Introduction to Sequential Monte Carlo

Rui-Yang Zhang

Contents

Chapter 1

Introduction

Consider the task of drawing samples from a known distribution π, supported on $\mathbb{R}^n$. This is a common task in statistics, and especially in Bayesian inference, as we would want to compute certain quantities (such as the posterior mean of a parameter of interest) that are dependent on integrals with respect to (w.r.t.) a probability distribution (e.g. expectation of the posterior distribution). These integrals are often of high dimensions and of complex form, which stops us from computing them directly by hand and turning to numerical approximations for help. Such complicated integrals are often called **intractable integrals**.

One fundamental idea of doing numerical approximations of such integrals (integrals that could be written as an expectation) is to use the Monte Carlo methods. Given a random variable X with probability measure π that admits a density (i.e. we have $\pi(dx) = \pi(x)dx$), we may wish to compute the integral of the form

$$\int f(x)\pi(dx) = \int f(x)d\pi(x) = \int f(x)\pi(x)dx = \mathbb{E}_X[f(X)] \tag{1}$$

where f is some arbitrary function. Note that the first equality above is a mere notation change - these two forms of integration w.r.t. a probability measure would be used interchangeably in the rest of this note. Assuming that we could obtain independent and identically distributed (i.i.d.) samples $X_1, X_2, \ldots, X_N$ of the target random variable/probability distribution X, we can approximate the above quantity by the empirical mean, which is

$$\mathbb{E}_X[f(X)] \approx \frac{1}{N}\sum_{i=1}^{N} f(X_i). \tag{2}$$

This method of numerical integration is known as the **Monte Carlo method**, and the approximation is known as the **Monte Carlo approximations**.

As a conceptual remark, the Monte Carlo method is actually doing an approximation to the target distribution π using the empirical distribution from the samples. Formally, given N i.i.d. samples $X_1, X_2, \ldots, X_N$ of the target random variable/probability distribution X, the Monte Carlo estimate of π is provided by

$$\pi(dx) \approx \sum_{i=1}^{N} \frac{1}{N}\delta_{X_i}(dx) \tag{3}$$

where δ_{X_i} is the Dirac delta function, and $\delta_{X_i}(dx) = 1$ when $x = X_i$ and $\delta_{X_i}(dx) = 0$ otherwise. Intuitively, we can think about this as approximating a density function using a histogram, as illustrated in Figure 1. This also allows us to derive the Monte Carlo approximation of the integral, as stated by Equation (2), by combining Equations (3) and (1), which is given by

$$\mathbb{E}_X[f(X)] = \int f(x)\pi(dx) \approx \int f(x)\sum_{i=1}^{N}\frac{1}{N}\delta_{x_i}(dx)$$
$$= \sum_{i=1}^{N}\frac{1}{N}\int f(x)\delta_{x_i}(dx) = \sum_{i=1}^{N}\frac{1}{N}f(x_i).$$

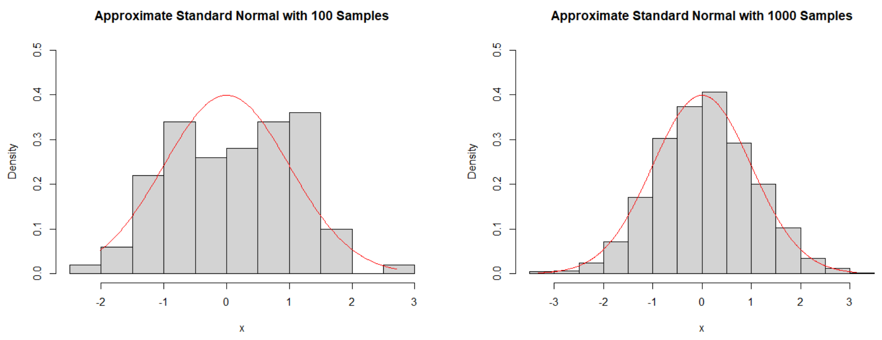

Figure 1: Simulations of approximating a standard normal using Monte Carlo samples. Sample sizes are 100 (left) and 1000 (right).

The Monte Carlo approximation is justified by the **law of large numbers** which states that given i.i.d. samples of a random variable with finite mean, the sample mean will converge, as the number of samples tends to infinity, to the true mean of the random variable. The actual modes of convergences are almost surely (for the strong law of large numbers) and in probability (for the weak law of large numbers). In addition, the above case involves the expectation of a function of a random variable. Its Monte Carlo approximation is justified using the uniform law of large numbers, as long as the function is continuous and is bound above by another function with finite mean. Further details and proofs of such theoretical results can be found in standard measure-theoretic probability theory textbooks such as Williams (1991).

Another benefit of the Monte Carlo method is that we could use the central limit theorem to provide a confidence interval for the approximation when the variance of the random variable (or the function of the random variable) is finite. Although the actual confidence interval would require the knowledge of true variance - which is often not accessible for numerical integration tasks - we could replace it with an empirical estimate of variance.

The Monte Carlo method is the fundamental principle that the rest of this report is based on. To actually conduct this method in practice would require us to have samples (and i.i.d. ones) of the target probability distribution. This challenge is the key question that the rest of this report (and multiple decades of academic research by thousands of researchers) is trying to tackle.

In this notes, we will introduce Sequential Monte Carlo, and describe some of the common types of Sequential Monte Carlo algorithms with their respective properties. We will introduce some elementary sampling techniques in Chapter 2, such as rejection sampling and importance sampling. Then, we will introduce the hidden Markov model in Chapter 3 to set up the notations as well as the foundations for particle filters. The sequential importance sampling, a first attempt to approximate the hidden Markov models is provided in Chapter 4, and some of its problems are addressed in Chapter 5, which leads us to the full particle filters in Chapter 6. Finally, we will consider the problem of smoothing in Chapter 7 and the problem of parameter estimation in Chapter 8.

The content of this note is mostly introductory, so exact references will be omitted. Standard textbooks such as Chopin and Papaspiliopoulos (2020), Liu (2001), and Robert and Casella (1999) would cover all of the material with more detailed references.

Chapter 2

Elementary Sampling Techniques

We will start the chapter with some elementary sampling techniques. Some of those would be developed and extended to sample from more complicated distributions.

2.1 Inverse CDF

Consider a continuous random variable X with an invertible (on the support[1]) cumulative distribution function (CDF) $F(x) = \mathbb{P}(X \leq x)$. This is a very simple setup. Such random variables include a normal distribution with mean μ and variance σ^2 or an exponential distribution with mean μ. We first have the following result.

Proposition 2.1. *For a continuous random variable X with invertible (on the support) cumulative distribution function F, the random variable $F(X)$ follows a Uniform$[0,1]$ distribution.*

Proof. For any $u \in [0,1]$ and x in the support of X, we have

$$\mathbb{P}(F(X) \leq u) = \mathbb{P}(X \leq F^{-1}(u)) = F(F^{-1}(u)) = u$$

as required for $F(X)$ to follow the uniform distribution on $[0,1]$. $\square$

This result implies that if we can get the CDF of the target distribution, and we can invert it, then sampling from that distribution can be achieved by first sample from the Uniform$[0,1]$ distribution[2] then passing the sample through the inverse of the cumulative distribution F^{-1} will yield a sample from the target distribution. This method is known as the **inverse CDF** method for obvious reasons, and the procedure is listed below which generates a list of samples $\{X_i\}_i$.

Algorithm 1 Inverse CDF Algorithm

```
Require: Cumulative distribution function $F$ of the target random variable that is invertible.
1: for $i = 1, 2, \cdots$ do
2:     Draw $U \sim$ Uniform$[0,1]$.
3:     Set $X_i = F^{-1}(U)$.
4: end for
```

2.2 Rejection Sampling

Common distributions like an exponential distribution and a normal distribution can be sampled this way, but it is quite restrictive. Next, we will introduce the **rejection sampling** method.

[1]The support of a random variable/function is the subset of the domain where the random variable takes a non-zero value.

[2]which can be easily done by the random number generation functions on modern day computers.

We still assume the target random variable X is continuous, and we denote its probability density function as f. Consider we can find another (easy to sample from, say via inverse CDF) random variable Y with probability density function g such that there exists a constant $M > 0$ such that $Mg(x) \geq f(x)$ on the support of X, then we can draw samples from X using samples from Y in the following way.

Algorithm 2 Rejection Sampling Algorithm

```
Require: Target random variable $X$ with density $f$. Proposal random variable $Y$ with density
    $g$. Constant $M > 0$ such that $Mg(x) \geq f(x)$ on the support of $X$.
1: for $i = 1, 2, \cdots$ do
2:     Draw $y \sim Y$.
3:     Draw $U \sim \text{Uniform}[0, 1]$.
4:     if $U < f(y)/(Mg(y))$ then
5:         Set $X_i = y$.
6:     else
7:         return to Step 2
8:     end if
9: end for
```

First, we will prove that the above algorithm is doing the right thing.

Proposition 2.2. *The output of Algorithm 2 are indeed samples from the target distribution.*

Proof. For a sample X_i to be accepted, it must be first drawn from Y and then accepted with probability $f(X_i)/(Mg(X_i))$. This means we have

$$\mathbb{P}(X_i = x, X_i \text{ is accepted}) = \mathbb{P}(Y = x)\frac{f(x)}{Mg(x)} = g(x) \cdot \frac{f(x)}{Mg(x)} = \frac{f(x)}{M}$$

and

$$\mathbb{P}(X_i \text{ is accepted}) = \int_x \mathbb{P}(X_i = x, X_i \text{ is accepted})dx = \int_x \frac{f(x)}{M}dx = \frac{1}{M}.$$

So, using the Bayes formula, we have

$$\mathbb{P}(X_i = x \mid X_i \text{ is accepted}) = \frac{\mathbb{P}(X_i = x, X_i \text{ is accepted})}{\mathbb{P}(X_i \text{ is accepted})} = f(x)$$

as desired. □

Rejection sampling will draw (possibly) multiple samples from the proposal distribution to get a sample from the target distribution. The number of draws one needs to get a sample is clearly highly linked to the value of M. This relationship is made precise in the following result.

Proposition 2.3. *The number of draws from the proposal distribution until one draw is accepted follows a geometric distribution with a mean of M.*

The above result is easy to see as each draw is independent, and the success probability is $1/M$ for each draw, and the rest follows from the definition of a geometric distribution.

Therefore, we should pick the proposal distribution such that it is easy to sample from, but more importantly it is uniformly (i.e. across values in the support) close to the target distribution, so we would have a small M. It is often not very easy to choose a good proposal distribution that satisfies both these properties, especially when the distributions are of high dimensions and complicated form which prevents direct visualisation.

2.3 Importance Sampling

As an improvement of the rejection sampling, we will introduce the **importance sampling** algorithm. In rejection sampling, the rejected samples are simply omitted, but they still could be recycled and be somewhat useful - maybe not as a full sample from the target but as a part sample.

Consider the setup where we have a continuous target random variable X with density f and a proposal random variable Y with density g. Assuming we further have the condition that $f(x) > 0 \implies g(x) > 0$[3], we have

$$\int f(x)dx = \int \frac{f(x)}{g(x)} g(x)dx =: \int w(x)g(x)dx$$

where we define $w(x) = f(x)/g(x)$. This means, assuming we have $h(x)f(x) > 0 \implies g(x) > 0$, we can make the following statement

$$\begin{aligned}\mathbb{E}_X[h(X)] &= \int h(x)f(x)dx = \int h(x)\frac{f(x)}{g(x)}g(x)dx \\ &= \int h(x)w(x)g(x)dx = \mathbb{E}_Y[h(Y)w(Y)] \approx \frac{1}{N}\sum_{i=1}^{N} h(Y_i)w(Y_i)\end{aligned}$$

where $\{Y_i\}_{i=1}^N$ are i.i.d. samples of Y. This allows us to keep all the drawn samples from the proposal random variable, and reweigh them accordingly to make sure they are all samples from the target random variable. This is obviously a more versatile sampling algorithm than the other two described above. A formal description of the rejection sampling is included below.

Algorithm 3 Importance Sampling Algorithm

```
Require: Target random variable $X$ with density $f$. Proposal random variable $Y$ with density
    $g$. Function $h$ of the target integral. Require $h(x)f(x) > 0 \implies g(x) > 0$. Sample size $N$.
1: for $i = 1, 2, \cdots, N$ do
2:     Draw $X_i \sim Y$.
3:     Compute weight $w_i = f(X_i)/g(X_i)$.
4: end for
5: return approximate $\frac{1}{N}\sum_{i=1}^{N} w_i h(X_i)$ of $\mathbb{E}_X[f(X)]$.
```

Using the interpretation of the Monte Carlo method as approximations to the target distribution, the importance sampling of Algorithm 3 outputs an approximation

$$f(x) \approx \sum_{i=1}^{N} \frac{w_i}{N}\delta_{X_i}(x).$$

2.3.1 Proposal Distribution Assessments

One would like to make statements about the quality of the proposal random variable (and therefore the output) from the importance sampling algorithm to help us decide which proposal random variable we should use. Unlike in the case of rejection sampling where the quality of the proposal random variable can be directly reflected by the simple quantity of the constant M, things are slightly more complicated here.

One metric people can use to assess the quality of the estimator is by computing the **effective sample size**. This concept is very commonly used in numerical experiments to compare the

[3]in the case of f, g being probability measures, we would require f to be absolutely continuous with respect to g, which means that for any measurable subset A of the space f, g is defined on, we have $f(A) > 0 \implies g(A) > 0$.

quality and effectiveness of various algorithms. In the context of importance sampling, the effective sample size of the output of Algorithm 3 would be defined as

$$\mathrm{ESS} = \frac{\left[\sum_{i=1}^{N} w_i\right]^2}{\sum_{i=1}^{N} w_i^2}.$$

It is not hard to notice that the value of ESS ranges between 1 and N. In the case where the proposed distribution is identical to the target distribution, each weight would be equal to 1, and we would get an ESS of N. In the case where the proposed distribution is very different from the target, we would expect most of the weights to be very small, and the ESS would be small too. In short, the larger the ESS the better. Intuitively, ESS denotes the value of the obtained samples in terms of independent samples from the target distribution. This should somewhat justify the use of ESS as a simple (and rough) indicator of the quality of the estimator.

Another way to assess the quality of the estimator is by looking at its variance. We know from the concept of the Monte Carlo method that such estimators are unbiased. If we denote the true integral to be I, and the importance sampling estimator with N samples to be $\tilde{I}_N$, then we have $\mathbb{E}[\tilde{I}_N] = I$ and for each sample $w(X_i)h(X_i)$, we have

$$\mathrm{Var}_Y[w(X_i)h(X_i)] = \mathbb{E}_Y[w^2(X_i)h^2(X_i)] - I^2 = \int g(x)\frac{f^2(x)}{g^2(x)}h^2(x)dx - I^2.$$

Using the Jensen's inequality[4], we have

$$\mathbb{E}_Y[w^2(X_i)h^2(X_i)] \geq \mathbb{E}_Y[w(X_i)|h(X_i)|]^2 = \left(\int g(x)\frac{f(x)}{g(x)}|h(x)|dx\right)^2 = \left(\int f(x)|h(x)|dx\right)^2$$

by noticing $k(x) = x^2$ is a convex function. Therefore, the choice of g that minimises the variance would match the lower bound, and satisfy the identity

$$\int g(x)\frac{f^2(x)}{g^2(x)}h^2(x)dx = \left(\int f(x)|h(x)|dx\right)^2$$

which gives us the **optimal proposal density** g_{opt} defined as

$$g_{\mathrm{opt}}(x) = \frac{|h(x)|f(x)}{\int |h(x)|f(x)dx}. \tag{4}$$

This result is merely a theoretical one, as obtaining this optimal density would require us to compute $\int |h(x)|f(x)dx$ which is as hard as the original task. This makes the problem a bit cyclical. The value of this result is mostly to set a goal for our selection of proposal density.

2.3.2 Unnormalised Distributions

A practical question to think about is we may only obtain an unnormalised version of the densities. This is common in Bayesian inference, for example, where the distributions of interest would be posterior distributions and the normalising constants are unknown. We will extend the importance sampling algorithm to cases where only unnormalised versions of the proposal and the target distribution are known.

Consider we have target X with density f, but we only know f_u defined by $f_u(x) = Kf(x)$ for unknown constant $K > 0$. Similarly, we assume we only know the unnormalised density g_u of the proposal Y with true density g such that $g_u(x) = Jg(x)$ for unknown constant $J > 0$. Importance sampling can still be conducted according to the following identity:

$$\mathbb{E}_X[h(X)] = \int h(x)f(x)dx = \frac{\int h(x)\frac{f(x)}{g(x)}g(x)dx}{\int \frac{f(x)}{g(x)}g(x)dx} = \frac{\int h(x)\frac{f_u(x)}{g_u(x)}g(x)dx}{\int \frac{f_u(x)}{g_u(x)}g(x)dx}.$$

[4]Jensen's inequality states that, for any convex function k, we have $\mathbb{E}[k(X)] \geq k(\mathbb{E}[X])$. A real-valued function k is **convex** if for any two points x, y in its domain, and $\alpha \in [0, 1]$, we have $k(\alpha x + (1-\alpha)y) \leq \alpha k(x) + (1-\alpha)k(y)$.

So, we can compute the weight using $w(x) = f_u(x)/g_u(x)$, and we just need to normalise it when computing the final estimate using a Monte Carlo approximation of the denominator above, which is

$$\mathbb{E}_X[h(X)] \approx \frac{\frac{1}{N}\sum_{i=1}^{N} h(Y_i)w(Y_i)}{\frac{1}{N}\sum_{i=1}^{N} w(Y_i)} = \sum_{i=1}^{N} \frac{w(Y_i)}{\sum_{j=1}^{N} w(Y_j)} h(Y_i) =: \sum_{i=1}^{N} W_i h(Y_i)$$

where we define the **normalised weight** $W_i := w(Y_i)/\sum_j w(Y_j)$. This version of the importance sampling is called the **auto-normalised importance sampling**. A formal algorithm is listed below, and its properties are identical to the standard importance sampling algorithm. The ESS of the auto-normalised would simply become

$$\text{ESS} = \frac{1}{\sum_{i=1}^{N}(W_i)^2}$$

by definition.

Algorithm 4 Auto-Normalised Importance Sampling Algorithm

```
Require: Target random variable $X$ with unnormalised density $f_u$. Proposal random variable $Y$
    with unnormalised density $g_u$. Function $h$ of the target integral. Require $h(x)f_u(x) > 0 \implies$
    $g_u(x) > 0$. Sample size $N$.
1: for $i = 1, 2, \cdots, N$ do
2:     Draw $X_i \sim Y$.
3:     Compute weight $w_i = f_u(X_i)/g_u(X_i)$.
4: end for
5: For each $i$, define the normalised weight $W_i = w_i/\sum_{j=1}^{N} w_j$.
6: return approximate $\sum_{i=1}^{N} W_i h(X_i)$ of $\mathbb{E}_X[f(X)]$.
```

Finally, using the interpretation of the Monte Carlo method as approximations to the target distribution, the auto-normalised importance sampling of Algorithm 4 outputs an approximation

$$f(x) \approx \sum_{i=1}^{N} W_i \delta_{X_i}(x).$$

Chapter 3

Hidden Markov Models

In this chapter, we will consider a class of models called **hidden Markov models** (HMM), also known as **state space models** (SSM). Hidden Markov models are extremely versatile and are able to capture time-dependent events where there is a true (Markovian) process - often called the signal process - that we cannot observe and a noisy process - often called the observation process - that depends on the true process which we can observe.

3.1 Examples of Hidden Markov Models

For example, if we wish to track a moving plane using radar stations, we can only make observations of the location of the moving plane using radar stations, say the distance and the angle, which need to be translated into the Euclidean coordinate system to reflect the 'true' location. This conversion needs to be done at each observation time. Formulated using an HMM, the signal process is the Euclidean location of the plane, while the observation process is the polar location of the plane according to the radar station. This is illustrated graphically in Figure 2.

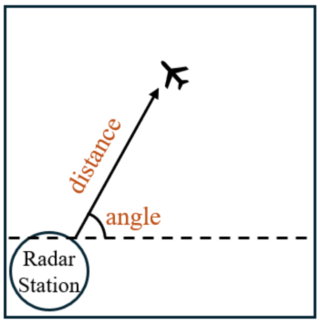

Figure 2: Tracking of plane movements using radar stations.

Another example of events that could be modelled by an HMM is the progression of infectious diseases in a closed (i.e. assuming no birth and death has happened) population. Using the framework of a compartmental model, we divide the population into disjoint compartments and model the change in the number of people in each compartment over time. For example, the SIR model divides the population into **S**uspectible (currently uninfected but can be infected), **I**nfectious (currently infected), and **R**emoved (immune to infection or dead after infection). Infection in this model happens with a fixed probability when a susceptible individual meets an infectious individual. The model dynamics assume that the population is well-mixed so each individual has an equal chance of being in contact with everyone else, and an equal chance of getting infected and an equal chance of getting removed. This leads to the following system of differential equations that can be used to model, in continuous time, the change in the number in each compartment. Assuming the population size is N, and we use S_t, I_t, R_t to denote the number of people in each of these three compartments at time t, we then have

$$\begin{cases} \frac{d}{dt}S_t &= -\beta \frac{I_t}{N} S_t \\ \frac{d}{dt}I_t &= \beta \frac{I_t}{N} S_t - \gamma I_t \\ \frac{d}{dt}R_t &= \gamma I_t \\ N &= S_t + I_t + R_t \end{cases}$$

where β is the infection rate constant and γ is the recovery rate constant. So far, we have described a physical model for the true underlying process. In reality, we would only be able to access noisy and partial versions of S_t, I_t, R_t through surveys or tests, which serve as the observation process.

3.2 Definition of Hidden Markov Models

It should be clear at this stage that hidden Markov models are useful modelling tools in many application areas, and are capable of incorporating expert knowledge about processes with observed data. We will now define an HMM formally.

Consider two discrete-time stochastic processes $\{X_t\}_t$ and $\{Y_t\}_t$. We let $\{X_t\}_t$ denote the signal process, and let $\{Y_t\}_t$ denote the observation process. We assume that each Y_t only depends on X_t, and the signal process is Markovian so that each X_t only depends on X_{t-1}. This relationship can be captured in the following illustration where the pointed arrows represent the conditional dependencies.

$$\begin{tikzcd}
\cdots \arrow[r, "P"] & X_{t-1} \arrow[r, "P"] \arrow[d, "g"] & X_t \arrow[r, "P"] \arrow[d, "g"] & X_{t+1} \arrow[r, "P"] \arrow[d, "g"] & \cdots & \text{(signal)} \\
\cdots & Y_{t-1} & Y_t & Y_{t+1} & \cdots & \text{(observation)}
\end{tikzcd}$$

We would describe the relationship between states using transition kernels and conditional distributions as labelled in the illustration. To simplify the notation, we would use $X_{a:b}$ to denote $(X_a, X_{a+1}, \dots, X_b)$. Also, we would use capital letters to denote random variables while using lowercase letters to denote their realisations. We have

$$\begin{aligned} X_0 &\sim \pi_0 \\ X_t \mid (x_{0:t-1}, y_{1:t-1}) &\sim P(dx_t|x_{t-1}) \\ Y_t \mid (x_{0:t}, y_{1:t-1}) &\sim g(y_t|x_t)d\nu(y_t) \end{aligned}$$

where π_0 is the initial distribution, P is the transition kernel of the signal process, and g is the conditional distribution of the observation process given the signal process. Note that P is a kernel and it does not need to admit a density, whereas g admits a density with respect to a reference measure ν which is usually the Lebesgue (or counting) measure.

There are four main tasks associated with a hidden Markov model like the one above: **predicting**, **filtering**, **smoothing**, and **parameter estimation**. In this section, we will give a brief description of them. Further studies will appear in later chapters.

The transition kernel P and the conditional distribution g usually depend on some parameters, and we denote the full vector of parameters by θ. The parameter dependency would not be made explicit most of the time to make the notation clean. In a Bayesian framework, we can think about an HMM as a Bayesian inference problem: π_0 is the prior distribution of the signal process, and as we make further observations $y_{1:t}$, we update our belief. The likelihood functions, denoted in general by p, are

$$p(x_{0:t}) = \pi_0(dx_0) \prod_{i=1}^{t} P(dx_i|x_{i-1}) \qquad p(y_{1:t}|x_{0:t}) = \prod_{i=1}^{t} g(y_i|x_i).$$

We can use the Bayes formula to get the posterior distribution of the signal process $X_{0:t}$ after observing $y_{1:t}$, which is given by

$$p(x_{0:t}|y_{1:t}) = \frac{p(x_{0:t}, y_{1:t})}{p(y_{1:t})} = \frac{p(x_{0:t})p(y_{1:t}|x_{0:t})}{\int p(y_{1:t}|x_{0:t})dx_{0:t}}.$$

The distribution $p(x_{0:t}|y_{1:t})$ above is called the **smoothing distribution**, and the task of finding it is called **(complete) smoothing**. Roughly speaking, this is the task of learning the distribution of the full trajectory of the signal process given all the available data.

Sometimes, we may be only interested in knowing the distribution of the current state in the signal process instead of the whole trajectory. We wish to find the conditional distribution of X_t given observations $y_{1:t}$, which is

$$\begin{aligned}
p(x_t|y_{1:t}) &= \frac{p(x_t, y_{1:t})}{p(y_{1:t})} \\
&= \frac{g(y_t|x_t)p(x_t|y_{1:t-1})p(y_{1:t-1})}{p(y_t|y_{1:t-1})p(y_{1:t-1})} \\
&= \frac{g(y_t|x_t)p(x_t|y_{1:t-1})}{p(y_t|y_{1:t-1})} \\
&= g(y_t|x_t)\frac{\int P(x_t|x_{t-1})p(x_{t-1}|y_{1:t-1})dx_{t-1}}{\int g(y_t|x_t)p(x_t|y_{1:t-1})dx_t}.
\end{aligned}$$

This distribution $p(x_t|y_{1:t})$ is called the **filtering** distribution, and the task of finding it is called **filtering**. Notice that on the right-hand side of the above equation, the integral in the denominator replies on knowing $p(x_t|y_{1:t-1})$, which is the full integral in the numerator - which depends on $p(x_{t-1}|y_{1:t-1})$. This indicates a highly iterative structure of the filtering distribution - the filtering distribution at time t depends on that at time $t-1$, which depends on that at time $t-2$, etc. Furthermore, the numerator of the right-hand side of the equation above is the **prediction distribution** $p(x_t|y_{1:t-1})$, which is essentially making the prediction of the next state in the signal distribution given all observations. The denominator $p(y_{1:t})$ of the right-hand side of the equation above is the **likelihood** of observing the data, which depends on the parameters θ. The likelihood would allow us to estimate θ, say using maximum likelihood estimation.

Therefore, we have described all four main tasks associated with an HMM. They are summarised below.

- **(predict)** Find $p(x_t|y_{1:t-1})$
- **(filter)** Find $p(x_t|y_{1:t})$
- **(smooth)** Find $p(x_{0:t}|y_{1:t})$
- **(parameter estimation)** Estimate θ using likelihood $p(y_{1:t})$

The first two tasks will be fully covered in Chapter 6 after some preparations in Chapters 4 and 5. The smoothing problem will be considered in Chapter 7 and the parameter estimation problem will be considered in Chapter 8.

Chapter 4

Sequential Importance Sampling

In this chapter, we will directly apply some of the sampling techniques of Chapter 2 to solve some of the tasks described in Chapter 3. The focus would be put on solving the filtering problem in particular, as it is the easiest to solve and serves as a foundation for solving the rest of the problem. This will be a starting point for studying a hidden Markov model, which will be extended in the next Chapter.

4.1 A First Attempt

To solve the filtering problem of HMM, we could apply some techniques from importance sampling and exploit the conditional structure of HMM. We will use the same setup as in Chapter 3, and we will put the following illustration again to indicate the HMM we are interested in.

$$\begin{tikzcd}
\cdots \arrow[r, "P"] & X_{t-1} \arrow[r, "P"] \arrow[d, "g"] & X_t \arrow[r, "P"] \arrow[d, "g"] & X_{t+1} \arrow[r, "P"] \arrow[d, "g"] & \cdots & \text{(signal)} \\
\cdots & Y_{t-1} & Y_t & Y_{t+1} & \cdots & \text{(observation)}
\end{tikzcd}$$

The things that we assume we know about the above HMM, other than its structure, are the initial distribution π_0 of X_0, the transition kernel $P(dx_t|x_{t-1})$ of the signal process, and the conditional distribution $g(y_t|x_t)$ of the observation given the signal. The dependency on parameter vector θ is omitted to simplify the notation.

Recall that in Chapter 2, we introduce the (auto-normalised) importance sampling where we draw N samples from an (unbiased) proposal distribution g_u that is supported on a set which includes the support of the (unbiased) target distribution f_u, and we compute the (auto-normalised) weight of each sample to obtain a Monte Carlo approximation to the target density f. We will use the same algorithm here.

Starting from π_0, we will draw N samples (which we will also call **particles**) from it, denote them as x_0^i and compute their respective (auto-normalised) weight $W_0^{(i)}$ for $i = 1, 2, \ldots, N$. Note that if we can sample directly from π_0, the weights would be $1/N$ each. If not, we will use the auto-normalised weight as stated in Algorithm 4.

We wish to extend our samples from X_0 to samples from X_1, which is a step called the **propagation step**. This is relatively easy as we know the exact transition kernel P. For each sample x_0^i, we will draw its propagated sample $x_1^i \sim P(\cdot|x_0^i)$. The weight for each sample will remain unchanged at this step, i.e. $\tilde{w}_1^i = W_0^i$.

The HMM would make observations (y_t), and we would adjust the weights of the particles accordingly - a step called the **assimilation step**. Due to the fact that

$$p(x_1|y_1) \propto g(y_1|x_1)p(x_1),$$

we should account for the $g(y_1|x_1)$ factor into our samples from X_1. This gives us the assimilated unnormalised weight

$$w_1^{(i)} = g(y_1|x_1^{(i)})\tilde{w}_1^i$$

and the assimilated normalised weight

$$W_1^{(i)} = w_1^{(i)} / \sum_{j=1}^{N} w_1^{(j)}.$$

The propagation and assimilation steps transform the original particle-weight pair $(x_0^{(i)}, W_0^{(i)})_i$ of X_0 to the updated particle-weight pair $(x_1^{(i)}, W_1^{(i)})_i$ of X_1 given the observation y_1. Note that the pairs will form Monte Carlo approximations, i.e.

$$\pi_0(dx) \approx \sum_{i=1}^{N} W_0^{(i)} \delta_{x_0^{(i)}}(dx) \qquad\qquad p(dx_1|y_1) \approx \sum_{i=1}^{N} W_1^{(i)} \delta_{x_1^{(i)}}(dx_1).$$

Repeating the propagation and assimilation steps recursively until we have run out of observations would give us a vanilla version of the **sequential importance sampling** algorithm, and a formal version of the algorithm is listed below as Algorithm 5.

Algorithm 5 Vanilla Sequential Importance Sampling Algorithm

```
Require: Number of particles $N$. Observations $y_{0:T}$. Transition kernel $P(dx_t|x_{t-1})$ of the signal process. Conditional distribution $g(y_t|x_t)$ of the observation given the signal. Initial distribution $\pi_0$ of $X_0$.
1: Draw samples $x_0^i \sim \pi_0(dx_0)$ and assign weights $W_0^{(i)} = 1/N$ for $i = 1, 2, \ldots, N$.
2: for $t = 1, 2, \cdots, T$ do
3:     (propagation) Draw samples $x_t^i \sim P(dx_t|x_{t-1}^i)$ and assign weights $\tilde{w}_t^i = W_{t-1}^i$ for $i = 1, 2, \ldots, N$.
4:     (assimilation) Update weights $w_t^{(i)} = g(y_t|x_t^{(i)})\tilde{w}_t^i$ for $i = 1, 2, \ldots, N$. Normalise weights by $W_t^{(i)} = w_t^{(i)} / \sum_{j=1}^{N} w_t^{(j)}$ for $i = 1, 2, \ldots, N$.
5: end for
6: return approximate $\sum_{i=1}^{N} W_T^{(i)} \delta_{x_T^{(i)}}(dx_T)$ of the filtering distribution $p(x_T|y_{0:T})$.
```

An important remark at this stage is that the prediction distribution can be approximated as a by-product of the above algorithm. At the propagation step, the particle-weight pairs $(x_t^i, \tilde{w}_t^i)_i$ can serve as Monte Carlo samples for the prediction distribution $p(x_t|y_{1:t-1})$.

4.2 The General Version

The sequential importance sampling, as the name may suggest, is doing importance samplings sequentially. Let us first make the importance sampling explicit.

At time 1, we have the particle-weight pair $(x_0^{(i)}, W_0^{(i)})$ that yields a distribution $p_0(\cdot)$ which approximates π_0, and an observation y_1. The target distribution at this point is $p(x_1|y_1)$. The proposal distribution is $P(x_1|x_0)p_0(x_0)$ as we propagate $x_0^{(i)}$ using the transition kernel P. This gives us

$$\frac{p(x_1|y_1)}{P(x_1|x_0)p_0(x_0)} \propto \frac{g(y_1|x_1)P(x_1|x_0)\pi(x_0)}{P(x_1|x_0)p_0(x_0)} = g(y_1|x_1)\frac{\pi(x_0)}{p_0(x_0)} = g(y_1|x_1)W_0$$

so we get the weight update rule of $w_1^{(i)} = g(y_1|x_1^{(i)})W_0^i$. The weights are then normalised since our distributions are unnormalised to yield $W_1^{(i)} = w_1^{(i)} / \sum_j w_1^{(j)}$.

At time 2, we have the particle-weight pair $(x_1^{(i)}, W_1^{(i)})$ that yields a distribution $p_1(\cdot|y_1)$ which approximates $p(x_1|y_1)$, and an observation y_2. The target distribution at this point is $p(x_2|y_{1:2})$, and the proposal distribution is $P(x_2|x_1)p_1(x_1|y_1)$ as we propagate $x_1^{(i)}$ using the transition kernel P. This gives us

$$\frac{p(x_2|y_{1:2})}{P(x_2|x_1)p_1(x_1|y_1)} \propto \frac{g(y_2|x_2)p(x_2|y_1)}{P(x_2|x_1)p_1(x_1|y_1)} = g(y_2|x_2)\frac{p(x_1|y_1)}{p_1(x_1|y_1)} = g(y_2|x_2)W_1$$

so we get the weight update rule of $w_2^{(i)} = g(y_2|x_2^{(i)})W_1^i$. The weights are then normalised since our distributions are unnormalised to yield $W_2^{(i)} = w_2^{(i)}/\sum_j w_2^{(j)}$. The same logic and derivation are used to establish the rest of the update rules.

Note that at each of the above steps, we are choosing the proposal distribution simply by propagating the previous target distribution using the transition kernel P. This is certainly not the optimal choice compared to the optimal proposal density of Equation (4) outlined in Chapter 2. This motivates using a more general formulation of the sequential importance sampling which uses an arbitrary transition kernel Q that admits density to propagate the particles. At time 1, for example, we have the particle-weight pair $(x_0^{(i)}, W_0^{(i)})$ that yields a distribution $p_0(\cdot)$ which approximates π_0, and an observation y_1. The target distribution at this point is $p(x_1|y_1)$. The proposal distribution becomes $Q(x_1|x_0)p_0(x_0)$ as we propagate $x_0^{(i)}$ using the transition kernel Q. This gives us

$$\frac{p(x_1|y_1)}{Q(x_1|x_0)p_0(x_0)} \propto \frac{g(y_1|x_1)P(x_1|x_0)\pi(x_0)}{Q(x_1|x_0)p_0(x_0)} = g(y_1|x_1)\frac{P(x_1|x_0)}{Q(x_1|x_0)}\frac{\pi(x_0)}{p_0(x_0)} = \frac{P(x_1|x_0)}{Q(x_1|x_0)}g(y_1|x_1)W_0$$

which means the weight update rule becomes

$$w_1^{(i)} = g(y_1|x_1^{(i)})\frac{P(x_1|x_0)}{Q(x_1|x_0)}W_0^i \qquad\qquad W_1^{(i)} = \frac{w_1^{(i)}}{\sum_j w_1^{(j)}}.$$

The same applies to all the other iterations. It is not hard to notice that we recover the vanilla sequential importance sampling by setting $Q = P$. A formal version of the general sequential importance sampling is listed below as Algorithm 6.

Algorithm 6 General Sequential Importance Sampling Algorithm

```
Require: Number of particles $N$. Observations $y_{0:T}$. Transition kernel $P(dx_t|x_{t-1})$ of the signal
    process. Conditional distribution $g(y_t|x_t)$ of the observation given the signal. Propagation
    kernel $Q(dx_t|x_{t-1})$. Initial distribution $\pi_0$ of $X_0$.
1: Draw samples $x_0^i \sim \pi_0(dx_0)$ and assign weights $W_0^{(i)} = 1/N$ for $i = 1, 2, \ldots, N$.
2: for $t = 1, 2, \cdots, T$ do
3:     (propagation) Draw samples $x_t^i \sim Q(dx_t|x_{t-1}^i)$ and assign weights $\tilde{w}_t^i = W_{t-1}^i$ for $i =$
    $1, 2, \ldots, N$.
4:     (assimilation) Update weights
           $$w_t^{(i)} = g(y_t|x_t^{(i)})\frac{P(x_t^{(i)}|x_{t-1}^{(i)})}{Q(x_t^{(i)}|x_{t-1}^{(i)})}\tilde{w}_t^i$$
    for $i = 1, 2, \ldots, N$. Normalise weights by $W_t^{(i)} = w_t^{(i)}/\sum_{j=1}^N w_t^{(j)}$ for $i = 1, 2, \ldots, N$.
5: end for
6: return approximate $\sum_{i=1}^N W_T^{(i)}\delta_{x_T^{(i)}}(dx_T)$ of the filtering distribution $p(x_T|y_{0:T})$.
```

We could certainly obtain some theoretical optimal transition kernel using the optimal proposal density result of Equation (4), which is hardly attainable in practice.

Chapter 5

Particle Degeneracy and Resampling

In this Chapter, we will implement the sequential importance sampling on a simple example to illustrate how it works concretely, and highlight one particular practical problem - particle degeneracy - in Section 5.1, as well as how one could remedy this problem using resampling in Section 5.3.

5.1 Particle Degeneracy

Let us apply sequential importance sampling on a toy example of a linear Gaussian model defined by

$$\begin{aligned} X_0 &\sim N(0,1) \\ X_t &= \alpha X_{t-1} + \sigma U_t \\ Y_t &= X_t + \sigma V_t \end{aligned} \tag{5}$$

for $t = 1, 2, \dots$ where U_t, V_t are i.i.d. standard Gaussians, α, σ are constants. The filtering of the above linear Gaussian model could in fact be solved using a special tool called the Kalman filter, which we will not use here. Instead, we will apply the vanilla sequential importance sampling of Algorithm 5 to illustrate concretely how this algorithm works.

We will use $N = 10$ samples, and we will generate observations using Equation 5 for 5 time steps. The transition kernel is

$$P(x_t|x_{t-1}) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x_t - \alpha x_{t-1}}{\sigma}\right)^2\right)$$

which is the density of $N(\alpha x_{t-1}, \sigma^2)$. The conditional distribution is

$$g(y_t|x_t) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{y_t - x_t}{\sigma}\right)^2\right)$$

which is the density of $N(x_t, \sigma^2)$. The initial distribution π_0 will be the standard normal distribution. For the linear Gaussian model of Equation (5), we set $\alpha = 0.9$ and $\sigma = 1$, and we will look at five timesteps.

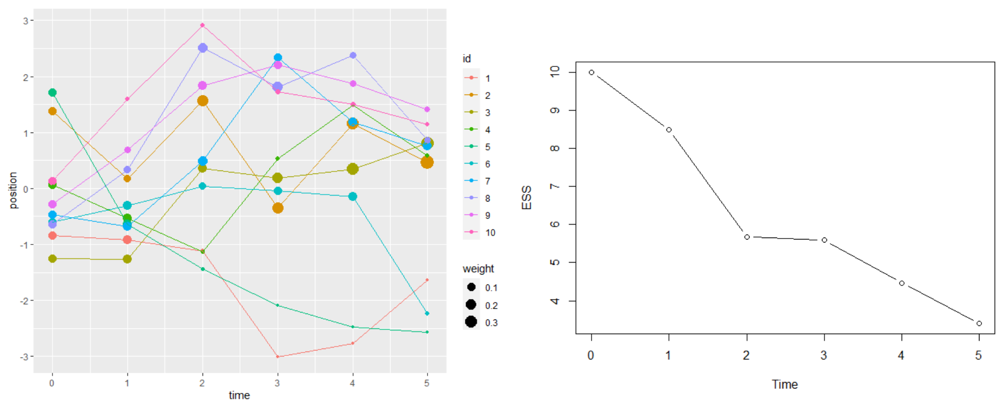

Figure 3: Filtering of a Linear Gaussian Model using Vanilla Sequential Importance Sampling. (Left) the trajectory of the particles over time with sizes representing weights. (Right) the effective sample size of the particles over time.

The result can be found in Figure 3. Notice that as time progresses, the weights are increasingly dominated by a few particles, and the effective sample size drops significantly, representing a drastic deterioration of the sample quality. The phenomenon of a small number of particles holding a majority of the weights is frequently called the **particle degeneracy**, or the **particle collapse**. This is quite a severe problem as illustrated in Figure 3, since it took only five timesteps for the ESS to drop from 10 to 3.

5.2 Particle Weights and Informative Observations

Here, we try to provide an intuition on why the ESS of particle weights decay as they did in the case of Figure 3. For importance sampling, as we have derived in Chapter 2, the quality of the proposal distribution is closely linked to how close it is to the target distribution - the closer, the better. The assessment we used to quantify (in an ad hoc manner) the quality of the proposal is also via ESS. In the case of the linear Gaussian example, we notice that at each iteration, the proposal distribution is just the propagation of the previous filter distribution, and the target distribution is that multiplies the information from the observation (up to the normalising constant). Therefore, the gap between the proposal and the target will deviate greatly if the information from the observation is significant, as one would hypothese.

To verify our hypothesis, at least using some experiments, we adjust the linear Gaussian example above by considering

$$
\begin{aligned}
X_0 &\sim N(0,1) \\
X_t &= \alpha X_{t-1} + \sigma U_t \\
Y_t &= \beta X_t + \sigma V_t
\end{aligned}
\tag{6}
$$

for $t = 1, 2, \ldots$ where U_t, V_t are i.i.d. standard Gaussians, α, σ, β are constants. This added β indicates the (linear) dependency between the observation and the signal. Notice that, for very big β, the value of Y_t would be dominated by the influence of the signal X_t, while for very small β, the value of Y_t would be dominated by the influence of the noise σV_t - when everything else remains unchanged. Figure 4 lists three graphs of the ESS (each starting at the same seed to make the comparison fair) for varying values of β: $\beta = 1/3, 1, 3$.

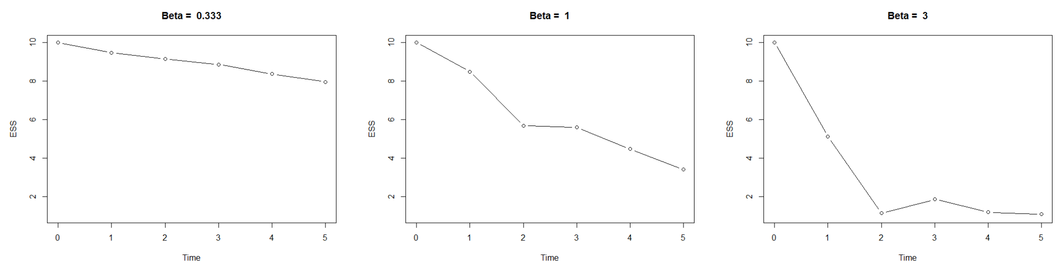

Figure 4: Effective sample sizes (ESS) over time for sequential importance sampling on linear Gaussian models with 10 particles. The signal and observation dependency coefficient β varies: (left) $\beta = 1/3$, (middle) $\beta = 1$, (right) $\beta = 3$.

The results align with our hypothesis. Therefore, as the observations are increasingly informative about our signals, the vanilla sequential importance sampling performs decreasingly well. In which case, the general sequential importance sampling of Algorithm 6 with a conditional distribution g informed propagation kernel would be preferred.

5.3 Resampling

A remedy to the problem of particle degeneracy is **resampling**. If at each iteration, we would resample/bootstrap the particles according to their weights, then we would get a collection of particles with balanced weights, which would yield a slower decay of ESS. Intuitively, we are removing particles with small weights and replicating particles with large weights. To ensure the particles still have diversity, we would do the resampling before propagation so that the randomness of the propagation step would recover some diversity. The key problem in resampling is which type of resampling is performed.

There are many types of resamplings - multinomial, residual, stratified, and systematic, to name a few. A more in-depth discussion on various resampling techniques and their comparisons can be found in Chapter 9 of Chopin and Papaspiliopoulos (2020). There, the authors recommend using **systematic resampling**, which we will discuss below.

Given N particles with normalised weights $(W_i)_{i=1}^N$, we wish to generate a N-vector of indices $(A_i)_{i=1}^N$ representing the resampled particles. The indices can be repeated. The starting point is to use an inverse CDF method to draw N samples from a multinomial distribution with weights according to $(W_i)_i$. We consider the cumulative weights

$$C_0 = 0 \qquad C_n := \sum_{i=1}^{n} W_i$$

for $n = 1, 2, \ldots, N$. Then, the inverse CDF methods would yield $(A_i)_i$ using N samples $(U_i)_i$ from Uniform$[0, 1]$ by setting

$$A_m = n \iff C_{n-1} \leq U_i \leq C_n.$$

The correctness of this method is supported by the inverse CDF method justification outlined in Chapter 2.

Building up on this, we would wish to use a better method such that the variance of the drawn samples is reduced. This can be achieved by using a quasi-Monte Carlo approach and the common random number.

Quasi-Monte Carlo uses similar ideas from Monte Carlo, where we still wish to use empirical distribution built by samples to approximate the target distribution. The difference is that Monte Carlo draws the samples randomly, whereas quasi-Monte Carlo uses evenly spaced out (and deterministic!) points to represent the samples. **Common random number** reduces the variance among samples directly by noticing a lot of the samples are drawn using a set of randomly generated values. If the same value is used for all of the randomly generated values, the variance of the samples would then be much smaller.

In this case, the randomness occurs from the N draws from Uniform$[0, 1]$. This can be improved by first dividing the $[0, 1]$ interval into N equally sized smaller intervals $[(n-1)/N, n/N]$ for $n = 1, 2, \dots, N$; and second drawing a uniform random variables in each of the small intervals. This guarantees the drawn samples are evenly spaced out to represent a more complete picture of the target distribution - ultimately improving the quality of the approximation. Next, instead of drawing each of the uniform random variables from the small intervals independently, we will draw only one, say from $[0, 1/N]$, and extend this to the other intervals. To be more precise, the N draws from the Uniform$[0, 1]$, denoted by $(U_i)_i$, will now be obtained as

$$U_1 \sim \text{Uniform}[0, 1/N], \qquad U_n = U_1 + (n-1)/N$$

for $n = 2, 3, \dots, N$. The draws are also ordered, as they are all increasing, so we would denote them by $(U_{(i)})_i$ to highlight that.

Combining them yields the following algorithm of systematic resampling.

Algorithm 7 Systematic Resampling

```
Require: Normalised weights $(W_i)_{i=1}^N$.
 1: Draw $U \sim \text{Uniform}[0, 1]$.
 2: Compute the cumulative weights $v_i = N \sum_{m=1}^{i} W_m$ for $i = 1, 2, \dots, N$.
 3: Set $s \leftarrow U$.
 4: Set $m \leftarrow 1$.
 5: for $i = 1, 2, \cdots, N$ do
 6:     while $v_i < s$ do
 7:         Set $m \leftarrow m + 1$.
 8:     end while
 9:     Set $A_i \leftarrow m$.
10:     Set $s \leftarrow s + 1$.
11: end for
12: return Indices $(A_i)_{i=1}^N$.
```

The outputted indices $(A_i)_i$ would be used to decide which particles are used to do propagation and assimilation. Using the same example in Section 5.1 but with resampling added at each sequential importance sampling iteration, we have the following result as shown in Figure 5. It is not hard to notice that the ESS deteriorates much more slowly than before in Figure 3.

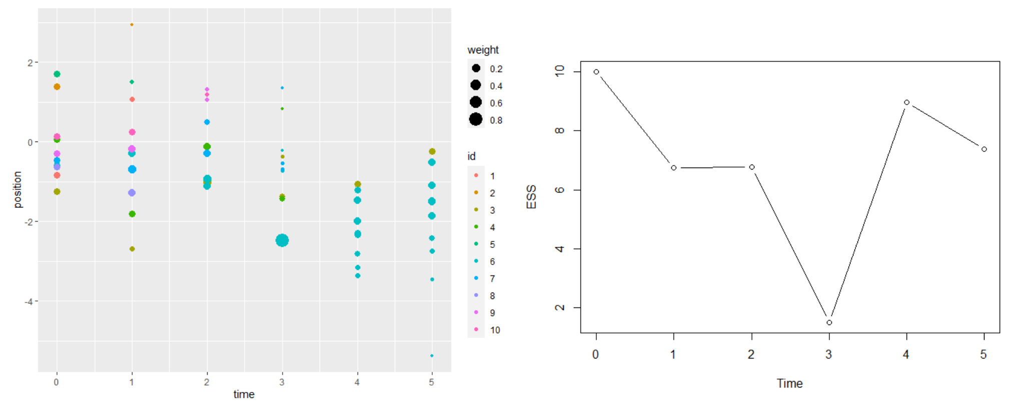

Figure 5: Filtering of a Linear Gaussian Model using Vanilla Sequential Importance Sampling with Systematic Resampling. (Left) the trajectory of the particles over time with sizes representing weights. The colour of the particles denotes the original index of the particle at $t = 0$. (Right) the effective sample size of the particles over time.

Chapter 6

Particle Filters: Bootstrap, Guided, and Auxiliary

In this chapter, we will combine the ideas derived from the previous chapters to introduce **particle filters**, also known as **sequential Monte Carlo** (SMC). There are three fundamental types of particle filters - bootstrap particle filter in Section 6.1, guided particle filter in Section 6.2, and auxiliary particle filter in Section 6.3.

6.1 Bootstrap Particle Filter

The **bootstrap particle filter**, also known as **sequential importance resampling**, is exactly the vanilla sequential importance sampling algorithm of Algorithm 5 with resampling at the start (before propagation) of each iteration. We will use by default the systematic resampling of Algorithm 7 in this report unless stated otherwise.

Resampling is beneficial as it stochastically removes particles with low weights and replicates particles with high weights, allowing future states of the processes to be better explored. However, resampling would always increase the Monte Carlo error of the approximation to the filtering distribution by introducing more variance. Therefore, a common ad hoc practice, which we would adopt here too, is to only trigger the resampling when the variance of particle weights is too high, assessed using ESS. We would set a threshold $\mathrm{ESS}_{\mathrm{min}}$, usually at $N/2$ where N is the number of particles, and trigger the resampling step whenever the ESS of the weights is lower than the threshold. This version of resampling is called **adaptive resampling**.

Combining the two, we have the following formal description of the bootstrap particle filter as Algorithm 8.

6.2 Guided Particle Filter

Just like we have developed the general sequential importance sampling of Algorithm 6 as a generalised of the vanilla sequential importance sampling of Algorithm 5 to incorporate the possibility of using more optimal proposal distribution, we can do the same to generalised the bootstrap particle filter. This gives us the **guided particle filter**, as formally described in Algorithm 9 below.

The reason why this algorithm is called the ‘guided’ algorithm is that, ideally, the propagation kernel Q is going to match the target filter distribution better, i.e. guiding our particles to move in the directions that give a higher weight. Recall that the bootstrap particle filter of Algorithm 8 completely ignores the observation when propagating the particles forward, here a good choice of propagation kernel Q should take the observation into account and direct the particles accordingly.

Algorithm 8 Bootstrap Particle Filter

```
Require: Number of particles $N$. Observations $y_{0:T}$. Transition kernel $P(dx_t|x_{t-1})$ of the signal process. Conditional distribution $g(y_t|x_t)$ of the observation given the signal. Initial distribution $\pi_0$ of $X_0$. Resampling algorithm resample (e.g. systematic resampling of Algorithm 7). Effective sample size threshold $\text{ESS}_{\min}$, with default being $N/2$.
 1: Draw samples $x_0^i \sim \pi_0(dx_0)$ and assign weights $W_0^{(i)} = 1/N$ for $i = 1, 2, \ldots, N$.
 2: Set index vector $A_0 = 1 : N$.
 3: for $t = 1, 2, \cdots, T$ do
 4:     if $\text{ESS}(W_{t-1}) < \text{ESS}_{\min}$ then
 5:         (resample) Set index vector $A_t = \text{resample}(W_{t-1})$.
 6:         Assign weights $\tilde{w}_t^i = 1$ for $i = 1, 2, \ldots, N$.
 7:     else
 8:         Set index vector $A_t = A_{t-1}$.
 9:         Assign weights $\tilde{w}_t^i = W_{t-1}^{A_t^i}$ for $i = 1, 2, \ldots, N$.
10:     end if
11:     (propagation) Draw samples $x_t^i \sim P(dx_t|x_{t-1}^{A_t^i})$.
12:     (assimilation) Update weights $w_t^{(i)} = g(y_t|x_t^{(i)})\tilde{w}_t^i$ for $i = 1, 2, \ldots, N$. Normalise weights by $W_t^{(i)} = w_t^{(i)} / \sum_{j=1}^N w_t^{(j)}$ for $i = 1, 2, \ldots, N$.
13: end for
14: return approximate $\sum_{i=1}^N W_T^{(i)} \delta_{x_T^{(i)}}(dx_T)$ of the filtering distribution $p(x_T|y_{0:T})$.
```

Algorithm 9 Guided Particle Filter

```
Require: Number of particles $N$. Observations $y_{0:T}$. Transition kernel $P(dx_t|x_{t-1})$ of the signal process. Conditional distribution $g(y_t|x_t)$ of the observation given the signal. Propagation kernel $Q(dx_t|x_{t-1})$. Initial distribution $\pi_0$ of $X_0$. Resampling algorithm resample (e.g. systematic resampling of Algorithm 7). Effective sample size threshold $\text{ESS}_{\min}$, with the default being $N/2$.
 1: Draw samples $x_0^i \sim \pi_0(dx_0)$ and assign weights $W_0^{(i)} = 1/N$ for $i = 1, 2, \ldots, N$.
 2: Set index vector $A_0 = 1 : N$.
 3: for $t = 1, 2, \cdots, T$ do
 4:     if $\text{ESS}(W_{t-1}) < \text{ESS}_{\min}$ then
 5:         (resample) Set index vector $A_t = \text{resample}(W_{t-1})$.
 6:         Assign weights $\tilde{w}_t^i = 1$ for $i = 1, 2, \ldots, N$.
 7:     else
 8:         Set index vector $A_t = A_{t-1}$.
 9:         Assign weights $\tilde{w}_t^i = W_{t-1}^{A_t^i}$ for $i = 1, 2, \ldots, N$.
10:     end if
11:     (propagation) Draw samples $x_t^i \sim Q(dx_t|x_{t-1}^{A_t^i})$.
12:     (assimilation) Update weights
            $$w_t^{(i)} = g(y_t|x_t^{(i)}) \frac{P(x_t^{(i)}|x_{t-1}^{A_t^i})}{Q(x_t^{(i)}|x_{t-1}^{A_t^i})} \tilde{w}_t^i$$
        for $i = 1, 2, \ldots, N$. Normalise weights by $W_t^{(i)} = w_t^{(i)} / \sum_{j=1}^N w_t^{(j)}$ for $i = 1, 2, \ldots, N$.
13: end for
14: return approximate $\sum_{i=1}^N W_T^{(i)} \delta_{x_T^{(i)}}(dx_T)$ of the filtering distribution $p(x_T|y_{0:T})$.
```

The local optimal propagation density exists too which we will state below without proof, although it is, again, hard to achieve in practice. Attempts to approximate the optimal density have been made, and we will delay such discussions to later chapters.

Proposition 6.1 (Local Optimality of Guided Particle Filters)**.** *Using the guided particle filter of Algorithm 9, the propagation kernel Q that minimises the variance is provided by*

$$Q_{opt}(dx_t|x_{t-1}) = \frac{g(y_t|x_t)P(dx_t|x_{t-1})}{\int g(y_t|x)P(dx|x_{t-1})}.$$

6.3 Auxiliary Particle Filter

The guided particle filter generalised the choice of propagation kernel from the bootstrap particle filter. The **auxiliary particle filter** makes another generalisation to the guided particle filter, in which the distribution used for the resampling is generalised too.

The weights used in the resampling of the two particle filters described so far are all directly about the particles. For particles $X_t^{(i)}$, we have the weight $W_t^{(i)}$. The auxiliary particle filter introduces a time-dependent auxiliary function η_t defined on the space of signal process states $\mathcal{X}$ and $\eta_t : \mathcal{X} \to \mathbb{R}^+$. This function helps us to change the weights into

$$\bar{w}_t^{(i)} = W_t^{(i)}\eta(X_t^{(i)})$$

which we then normalise to get

$$\bar{W}_t^{(i)} = \frac{\bar{w}_t^{(i)}}{\sum_{j=1}^N \bar{w}_t^{(j)}}.$$

The effect of this weight is to change the underlying particle weight distribution that we use to resample. If $\eta = 1$ and does nothing, then we recover the original resampling.

This change will influence the assimilation weight update too, in that we would need to multiply an extra ratio $W_t^{(i)}/\bar{W}_t^{(i)}$ to account for the above change. This gives us the following description of the auxiliary particle filter as Algorithm 10.

There exists a theoretical optimal choice of the auxiliary function η, as stated by the following result.

Proposition 6.2 (Local Optimality of Auxiliary Particle Filter)**.** *Using the auxiliary particle filter of Algorithm 10, at time $t-1$, assuming the auxiliary functions η_s for $s < t-1$ are fixed. For simplicity, we set $ESS_{\min} = N$. The auxiliary function η_{t-1} that minimised the variance at iteration t is*

$$\eta_{t-1}^{opt}(x_{t-1}) = \sqrt{\int Q(dx_t|x_{t-1})\left[\frac{g(y_t|x_t)P(dx_t|x_{t-1})}{Q(dx_t|x_{t-1})}\right]^2}.$$

As a remark, if we pick the propagation kernel Q as the optimal one as outlined in Proposition 6.1, the optimal auxiliary function would become

$$\eta_{t-1}^{\text{opt}}(x_{t-1}) = \int P(dx_t|x_{t-1})g(y_t|x_t)$$

which is the density of y_t condition on $X_t = x_{t-1}$.

It has also been observed that, at least numerically, the performance of the auxiliary particle filter deviates minimally from that of the guided particle filter.

6.4 Likelihood Approximation

In this short section, we will highlight how the likelihood of the observations can be approximated from the output of a particle filter algorithm. This will be used again in Chapter 8 when we consider parameter estimation.

Algorithm 10 Auxiliary Particle Filter

```
Require: Number of particles $N$. Observations $y_{0:T}$. Transition kernel $P(dx_t|x_{t-1})$ of the signal
    process. Conditional distribution $g(y_t|x_t)$ of the observation given the signal. Propagation
    kernel $Q(dx_t|x_{t-1})$. Initial distribution $\pi_0$ of $X_0$. Resampling algorithm resample (e.g. sys-
    tematic resampling of Algorithm 7). Auxiliary function $\eta_t$. Effective sample size threshold
    $\mathrm{ESS}_{\min}$, with the default being $N/2$.
 1: Draw samples $x_0^i \sim \pi_0(dx_0)$ and assign weights $W_0^{(i)} = 1/N$ for $i = 1, 2, \ldots, N$.
 2: Update weights $\bar{w}_0^{(i)} \leftarrow W_0^{(i)} \eta_0(X_0^{(i)})$ for $i = 1, 2, \ldots, N$.
 3: Normalise weights by $\bar{W}_0^{(i)} \leftarrow \bar{w}_0^{(i)} / \sum_{j=1}^N \bar{w}_0^{(j)}$ for $i = 1, 2, \ldots, N$.
 4: Set index vector $A_0 = 1 : N$.
 5: for $t = 1, 2, \cdots, T$ do
 6:     if $\mathrm{ESS}(W_{t-1}) < \mathrm{ESS}_{\min}$ then
 7:         (resample) Set index vector $A_t = \mathrm{resample}(W_{t-1})$.
 8:         Assign weights $\tilde{w}_t^i = W_{t-1}^{(i)} / \bar{W}_{t-1}^{(i)}$ for $i = 1, 2, \ldots, N$.
 9:     else
10:         Set index vector $A_t = A_{t-1}$.
11:         Assign weights $\tilde{w}_t^i = W_{t-1}^{A_t^i}$ for $i = 1, 2, \ldots, N$.
12:     end if
13:     (propagation) Draw samples $x_t^i \sim Q(dx_t|x_{t-1}^{A_t^i})$.
14:     (assimilation) Update weights

            $$w_t^{(i)} = g(y_t|x_t^{(i)}) \frac{P(x_t^{(i)}|x_{t-1}^{A_t^i})}{Q(x_t^{(i)}|x_{t-1}^{A_t^i})} \tilde{w}_t^i$$

        for $i = 1, 2, \ldots, N$. Normalise weights by $W_t^{(i)} = w_t^{(i)} / \sum_{j=1}^N w_t^{(j)}$ for $i = 1, 2, \ldots, N$.
15:     Assign weights $\bar{w}_t^i = W_t^{(i)} \eta_t(x_t^{(i)})$ for $i = 1, 2, \ldots, N$.
16:     Normalise weights by $\bar{W}_t^{(i)} \leftarrow \bar{w}_t^{(i)} / \sum_{j=1}^N \bar{w}_t^{(j)}$ for $i = 1, 2, \ldots, N$.
17: end for
18: return approximate $\sum_{i=1}^N W_T^{(i)} \delta_{x_T^{(i)}}(dx_T)$ of the filtering distribution $p(x_T|y_{0:T})$.
```

For any algorithms 8, 9, and 10 with N particles, we can define the approximated likelihood increment at iteration t by the following

$$l_t^N = \begin{cases} \frac{1}{N}\sum_{i=1}^N w_t^{(i)} & \text{if resampling happened} \\ \frac{\sum_{i=1}^N w_t^{(i)}}{\sum_{i=1}^N w_{t-1}^{(i)}} & \text{elsewise.} \end{cases}$$

This is the particle approximation of $p_t(y_t|y_{1:t-1})$. To obtain the full likelihood, as we have

$$p(y_{1:t}) = p(y_{1:t-1})p(y_t|y_{1:t-1}),$$

we have

$$L_t^N = \prod_{s=1}^t l_s^N. \tag{7}$$

It can be shown that this estimation is unbiased.

Chapter 7

Particle Smoothings

In Chapter 6, we have tackled two of the four main tasks of a hidden Markov model (predicting and filtering) as described in Chapter 3. In this chapter, we will attempt to tackle the problem of smoothing. The last task of parameter estimation will be looked at in the next chapter.

Observations of a hidden Markov model may arrive at once or sequentially. When all the observations are available at the same instance, the inferences would be classified as **offline** inference. When new observations constantly arrive when we are still processing existing data, the inferences would be classified as **online** inference. The key property of online problems is that one must process the data faster than the arrival of new data, to avoid backlogging. This leads to the constraint that the time complexity for algorithms tackling online problems should be at most linearly dependent on time T. It is not hard to realise that this is not a problem for filtering algorithms, as there is no time dependency in the time complexity of such algorithms - only particle number dependencies. This extra layer of practical obstacles requires more thought to be taken when designing smoothing algorithms in the online setting.

In this chapter, we will take a look at some common smoothing algorithms, in both online and offline settings. For online smoothing, we will cover the genealogy tracking, and for offline smoothing, we will cover the forward filtering backward sampling. Some immediate extensions of these two algorithms will be discussed too.

7.1 Online Smoothings

7.1.1 Genealogy Tracking and Block Sampling

The **full smoothing** problem aims to approximate the distribution of the full trajectory of the signal process given all the observations, i.e. approximate the distribution $p(x_{0:T}|y_{1:T})$. The filtering algorithms output N full trajectories of $x_{0:T}$ if we trace the particles back from $t = T$ to $t = 0$ using the ancestors of the resampled particles as the previous step. From the toy example of Figure 5, we can notice that the colours of the particles at time 5 are mostly the same - which means they share the same ancestor at time 0. This indicates the naive approach of using the filtering results to do smoothing is destined to yield a very high variance estimate of the full trajectory, especially in the earlier parts of the trajectory, due to particle degeneracy. This problem might not be as severe if we consider a **fixed-lag smoothing** instead of a full smoothing, i.e. consider a lag L and try to approximate the distribution $p(x_{T-L:T}|y_{1:T})$. For sufficiently small L, the particle degeneracy may not be as catastrophic. This approach is called **genealogy tracking** due to its constant tracking of the ancestors of the particles at each time. This approach turns the smoothing problem into a filtering problem.

An alternative and equivalent way to view the above algorithm is to consider creating a new process $\{z_t\}$ such that $z_t = x_{0:t}$ if we want to do full smoothing and $z_t = x_{t-L:t}$ if we want to do

L-lag smoothing. In the case of L-lag smoothing, the transition kernel of Z_t is defined as

$$M(dz'_t|z_{t-1}) = \delta_{x_{t-L:t-1}}(dx'_{t-L:t-1})P(dx'_t|x_{t-1})$$

where P is the transition kernel of the signal process $\{x_t\}$. This is the same as maintaining the bulk of the states to remain unchanged while doing a transition to the last state. The above transition kernel M of $\{z_t\}$ can be easily extended to the case of full smoothing. In this case, smoothing is the same as filtering for $\{z_t\}$ using any filtering algorithms we mentioned in Chapter 6, and its equivalence to the description in the above paragraph is obvious. The new process $\{z_t\}$ is called the lifted process, and this algorithm was proposed in **?**.

Algorithm 11 Genealogy Tracking

```
Require: Particle-weight pairs $(x_t^{(i)}, W_t^{(i)})$ for $t = 0, 1, \ldots, T$ and $i = 1, 2, \ldots, N$ from the output
    of a filtering algorithm. Ancestor indices $(A_t^{(i)})_{t\in 0:T-1, i\in 1:N}$ that denotes the index of the
    ancestor of particle $x_t^{(i)}$ at time $t-1$. By default, $A_0^{(i)} = i$.
1: Draw $B_T \sim \text{Multinomial}(W_T^{1:N})$.
2: for $t = T-1, T-2, \ldots, 0$ do
3:     Set $B_t \leftarrow A_t^{(B_{t+1})}$.
4: end for
5: return indices $B_0, B_1, \ldots, B_T$ that represents one simulated full smoothing trajectory
    $(X_t^{(B_t)})_{t=0}^T$.
```

The genealogy tracking algorithm, as stated formally as Algorithm 11, has particle degeneracy problems as we try to smooth earlier parts of the trajectory. **?** proposed a remedy to this problem in the case of fixed-lag smoothing by doing a **block sampling** of the lag segment at each time step to reduce the variance. The intuition behind this approach is that we would leverage information from the new observations to better approximate the previous states of the signal process by doing the extra block sampling. One should note that the direct implementation of this idea will run into a problem - at each iteration, the naive weight calculation involves an integral that is hard to compute. This can be remedied by extending the target distribution artificially to achieve a simple, recursive weight calculation formula. The details of the derivations as well as the full algorithm can be found in Section 3.1 and the Appendix of **?**.

A theoretical justification for doing fixed-lag smoothing is due to the forgetting properties of the hidden Markov model, which says that the influence of early states of the signal process influences the later states at a decreasing rate. Formally, for any x_0, x'_0 in the state space of the signal process and data $y_{1:n}$, there exist constants $B < \infty$ and $\lambda \in [0, 1)$ such that

$$\|p(dx_n|y_{1:n}, x_0) - p(dx_n|y_{1:n}, x'_0)\|_{\text{TV}} \leq B\lambda^n$$

where $\|\cdot\|_{\text{TV}}$ is the total variation distance between distributions. This result indicates that, if the particle filter uses the locally optimal proposal kernel, the influence of the initial condition decays at an exponential rate. Therefore, this result is often called the **exponential forgetting** property (**?**). The conditions required for this result to hold, in simple terms, are the signal process $\{x_t\}$ being a uniformly ergodic Markov chain, and the observation process $\{y_t\}$ are not too informative.

In the case of smoothing, we can apply the exponential forgetting property and argue for a fixed-lag smoothing as a close enough approximation for the full smoothing, as long as the lag L is sufficiently large. However, the fixed-lag smoothing via genealogy tracking (or block sampling) works well only for small L. This dilemma motivates further development of smoothing algorithms.

7.2 Offline Smoothings

7.2.1 Smoothing Skeleton

In the block sampling approach of online smoothing, we began thinking about using later observations to re-learn previous states of the signal process. This idea is fully extended in backward smoothing via the smoothing skeleton, by formally exploiting the backwards transition kernel of the signal process given all the observations.

We will first present the backward decomposition of the full smoothing distribution.

$$\begin{aligned}
&p(x_{0:T}|y_{1:T}) \\
&= p(x_T|y_{1:T})p(x_{0:T-1}|x_T, y_{1:T}) \qquad \textit{conditional prob.} \\
&= p(x_T|y_{1:T})p(x_{0:T-1}|x_T, y_{1:T-1}) \qquad \textit{dependency structure} \\
&= p(x_T|y_{1:T})p(x_{T-1}|x_T, y_{1:T-1})p(x_{0:T-2}|x_T, x_{T-1}, y_{1:T-1}) \qquad \textit{conditional prob.} \\
&= p(x_T|y_{1:T})p(x_{T-1}|x_T, y_{1:T-1})p(x_{0:T-2}|x_{T-1}, y_{1:T-2}) \qquad \textit{dependency structure} \\
&\vdots \\
&= p(x_T|y_{1:T}) \prod_{n=0}^{T-1} p(x_n|x_{n+1}, y_{1:n})
\end{aligned} \tag{8}$$

where $p(x_n|x_{n+1}, y_{1:n})$ is the backward transition kernel. The backward transition kernel can be computed using results from forward transition, as we have

$$p(x_n|x_{n+1}, y_{1:n}) = \frac{P(x_{n+1}|x_n)p(x_n|y_{1:n})}{p(x_{n+1}|y_{1:n})} \tag{9}$$

which consists of filtering distribution, propagation kernel, and predicting distribution - all can be approximated using outputs from a filtering algorithm.

If each of the terms in Equation (8) is replaced by its corresponding particle approximation, we would obtain the **smoothing skeleton**. The particle approximation of the backward transition kernel is obtained by the expression provided in Equation (9). One could notice that since each component of Equation (8) requires all N particles to approximate, the full backward decomposition of the full smoothing distribution is of $O(N^{T+1})$ which is quite computationally expensive to use for Monte Carlo approximations that depend on the full trajectory.

Another property about the backward decomposition of full smoothing is that it has a recursive formulation, as we have, by using Equations (8) and (9),

$$\begin{aligned}
p(x_{0:T+1}|y_{1:T+1}) &= p(x_{T+1}|y_{1:T+1}) \prod_{n=0}^{T} p(x_n|x_{n+1}, y_{1:n}) \\
&= p(x_{T+1}|y_{1:T+1})p(x_T|x_{T+1}, y_{1:T}) \prod_{n=0}^{T-1} p(x_n|x_{n+1}, y_{1:n}) \\
&= \frac{p(x_{T+1}|y_{1:T+1})p(x_T|x_{T+1}, y_{1:T})}{p(x_T|y_{1:T})} \cdot p(x_{0:T}|y_{1:T}) \\
&= \frac{p(x_{T+1}|y_{1:T+1})P(x_{T+1}|x_T)}{p(x_{T+1}|y_{1:T})} \cdot p(x_{0:T}|y_{1:T})
\end{aligned} \tag{10}$$

where the last step uses Equation (9). This allows us to build the backward decomposition recursively as the filtering algorithm proceeds.

7.2.2 Forward Filtering Backward Sampling

One smoothing algorithm by doing direct sampling of the smoothing skeleton using outputs from the filtering algorithms is the **forward filtering backward sampling** (FFBS) algorithm, as shown in the following pseudo-code of Algorithm 12. The weights are calculated by considering the recursion formulation of Equation (10). Generating M full smoothing trajectories using FFBS requires a cost of $O(TMN)$ where we use the outputs of a filtering algorithm with N particles and T time steps. These trajectories serve as i.i.d. Monte Carlo samples of smoothing distribution $p(x_{0:T}|y_{1:T})$, so are unbiased.

Algorithm 12 Forward Filtering Backward Sampling

```
Require: Particle-weight pairs $(x_t^{(i)}, W_t^{(i)})$ for $t = 0, 1, \dots, T$ and $i = 1, 2, \dots, N$ from the output
    of a filtering algorithm.
 1: Draw $B_T \sim \text{Multinomial}(W_T^{1:N})$.
 2: for $t = T-1, T-2, \dots, 0$ do
 3:     Set weights $\hat{w}_t^n \leftarrow W_t^n P(x_{t+1}^{(B_{t+1})} | x_t^{(n)})$ for $n = 1, 2, \dots, N$.
 4:     Normalise weights $\hat{W}_t^n \leftarrow \hat{w}_t^n / \sum_{m=1}^N \hat{w}_t^m$ for $n = 1, 2, \dots, N$.
 5:     Draw $B_t \sim \text{Multinomial}(\hat{W}_t^{1:N})$.
 6: end for
 7: return indices $B_0, B_1, \dots, B_T$ that represents one simulated full smoothing trajectory
    $(X_t^{(B_t)})_{t=0}^T$.
```

The above algorithm is a direct Monte Carlo and thus can be sped up using common sampling techniques. In Algorithm 12, the N factor in its complexity arises from the weight calculations for each iteration for t. This can be improved using rejection sampling if a proposal distribution can be found. In particular, at each time t, we are simulating from the distribution proportional to

$$\propto \sum_{n=1}^{N} W_t^n P(x_{t+1}^{(B_{t+1})} | x_t^{(n)}) \delta_{x_t^{(n)}}(dx_t).$$

If we can assume that $P(x_{t+1}|x_t) \leq C_t(x_t)$ for some known function C_t at all t, then we have

$$\sum_{n=1}^{N} W_t^n P(x_{t+1}^{(B_{t+1})} | x_t^{(n)}) \delta_{x_t^{(n)}}(dx_t) \leq \sum_{n=1}^{N} W_t^n C_t(x_t^{(n)}) \delta_{x_t^{(n)}}(dx_t).$$

and this allows us to construct a rejection sampling at each time t. The algorithm is presented as Algorithm 13.

Algorithm 13 Forward Filtering Backward Sampling with Rejection

```
Require: Particle-weight pairs $(x_t^{(i)}, W_t^{(i)})$ for $t = 0, 1, \dots, T$ and $i = 1, 2, \dots, N$ from the output
    of a filtering algorithm. Proposal function $C_t$ such that $P(x_{t+1}|x_t) \leq C_t(x_t)$ for all $t$.
 1: Draw $B_T \sim \text{Multinomial}(W_T^{1:N})$.
 2: for $t = T-1, T-2, \dots, 0$ do
 3:     Set weights $\hat{w}_t^n = W_t^n C_t(x_t^{(n)})$ for all $n = 1, 2, \dots, N$.
 4:     Normalise weights $\hat{W}_t^n \leftarrow \hat{w}_t^n / \sum_{m=1}^N \hat{w}_t^m$ for $n = 1, 2, \dots, N$.
 5:     Draw $b_t \sim \text{Multinomial}(\hat{W}_t^{1:N})$.
 6:     Draw $U \sim \text{Uniform}[0, 1]$.
 7:     if $U \leq P(x_{t+1}^{(B_{t+1})} | x_t^{b_t}) / C_t(x_t^{b_t})$ then
 8:         Set $B_t = b_t$.
 9:     else
10:         return to Step 5.
11:     end if
12: end for
13: return indices $B_0, B_1, \dots, B_T$ that represents one simulated full smoothing trajectory
    $(X_t^{(B_t)})_{t=0}^T$.
```

One could realise from standard results about rejection sampling stated in Chapter 2 that the efficiency of the rejection sampling depends on the quality of the proposal distribution. In this case, the rejection sampling version of FFBS may be faster, as we replaced the $O(N)$ portion in each iteration with a rejection sampling, which can be empirically monitored by keeping track of the acceptance probability of the rejection sampling - if these rates are not too low, the Algorithm 13 is likely to be much faster than Algorithm 12.

A fundamental problem with the two algorithms 12 and 13 is that any Monte Carlo computation using them would be $O(N^{T+1})$. To remedy this problem, we will look at the marginal smoothing distribution $p(x_n|y_{1:T})$ for all n rather than the full smoothing distribution. This will ease the Monte Carlo computation burden later on as the marginal approach is $O(N^2)$ rather than $O(N^{T+1})$, which will be obvious after we present the algorithm.

The marginal approach is based on the following derivation of the marginal smoothing distribution.

$$\begin{aligned}
p(x_n|y_{1:T}) &= \int p(x_n|x_{n+1}, y_{1:T})p(x_{n+1}|y_{1:T})dx_{n+1} && \textit{Chapman-Kolmogorov} \\
&= \int p(x_n|x_{n+1}, y_{1:n})p(x_{n+1}|y_{1:T})dx_{n+1} && \textit{dependency structure} \\
&= \int p(x_n|y_{1:n})\frac{P(x_{n+1}|x_n)}{p(x_{n+1}|y_{1:n})}p(x_{n+1}|y_{1:T})dx_{n+1} && \textit{Equation (9)} \\
&= p(x_n|y_{1:n})\int \frac{P(x_{n+1}|x_n)p(x_{n+1}|y_{1:T})}{p(x_{n+1}|y_{1:n})}dx_{n+1}.
\end{aligned} \tag{11}$$

The above derivation tells us how to weigh the samples correctly to account for further observations. Notice that it depends on $p(x_{n+1}|y_{1:T})$, which suggests a backward recursion. The full algorithm is listed below as Algorithm 14.

Algorithm 14 Forward Filtering Backward Sampling for Marginals

```
Require: Particle-weight pairs $(x_t^{(i)}, W_t^{(i)})$ for $t = 0, 1, \dots, T$ and $i = 1, 2, \dots, N$ from the output
    of a filtering algorithm.
 1: Set $W_{T|T}^m \leftarrow W_T^m$ for $m = 1, 2, \dots, N$.
 2: for $t = T, T-1, \dots, 1$ do
 3:     for $n = 1, 2, \dots, N$ do
 4:         Set $S_t^n \leftarrow \sum_{l=1}^N W_{t-1}^l P(x_t^n|x_{t-1}^l)$.
 5:         for $m = 1, 2, \dots, N$ do
 6:             Set $W_{t-1:t|T}^{m,n} \leftarrow W_{t|T}^n W_{t-1}^m P(x_t^n|x_{t-1}^m)/S_t^n$.
 7:         end for
 8:     end for
 9:     Set $W_{t|T}^m \leftarrow \sum_{n=1}^N W_{t:t+1|T}^{m,n}$ for $m = 1, 2, \dots, N$.
10: end for
11: Set $W_{0|T}^m \leftarrow \sum_{n=1}^N W_{0:1|T}^{m,n}$ for $m = 1, 2, \dots, N$.
12: return smoothing weights $W_{t|T}^n$ for particle $x_t^{(n)}$ and pairwise smoothing weights $W_{t-1:t|T}^{m,n}$ for
    particles $x_{t-1}^{(m)}, x_t^{(n)}$.
```

The complexity of Algorithm 14 is therefore clearly $O(N^2)$ due to the double for loops. The resulting smoothing distribution approximations would be helpful for direct marginal Monte Carlo approximations, and it does a speed up to the basic FFBS algorithm of Algorithm 12 with no extra condition, unlike the FFBS with rejection algorithm of Algorithm 13.

As a side remark, one could notice that the FFBS algorithm generates the backward trajectory without considering the filtering genealogy - whereas the filtering genealogy is all the things one considers for genealogy tracking of Algorithm 11.

Chapter 8

Parameter Estimations

One of the key problems of inferences with hidden Markov models, which we have ignored so far, is that of parameter estimations. It is often the case where the transition kernel of the truth process and the emission / observation distribution of the observation process are dependent on some parameters which we do not know and need to estimate. This is the problem of parameter estimations, which is the central topic of this chapter.

There are two main approaches to parameter estimations in Statistics - the (frequentist) maximum likelihood approach and the Bayesian approach. Both approaches will be discussed, and the focus will be put more on the Bayesian approach as it is just much better here due to its ability to capture more information about the data and produce more comprehensive outputs.

Consider the standard hidden Markov model with truth process $\{X_t\}$ and the observation process $\{Y_t\}$. The transition kernel of the truth process is denoted by P and the observation distribution is denoted by g and we assume it admits a density. Both P and g depend on parameters, and we will use θ to denote the vector of parameters for both P and g. We will sometimes use P_θ and g_θ notation to highlight the parameters, but most of the time we will drop them for simplicity. The likelihood of the full HMM is $p_\theta(y_{1:t})$ as we are making all the inferences using our observations. All of these are just revisions of what was covered in Chapter 3.

8.1 Maximum Likelihood Approaches

In the following, we will present three approaches to find the θ that maximises the likelihood function $p_\theta(y_{1:t})$.

8.1.1 MLE using Direct Particle Filter Outputs

Notice that the likelihood function can also be written as

$$\begin{aligned} p(y_{1:t}) = p(y_{1:t-1})p(y_t|y_{1:t-1}) &= p(y_{1:t-1}) \int p(x_{t-1:t}, y_t|y_{1:t-1})dx_{t-1:t} \\ &= p(y_{1:t-1}) \int p(x_{t-1}|y_{1:t-1})P(x_t|x_{t-1})g(y_t|x_t)dx_{t-1:t} \end{aligned}$$

which is of a recursive form that allows us to do particle approximations using particle filter algorithms. Recall from Section 6.4, we can approximate the above quantity using L_t^N as defined in Equation (7).

So, for any fixed θ, we can run a full particle filter algorithm and generate a likelihood using Equation (7). Then, in order to maximise the likelihood, we can do stochastic optimisation and just search through enough points in the parameter space and pick out the best. Some tricks like common random numbers are commonly used to speed up stochastic optimisations, but they are ultimately not helpful here due to various reasons. Ultimately, this approach can only be feasible using the naive, straightforward method of setting a very high N and searching through many possible values of θ.

8.1.2 MLE using Gradient Ascent

Optimisation can be done using gradient ascent/descent, and we can try this approach here. We wish to maximise $p(y_{0:t})$, or the log of it, and we need to access its gradient $\nabla \log p(y_{0:t})$ for gradient descent to work. We have the following trick to rewrite the gradient as an expectation, which is known as the **Fisher identity**.

Proposition 8.1 (Fisher Identity)**.** *For a hidden Markov model with transition kernels and observation distributions admitting differentiable density and are regular enough such that we can interchange differentiation (w.r.t. parameters) and integration (w.r.t. states), we have*

$$\nabla_\theta p_\theta(y_{1:t}) = \mathbb{E}_{P_\theta}\left[\nabla_\theta \log p_\theta(X_{0:t}, y_{1:t}) | Y_{1:t} = y_{1:t}\right].$$

Proof. For simplicity, we will drop all dependencies of θ and all differentiation would be w.r.t. θ unless stated otherwise.

First, we know $p(y_{1:t}) = \int p(x_{1:t}, y_{1,t}) dx_{1:t}$. Then, we have

$$\begin{aligned}
\nabla \log p(y_{1:t}) &= \frac{\nabla p(y_{1:t})}{p(y_{1:t})} = \frac{\nabla \int p(x_{1:t}, y_{1,t}) dx_{1:t}}{p(y_{1:t})} \\
&= \frac{\int \nabla p(x_{1:t}, y_{1,t}) dx_{1:t}}{p(y_{1:t})} = \frac{\int \nabla \log p(x_{1:t}, y_{1,t}) \cdot p(x_{1:t}, y_{1,t}) dx_{1:t}}{p(y_{1:t})} \\
&= \int \nabla \log p(x_{1:t}, y_{1,t}) \frac{p(x_{1:t}, y_{1,t})}{p(y_{1:t})} dx_{1:t} \\
&= \int \nabla \log p(x_{1:t}, y_{1,t}) p(x_{1:t} | y_{1,t}) dx_{1:t}
\end{aligned}$$

as desired. $\square$

Notice further that

$$\nabla \log p(x_{1:t}, y_{1,t}) = \nabla p_0(x_0) + \sum_{i=1}^{t} \nabla \log P(x_i | x_{i-1}) + \sum_{i=0}^{t} \nabla \log g(y_i | x_i)$$

based on standard HMM decomposition. This allows us to use outputs from particle smoothing algorithms discussed in Chapter 7 as Monte Carlo estimates to assist gradient approximations.

8.1.3 MLE using EM

Expectation-Maximisation (EM) algorithm is an iterative procedure to do maximum likelihood estimation whenever there exists a missing/latent data structure. For example, if we wish to maximise the likelihood $p_\theta(y)$ that could be written as

$$p_\theta(y) = \int p_\theta(x, y) dx = \int f_\theta(y|x) p_\theta(x) dx.$$

The hidden Markov model has the same latent structure.

We will first consider the standard EM, and then consider how one could apply it to the case of HMM parameter estimation.

The EM algorithm, as the name may suggest, contains an expectation step and a maximisation step. A full iteration of EM goes through both steps. To be more precise, we have, for starting point θ_0, we recursively do for $n = 1, 2, \ldots$

(E) Define $Q(\theta, \theta_{n-1}) = \mathbb{E}_{X\sim\mathbb{P}_{\theta_{n-1}}}[\log p_\theta(X, y)|Y = y]$ where $\mathbb{P}_{\theta_{n-1}}$ is the distribution of X with parameter θ_{n-1}.

(M) Find $\theta_n = \arg\max_\theta Q(\theta, \theta_{n-1})$.

To justify the EM algorithm, we first have the following result.

Proposition 8.2. *For some function $c(\theta)$, we have*

$$Q(\theta, \theta') + c(\theta') \leq \log p_\theta(y)$$

for all θ in the parameter space with equality only when $\theta = \theta'$, assuming parameter θ is identifiable.

Proof. Consider the KL divergence between $p_{\theta'}(x|y)$ and $p_\theta(x|y)$, which is

$$\begin{aligned}\mathrm{KL}(p_{\theta'}(x|y)\|p_\theta(x|y)) &= \mathbb{E}_{p_{\theta'}}[\log p_{\theta'}(x|y) - \log p_\theta(x|y)]\\ &= \mathbb{E}_{p_{\theta'}}[\log p_{\theta'}(X|y)] - \mathbb{E}_{p_{\theta'}}[\log p_\theta(X, y) - \log p_\theta(y)]\\ &= \mathbb{E}_{p_{\theta'}}[\log p_{\theta'}(X|y)] - Q(\theta, \theta') + \log p_\theta(y).\end{aligned}$$

The above quantity is non-negative by the definition of the KL divergence, and it will only be zero when $\theta = \theta'$ since it is identifiable. Some rearrangement and setting the first term of the last line of the above equation to $-c(\theta')$ will give us the desired inequality. □

The consequence of the above result is that maximising $Q(\theta, \theta')$ w.r.t θ is as good as maximising $\log p_\theta$, and thus we have transformed the maximum likelihood estimator to maximising Q, as prescribed in the EM algorithm.

The next thing to consider is how one could actually compute Q. There exists a nice expression in the case of $p_\theta(x, y)$ being from the exponential family, i.e.

$$p_\theta(x, y) = \exp[\theta^T s(x, y) - \kappa(\theta) - \xi(x, y)].$$

In this case, we have

$$\begin{aligned}\arg\max_\theta Q(\theta, \theta') &= \arg\max_\theta \mathbb{E}_{X\sim\mathbb{P}_{\theta'}}[\log p_\theta(X, y)|Y = y]\\ &= \arg\max_\theta \mathbb{E}_{X\sim\mathbb{P}_{\theta'}}[\theta^T s(X, y) - \kappa(\theta) - \xi(X, y)|Y = y]\\ &= \arg\max_\theta \mathbb{E}_{X\sim\mathbb{P}_{\theta'}}[\theta^T s(X, y) - \kappa(\theta)|Y = y]\\ &= \arg\max_\theta \theta^T \mathbb{E}_{X\sim\mathbb{P}_{\theta'}}[s(X, y)|Y = y] - \kappa(\theta).\end{aligned}$$

The maximisation can be done by taking the derivative and finding a zero, which is provided by finding θ that solves

$$\nabla_\theta \kappa(\theta) = \mathbb{E}_{X\sim\mathbb{P}_{\theta'}}[s(X, y)|Y = y]$$

In the case where such a nice form is not available, we can still do numerical maximisation. One should note that global (or even fully local) optimisation is not needed because as long as we are improving Q, the whole EM algorithm will converge quickly.

To use this for the case of HMM, we can simply substitute the right distributions and certain particle approximations using outputs from a particle smoothing algorithm can be applied, in a similar way as that of the last two maximum likelihood approaches.

8.1.4 Online MLE

The three approaches mentioned so far are all assuming we are in the offline setting, where all data are ready when we start analysing them. There are certain scenarios where this is not the case, and we need to process the data as new data points arrive. Most of the above methods can all be adapted to the online setting by doing stochastic approximations. The details are omitted for now.

8.2 Bayesian Approaches: A First Attempt

The Bayesian approach to parameter estimation is often of the following form. Consider the parameter of interest θ. We set a prior distribution $p(\theta)$, and based on the likelihood $p_\theta(y_{1:t})$ computed using the observations $y_{1:t}$ we can have the following posterior distribution

$$p(\theta|y_{1:t}) \propto p(\theta)p_\theta(y_{1:t})$$

which captures the full picture of the parameter θ. We can compute certain summary statistics using the posterior such as posterior mean and posterior mode, and some of them will recover familiar frequentist estimators. More importantly, the Bayesian approach of recursively updating our belief of the parameter of interest θ using observations $y_{1:t}$ fits perfectly to the data structure of HMM. In this case, we would also need to consider the truth process $\{X_t\}$ as they are unknown to us too. So the target posterior is likely to be

$$p(x_{0:t}, \theta|y_{1:t}).$$

The likelihood mentioned above is certainly of a very complicated form and will be impossible to compute exactly. Computational tools are needed to tackle this problem (often known as intractable likelihood). We want to learn about the posterior distribution and do some computations about it. It will be hard to do so directly, but we could use Monte Carlo samples from the posterior distribution to approximate the posterior distribution and compute any interesting integrals using the Monte Carlo approximations instead.

8.2.1 A First Attempt

The naive approach of doing Bayesian parameter estimation for HMM is to consider the parameter θ as a part of the truth state, so we have $\{\theta, X_t\}$ as the truth process instead where θ starts from a sample from the prior distribution. To obtain an estimate of θ requires only getting the particle approximation of it at each particle filter iteration. This approach is clearly online when we use a particle filter that can be implemented in an online fashion.

The problem with this naive implementation that we will certainly encounter is about the evolution of the θ state and the quality of samples of θ. Here we are not doing any evolution to θ as we are treating it as fixed. Consequently, the particle degeneracy with θ will be very severe.

Randomness needs to be included in the θ part of the truth process, and there are two main approaches.

The first is to add a small random noise, like adding a centred Gaussian. There is some evidence that this approach will do well in some cases, but theoretical guarantees are lacking and there are a few tuning parameters which could be problematic to tune.

The second is to use Markov chain Monte Carlo moves with θ at each iteration while maintaining the right target distribution. This method does not fully avoid the particle degeneracy problem either.

Regardless, neither of the two methods is too promising. The straightforward approach of doing Bayesian inference is to directly tackle the full posterior distribution, and aim to obtain samples from it to do Monte Carlo approximations. In the section below, we will present a class of methods that do exactly that.

8.3 Particle MCMC

In this section, we will consider a special type of Markov chain Monte Carlo (MCMC) method, called particle MCMC, that allows us to draw samples from the target posterior distribution so that we can use those samples to do Monte Carlo approximations. This type of MCMC is special as it leverages the structure of the target posterior. More specifically, since the posterior is proportional to the product of a prior and the likelihood, while the likelihood is that of an HMM, we can leverage all the particle filter algorithms to allow us to estimate the likelihood using things such as Equation (7).

8.3.1 A Brief Introduction to MCMC

In Chapter 2, we have described the Monte Carlo method of approximating expectations using samples from the distribution. The main problem of Monte Carlo is about getting the samples, and some ways to do so have been outlined in Chapter 2 such as importance sampling and rejection sampling.

Another way of obtaining those samples is to construct an ergodic Markov chain with the equilibrium distribution being the target distribution. A Markov chain is ergodic when it is aperiodic, irreducible, and reversible. A consequence of an ergodic Markov chain is that it will converge in the long run, in the sense that the distribution of the states of the Markov chain will converge to some probability distribution. This property inspires us to design a Markov chain with the right equilibrium distribution so that drawing samples from that equilibrium distribution is the same as running the Markov chain (from any initial position) for long enough and keeping the states of the chain after it converges. More details can be found in standard texts on MCMC, such as Robert and Casella (1999).

The question now becomes, how could one construct such a chain that has the right equilibrium. There are two main constructions - the Gibbs algorithm and the Metropolis-Hastings algorithm. To set the notation, we will denote the target distribution as π and we assume it admits density.

One of the first MCMC algorithms is the **Metropolis-Hastings Algorithm** (MH). Given a target distribution $\pi \in \mathbb{R}^n$ for some dimension n, a proposal kernel $Q(\cdot,\cdot)$ with $Q(x, A) = \int_A q(x, y)dy$ where $q(x, y)$ is the probability of moving from x to y, and a starting position x, we have the following algorithm.

Algorithm 15 Metropolis-Hastings Algorithm

```
Require: Target distribution $\pi$, proposal kernel $q(\cdot,\cdot)$, starting position $x$
 1: $X_0 = x$
 2: for $i = 0, 1, 2, \cdots$ do
 3:     $X_{curr} = X_i$
 4:     $X_{prop} \sim q(X_{curr}, \cdot)$
 5:     Draw $U \sim Unif[0, 1]$
 6:     if $\alpha(X_{curr}, X_{prop}) < U$ then
 7:         Accept $X_{prop}$ and $X_{i+1} = X_{prop}$.
 8:     else
 9:         Reject $X_{prop}$ and $X_{i+1} = X_{curr}$.
10:     end if
11: end for
```

Here, $\alpha(x, y) = \min\left\{1, \frac{\pi(y)q(y,x)}{\pi(x)q(x,y)}\right\}$ is the acceptance probability. The above algorithm will output a sequence $\{X_n\}$, and under some conditions on Q the distribution of X_n will converge to π. The part of the above algorithm that decides whether or not we should keep the proposed move is commonly referred to as the **Metropolis adjustment**.

We would want the transition kernel to have π as its invariant measure / distribution. It turns out that if the kernel satisfies **the detailed balance equation(s)**, the kernel will be π-reversible, or simply **reversible**, and the π-invariance is guaranteed. Recall that for a Markov chain with transition kernel P to be π-invariant, it means that we have

$$\int_x \pi(dx)P(x, dy) = \pi(dy).$$

Definition 8.3. *A Markov chain with transition kernel P is* ***π-reversible*** *for some distribution π when it satisfies the* ***detailed balance equation(s)***

$$\pi(dx)P(x,dy) = \pi(dy)P(y,dx)$$

for all possible x, y.

Proposition 8.4. *If a Markov chain with transition kernel P is π-reversible, then it is π-invariant.*

Proof. Using the detailed balance equation, we have

$$\int_x \pi(dx)P(x,dy) = \int_x \pi(dy)P(y,dx) = \pi(dy)\int_x P(y,dx) = \pi(dy),$$

as desired. □

So, if a Markov chain with transition kernel P satisfies the detailed balance equation, it would be π-reversible and therefore π-invariant, and this means, given that the chain is ergodic[1], the chain has π as its equilibrium measure / distribution, which is extremely desirable.

Theorem 8.5. *The Metropolis-Hastings algorithm, as constructed in Algorithm 15, produces a Markov chain $\{X_n\}_{n\in\mathbb{N}}$ that is π-reversible if target π and proposal kernel Q admit densities.*

Proof. We just need to show that the transition kernel $P(x, dy)$ of the algorithm satisfies the detailed balance equation

$$\pi(dx)P(x,dy) = \pi(dy)P(y,dx).$$

The equation is trivial when $x = y$, so we will only consider $x \neq y$. We have

$$\begin{aligned}
\pi(dx)P(x,dy) &= \pi(dx)\left[Q(x,dy)\alpha(x,y) + \delta_x(dy)\int(1-\alpha(x,u))Q(x,du)\right] \\
&= [\pi(x)dx][q(x,y)dy]\alpha(x,y) \\
&= [\pi(x)dx][q(x,y)dy]\min\left\{1, \frac{\pi(y)q(y,x)}{\pi(x)q(x,y)}\right\} \\
&= \min\left\{\pi(x)q(x,y), \pi(y)q(y,x)\right\} dxdy \\
&= \pi(y)q(y,x)\min\left\{\frac{\pi(x)q(x,y)}{\pi(y)q(y,x)}, 1\right\} dxdy \\
&= \pi(dy)Q(y,dx)\alpha(y,x) = \pi(dy)P(y,dx),
\end{aligned}$$

as desired. □

From Algorithm 15, we can choose any sensible proposal kernel and that would give us an MH algorithm. Two common choices are the symmetric Gaussian $N(0, h^2)$ where h is the tuning parameter, and the Gaussian with a drift towards the gradient $N(\nabla \log \pi(x), 2h^2)$. The first choice yields the **random walk Metropolis** (RWM), whereas the second choice yields the **Metropolis adjusted Langevin algorithm** (MALA).

Next, we introduce the Gibbs algorithm/sampler. The idea of the Gibbs sampler is that, for an n-dimensional target distribution f, we could find a series of sub-distributions f_i such that $f = \prod_i f_i$. Then, instead of sampling from f directly, we could sample from each f_i marginally and collect samples for each f_i to make up for one sample for f. We can show that the Markov chain constructed in this way would satisfy the detailed balance equations too.

Note that we sample from the marginals in a fixed fashion, i.e. from the first component, second component, etc. Also, for each of the marginal, we will use the latest component-wise state rather than the recent full state. A modified version of this algorithm is the **random-scan Gibbs**

[1] Almost always the case for Markov chains generated by MCMC algorithms, and is easy to check.

Algorithm 16 Gibbs Sampler

```
Require: Target $n$-dimensional distribution $\pi$, starting position $x^0 = (x_1^0, x_2^0, \dots, x_n^0)$
1: $X^0 = x^0$
2: for $i = 0, 1, 2, \cdots$ do
3:     for $j = 1, 2, \dots, n$ do
4:         Draw $X_j^{i+1}$ from marginal distribution $\pi(x_j | x_1^{i+1}, x_2^{i+1}, \dots, x_{j-1}^{i+1}, x_{j+1}^{i}, \dots, x_n^{i})$
5:     end for
6: end for
```

sampler, where instead of updating the components in turn, we will pick one component at random and update that marginally. It can be shown that the random-scan Gibbs satisfies the detailed balance equations, while the standard Gibbs does not, although the standard Gibbs is still π-invariant.

The idea for **Metropolis-Within-Gibbs** is really simple. If some of the marginal distributions of the target are not easy to sample from directly, we will need additional tools to draw a sample, and we would use Metropolis-Hastings to update that component instead.

In the case where our target distribution is the posterior distribution, we have

$$\pi = \pi(\theta|x) \propto p(x|\theta)\pi(\theta)$$

where x is the data, $\pi(\theta)$ is the prior, and $p(x|\theta)$ is our likelihood. In this case, the acceptance probability of the MH algorithm becomes

$$\alpha(\theta, \theta') = \min\left\{1, \frac{p(x|\theta)\pi(\theta)q(\theta', \theta)}{p(x|\theta)\pi(\theta)q(\theta, \theta')}\right\}.$$

In order to do MH with such distributions, we would need to compute the likelihood term $p(x|\theta)$ exactly for each θ. This might not be very easy - either it is too time-consuming due to the size of the data or the computation itself lacks a computable form - and we would often call such likelihoods as **intractable likelihoods**. The first case is common in the big data setting, where there are thousands and thousands of data points, and the full likelihood is a product of all thousands of terms. It is not hard to imagine that computational time will be very large in that case. Some ways to work around the issue, including stochastic gradient MCMC (Nemeth and Fearnhead, 2021), exist in the literature, which are beyond the scope of this notes. The second case is of more relevance here, as the likelihood of a hidden Markov model is of such a complex form (especially due to the existence of a latent truth process), and will be the focus. One extremely general class of methods to deal with such cases is the **pseudo-marginal Metropolis-Hastings**, which we will describe below.

8.3.2 Pseudo-Marginal Metropolis-Hastings

Here, we will look at the pseudo-marginal Metropolis-Hasting. Let the target distribution be a posterior distribution $p(\theta|y) = p_0(\theta)p(y|\theta)/p(y)$ such that the data y is actually related to a latent variable V with distribution $p(v|\theta)$ and the relationship between V and y is captured by the conditional distribution $p(y|\theta, v)$. One example of such a structure is where V is the truth process of the HMM and y is the observations from the observation process. In this case, we can write the likelihood as

$$p(y|\theta) = \int p(y, v|\theta)dv = \int p(y|\theta, v)p(v|\theta)dv$$

which we will use again below.

The likelihood may be hard to compute, but we can estimate it using some auxiliary random variable U in the following sense:

$$p(y|\theta) \approx \hat{p}(y|\theta, u) := p(y|\theta, v)$$

where $u \sim \tilde{q}(\theta, \cdot)$ and $v \sim p(\theta, \cdot)$. The precise way of how one could obtain $\hat{p}$ is omitted here and will be discussed later. As a hint, one could recall the likelihood estimation using particle methods, as mentioned earlier in this chapter.

One key property of this estimator is that it is unbiased since we have

$$\mathbb{E}[\hat{p}(y|\theta, U)] = \mathbb{E}_V[p(y|\theta, V)] = \int p(y|\theta, v)p(\theta, v)dv = p(y|\theta)$$

as desired.

This estimator can be used consequently to estimate the posterior in a plug-in way. We have

$$p(\theta|y) = p_0(\theta)p(y|\theta)/p(y) \approx p_0(\theta)\hat{p}(y|\theta, u)/p(y)$$

and we define

$$\hat{\pi}(\theta, u) := p_0(\theta)\hat{p}(y|\theta, u) =: W p_0(\theta)p(y|\theta), \qquad W = \frac{\hat{p}(y|\theta, u)}{p(y|\theta)}.$$

One can notice that the likelihood estimation induces a multiplicative perturbation W to the true likelihood.

Using these results, we can establish the pseudo-marginal Metropolis-Hastings algorithm.

Algorithm 17 Pseudo-Marginal Metropolis-Hastings Algorithm

```
Require: Target distribution $\pi$, proposal kernel $q(\cdot, \cdot)$, auxiliary proposal kernel $\tilde{q}(\cdot, \cdot)$, starting
    position $\theta$, likelihood estimator $\hat{\pi}(\theta, u)$.
 1: Set $\theta_0 = \theta$.
 2: Draw $u_0 \sim \tilde{q}(\theta_0, \cdot)$.
 3: Compute $\hat{\pi}(\theta_0, u_0)$.
 4: for $i = 0, 1, 2, \cdots$ do
 5:     Set $\theta_{curr} = \theta_i$.
 6:     Draw $\theta_{prop} \sim q(\theta_{curr}, \cdot)$.
 7:     Draw $u_{prop} \sim \tilde{q}(\theta_{prop}, \cdot)$.
 8:     Compute $\hat{\pi}(\theta_{prop}, u_{prop})$.
 9:     Compute acceptance probability
            $\alpha(\theta_{curr}, u_{curr}; \theta_{prop}, u_{prop}) = \min\left\{1, \frac{\hat{\pi}(\theta_{prop}, u_{prop})q(\theta_{prop}, \theta_{curr})}{\hat{\pi}(\theta_{curr}, u_{curr})q(\theta_{curr}, \theta_{prop})}\right\}.$
10:     Draw $U \sim Unif[0, 1]$
11:     if $\alpha(\theta_{curr}, u_{curr}; \theta_{prop}, u_{prop}) < U$ then
12:         Accept $\theta_{prop}$ and $\theta_{i+1} = \theta_{prop}$.
13:     else
14:         Reject $\theta_{prop}$ and $\theta_{i+1} = \theta_{curr}$.
15:     end if
16: end for
```

The immediate thing we could do is to check if this algorithm is indeed targeting the right distribution. We should note that the target is

$$\tilde{\pi}(\theta, u) = \hat{\pi}(\theta, u)\tilde{q}(\theta, u) = \pi_0(\theta)\hat{p}(y|\theta, u)\tilde{q}(\theta, u)$$

which has the right marginal $\pi(\theta|y)$ since we have

$$\int \tilde{\pi}(\theta, u)du = \int \pi_0(\theta)\hat{p}(y|\theta, u)\tilde{q}(\theta, u)du = \pi_0(\theta)\int p(y|\theta, v)p(\theta, v)dv = \pi_0(\theta)p(y|\theta) \propto \pi(\theta|y).$$

Proposition 8.6. *Pseudo-marginal Metropolis-Hastings of Algorithm 17 satisfies the detailed balance equation where the joint target is $\tilde{\pi}(\theta, u)$.*

Proof. We have

$$\begin{aligned}
&\tilde{\pi}(\theta, u)q(\theta, \theta')\tilde{q}(\theta', u')\alpha(\theta, u; \theta', u') \\
&= \hat{\pi}(\theta, u)\tilde{q}(\theta, u)q(\theta, \theta')\tilde{q}(\theta', u') \min\left\{1, \frac{\hat{\pi}(\theta', u')q(\theta', \theta)}{\hat{\pi}(\theta, u)q(\theta, \theta')}\right\} \\
&= \hat{\pi}(\theta, u)\tilde{q}(\theta, u)\tilde{q}(\theta', u')q(\theta', \theta) \min\left\{\frac{q(\theta, \theta')}{q(\theta', \theta)}, \frac{\hat{\pi}(\theta', u')}{\hat{\pi}(\theta, u)}\right\} \\
&= \tilde{\pi}(\theta', u')\tilde{q}(\theta, u)q(\theta, \theta') \min\left\{1, \frac{\hat{\pi}(\theta', u')q(\theta', \theta)}{\hat{\pi}(\theta, u)q(\theta, \theta')}\right\} \\
&= \tilde{\pi}(\theta', u')q(\theta', \theta)\tilde{q}(\theta, u)\alpha(\theta', u'; \theta, u),
\end{aligned}$$

as desired to detailed balance equation. □

Previously, we have shown that the likelihood estimator can be viewed as the true likelihood with multiplicative noise. It can also be established that, using the Jensen's inequality, the acceptance rate of the pseudo-marginal is never greater than the MH with true likelihood, as established by Andrieu and Vihola (2015).

The delayed problem of doing the estimation of the likelihood will be slightly addressed here. One way of doing so is to use the approximate Bayesian computation (ABC) approach, where the likelihood estimator is closely related to a distance function. The details are omitted here and can be found in Sisson et al. (2018). Another way of doing so is to use the likelihood estimate from a particle filter algorithm when the distribution of interest is from a hidden Markov model structure. This way will be investigated in the rest of this section.

8.3.3 Particle Marginal Metropolis-Hastings

Our main goal for this chapter is to do parameter estimation for the hidden Markov model, and here we are using a Bayesian setting and the parameter estimation relies on us obtaining a good knowledge about the posterior distribution of the parameter. Fundamentally, we wish to do MCMC sampling of the posterior distribution. A straightforward implementation, although not explicitly shown here, will not work. Extra tricks and adjustments need to be taken.

One of the ways to do so is to leverage the pseudo-marginal Metropolis-Hastings framework described previously, and the unbiased likelihood estimator can be provided by a standard particle filter approximation, as outlined in the maximum likelihood approaches of parameter estimation. The full algorithm is presented below as Algorithm 18.

The correctness of this algorithm can be directly followed by the fact that pseudo-marginal Metropolis-Hastings is exact, as proved in Proposition 8.6, and the fact that the particle filter estimation of the likelihood is unbiased, as mentioned in Section 6.4.

8.3.4 Particle Gibbs

The Gibbs sampler draws samples from a multi-dimensional target distribution by drawing from each of the marginal distributions sequentially. In the case of posterior distributions with HMMs, we will be targeting the joint posterior distribution $\pi(\theta, x_{1:t}|y_{1:t})$, which can be broken down into $\pi(\theta|y_{1:t}, x_{1:t})$ and $\pi(x_{1:t}|\theta, y_{1:t})$. The first distribution is relatively easy to sample from, but the second one is not. A Metropolis-Hastings step is required to be embedded into the Gibbs framework. More importantly, we should design a Markov kernel that moves our current states of $x_{1:t}$ to a new set of states for given θ and observations $y_{1:t}$, which can be achieved by a **conditional SMC** algorithm.

The idea of a conditional SMC algorithm is very similar to that of a standard SMC algorithm. A conditional SMC is conditional on an existing trajectory, and that trajectory is always in our particles as we do the propagation and assimilation steps in the standard particle filter updates. An example of a conditional SMC algorithm is presented below as Algorithm 19 using the bootstrap

Algorithm 18 Particle Marginal Metropolis-Hastings Algorithm

```
Require: Target distribution $\pi$, proposal kernel $q(\cdot,\cdot)$, starting position $\theta$, likelihood estimator using particle filter with $\theta$ parameterisation $PF(\theta)$.
1: Set $\theta_0 = \theta$.
2: Compute $\hat{L} = PF(\theta_0)$.
3: for $i = 0, 1, 2, \cdots$ do
4:     Set $\theta_{curr} = \theta_i$.
5:     Set $\hat{L}_{curr} = \hat{L}_i$
6:     Draw $\theta_{prop} \sim q(\theta_{curr}, \cdot)$.
7:     Compute $\hat{L}_{prop} = PF(\theta_{prop})$.
8:     Compute acceptance probability
           $$\alpha(\theta_{curr}, u_{curr}; \theta_{prop}, u_{prop}) = \min\left\{1, \frac{\hat{L}_{prop} q(\theta_{prop}, \theta_{curr})}{\hat{L}_{curr} q(\theta_{curr}, \theta_{prop})}\right\}.$$
9:     Draw $U \sim Unif[0,1]$
10:    if $\alpha(\theta_{curr}, u_{curr}; \theta_{prop}, u_{prop}) < U$ then
11:        Accept $\theta_{prop}$ and $\theta_{i+1} = \theta_{prop}$, $\hat{L}_{i+1} = \hat{L}_{prop}$.
12:    else
13:        Reject $\theta_{prop}$ and $\theta_{i+1} = \theta_{curr}$, $\hat{L}_{i+1} = \hat{L}_{curr}$.
14:    end if
15: end for
```

particle filter. Note that any other types of particle filters that we described in Chapter 6 will work here, and we just need to adjust the propagation and weight computation formula.

Algorithm 19 Conditional Bootstrap Particle Filter

```
Require: Current path $X_{1:t}$ with ancestral lineage $B_{1:t}$. Conditional distribution $g(y|x)$, propagation kernel $p(x_n|x_{n-1}, \theta)$ and initial propagation kernel $p(\cdot|\theta)$. The number of particles $N$. Resampling distribution $C$.
1: Draw $X_1^{(i)} \sim p(\cdot|\theta)$ for $i = 1, 2, \ldots, N-1$.
2: Set $X_1^{(t)} = X_1$.
3: Set $W_1^{(i)} = 1/N$ for $i = 1, 2, \ldots, N$.
4: for $n = 1, 2, \cdots t$ do
5:     Draw $a_n^i \sim C(W_1)$ for $i = 1, 2, \ldots, N-1$.
6:     Draw $x_n^i \sim p(\cdot|x_{n-1}^{a_n^i}, \theta)$ for $i = 1, 2, \ldots, N-1$.
7:     Set $X_n^N = X_n$ and $a_n^N = N$.
8:     Set $w_n^i = g(y_n|x_n^i)$ for $i = 1, 2, \ldots, N$. Normalise weights $W_n^i = w_n^i / \sum_i w_n^i$.
9: end for
10: Draw $b \sim C(W_T)$ and output $X_{1:t}^b$.
```

An immediate improvement one can make is that we can introduce variability when we trace the trajectory backwards. So, for each n, we sample a_n^N with

$$\mathbb{P}(a_n^N = j) \propto w_{n-1}^j p(x_n|x_{n-1}^j, \theta).$$

Bibliography

Andrieu, C. and Vihola, M. (2015). Convergence properties of pseudo-marginal markov chain monte carlo algorithms, *Annals of Applied Probability* **25**(2): 1030–1077.

Chopin, N. and Papaspiliopoulos, O. (2020). *An introduction to sequential Monte Carlo*, Vol. 4, Springer.

Liu, J. S. (2001). *Monte Carlo strategies in scientific computing*, Vol. 10, Springer.

Nemeth, C. and Fearnhead, P. (2021). Stochastic gradient markov chain monte carlo, *Journal of the American Statistical Association* **116**(533): 433–450.

Robert, C. P. and Casella, G. (1999). *Monte Carlo statistical methods*, Vol. 2, Springer.

Sisson, S. A., Fan, Y. and Beaumont, M. (2018). *Handbook of approximate Bayesian computation*, CRC Press.

Williams, D. (1991). *Probability with martingales*, Cambridge university press.