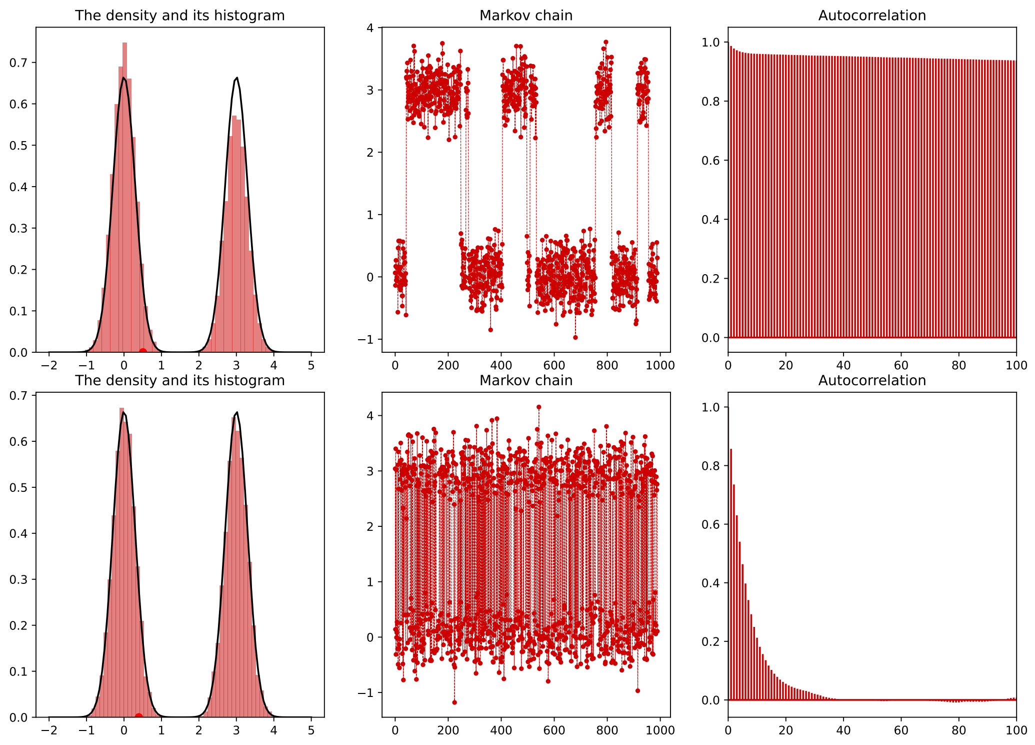

Figure 5.7: Random walk Metropolis-Hastings for a mixture of two Gaussians. The top panel shows the situation where $\sigma_q = 0.5$, so chain gets stuck in modes. This causes a high autocorrelation, and as such, this sampler is not considered to be a good one. When we set $\sigma_q = 4$, then the chain exhibits low autocorrelation and is a good sampler.

One can see that as $\beta \to \infty$, we have $p_\star^\beta(x) \to \delta_\mu(x)$, i.e., the target distribution is a Dirac delta at μ. This is an example of a more general result where sampling from $p_\star^\beta(x) \propto \exp(-\beta f(x))$ (as it is what the sampler is doing) leads to distributions that concentrate on the minima of $f(x)$ as $\beta \to \infty$.

In our case, for large β, the distribution would be concentrated around μ, that is maximum. Therefore, samples from this distribution would be very close to μ. The error can be verified and quantified in a number of challenging and nonconvex settings (Zhang et al., 2019).

5.7 MONITORING AND POSTPROCESSING MCMC OUTPUT

There are a number of ways to monitor the MCMC samples to ensure that the algorithm is working as expected. We will discuss a few of them here.

5.7.1 TRACE PLOTS

The simplest way to monitor the MCMC output is to plot the trace of the samples. This is a plot of the samples against the iteration number. This is what we have been doing in previous examples. If the trace plots show you that the chain is still "moving", then you can conclude that the chain is not yet converged. On the other hand, a trace plot from MH

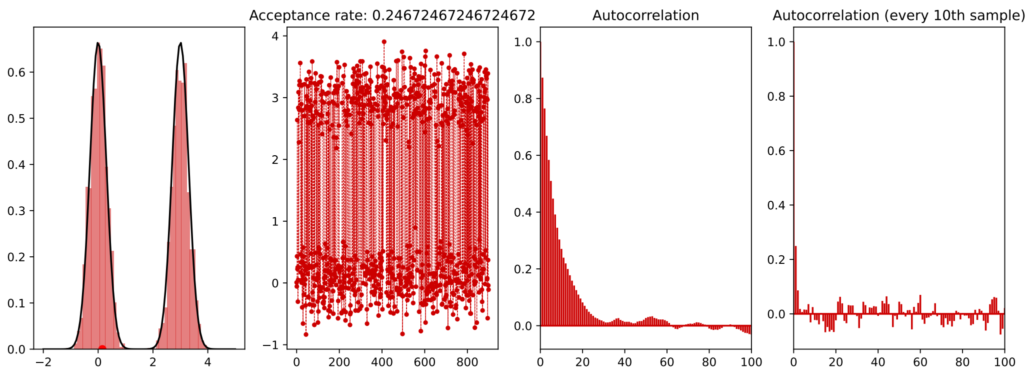

Figure 5.8: Thinning of MCMC samples. We keep every 10th sample for the same mixture of Gaussians example with $\sigma_q = 2$. It can be seen that the thinned MCMC chain exhibits significantly lower autocorrelation.

can also show you that the chain is stuck. It is then straightforward to conclude simple convergence issues from trace plots.

5.7.2 AUTOCORRELATION PLOTS

The autocorrelation plot is a plot of the autocorrelation function of the samples. The autocorrelation function is defined as

$$\rho_k = \frac{\mathrm{Cov}(X_t, X_{t+k})}{\mathrm{Var}(X_t)}.$$

This can be empirically computed on the samples coming from the Markov chain $(x_k)_{k\in\mathbb{N}}$. Since the aim of MCMC is to obtain nearly independent samples from a target $p_\star$, we expect a good MCMC chain to exhibit low autocorrelation. A bad chain which is not *mixing* well will exhibit high autocorrelation. An example can be seen from Fig. 5.7 and see its caption for more details. One way to choose the proposal variance is to ensure that the chain has a low autocorrelation. This is a very simple way to monitor the chain.

5.7.3 EFFECTIVE SAMPLE SIZE

There is a notion of effective sample size for MCMC methods. However, its computation is trickier than the IS one and it is usually implemented using software packages. The definition of the ESS for MCMC chains is given as

$$\mathrm{ESS} = \frac{N}{1 + 2\sum_{k=1}^{\infty} \rho_k},$$

where ρ_k is the autocorrelation function. The ESS is an approximate measure of the number of independent samples that we have. For example, if the chain exhibits no autocorrelation, then the ESS is equal to the number of samples. If the chain exhibits high autocorrelation, then the ESS will be very low, as the sum in denominator will be large.

The computation of effective sample size in MCMC is usually done by software packages. We will not go into the details of this computation here.

5.7.4 THINNING THE MCMC OUTPUT

One way to reduce the autocorrelation of the MCMC samples is to *thin* them. This is a postprocessing step that is done after the MCMC chain has been generated. The idea is to discard some of the samples and keep only a subset of them. This is done by keeping every kth sample. Since autocorrelation in an MCMC chain decays naturally over time, after reaching stationary, we can choose every kth of them and discard the rest. This will still give us a chain with the same stationary measure but with a lower autocorrelation. A demonstration of this can be seen from Fig .5.8.

5.8 EXAMPLES

In this section, we provide solved examples about the MCMC methods that we have discussed within this chapter.

Example 5.16 (Beta-Binomial Gibbs sampler)**.** Consider the following model

$$p(\theta) = \text{Beta}(\theta; \alpha, \beta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} \theta^{\alpha-1}(1-\theta)^{\beta-1},$$

and

$$p(x|\theta) = \text{Bin}(x; n, \theta) = \binom{n}{x} \theta^x (1-\theta)^{n-x}.$$

We would like to sample from (x, θ) joint using the Gibbs sampler. We know that, for this, we need full conditionals, i.e., we need $p(x|\theta)$ and $p(\theta|x)$. We can see that $p(x|\theta)$ is already provided in the definition of the model. Therefore, we only need to derive the posterior. We can write the joint distribution as

$$\begin{aligned} p(x, \theta) = p(x|\theta)p(\theta) &= \binom{n}{x} \theta^x (1-\theta)^{n-x} \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} \theta^{\alpha-1}(1-\theta)^{\beta-1}, \\ &= \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} \binom{n}{x} \theta^{x+\alpha-1}(1-\theta)^{n-x+\beta-1}. \end{aligned}$$

For Bayes theorem $p(\theta|x) = p(x|\theta)p(\theta)/p(x)$, we also need to compute $p(x)$. This is given by

$$\begin{aligned} p(x) = \int_0^1 p(x, \theta) \mathrm{d}\theta &= \int_0^1 \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} \binom{n}{x} \theta^{x+\alpha-1}(1-\theta)^{n-x+\beta-1} \mathrm{d}\theta, \\ &= \binom{n}{x} \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} \frac{\Gamma(x+\alpha)\Gamma(n-x+\beta)}{\Gamma(n+\alpha+\beta)}. \end{aligned}$$

Therefore, we can compute the posterior as

$$\begin{aligned} p(\theta|x) = \frac{p(x, \theta)}{p(x)} &= \frac{\cancel{\binom{n}{x} \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}} \theta^{x+\alpha-1}(1-\theta)^{n-x+\beta-1}}{\cancel{\binom{n}{x} \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}} \frac{\Gamma(x+\alpha)\Gamma(n-x+\beta)}{\Gamma(n+\alpha+\beta)}}, \\ &= \text{Beta}(\theta; x+\alpha, n-x+\beta). \end{aligned}$$

Therefore we can sample from $p(\theta|x)$ using any method to simulate a Beta variable. The Gibbs sampler is then defined as follows:

- Initialise x_0, θ_0
- For $k = 1, 2, \ldots$:
  - Sample $\theta_k \sim p(\theta | x_{k-1})$
  - Sample $x_k \sim p(x | \theta_k)$
- Return x_k, θ_k for $k = 1, 2, \ldots$.

We also note that simulated x_k are approximately from $p(x)$ which also gives us a way to approximate $p(x)$.

Example 5.17 (Metropolis-within-Gibbs)**.** One remarkable feature of the Gibbs sampler is that when we cannot derive the full conditionals (or too lazy to do it), we can instead target the full conditional with a single Metropolis step at each iteration. This is called the Metropolis-within-Gibbs algorithm and, remarkably, it samples from the correct posterior!

Let us return to Example 5.7. To recall the model, assume that we observe

$$Y_1, \ldots, Y_n | z, s \sim \mathcal{N}(y_i; z, s)$$

where we do not know z and s. Assume we have an independent prior on z and s:

$$p(z)p(s) = \mathcal{N}(z; m, \kappa^2)\mathcal{IG}(s; \alpha, \beta).$$

where $\mathcal{IG}(s; \alpha, \beta)$ is the inverse Gamma distribution

$$\mathcal{IG}(s; \alpha, \beta) = \frac{\beta^\alpha}{\Gamma(\alpha)} s^{-\alpha-1} \exp\left(-\frac{\beta}{s}\right).$$

In other words, we have

$$p(z)p(s) = \frac{1}{\sqrt{2\pi\kappa^2}} \exp\left(-\frac{(z-m)^2}{2\kappa^2}\right) \frac{\beta^\alpha}{\Gamma(\alpha)} s^{-\alpha-1} \exp\left(-\frac{\beta}{s}\right).$$

We are after the posterior distribution

$$\begin{aligned} p(z, s | y_1, \ldots, y_n) &\propto p(y_1, \ldots, y_n | z, s) p(z) p(s), \\ &= \prod_{i=1}^n \mathcal{N}(y_i; z, s) \mathcal{N}(z; m, \kappa^2) \mathcal{IG}(s; \alpha, \beta). \end{aligned}$$

Let us call our unnormalised posterior as $\bar{p}_\star(z, s | y_{1:n})$. Now instead of MH or defining Gibbs (requires us to derive full conditionals), we can use the Metropolis-within-Gibbs algorithm. For this, note the unnormalised full conditionals:

$$\bar{p}_\star(z | s, y_{1:n}) = \prod_{i=1}^n \mathcal{N}(y_i; z, s) \mathcal{N}(z; m, \kappa^2),$$

and

$$\bar{p}_\star(s|z, y_{1:n}) = \prod_{i=1}^n \mathcal{N}(y_i; z, s)\mathcal{IG}(s; \alpha, \beta).$$

In order to do this, we need to design proposals over z and s to target $\bar{p}(z|s, y_{1:n})$ and $\bar{p}(s|z, y_{1:n})$ respectively. This step will be a standard Metropolis as if we are solving each problem independently. We choose a random walk proposal for z:

$$q(z'|z) = \mathcal{N}(z'; z, \sigma_q^2).$$

and an independent proposal for s:

$$q(s') = \mathcal{IG}(s'; \alpha, \beta).$$

Therefore, we Metropolis-within-Gibbs can be implemented as follows

- Initialise z_0, s_0
- For $k = 1, 2, \ldots$:
- Metropolis step for z-marginal:
  - Sample $z' \sim q(z'|z_{k-1})$
  - Accept z' and set $z_k = z'$ with probability

    $$\mathsf{r}_z = \frac{\bar{p}_\star(z'|s_{k-1}, y_{1:n})}{\bar{p}_\star(z_{k-1}|s_{k-1}, y_{1:n})}$$

    which is simplified due to the symmetric proposal.
  - Otherwise set $z_k = z_{k-1}$.
- Metropolis step for s-marginal:
  - Sample $s' \sim q(s')$
  - Accept s' and set $s_k = s'$ with probability

    $$\begin{aligned}\mathsf{r}_s &= \frac{\bar{p}_\star(s'|z_k, y_{1:n})q(s_{k-1})}{\bar{p}_\star(s_{k-1}|z_k, y_{1:n})q(s')} \\ &= \frac{\prod_{i=1}^n \mathcal{N}(y_i; z, s')}{\prod_{i=1}^n \mathcal{N}(y_i; z, s_{k-1})}\end{aligned}$$

  - Otherwise set $s_k = s_{k-1}$.
- Return z_k, s_k for $k = 1, 2, \ldots$.

6

SEQUENTIAL MONTE CARLO

In this chapter, we introduce sequential Monte Carlo (SMC) methods. These methods are used to approximate a sequence of target distributions rather than just a single, fixed target. This can have a number of applications, including filtering and smoothing in state space models. We will briefly introduce state-space models, SMC and its connection to importance sampling, and application of SMC to filtering in state-space models, which is also called particle filtering.

6.1 INTRODUCTION

In this section, we depart from our standard setting where we have a single, fixed target $p_\star(x)$. In many problems in the real world, the target distributions are *evolving* over time. For example, consider the example of tracking a target, a straightforward extension of the source localisation problem we discussed in Example 5.6. Instead of a fixed target and fixed measurements, we could have easily the case of a moving target and fixed/moving sensors. In this case, we could recompute our posterior every time we get a new measurement, however, this could become very prohibitive (imagine every time you get new data, you need to run a new MCMC chain!). However, the applications of this framework is not limited to simple localisation examples, it broadly generalises to many dynamical systems. A few examples are volatility estimation in financial time series, robotics (tracking and control of moving arms), infectious disease modelling (tracking the spread of a disease), and many more. The idea of evolving sequence of distributions can also be used to target static problems, as we have seen in the example of simulated annealing.

Our running example in this section will be *state-space* models. A good example within this setting will be the target tracking example which summarises the notion of a *hidden state* and a sequence of *observations*. However, it is crucial to observe that the example generalises to any situation where a hidden, evolving quantity to be estimated (out in the wild) and a stream of data is received to update our latest belief on the state of the object.

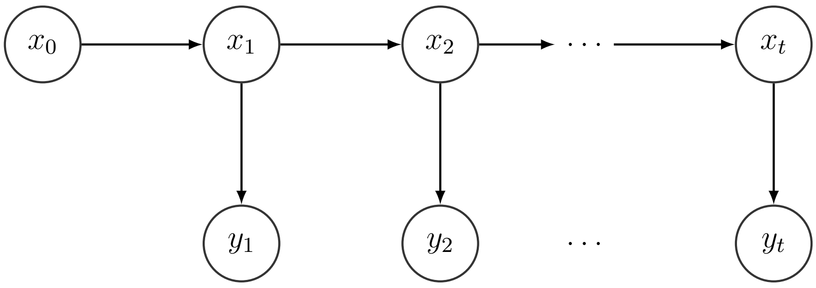

Figure 6.1: The conditional independence structure of a state-space model.

6.2 STATE-SPACE MODELS

Consider a Markov process $(X_t)_{t\geq 0}$ defined on the measurable space X with $\mathsf{X} \subset \mathbb{R}^{d_x}$. This process denotes the signal of interest, e.g., the state of an object, the velocity field of a partial differential equation (PDE), hence we call it *the signal process*. Similarly, we define another sequence of random variables $(y_t)_{t\geq 1}$, defined on $\mathsf{Y} \subset \mathbb{R}^{d_y}$, to denote our observation sequence, or *the observation process*. This sequence denotes the observed data coming from the signal process and it can typically consist of noisy sensor measurements or noisy observations. Based on this two sequences, we can define a *model* which we name as a state-space model. This is typically by three distributions (Doucet et al., 2000)

$$\begin{aligned} X_0 &\sim \mu(x_0) \\ X_t | \{X_{t-1} = x_{t-1}\} &\sim f(x_t | x_{t-1}), \\ Y_t | \{X_t = x_t\} &\sim g(y_t | x_t), \end{aligned}$$

where μ is called the prior distribution, f is a Markov transition kernel defined on X, and g as the likelihood function. For convenience, we always assume the densities exist in this document but a general construction is possible. See Fig. 6.1 for the conditional independence structure of this class of models.

6.2.1 THE FILTERING PROBLEM

Given a sequence of observations, a typical problem is to estimate the conditional distributions of the signal process $(X_t)_{t\geq 0}$ given the observed data. We denote this distribution with $\pi_t(x_t|y_{1:t})$ which is called *the filtering distribution*. The problem of sequentially updating the sequence of filtering distributions $(\pi_t(x_t|y_{1:t}))_{t\geq 1}$ is called *the filtering problem*.

To introduce the idea intuitively, consider the scenario of tracking a target. We denote the states of the target with $(x_t)_{t\geq 0}$ which may include positions and velocities. We assume that the target moves in space w.r.t. f, i.e., the transition model of the target is given by $f(x_t|x_{t-1})$. Observations may consist of the locations of the target on $\mathbb{R}^2$ or power measurements with associated sensors (which may result in high-dimensional observations). At each time t, we receive a measurement vector y_t conditional on the true state of the system x_t. The likelihood of each observation is assumed to follow $g(y_t|x_t)$.

We now provide a simple recursion to demonstrate one possible solution to the filtering problem. Assume that we are given the distribution at time $t-1$ (to define our sequential recursion) and would like to incorporate a recent observation y_t. One way to do so is to first perform *prediction*

$$\xi_t(x_t|y_{1:t-1}) = \int f(x_t|x_{t-1})\pi_{t-1}(x_{t-1}|y_{1:t-1})\mathrm{d}x_{t-1}, \qquad (6.1)$$

and obtain the predictive measure and then perform *update*

$$\pi_t(x_t|y_{1:t}) = \xi_t(x_t|y_{1:t-1}) \frac{g(y_t|x_t)}{p(y_t|y_{1:t-1})}, \tag{6.2}$$

where $p(y_t|y_{1:t-1}) = \int \xi_t(x_t|y_{1:t-1}) g(y_t|x_t) \mathrm{d}x_t$ is the incremental marginal likelihood.

Remark 6.1. We remark that the celebrated *Kalman filter* (Kalman, 1960) exactly implements recursions (6.1)–(6.2) in the case of

$$\begin{aligned} \mu(x_0) &= \mathcal{N}(x_0; \mu_0, \Sigma_0), \\ f(x_t|x_{t-1}) &= \mathcal{N}(x_t; Ax_{t-1}, Q), \\ g(y_t|x_t) &= \mathcal{N}(y_t; Cx_t, R). \end{aligned}$$

For this Gaussian system, computing the integral (6.1) and the update (6.2) is analytically tractable, which results in Kalman filtering recursions of the mean and the covariance of the filtering distribution $\pi_t(x_t|y_{1:t})$. We skip the update rules of the Kalman filter, as our main aim is to focus on sequential Monte Carlo in this course.

Finally, we can move on to show how to update joint filtering distribution of the states $x_{0:t}$. To see this, note the recursion

$$\begin{aligned} \pi_t(x_{0:t}|y_{1:t}) &= \frac{\bar{\pi}_t(x_{0:t}, y_{1:t})}{p(y_{1:t})} \\ &= \frac{\bar{\pi}_{t-1}(x_{0:t-1}, y_{1:t-1})}{p(y_{1:t-1})} \frac{f(x_t|x_{t-1}) g(y_t|x_t)}{p(y_t|y_{1:t-1})} \\ &= \pi_t(x_{0:t-1}|y_{1:t-1}) \frac{f(x_t|x_{t-1}) g(y_t|x_t)}{p(y_t|y_{1:t-1})}. \end{aligned}$$

This recursion will be behind the sequential Monte Carlo method we use for filtering in the next sections.

6.3 SEQUENTIAL MONTE CARLO FOR FILTERING

6.3.1 IMPORTANCE SAMPLING: RECAP

Before we introduce the sequential Monte Carlo sampling for filtering, we recall the basic importance sampling idea and its terminology accounting for the change of notation within this chapter. Assume that we aim at estimating expectations of a given density π, i.e., we would like to compute

$$\mathbb{E}_\pi[\varphi(X)] = \int \varphi(x) \pi(x) \mathrm{d}x.$$

We also assume that sampling from this density is not possible and we can only evaluate the *unnormalised* density $\bar{\pi}(x)$. One way to estimate this expectation is to sample from a proposal measure q and rewrite the integral as

$$\begin{aligned}\mathbb{E}_\pi[\varphi(X)] &= \int \varphi(x)\pi(x)\mathrm{d}x, \\ &= \frac{\int \varphi(x)\frac{\bar{\pi}(x)}{q(x)}q(x)\mathrm{d}x}{\int \frac{\bar{\pi}(x)}{q(x)}q(x)\mathrm{d}x}, \\ &\approx \frac{\frac{1}{N}\sum_{i=1}^N \varphi(x^{(i)})\frac{\bar{\pi}(x^{(i)})}{q(x^{(i)})}}{\frac{1}{N}\sum_{i=1}^N \frac{\bar{\pi}(x^{(i)})}{q(x^{(i)})}}, \qquad x^{(i)} \sim q, \quad i = 1, \ldots, N. \end{aligned} \tag{6.3}$$

Let us now introduce the unnormalised weight function[1]

$$W(x) = \frac{\bar{\pi}(x)}{q(x)}. \tag{6.4}$$

With this, the Eq. (6.3) becomes

$$\begin{aligned}\hat{\varphi}_{\text{IS}}^N &= \frac{\frac{1}{N}\sum_{i=1}^N \varphi(x^{(i)})W(x^{(i)})}{\frac{1}{N}\sum_{i=1}^N W(x^{(i)})}, \qquad x^{(i)} \sim q, \quad i = 1, \ldots, N, \\ &= \frac{\sum_{i=1}^N \varphi(x^{(i)})\mathsf{W}^{(i)}}{\sum_{i=1}^N \mathsf{W}^{(i)}}, \qquad x^{(i)} \sim q, \quad i = 1, \ldots, N,\end{aligned}$$

where $\mathsf{W}^{(i)} = W(x^{(i)})$ are called *the unnormalised weights*. Finally, we can obtain the estimator in a more convenient form,

$$\hat{\varphi}_{\text{IS}}^N = \sum_{i=1}^N \mathsf{w}^{(i)}\varphi(x^{(i)}),$$

by introducing the *normalised importance weights*

$$\mathsf{w}^{(i)} = \frac{\mathsf{W}^{(i)}}{\sum_{i=1}^N \mathsf{W}^{(i)}}, \tag{6.5}$$

for $i = 1, \ldots, N$. We note that the particle approximation of π in this case is given as

$$\pi^N(x)\mathrm{d}x = \sum_{i=1}^N \mathsf{w}^{(i)}\delta_{x^{(i)}}(x)\mathrm{d}x. \tag{6.6}$$

In the following section, we will derive the importance sampler aiming at building particle approximations of $\pi_t(x_{0:t}|y_{1:t})$ for a state-space model.

6.3.2 IMPORTANCE SAMPLING FOR STATE-SPACE MODELS: THE EMERGENCE OF THE GENERAL PARTICLE FILTER

In this section, we simply derive an importance sampler for the joint filtering distribution $\pi_t(x_{0:t}|y_{1:t})$. We will see in the process that the particle filter is a special case of this conceptually simple importance sampler (defined just in many variables instead of one) and the infamous bootstrap particle filter is a further simplified case.

[1]More technically, these weights are the evaluations of the Radon-Nikodym derivative $W(x) = \frac{\mathrm{d}\gamma}{\mathrm{d}q}(x)$ (which, in this case, is just a ratio as we assume absolute continuity implicitly).

Let us assume that, in order to build an estimator of $\pi_t(x_{0:t}|y_{1:t})$, we have a proposal distribution over the entire path space $x_{0:t}$ denoted $q(x_{0:t})$. Note that, we also denote the unnormalised distribution of $x_{0:t}$ as $\bar{\pi}(x_{0:t}, y_{1:t})$ which is given as

$$\bar{\pi}(x_{0:t}, y_{1:t}) = \mu(x_0) \prod_{k=1}^{t} f(x_k|x_{k-1})g(y_k|x_k). \tag{6.7}$$

This simply the joint distribution of all variables $(x_{0:t}, y_{1:t})$. Just as in the regular importance sampling case in eq. (6.4), we write

$$W_{0:t}(x_{0:t}) = \frac{\bar{\pi}(x_{0:t}, y_{1:t})}{q(x_{0:t})}.$$

Obviously, given samples from the proposal $x_{0:t}^{(i)} \sim q(x_{0:t})$, one can easily build the same weighted measure as in (6.6) on the path space by evaluating the weight $\mathsf{W}_{0:t}^{(i)} = W_{0:t}(x_{0:t}^{(i)})$ for $i = 1, \ldots, N$ and building a particle approximation

$$\pi^N(x_{0:t})\mathrm{d}x_{0:t} = \sum_{i=1}^{N} \mathsf{W}_{0:t}^{(i)} \delta_{x_{0:t}^{(i)}}(x_{0:t})\mathrm{d}x_{0:t}.$$

However, this would be an undesirable scheme: We would need to store all variables in memory which is infeasible as t grows. Furthermore, with the arrival of a new observation y_{t+1}, this would have to be re-done, as this importance sampling procedure does not take into account the dynamic properties of the SSM. Therefore, implementing this sampler to build estimators sequentially is out of question.

Fortunately, we can design our proposal in certain ways so that this process can be done sequentially, starting from 0 to t. Furthermore, this would allow us to run the filter *online* and incorporate new observations. The clever choices of the proposal here lead to a variety of different *particle filters* as we shall see next. Let us consider a decomposition of the proposal

$$q(x_{0:t}) = q(x_0) \prod_{k=1}^{t} q(x_k|x_{1:k-1}).$$

Note that, based on this, we can build a recursion for the function $W(x_{0:t})$ by writing

$$\begin{aligned} W_{0:t}(x_{0:t}) &= \frac{\bar{\pi}(x_{0:t}, y_{1:t})}{q(x_{0:t})}, \\ &= \frac{\bar{\pi}(x_{0:t-1}, y_{1:t-1})}{q(x_{0:t-1})} \frac{f(x_t|x_{t-1})g(y_t|x_t)}{q(x_t|x_{0:t-1})}, \\ &= W_{0:t-1}(x_{0:t-1}) \frac{f(x_t|x_{t-1})g(y_t|x_t)}{q(x_t|x_{0:t-1})}, \\ &= W_{0:t-1}(x_{0:t-1}) W_t(x_{0:t}). \end{aligned} \tag{6.8}$$

That is, under this scenario, the weights can be computed *recursively* – given the weights of time $t-1$, one can evaluate $W_{0:t}(x_{0:t})$ and update the weights. However, this would not solve the infeasibility problem mentioned earlier, as the cost of evaluating using the whole path of samples is still out of question. Finally, to remedy this, we can further simplify our proposal

$$q(x_{0:t}) = q(x_0) \prod_{k=1}^{t} q(x_k|x_{k-1}).$$

by removing dependence to the past, essentially choosing a Markov process as a proposal. This allows us to obtain purely recursive weight computation

$$
\begin{aligned}
W_{0:t}(x_{0:t}) &= \frac{\bar{\pi}(x_{0:t}, y_{1:t})}{q(x_{0:t})}, && (6.9)\\
&= \frac{\bar{\pi}(x_{0:t-1}, y_{1:t-1})}{q(x_{0:t-1})} \frac{f(x_t|x_{t-1})g(y_t|x_t)}{q(x_t|x_{t-1})}, && (6.10)\\
&= W_{0:t-1}(x_{0:t-1}) \frac{f(x_t|x_{t-1})g(y_t|x_t)}{q(x_t|x_{t-1})}, && (6.11)\\
&= W_{0:t-1}(x_{0:t-1}) W_t(x_t, x_{t-1}), && (6.12)
\end{aligned}
$$

using only the samples from time $t-1$ and time t. The advantage of this scheme is explicit in the notation: Note that the final weight function W_t only depends on (x_t, x_{t-1}), but not the whole past as in (6.8). The function $W_t(x_t, x_{t-1})$ is called the incremental weight function.

6.3.3 SEQUENTIAL IMPORTANCE SAMPLING

We can now see how the one-step update of this sampler works given a new observation. Assume that we have computed the unnormalised weights $\mathsf{W}_{1:t-1}^{(i)} = W(x_{0:t-1}^{(i)})$ recursively and obtained samples $x_{0:t-1}^{(i)}$. As we mentioned earlier, we only need the last sample $x_{t-1}^{(i)}$ to obtain the weight update given in (6.12). And also note that $\mathsf{W}_{1:t-1}^{(i)}$ for $i = 1, \ldots, N$ are just numbers, they do not need the storage of previous samples. Given this, we can now sample from the Markov proposal $x_t^{(i)} \sim q(x_t|x_{t-1}^{(i)})$ and compute the weights of the path sampler at time t as

$$\mathsf{W}_{1:t}^{(i)} = \mathsf{W}_{1:t-1}^{(i)} \times \mathsf{W}_t^{(i)},$$

where

$$\mathsf{W}_t^{(i)} = \frac{f(x_t^{(i)}|x_{t-1}^{(i)})g(y_t|x_t^{(i)})}{q(x_t^{(i)}|x_{t-1}^{(i)})}.$$

What we described in other words is that, given the samples $x_{t-1}^{(i)}$, we first perform sampling step

$$x_t^{(i)} \sim q(x_t|x_{t-1})$$

and then compute

$$\mathsf{W}_t^{(i)} = \frac{f(x_t^{(i)}|x_{t-1}^{(i)})g(y_t|x_t^{(i)})}{q(x_t^{(i)}|x_{t-1}^{(i)})}.$$

and update

$$\mathsf{W}_{1:t}^{(i)} = \mathsf{W}_{1:t-1}^{(i)} \times \mathsf{W}_t^{(i)}.$$

These are unnormalised weights and we normalise them to obtain,

$$\mathsf{w}_{1:t}^{(i)} = \frac{\mathsf{W}_{1:t}^{(i)}}{\sum_{i=1}^N \mathsf{W}_{1:t}^{(i)}},$$

Algorithm 14 Sequential Importance Sampling (SIS)

```
1: Sample $x_0^{(i)} \sim q(x_0)$ for $i = 1, \ldots, N$.
2: for $t \geq 1$ do
3:     Sample: $x_t^{(i)} \sim q(x_t | x_{t-1}^{(i)})$,
4:     Compute weights:
           $$\mathsf{W}_t^{(i)} = \frac{f(x_t^{(i)} | x_{t-1}^{(i)}) g(y_t | x_t^{(i)})}{q(x_t^{(i)} | x_{t-1}^{(i)})}.$$
       and update
           $$\mathsf{W}_{1:t}^{(i)} = \mathsf{W}_{1:t-1}^{(i)} \times \mathsf{W}_t^{(i)}.$$
       Normalise weights,
           $$\mathsf{w}_{1:t}^{(i)} = \frac{\mathsf{W}_{1:t}^{(i)}}{\sum_{i=1}^N \mathsf{W}_{1:t}^{(i)}}.$$
5:     Report
           $$\pi_t^N(x_{0:t}) \mathrm{d}x_{0:t} = \sum_{i=1}^N \mathsf{w}_{1:t}^{(i)} \delta_{x_{0:t}^{(i)}}(x_{0:t}) \mathrm{d}x_{0:t}.$$
6: end for
```

which finally leads to the empirical measure,

$$\pi^N(x_{0:t}) \mathrm{d}x_{0:t} = \sum_{i=1}^N \mathsf{w}_{1:t}^{(i)} \delta_{x_{0:t}^{(i)}}(x_{0:t}) \mathrm{d}x_{0:t}.$$

The full scheme is given in Algorithm 14. This method is called sequential importance sampling (SIS). This is not very popular in the literature due to the well known *weight degeneracy* problem. We next introduce a resampling step to this method and will obtain the first particle filter in this lecture.

6.3.4 SEQUENTIAL IMPORTANCE SAMPLING WITH RESAMPLING: THE GENERAL PARTICLE FILTER

We finally describe the general particle filter by extending the above method with a resampling step employed after the weighting step. We will show in a practical session that the SIS method without resampling easily degenerates, i.e., after some time, only a single weight approximates to 1 and others to 0, rendering the method a point estimate. To keep the particle diversity, a resampling method is introduced in between weighting and sampling steps. This step does not introduce a systematic bias, although, it adds additional terms to the overall L_p error.

With the additional resampling step, the sequential importance sampling with resampling (SISR) takes the form given in Algorithm 15. We note that, effectively, resampling step sets $\mathsf{W}_{1:t-1}^{(i)} = 1/N$ for $i = 1, \ldots, N$. Therefore, we only need to compute the last

Algorithm 15 Sequential Importance Sampling with Resampling (SISR)

```
1: Sample $x_0^{(i)} \sim q(x_0)$ for $i = 1, \ldots, N$.
2: for $t \geq 1$ do
3:     Sample: $\tilde{x}_t^{(i)} \sim q(x_t|x_{t-1}^{(i)})$,
4:     Compute weights:
           $$\mathsf{W}_t^{(i)} = \frac{f(\tilde{x}_t^{(i)}|x_{t-1}^{(i)})g(y_t|\tilde{x}_t^{(i)})}{q(\tilde{x}_t^{(i)}|x_{t-1}^{(i)})}.$$
       Normalise weights,
           $$\mathsf{w}_t^{(i)} = \frac{\mathsf{W}_t^{(i)}}{\sum_{i=1}^N \mathsf{W}_t^{(i)}}.$$
5:     Report
           $$\pi_t^N(x_t)\mathrm{d}x_t = \sum_{i=1}^N \mathsf{w}_t^{(i)} \delta_{\tilde{x}_t^{(i)}}(x_t)\mathrm{d}x_t.$$
6: Resample:
           $$x_t^{(i)} \sim \sum_{i=1}^N \mathsf{w}_t^{(i)} \delta_{\tilde{x}_t^{(i)}}(x_t)\mathrm{d}x_t.$$
7: end for
```

incremental weight and weight our particles with the current weight. Also, note that the resampling step does introduce extra error but does not induce bias, since moments of π_t^N does not change.

6.3.5 THE BOOTSTRAP PARTICLE FILTER

In the general particle filter, the proposal $q(x_t|x_{t-1})$ is a design choice to be made and this depends on our specific knowledge of a good proposal for a given system. For example, one can incorporate future observations into this proposal in an ad-hoc or use the proposal choices like in the auxiliary particle filter (APF).

A generic choice exists, however, that is simply setting $q(x_t|x_{t-1}) = f(x_t|x_{t-1})$, i.e., using the transition density of the SSM under consideration as a proposal. The algorithm simplifies considerably in this case and the resulting method is called the bootstrap particle filter (BPF) which is given in Alg. 16. This algorithm has multiple appealing intuitive explanations beyond the derivation we provided based on importance sampling here. It can be most generally thought as an evolutionary method. To uncover some of this intuition, see Fig. 6.2.

To elaborate the interpretation, consider a set of particles $x_{t-1}^{(i)}$ representing the state of the system at time $t-1$. If our state-space transition model $f(x_t|x_{t-1})$ is well-specified (that is, if the underlying system we aim at tracking does indeed move according to f), then the first intuivite step we can do to predict where the state would be at time t would be to move particles according to f, that is sampling $\tilde{x}_t^{(i)} \sim f(x_t|x_{t-1}^{(i)})$ which is the first step

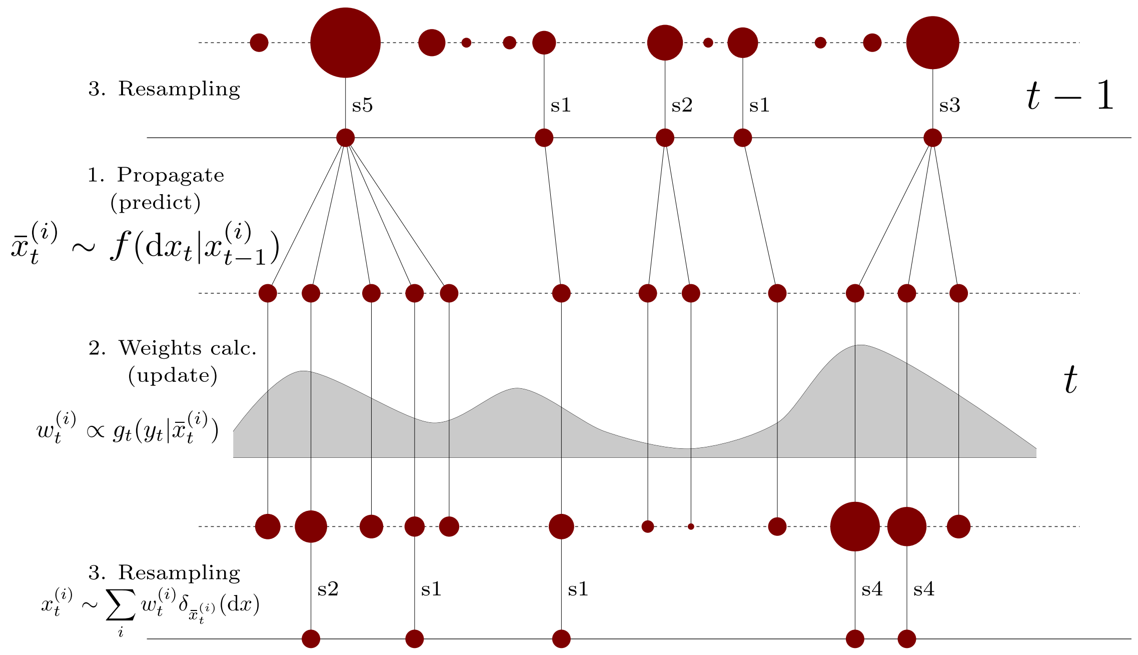
3. Resampling
s5
s1
s2
s1
s3
$t-1$
1. Propagate (predict)
$\bar{x}_t^{(i)} \sim f(\mathrm{d}x_t|x_{t-1}^{(i)})$
2. Weights calc. (update)
$w_t^{(i)} \propto g_t(y_t|\bar{x}_t^{(i)})$
t
3. Resampling
$x_t^{(i)} \sim \sum_i w_t^{(i)} \delta_{\bar{x}_t^{(i)}}(\mathrm{d}x)$
s2
s1
s1
s4
s4

Figure 6.2: Intuitive model of BPF (Figure courtesy Victor Elvira).

of the BPF. This gives us a predictive distribution which consists of $\tilde{x}_t^{(i)}$ for $i = 1, \ldots, N$. The prediction step (naturally) does not require to observe the data point at y_t. Once we observe the data point y_t, we can then use this data point to evaluate a fitness measure for our particles. In other words, if a predictive particle $\tilde{x}_t^{(i)}$ is a good fit to the observation, we would expect its likelihood $g(y_t|\tilde{x}_t^{(i)})$ to be high. Otherwise, this likelihood would be low. Thus, it intuitively makes sense to use our likelihood evaluations as "weights", that is to compute a measure of fitness for each particle. That is exactly what the BPF does at the second step by computing weights using the likelihood evaluations. The final step is then to use these relative weights to *resample* – a step that is used to refine the cloud of particles we have. Simply, the resampling step removes some of the particles with low weights (that are bad fits to the observation) and regenerates the particles with high weights.

The connection to evolutionary terms are clearer within this interpretation. The sampling step in the BPF can be seen as "mutation" that introduces changes to an individual particle according to some mutation mechanism (in our case, the dynamics). Then, weighting and resampling correspond to "selection" step, where individual particles are evaluated w.r.t. a fitness measure coming from the environment (defined by an observation) and individuals are reproduced in a random manner w.r.t. their fitness.

6.3.6 PRACTICAL IMPLEMENTATION OF THE BPF

Of course, the BPF can become numerically unstable if the weights are too small or too large. This is in line with the theme we have seen about computing small or large numbers (especially involving normalisation) throughout this course. To avoid a problem here, too, we need to perform the comptutations in the log-domain. For example, after sampling from the proposal $\tilde{x}_t^{(i)} \sim f(x_t|x_{t-1})$, we can compute the log-weights as

$$\log \mathsf{W}_t^{(i)} = \log g(y_t|\tilde{x}_t^{(i)})$$

We can then compute the normalised weights $\mathsf{w}_t^{(i)}$ using the trick introduced in . This will ensure the stable computation of weights and prevent instability.

Algorithm 16 Bootstrap particle filter (BPF)

```
1: Sample $x_0^{(i)} \sim q(x_0)$ for $i = 1, \ldots, N$.
2: for $t \geq 1$ do
3:     Sample: $\tilde{x}_t^{(i)} \sim f(x_t | x_{t-1}^{(i)})$,
4:     Compute weights:
           $$\mathsf{W}_t^{(i)} = g(y_t | \tilde{x}_t^{(i)}).$$
       Normalise weights,
           $$\mathsf{w}_t^{(i)} = \frac{\mathsf{W}_t^{(i)}}{\sum_{i=1}^N \mathsf{W}_t^{(i)}}.$$
5:     Report
           $$\pi_t^N(x_t)\mathrm{d}x_t = \sum_{i=1}^N \mathsf{w}_t^{(i)} \delta_{\tilde{x}_t^{(i)}}(x_t)\mathrm{d}x_t.$$
6: Resample:
           $$x_t^{(i)} \sim \sum_{i=1}^N \mathsf{w}_t^{(i)} \delta_{\tilde{x}_t^{(i)}}(x_t)\mathrm{d}x_t.$$
7: end for
```

6.4 EXAMPLES

We will next consider some examples of the BPF in action.

BIBLIOGRAPHY

Agapiou, Sergios; Papaspiliopoulos, Omiros; Sanz-Alonso, Daniel; and Stuart, Andrew M. 2017. *Importance sampling: Intrinsic dimension and computational cost*. In Statistical Science, pp. 405–431. Cited on p. 73.

Akyildiz, Omer Deniz. March 2019. *Sequential and adaptive Bayesian computation for inference and optimization*. Ph.D. thesis, Universidad Carlos III de Madrid. Can be accessed from: http://akyildiz.me/works/thesis.pdf. Cited on pp. 59, 62, and 69.

Akyildiz, Ömer Deniz and Míguez, Joaquín. 2021. *Convergence rates for optimised adaptive importance samplers*. In Statistics and Computing, vol. 31, no. 2, pp. 1–17. Cited on pp. 71 and 73.

Barber, David. 2012. *Bayesian reasoning and machine learning*. Cambridge University Press. Cited on p. 50.

Bishop, Christopher M. 2006. *Pattern Recognition and Machine Learning*. Springer. Cited on p. 50.

Box, GEP and Müller, Mervin E. 1958. *A Note on the Generation of Random Normal Deviates*. In The Annals of Mathematical Statistics, vol. 29, no. 2, pp. 610–611. Cited on p. 10.

Cemgil, A Taylan. 2014. *A tutorial introduction to Monte Carlo methods, Markov Chain Monte Carlo and particle filtering*. In Academic Press Library in Signal Processing, vol. 1, pp. 1065–1114. Cited on p. 95.

Devroye, Luc. 1986. *Non-Uniform Random Variate Generation*. Cited on p. 4.

Douc, Randal; Moulines, Eric; Priouret, Pierre; and Soulier, Philippe. 2018. *Markov chains*. Springer. Cited on pp. 87 and 91.

Douc, Randal; Moulines, Éric; and Stoffer, David. 2013. *Nonlinear Time Series: Theory, Methods and Applications with R Examples*. Chapman & Hall. Cited on p. 87.

Doucet, Arnaud; Godsill, Simon; and Andrieu, Christophe. 2000. *On sequential Monte Carlo sampling methods for Bayesian filtering*. In Statistics and computing, vol. 10, no. 3, pp. 197–208. Cited on p. 115.

Elvira, Víctor; Martino, Luca; and Robert, Christian P. 2018. *Rethinking the effective sample size*. In International Statistical Review. Cited on p. 75.

Hwang, Chii-Ruey. 1980. *Laplace's method revisited: weak convergence of probability measures*. In The Annals of Probability, pp. 1177–1182. Cited on p. 106.

Kalman, Rudolph Emil. 1960. *A new approach to linear filtering and prediction problems*. In Journal of Fluids Engineering, vol. 82, no. 1, pp. 35–45. Cited on p. 116.

Lamberti, Roland; Petetin, Yohan; Septier, François; and Desbouvries, François. 2018. *A double proposal normalized importance sampling estimator*. In 2018 IEEE Statistical Signal Processing Workshop (SSP), pp. 238–242. IEEE. Cited on p. 72.

Martino, Luca; Luengo, David; and Míguez, Joaquín. 2018. *Independent random sampling methods*. Springer. Cited on pp. i, 5, 12, 13, 16, 17, and 24.

Murphy, Kevin P. 2007. *Conjugate Bayesian analysis of the Gaussian distribution*. In def, vol. 1, no. $2\sigma2$, p. 16. Cited on pp. 39 and 48.

———. 2022. *Probabilistic machine learning: an introduction*. MIT press. Cited on p. 50.

Owen, Art B. 2013. *Monte Carlo theory, methods and examples*. Cited on p. 75.

Robert, Christian P and Casella, George. 2004. *Monte Carlo statistical methods*. Springer. Cited on pp. i, 59, and 75.

———. 2010. *Introducing Monte Carlo methods with R*, vol. 18. Springer. Cited on p. 69.

Yıldırım, Sinan. 2017. *Sabanci University IE 58001 Lecture notes: Simulation Methods for Statistical Inference*. Cited on pp. i, 19, 86, 95, and 96.

Zhang, Ying; Akyildiz, Ömer Deniz; Damoulas, Theodoros; and Sabanis, Sotirios. 2019. *Nonasymptotic estimates for Stochastic Gradient Langevin Dynamics under local conditions in nonconvex optimization*. In arXiv preprint arXiv:1910.02008. Cited on p. 109.