quantity, which was first proposed in (Gelman and Rubin 1992), measures the degree to which the posterior variance would decrease if we were to continue sampling in the $S \rightarrow \infty$ limit. If $\hat{R} \approx 1$ for any given quantity, then that estimate is reliable (or at least is not unreliable). The $\hat{R}$ values for the four samplers in Figure 24.12 are 1.493, 1.039, 1.005 and 1.007. So this diagnostic has correctly identified that the sampler using the first ($\sigma^2 = 1$) proposal is untrustworthy.

24.4.4 Accuracy of MCMC

The samples produced by MCMC are auto-correlated, and this reduces their information content relative to independent or "perfect" samples. We can quantify this as follows.[4] Suppose we want

4. This Section is based on (Hoff 2009, Sec 6.6).

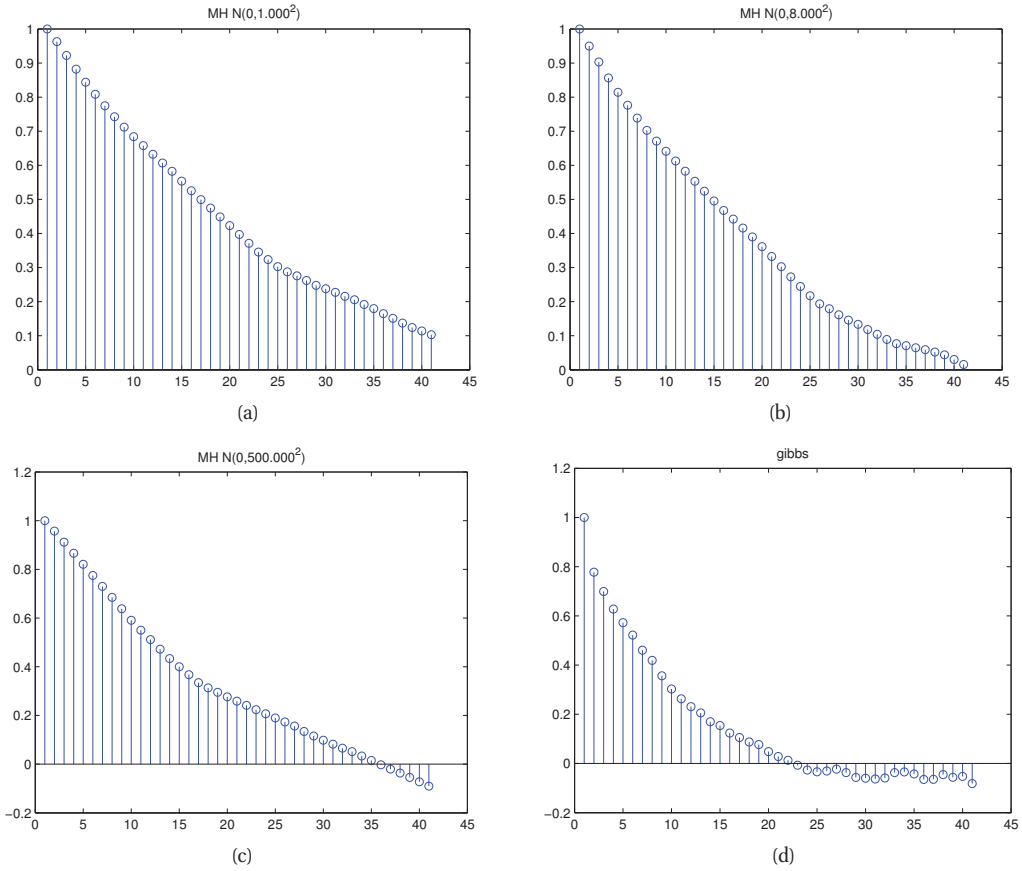


Figure 24.13 Autocorrelation functions corresponding to Figure 24.12. Figure generated by `mcmcGmmDemo`.

to estimate the mean of $f(X)$, for some function f, where $X \sim p()$. Denote the true mean by

$$f^* \triangleq \mathbb{E}\left[f(X)\right] \tag{24.75}$$

A Monte Carlo estimate is given by

$$\overline{f} = \frac{1}{S}\sum_{s=1}^{S} f_s \tag{24.76}$$

where $f_s \triangleq f(x_s)$ and $x_s \sim p(x)$. An MCMC estimate of the variance of this estimate is given by

$$\begin{aligned}
\mathrm{Var}_{MCMC}[\overline{f}] &= \mathbb{E}\left[(\overline{f} - f^*)^2\right] && (24.77)\\
&= \mathbb{E}\left[\left\{\frac{1}{S}\sum_{s=1}^{S}(f_s - f^*)\right\}^2\right] && (24.78)\\
&= \frac{1}{S^2}\mathbb{E}\left[\sum_{s=1}^{S}(f_s - f^*)^2\right] + \frac{1}{S^2}\sum_{s\neq t}\mathbb{E}\left[(f_s - f^*)(f_t - f^*)\right] && (24.79)\\
&= \mathrm{Var}_{MC}(\overline{f}) + \frac{1}{S^2}\sum_{s\neq t}\mathbb{E}\left[(f_s - f^*)(f_t - f^*)\right] && (24.80)
\end{aligned}$$

where the first term is the Monte Carlo estimate of the variance if the samples weren't correlated, and the second term depends on the correlation of the samples. We can measure this as follows. Define the sample-based auto-correlation at lag t of a set of samples $f_1, \ldots, f_S$ as follows:

$$\rho_t \triangleq \frac{\frac{1}{S-t}\sum_{s=1}^{S-t}(f_s - \overline{f})(f_{s+t} - \overline{f})}{\frac{1}{S-1}\sum_{s=1}^{S}(f_s - \overline{f})^2} \tag{24.81}$$

This is called the **autocorrelation function** (ACF). This is plotted in Figure 24.13 for our four samplers for the Gaussian mixture model. We see that the ACF of the Gibbs sampler (bottom right) dies off to 0 much more rapidly than the MH samplers, indicating that each Gibbs sample is "worth" more than each MH sample.

A simple method to reduce the autocorrelation is to use **thinning**, in which we keep every n'th sample. This does not increase the efficiency of the underlying sampler, but it does save space, since it avoids storing highly correlated samples.

We can estimate the information content of a set of samples by computing the **effective sample size** (**ESS**) S_{eff}, defined by

$$S_{\text{eff}} \triangleq \frac{\mathrm{Var}_{MC}(f)}{\mathrm{Var}_{MCMC}(\overline{f})} \tag{24.82}$$

From Figure 24.12, it is clear that the effective sample size of the Gibbs sampler is higher than that of the other samplers (in this example).

24.4.5 How many chains?

A natural question to ask is: how many chains should we run? We could either run one long chain to ensure convergence, and then collect samples spaced far apart, or we could run many short chains, but that wastes the burnin time. In practice it is common to run a medium number of chains (say 3) of medium length (say 100,000 steps), and to take samples from each after discarding the first half of the samples. If we initialize at a local mode, we may be able to use all the samples, and not wait for burn-in.

Model	Goal	Method	Reference
Probit	MAP	Gradient	Section 9.4.1
Probit	MAP	EM	Section 11.4.6
Probit	Post	EP	(Nickisch and Rasmussen 2008)
Probit	Post	Gibbs+	Exercise 24.6
Probit	Post	Gibbs with ARS	(Dellaportas and Smith 1993)
Probit	Post	MH using IRLS proposal	(Gamerman 1997)
Logit	MAP	Gradient	Section 8.3.4
Logit	Post	Gibbs+ with Student	(Fruhwirth-Schnatter and Fruhwirth 2010)
Logit	Post	Gibbs+ with KS	(Holmes and Held 2006)

Table 24.1 Summary of some possible algorithms for estimation and inference for binary classification problems using Gaussian priors. Abbreviations: Aux. = auxiliary variable sampling, ARS = adaptive rejection sampling, EP = expectation propagation, Gibbs+ = Gibbs sampling with auxiliary variables, IRLS = iterative reweighted least squares, KS = Kolmogorov Smirnov, MAP = maximum a posteriori, MH = Metropolis Hastings, Post = posterior.

24.5 Auxiliary variable MCMC *

Sometimes we can dramatically improve the efficiency of sampling by introducing dummy **auxiliary variables**, in order to reduce correlation between the original variables. If the original variables are denoted by $\mathbf{x}$, and the auxiliary variables by $\mathbf{z}$, we require that $\sum_{\mathbf{z}} p(\mathbf{x}, \mathbf{z}) = p(\mathbf{x})$, and that $p(\mathbf{x}, \mathbf{z})$ is easier to sample from than just $p(\mathbf{x})$. If we meet these two conditions, we can sample in the enlarged model, and then throw away the sampled $\mathbf{z}$ values, thereby recovering samples from $p(\mathbf{x})$. We give some examples below.

24.5.1 Auxiliary variable sampling for logistic regression

In Section 9.4.2, we discussed the latent variable interpretation of probit regression. Recall that this had the form

$$z_i \triangleq \mathbf{w}^T \mathbf{x}_i + \epsilon_i \tag{24.83}$$

$$\epsilon_i \sim \mathcal{N}(0, 1) \tag{24.84}$$

$$y_i = 1 = \mathbb{I}(z_i \geq 0) \tag{24.85}$$

We exploited this representation in Section 11.4.6, where we used EM to find an ML estimate. It is straightforward to convert this into an auxiliary variable Gibbs sampler (Exercise 24.6), since $p(\mathbf{w}|\mathcal{D})$ is Gaussian and $p(z_i|\mathbf{x}_i, y_i, \mathbf{w})$ is truncated Gaussian, both of which are easy to sample from.

Now let us discuss how to derive an auxiliary variable Gibbs sampler for logistic regression. Let ϵ_i follow a **logistic distribution**, with pdf

$$p_{\text{Logistic}}(\epsilon) = \frac{e^{-\epsilon}}{(1 + e^{-\epsilon})^2} \tag{24.86}$$

with mean $\mathbb{E}[\epsilon] = 0$ and variance $\text{var}[\epsilon] = \pi^2/3$. The cdf has the form $F(\epsilon) = \text{sigm}(\epsilon)$, which

is the logistic function. Since $y_i = 1$ iff $\mathbf{w}^T\mathbf{x}_i + \epsilon > 0$, we have, by symmetry, that

$$p(y_i = 1|\mathbf{x}_i, \mathbf{w}) = \int_{-\mathbf{w}^T\mathbf{x}_i}^{\infty} f(\epsilon)d\epsilon = \int_{-\infty}^{\mathbf{w}^T\mathbf{x}_i} f(\epsilon)d\epsilon = F(\mathbf{w}^T\mathbf{x}_i) = \text{sigm}(\mathbf{w}^T\mathbf{x}_i) \quad (24.87)$$

as required.

We can derive an auxiliary variable Gibbs sampler by sampling from $p(\mathbf{z}|\mathbf{w}, \mathcal{D})$ and $p(\mathbf{w}|\mathbf{z}, \mathcal{D})$. Unfortunately, sampling directly from $p(\mathbf{w}|\mathbf{z}, \mathcal{D})$ is not possible. One approach is to define $\epsilon_i \sim \mathcal{N}(0, \lambda_i)$, where $\lambda_i = (2\psi_i)^2$ and $\psi_i \sim$ KS, the Kolmogorov Smirnov distribution, and then to sample $\mathbf{w}$, $\mathbf{z}$, $\boldsymbol{\lambda}$ and $\boldsymbol{\psi}$ (Holmes and Held 2006).

A simpler approach is to approximate the logistic distribution by the Student distribution (Albert and Chib 1993). Specifically, we will make the approximation $\epsilon_i \sim \mathcal{T}(0, 1, \nu)$, where $\nu \approx 8$. We can now use the scale mixture of Gaussians representation of the Student to simplify inference. In particular, we write

$$\begin{aligned} \lambda_i &\sim \text{Ga}(\nu/2, \nu/2) && (24.88)\\ \epsilon_i &\sim \mathcal{N}(0, \lambda_i^{-1}) && (24.89)\\ z_i &\triangleq \mathbf{w}^T\mathbf{x}_i + \epsilon_i && (24.90)\\ y_i = 1|z_i &= \mathbb{I}(z_i \geq 0) && (24.91) \end{aligned}$$

All of the full conditionals now have a simple form; see Exercise 24.7 for the details.

Note that if we set $\nu = 1$, then $z_i \sim \mathcal{N}(\mathbf{w}^T\mathbf{x}_i, 1)$, which is equivalent to probit regression (see Section 9.4). Rather than choosing between probit or logit regression, we can simply estimate the ν parameter. There is no convenient conjugate prior, but we can consider a finite range of possible values and evaluate the posterior as follows:

$$p(\nu|\boldsymbol{\lambda}) \propto p(\nu) \prod_{i=1}^{N} \frac{1}{\Gamma(\nu/2)(\nu/2)^{\nu/2}} \lambda_i^{\nu/2-1} e^{-\nu\lambda_i/2} \quad (24.92)$$

Furthermore, if we define $\mathbf{V}_0 = v_0\mathbf{I}$, we can sample v_0 as well. For example, suppose we use a $\text{IG}(\delta_1, \delta_2)$ prior for v_0. The posterior is given by $p(v_0|\mathbf{w}) = \text{IG}(\delta_1 + \frac{1}{2}D, \delta_2 + \frac{1}{2}\sum_{j=1}^{D} w_j^2)$. This can be interleaved with the other Gibbs sampling steps, and provides an appealing Bayesian alternative to cross validation for setting the strength of the regularizer.

See Table 24.1 for a summary of various algorithms for fitting probit and logit models. Many of these methods can also be extended to the multinomial logistic regression case. For details, see (Scott 2009; Fruhwirth-Schnatter and Fruhwirth 2010).

24.5.2 Slice sampling

Consider sampling from a univariate, but multimodal, distribution $\tilde{p}(x)$. We can sometimes improve the ability to make large moves by adding an auxiliary variable u. We define the joint distribution as follows:

$$\hat{p}(x, u) = \begin{cases} 1/Z_p & \text{if } 0 \leq u \leq \tilde{p}(x) \\ 0 & \text{otherwise} \end{cases} \quad (24.93)$$

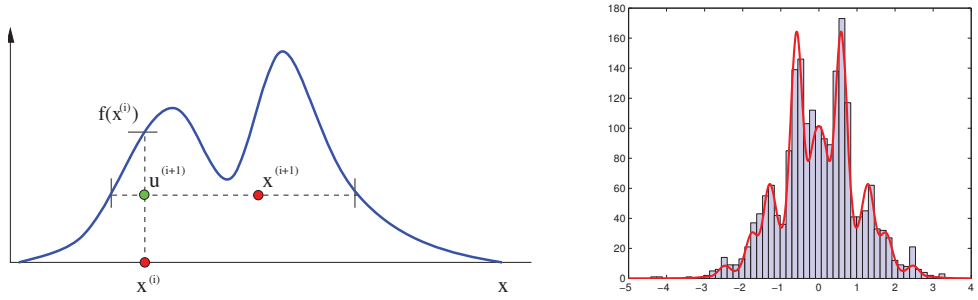


Figure 24.14 (a) Illustration of the principle behind slice sampling. Given a previous sample x^i, we sample u^{i+1} uniformly on $[0, f(x^i)]$, where f is the target density. We then sample x^{i+1} along the slice where $f(x) \geq u^{i+1}$. Source: Figure 15 of (Andrieu et al. 2003) . Used with kind permission of Nando de Freitas. (b) Slice sampling in action. Figure generated by `sliceSamplingDemo1d`.

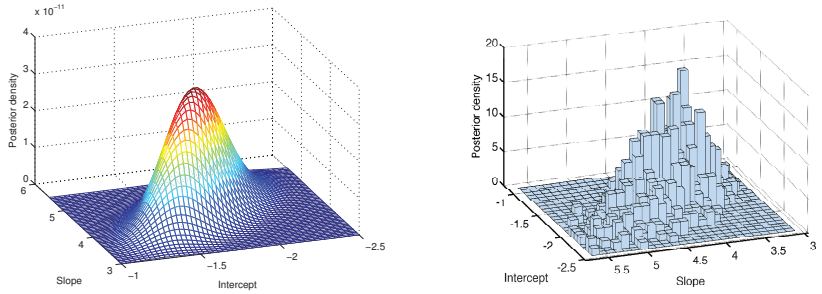


Figure 24.15 Binomial regression for 1d data. (a) Grid approximation to posterior. (b) Slice sampling approximation. Figure generated by `sliceSamplingDemo2d`.

where $Z_p = \int \tilde{p}(x)dx$. The marginal distribution over x is given by

$$\int \hat{p}(x,u)du = \int_0^{\tilde{p}(x)} \frac{1}{Z_p} du = \frac{\tilde{p}(x)}{Z_p} = p(x) \tag{24.94}$$

so we can sample from $p(x)$ by sampling from $\hat{p}(x,u)$ and then ignoring u. The full conditionals have the form

$$p(u|x) = U_{[0,\tilde{p}(x)]}(u) \tag{24.95}$$

$$p(x|u) = U_A(x) \tag{24.96}$$

where $A = \{x : \tilde{p}(x) \geq u\}$ is the set of points on or above the chosen height u. This corresponds to a slice through the distribution, hence the term **slice sampling** (Neal 2003a). See Figure 24.14(a).

In practice, it can be difficult to identify the set A. So we can use the following approach: construct an interval $x_{min} \leq x \leq x_{max}$ around the current point x^s of some width. We then

test to see if each end point lies within the slice. If it does, we keep extending in that direction until it lies outside the slice. This is called **stepping out**. A candidate value x' is then chosen uniformly from this region. If it lies within the slice, it is kept, so $x^{s+1} = x'$. Otherwise we shrink the region such that x' forms one end and such that the region still contains x^s. Then another sample is drawn. We continue in this way until a sample is accepted.

To apply the method to multivariate distributions, we can sample one extra auxiliary variable for each dimension. The advantage of slice sampling over Gibbs is that it does not need a specification of the full-conditionals, just the unnormalized joint. The advantage of slice sampling over MH is that it does not need a user-specified proposal distribution (although it does require a specification of the width of the stepping out interval).

Figure 24.14(b) illustrates the algorithm in action on a synthetic 1d problem. Figure 24.15 illustrates its behavior on a slightly harder problem, namely binomial logistic regression. The model has the form

$$y_i \sim \text{Bin}(n_i, \text{logit}(\beta_1 + \beta_2 x_i)) \tag{24.97}$$

We use a vague Gaussian prior for the β_j's. Figure 24.15(a) shows a grid-based approximation to the posterior, and Figure 24.15(b) shows a sample-based approximation. In this example, the grid is faster to compute, but for any problem with more than 2 dimensions, the grid approach is infeasible.

24.5.3 Swendsen Wang

Consider an Ising model of the following form:

$$p(\mathbf{x}) = \frac{1}{Z} \prod_e f_e(\mathbf{x}_e) \tag{24.98}$$

where $\mathbf{x}_e = (x_i, x_j)$ for edge $e = (i, j)$, $x_i \in \{+1, -1\}$, and the edge factor f_e is defined by $\begin{pmatrix} e^J & e^{-J} \\ e^{-J} & e^J \end{pmatrix}$, where J is the edge strength. Gibbs sampling in such models can be slow when J is large in absolute value, because neighboring states can be highly correlated. The **Swendsen Wang** algorithm (Swendsen and Wang 1987) is a auxiliary variable MCMC sampler which mixes much faster, at least for the case of attractive or ferromagnetic models, with $J > 0$.

Suppose we introduce auxiliary binary variables, one per edge. [5] These are called **bond variables**, and will be denoted by $\mathbf{z}$. We then define an extended model $p(\mathbf{x}, \mathbf{z})$ of the form

$$p(\mathbf{x}, \mathbf{z}) = \frac{1}{Z'} \prod_e g_e(\mathbf{x}_e, z_e) \tag{24.99}$$

where $z_e \in \{0, 1\}$, and we define the new factor as follows: $g_e(\mathbf{x}_e, z_e = 0) = \begin{pmatrix} e^{-J} & e^{-J} \\ e^{-J} & e^{-J} \end{pmatrix}$, and $g_e(\mathbf{x}_e, z_e = 1) = \begin{pmatrix} e^J - e^{-J} & 0 \\ 0 & e^J - e^{-J} \end{pmatrix}$. It is clear that $\sum_{z_e=0}^{1} g_e(\mathbf{x}_e, z_e) = f_e(\mathbf{x}_e)$,

5. Our presentation of the method is based on some notes by David Mackay, available from `http://www.inference.phy.cam.ac.uk/mackay/itila/swendsen.pdf`.

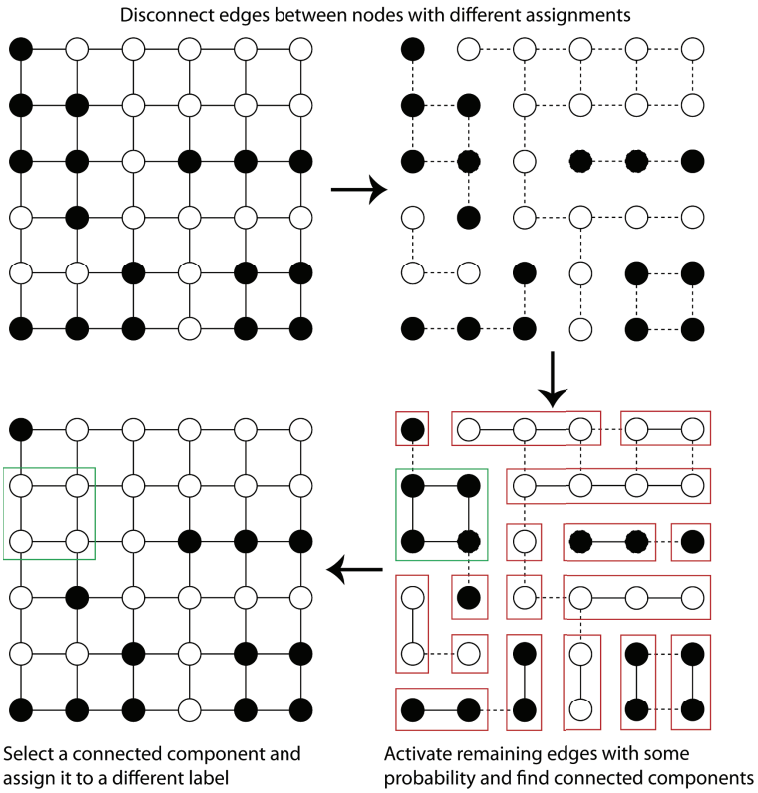


Figure 24.16 Illustration of the Swendsen Wang algorithm on a 2d grid. Used with kind permission of Kevin Tang.

and hence that $\sum_{\mathbf{z}} p(\mathbf{x}, \mathbf{z}) = p(\mathbf{x})$. So if we can sample from this extended model, we can just throw away the $\mathbf{z}$ samples and get valid $\mathbf{x}$ samples from the original distribution.

Fortunately, it is easy to apply Gibbs sampling to this extended model. The full conditional $p(\mathbf{z}|\mathbf{x})$ factorizes over the edges, since the bond variables are conditionally independent given the node variables. Furthermore, the full conditional $p(z_e|\mathbf{x}_e)$ is simple to compute: if the nodes on either end of the edge are in the same state ($x_i = x_j$), we set the bond z_e to 1 with probability $p = 1 - e^{-2J}$, otherwise we set it to 0. In Figure 24.16 (top right), the bonds that could be turned on (because their corresponding nodes are in the same state) are represented by dotted edges. In Figure 24.16 (bottom right), the bonds that are randomly turned on are represented by solid edges.

To sample $p(\mathbf{x}|\mathbf{z})$, we proceed as follows. Find the connected components defined by the graph induced by the bonds that are turned on. (Note that a connected component may consist of a singleton node.) Pick one of these components uniformly at random. All the nodes in each such component must have the same state, since the off-diagonal terms in the $g_e(\mathbf{x}_e, z_e = 1)$ factor are 0. Pick a state ± 1 uniformly at random, and force all the variables in this component to adopt this new state. This is illustrated in Figure 24.16 (bottom left), where the green square

denotes the selected connected component, and we choose to force all nodes within in to enter the white state.

The validity of this algorithm is left as an exercise, as is the extension to handle local evidence and non-stationary potentials.

It should be intuitively clear that Swendsen Wang makes much larger moves through the state space than Gibbs sampling. In fact, SW mixes much faster than Gibbs sampling on 2d lattice Ising models for a variety of values of the coupling parameter, provided $J > 0$. More precisely, let the edge strength be parameterized by J/T, where $T > 0$ is a computational temperature. For large T, the nodes are roughly independent, so both methods work equally well. However, as T approaches a **critical temperature** T_c, the typical states of the system have very long correlation lengths, and Gibbs sampling takes a very long time to generate independent samples. As the temperature continues to drop, the typical states are either all on or all off. The frequency with which Gibbs sampling moves between these two modes is exponentiall small. By contrast, SW mixes rapidly at all temperatures.

Unfortunately, if any of the edge weights are negative, $J < 0$, the system is **frustrated**, and there are exponentially many modes, even at low temperature. SW does not work very well in this setting, since it tries to force many neighboring variables to have the same state. In fact, computation in this regime is provably hard for any algorithm (Jerrum and Sinclair 1993, 1996).

24.5.4 Hybrid/Hamiltonian MCMC *

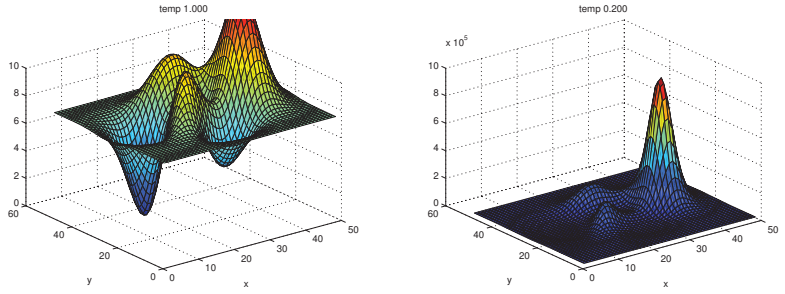
In this section, we briefly mention a way to perform MCMC sampling for continuous state spaces, for which we can compute the gradient of the (unnormalized) log-posterior. This is the case in neural network models, for example.

The basic idea is to think of the parameters as a particle in space, and to create auxiliary variables which represent the "momentum" of this particle. We then update this parameter/ momentum pair according to certain rules (see e.g., (Duane et al. 1987; Neal 1993; MacKay 2003; Neal 2010) for details). The resulting method is called **hybrid MCMC** or **Hamiltonian MCMC**. The two main parameters that the user must specify are how many **leapfrog steps** to take when updating the position/ momentum, and how big to make these steps. Performance can be quite sensitive to these parameters (although see (Hoffman and Gelman 2011) for a recent way to set them automatically). This method can be combined with stochastic gradient descent (Section 8.5.2) in order to handle large datasets, as explained in (Ahn et al. 2012).

Recently, a more powerful extension of this method has been developed, that exploits second-order gradient information. See (Girolami et al. 2010) for details.

24.6 Annealing methods

Many distributions are multimodal and hence hard to sample from. However, by analogy to the way metals are heated up and then cooled down in order to make the molecules align, we can imagine using a computational temperature parameter to smooth out a distribution, gradually cooling it to recover the original "bumpy" distribution. We first explain this idea in more detail in the context of an algorithm for MAP estimation. We then discuss extensions to the sampling case.



(a) (b)

Figure 24.17 An energy surface at different temperatures. Note the different vertical scales. (a) $T = 1$. (b) $T = 0.5$. Figure generated by `saDemoPeaks`.

24.6.1 Simulated annealing

Simulated annealing (Kirkpatrick et al. 1983) is a stochastic algorithm that attempts to find the global optimum of a black-box function $f(\mathbf{x})$. It is closely related to the Metropolis-Hastings algorithm for generating samples from a probability distribution, which we discussed in Section 24.3. SA can be used for both discrete and continuous optimization.

The method is inspired by statistical physics. The key quantity is the **Boltzmann distribution**, which specifies that the probability of being in any particular state $\mathbf{x}$ is given by

$$p(\mathbf{x}) \propto \exp(-f(\mathbf{x})/T) \tag{24.100}$$

where $f(\mathbf{x})$ is the "energy" of the system and T is the computational temperature. As the temperature approaches 0 (so the system is cooled), the system spends more and more time in its minimum energy (most probable) state.

Figure 24.17 gives an example of a 2d function at different temperatures. At high temperatures, $T \gg 1$, the surface is approximately flat, and hence it is easy to move around (i.e., to avoid local optima). As the temperature cools, the largest peaks become larger, and the smallest peaks disappear. By cooling slowly enough, it is possible to "track" the largest peak, and thus find the global optimum. This is an example of a **continuation method**.

We can generate an algorithm from this as follows. At each step, sample a new state according to some proposal distribution $\mathbf{x}' \sim q(\cdot|\mathbf{x}_k)$. For real-valued parameters, this is often simply a random walk proposal, $\mathbf{x}' = \mathbf{x}_k + \boldsymbol{\epsilon}_k$, where $\boldsymbol{\epsilon}_k \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$. For discrete optimization, other kinds of local moves must be defined.

Having proposed a new state, we compute

$$\alpha = \exp\left((f(\mathbf{x}) - f(\mathbf{x}'))/T\right) \tag{24.101}$$

We then accept the new state (i.e., set $\mathbf{x}_{k+1} = \mathbf{x}'$) with probability $\min(1, \alpha)$, otherwise we stay in the current state (i.e., set $\mathbf{x}_{k+1} = \mathbf{x}_k$). This means that if the new state has lower energy (is more probable), we will definitely accept it, but it it has higher energy (is less probable), we might still accept, depending on the current temperature. Thus the algorithm allows "down-hill" moves in probability space (up-hill in energy space), but less frequently as the temperature drops.

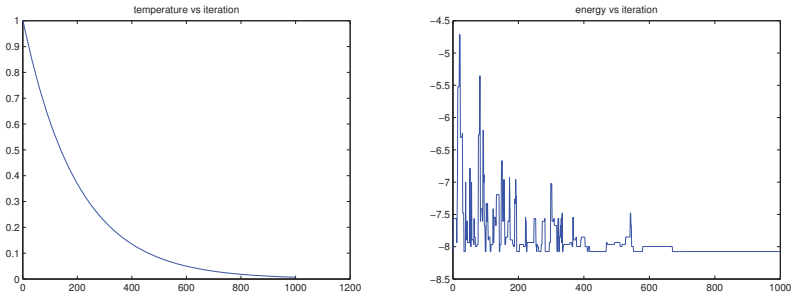


Figure 24.18 A run of simulated annealing on the energy surface in Figure 24.17. (a) Temperature vs iteration. (b) Energy vs iteration. Figure generated by `saDemoPeaks`.

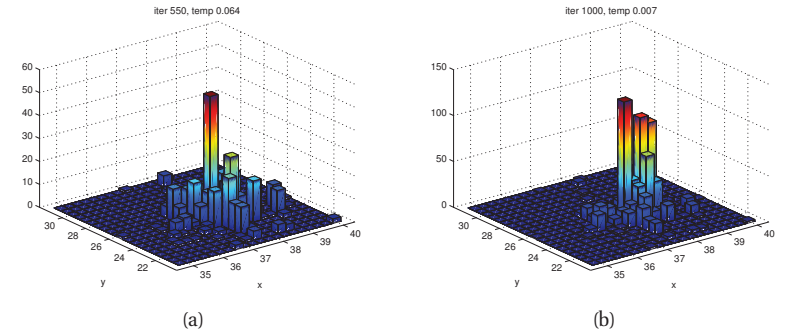


Figure 24.19 Histogram of samples from the annealed "posterior" at 2 different time points produced by simulated annealing on the energy surface shown in Figure 24.17. Note that at cold temperatures, most of the samples are concentrated near the peak at (38,25). Figure generated by `saDemoPeaks`.

The rate at which the temperature changes over time is called the **cooling schedule**. It has been shown (Kirkpatrick et al. 1983) that if one cools sufficiently slowly, the algorithm will provably find the global optimum. However, it is not clear what "sufficient slowly" means. In practice it is common to use an **exponential cooling schedule** of the following form: $T_k = T_0 C^k$, where T_0 is the initial temperature (often $T_0 \sim 1$) and C is the cooling rate (often $C \sim 0.8$). See Figure 24.18(a) for a plot of this cooling schedule. Cooling too quickly means one can get stuck in a local maximum, but cooling too slowly just wastes time. The best cooling schedule is difficult to determine; this is one of the main drawbacks of simulated annealing.

Figure 24.18(b) shows an example of simulated annealing applied to the function in Figure 24.17 using a random walk proposal. We see that the method stochastically reduces the energy over time. Figures 24.19 illustrate (a histogram of) samples drawn from the cooled probability distribution over time. We see that most of the samples are concentrated near the global maximum. When the algorithm has converged, we just return the largest value found.

24.6.2 Annealed importance sampling

We now describe a method known as **annealed importance sampling** (Neal 2001) that combines ideas from simulated annealing and importance sampling in order to draw independent samples from difficult (e.g., multimodal) distributions.

Suppose we want to sample from $p_0(\mathbf{x}) \propto f_0(\mathbf{x})$, but we cannot do so easily; for example, this might represent a multimodal posterior. Suppose however that there is an easier distribution which we can sample from, call it $p_n(\mathbf{x}) \propto f_n(\mathbf{x})$; for example, this might be the prior. We can now construct a sequence of intermediate distributions than move slowly from p_n to p_0 as follows:

$$f_j(\mathbf{x}) = f_0(\mathbf{x})^{\beta_j} f_n(\mathbf{x})^{1-\beta_j} \tag{24.102}$$

where $1 = \beta_0 > \beta_1 > \cdots > \beta_n = 0$, where β_j is an inverse temperature. (Contrast this to the scheme used by simulated annealing which has the form $f_j(\mathbf{x}) = f_0(\mathbf{x})^{\beta_j}$; this makes it hard to sample from p_n.) Furthermore, suppose we have a series of Markov chains $T_j(\mathbf{x}, \mathbf{x}')$ (from $\mathbf{x}$ to $\mathbf{x}'$) which leave each p_j invariant. Given this, we can sample $\mathbf{x}$ from p_0 by first sampling a sequence $\mathbf{z} = (\mathbf{z}_{n-1}, \ldots, \mathbf{z}_0)$ as follows: sample $\mathbf{z}_{n-1} \sim p_n$; sample $\mathbf{z}_{n-2} \sim T_{n-1}(\mathbf{z}_{n-1}, \cdot)$; ...; sample $\mathbf{z}_0 \sim T_1(\mathbf{z}_1, \cdot)$. Finally we set $\mathbf{x} = \mathbf{z}_0$ and give it weight

$$w = \frac{f_{n-1}(\mathbf{z}_{n-1})}{f_n(\mathbf{z}_{n-1})} \frac{f_{n-2}(\mathbf{z}_{n-2})}{f_{n-1}(\mathbf{z}_{n-2})} \cdots \frac{f_1(\mathbf{z}_1)}{f_2(\mathbf{z}_1)} \frac{f_0(\mathbf{z}_0)}{f_1(\mathbf{z}_0)} \tag{24.103}$$

This can be shown to be correct by viewing the algorithm as a form of importance sampling in an extended state space $\mathbf{z} = (\mathbf{z}_0, \ldots, \mathbf{z}_{n-1})$. Consider the following distribution on this state space:

$$p(\mathbf{z}) \propto f(\mathbf{z}) = f_0(\mathbf{z}_0)\tilde{T}_1(\mathbf{z}_0, \mathbf{z}_1)\tilde{T}_2(\mathbf{z}_1, \mathbf{z}_2) \cdots \tilde{T}_{n-1}(\mathbf{z}_{n-2}, \mathbf{z}_{n-1}) \tag{24.104}$$

where $\tilde{T}_j$ is the reversal of T_j:

$$\tilde{T}_j(\mathbf{z}, \mathbf{z}') = T_j(\mathbf{z}', \mathbf{z}) p_j(\mathbf{z}')/p_j(\mathbf{z}) = T_j(\mathbf{z}', \mathbf{z}) f_j(\mathbf{z}')/f_j(\mathbf{z}) \tag{24.105}$$

It is clear that $\sum_{\mathbf{z}_1, \ldots, \mathbf{z}_{n-1}} f(\mathbf{z}) = f_0(\mathbf{z}_0)$, so we can safely just use the $\mathbf{z}_0$ part of these sequences to recover the original ditribution.

Now consider the proposal distribution defined by the algorithm:

$$q(\mathbf{z}) \propto g(\mathbf{z}) = f_n(\mathbf{z}_{n-1}) T_{n-1}(\mathbf{z}_{n-1}, \mathbf{z}_{n-2}) \cdots T_2(\mathbf{z}_2, \mathbf{z}_1) T_1(\mathbf{z}_1, \mathbf{z}_0) \tag{24.106}$$

One can show that the importance weights $w = \frac{f(\mathbf{z}_0, \ldots, \mathbf{z}_{n-1})}{g(\mathbf{z}_0, \ldots, \mathbf{z}_{n-1})}$ are given by Equation 24.103.

24.6.3 Parallel tempering

Another way to combine MCMC and annealing is to run multiple chains in parallel at different temperatures, and allow one chain to sample from another chain at a neighboring temperature. In this way, the high temperature chain can make long distance moves through the state space, and have this influence lower temperature chains. This is known as **parallel tempering**. See e.g., (Earl and Deem 2005) for details.

24.7 Approximating the marginal likelihood

The marginal likelihood $p(\mathcal{D}|M)$ is a key quantity for Bayesian model selection, and is given by

$$p(\mathcal{D}|M) = \int p(\mathcal{D}|\boldsymbol{\theta}, M)p(\boldsymbol{\theta}|M)d\boldsymbol{\theta} \tag{24.107}$$

Unfortunately, this integral is often intractable to compute, for example if we have non conjugate priors, and/or we have hidden variables. In this section, we briefly discuss some ways to approximate this expression using Monte Carlo. See (Gelman and Meng 1998) for a more extensive review.

24.7.1 The candidate method

There is a simple method for approximating the marginal likelihood known as the **Candidate method** (Chib 1995). This exploits the following identity:

$$p(\mathcal{D}|M) = \frac{p(\mathcal{D}|\boldsymbol{\theta}, M)p(\boldsymbol{\theta}|M)}{p(\boldsymbol{\theta}|\mathcal{D}, M)} \tag{24.108}$$

This holds for any value of $\boldsymbol{\theta}$. Once we have picked some value, we can evaluate $p(\mathcal{D}|\boldsymbol{\theta}, M)$ and $p(\boldsymbol{\theta}|M)$ quite easily. If we have some estimate of the posterior near $\boldsymbol{\theta}$, we can then evaluate the denominator as well. This posterior is often approximated using MCMC.

The flaw with this method is that it relies on the assumption that $p(\boldsymbol{\theta}|\mathcal{D}, M)$ has marginalized over all the modes of the posterior, which in practice is rarely possible. Consequently the method can give very inaccurate results in practice (Neal 1998).

24.7.2 Harmonic mean estimate

Newton and Raftery (1994) proposed a simple method for approximating $p(\mathcal{D})$ using the output of MCMC, as follows:

$$1/p(\mathcal{D}) \approx \frac{1}{S}\sum_{s=1}^{S}\frac{1}{p(\mathcal{D}|\boldsymbol{\theta}^s)} \tag{24.109}$$

where $\boldsymbol{\theta}^s \sim p(\boldsymbol{\theta}|\mathcal{D})$. This expression is the harmonic mean of the likelihood of the data under each sample. The theoretical correctness of this expression follows from the following identity:

$$\int \frac{1}{p(\mathcal{D}|\boldsymbol{\theta})}p(\boldsymbol{\theta}|\mathcal{D})d\boldsymbol{\theta} = \int \frac{1}{p(\mathcal{D}|\boldsymbol{\theta})}\frac{p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\mathcal{D})}d\boldsymbol{\theta} = \frac{1}{p(\mathcal{D})}\int p(\boldsymbol{\theta}|\mathcal{D})d\boldsymbol{\theta} = \frac{1}{p(\mathcal{D})} \tag{24.110}$$

Unfortunately, in practice this method works very poorly. Indeed, Radford Neal called this "the worst Monte Carlo method ever".[6]. The reason it is so bad is that it depends only on samples drawn from the posterior. But the posterior is often very insensitive to the prior, whereas the marginal likelihood is not. We only mention this method in order to warn against its use. We present a better method below.

6. Source: radfordneal.wordpress.com/2008/08/17/the-harmonic-mean-of-the-likelihood-worst-monte-carlo-method-ever.

24.7.3 Annealed importance sampling

We can use annealed importance sampling (Section 24.6.2) to evaluate a ratio of partition functions. Notice that $Z_0 = \int f_0(\mathbf{x})d\mathbf{x} = \int f(\mathbf{z})d\mathbf{z}$, and $Z_n = \int f_n(\mathbf{x})d\mathbf{x} = \int g(\mathbf{z})d\mathbf{z}$. Hence

$$\frac{Z_0}{Z_n} = \frac{\int f(\mathbf{z})d\mathbf{z}}{\int g(\mathbf{z})d\mathbf{z}} = \frac{\int \frac{f(\mathbf{z})}{g(\mathbf{z})} g(\mathbf{z})d\mathbf{z}}{\int g(\mathbf{z})d\mathbf{z}} = \mathbb{E}_q\left[\frac{f(\mathbf{z})}{g(\mathbf{z})}\right] \approx \frac{1}{S}\sum_{s=1}^{S} w_s \tag{24.111}$$

If f_n is a prior and f_0 is the posterior, we can estimate $Z_n = p(\mathcal{D})$ using the above equation, provided the prior has a known normalization constant Z_0. This is generally considered the method of choice for evaluating difficult partition functions.

Exercises

Exercise 24.1 Gibbs sampling from a 2D Gaussian

Suppose $\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, where $\boldsymbol{\mu} = (1, 1)$ and $\boldsymbol{\Sigma} = (1, -0.5; -0.5, 1)$. Derive the full conditionals $p(x_1|x_2)$ and $p(x_2|x_1)$. Implement the algorithm and plot the 1d marginals $p(x_1)$ and $p(x_2)$ as histograms. Superimpose a plot of the exact marginals.

Exercise 24.2 Gibbs sampling for a 1D Gaussian mixture model

Consider applying Gibbs sampling to a univariate mixture of Gaussians, as in Section 24.2.3. Derive the expressions for the full conditionals. Hint: if we know $z_n = j$ (say), then μ_j gets "connected" to x_n, but all other values of μ_i, for all $i \neq j$, are irrelevant. (This is an example of context-specific independence, where the structure of the graph simplifies once we have assigned values to some of the nodes.) Hence, given all the z_n values, the posteriors of the μ's should be independent, so the conditional of μ_j should be independent of $\boldsymbol{\mu}_{-j}$. (Similarly for σ_j.)

Exercise 24.3 Gibbs sampling from the Potts model

Modify the code in `gibbsDemoIsing` to draw samples from a Potts prior at different temperatures, as in Figure 19.8.

Exercise 24.4 Full conditionals for hierarchical model of Gaussian means

Let us reconsider the Gaussian-Gaussian model in Section 5.6.2 for modelling multiple related mean parameters θ_j. In this exercise we derive a Gibbs sampler instead of using EB. Suppose, following (Hoff 2009, p134)), that we use the following conjugate priors on the hyper-parameters:

$$\mu \sim \mathcal{N}(\mu_0, \gamma_0^2) \tag{24.112}$$

$$\tau^2 \sim \mathrm{IG}(\eta_0/2, \eta_0\tau_0^2/2) \tag{24.113}$$

$$\sigma^2 \sim \mathrm{IG}(\nu_0/2, \nu_0\sigma_0^2/2) \tag{24.114}$$

We can set $\boldsymbol{\eta} = (\mu_0, \gamma_0, \eta_0, \tau_0, \nu_0, \sigma_0)$ to uninformative values. Given this model specification, show that the full conditionals for μ, τ, σ and the θ_j are as follows:

$$p(\mu|\theta_{1:D}, \tau^2) = \mathcal{N}(\mu|\frac{D\bar{\theta}/\tau^2 + \mu_0/\gamma_0^2}{D/\tau^2 + 1/\gamma_0^2}, [D/\tau^2 + 1/\gamma_0^2]^{-1}) \quad (24.115)$$

$$p(\theta_j|\mu, \tau^2, \mathcal{D}_j, \sigma^2) = \mathcal{N}(\theta_j|\frac{N_j\overline{x}_j/\sigma^2 + 1/\tau^2}{N_j/\sigma^2 + 1/\tau^2}, [N_j/\sigma^2 + 1/\tau^2]^{-1}) \quad (24.116)$$

$$p(\tau^2|\theta_{1:D}, \mu) = \mathrm{IG}(\tau^2|\frac{\eta_0 + D}{2}, \frac{\eta_0\tau_0^2 + \sum_j(\theta_j - \mu)^2}{2}) \quad (24.117)$$

$$p(\sigma^2|\boldsymbol{\theta}_{1:D}, \mathcal{D}) = \mathrm{IG}(\sigma^2|\frac{1}{2}[\nu_0 + \sum_{j=1}^{D} N_j], \frac{1}{2}[\nu_0\sigma_0^2 + \sum_{j=1}^{D}\sum_{i=1}^{N_j}(x_{ij} - \theta_j)^2]) \quad (24.118)$$

Exercise 24.5 Gibbs sampling for robust linear regression with a Student t likelihood

Modify the EM algorithm in Exercise 11.12 to perform Gibbs sampling for $p(\mathbf{w}, \sigma^2, \mathbf{z}|\mathcal{D}, \nu)$.

Exercise 24.6 Gibbs sampling for probit regression

Modify the EM algorithm in Section 11.4.6 to perform Gibbs sampling for $p(\mathbf{w}, \mathbf{z}|\mathcal{D})$. Hint: we can sample from a truncated Gaussian, $\mathcal{N}(z|\mu, \sigma)\mathbb{I}(a \leq z \leq b)$ in two steps: first sample $u \sim U(\Phi((a - \mu)/\sigma), \Phi((b - \mu)/\sigma))$, then set $z = \mu + \sigma\Phi^{-1}(u)$ (Robert 1995).

Exercise 24.7 Gibbs sampling for logistic regression with the Student approximation

Derive the full conditionals for the joint model defined by Equations 24.88 to 24.91.