Journal of Statistical Software

October 2015, Volume 67, Issue 10. doi: 10.18637/jss.v067.i10

Bayesian State-Space Modelling on High-Performance Hardware Using LibBi

Lawrence M. Murray
University of Oxford

Abstract

LibBi is a software package for state space modelling and Bayesian inference on modern computer hardware, including multi-core central processing units, many-core graphics processing units, and distributed-memory clusters of such devices. The software parses a domain-specific language for model specification, then optimizes, generates, compiles and runs code for the given model, inference method and hardware platform. In presenting the software, this work serves as an introduction to state space models and the specialized methods developed for Bayesian inference with them. The focus is on sequential Monte Carlo (SMC) methods such as the particle filter for state estimation, and the particle Markov chain Monte Carlo and SMC^2 methods for parameter estimation. All are well-suited to current computer hardware. Two examples are given and developed throughout, one a linear three-element windkessel model of the human arterial system, the other a nonlinear Lorenz '96 model. These are specified in the prescribed modelling language, and **LibBi** demonstrated by performing inference with them. Empirical results are presented, including a performance comparison of the software with different hardware configurations.

Keywords: Bayesian hierarchical modelling, state space modelling, sequential Monte Carlo, particle Markov chain Monte Carlo, **LibBi**.

1. Introduction

State space models (SSMs) have important applications in the study of physical, chemical and biological processes. Examples are numerous, but include marine biogeochemistry (Dowd 2006; Jones, Parslow, and Murray 2010; Dowd 2011; Parslow, Cressie, Campbell, Jones, and Murray 2013), ecological population dynamics (Wikle 2003; Newman, Fernández, Thomas, and Buckland 2009; Peters, Hosack, and Hayes 2011; Hosack, Peters, Hayes, and R. 2012), Functional Magnetic Resonance Imaging (Riera *et al.* 2004; Murray and Storkey 2008, 2011),

biochemistry (Golightly and Wilkinson 2008, 2011) and object tracking (Vo and Ma 2006). They are particularly useful for modelling uncertainties in the parameters, states and observations of such processes, and particularly successful where a rigorous probabilistic treatment of these uncertainties leads to improved scientific insight or risk-informed decision making.

SSMs can be interpreted as a special class within the more general class of Bayesian hierarchical models (BHMs). For a given data set, existing methods for inference with BHMs, such as Gibbs sampling (Geman and Geman 1984) or Hamiltonian Monte Carlo (HMC, Neal 2011), might be applied. However, specialist machinery for SSMs has been developed to better deal with peculiarities typical of models in the class: nonlinearity, multiple modality, prior densities without closed form that cannot be evaluated pointwise, and significant correlations between state variables and parameters. Variants of sequential Monte Carlo (SMC, Doucet, Freitas, and Gordon 2001) are particularly attractive, including particle Markov chain Monte Carlo (PMCMC, Andrieu, Doucet, and Holenstein 2010) and SMC^2 (Chopin, Jacob, and Papaspiliopoulos 2013). These methods have two particularly pragmatic qualities: they admit a wide range of SSMs, including nonlinear and non-Gaussian models, without approximation bias, and they are well-suited to recent, highly parallel computer architectures (Lee, Yau, Giles, Doucet, and Holmes 2010).

Commodity computing hardware has evolved from the monoculture of x86 CPUs in the 1990s to the ecosystem of diverse desktop central processing units (CPUs), mobile processors and specialist graphics processing units (GPUs) of today. Adding to the challenge since 2004 is that Moore's Law, as applied to computing performance, has been upheld not by increasing clock speed, but by broadening parallelism (Sutter 2005). This should not be understood as a passing fad, nor a deliberate design choice for modern applications, but as a necessity enforced by physical limits, the most critical of which is energy consumption (Ross 2008). Thus, while architectures continue to change, their reliance on parallelism is unlikely to, at least in the foreseeable future. The implication for statistical computing is clear: in order to make best use of current and future architectures, algorithms must be parallelized. On this criterion, SMC methods are a good fit.

Given these specialized methods for SSMs, it is appropriate that specialized software be available also. **LibBi** (Murray 2013) is such a package. Nominally, the name is a contraction of "Library for Bayesian inference", and pronounced "Libby". Its design goals are accessibility and speed. It accepts SSMs specified in its own domain-specific modelling language, which is parsed and optimized to generate and compile C++ code for the execution of inference methods. The code exploits technologies for SIMD (Single Instruction Multiple Data) vector operations including SSE (Streaming SIMD Extensions) and AVX (Advanced Vector Extensions), multithreading with OpenMP for shared-memory parallelism, MPI for distributed-memory parallelism, and CUDA for GPU parallelism. The user interacts with the package via a command-line interface, with input and output files handled in the standard NetCDF format, based on the high-performance HDF5.

This work serves as a brief introduction to SSMs, appropriate Bayesian inference methods for them, the modern computing context, and the consideration of all three in **LibBi**. The material is presented in three parts: Section 2 introduces SSMs as a special class of BHMs; Section 3 provides specialist methods for inference with the class; Section 4 provides technical information on **LibBi** itself. Two examples are developed throughout. At the end of Section 2, these examples are developed as SSMs and specified in the **LibBi** modelling language. At the end of Section 3, posterior distributions are obtained for each model conditioned on simulated

data sets, using the methods presented. In Section 4, the performance of **LibBi** on various hardware platforms is compared. Section 5 summarizes results. Section 6 provides information on available supplementary materials to reproduce the example results throughout this work.

2. State space models

State space models (SSMs) are suitable for modelling dynamical systems that have been observed at one or more instances in time. They consist of parameters $\boldsymbol{\Theta} \in \mathbb{R}^{N_\theta}$, a latent continuous- or discrete-time state process $\mathbf{X}(t) \in \mathbb{R}^{N_x}$, and an observed continuous- or discrete-time process $\mathbf{Y}(t) \in \mathbb{R}^{N_y}$. A starting time is given by t_0, and an ordered sequence of observation times by $t_1, \ldots, t_T$.

The most general BHM over these random variables admits a joint density of the form:

$$\underbrace{p\left(\mathbf{Y}(t_{1:T}), \mathbf{X}(t_{0:T}), \boldsymbol{\Theta}\right)}_{\text{joint}} = \underbrace{p\left(\mathbf{X}(t_{0:T}), \boldsymbol{\Theta}\right)}_{\text{prior}} \underbrace{p\left(\mathbf{Y}(t_{1:T}) | \mathbf{X}(t_{0:T}), \boldsymbol{\Theta}\right)}_{\text{likelihood}}. \tag{1}$$

The SSM class can be considered a specialization of the general BHM class. It imposes the Markov property on the state process $\mathbf{X}(t)$, and at each time t_i permits the observation $\mathbf{Y}(t_i)$ to depend only on parameters $\boldsymbol{\Theta}$ and the state $\mathbf{X}(t_i)$. Formally, the joint density takes the form:

$$\begin{aligned} &\underbrace{p\left(\mathbf{Y}(t_{1:T}), \mathbf{X}(t_{0:T}), \boldsymbol{\Theta}\right)}_{\text{joint}} \\ &= \underbrace{\underbrace{p\left(\boldsymbol{\Theta}\right)}_{\text{parameter}} \underbrace{p\left(\mathbf{X}(t_0) | \boldsymbol{\Theta}\right)}_{\text{initial}} \left(\prod_{i=1}^{T} \underbrace{p\left(\mathbf{X}(t_i) | \mathbf{X}(t_{i-1}), \boldsymbol{\Theta}\right)}_{\text{transition}} \right)}_{\text{prior}} \underbrace{\left(\prod_{i=1}^{T} \underbrace{p\left(\mathbf{Y}(t_i) | \mathbf{X}(t_i), \boldsymbol{\Theta}\right)}_{\text{observation}} \right)}_{\text{likelihood}}, \end{aligned} \tag{2}$$

also depicted as a graphical model in Figure 1. The prior density is factored into a parameter density, an initial state density, and a product of one or more transition densities. The likelihood function is given as a product of observation densities. Specific models in the class may, of course, exploit further conditional independencies and so introduce additional hierarchical structure. This is demonstrated in the examples that follow.

2.1. Examples

Two example SSMs are introduced here. The first is a linear three-element windkessel model of arterial blood pressure (Westerhof, Lankhaar, and Westerhof 2009), the second a nonlinear eight-dimensional Lorenz '96 model of chaotic atmospheric processes (Lorenz 2006). In each case the original deterministic model is introduced, then stochasticity added, and a prior distribution over parameters specified, to massage it into the SSM framework.

Windkessel model

A windkessel model can be used to relate blood pressure and blood flow in the human arterial system (Westerhof *et al.* 2009). The simplest two-element windkessel (Frank 1899) is physically inspired: it couples a pump (the heart) and a chamber (the arterial system), with

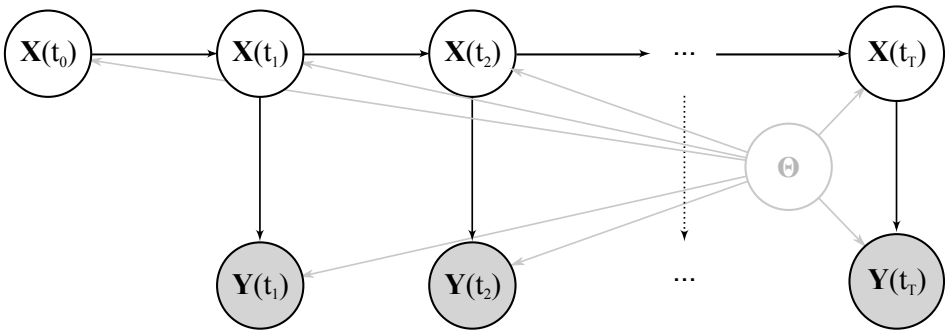

Figure 1: The state space model, with parameters $\boldsymbol{\Theta}$, latent Markov state process $\mathbf{X}(t_{0:T})$ and observation process $\mathbf{Y}(t_{1:T})$.

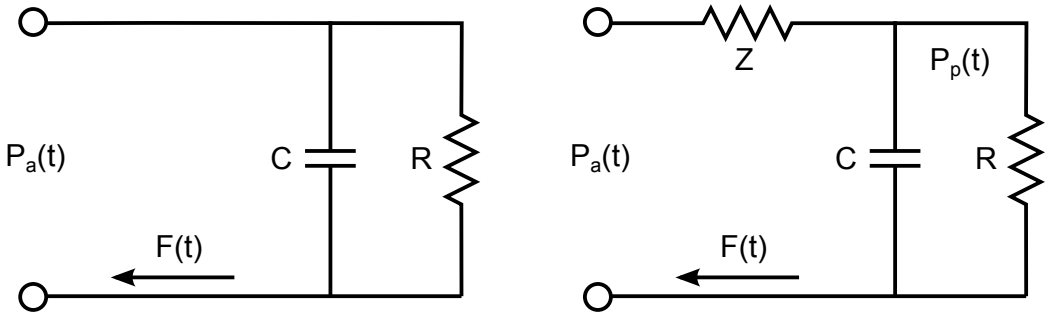

Figure 2: Windkessel models represented as circuit diagrams, *(left)* two-element, and *(right)* three-element. $P_a(t)$ gives aortal pressure and $P_p(t)$ peripheral pressure. $F(t)$ gives blood flow.

fluid (blood) flowing from the pump to the chamber, and returning via a closed loop. Air in the chamber is compressed according to the volume of fluid (analogous to the compliance of arteries).

The two-element windkessel model is commonly represented as an RC circuit as in Figure 2 (left). Voltage models blood pressure and current blood flow. A resistor models narrowing vessel width in the periphery of the arterial system, and a capacitor the compliance of arteries.

The equations of the two-element windkessel are readily derived from circuit theory applied to Figure 2 (left), see Kind, Faes, Lankhaar, Vonk-Noordegraaf, and Verhaegen (2010):

$$\frac{1}{R}P_p(t) + C\frac{dP_p(t)}{dt} = F(t).$$

This is a linear differential equation. Assuming a discrete time step Δt and constant flow $F(\mathrm{t})$ over the interval $[t, t + \Delta t)$, it may be solved analytically to give:

$$P_p(t + \Delta t) = \exp\left(-\frac{\Delta t}{RC}\right) P_p(t) + R\left(1 - \exp\left(-\frac{\Delta t}{RC}\right)\right) F(t). \tag{3}$$

The two-element windkessel captures blood pressure changes during diastole (heart dilatation) well, but not during systole (heart contraction), see Westerhof *et al.* (2009). An improvement

is the three-element windkessel in Figure 2 (right), which introduces additional impedence from the aortic valve. Again using circuit theory, aortal pressure, P_a, may be related to peripheral pressure, P_p, by (Kind *et al.* 2010):

$$P_a(t) = P_p(t) + F(t)Z. \tag{4}$$

The three-element windkessel is adopted henceforth. Blood flow, $F(t)$, would usually be measured, but for demonstrative purposes it is simply prescribed the following functional form:

$$F(t) = \begin{cases} F_{\max} \sin^2\left(\frac{\pi t}{T_s+T_d}\right) & \text{if } \operatorname{mod}(t, T_s+T_d) \in [0, T_s] \\ 0 & \text{if } \operatorname{mod}(t, T_s+T_d) \in (T_s, T_s+T_d), \end{cases} \tag{5}$$

where $F_{\max} = 500\,\text{ml s}^{-1}$ gives maximum flow, $T_s = 0.3\,\text{s}$ is time spent in systole, $T_d = 0.5\,\text{s}$ is time spent in diastole, with $\operatorname{mod}(x, y)$ giving the non-integer part of x/y. This models quickly increasing then decreasing blood flow during systole, and no flow during diastole. The function is discretized to time steps of Δt and held constant in between.

An SSM can be constructed around the above equations. Input $F(t)$ is prescribed as above, and a Gaussian noise term $\xi(t)$ of zero mean and variance σ^2 is introduced to extend the state process from deterministic to stochastic behavior (details below). The SSM has parameters $\boldsymbol{\Theta} \equiv (R, C, Z, \sigma^2)^T$, state $\mathbf{X}(t) \equiv P_p(t)$ and observation $\mathbf{Y}(t) \equiv P_a(t)$. The time step is fixed to $\Delta t = 10^{-2}$s.

The complete model is specified in the **LibBi** modelling language in Figure 3. The specification begins with a `model` statement to name the model. It proceeds with the time step size declared on line 5 as a constant value. Following this, the four parameters of the model, `R` (R), `C` (C), `Z` (Z) and `sigma2` (σ^2) are declared on lines 7–10, the input `F` (F(t)) on line 11, the noise term `xi` ($\xi(t)$) on line 12, the state variable `Pp` ($P_p(t)$) on line 13, and the observation `Pa` ($P_a(t)$) on line 14.

Recall (2), which gives the general joint distribution of an SSM. The specific joint distribution for this SSM is:

$$\begin{aligned} \underbrace{p(P_a(t_{1:T}), P_p(t_{0:T}), R, C, Z, \sigma^2)}_{\text{joint}} &= \underbrace{p(R)p(C)p(Z)p(\sigma^2)}_{\text{parameter}} \underbrace{p(P_p(t_0))}_{\text{initial}} \\ &\times \left(\prod_{i=1}^{T} \underbrace{p(P_p(t_i)|P_p(t_{i-1}), R, C, \sigma^2, F(t_{i-1}))}_{\text{transition}}\right) \left(\prod_{i=1}^{T} \underbrace{p(P_a(t_i)|P_p(t_i), Z, F(t_{i-1}))}_{\text{observation}}\right). \end{aligned} \tag{6}$$

Following the variable declarations in Figure 3 are four *blocks*, declared using the `sub` keyword, each with the same name as, and describing the form of, one of the factors in (6). The prior distribution over parameters is given by:

$$\begin{aligned} R &\sim \text{Gamma}(2, 0.9) \\ C &\sim \text{Gamma}(2, 1.5) \\ Z &\sim \text{Gamma}(2, 0.03) \\ \sigma^2 &\sim \text{Inv-Gamma}(2, 500000), \end{aligned}$$

described in the `parameter` block on lines 16–21 of Figure 3. The hyperparameters of these distributions are guided by Kind *et al.* (2010). The prior distribution over the initial state is

Windkessel.bi

```
/**
 * Three-element Windkessel model.
 */
model Windkessel {
  const h = 0.01  // time step (s)

  param R          // peripheral resistance, mm Hg (ml s**-1)**-1
  param C          // arterial compliance, ml (mm Hg)**-1
  param Z          // characteristic impedence, mm Hg s ml**-1
  param sigma2     // process noise variance, (mm Hg)**2
  input F          // aortic flow, ml s**-1
  noise xi         // noise, ml s**-1
  state Pp         // peripheral pressure, mm Hg
  obs Pa           // observed aortic pressure, mm Hg

  sub parameter {
    R ~ gamma(2.0, 0.9)
    C ~ gamma(2.0, 1.5)
    Z ~ gamma(2.0, 0.03)
    sigma2 ~ inverse_gamma(2.0, 500000.0)
  }

  sub initial {
    Pp ~ gaussian(90.0, 15.0)
  }

  sub transition(delta = h) {
    xi ~ gaussian(0.0, sqrt(h*sigma2))
    Pp <- exp(-h/(R*C))*Pp + R*(1.0 - exp(-h/(R*C)))*(F + xi)
  }

  sub observation {
    Pa ~ gaussian(Pp + Z*F, 2.0)
  }

  sub proposal_parameter {
    R ~ truncated_gaussian(R, 0.03, lower = 0.0)
    C ~ truncated_gaussian(C, 0.1, lower = 0.0)
    Z ~ truncated_gaussian(Z, 0.002, lower = 0.0)
    sigma2 ~ inverse_gamma(2.0, 3.0*sigma2)
  }
}
```

Figure 3: The windkessel SSM specified in the **LibBi** modelling language.

given by:

$$P_p(t_0) \quad \sim \quad \mathrm{N}(90, 15),$$

where the second argument is the standard deviation. This is described in the `initial` block on lines 23–25 of Figure 3. The transition model is a stochastic extension of (3). This use of stochasticity might be interpreted as representing some uncertainty in the formulation of the model given the biological phenomena it is meant to represent. The noise term $\xi(t)$ is introduced additively to blood flow, $F(t)$:

$$P_p(t + \Delta t) = \exp\left(-\frac{\Delta t}{RC}\right) P_p(t) + R\left(1 - \exp\left(-\frac{\Delta t}{RC}\right)\right)(F(t) + \xi(t)).$$

This form is described in the `transition` block on lines 27–30 of Figure 3. The observation model is based on (4), with additional noise:

$$P_a(t) \sim \mathrm{N}(P_p(t) + ZF(t), 2).$$

This is described in the `observation` block on lines 32–34 of Figure 3.

One additional block is specified for the windkessel model. This is the `proposal_parameter` block on lines 36–41. It gives the proposal distribution over parameters that is used during marginal Metropolis-Hastings sampling, described in Section 3.2.

Note that the input `F` does not appear on the left side in any block. Instead, as an `input` variable, its value at each time is drawn from an input file. This is prepared in advance from (5) as a NetCDF file.

Lorenz '96 model

Lorenz '96 models are useful for the simulation of some important atmospheric processes (Lorenz 2006). They can be challenging to handle, owing to chaotic behavior in most regimes. An N-dimensional Lorenz '96 model over the vector $\mathbf{x} \in \mathbb{R}^N$ is given by the ordinary differential equations (ODEs):

$$\frac{dx_n}{dt} = x_{n-1}(x_{n+1} - x_{n-2}) - x_n + F, \tag{7}$$

where indices into $\mathbf{x}$ are taken to be cyclic, so that $x_{n-N} \equiv x_n \equiv x_{n+N}$. F acts as a constant forcing term that induces various behaviors ranging from decay, to periodicity, to chaos. The bifurcation diagram of this deterministic system is given in Figure 4 (left). A deterministic model such as this is inappropriate for the SSM framework. To derive a suitable stochastic model, the ODEs of (7) can be converted to stochastic differential equations (SDEs):

$$dX_n = (X_{n-1}(X_{n+1} - X_{n-2}) - X_n + F)\, dt + \sigma\, dW_n. \tag{8}$$

Each W_n represents a standard Wiener process (Gardiner 2004, §3.8.1), and σ is a parameter used to scale these. As the diffusion term is additive the SDEs may be interpreted equivalently (Kloeden and Platen 1992, p157) in either the Itô (Gardiner 2004, §4.2.1) or Stratonovich (Gardiner 2004, §4.2.3) sense. The bifurcation diagram of this stochastic system is given in Figure 4 (right).

There is no analytical solution to integrate either the ODEs or SDEs to obtain a form for X_n; they must be integrated numerically. For the deterministic model, Lorenz (2006) uses a

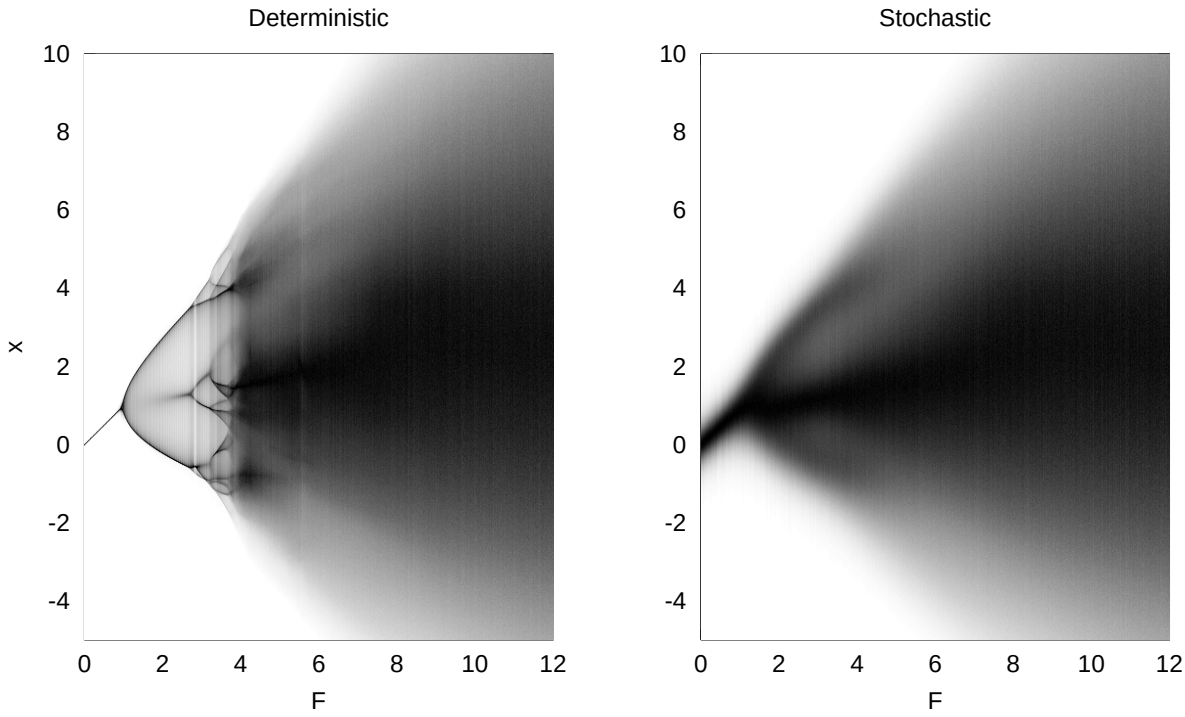

Figure 4: Density of the state variables $\mathbf{x}$ relative to the forcing parameter F in an eight-dimensional Lorenz '96 model. Densities are rescaled by column, to reveal the bifurcation of *(left)* the original deterministic model and *(right)* the modified stochastic model with state marginalized over σ^2. Note the way that decay behavior at $0 \leq F \leq 1$ yields to periodic behavior for $F > 1$, and eventually to chaotic behavior. The prescribed prior distribution $F \sim \mathrm{U}(8, 12)$ places the state space model firmly within the chaotic regime.

fixed step-size Δt and classic $\mathcal{O}(\Delta t^4)$ Runge-Kutta scheme. The same scheme yields a solution (Milstein and Tretyakov 2004) for the stochastic model by substituting the derivatives:

$$\frac{dx_n}{dt} = x_{n-1}(x_{n+1} - x_{n-2}) - x_n + F + \sigma \frac{\Delta W_n}{\Delta t},$$

where each noise term $\Delta W_n \sim \mathrm{N}(0, \sqrt{\Delta t})$ is an increment of the Wiener process over the time step of size Δt. This corresponds to a numerical solution to the SDEs (8) with strong order $\mathcal{O}(\Delta t)$, with the additional advantage that, for small noise, it approaches the $\mathcal{O}(\Delta t^4)$ deterministic solution[1].

The SSM consists of two parameters, $\boldsymbol{\Theta} \equiv (F, \sigma^2)^T$, along with a state vector $\mathbf{x}(t)$ and observation vector $\mathbf{y}(t)$, both of length 8. The complete model is specified in the **LibBi** modelling language in Figure 5.

In Figure 5, a dimension named `n`, of size 8, is first declared on line 5. It is given a cyclic boundary condition, recalling that, in (7), indices are interpreted cyclically. The two parameters of the model, `F` (F) and `sigma2` (σ^2) are then declared on lines 9–10, the state vector

[1]The scheme corresponds to that of (2.8) in Section 3.2.2 of Milstein and Tretyakov (2004). As noted on p. 185 of that work, for a system with additive noise scaled by some small ϵ, the scheme has strong order (mean squared error) $\mathcal{O}(\Delta t^4 + \epsilon \Delta t)$. While equivalent to $\mathcal{O}(\Delta t)$ in general, this can be close to $\mathcal{O}(\Delta t^4)$ if ϵ is small.

Lorenz96.bi

```
/**
 * Lorenz '96 model.
 */
model Lorenz96 {
  dim n(size = 8, boundary = 'cyclic')

  const h = 0.05    // step size

  param F           // forcing
  param sigma2      // diffusion variance
  state x[n]        // state variables
  noise deltaW[n]   // Wiener process increments
  obs y[n]          // observations

  sub parameter {
    F ~ uniform(8.0, 12.0)
    sigma2 ~ inverse_gamma(2.0, 0.9)
  }

  sub initial {
    x[n] ~ uniform(-1.0, 3.0)
  }

  sub transition(delta = h) {
    deltaW[n] ~ wiener()
    ode(h = h, alg = 'RK4') {
      dx[n]/dt = x[n-1]*(x[n+1] - x[n-2]) - x[n] + F +
          sqrt(sigma2)*deltaW[n]/h
    }
  }

  sub observation {
    y[n] ~ normal(x[n], 0.5)
  }

  sub proposal_parameter {
    F ~ truncated_gaussian(F, 0.1, 8.0, 12.0);
    sigma2 ~ inverse_gamma(2.0, 3.0*sigma2)
  }

  sub proposal_initial {
    x[n] ~ truncated_gaussian(x[n], 0.1, -1.0, 3.0)
  }
}
```

Figure 5: The Lorenz '96 SSM specified in the **LibBi** modelling language.

`x` ($\mathbf{x}(t)$) on line 11, the noise vector `deltaW` ($\Delta\mathbf{W}(t)$) on line 12, and the observation vector `y` ($\mathbf{y}(t)$) on line 13. The square bracket syntax is used to extend the vector variables over the previously declared dimension `n`. Note that while $\mathbf{x}(t)$, $\Delta\mathbf{W}(t)$ and $\mathbf{y}(t)$ all vary in time, there is no explicit declaration for the time dimension in **LibBi**.

Recall (2), which gives the general joint distribution of an SSM. The specific joint distribution for the Lorenz '96 SSM is:

$$\underbrace{p(\mathbf{y}(t_{1:T}), \mathbf{x}(t_{0:T}), F, \sigma^2)}_{\text{joint}} = \underbrace{p(F)p(\sigma^2)}_{\text{parameter}} \underbrace{p(\mathbf{x}(t_0))}_{\text{initial}} \tag{9}$$

$$\times \left(\prod_{i=1}^{T} \underbrace{p(\mathbf{x}(t_i)|\mathbf{x}(t_{i-1}), F, \sigma^2)}_{\text{transition}} \right) \left(\prod_{i=1}^{T} \underbrace{\prod_{j=1}^{N} p(y_j(t_i)|x_j(t_i))}_{\text{observation}} \right). \tag{10}$$

The prior distribution over parameters is given by:

$$\begin{aligned} F &\sim \mathrm{U}(8, 12) \\ \sigma^2 &\sim \textsc{Inv-Gamma}(2, 0.9), \end{aligned}$$

described in the `parameter` block on lines 15–18 of Figure 5. The prior distribution over the initial state of `x` is given by:

$$x_n(t_0) \sim \mathrm{U}(-1, 3),$$

described in the `initial` block on lines 20–22 of Figure 5.

The final form of the ODEs derived above is specified in the `transition` block on lines 24–29 of Figure 5. Notice the use of indexing on line 27 to show the relationship between elements of the `x` vector, without the use of a loop.

Each component of the vector `y` is an observation of the corresponding component of the vector `x`, with some additive Gaussian noise. The observation model is given by:

$$y_n \sim \mathrm{N}(x_n, 0.5),$$

where the second argument is the standard deviation. This is described in the `observation` block on lines 31–33 of Figure 5.

Two additional blocks are specified for the Lorenz '96 model. These are the `proposal_parameter` and `proposal_initial` blocks on lines 35–38 and 40–42, respectively. These specify the proposal distributions used for Metropolis-Hastings sampling, detailed in Section 3.2.

3. Inference methods

An SSM is constructed as the joint distribution $p\left(\mathbf{Y}(t_{1:T}), \mathbf{X}(t_{0:T}), \boldsymbol{\Theta}\right)$, typically factorized as in (2). The task of Bayesian inference is to condition this joint distribution on some particular data set, $\mathbf{Y}(t_{1:T}) = \mathbf{y}(t_{1:T})$, to obtain the posterior distribution $p(\mathbf{X}(t_{0:T}), \boldsymbol{\Theta}|\mathbf{y}(t_{1:T}))$.

The posterior distribution may be written:

$$\underbrace{p(\mathbf{X}(t_{0:T}), \boldsymbol{\Theta}|\mathbf{y}(t_{1:T}))}_{\text{posterior}} = p\left(\boldsymbol{\Theta}|\mathbf{y}(t_{1:T})\right) p\left(\mathbf{X}(t_{0:T})|\boldsymbol{\Theta}, \mathbf{y}(t_{1:T})\right). \tag{11}$$

Obtaining the first factor constitutes *parameter estimation*, while obtaining the second factor, conditioning on some particular $\boldsymbol{\Theta} = \boldsymbol{\theta}$ drawn from the first, constitutes *state estimation*. Methods for Bayesian inference will in limited cases derive a closed form for the posterior distribution. In most cases this is unachievable, however, and either an approximate closed form is fit, or Monte Carlo sampling is performed.

Because an SSM is a member of the broader BHM class, any method for inference over BHMs, such as Gibbs sampling or HMC, might be applied to SSMs also. Methods that do not exploit the additional hierarchical structure of SSMs may be sub-optimal, however. SSMs also commonly exhibit peculiarities such as strong autocorrelations in the state process relative to the observation frequency (e.g., Murray, Jones, and Parslow 2013), and strong correlations between the model parameters and state (e.g., Newman *et al.* 2009). The Gibbs sampler, a mainstay of inference for BHMs, is known to mix slowly in such conditions (Papaspiliopoulos, Roberts, and Sköld 2007). For particularly complex SSMs, it may be that no convenient closed form transition density exists (e.g., Beskos, Papaspiliopoulos, Roberts, and Fearnhead 2006; Fearnhead, Papaspiliopoulos, and Roberts 2008; Golightly and Wilkinson 2008, 2011; Murray and Storkey 2011; Murray *et al.* 2013). This precludes the analytical derivation of conditional distributions for Gibbs sampling, and even the pointwise evaluation of the prior density needed to apply Metropolis-Hastings or HMC approaches.

Fortunately, specialized Bayesian methods for these cases do exist. Those based on *sequential Monte Carlo* (SMC) are the focus of this work, and of the **LibBi** software. An important exception is the Kalman filter (Kalman 1960), which is optimal for a further specialization of the SSM class: that of linear-Gaussian models. An introduction to the Kalman filter is given here also.

We begin with state estimation in Section 3.1, which will lead into parameter estimation in Section 3.2.

3.1. State estimation

Sampling the second factor of (11), conditioned on a particular parameter setting $\boldsymbol{\Theta} = \boldsymbol{\theta}$, constitutes state estimation. This has been well-studied in the Bayesian filtering literature. Two methods, the Kalman filter and the particle filter, are introduced here.

The Kalman filter

The Kalman filter (Kalman 1960) can be used in the special case of SSMs where both the transition and observation densities are linear and Gaussian, and the initial state model is Gaussian[2]. It is optimal in such cases, producing the exact closed-form solution. Models that fit this class can be expressed in the following form:

$$\mathbf{X}(t_0) \sim \mathrm{N}\left(\boldsymbol{\mu}(t_0), \mathbf{U}(t_0)\right) \tag{12}$$

$$\mathbf{X}(t_i)|\mathbf{X}(t_{i-1}) \sim \mathrm{N}\left(\mathbf{F}^{\top}(t_i)\mathbf{X}(t_{i-1}), \mathbf{Q}(t_i)\right) \text{ for } i = 1, \ldots, T \tag{13}$$

$$\mathbf{Y}(t_i)|\mathbf{X}(t_i) \sim \mathrm{N}\left(\mathbf{G}^{\top}(t_i)\mathbf{X}(t_i), \mathbf{R}(t_i)\right) \text{ for } i = 1, \ldots, T, \tag{14}$$

where $\mathrm{N}(\boldsymbol{\mu}, \mathbf{U})$ denotes the normal distribution with mean vector $\boldsymbol{\mu}$ and square-root of the covariance matrix $\mathbf{U}$. It is understood that all symbols may depend on $\boldsymbol{\theta}$. The Kalman filter

[2]A fixed starting point is also admitted as a degenerate case of a Gaussian distribution.

is usually initialized with the mean vector $\boldsymbol{\mu}(t_0)$ and covariance matrix $\boldsymbol{\Sigma}(t_0)$ of the initial state $p(\mathbf{X}(t_0)|\boldsymbol{\theta})$. Here the *square-root* Kalman filter is presented, where the covariance matrix is replaced with its upper-triangular Cholesky square-root $\mathbf{U}(t_0)$, i.e., $\boldsymbol{\Sigma}(t_0) = \mathbf{U}^\top(t_0)\mathbf{U}(t_0)$. Using $\mathbf{U}(t_0)$ has certain numerical and computational advantages; indeed most computational manipulations of Gaussian distributions ultimately use the Cholesky square-root and not $\boldsymbol{\Sigma}(t_0)$ directly.

After initialization, the Kalman filter iterates through observation times with interleaved *prediction* and *correction* steps. Pseudocode is given in Algorithm 1. The prediction step computes $p(\mathbf{X}(t_{i-1:i})|\mathbf{y}(t_{1:i-1}), \boldsymbol{\theta})$, which is Gaussian with mean

$$\begin{pmatrix} \boldsymbol{\mu}(t_{i-1}) \\ \hat{\boldsymbol{\mu}}(t_i) \end{pmatrix}$$

and upper-triangular Cholesky factor of the covariance matrix

$$\begin{pmatrix} \mathbf{U}(t_{i-1}) & \mathbf{C}(t_i) \\ \mathbf{0} & \hat{\mathbf{U}}(t_i) \end{pmatrix}.$$

The cross term $\mathbf{C}(t_i)$ is used later. The correction step first computes $p(\mathbf{X}(t_i), \mathbf{Y}(t_i)|\mathbf{y}(t_{1:i-1}), \boldsymbol{\theta})$, also Gaussian, with mean

$$\begin{pmatrix} \hat{\boldsymbol{\mu}}(t_i) \\ \boldsymbol{\nu}(t_i) \end{pmatrix}$$

and upper-triangular Cholesky factor of the covariance matrix

$$\begin{pmatrix} \hat{\mathbf{U}}(t_i) & \mathbf{D}(t_i) \\ \mathbf{0} & \mathbf{V}(t_i) \end{pmatrix}.$$

It then conditions this on $\mathbf{Y}(t_i) = \mathbf{y}(t_i)$, in the usual fashion for a Gaussian distribution, to obtain $p(\mathbf{X}(t_i)|\mathbf{y}(t_{1:i}), \boldsymbol{\theta})$, Gaussian with mean $\boldsymbol{\mu}(t_i)$ and Cholesky factor $\mathbf{U}(t_i)$.

In the context of parameter estimation in Section 3.2, two further outputs are required of the Kalman filter. The first is to deliver the likelihood of $\boldsymbol{\theta}$, marginalized over $\mathbf{X}(t_{0:T})$. The computation is given on line 15 of Algorithm 1 ($|\cdot|$ denotes the matrix determinant and $\|\cdot\|$ the Euclidean norm). The second requirement is to deliver a single sample $\mathbf{x}'(t_{0:T}) \sim p(\mathbf{X}(t_{0:T})|\boldsymbol{\theta}, \mathbf{y}(t_{1:T}))$. This is achieved with a backward pass through time at the end of Algorithm 1, in a manner similar to the Rauch-Tung-Striebel smoother (Rauch, Tung, and Striebel 1965).

A number of Kalman-type filters exist for nonlinear and non-Gaussian models. These include the mixture-of-Gaussians (Alspach and Sorenson 1972), extended (Smith, Schmidt, and McGee 1962), ensemble (Evensen 1994) and unscented (Julier and Uhlmann 1997) Kalman filters and accompanying smoothers (Rauch *et al.* 1965; Särkkä 2008). However, for nonlinear and non-Gaussian cases these are approximate methods that introduce bias. The *particle filter* (Gordon, Salmond, and Smith 1993; Doucet *et al.* 2001) is an alternative that does not, and admits any SSM of the form (2). The particle filter is the focus for the nonlinear case in this work. The approximate Kalman-type filters are not treated.

Algorithm 1 The Kalman filter for given $\boldsymbol{\theta}$. This is presented over a time interval $[t_r, t_s]$, returning the marginal likelihood $l = p(\mathbf{y}(t_{0:s})|\boldsymbol{\theta})$ and a state sample $\mathbf{x}'(t_{0:s}) \sim p(\mathbf{X}(t_{0:s})|\boldsymbol{\theta}, \mathbf{y}(t_{0:s}))$, so that it may be called from Algorithms 3 and 4 later. The usual, standalone algorithm sets $r = 0$ and $s = T$.

KALMAN-FILTER$(\boldsymbol{\theta}, t_{r:s}) \to (l, \mathbf{x}'(t_{0:s}))$

if $r = 0$
  initialize $\mathbf{X}(t_0) \sim \mathrm{N}(\boldsymbol{\mu}(t_0), \mathbf{U}(t_0))$
for $i = r+1, \ldots, s$
  // state prediction
  $\hat{\boldsymbol{\mu}}(t_i) \leftarrow \mathbf{F}_{\boldsymbol{\theta}}(t_i)\boldsymbol{\mu}(t_{i-1})$
  $\mathbf{C}(t_i) \leftarrow \mathbf{U}(t_{i-1})^\top \mathbf{U}(t_{i-1})\mathbf{F}_{\boldsymbol{\theta}}(t_i)$
  $\hat{\boldsymbol{\Sigma}}(t_i) \leftarrow \mathbf{F}_{\boldsymbol{\theta}}(t_i)^\top \mathbf{U}(t_{i-1})^\top \mathbf{U}(t_{i-1})\mathbf{F}_{\boldsymbol{\theta}}(t_i) + \mathbf{Q}_{\boldsymbol{\theta}}(t_i)^\top \mathbf{Q}_{\boldsymbol{\theta}}(t_i)$
  $\hat{\mathbf{U}}(t_i) \leftarrow$ CHOLESKY-FACTORIZE$(\hat{\boldsymbol{\Sigma}}(t_i))$

  // observation prediction
  $\boldsymbol{\nu}(t_i) \leftarrow \mathbf{G}_{\boldsymbol{\theta}}(t_i)\hat{\boldsymbol{\mu}}(t_i)$
  $\mathbf{D}(t_i) \leftarrow \hat{\mathbf{U}}(t_i)^\top \hat{\mathbf{U}}(t_i)\mathbf{G}_{\boldsymbol{\theta}}(t_i)$
  $\mathbf{T}(t_i) \leftarrow \mathbf{G}_{\boldsymbol{\theta}}(t_i)^\top \hat{\mathbf{U}}(t_i)\hat{\mathbf{U}}(t_i)^\top \mathbf{G}_{\boldsymbol{\theta}}(t_i) + \mathbf{R}_{\boldsymbol{\theta}}(t_i)^\top \mathbf{R}_{\boldsymbol{\theta}}(t_i)$
  $\mathbf{V}(t_i) \leftarrow$ CHOLESKY-FACTORIZE$(\mathbf{T}(t_i))$

  // correction
  $\mathbf{K} \leftarrow \mathbf{D}(t_i)\mathbf{V}(t_i)^{-1}$
  $\boldsymbol{\mu}(t_i) \leftarrow \hat{\boldsymbol{\mu}}(t_i) + \mathbf{K}\mathbf{V}(t_i)^{-\top}(\mathbf{y}(t_i) - \boldsymbol{\nu}(t_i))$
  $\mathbf{U}(t_i) \leftarrow$ CHOLESKY-DOWNDATE$(\hat{\mathbf{U}}(t_i), \mathbf{K})$

$l \leftarrow \prod_{i=1}^{s} \frac{1}{\sqrt{2\pi}|\mathbf{V}(t_i)|} \exp\left(-\frac{1}{2}\left\|\mathbf{V}^{-1}(t_i)\left(\mathbf{y}(t_i) - \boldsymbol{\nu}(t_i)\right)\right\|^2\right)$ // marginal likelihood

// single state sample
$\mathbf{x}'(t_s) \sim \mathrm{N}(\boldsymbol{\mu}(t_s), \mathbf{U}(t_s))$
for $i = s-1, \ldots, 0$
  $\mathbf{K} \leftarrow \mathbf{C}(t_{i+1})\hat{\mathbf{U}}(t_{i+1})^{-1}$
  $\boldsymbol{\omega} \leftarrow \boldsymbol{\mu}(t_i) + \mathbf{K}\hat{\mathbf{U}}(t_{i+1})^{-\top}(\mathbf{x}'(t_{i+1}) - \hat{\boldsymbol{\mu}}(t_{i+1}))$
  $\mathbf{W} \leftarrow$ CHOLESKY-FACTORIZE$(\mathbf{U}(t_i)^\top \mathbf{U}(t_i) - \mathbf{K}^\top \mathbf{K})$
  $\mathbf{x}'(t_i) \sim \mathrm{N}(\boldsymbol{\omega}, \mathbf{W})$

return $(l, \mathbf{x}'(t_{0:s}))$

The particle filter

The particle filter (Gordon *et al.* 1993; Doucet *et al.* 2001), of the family of SMC methods, can be used for any SSM as defined by (2). For this general form an analytical solution is not forthcoming, and the particle filter instead relies on importance sampling. Numerous variants are available, but the most basic, that of the *bootstrap* particle filter (Gordon *et al.* 1993), is

Algorithm 2 The bootstrap particle filter for given $\boldsymbol{\theta}$. This is presented over a time interval $[t_r, t_s]$, returning an estimate of the marginal likelihood $\hat{l} = p(\mathbf{y}(t_{0:s})|\boldsymbol{\theta})$ and an unbiased state sample $\hat{\mathbf{x}}'(t_{0:s}) \sim p(\mathbf{X}(t_{0:s})|\boldsymbol{\theta}, \mathbf{y}(t_{0:s}))$, so that it may be called from Algorithms 3 and 4 later. The usual, standalone algorithm sets $r = 0$ and $s = T$.

PARTICLE-FILTER($\boldsymbol{\theta}, t_{r:s}$) $\rightarrow (\hat{l}, \hat{\mathbf{x}}'(t_{0:s}))$

1 **if** $r = 0$
2 **foreach** $j \in \{1, \ldots, P_x\}$
3 $\mathbf{x}^j(t_0) \sim p(\mathbf{X}_0|\boldsymbol{\theta})$ // initialize particle j
4 $w^j(t_0) \leftarrow 1/P_x$ // initialize weight j
5 **for** $i = r+1, \ldots, s$
6 **foreach** $j \in \{1, \ldots, P_x\}$
7 $a^j(t_i) \sim$ MULTINOMIAL($\mathbf{w}(t_{i-1})$) // ancestor for particle j
8 $\mathbf{x}^j(t_i) \sim p(\mathbf{X}(t_i)|\mathbf{x}^{a^j(t_i)}(t_{i-1}), \boldsymbol{\theta})$ // propagate particle j
9 $w^j(t_i) \leftarrow p(\mathbf{y}(t_i)|\mathbf{x}^j(t_i), \boldsymbol{\theta})$ // weight particle j
10 $\hat{l} \leftarrow \prod_{i=1}^{s} \left(\frac{1}{P_x} \sum_{j=1}^{P_x} w^j(t_i) \right)$ // marginal likelihood

 // single state sample
11 $j \sim$ MULTINOMIAL($\mathbf{w}(t_s)$)
12 $\hat{\mathbf{x}}'(t_s) \leftarrow \mathbf{x}^j(t_s)$
13 **for** $i = s, \ldots, 1$
14 $j \leftarrow a^j(t_i)$
15 $\hat{\mathbf{x}}'(t_{i-1}) \leftarrow \mathbf{x}^j(t_{i-1})$

16 **return** $(\hat{l}, \hat{\mathbf{x}}'(t_{0:s}))$

described in pseudocode in Algorithm 2 and visualized in Figure 6.

For a given $\boldsymbol{\theta}$, the particle filter is initialized by drawing P_x number of random samples, $\mathbf{x}^j(t_0) \sim p(\mathbf{X}(t_0)|\boldsymbol{\theta})$, for $j = 1, \ldots, P_x$, and weighting each uniformly with $w^j(t_0) = 1/P_x$. These are referred to as *particles*. It proceeds sequentially through observation times through a series of *propagation*, *weighting* and *resampling* steps. In the propagation step each particle is advanced to the next observation time with $\mathbf{x}^j(t_i) \sim p(\mathbf{X}(t_i)|\mathbf{x}^{a^j(t_i)}(t_{i-1}), \boldsymbol{\theta})$, where $a^j(t_i)$ is the index of the particle's *ancestor* at the previous time, t_{i-1} (more below). It is then weighted with the likelihood of the new observation, $w^j(t_i) = p(\mathbf{y}(t_i)|\mathbf{x}^j(t_i), \boldsymbol{\theta})$. The resampling step restores the particle stock to equal weights by resampling particles with replacement, where the probability of each particle being drawn is proportional to its weight w^j. Particles with high weight tend to be replicated, while particles with low weight tend to be eliminated. It is this process that determines the ancestor indices for the next time propagation.

The particle filter can be used to compute an unbiased estimate of the likelihood of $\boldsymbol{\theta}$, marginalized over $\mathbf{X}(t_{0:T})$. The computation (Del Moral 2004) is given on line 10 of Algorithm 2. It can also produce a sample $\hat{\mathbf{x}}'(t_{0:T}) \sim p(\mathbf{X}(t_{0:T})|\boldsymbol{\theta}, \mathbf{y}(t_{1:T}))$ by tracing a single particle back through its ancestry (Andrieu *et al.* 2010). This occurs as the last step in Algorithm 2.

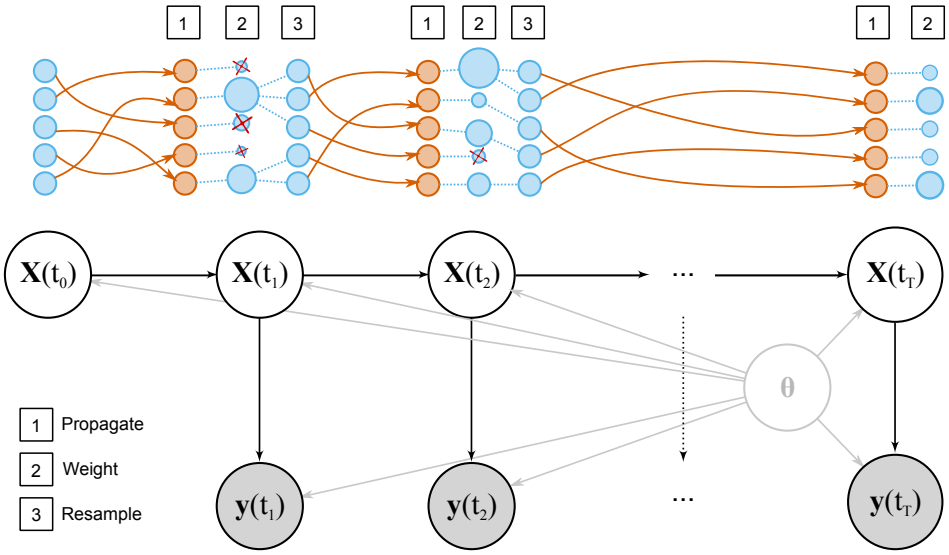

Figure 6: Visualization of the particle filter algorithm where, at each time, particles proceed through propagation, weighting and resampling steps. Particles may be eliminated in each resampling step, with remaining particles forming an ancestry tree.

3.2. Parameter estimation

Sampling from the first factor of (11) constitutes parameter estimation, for which two approaches are given here. The first is marginal Metropolis-Hastings (Metropolis, Rosenbluth, Rosenbluth, Teller, and Teller 1953; Hastings 1970), of the family of Markov chain Monte Carlo (MCMC), using either a Kalman filter to compute the likelihood of parameters, marginalized over the state, or a particle filter to estimate it. When a particle filter is used the approach is more specifically known as particle marginal Metropolis-Hastings (PMMH), from the family of particle Markov chain Monte Carlo (PMCMC) methods (Andrieu *et al.* 2010). The second approach is sequential Monte Carlo (SMC, Del Moral, Doucet, and Jasra 2006). Recall that the particle filter for state estimation is a type of SMC algorithm, indeed it is prototypical. Similar methods may be employed for parameter estimation, and this is considered here. Within the SMC algorithm over parameters, a Kalman or particle filter is used to compute weights; when the latter is used the method is more specifically known as SMC^2 (Chopin *et al.* 2013).

Marginal Metropolis-Hastings

The marginal Metropolis-Hastings algorithm is given in Algorithm 3. It is initialized with some arbitrary setting of parameters $\boldsymbol{\theta}^0$, then for $i = 1, 2, \ldots$, a new setting $\boldsymbol{\theta}' \sim q(\boldsymbol{\theta}'|\boldsymbol{\theta})$ is proposed from some proposal distribution q. The move is accepted with probability given by the Metropolis-Hastings rule:

$$\min\left(1, \frac{p(\mathbf{y}(t_{1:T})|\boldsymbol{\theta}')p(\boldsymbol{\theta}')q(\boldsymbol{\theta}|\boldsymbol{\theta}')}{p(\mathbf{y}(t_{1:T})|\boldsymbol{\theta})p(\boldsymbol{\theta})q(\boldsymbol{\theta}'|\boldsymbol{\theta})}\right). \tag{15}$$

If accepted then $\boldsymbol{\theta}^i = \boldsymbol{\theta}'$, otherwise $\boldsymbol{\theta}^i = \boldsymbol{\theta}^{i-1}$. By construction, the Markov chain $\boldsymbol{\theta}^0, \boldsymbol{\theta}^1, \ldots$ is ergodic to the posterior distribution, so that, after convergence, states of the chain may

Algorithm 3 Marginal Metropolis-Hastings for parameter estimation. When the particle filter is used (i.e., PARTICLE-FILTER for FILTER), the algorithm gives particle marginal Metropolis-Hastings (PMMH, Andrieu *et al.* 2010).

MARGINAL-METROPOLIS-HASTINGS$(\boldsymbol{\theta}, \mathbf{x}(t_{0:T}), l) \rightarrow (\boldsymbol{\theta}, \mathbf{x}(t_{0:T}), l)$

1 $\boldsymbol{\theta}' \sim q(\boldsymbol{\theta}'|\boldsymbol{\theta})$ // propose
2 $(l', \mathbf{x}'(t_{0:T})) \leftarrow$ FILTER$(\boldsymbol{\theta}', t_{0:T})$ // likelihood and state sample from filter
3 $\alpha \sim \mathrm{U}(0,1)$
4 **if** $\alpha \leq \frac{l' p(\boldsymbol{\theta}') q(\boldsymbol{\theta}|\boldsymbol{\theta}')}{l p(\boldsymbol{\theta}) q(\boldsymbol{\theta}'|\boldsymbol{\theta})}$
5 **return** $(\boldsymbol{\theta}', \mathbf{x}'(t_{0:T}), l')$ // accept
6 **else**
7 **return** $(\boldsymbol{\theta}, \mathbf{x}(t_{0:T}), l)$ // reject

be considered samples from it. The process continues until as many samples as desired have been drawn, a number denoted P_θ.

Either a Kalman or particle filter may be used to compute or estimate the marginal likelihood term $p(\mathbf{y}(t_{1:T})|\boldsymbol{\theta}')$ in (15) at each step. When estimated, a valid sampler is still obtained (Andrieu *et al.* 2010). The single state sample drawn from the Kalman or particle filter completes the sample with a draw from the second factor of (11).

Sequential Monte Carlo

An alternative approach to sampling the first factor of (11) is to replace the MCMC over parameters with SMC over parameters. SMC over parameters works similarly to SMC over state variables. It is initialized by drawing P_θ number of random samples from the prior distribution over parameters, $p(\boldsymbol{\Theta})$, and weighting them uniformly with $v^j(t_0) = 1/P_\theta$ for $j = 1, \ldots, P_\theta$. These are referred to as *θ-particles* and *θ-weights*. It proceeds sequentially through observation times with a series of propagation, weighting and resampling steps, just as for the particle filter for state estimation, along with a new *rejuvenation* step. Pseudocode for the algorithm is given in Algorithm 4.

To each θ-particle is attached a Kalman or particle filter. Propagating a θ-particle involves advancing the attached filter through time. Weighting of a θ-particle uses the marginal likelihood $p(\mathbf{y}(t_i)|\boldsymbol{\theta}^j(t_i))$, computed by the attached Kalman filter or estimated by the attached particle filter. Weighting with an unbiased estimate of that marginal likelihood gives a valid SMC algorithm as in the random-weight particle filter (Fearnhead *et al.* 2008; Fearnhead, Papaspiliopoulos, Roberts, and Stuart 2010), and is more specifically called SMC^2 (Chopin *et al.* 2013). Resampling involves drawing a new set of unweighted θ-particles by weight, along with their attached filters.

The resampling of θ-particles at each time t_i depletes the number of unique values represented. Resampling has the same effect in the particle filter for state estimation, but in that case particles diversify again in the next propagation step. As parameters do not change in time, they cannot diversify in this way. To fix this, an additional step, called the *rejuvenation* step, is inserted after the resampling of θ-particles (Chopin *et al.* 2013). The aim of the step is to diversify the values of θ-particles while preserving their distribution. To this end it is

Algorithm 4 Sequential Monte Carlo (SMC) for parameter estimation. When the particle filter is used (i.e., Particle-Filter for Filter), the algorithm gives SMC^2 (Chopin *et al.* 2013).

Sequential-Monte-Carlo

1 **foreach** $j \in \{1, \dots, P_\theta\}$
2 $\quad \boldsymbol{\theta}^j(t_0) \sim p(\boldsymbol{\theta})$ // initialize θ-particle j
3 $\quad v^j(t_0) \leftarrow 1/P_\theta$ // initialize θ-weight j
4 $\quad \mathbf{x}^j(t_0) \sim p(\mathbf{X}_0|\boldsymbol{\theta}^j(t_0))$ // initialize state sample j
5 $\quad l^j(t_0) \leftarrow 1$ // initialize likelihood j
6 **for** $i = 1, \dots, T$
7 $\quad$ **foreach** $j \in \{1, \dots, P_\theta\}$
8 $\quad\quad a^j(t_i) \sim \text{Multinomial}(\mathbf{v}(t_{i-1}))$ // ancestor for θ-particle j
9 $\quad\quad (\boldsymbol{\theta}^j(t_i), \mathbf{x}^j(t_{0:i-1}), l^j(t_{i-1})) \leftarrow \text{Marginal-Metropolis-Hastings}($
$\quad\quad\quad \boldsymbol{\theta}^{a^j(t_i)}(t_{i-1}), \mathbf{x}^{a^j(t_i)}(t_{0:i-1}), l^{a^j(t_i)}(t_{i-1}))$ // rejuvenate θ-particle j
10 $\quad\quad (v^j(t_i), \mathbf{x}^j(t_{0:i})) \leftarrow \text{Filter}(\boldsymbol{\theta}^j(t_i), t_{i-1:i})$ // propagate and weight θ-particle j

sufficient to take a single marginal Metropolis-Hastings step for each θ-particle, $\boldsymbol{\theta}^j(t_i)$. This works by proposing a move to a new value $\boldsymbol{\theta}'(t_i) \sim q(\boldsymbol{\theta}'(t_i)|\boldsymbol{\theta}^j(t_i))$, estimating the marginal likelihood $\hat{p}(\mathbf{y}(t_{1:i})|\boldsymbol{\theta}'(t_i))$ with the Kalman or particle filter, and then accepting or rejecting the move using the acceptance probability (15). Line 7 of Algorithm 4 achieves this by calling the Marginal-Metropolis-Hastings function of Algorithm 3.

3.3. Parallelization

The methods presented exhibit varying degrees of parallelisability, and therefore suitability to modern computing hardware. SMC is particularly promising in this regard (Lee *et al.* 2010). The propagation, weighting and (in the case of parameter estimation) rejuvenation steps can be performed in parallel for each (θ-)particle. There is limited scope for parallelization of the Kalman filter: its matrix operations can be multithreaded or even performed on GPU, but this will only be faster for very large matrices (Song, Tomov, and Dongarra 2012), and may be slower for small matrices.

For the particle filter, the minimum degree of parallelism is P_x, the number of particles, with the potential for further parallelization according to model-specific structure. The bottleneck in high-performance implementation of the particle filter is the resampling step. Resampling algorithms, such as the multinomial, systematic or stratified (Kitagawa 1996) approaches, require a collective operation over the weight vector. This means that all threads must synchronize. The development of asynchronous resampling methods to alleviate this bottleneck is still an active area of research (Murray 2011; Del Moral, Jacob, Lee, Murray, and Peters 2013; Whiteley, Lee, and Heine 2013; Murray, Lee, and Jacob 2014; Lee and Whiteley. 2014). In the meantime, the particle filter is best suited to shared-memory architectures where the collective operation can be performed efficiently, although approaches for distributed-memory resampling have been proposed (Bolić, Djurić, and Hong 2005).

Marginal Metropolis-Hastings is itself a sequential method, but inherits the degree of par-

allelism of the filter used at each step. Again, the Kalman filter offers limited scope, but using the particle filter (the PMMH sampler) gives P_x-way parallelism. The setting of P_x is complicated in this context, however. The tradeoff is to maximize the mixing rate of the Metropolis-Hastings chain against the computational expense of running P_x particles. The optimal choice relates to the variance in the marginal likelihood estimator (Pitt, Silva, Giordani, and Kohn 2012; Doucet, Pitt, and Kohn 2013; Murray *et al.* 2013). Increasing P_x decreases this variance, but does not necessarily improve the real-time mixing of the Metropolis-Hastings chain for a fixed computational budget. Because arbitrarily increasing P_x to consume all available hardware resources has depreciating returns on mixing rate, there are limits on the degree of parallelism of PMMH.

A possible solution is to run multiple marginal Metropolis-Hastings chains. This also has limits. In practice, one usually removes some number, B_θ, of steps from the start of each chain to correct for the initialization bias of $\boldsymbol{\theta}^0$. B_θ depends on the autocorrelation of the chain, and should be sufficiently large for the influence of $\boldsymbol{\theta}^0$ to be forgotten. With multiple chains, each chain must take at least B_θ steps before drawing one or more samples that will be preserved. This imposes a serial limitation on the maximum speedup achievable by parallelization with multiple chains, which may be quantified by Amdahl's law (Amdahl 1967) as P_θ/B_θ. The performance gains of multiple chains might therefore be disappointing without some additional strategy to reduce B_θ. Adaptation (Gilks, Roberts, and Sahu 1998; Haario, Saksman, and Tamminen 2001; Andrieu and Thoms 2008) and tempering (Frantz, Freeman, and Doll 1990; Geyer 1991; Marinari and Parisi 1992; Geyer and Thompson 1995; Neal 1996) are both established means of reducing B_θ for single chains. Using these for each chain in isolation can reduce B_θ, but only by a factor that is independent of the number of chains. Reducing B_θ relative to the number of chains is to be preferred. Population MCMC (Laskey and Myers 2003) attempts this via an evolutionary (Back 1996) selection, crossing and mutation of multiple chains. Craiu, Rosenthal, and Yang (2009) more explicitly target the posterior with an ensemble of chains, using the covariance of samples across all chains to adapt the proposal covariance of individual chains. This remains an active area of research.

Using SMC for parameter estimation gives at least a P_θ-way parallelism, even with the Kalman filter. Using the particle filter (the SMC^2 method) has a much higher degree of parallelism, at $P_\theta P_x$. This is very promising. As for the particle filter, however, the resampling step is synchronous, which can be a bottleneck to the scaling of the algorithm. This also remains an active area of research.

3.4. Examples

The example models introduced in Section 2 are used within **LibBi** to demonstrate the inference methods above. The three-element windkessel model is an example where Kalman filter-based methods are suitable, while the nonlinear Lorenz '96 model requires the particle filter. In both cases simulated data sets are used.

LibBi provides a `libbi` command to access its functionality from the command line. It is used as follows:

```
libbi command [options...]
```

where `command` is the particular command to execute, followed by zero or more command-

prior.conf

```
--target prior
--model-file Windkessel.bi
--nsamples 50000
--start-time 0.0
--end-time 2.4
--noutputs 240
--input-file data/input.nc
--output-file results/prior.nc
```

posterior.conf

```
--target posterior
--model-file Windkessel.bi
--filter kalman
--nsamples 50000
--start-time 0.0
--end-time 2.4
--noutputs 240
--input-file data/input.nc
--obs-file data/obs.nc
--output-file results/posterior.nc
```

Figure 7: **LibBi** configuration files, containing command-line options, for the the windkessel example.

line options to specify input files and configuration parameters. The pertinent command for these examples is `sample`, which draws samples from the joint, prior or posterior distribution. Options may be given on the command line itself, or, as is often convenient, by listing them in a configuration file and giving the name of that file on the command line instead, like so:

```
libbi sample @config.conf
```

All of the examples use configuration files in this way. The contents of these files are given in Figure 7.

Windkessel example

The three-element windkessel model has Gaussian initial state, linear and Gaussian transition model, and linear and Gaussian observation model. Kalman filter-based methods are suitable in this case. The model can be coerced into the matrix form of (12–14), but it is not necessary to do so by hand: **LibBi** uses symbolic differentiation to derive the matrix form internally.

Consequently, no changes are required to the model file, as given in Figure 3, to apply the Kalman filter.

The windkessel model requires two input files: one giving the values of the input variable $F(t)$, and one giving the values of the observed variable $P_a(t)$. The input is shown in the top-left plot of Figure 8, and observations in the top-right. These have been prepared in advance as NetCDF files `data/input.nc` and `data/obs.nc`, available in the supplementary materials (see Section 6).

A first step is often to simulate the model's prior distribution. This is useful to validate the model specification. Command-line options for this purpose are given in the `prior.conf` file in Figure 7. The command is then:

```
libbi sample @prior.conf
```

with results output to the NetCDF file `results/prior.nc` and shown in the top-right of Figure 8. The posterior distribution can be sampled using the options in the `posterior.conf` file in Figure 7. Note in particular the `--filter kalman` option to indicate that a Kalman filter should be used for state estimates. The default sampling method is marginal Metropolis-Hastings, so no options are required to select this. The command is:

```
libbi sample @posterior.conf
```

with results output to the NetCDF file `results/posterior.nc`. SMC does work to sample from the posterior distribution of parameters also. It may be selected by adding the `--sampler sir` option (see the `posterior_sir.conf` file in the supplementary materials). State estimates are shown in the top-right of Figure 8, and parameter estimates in the lower four plots.

Lorenz '96 example

A similar procedure is followed for the Lorenz '96 model, although a prediction forward in time will also be made. Because it is a nonlinear model, the particle filter is used for state estimation and marginal likelihood estimates.

The Lorenz '96 model requires one input file, containing the observations. This has been prepared in advance as `data/obs_sparse.nc`, available in the supplementary materials (see Section 6). It contains observations of the first four components of $\mathbf{y}(t)$ at every other time step (recall $\Delta t = 0.05$) on the interval $t = [0, 3]$. It is an example of how sparse or missing data can be handled by **LibBi**. The aim is to condition the joint distribution on those observations that fall in the interval $t = [0, 2]$, and set aside the remainder to validate a forward prediction on the interval $t = (2, 3]$.

The first task is to simulate the prior distribution. The `prior.conf` configuration file in Figure 9 is set up for this purpose. The command:

```
libbi sample @prior.conf
```

outputs samples to the `results/prior.nc` file. These are shown in grey in Figures 11 and 12. The posterior is sampled with options from the `posterior.conf` configuration file of Figure 9. The particle filter is the default option for state estimation, and marginal Metropolis-Hastings for parameter estimation, so no options to select an appropriate method are required. The command is:

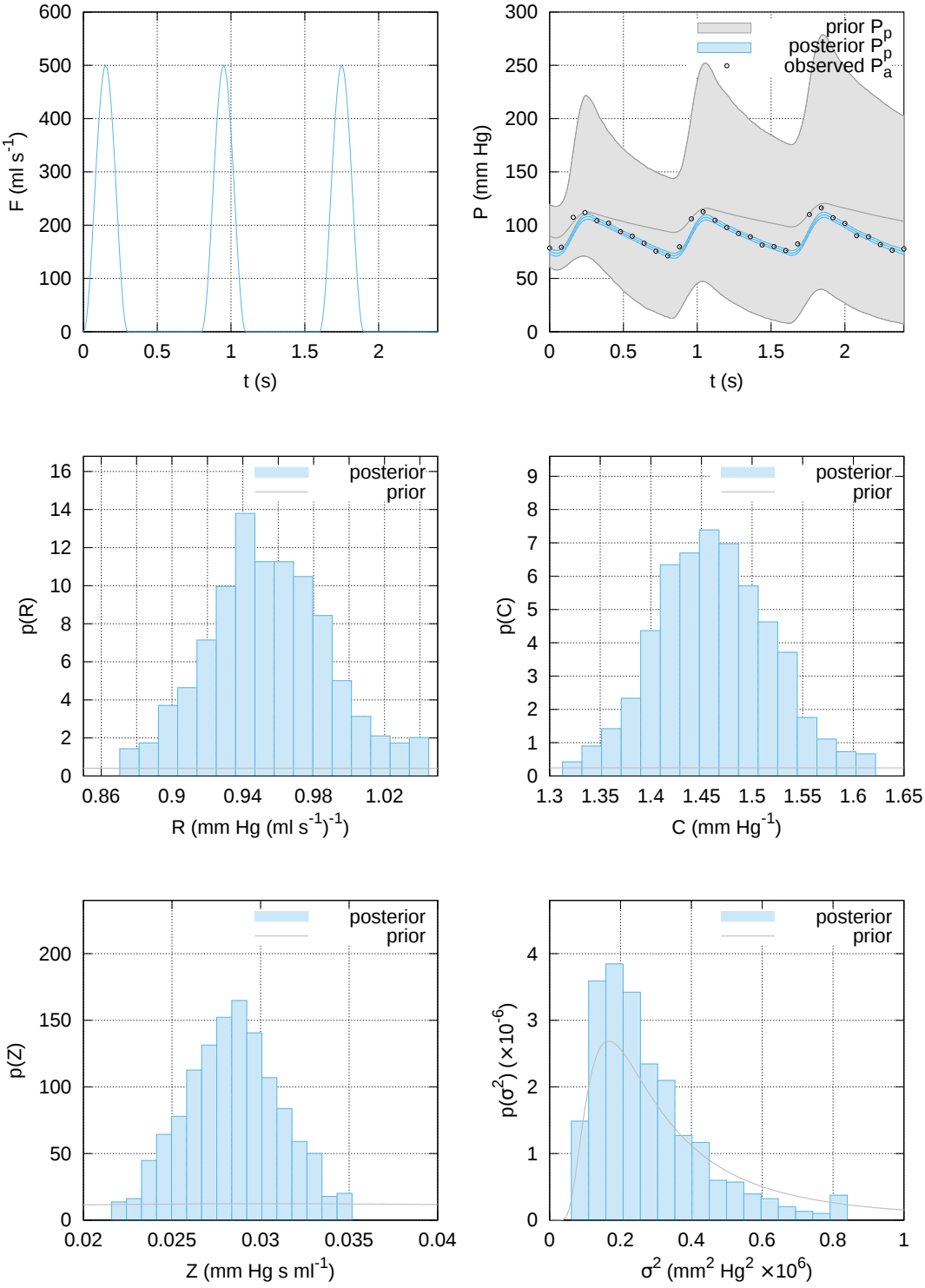

Figure 8: *(top left)* input blood flow $F(t)$, *(top right)* prior and posterior distribution of the state $P_p(t)$, bold lines giving the median and shaded regions the 95% credibility interval at each time, with observations $P_a(t)$ plotted over the top, and *(remainder)* histograms of the posterior samples of parameters against prior distributions. Note that the 95% credibility interval for the state estimate of $P_p(t)$ in the top right is difficult to distinguish owing to its narrowness.

prior.conf

```
--target prior
--model-file Lorenz96.bi
--end-time 3
--noutputs 60
--nsamples 100000
--output-file results/prior.nc
```

posterior.conf

```
--target posterior
--model-file Lorenz96.bi
--end-time 2
--noutputs 40
--nsamples 100000
--nparticles 512
--obs-file data/obs_sparse.nc
--output-file results/posterior.nc
--with-transform-initial-to-param
```

prediction.conf

```
--target prediction
--model-file Lorenz96.bi
--start-time 2
--end-time 3
--noutputs 20
--nsamples 100000
--init-file results/posterior.nc
--output-file results/prediction.nc
```

Figure 9: **LibBi** configuration files, containing command-line options, for the the Lorenz '96 example.

```
libbi sample @posterior.conf
```

Results are output to `results/posterior.nc`, and plot in blue in Figures 10–12. SMC^2 does work to sample from the posterior distribution also. It may be selected by adding the `--sampler sir` option (see the `posterior_sir.conf` file in the supplementary materials).

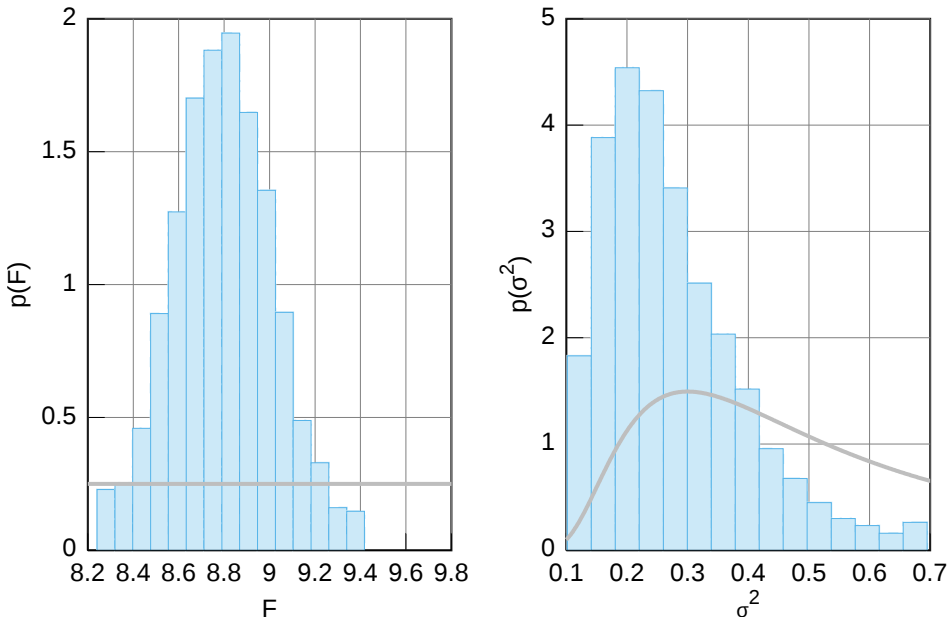

Figure 10: Parameter estimates for the Lorenz '96 case study, *(left)* F and *(right)* σ^2. In both cases the blue histogram summarizes posterior samples, while the grey curve denotes the prior distribution.

Finally, a prediction can be made. The configuration file `prediction.conf` of Figure 9 is set up for this. The output file of the posterior sample is used as an input file for the prediction (`--init-file results/posterior.nc`) so as to extend the state estimate forward in time to form a posterior prediction. The command is:

```
libbi sample @prediction.conf
```

Results are output to `results/prediction.nc`, and plot in red in Figures 11–12. Note that the observations on $t = (2, 3]$ do not factor in to the prediction, they are shown in Figure 11 only for validation of the prediction.

4. The LibBi software

LibBi is made up of a C++ template library and Perl frontend. The C++ template library provides the inference methods, along with supporting functionality for, among other things, input and output, memory management, matrices and vectors, and pseudorandom number generation. The Perl frontend parses the **LibBi** modelling language, generates model-specific C++ code using the Perl Template Toolkit (Wardley 2013), uses a GNU Autotools (GNU 2013)build system to compile and link this against the C++ template library, and finally runs the program. Code is configured for the hardware platform according to options set by the user when calling the `libbi` command. **LibBi** supports several hardware architectures and high-performance computing technologies, including SIMD vector operations on CPU using SSE and AVX, multithreading on multicore CPUs using OpenMP (OpenMP Architecture Review Board 2013), general-purpose GPU programming on NVIDIA GPUs using CUDA (Compute Unified Device Architecture. NVIDIA Corporation 2013), and distributed-memory

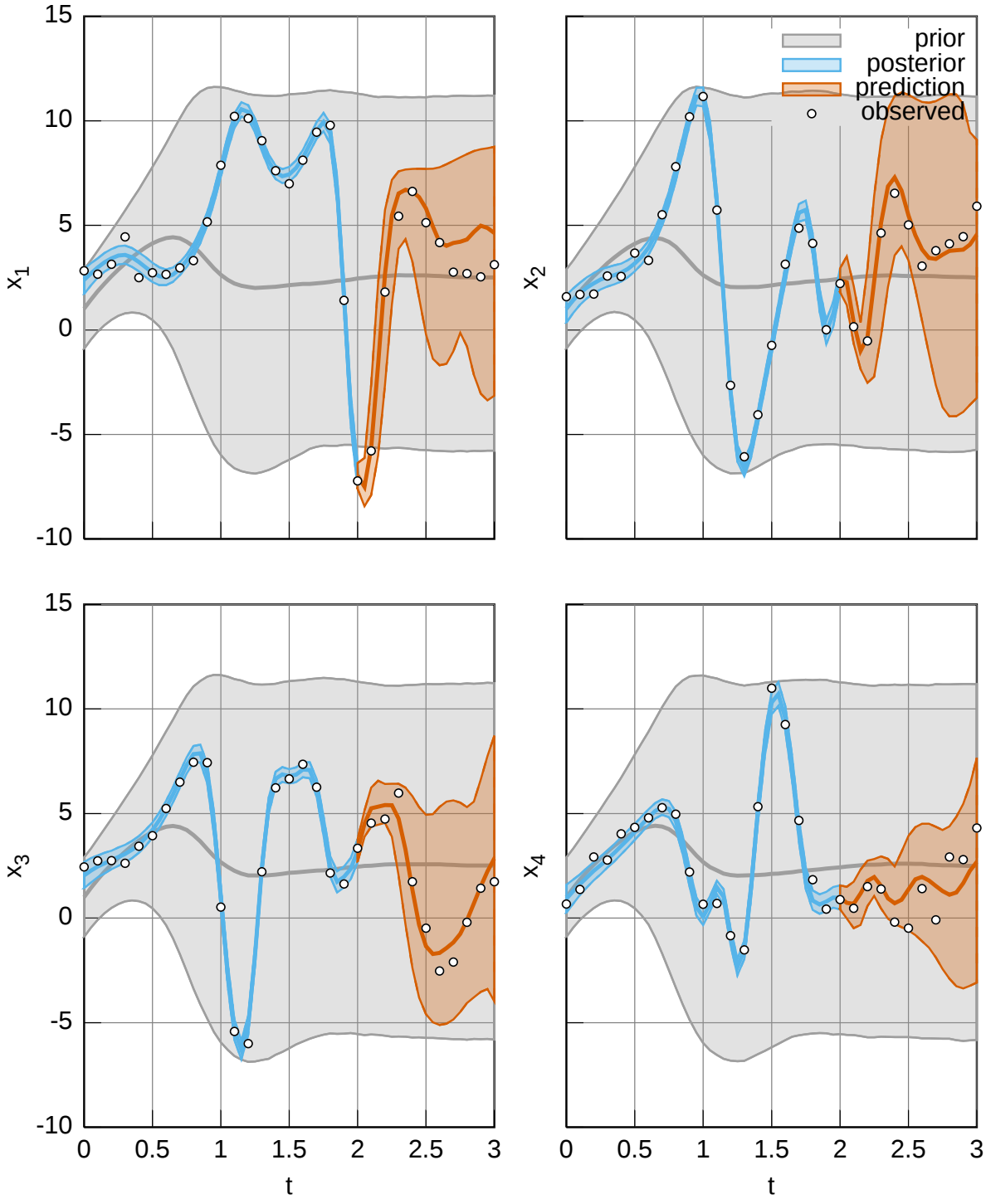

Figure 11: State estimates for the Lorenz '96 case study, giving the four components of the state vector $\mathbf{x}(t)$ that are observed with additional noise. Grey denotes the prior distribution, blue the posterior and red the prediction. For each, the central line gives the time-marginal median and the shaded region about it the 95% credibility interval. White circles indicate the observations. Observations that fall in the time interval of prediction are witheld from the inference, but shown for validation purposes.

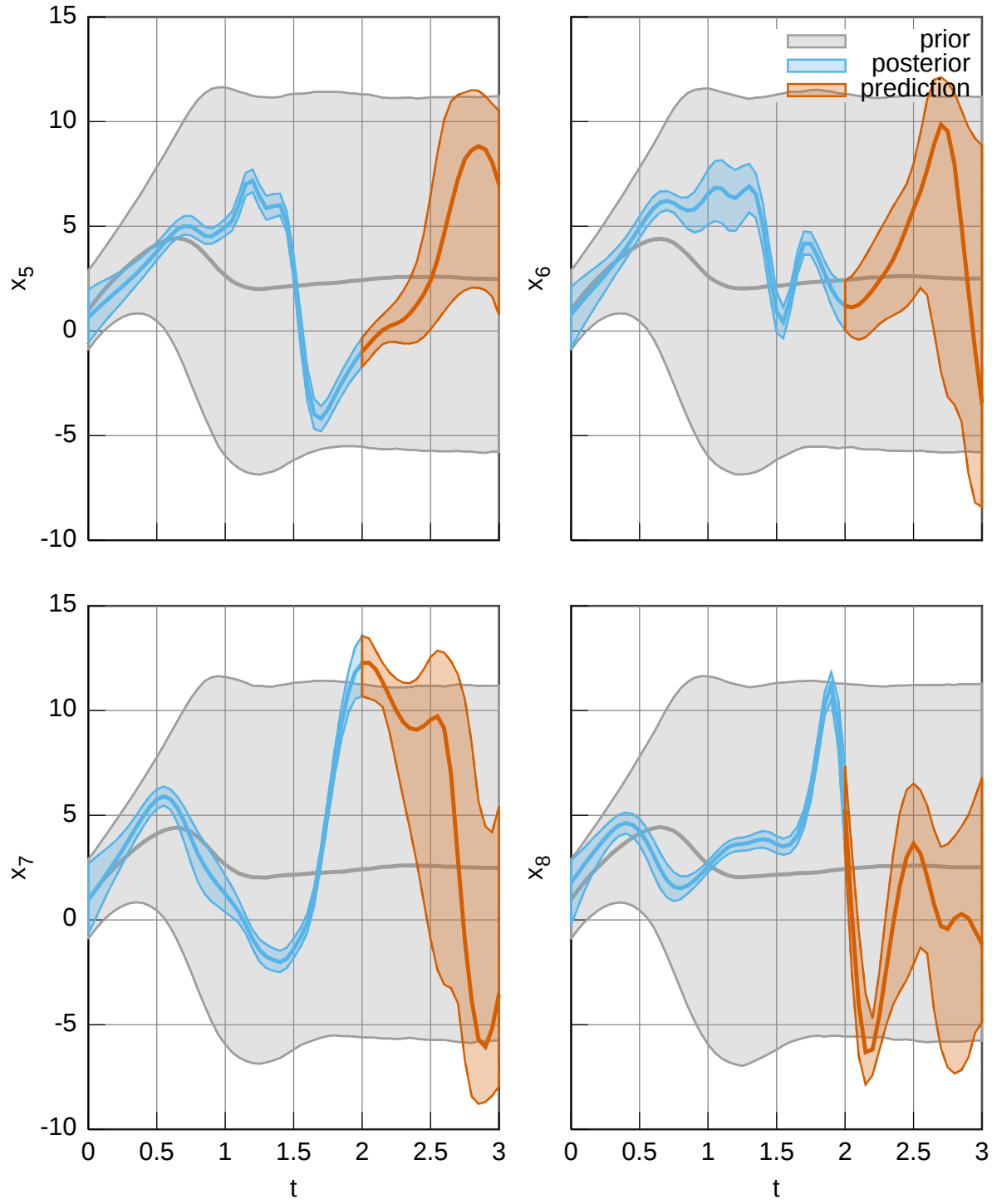

Figure 12: State estimates for the Lorenz '96 case study, giving the four components of the state vector $\mathbf{x}(t)$ that are not observed. See Figure 11 for explanation of components.

computing using MPI (Message Passing Interface, MPI Forum 2013). The compilation process is hidden from the user. Intermediate files are preserved in a hidden directory to avoid repeated effort, although a short wait is noticeable after making changes to a model that require recompilation.

A number of algorithmic innovations are realized in the **LibBi** software. Among these are parallelized implementations of numerical integrators for ODEs (Murray 2012), alternative resampling algorithms for SMC (Murray 2011; Murray *et al.* 2014), and the memory-efficient storage of the SMC ancestry tree (Jacob, Murray, and Rubenthaler 2015).

The **LibBi** modelling language is described with an LALR grammar (DeRemer 1969). Tools such as **Yacc** (Johnson 2013) and GNU **Bison** (Donnelly and Stallman 2013)may be used to compile parsers from the grammar. **Yapp** (Desarmenien 2013), a **Yacc**-like parser compiler for Perl, is used for this purpose. Input and output files use the standard NetCDF (UniData 2013) format, based on HDF5 (HDF Group 2013), and are readily created and analyzed within various mathematical and statistical packages such as MATLAB (The MathWorks, Inc. 2013), R (R Core Team 2015), and GNU Octave (Eaton 2002). The workflow for **LibBi** typically sees the user prepare input files in their preferred statistical package, run **LibBi** from the command-line, then return to their statistical package to summarize and visualize the results. It is hoped that closer integration will be realized in future.

Existing software for general BHMs exists, including the BUGS pair **WinBUGS** (Lunn, Thomas, Best, and Spiegelhalter 2000) and more recently **OpenBUGS** (Lunn, Jackson, Best, Thomas, and Spiegelhalter 2012), as well as **JAGS** (Plummer 2013, 2003) and **Stan** (**Stan** Development Team 2015). Specialist software for SSMs also exists, notably **BiiPS** (Todeschini and Caron 2013). It is against these programs that **LibBi** might be most closely compared. All of these existing packages accept models specified in the BUGS language or extensions of it (Plummer 2003), while **LibBi** prescribes its own, albeit similar, language. **WinBUGS**, **OpenBUGS**, **JAGS** and **BiiPS** build a data structure from the model which is manipulated internally by the client program, while **Stan** and **LibBi** take a code generation approach. **WinBUGS**, **OpenBUGS** and **JAGS** target general BHMs, while **BiiPS** and **LibBi** target the more-specific class of SSMs. In line with this more specialized focus, **BiiPS** and **LibBi** use SMC as a staple method, rather than Gibbs (as in **WinBUGS**, **OpenBUGS** and **JAGS**) or HMC (as in **Stan**, see Hoffman and Gelman 2014). The most notable distinction of **LibBi** against all of these packages is its high-performance computing focus, where its hardware support, particularly for GPUs and distributed clusters, is unique.

The C++ template library component of **LibBi** might also be compared to similar libraries such as **SMCTC** (Johansen 2009). The template metaprogramming techniques advocated in Johansen (2009) are mirrored in **LibBi**. But where a library requires its user to program their model to a prescribed interface, an additional code generator component, as in **LibBi**, automates compliance with the interface from a higher-level language in which the model is specified. In **LibBi** this also facilitates the generation of code for different combinations of methods and hardware.

4.1. Suitability

The suitability of **LibBi** for a particular problem is inherited from the SMC-based methods upon which it relies. While clearly model dependent, some advice can be given.

At each time step, SMC methods draw an importance sample; the efficacy of this hinges on

	Intel Xeon E5-2650 CPU	Nvidia Tesla S2050 GPU
Task parallism	8-way	14-way
Data parallism (per task)	8-way*	32-way†
Clock rate	2.00 GHz	1.15 GHz
Cache	20480 KB	768 KB

Table 1: Capabilities of the CPU and GPU devices used for empirical experiments, from a programming perspective. Specifications are obtained from `/proc/cpuinfo` for the CPU, and the CUDA SDK `deviceQuery` program for the GPU. $*$ = Using single-precision AVX instructions, 4-way for double precision. Earlier generation CPUs can use SSE instructions, 4-way for single and 2-way for double precision, and indeed **LibBi** is limited to this at time of writing. † = Threads per warp.

the design of a good proposal distribution. As this becomes difficult in high dimensions, SMC tends to be better suited to models with few state variables. For PMMH and SMC^2, a critical measure of performance is the variance of the estimate of the marginal likelihood delivered by SMC (see e.g., Pitt *et al.* 2012; Doucet *et al.* 2013; Murray *et al.* 2013). This variance is known (Cérou, Del Moral, and Guyader 2011) to scale linearly with the length of the time series, T, so that the number of particles, P_x, should also scale linearly with T. As such, while few state variables are preferred, reasonably long time series are manageable. Large problems that can be structured in this way may be tractable.

A particular advantage of SMC methods is that they can accommodate highly complex transition models. In the most basic SMC methods, the transition model is only ever simulated, and not evaluated pointwise, so that it is unnecessary that it have a closed form density function. This permits SMC to be applied to models that cannot be handled by, for example, Gibbs sampling or HMC.

To date, **LibBi** has been used for published studies in soil carbon (Clifford *et al.* 2014) and marine biogeochemistry (Parslow *et al.* 2013), in both cases utilising its PMMH implementation. The latter of these studies uses a complex nonlinear differential model of 15 parameters, 15 state variables, nine noise terms, nine exogenous inputs and two observed variables, with an observed time series of 227 points. These dimensions may provide some anecdotal guidance as to what can be achieved. Work to scale the methods and software to larger models is ongoing.

4.2. Performance results

This section offers a comparison of **LibBi** performance under different hardware configurations. The Lorenz '96 model is used for this purpose. Experiments are conducted on a single machine with two 8-core Intel Xeon E5-2650 CPUs, three NVIDIA Tesla 2075 GPUs, and 128 GB main memory. The important technical specifications of these devices are given in Table 1. The particle filter, PMMH and SMC^2 methods described in Section 3 are configured according to Table 2 using hardware configurations in Table 3. All methods are tested with the first six configurations in Table 3. The SMC^2 method is also tested with the last two configurations, which use MPI to distribute θ-particles across multiple processes.

Figure 13 gives the execution times for each method in each configuration, summarized across 100 repeated runs with different random number seeds. For context, one would ideally like

	P_θ `--nsamples` *n*	P_x `--nparticles` *n*	T `--noutputs` *n*	t_T `--end-time` *n*
Particle filter	1	8192	40	2
PMCMC	500	8192	40	2
SMC^2	384	8192	40	2

Table 2: Method configurations for timing results.

	Processes (MPI) `--enable-mpi --mpi-np` *n*	Threads/process (OpenMP) `--nthreads` *n*	SIMD enabled (AVX) `--enable-avx`	GPU enabled (CUDA) `--enable-cuda`
#1	1	1		
#2	1	4		
#3	1	1	✓	
#4	1	4	✓	
#5	1	1		✓
#6	1	4		✓
#7	3	4	✓	
#8	3	4		✓

Table 3: System configurations for timing results. Headings additionally give the technology (e.g., MPI) and **LibBi** command-line option (e.g., `--enable-mpi`) used to enable it. All configurations additionally use the `--disable-assert` command-line option to disable runtime assertion checks.

to see:

1. that configurations #2 and #4 execute four times faster than configurations #1 and #3, respectively, due to OpenMP multithreading,
2. that configurations #3 and #4 execute eight times faster than configurations #1 and #2, respectively, due to four-way AVX vector parallelism in single precision floating point,
3. that configurations #5 and #6 execute faster than configuration #4, as the algorithms are well-suited to GPU, and,
4. that configurations #7 and #8 execute three times faster than configurations #4 and #6, respectively, due to three MPI processes on uncontested hardware.

Of course, these ideal targets are rarely met in practice due to synchronization overhead and necessarily serial sections of code. Results in Figure 13 do suggest good gains with OpenMP, AVX and GPU use, however, and modest gains with MPI for a CPU (but not GPU) configuration. Improving gains with MPI is a topic of current research.

Performance of these methods is very much model-dependent, and the results of Figure 13 should be considered demonstrative only. For example, **LibBi** parallelizes predominantly (but not only, e.g., Murray 2012) across particles. For GPU performance to exceed CPU-only performance, typically at least a thousand particles will be required, and this may increase with future hardware. The number of x-particles per θ sample in these configurations (Table 2)

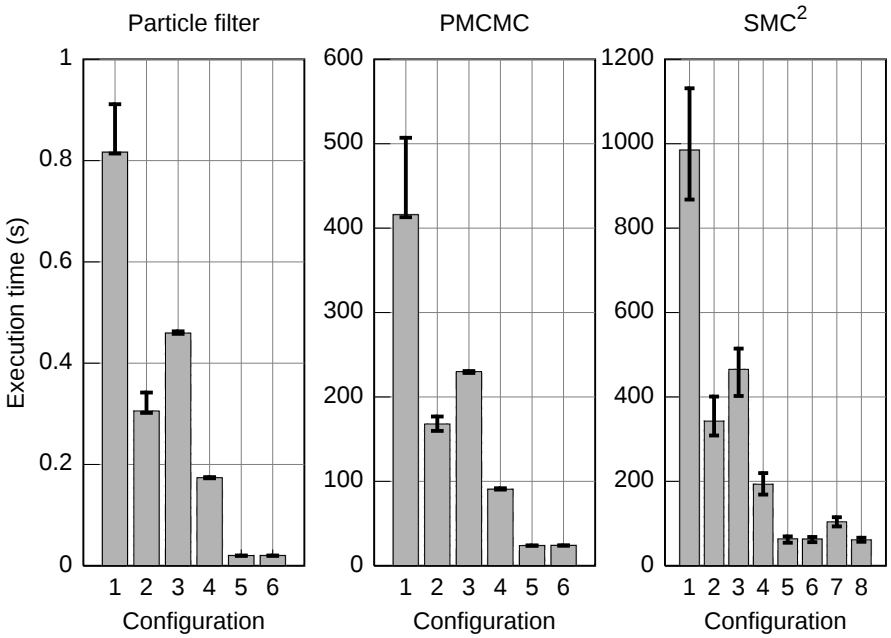

Figure 13: Execution times for each method under each configuration. Results are based on 100 runs with different random number seeds. Columns give the median, while error bars straddle the 50% credibility interval.

is sufficiently large that good returns are expected. Not all models justify the use of this many x-particles, however, and for fewer, CPU performance can, and in many cases does, exceed GPU performance.

5. Summary

LibBi combines the utility the SSMs with the generality and computational scalability of SMC methods, in a manner sensitive to modern hardware realities. Its two aims are accessibility and speed. Accessibility is achieved by decoupling models, from methods, from the hardware on which they run. Models are specified in a modelling language with a rich variety of features, the focus on SMC methods ensures broad inferential support, and code generation alleviates the user from the combinatorial task of writing model-, method- and hardware-specific code. Speed is achieved through code generation and template metaprogramming techniques, along with support for high-performance technologies such as SSE, AVX, OpenMP, MPI and CUDA to make full use of available hardware resources, including GPUs. Good performance gains are obtained with these technologies as demonstrated in Figure 13. Research continues into both statistical and computational advances to scale to higher-dimensional models on large compute clusters.

6. Supplementary material

The two examples given in this work are available in the supplementary files along with this manuscript or for download at `http://www.libbi.org/examples.html`, as the **Windkessel**

and **Lorenz96** packages. Each package includes the model, data, and input files required to reproduce the results in this work. A number of other examples are also available.

References

Alspach D, Sorenson H (1972). "Nonlinear Bayesian Estimation Using Gaussian Sum Approximations." *IEEE Transactions on Automatic Control*, **17**, 439–448. doi:10.1109/tac.1972.1100034.

Amdahl G (1967). "Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities." *AFIPS Conference Proceedings*, **30**, 483–485. doi:10.1145/1465482.1465560.

Andrieu C, Doucet A, Holenstein R (2010). "Particle Markov Chain Monte Carlo Methods." *Journal of the Royal Statistical Society B*, **72**, 269–302. doi:10.1111/j.1467-9868.2009.00736.x.

Andrieu C, Thoms J (2008). "A Tutorial on Adaptive MCMC." *Statistical Computing*, **18**, 343–373. doi:10.1007/s11222-008-9110-y.

Back T (1996). *Evolutionary Algorithms in Theory and Practice*. Oxford University Press.

Beskos A, Papaspiliopoulos O, Roberts GO, Fearnhead P (2006). "Exact and Computationally Efficient Likelihood-Based Estimation for Discretely Observed Diffusion Processes." *Journal of the Royal Statistical Society B*, **68**, 333–382. doi:10.1111/j.1467-9868.2006.00552.x.

Bolić M, Djurić PM, Hong S (2005). "Resampling Algorithms and Architectures for Distributed Particle Filters." *IEEE Transactions on Signal Processing*, **53**, 2442–2450. doi:10.1109/tsp.2005.849185.

Cérou F, Del Moral P, Guyader A (2011). "A Nonasymptotic Theorem for Unnormalized Feynman-Kac Particle Models." *Annales de l'Institut Henri Poincaré, Probabilités et Statistiques*, **47**(3), 629–649. doi:10.1214/10-aihp358.

Chopin N, Jacob PE, Papaspiliopoulos O (2013). "SMC^2: An Efficient Algorithm for Sequential Analysis of State Space Models." *Journal of the Royal Statistical Society B*, **75**, 397–426. doi:10.1111/j.1467-9868.2012.01046.x.

Clifford D, Pagendam D, Baldock J, Cressie N, Farquharson R, Farrell M, Macdonald L, Murray L (2014). "Rethinking Soil Carbon Modelling: A Stochastic Approach to Quantify Uncertainties." *Environmetrics*, **25**(4), 265–278. doi:10.1002/env.2271.

Craiu R, Rosenthal J, Yang C (2009). "Learn From Thy Neighbor: Parallel-Chain and Regional Adaptive MCMC." *Journal of the American Statistical Association*, **104**, 1454–1466. doi:10.1198/jasa.2009.tm08393.

Del Moral P (2004). *Feynman-Kac Formulae: Genealogical and Interacting Particle Systems with Applications*. Springer-Verlag.

Del Moral P, Doucet A, Jasra A (2006). "Sequential Monte Carlo Samplers." *Journal of the Royal Statistical Society B*, **68**, 441–436. doi:10.1111/j.1467-9868.2006.00553.x.

Del Moral P, Jacob P, Lee A, Murray L, Peters G (2013). "Feynman-Kac Particle Integration with Geometric Interacting Jumps." *Journal of Stochastic Analysis and Applications*, **31**(5), 830–871. doi:10.1080/07362994.2013.817247.

DeRemer F (1969). *Practical Translators for LR(k) Languages*. Ph.D. thesis, Massachusetts Institute of Technology, Cambridge.

Desarmenien F (2013). "**Yapp**." URL http://search.cpan.org/~fdesar/Parse-Yapp-1.05/yapp/.

Donnelly C, Stallman R (2013). "**Bison**: GNU Parser Generator." URL http://www.gnu.org/software/bison/.

Doucet A, Freitas N, Gordon N (eds.) (2001). *Sequential Monte Carlo Methods in Practice*. Springer-Verlag.

Doucet A, Pitt M, Kohn R (2013). "Efficient Implementation of Markov Chain Monte Carlo When Using an Unbiased Likelihood Estimator." URL http://arxiv.org/abs/1210.1871.

Dowd M (2006). "A Sequential Monte Carlo Approach for Marine Ecological Prediction." *Environmetrics*, **17**, 435–455. doi:10.1002/env.780.

Dowd M (2011). "Estimating Parameters for a Stochastic Dynamic Marine Ecological System." *Environmetrics*, **22**(4), 501–515. doi:10.1002/env.1083.

Eaton J (2002). *GNU Octave Manual*. Network Theory Limited. ISBN 0-9541617-2-6.

Evensen G (1994). "Sequential Data Assimilation with a Nonlinear Quasi-Geostrophic Model Using Monte-Carlo Methods to Forecast Error Statistics." *Journal of Geophysical Research-Oceans*, **99**(C5), 10143–10162. doi:10.1029/94jc00572.

Fearnhead P, Papaspiliopoulos O, Roberts G (2008). "Particle Filters for Partially Observed Diffusions." *Journal of the Royal Statistical Society B*, **70**, 755–777. doi:10.1111/j.1467-9868.2008.00661.x.

Fearnhead P, Papaspiliopoulos O, Roberts G, Stuart A (2010). "Random-Weight Particle Filtering of Continuous Time Processes." *Journal of the Royal Statistical Society B*, **72**(4), 497–512. doi:10.1111/j.1467-9868.2010.00744.x.

Frank O (1899). "Die Grundform des arteriellen Pulses. Erste Abhandlung: Mathematische Analyse." *Zeitschrift fuer Biologie*, **37**, 483–526.

Frantz DD, Freeman DL, Doll JD (1990). "Reducing Quasi-Ergodic Behavior in Monte Carlo Simulations by J-Walking: Applications to Atomic Clusters." *Journal of Chemical Physics*, **93**, 2769–2784. doi:10.1063/1.458863.

Gardiner C (2004). *Handbook of Stochastic Methods for Physics, Chemistry and the Natural Sciences*. 3rd edition. Springer-Verlag.

Geman S, Geman D (1984). "Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **6**, 721–741. doi:10.1109/tpami.1984.4767596.

Geyer C, Thompson E (1995). "Annealing Markov Chain Monte Carlo with Applications to Ancestral Inference." *Journal of the American Statistical Association*, **90**, 909–920. doi:10.1080/01621459.1995.10476590.

Geyer CJ (1991). "Markov Chain Monte Carlo Maximum Likelihood." In ED Keramidas (ed.), *Computing Science and Statistics: Proceedings of the 23rd Symposium on the Interface*, pp. 156–163. doi:10.1080/01621459.1995.10476590.

Gilks W, Roberts G, Sahu S (1998). "Adaptive Markov Chain Monte Carlo through Regeneration." *Journal of the American Statistical Association*, **93**, 1045–1054. doi:10.1080/01621459.1998.10473766.

GNU (2013). "GNU Autotools." URL http://www.gnu.org/software/.

Golightly A, Wilkinson D (2011). "Bayesian Parameter Inference for Stochastic Biochemical Network Models Using Particle Markov Chain Monte Carlo." *Interface Focus*, **1**, 807–820.

Golightly A, Wilkinson DJ (2008). "Bayesian Inference for Nonlinear Multivariate Diffusion Models Observed with Error." *Computational Statistics & Data Analysis*, **52**, 1674–1693. doi:10.1016/j.csda.2007.05.019.

Gordon NJ, Salmond DJ, Smith AFM (1993). "Novel Approach to Nonlinear/Non-Gaussian Bayesian State Estimation." *IEE Proceedings-F*, **140**, 107–113. doi:10.1049/ip-f-2.1993.0015.

Haario H, Saksman E, Tamminen J (2001). "An Adaptive Metropolis Algorithm." *Bernoulli*, **7**, 223–242. doi:10.2307/3318737.

Hastings WK (1970). "Monte Carlo Sampling Methods Using Markov Chains and Their Applications." *Biometrika*, **57**, 97–109.

HDF Group (2013). "HDF5." Hierarchical Data Format, URL http://www.hdfgroup.org/HDF5/.

Hoffman M, Gelman A (2014). "The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo." *Journal of Machine Learning Research*, **15**, 1593–1623. In press.

Hosack G, Peters G, Hayes, R K (2012). "Estimating Density Dependence and Latent Population Trajectories with Unknown Observation Error." *Methods in Ecology and Evolution*, **3**(6), 1028–1038. doi:10.1111/j.2041-210x.2012.00218.x.

Jacob P, Murray L, Rubenthaler S (2015). "Path Storage in the Particle Filter." *Statistics and Computing*, **25**(2), 487–496. doi:10.1007/s11222-013-9445-x.

Johansen AM (2009). "**SMCTC**: Sequential Monte Carlo in C++." *Journal of Statistical Software*, **30**(6), 1–41. doi:10.18637/jss.v030.i06.

Johnson S (2013). "**Yacc**." URL http://dinosaur.compilertools.net/yacc/.

Jones E, Parslow J, Murray L (2010). "A Bayesian Approach to State and Parameter Estimation in a Phytoplankton-Zooplankton Model." *Australian Meteorological and Oceanographic Journal*, **59**(SP), 7–16.

Julier S, Uhlmann J (1997). "A New Extension of the Kalman Filter to Nonlinear Systems." In *The Proceedings of AeroSense: The 11th International Symposium on Aerospace/Defense Sensing, Simulation and Controls, Multi Sensor Fusion, Tracking and Resource Management*. doi:10.1117/12.280797.

Kalman RE (1960). "A New Approach to Linear Filtering and Prediction Problems." *Journal of Basic Engineering*, **82**, 35–45. doi:10.1115/1.3662552.

Kind T, Faes T, Lankhaar JW, Vonk-Noordegraaf A, Verhaegen M (2010). "Estimation of Three- And Four-Element Windkessel Parameters Using Subspace Model Identification." *IEEE Transactions on Biomedical Engineering*, **57**, 1531–1538. doi:10.1109/tbme.2010.2041351.

Kitagawa G (1996). "Monte Carlo Filter and Smoother for Non-Gaussian Nonlinear State Space Models." *Journal of Computational and Graphical Statistics*, **5**, 1–25. doi:10.2307/1390750.

Kloeden P, Platen E (1992). *Numerical Solution of Stochastic Differential Equations*. Springer-Verlag. doi:10.1007/978-3-662-12616-5.

Laskey K, Myers J (2003). "Population Markov Chain Monte Carlo." *Machine Learning*, **50**, 175–196. doi:10.1023/a:1020206129842.

Lee A, Whiteley N (2014). "Forest Resampling for Distributed Sequential Monte Carlo." URL http://arxiv.org/abs/1406.6010.

Lee A, Yau C, Giles M, Doucet A, Holmes C (2010). "On the Utility of Graphics Cards to Perform Massively Parallel Simulation of Advanced Monte Carlo Methods." *Journal of Computational and Graphical Statistics*, **19**, 769–789. doi:10.1198/jcgs.2010.10039.

Lorenz E (2006). "3: Predictability – A Problem Partly Solved." In T Palmer, R Hagedorn (eds.), *Predictability of Weather and Climate*, pp. 40–58. Cambridge University Press. doi:10.1017/cbo9780511617652.004.

Lunn D, Jackson C, Best N, Thomas A, Spiegelhalter D (2012). *The BUGS Book: A Practical Introduction to Bayesian Analysis*. CRC Press / Chapman and Hall.

Lunn D, Thomas A, Best N, Spiegelhalter D (2000). "**WinBUGS** – A Bayesian Modelling Framework: Concepts, Structure and Extensibility." *Statistics and Computing*, **10**, 325–337. doi:10.1023/a:1008929526011.

Marinari E, Parisi G (1992). "Simulated Tempering: A New Monte Carlo Scheme." *Europhysics Letters*, **19**, 451–458.

Metropolis N, Rosenbluth AW, Rosenbluth MN, Teller AH, Teller E (1953). "Equation of State Calculations by Fast Computing Machines." *Journal of Chemical Physics*, **21**, 1087–1092. doi:10.1063/1.1699114.

Milstein G, Tretyakov M (2004). *Stochastic Numerics for Mathematical Physics*. Scientific Computation. Springer-Verlag. doi:10.1007/978-3-662-10063-9.

MPI Forum (2013). "MPI: Message Passing Interface." URL http://www.mpi-forum.org/.

Murray L (2011). "GPU Acceleration of the Particle Filter: The Metropolis Resampler." In *DMMD: Distributed Machine Learning and Sparse Representation with Massive Data Sets*. URL http://arxiv.org/abs/1202.6163.

Murray L (2012). "GPU Acceleration of Runge-Kutta Integrators." *IEEE Transactions on Parallel and Distributed Systems*, **23**, 94–101. doi:10.1109/tpds.2011.61.

Murray L (2013). "**LibBi**: Library for Bayesian Inference." URL http://www.libbi.org/.

Murray L, Jones E, Parslow J (2013). "On Disturbance State-Space Models and the Particle Marginal Metropolis-Hastings Sampler." *SIAM/ASA Journal of Uncertainty Quantification*, **1**(1), 494–521. doi:10.1137/130915376.

Murray L, Lee A, Jacob P (2014). "Parallel Resampling in the Particle Filter." In review, URL http://arxiv.org/abs/1301.4019.

Murray L, Storkey A (2008). "Continuous Time Particle Filtering for fMRI." In *Advances in Neural Information Processing Systems*, volume 20, pp. 1049–1056. MIT Press.

Murray L, Storkey A (2011). "Particle Smoothing in Continuous Time: A Fast Approach via Density Estimation." *IEEE Transactions on Signal Processing*, **59**, 1017–1026. doi:10.1109/tsp.2010.2096418.

Neal R (1996). "Sampling from Multimodal Distributions Using Tempered Transitions." *Statistics and Computing*, **6**, 353–366. doi:10.1007/bf00143556.

Neal R (2011). "5: MCMC Using Hamiltonian Dynamic." In S Brooks, A Gelman, G Jones, XL Meng (eds.), *Handbook of Markov Chain Monte Carlo*, pp. 113–162. CRC Press / Chapman and Hall.

Newman K, Fernández C, Thomas L, Buckland S (2009). "Monte Carlo Inference for State-Space Models of Wild Animal Populations." *Biometrics*, **65**(2), 572–583. doi:10.1111/j.1541-0420.2008.01073.x.

NVIDIA Corporation (2013). "CUDA: Compute Unified Device Architecture." URL http://www.nvidia.com/cuda/.

OpenMP Architecture Review Board (2013). "OpenMP: The OpenMP API Specification for Parallel Computing." URL http://www.openmp.org/.

Papaspiliopoulos O, Roberts G, Sköld M (2007). "A General Framework for the Parameterisation of Hierarchical Models." *Statistical Science*, **22**(1), 59–73. doi:10.1214/088342307000000014.

Parslow J, Cressie N, Campbell E, Jones E, Murray L (2013). "Bayesian Learning and Predictability in a Stochastic Nonlinear Dynamical Model." *Ecological Applications*, **23**(4), 679–698. doi:10.1890/12-0312.1.

Peters G, Hosack G, Hayes K (2011). "Ecological Non-Linear State Space Model Selection via Adaptive Particle Markov Chain Monte Carlo (AdPMCMC)." URL http://arxiv.org/abs/1005.2238.

Pitt M, 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. doi:10.1016/j.jeconom.2012.06.004.

Plummer M (2003). "**JAGS**: A Program for Analysis of Bayesian Graphical Models Using Gibbs Sampling." In K Hornik, F Leisch, A Zeileis (eds.), *Proceedings of the 3rd International Workshop on Distributed Statistical Computing, Vienna, Austria*. ISSN 1609-395X, URL http://www.ci.tuwien.ac.at/Conferences/DSC-2003/Proceedings/.

Plummer M (2013). "**JAGS**: Just Another Gibbs Sampler." URL http://mcmc-jags.sourceforge.net/.

Rauch H, Tung F, Striebel C (1965). "Maximum Likelihood Estimates of Linear Dynamic Systems." *AIAA Journal*, **3**, 1445–1450. doi:10.2514/3.3166.

R Core Team (2015). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL http://www.R-project.org/.

Riera J, Watanabe J, Kazuki I, Naoki M, Aubert E, Ozaki T, Kawashim R (2004). "A State-Space Model of the Hemodynamic Approach: Nonlinear Filtering of BOLD Signals." *NeuroImage*, **21**, 547–567. doi:10.1016/j.neuroimage.2003.09.052.

Ross P (2008). "Why CPU Frequency Stalled: The Data." *IEEE Spectrum*. doi:10.1109/mspec.2008.4476447.

Särkkä S (2008). "Unscented Rauch-Tung-Striebel Smoother." *IEEE Transactions on Automated Control*, **53**, 845–849. doi:10.1109/tac.2008.919531.

Smith G, Schmidt S, McGee L (1962). "Application of Statistical Filter Theory to the Optimal Estimation of Position and Velocity On Board a Circumlunar Vehicle." *Technical report*, National Aeronautics and Space Administration.

Song F, Tomov S, Dongarra J (2012). "Enabling and Scaling Matrix Computations on Heterogeneous Multi-Core and Multi-GPU Systems." In *Proceedings of the 26th ACM International Conference on Supercomputing*, pp. 365–376. ACM, New York. doi:10.1145/2304576.2304625.

Stan Development Team (2015). "**Stan**: A C++ Library for Probability and Sampling, Version 2.8.0." URL http://mc-stan.org/.

Sutter H (2005). "The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software." *Dr. Dobb's Journal*, **30**. Online version updates CPU trend results to 2009, URL http://www.gotw.ca/publications/concurrency-ddj.htm.

The MathWorks, Inc (2013). *MATLAB – The Language of Technical Computing, Version R2013b*. The MathWorks, Inc., Natick, Massachusetts. URL http://www.mathworks.com/products/matlab/.

Todeschini A, Caron F (2013). "**BiiPS**: Bayesian Inference with Interacting Particle Systems." URL https://alea.bordeaux.inria.fr/biips/.

UniData (2013). "NetCDF." Network Common Data Form, URL http://www.unidata.ucar.edu/software/netcdf/.

Vo BN, Ma WK (2006). "The Gaussian Mixture Probability Hypothesis Density Filter." *IEEE Transactions on Signal Processing*, **54**, 4091–4104. doi:10.1109/tsp.2006.881190.

Wardley A (2013). "The Perl Template Toolkit." URL http://www.template-toolkit.org/.

Westerhof N, Lankhaar JW, Westerhof B (2009). "The Arterial Windkessel." *Medical and Biological Engineering and Computing*, **47**(2), 131–141. doi:10.1007/s11517-008-0359-2.

Whiteley N, Lee A, Heine K (2013). "On the Role of Interaction in Sequential Monte Carlo Algorithms." URL http://arxiv.org/abs/1309.2918.

Wikle C (2003). "Hierarchical Bayesian Models for Predicting the Spread of Ecological Processes." *Ecology*, **84**(6), 1382–1394. doi:10.1890/0012-9658(2003)084[1382:hbmfpt]2.0.co;2.

Affiliation:

Lawrence M. Murray
Department of Statistics
University of Oxford
Oxford, United Kingdom
E-mail: murray@stats.ox.ac.uk

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
October 2015, Volume 67, Issue 10 *Submitted:* 2013-06-16
doi:10.18637/jss.v067.i10 *Accepted:* 2014-11-21