Western Swiss Doctoral School in
Statistics and Probability

Statistical computing for systems biology

CUSO, Villars-sur-Ollon, Switzerland
September 2019

Darren Wilkinson
Newcastle University, UK
and
The Alan Turing Institute, UK

darrenjw.github.io

Lecture 3: Likelihood free inference for Markov processes and pMCMC

Alternatives to MCMC

Approximate Bayesian computation (ABC)

There is a close connection between LF-MCMC methods and those of approximate Bayesian computation (ABC). To keep things simple, consider the case of a perfectly observed system, so that there is no latent variable layer. Then there are model parameters θ described by a prior $\pi(\theta)$, and a forwards-simulation model for the data x, defined by $\pi(x|\theta)$. It is clear that a simple algorithm for simulating from the desired posterior $\pi(\theta|x)$ can be obtained as follows. First simulate from the joint distribution $\pi(\theta, x)$ by simulating $\theta^\star \sim \pi(\theta)$ and then $x^\star \sim \pi(x|\theta^\star)$. This gives a sample $(\theta^\star, x^\star)$ from the joint distribution. A simple rejection algorithm which rejects the proposed pair unless $x^\star$ matches the true data x clearly gives a sample from the required posterior distribution.

Exact rejection sampling

1. Sample $\theta^\star \sim \pi(\theta^\star)$
2. Sample $x^\star \sim \pi(x^\star|\theta^\star)$
3. If $x^\star = x$, keep $\theta^\star$ as a sample from $\pi(\theta|x)$, otherwise reject.
4. Return to step 1.

This algorithm is exact, and for discrete x will have a non-zero acceptance rate. However, in most interesting problems the rejection rate will be intolerably high. In particular, the acceptance rate will typically be zero for continuous valued x.

ABC rejection sampling

The ABC "approximation" is to accept values provided that $x^\star$ is "sufficiently close" to x. In the first instance, we can formulate this as follows.

1. Sample $\theta^\star \sim \pi(\theta^\star)$
2. Sample $x^\star \sim \pi(x^\star|\theta^\star)$
3. If $\|x^\star - x\| < \varepsilon$, keep $\theta^\star$ as a sample from $\pi(\theta|x)$, otherwise reject.
4. Return to step 1.

Euclidean distance is usually chosen as the norm, though any norm can be used. This procedure is "honest", in the sense that it produces exact realisations from

$$\theta^\star \Big| \|x^\star - x\| < \varepsilon.$$

For suitable small choice of ε, this will closely approximate the true posterior. However, smaller choices of ε will lead to higher rejection rates. This will be a particular problem in the context of high-dimensional x, where it is often unrealistic to expect a close match between all components of x and the simulated data $x^\star$, even for a good choice of $\theta^\star$. In this case, it makes more sense to look for good agreement between particular aspects of x, such as the mean, or variance, or auto-correlation, depending on the exact problem and context.

In the simplest case, this is done by forming a (vector of) summary statistic(s), $s(x^\star)$ (ideally a **sufficient statistic**), and accepting provided that $\|s(x^\star) - s(x)\| < \varepsilon$ for some suitable choice of metric and ε (Beaumont et al. 2002).

Advanced ABC approaches

It is worth noting that in certain circumstances the "tolerance", ε can be interpreted as a measurement error model (Wilkinson 2013), and for problems involving large amount of data, ABC may be applied sequentially (Sisson et al. 2007). Sequential ABC approaches have been applied to systems biology problems by Toni et al. (2009), giving a variant known as ABC-SMC. Further, it is well known that ABC approaches can be combined with MCMC to get approximate LF-MCMC schemes (Marjoram et al. 2003).

Implementations of ABC, and the sequential variant ABC-SMC, are provided in the R package `smfsb`. Do `?abcRun` and `?abcSmc` for further details.

Importance resampling for Bayesian inference

Consider again the problem of generating an (approximate) sample from the posterior $\pi(\theta|x)$. We can use importance resampling using the prior distribution $\pi(\theta)$ as an auxiliary. Explicitly, we sample $\theta_1, \theta_2, \ldots, \theta_n \sim \pi(\theta)$, then compute unnormalised weights

$$w_i = \pi(x|\theta_i), \qquad i = 1, 2, \ldots, n.$$

These weights can be normalised via

$$\tilde{w}_i = \frac{w_i}{\sum_{j=1}^{n} w_j},$$

and the normalised weights can be used to resample a new sample of size n which will then be approximately distributed according to $\pi(\theta|x)$. This procedure will not be very effective if the posterior is very different from the prior, but when used sequentially, it forms the basis of the bootstrap particle filter we will consider shortly.

It should also be noted that the average of the unnormalised weights

$$\bar{w} = \frac{1}{n} \sum_{j=1}^{n} w_j$$

is an unbiased estimator of the marginal likelihood $\pi(x)$. We therefore get estimates of marginal likelihood as a by-product of importance resampling. This too has implications that we will consider in further detail later.

Likelihood-free methods for Bayesian inference

Partially observed Markov process models

Reconsider the latent variable modelling approach from the last lecture. 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 Markov process 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 $\pi(\theta, \boldsymbol{x}|y)$. A 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 last lecture that such a proposal should be have acceptance ratio

$$A = \frac{\pi(\theta^\star)\pi(y|\boldsymbol{x}^\star, \theta^\star)f(\theta|\theta^\star)}{\pi(\theta)\pi(y|\boldsymbol{x}, \theta)f(\theta^\star|\theta)}. \tag{1}$$

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 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 (1) 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 a 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).

Note that choosing an independence proposal of the form $f(\theta^\star|\theta) = \pi(\theta^\star)$ leads to the simpler acceptance ratio

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

This “canonical” choice of proposal also lends itself to more elaborate schemes, as we will consider shortly.

This “vanilla” LF-MCMC scheme should perform reasonably well provided that y is not high-dimensional, and there is sufficient “noise” in the measurement process to make the probability of acceptance non-negligible. However, in practice y is often of sufficiently large dimension that the overall acceptance rate of the scheme is intolerably low. In this case it is natural to try and “bridge” between the prior and the posterior with a sequence of intermediate distributions. There are several ways to do this, but here it is most natural to exploit the Markovian nature of the process and consider the sequence of posterior distributions obtained as each additional time point is observed. The posterior at time t can then be computed inductively as follows.

1. Assume at time t we have a (large) sample from $\pi(\theta, x_t|y_{1:t})$ (for time 0, initialise with sample from prior)

2. Run an MCMC algorithm which constructs a proposal in two stages:

   (a) First sample $(\theta^\star, x_t^\star) \sim \pi(\theta, x_t|y_{1:t})$ by picking at random and perturbing $\theta^\star$ slightly (sampling from a kernel density estimate of the distribution)

   (b) Next sample $\mathbf{x}_{t+1}^\star$ by forward simulation from $\pi(\mathbf{x}_{t+1}^\star|\theta^\star, x_t^\star)$

   (c) Accept/reject $(\theta^\star, x_{t+1}^\star)$ with probability $\min\{1, A\}$ where

$$A = \frac{\pi(y_{t+1}|x_{t+1}^\star, \theta^\star)}{\pi(y_{t+1}|x_{t+1}, \theta)}$$

3. Output the sample from $\pi(\theta, x_{t+1}|y_{1:(t+1)})$, put $t := t + 1$, and return to step 2.

For each observation y_t, an MCMC algorithm is run which takes as input the current posterior distribution prior to observation of y_t and outputs the posterior distribution given all observations up to y_t. As y_t is typically low-dimensional, this strategy usually leads to good acceptance rates.

Note that we have suggested using an MCMC algorithm at Step 2., but this can be replaced with an importance resampling step if desired.

It is worth emphasising the generality of this algorithm — it is applicable to any Markov process discretely observed with error. It is also trivially adaptable to non-uniform observations, and to observation of multiple independent time courses (the posterior distribution from one time course can be used to form the prior distribution for the next). It is also adaptable to data from multiple (different) models which share many parameters — an important scenario in systems biology, where data is often available on multiple genetic mutants.

Unfortunately, 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.

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 the previous lecture that this can be accomplished using a proposal $f(\theta^\star|\theta)$ together with an acceptance ratio of the form

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

where $\widehat{\pi}(y|\theta)$ 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).

Bootstrap particle filter

The bootstrap particle filter is a simple sequential importance resampling (SIR) 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 now, 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 $w'^k_0 = 1/M$.

2. Suppose by induction that we are currently at time t, and that we have a weighted sample $\{x_t^k, w'^k_t | k = 1, \ldots, M\}$ 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 $\sum_{i=1}^{M} w'^i_t \delta(x - x_t^i)$ (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 $w^k_{t+1} = \pi(y_{t+1}|x^k_{t+1})$, and then a normalised weight $w'^k_{t+1} = w^k_{t+1} / \sum_i w^i_{t+1}$.

   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

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

and

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

Again, from our understanding of importance resampling, it should be reasonably clear that $\widehat{\pi}(y_{1:T})$ 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 1. An example of use is given in Figure 2 — 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.

The algorithm will target $\pi(\theta|y)$ exactly irrespective of the number of particles, M, used in the particle filter, but the mixing of the chain will improve as M increases.

```
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], ...))
            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 1: 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 2.

```
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 2: An R session showing how to use the function `pfMLLik` from Figure 1. 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.

Likelihood free SMC–within–MCMC algorithm

1. At iteration i the current state of the MCMC chain is $(\theta, \widehat{\pi}(y|\theta))$

2. Propose $\theta^\star \sim f(\theta^\star|\theta)$

3. Evaluate $\widehat{\pi}(y|\theta^\star)$ by running a bootstrap particle filter with M particles

4. Evaluate the acceptance ratio

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

5. Accept $(\theta^\star, \widehat{\pi}(y|\theta^\star))$ as the new state with probability $\min\{1, A\}$, otherwise retain the current state

6. Increment i and return to step 1.

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

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

where $\widehat{\pi}_{\theta^\star}(y_{1:T})$ 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

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

and

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

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.

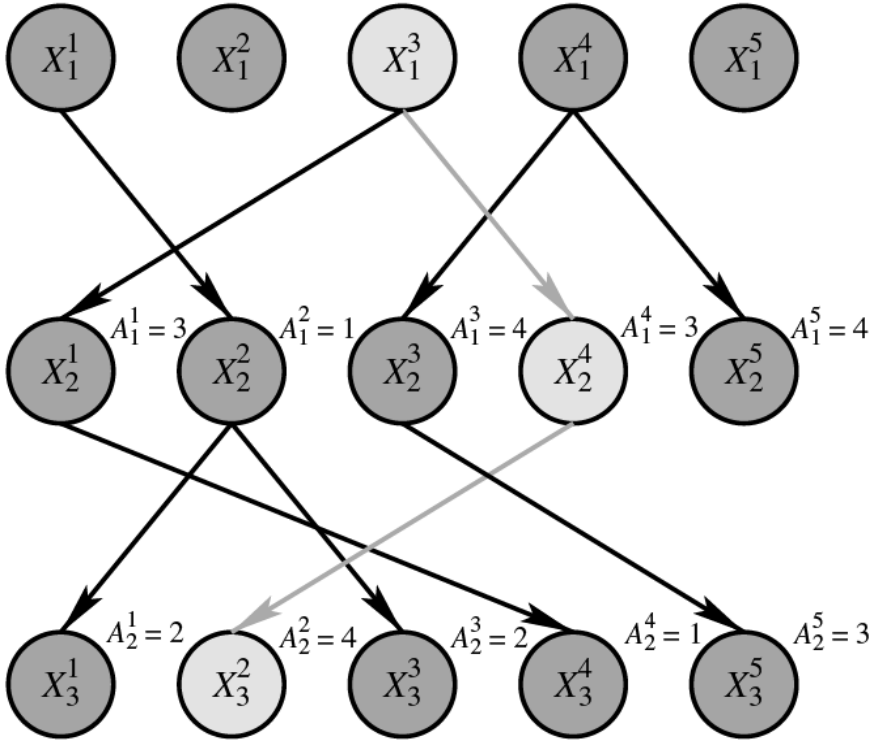

Fig. 1. Example of ancestral lineages generated by an SMC algorithm for $N = 5$ and $T = 3$: the lighter path is $X_{1:3}^2 = (X_1^3, X_2^4, X_3^2)$ and its ancestral lineage is $B_{1:3}^2 = (3,4,2)$

Figure 3: Particles and genealogies, taken from Andrieu et al. (2010).

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

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

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

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

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 $\widehat{\pi}_\theta(y_{1:T})$ 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

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

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 pseudo-marginal MCMC algorithm we have already considered is a special case of the PMMH algorithm, where we do not bother 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.

Case study: inference for the Lotka–Volterra model

We will next examine the problem of inferring Markov process 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 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 4. R code implementing an MCMC algorithm to infer the three model parameters using this data is given in Figure 5. Similar code is included in the `smfsb` package demo, `"PMCMC"`, which can easily be customised for different observation scenarios. Note that the acceptance ratio used in the MCMC sampler is just

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

and that the proposal distribution is symmetric on $\log(\theta)$. Since there is no prior ratio included in the acceptance ratio, this corresponds to a flat (improper) uniform prior on $\log(\theta)$. Different priors may be accommodated by modifying the acceptance ratio accordingly. 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.

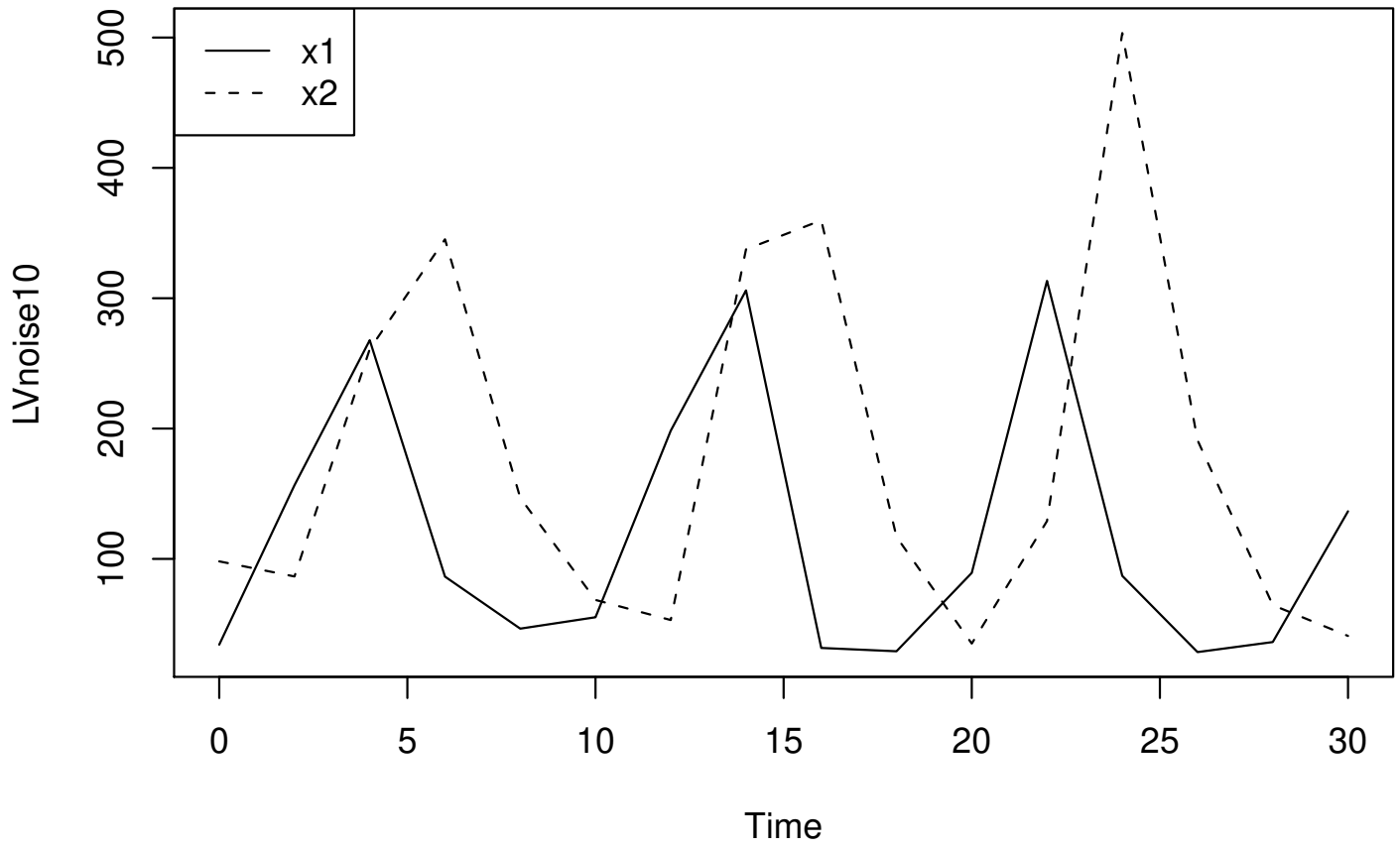

Figure 4: Simulated time series data set, `LVnoise10`, consisting of 16 equally spaced observations of a realisation of a stochastic kinetic Lotka–Volterra model subject to Gaussian measurement error with a standard deviation of 10.

```
# make sure the package is loaded
require(smfsb)
# 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)
                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
}
LVdata=as.timedData(LVnoise10)
# create marginal log-likelihood function, based on a particle
  filter
mLLik=pfMLLik(100,simx0,0,stepLVc,dataLik,LVdata)
# Now create an MCMC algorithm...
```

```
iters=1000
tune=0.01
thin=10
th=c(th1 = 1, th2 = 0.005, th3 = 0.6)
p=length(th)
ll=-1e99
thmat=matrix(0,nrow=iters,ncol=p)
colnames(thmat)=names(th)
# main MCMC loop
for (i in 1:iters) {
        message(paste(i,""),appendLF=FALSE)
        for (j in 1:thin) {
                thprop=th*exp(rnorm(p,0,tune))
                llprop=mLLik(thprop)
                if (log(runif(1)) < llprop - ll) {
                        th=thprop
                        ll=llprop
                        }
                }
        thmat[i,]=th
        }
message("Done!")
# plot the results
mcmcSummary(thmat)
```

Figure 5: R code implementing an MCMC sampler for fully Bayesian inference for the stochastic Lotka–Volterra model using time course data.

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 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
```

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.

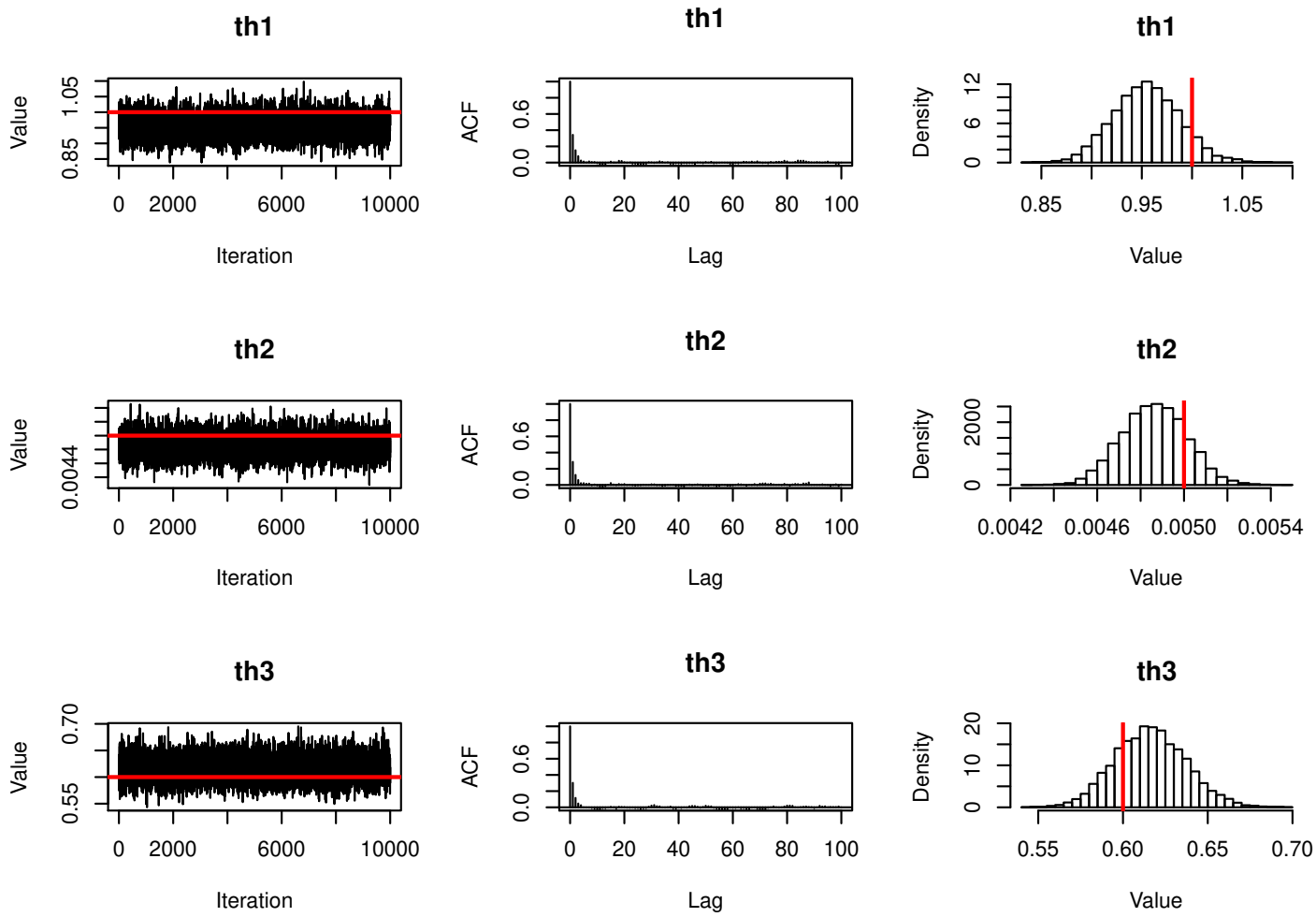

Figure 6: Marginal posterior distributions for the parameters of the Lotka–Volterra model, based on the data given in Figure 4. Note that the mixing of the MCMC sampler is good, and that the true parameters, $\theta = (1, 0.005, 0.6)$ are well identified by the data.

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

```
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 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
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.

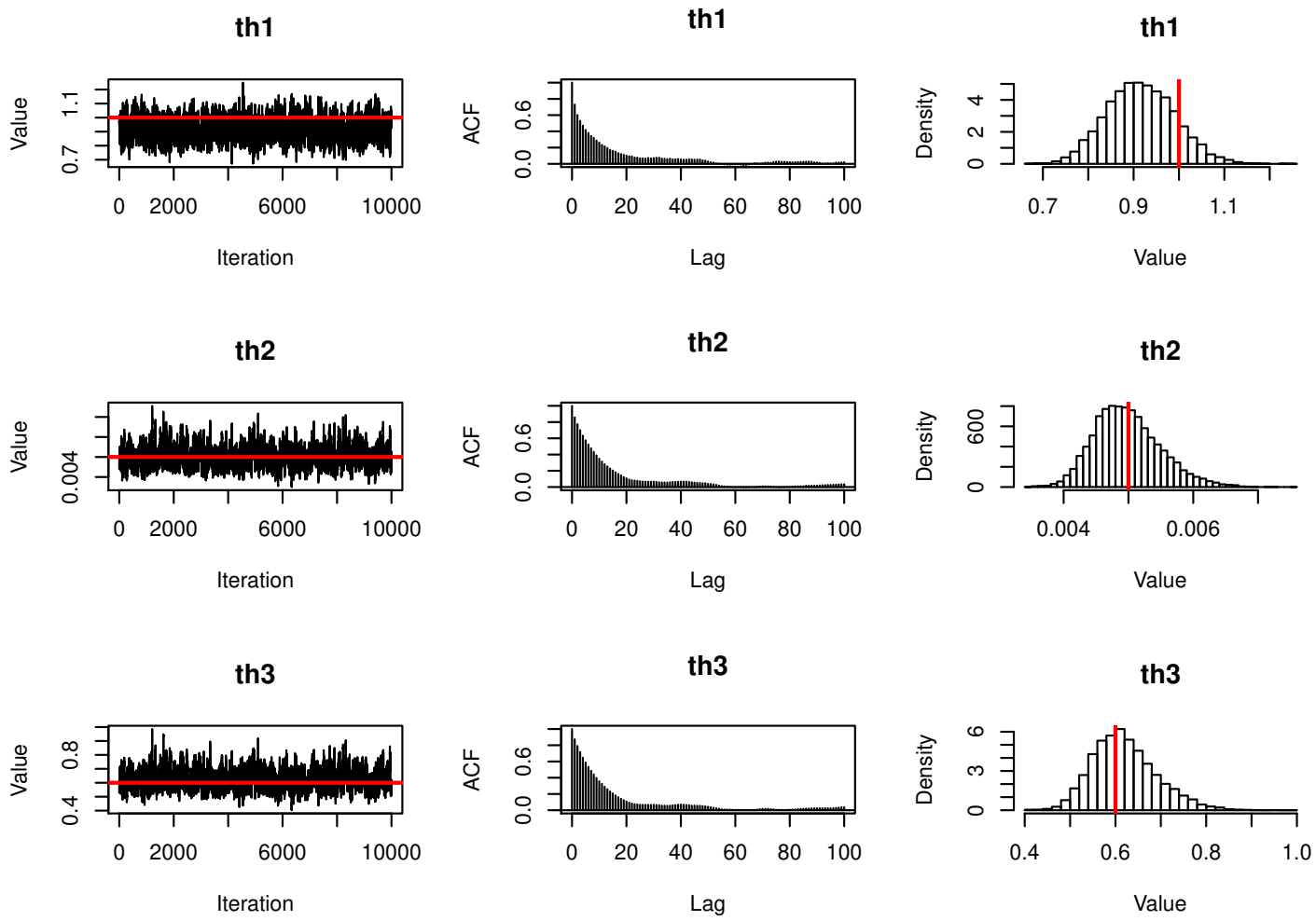

Figure 7: Marginal posterior distributions for the parameters of the Lotka–Volterra model, based only on observations of prey species levels. Note that the mixing of the MCMC sampler is reasonable, and that the true parameters, $\theta = (1, 0.005, 0.6)$ are quite well identified by the data.

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 issues 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)
                return(ll)
        else
                return(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.

Simply modifying the acceptance step of the algorithm to reject values of the standard deviation outside the range $[5, 50]$ is equivalent to assuming the prior

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

and identifying parameter bounds in this way 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 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, looking at the sampled MCMC values on the log scale can sometimes show 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 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 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 artifact 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.

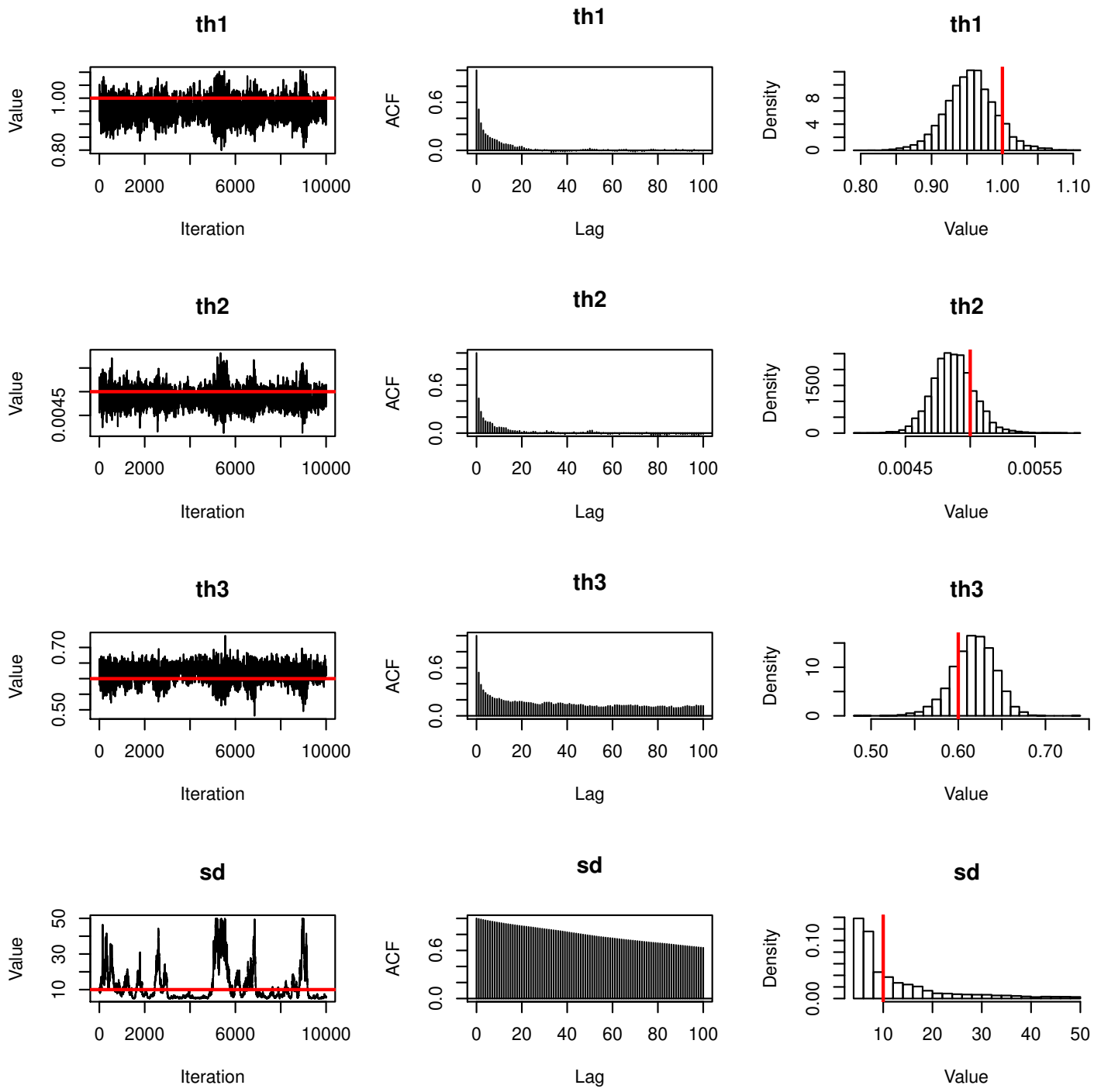
th1
Value
0.80
1.00
0
2000
6000
10000
Iteration
th1
ACF
0.0
0.6
0
20
40
60
80
100
Lag
th1
Density
0
4
8
0.80
0.90
1.00
1.10
Value
th2
Value
0.0045
0
2000
6000
10000
Iteration
th2
ACF
0.0
0.6
0
20
40
60
80
100
Lag
th2
Density
0
1500
0.0045
0.0055
Value
th3
Value
0.50
0.70
0
2000
6000
10000
Iteration
th3
ACF
0.0
0.6
0
20
40
60
80
100
Lag
th3
Density
0
10
0.50
0.60
0.70
Value
sd
Value
10
30
50
0
2000
6000
10000
Iteration
sd
ACF
0.0
0.6
0
20
40
60
80
100
Lag
sd
Density
0.00
0.10
10
20
30
40
50
Value

Figure 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.

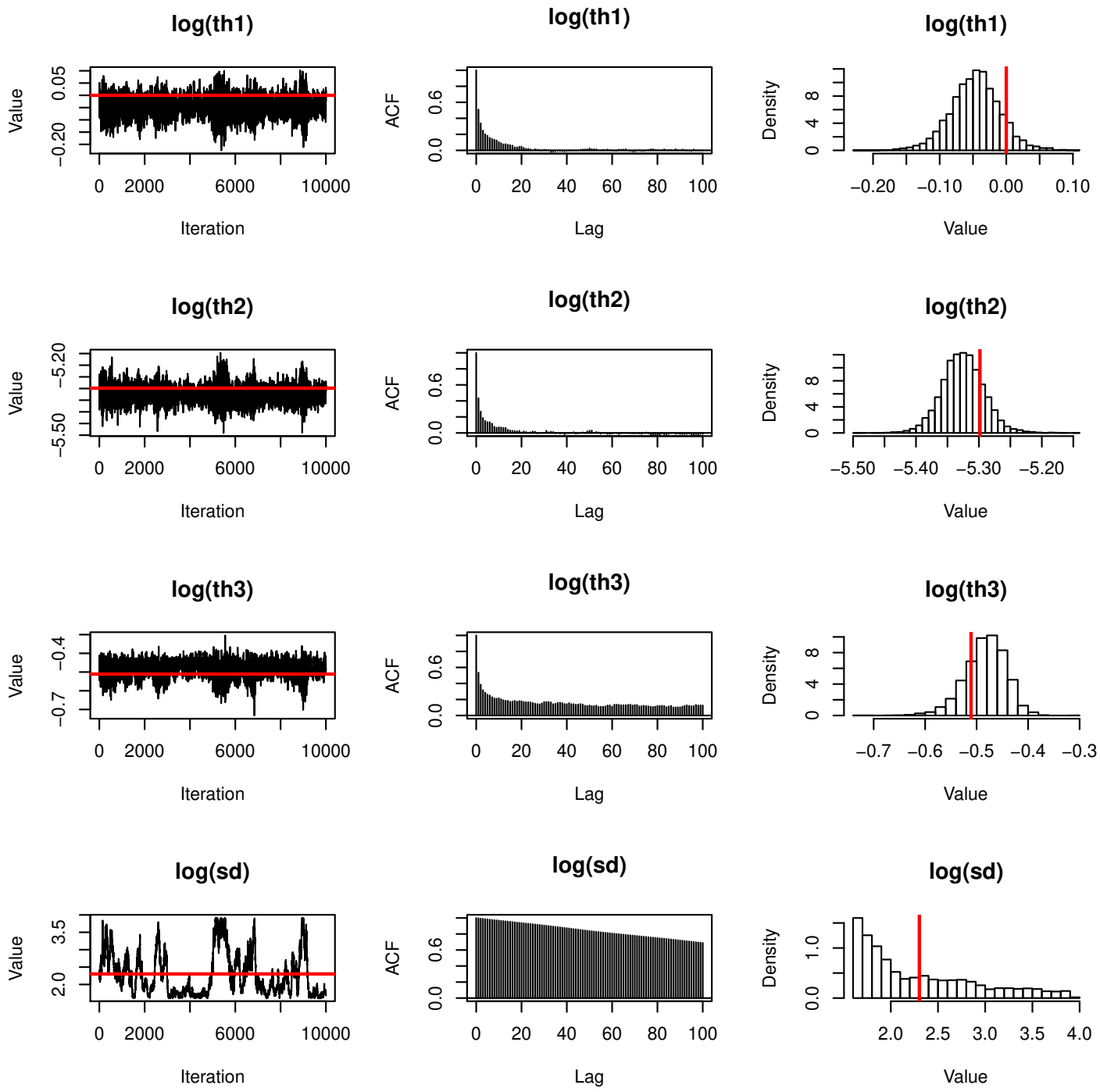
log(th1)
Value
Iteration
ACF
Lag
Density
log(th2)
log(th3)
log(sd)

Figure 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 $\log(\theta) = (0, -5.23, -0.51, 2.30)$.

Whenever there appear to be issues with poorly identified parameters and/or a poorly mixing sampler, it can be very useful to look at the posterior correlation structure for signs of parameter confounding. The simplest way to do this is to produce a pairwise scatterplot matrix. Such a plot is shown in Figure 10, which was produced with the command `pairs(thmat)`. Here it can be seen that there doesn't appear to be any very strong pairwise correlation between any of the variables. There is a weak correlation between variables `th1` and `th2`, and other pairs of variables are approximately uncorrelated. Note that such a plot may not be effective for highlighting confounding between sets of three or more variables. For this, multivariate data analysis techniques may be required.

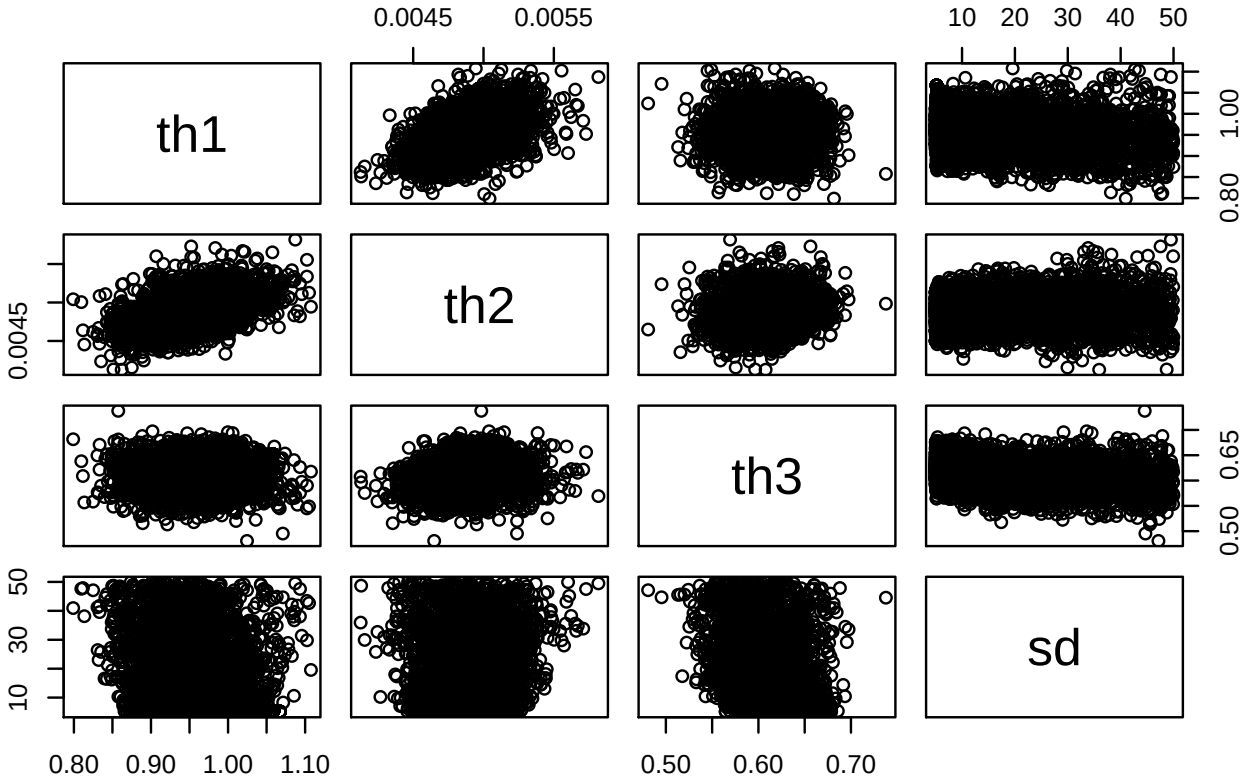

Figure 10: Pairwise scatterplot matrix for the MCMC sample from the posterior distribution.

Here we have just begun to explore the issues of parameter inference for Markov process 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 model (in the presence of measurement error), and that the techniques can be readily applied to a range of data-poor partial 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 the scale of the model output. Again, this can be handled with the introduction of a scaling parameter, though there can often be identifiability issues associated with this — see the exercises.

We have not considered computation time and the computational expense of SMC-within-MCMC algorithms. Some of the MCMC runs above 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), and I have done this as a free library, `scala-smfsb`. However, it will also port easily to any reasonable object-oriented language, such as Java (Flanagan 2005) or C++ (Satir & 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 diffusion approximation to the true model. Such strategies will be necessary for large and complex models.

My web site: https://darrenjw.github.io/

SMfSB 3e main web site: https://github.com/darrenjw/smfsb

Example Scala library: https://github.com/darrenjw/scala-smfsb

Exercises

1. Consider the Lotka–Volterra model considered in this lecture, and note that some of these exercises will require a lot of CPU time (potentially days) to complete.

   (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 auto-correlations? 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?

2. Implement a simple ABC scheme in R for parameter inference for the Lotka-Volterra model. Start with the partially observed case (`LVpreyNoise10`) and just use plain Euclidean distance between the data and the model output in the thresholding criteria.

Start with a large threshold ε and gradually decrease it to zero, keeping track of the acceptance rate. How small does ε need to be to accurately reproduce the posterior distribution as computed using pMCMC? What is the acceptance rate in this case? How does the computation time compare to pMCMC?

3. Implement a sequential LF-MCMC scheme for the same problem as above. How many MCMC samples are required at each time point in order to accurately reproduce the posterior distribution for the model parameters? Note that thinning of the MCMC output is likely to be required at each time point, so that a thinned sample can be passed forward to the next time. How does the overall computation time compare to pMCMC and ABC?

*

References

Andrieu, C., Doucet, A. & Holenstein, R. (2010), 'Particle Markov chain Monte Carlo methods (with discussion)', *Journal of the Royal Statistical Society, Series B* **72**(3), 269–342.

Beaumont, M. A., Zhang, W. & Balding, D. J. (2002), 'Approximate Bayesian computation in population genetics', *Genetics* **162**(4), 2025–2035.

Del Moral, P. (2004), *Feynman-Kac formulae: genealogical and interacting particle systems with applications*, Springer series in statistics: Probability and its applications, Springer, New York.

Doucet, A., de Freitas, N. & Gordon, N., eds (2001), *Sequential Monte Carlo Methods in Practice*, Springer, New York.

Flanagan, D. (2005), *Java in a Nutshell*, A Nutshell Handbook, O'Reilly, Sebastopol, CA.

Marjoram, P., Molitor, J., Plagnol, V. & Tavare, S. (2003), 'Markov chain Monte Carlo without likelihoods', *Proc. Natl. Acad. Sci. U.S.A.* **100**(26), 15324–15328.

Odersky, M., Spoon, L. & Venners, B. (2008), *Programming in Scala*, Artima.

Pitt, M. K., dos Santos Silva, R., Giordani, P. & Kohn, R. (2012), 'On some properties of Markov chain Monte Carlo simulation methods based on the particle filter', *Journal of Econometrics* **171**(2), 134 – 151.

Satir, G. & Brown, D. (1995), *C++: The Core Language*, O'Reilly Series, O'Reilly & Assoc., Inc., Sebastopol, CA.

Sisson, S. A., Fan, Y. & Tanaka, M. M. (2007), 'Sequential Monte Carlo without likelihoods', *Proc. Natl. Acad. Sci. U.S.A.* **104**(6), 1760–1765.

Toni, T., Welch, D., Strelkowa, N., Ipsen, A. & Stumpf, M. P. H. (2009), 'Approximate Bayesian computation scheme for parameter inference and model selection in dynamical systems', *J. R. Soc. Interface* **6**(31), 187–202.

Wilkinson, D. J. (2011), Parameter inference for stochastic kinetic models of bacterial gene regulation: a Bayesian approach to systems biology (with discussion), *in* J. M. Bernardo, ed., 'Bayesian Statistics 9', Oxford Science Publications, pp. 679–706.

Wilkinson, R. (2013), 'Approximate Bayesian computation (ABC) gives exact results under the assumption of model error', *Statistical Applications in Genetics and Molecular Biology* **12**(2), 129–141.