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}

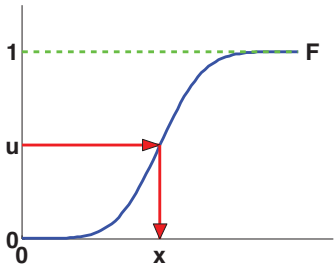


Figure 23.1 Sampling using an inverse CDF. Figure generated by `sampleCdf`.

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.

$$\begin{aligned} \Pr(F^{-1}(U) \leq x) &= \Pr(U \leq F(x)) \quad \text{(applying } F \text{ to both sides)} &(23.1)\\ &= F(x) \quad \text{(because } \Pr(U \leq y) = y &(23.2)\end{aligned}$$

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

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

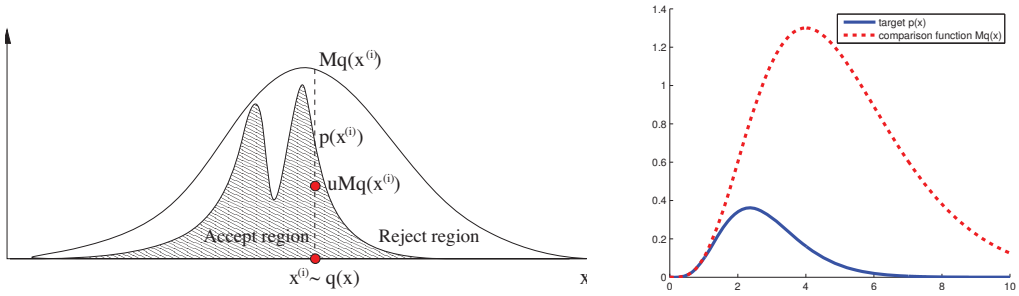


Figure 23.2 (a) Schematic illustration of rejection sampling. Source: Figure 2 of (Andrieu et al. 2003). Used with kind permission of Nando de Freitas. (b) Rejection sampling from a $\mathrm{Ga}(\alpha = 5.7, \lambda = 2)$ distribution (solid blue) using a proposal of the form $M\mathrm{Ga}(k, \lambda - 1)$ (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 $Mq(x) \geq \tilde{p}(x)$.

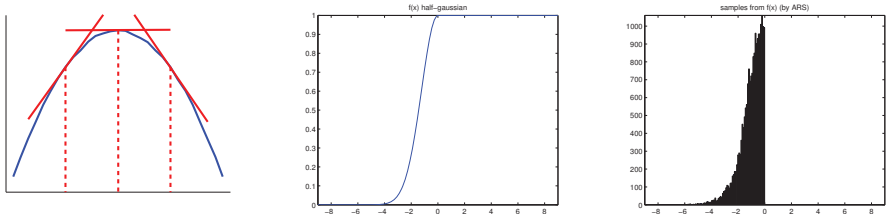
23.3.2 Example

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

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

One can show that if $X_i \overset{iid}{\sim} \mathrm{Expon}(\lambda)$, and $Y = X_1 + \cdots + X_k$, then $Y \sim \mathrm{Ga}(k, \lambda)$. 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.

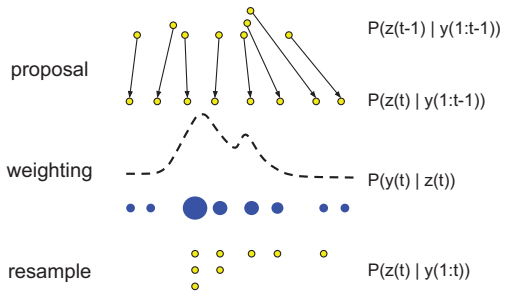


Figure 23.4 Illustration of particle filtering.

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

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

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 $\mathbf{z}_{i_{t-1}}$, 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.

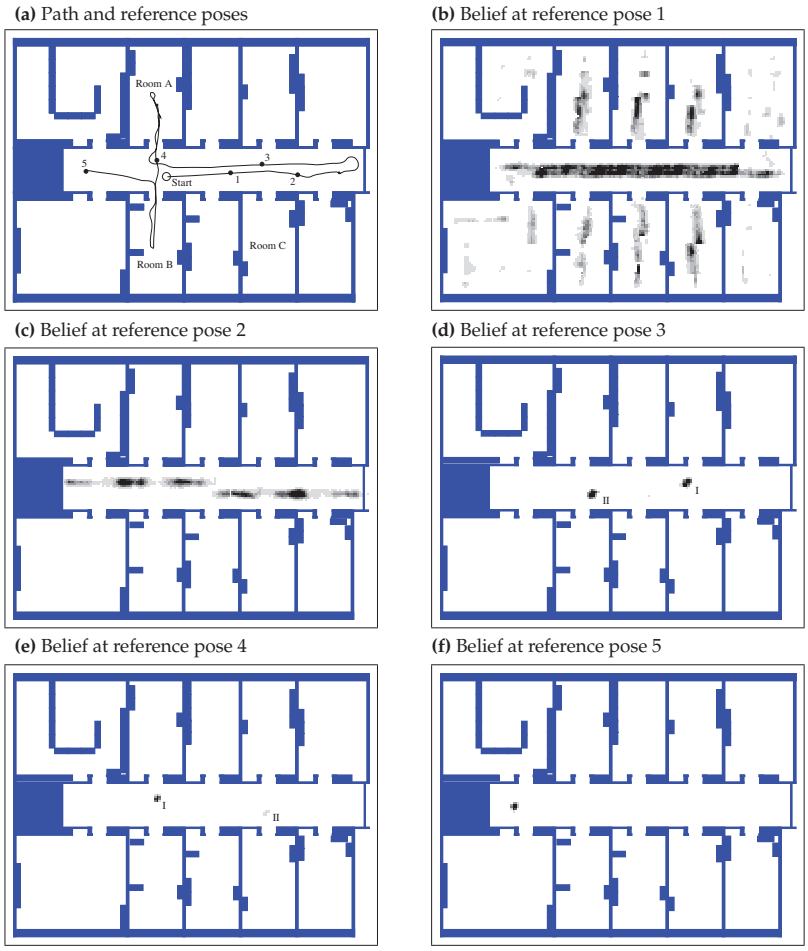


Figure 23.5 Illustration of Monte Carlo localization. Source: Figure 8.7 of (Thrun et al. 2006). Used with kind permission of Sebastian Thrun.

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.

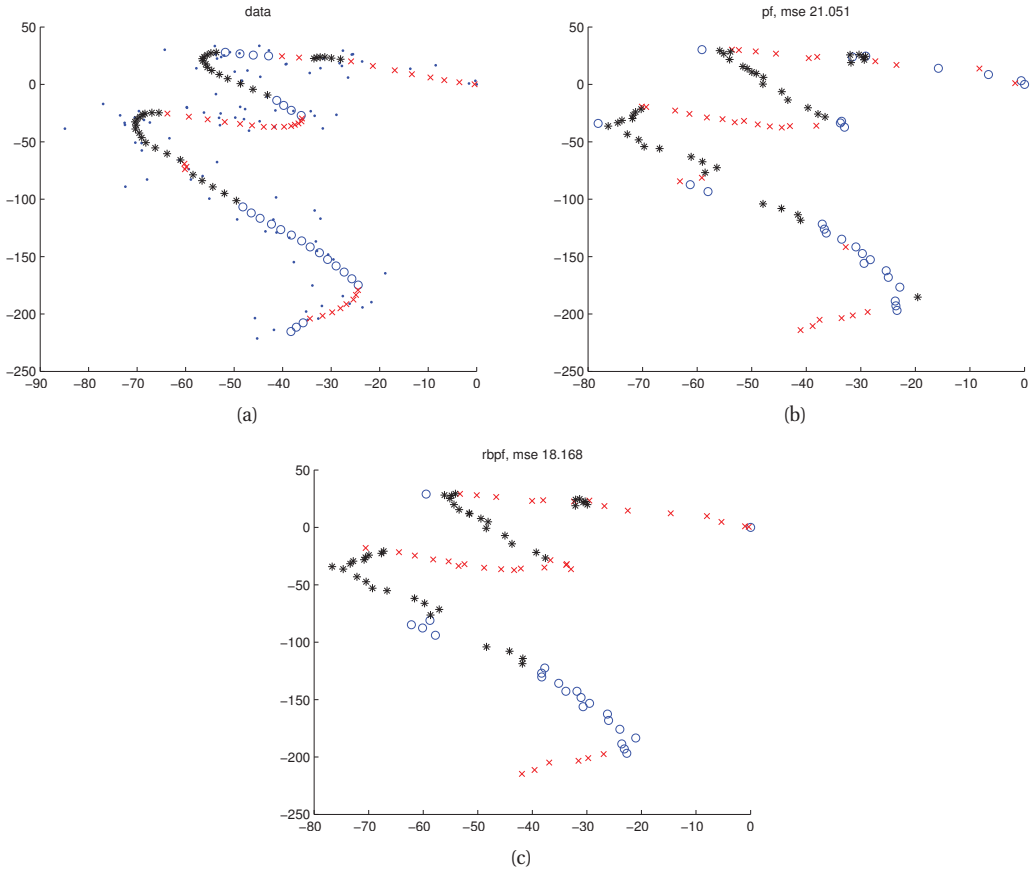
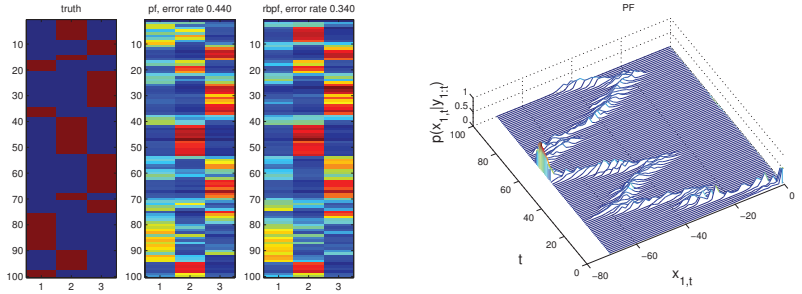


Figure 23.7 (a) A maneuvering target. The colored symbols represent the hidden discrete state. (b) Particle filter estimate. (c) RBPF estimate. Figure generated by `rbpfManeuverDemo`, based on code by Nando de Freitas.

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.