# Scalable SDE Filtering and Inference with Apache Spark

**Harish S. Bhat** HBHAT@UCMERCED.EDU
**R. W. M. A. Madushani** RMADUSHANI@UCMERCED.EDU
**Shagun Rawat** SRAWAT2@UCMERCED.EDU
*Applied Mathematics Unit, School of Natural Sciences, University of California, Merced, 5200 N. Lake Rd., Merced, CA, 95343, USA*

## Abstract

This is the abstract for this article.
**Keywords:** List of keywords

## 1. Introduction

- Overviews: (Sørensen, 2004; Iacus, 2009; Fuchs, 2013)
- Diffusion approx. for chemical kinetics: (Golightly and Wilkinson, 2004, 2005)
- Approximate Bayes: (Picchini, 2014)
- Using path-wise simulation to fill in "missing data": (Fuchs, 2013; Elerian et al., 2001; Roberts and Stramer, 2001)
- Variational Bayes and mean field: (Archambeau et al., 2007b; Vrettas et al., 2015)
- Gaussian process approximations: (Archambeau et al., 2007a; Ruttor et al., 2013)
- Nonlinear Gaussian filtering and adaptive MCMC: (Särkkä et al., 2015)
- Parametric inference with Gibbs sampling: (Donnet and Samson, 2008)

One way to carry out this inference is through numerical maximization of the likelihood function. For the actual SDE, the exact likelihood $p(\mathbf{x}|\theta)$ can only be computed in very special cases, i.e., when we can solve analytically for the transition density of (1a). Therefore, prior work has focused on approximating the exact likelihood, either through analytical methods, numerical methods, or a combination of the two.

For a thorough review of past work on this problem, we refer the reader to (Sørensen, 2004; Iacus, 2009; Fuchs, 2013). Here we focus on past work that is particularly relevant to understand our approach. Consider the transition density $p_{X_{t_{j+1}}}(x_{j+1}|X_{t_j} = x_j, \theta)$ of a process evolving to $X_{t_{j+1}} = x_{j+1}$ starting from $X_{t_j} = x_j$ according to the SDE (1a). Let $p(x,t)$ denote the density function of $X_t$. Then one approach to approximating the transition density is to numerically solve the forward Kolmogorov (or Fokker-Planck) equation with the initial condition $p(x,0) = \delta(x - x_j)$ up to time $T = t_{j+1} - t_j = \Delta t$. Then $p(x_{j+1}, T)$ will be a numerical approximation of the transition density.

Our approach is similar in that we also numerically track the density $p(x,t)$ without sampling. However, instead of numerically solving a partial differential equation, we track the density by applying quadrature to the Chapman-Kolmogorov equation associated with a time-discretization of the SDE (1a). We detail this method in Section 2. In (Bhat and Madushani, 2016), we consider the problem of computing the density function in the case where the drift and diffusion coefficients are known. With regularity assumptions on the drift and diffusion coefficients, and assuming that the temporal and spatial grid spacings approach zero at particular rates, we prove that the computed density converges to the true density of the SDE. The convergence results take into account the effect of truncating infinite integrals/sums to finite domains.

Other methods similar to ours are those of (Pedersen, 1995) and (Santa-Clara, 1997). In these methods, one also starts with the Chapman-Kolmogorov equation for the Euler-Maruyama scheme applied to (1a). However, instead of evaluating the resulting integrals by deterministic quadrature, Pedersen and Santa-Clara evaluate the integrals by Monte Carlo methods. These methods involve generating numerical sample paths of the SDE at times in between the observation times. This approach is problematic unless one generates sample paths conditional on both the initial condition $X_{t_j} = x_j$ and the final condition $X_{t_{j+1}} = x_{j+1}$.

The work of (Aït-Sahalia, 2002) shares our goal of computing an accurate approximation of the exact transition density and resulting likelihood function. Instead of applying quadrature, Aït-Sahalia expands the transition density in a Gram-Charlier series and then computes the expansion coefficients up to a certain order.

While the approach developed in this paper involves maximum likelihood estimation, computation of the likelihood is also required for many Bayesian approaches. Exceptions include likelihood-free approaches such as that of (Picchini, 2014).

In the present paper, we are primarily interested in developing properties of our algorithm, to establish a set of examples in which the algorithm succeeds. We reserve for future work a detailed comparison of our algorithm against existing approaches for inference in stochastic differential equation models.

The paper is structured as follows: in Section 2, we give detailed derivations of temporally and spatially discretized versions of the log likelihood and its gradient. We carry out the derivations for the cases where the data consists of either one or multiple sample paths. After deriving the algorithms, we conduct a series of numerical tests to study its performance when both model and algorithm parameters are varied. The results of these tests are described in Section 4. In Section 5, we discuss the implications of these results and how they will inform our future work on this problem.

Stochastic Differential Equations (SDE) are a widely used powerful mathematical tool to model real-world phenomena. However, parameter inference of SDE models is still a very challenging problem, due to the fact that the likelihood function is generally unknown for the case where time-discrete observations are available (Sørensen, 2004; Iacus, 2009; Fuchs, 2013). Most existing parametric inference methods for discretely observed SDE require inter-observation time to be small, to track the transition density for the discrete time observations. As a way to facilitate approximation of the transition density for parametric inference for large inter-observation times, Bayesian methods are used to simulate missing values of the observations to form a high-frequency data set. In situations where the

likelihood function is either analytically unavailable or computationally prohibitive to evaluate, Bayesian inference of SDE makes use of likelihood-free methods such as Approximate Bayesian Computation (Picchini, 2014), variational methods (Archambeau et al., 2007b; Vrettas et al., 2015), and/or Gaussian processes (Archambeau et al., 2007a; Ruttor et al., 2013).

In our work we have developed a Markov Chain Monte Carlo Method (MCMC) algorithm for Bayesian inference of parameters in SDE. The MCMC algorithm is derived using a Metropolis scheme; our innovation is to evaluate the log likelihood efficiently using density tracking by quadrature (DTQ). The DTQ method applies quadrature to the Chapman-Kolmogorov equation associated with a time-discretization of the original SDE (Bhat and Madushani, 2016). For the case of scalar SDE, the DTQ method's density function converges to the true density of the SDE at a rate that is linear in the time step. The method we have developed is applicable to the case where inter-observation times are large and/or irregular. In this paper, we present a Metropolis algorithm for Bayesian inference of unknown parameters of a 2-D SDE.

## 2. Statistical Method

The fundamental model considered in this paper is

$$dX_t = f(X_t;\theta)dt + g(X_t;\theta)dW_t \tag{1a}$$

$$Y_t = X_t + \epsilon_t \tag{1b}$$

The first part of the above system (1a) is a stochastic differential equation (SDE) driven by Brownian motion $W_t$. The second part (1b) models the observation process $Y_t$ by the addition of noise $\epsilon_t$ to the state process $X_t$. In this work, we assume that $\epsilon_t$ is i.i.d. (independent and identically distributed) Gaussian with mean 0 and variance $\sigma_\epsilon^2$.

### 2.1. Inference Problem

Suppose that we have data of the form $\mathbf{y} = (y_0, \ldots, y_L)$ where $y_j = Y_{t_j}$, the observation at time $t_j$. Here $t_0 < t_1 < \cdots < t_L$ is a sequence of times, not necessarily equispaced, at which we collect observations. Using $\mathbf{y}$, we seek to infer the following objects:

- the discrete-time path taken by the state process, $\mathbf{x} = (x_0, \ldots, x_L)$. Here $x_j = X_{t_j}$, the state of the SDE at time $t_j$.
- the parameter vector $\theta \in \mathbb{R}^N$, and
- the variance $\sigma_\epsilon^2$ of the noise term $\epsilon_t$.

We view this problem as a Bayesian inference problem, and our goal is to sample from the posterior

$$p(\mathbf{x}, \theta, \sigma_\epsilon^2 \,|\, \mathbf{y}) \propto p(\mathbf{y} \,|\, \mathbf{x}, \theta, \sigma_\epsilon^2) p(\mathbf{x}, \theta, \sigma_\epsilon^2) \tag{2}$$

In the above expression, the left-hand side is the conditional density of the random variables $X_{t_0}, X_{t_1}, \ldots, X_{t_L}, \theta, \sigma_\epsilon^2$ given the random variables $Y_{t_0}, Y_{t_1}, \ldots, Y_{t_L}$. To save space and make our equations more readable, we will omit these random variables in what follows.

It is clear from (1b) that $\mathbf{y}$ is conditionally independent of $\theta$ given $\mathbf{x}$ and $\sigma_\epsilon^2$. It is also clear from (1b) that the observation/state pair at time $t_j$ is independent of all other observation/state pairs. Hence we can write

$$p(\mathbf{y} \,|\, \mathbf{x}, \theta, \sigma_\epsilon^2) = \prod_{j=0}^{L} p(y_{t_j} \,|\, x_j, \sigma_\epsilon^2). \tag{3}$$

Next, we examine the second term on the right-hand side of (2). It is clear that $\sigma_\epsilon^2$ is independent of the other random variables, so we can write:

$$p(\mathbf{x}, \theta, \sigma_\epsilon^2) = p(\mathbf{x}, \theta)p(\sigma_\epsilon^2) = p(\mathbf{x}|\theta)p(\theta)p(\sigma_\epsilon^2). \tag{4}$$

Putting it all together, we have the following expression for the posterior:

$$p(\mathbf{x}, \theta, \sigma_\epsilon^2 \,|\, \mathbf{y}) \propto \left[\prod_{j=0}^{L} p(y_{t_j} \,|\, x_j, \sigma_\epsilon^2)\right] p(\mathbf{x}|\theta)p(\theta)p(\sigma_\epsilon^2) \tag{5}$$

From (1b), we have that $y_{t_j} \,|\, x_j, \sigma_\epsilon^2$ is Gaussian with mean $x_j$ and variance $\sigma_\epsilon^2$. The terms $p(\theta)$ and $p(\sigma_\epsilon^2)$ are priors. The only other term, $p(\mathbf{x}|\theta)$, is the likelihood of $\theta$ under the model (1a). We describe the computation of this likelihood next.

### 2.2. Likelihood Computation via Density Tracking by Quadrature

Here we describe how to compute the likelihood $p(\mathbf{x}|\theta)$ under the model (1a). Our first step is to apply a Markov property satisfied by (1a): the random variable $X_{t_{j+1}}$, given $X_{t_j}$, is conditionally independent of all random variables $X_{t_{j-k}}$ for $k \geq 1$. With this property, the likelihood factors:

$$p(\mathbf{x}|\theta) = p(x_0) \prod_{j=0}^{L-1} p(x_{j+1} \,|\, x_j, \theta). \tag{6}$$

Each term in the product can be interpreted as follows: we start the SDE (1a) with the initial condition $X_{t_j} = x_j$ and fixed parameter vector $\theta$. We then solve for the probability density function (pdf) of $X_{t_{j+1}}$, and evaluate that pdf at $x_{j+1}$. By following these steps, we have calculated $p(x_{j+1} \,|\, x_j, \theta)$.

We now outline a provably convergent method to compute the aforementioned pdf. Because this method computes an approximation to the density via iterated quadrature, we refer to the method as DTQ (density tracking by quadrature). The first step of the method consists of discretizing (1a) via the Euler-Maruyama discretization. When describing this discretization, we specialize to the case where we seek $p(x_{j+1} \,|\, x_j, \theta)$. That is, we take $\{\tau_i\}_{i=0}^n$ to be a temporal grid such that $\tau_0 = t_j$, $\tau_n = t_{j+1}$, and $h = (t_{j+1} - t_j)/n > 0$. Then, for $0 = 1, 2, \ldots, n$, we have $\tau_i = t_j + ih$. On this temporal grid, the Euler-Maruyama discretization of (1a) is:

$$\tilde{x}_{i+1} = \tilde{x}_i + f(\tilde{x}_i; \theta)h + g(\tilde{x}_i; \theta)h^{1/2} Z_{i+1} \tag{7}$$

where $\{Z_i\}_{i=1}^n$ is an i.i.d. family of Gaussian random variables, each with mean 0 and variance 1. For $i \geq 1$, we think of $\tilde{x}_i$ as a numerical approximation to $X_{\tau_i}$. Note that $\tilde{x}_0 = X_{\tau_0} = X_{t_j}$ which is constrained to equal the data point $x_j$ in this calculation.

The next step in deriving the DTQ method is to write down the Chapman-Kolmogorov equation corresponding to (7). Let $p(\tilde{x}_i)$ denote the pdf of $\tilde{x}_i$ given the initial condition $\tilde{x}_0 = x_j$. Then purely based on the laws of probability we can write:

$$p(\tilde{x}_{i+1}) = \int_{\tilde{x}_i} p(\tilde{x}_{i+1} \,|\, \tilde{x}_i) p(\tilde{x}_i) \, d\tilde{x}_i. \tag{8}$$

Inspecting (7), we see that conditional on $\tilde{x}_i$, the pdf of $\tilde{x}_{i+1}$ is Gaussian with mean $\tilde{x}_i + f(\tilde{x}_i; \theta)h$ and variance $g^2(\tilde{x}_i; \theta)h$. Let us define the function

$$G(a, b) = \frac{1}{\sqrt{2\pi g^2(b; \theta)h}} \exp\left( -\frac{(a - b - f(b; \theta)h)^2}{2g^2(b; \theta)h} \right). \tag{9}$$

Then (8) becomes

$$p(\tilde{x}_{i+1}) = \int_{\tilde{x}_i} G(x_{i+1}, x_i) p(\tilde{x}_i) \, d\tilde{x}_i. \tag{10}$$

The last step is to spatially discretize the pdf's and the integration over $\tilde{x}_i$. Let $k > 0$ be constant; then $z_j = jk$ with $j \in \mathbb{Z}$ is an equispaced grid with spacing $k$. We represent the function $p(\tilde{x}_i)$ by a vector $\mathbf{p}_i$ such that the $j$-th component of $\mathbf{p}_i$ is $p_i^j = p(\tilde{x}_i = z_j)$. We then apply the trapezoidal rule to (10), resulting in

$$p_{i+1}^{j'} = \sum_j k G(z_{j'}, z_j) p_i^j. \tag{11}$$

We now truncate the spatial domain. Let $M > 0$ be an integer. We take both $j, j' \in \{-M, \ldots, 0, \ldots, M\}$; this means that each vector $\mathbf{p}_i$ has dimension $2M + 1$.

Now that we have a method to compute the required pdf's, here is how we compute $p(x_{j+1} \,|\, x_j, \theta)$:

1. Because $\tilde{x}_0 = x_j$, we have that $p(\tilde{x}_0) = \delta(\tilde{x}_0 - x_j)$. Inserting this in (10) with $i = 0$, we obtain
$$p(\tilde{x}_1) = G(\tilde{x}_1, x_j). \tag{12}$$
Evaluating the right-hand side with $\tilde{x}_1$ equal to each point in the spatial grid $\{z_j\}_{j=-M}^{M}$, we obtain $\mathbf{p}_1$.

2. With $\mathbf{p}_1$ in hand, we iterate (11) a total of $n - 2$ times. This takes us to $\mathbf{p}_{n-1}$.

3. Then, by (10), we have that the density of $\tilde{x}_{i+1}$ evaluated at the data point $x_{j+1}$ is
$$p(\tilde{x}_{i+1})\Big|_{\tilde{x}_{i+1} = x_{j+1}} = \int_{\tilde{x}_i} G(x_{j+1}, \tilde{x}_i) p(\tilde{x}_i) \, d\tilde{x}_i.$$
Applying trapezoidal quadrature to the right-hand side, we have
$$p(x_{j+1} \,|\, x_j, \theta) \approx \sum_j k G(x_{j+1}, z_j) p_{n-1}^j. \tag{13}$$

### 2.3. Metropolis Algorithm

Now that we understand how to compute the likelihood term, we return to the problem of sampling from the posterior (5). We describe here a classical Metropolis algorithm that incorporates the DTQ method described above for computing the likelihood. The fundamental idea here is to construct a discrete-time, continuous-space Markov chain whose equilibrium density is the posterior density (5). We then compute a sample path of this Markov chain beginning at a particular initial condition in parameter space. The sample path consists of a sequence of iterates; let $\mathbf{x}_i$, $\theta_i$, and $(\sigma_\epsilon^2)_i$ denote the $i$-th iterates of the respective parameters.

We now describe how to proceed from the $i$-th to the $(i+1)$-st iterate of the Markov chain. To compute proposed iterates, we require access to random vectors/variables $Z_{\mathbf{x}} \in \mathbb{R}^{L+1}$, $Z_\theta \in \mathbb{R}^N$, and $Z_\sigma \in \mathbb{R}$. Then the Metropolis algorithm is as follows:

- Generate proposals by combining the old iterates with samples from the $Z$ random variables:

$$\begin{aligned} \mathbf{x}_{i+1}^* &= \mathbf{x}_i + Z_{\mathbf{x}} \\ \theta_{i+1}^* &= \theta_i + Z_\theta \\ \log(\sigma_\epsilon^2)_{i+1}^* &= \log(\sigma_\epsilon^2)_i + Z_\sigma \end{aligned}$$

  The log transformation ensures that the variance parameter $\sigma_\epsilon^2$ only takes positive values.

- Calculate the ratio

$$\rho = \frac{p(\mathbf{x}_{i+1}^*, \theta_{i+1}^*, (\sigma_\epsilon^2)_{i+1}^* \,|\, \mathbf{y})}{p(\mathbf{x}_i, \theta_i, (\sigma_\epsilon^2)_i \,|\, \mathbf{y})}. \tag{14}$$

  In practice, the denominator of this ratio has already been calculated at the previous step; only the numerator needs to be calculated. To compute the numerator, we use (5) together with the procedure described in Section 2.2, including (12), (10), and (13).

- Let $u$ be a sample from a uniform random variable on $[0, 1]$. If $\rho > u$, we accept the proposal, setting $\mathbf{x}_{i+1} = \mathbf{x}_{i+1}^*$, $\theta_{i+1} = \theta_{i+1}^*$, and $(\sigma_\epsilon^2)_{i+1} = (\sigma_\epsilon^2)_{i+1}^*$. If $\rho \leq u$, we reject the proposal, setting $\mathbf{x}_{i+1} = \mathbf{x}_i$, $\theta_{i+1} = \theta_i$, $(\sigma_\epsilon^2)_{i+1} = (\sigma_\epsilon^2)_i$.

## 3. Scalable Implementation

There are two elements to our strategy of implementing the MCMC algorithm from Section 2 in a scalable fashion. The first aspect has to do with representing the main DTQ step (11) using Scala and Breeze. The second aspect has to do with using Apache Spark to evaluate each term in the product (6) in a parallel, distributed fashion. Note that all of our codes and data are available at https://github.com/hbhat4000/sdeinference/tree/master/sparkdtq. The main code to perform MCMC inference is `sparkdtq.sc`. Also note that all development and testing was carried out on a server with 24 effective cores (2 Intel Xeon E5-2620 chips at 2.0 GHz), 16 GB of RAM, and 4 TB of disk space.

### 3.1. Scala/Breeze

A typical approach in computational statistics to implement (11) would be to view the equation as matrix multiplication. Indeed, it is conceptually simple to view $kG(z_{j'}, z_j)$ as the $(j', j)$ element of a $(2M+1) \times (2M+1)$ matrix $\mathcal{G}$, in which case (11) can be written as

$$\mathbf{p}_{i+1} = \mathcal{G}\mathbf{p}_i. \tag{15}$$

Multiplication by $\mathcal{G}$ steps the density forward by $h$ units of time; for this reason, we refer to $\mathcal{G}$ as the propagator.

The above approach, while mathematically correct, does not recognize the sparsity of $\mathcal{G}$. In fact, we have an accurate estimate of where the nonzero entries of $\mathcal{G}$ are located: near the diagonal. From (9), we see that the argument to the exponential is zero when $a = b + f(b;\theta)h$. If we suppose that $f$ is smooth, then the mean-value theorem implies that there exists $\xi$ such that $b = (a - f(0)h)/(1 + f'(\xi)h)$. If $f$ has bounded derivative (in this context, equivalent to assuming $f$ is Lipschitz), then this implies that $b = a + O(h)$. The upshot is that for fixed $a$, when $b$ is near $a$, the exponential term in (9) will be maximal. Similarly, we can conclude that for fixed $a$, when $b$ is far from $a$, the exponential term in (9) will be negligible. We have focused on the exponential term under the assumption that the diffusion coefficient $g$, and hence the normalization term in (9), does not itself grow exponentially. In practice, this and other assumptions made above are quite reasonable.

Because of the decay of the Gaussian, for each fixed $j'$, the $(j', j)$ element of $\mathcal{G}$ need not be computed for all $j$. We fix a window size $\gamma > 0$ and then only compute $\mathcal{G}_{(j',j)}$ for those $j$ that satisfy both $-M \le j \le M$ and $j' - \gamma \le j \le j' + \gamma$. We choose $\gamma$ large enough such that each density $\mathbf{p}_{i+1}$ is correctly normalized, i.e., such that $k\sum_j p_i^j$ is sufficiently close to 1. In all of our tests, we have been able to choose $\gamma \ll M$ while maintaining normalization to machine precision.

The main DTQ step (11), as it is implemented in Scala using the window parameter $\gamma$, is represented graphically in Figure 1. The code implementing this step makes use of the Breeze library (`https://github.com/scalanlp/breeze`) for numerical linear algebra in Scala. For additional efficiency, we utilize Breeze's support for the Intel Math Kernel Library (MKL).

Let us explain the procedure diagrammed in Figure 1. In the Metropolis algorithm given in Section 2.3, each time we must evaluate the numerator of $\rho$ in (14), we must evaluate the likelihood function for a particular choice of $\mathbf{x}$ and $\theta$. For this choice of $\theta$, we evaluate each row of the propagator matrix $\mathcal{G}$ over a window of size $2\gamma+1$. These rows are represented by the pink rectangles; there are $2M+1$ such rows. Because each such row has the same size, we store the resulting collection as a Breeze `DenseVector` of `DenseVector`. This computation, which comprises the upper half of Figure 1, occurs once per Metropolis step.

To implement (15), we must now multiply the propagator matrix by the vector representing the pdf at time step $i$. This matrix-vector product can be equivalently described as a collection of $2M+1$ vector-vector dot products, where each vector is of size $2\gamma+1$. To generate the vectors to dot against collection of propagator vectors (already computed), we apply a windowing technique. Namely, for each element $p_i^j$ of the pdf vector $\mathbf{p}_i$, we construct a window of size $2\gamma+1$ around the element $p_i^j$: the window consists of $(p_i^{j-\gamma}, \ldots, p_i^j, \ldots, p_i^{j+\gamma})$. Of course, it is understood here that $p_i^c = 0$ whenever $|c| > M$. In this way, we build a

collection of windowed pdf vectors, represented in Figure 1 as the lower-right stack of blue rectangles. As above, the collection consists of $2M + 1$ vectors, each of size $2\gamma + 1$.

With the propagator collection denoted by `propagator` and the collection of windowed pdf vectors denoted by `allwins`, the Scala and Breeze syntax for computing all required dot products at once is, simply

```
px = propagator dot allwins
```
(16)

This line of code completes the implementation of (15). To iterate the procedure, we apply the windowing procedure—diagrammed in the lower half of Figure 1—to `px`, store the resulting collection in `allwins`, and then repeat (16).

The entire procedure diagrammed in Figure 1 makes use of functional programming techniques—specifically, Scala's `map` construct—to entirely avoid explicit loops. Additionally, note that this procedure is inherently more efficient than using a sparse matrix representation for the propagator matrix; we know where the nonzero entries belong, so we do not need to allocate either space or time to this task.

### 3.2. Spark

Spark enables parallel/distributed computation using the notion of a resilient distributed dataset (RDD). Since the main bottleneck in our Metropolis algorithm is the computation of the likelihood $p(\mathbf{x} \,|\, \theta)$, we turn to the question of converting the state time series $\mathbf{x}$ into an RDD. We think of this time series as a sequence of pairs $(t_j, x_j)$ as depicted in the top line of Figure 2. When we examine (6), we see that to compute a given term in the product, we need access to neighboring pairs $(t_j, x_j)$ and $(t_{j+1}, x_{j+1})$.

Hence, we map the original sequence of pairs, labeled as $\overrightarrow{tx}$ in Figure 2, to `tslices`, a Scala `Array` where each element is a vector of neighboring pairs. We convert this array to an RDD, `tslicesRDD`, using Spark's `sc.parallelize` method. When we subsequently use a `map` operation to compute the log likelihood $\log p(x_{j+1} \,|\, x_j, \theta)$ term corresponding to each element of `tslicesRDD`, the computation takes place in parallel. Spark automatically distributes the propagator and the $\theta$ vector. For a non-equispaced time series problem, each calculation of $p(x_{j+1} \,|\, x_j, \theta)$ will take more (respectively, less) time when $t_{j+1} - t_j$ is larger (respectively, smaller). Again, Spark automatically assigns tasks to workers to compute the overall log likelihood efficiently.

## 4. Results

We begin with results on artificial data sets. The model used to generate these data sets is the Ornstein-Uhlenbeck SDE

$$dX_t = \theta_1(\theta_2 - X_t)dt + \theta_3 dW_t \tag{17}$$

together with the observation process

$$Y_t = X_t + \epsilon_t. \tag{18}$$

Specifically, what we do is apply the Euler-Maruyama discretization to (17) with a time step of $\kappa = 10^{-6}$. Suppose our temporal grid consists of $t_j = n(j)\kappa$, where $n(0) = 0$, and

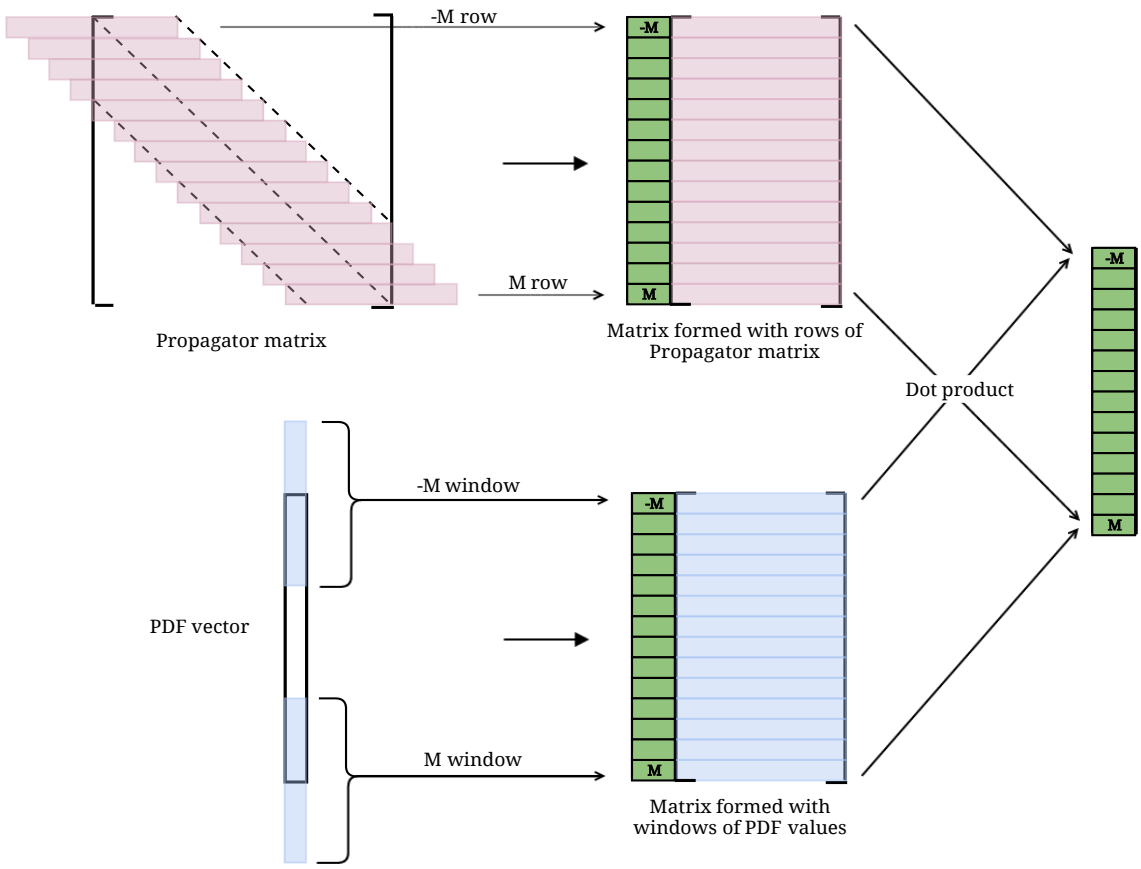


Figure 1: In order to implement the matrix-vector multiplication in (15) in a scalable way, we make use of the structure of the propagator matrix $\mathcal{G}$. Instead of computing all entries of this matrix, we compute and store only those entries that are close to the diagonal—the pink rectangles in the upper half of the diagram. The blue rectangles in the lower half of the diagram correspond to windowed versions of the pdf vector $\mathbf{p}_i$. In both cases, there is one windowed vector per row; the row numbers go from $-M$ to $M$ as labeled. Both the pink and blue rectangles correspond to vectors of length $2\gamma+1$, with $\gamma \ll M$. The matrix-vector multiplication $\mathcal{G}\mathbf{p}_i$ then corresponds to a collection of $2M+1$ vector-vector dot products. This representation of (15) makes efficient use of Scala, Breeze, and the Intel MKL. For more details, see the description in Section 3.1.

$n(j+1) > n(j)$. In some of the examples we pursue, $n(j)$ will be deterministic and the temporal grid will be equispaced, while in other examples, $n(j)$ will be stochastic and the temporal grid will be non-equispaced. Either way, we take $n(j)$ to have expected value $2 \times 10^5$, so that the average difference between temporal grid points $t_{j+1} - t_j$ is 0.2.

We then start at a random initial condition by sampling $X_0$ from a Gaussian random variable with mean 0 and variance 1. We step forward one step at a time (with time step

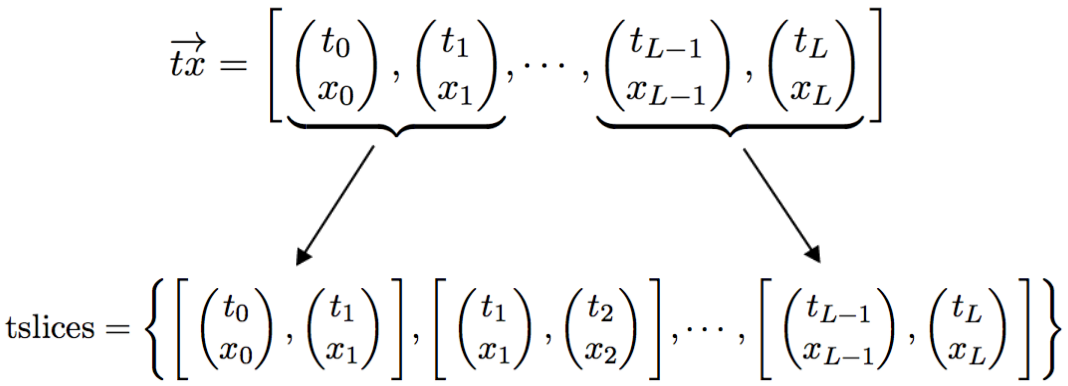


Figure 2: We use Spark to parallelize the computation of the likelihood (6). We accomplish this by converting the original time series (for states $\mathbf{x}$, not observations $\mathbf{y}$) from a vector of pairs to an array where each element is a vector of consecutive pairs. The original vector of pairs is labeled as $\overrightarrow{tx}$, and the Scala `Array` of consecutive pairs is `tslices`. This latter object can be easily converted into a Spark RDD; subsequent `map` operations on this RDD are executed in parallel.

$\kappa$), saving the numerical solution $X_t$ at points in time corresponding to the temporal grid points $\{t_j\}_{j=0}^{L}$. We label the points we save as $(x_0, x_1, \ldots, x_L) =: \mathbf{x}$, and then perturb them via (18) to generate $\mathbf{y}$. In particular, we set $y_j = x_j + Z$ where $Z$ is normally distributed with mean 0 and variance $\sigma^2$.

The DTQ method described in Section 2.2 has four internal parameters: the time step $h$, the spatial grid spacing $k$, the spatial grid cutoff $M$, and the decay width $\gamma$ of the Gaussian kernel $G$. For the tests described below, we will give the value of $h$ that was used. All other parameters are as follows: $k = h^{0.75}$, $M = \lceil \pi/k^{1.5} \rceil$, and $\gamma = 25$.

### 4.1. Equispaced Time Series

In the first set of experiments, we follow the procedure outlined above to generate artificial data $\mathbf{y}$ with the temporal grid defined by $t_j = n(j)\kappa = (0.2)j$. The ground truth for the parameters consists of $\theta = (0.5, 1, 0.25)$ and $\sigma^2 = 0.01$. In addition to the other parameters, we focus on inferring $\theta_1$ and $\theta_2$; we fix $\theta_3 = 0.25$ throughout. In the Metropolis algorithm, we take as initial conditions $\mathbf{x}_0 = \mathbf{y}$, $\theta = (1, 0.1, 0.25)$, and $\sigma_\epsilon^2 = 1$.

For priors for $\theta_1$ and $\theta_2$, we use Gaussian densities with respective parameters $(\mu = 0.5, \sigma = 1)$ and $(\mu = 2, \sigma = 10)$. For $\sigma_\epsilon^2$, we use an exponential prior with parameter $\lambda = 1$.

In the Metropolis algorithm, we take all proposal random variables to be independent Gaussians. In particular, $Z_{\mathbf{x}}$ is a collection of $L + 1$ independent Gaussians, each with parameters $(\mu = 0, \sigma = 0.02)$, $Z_\theta$ consists of two independent Gaussians, each with parameters $(\mu = 0, \sigma = 0.05)$, and $Z_\sigma$ consists of a Gaussian with parameters $(\mu = 0, \sigma = 0.02)$.

For two values of the DTQ internal time step ($h = 0.02$ and $h = 0.01$), we apply the Metropolis algorithm to generate $10,000$ samples of the posterior (5). Note that $h = 0.02$ implies $M = 257$ and $h = 0.01$ implies $M = 559$; hence $\gamma = 25$ gives at least a 10-fold reduction in computational effort.

We discard the first 100 samples as burn-in samples, i.e., samples that are taken before the Metropolis Markov chain has, to a good approximation, reached equilibrium.

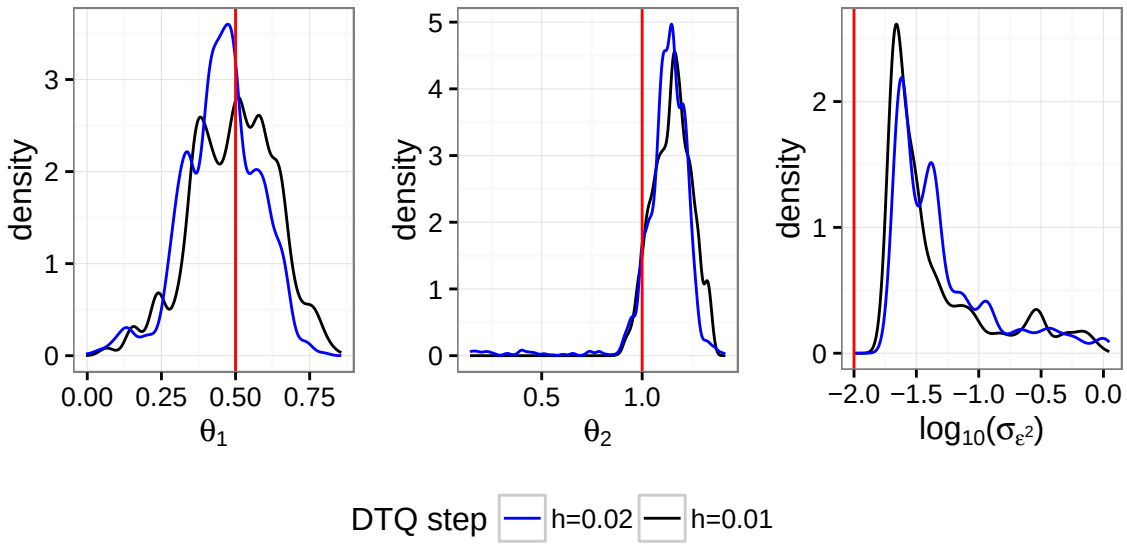


Figure 3: Posterior densities for equispaced time series

### 4.2. Non-equispaced Time Series

In this next set of experiments, we follow a nearly identical procedure to that described in Section 4.1. The main difference is that the temporal grid is defined by $t_j = n(j)\kappa$ where $n(j)$ is uniformly distributed on the integers between $4 \times 10^4$ and $4 \times 10^5 - 4 \times 10^4$. Effectively, this generates a time series with minimum, mean, and maximum spacings $t_{j+1} - t_j$ of, respectively, 0.04, 0.2, and 0.36.

## 5. Conclusion

## Acknowledgments

This work was partially supported by a grant from UC Merced's Committee on Research (awarded to H.S. Bhat) and by UC Merced USAP fellowships (awarded to R.W.M.A. Madushani and S. Rawat).

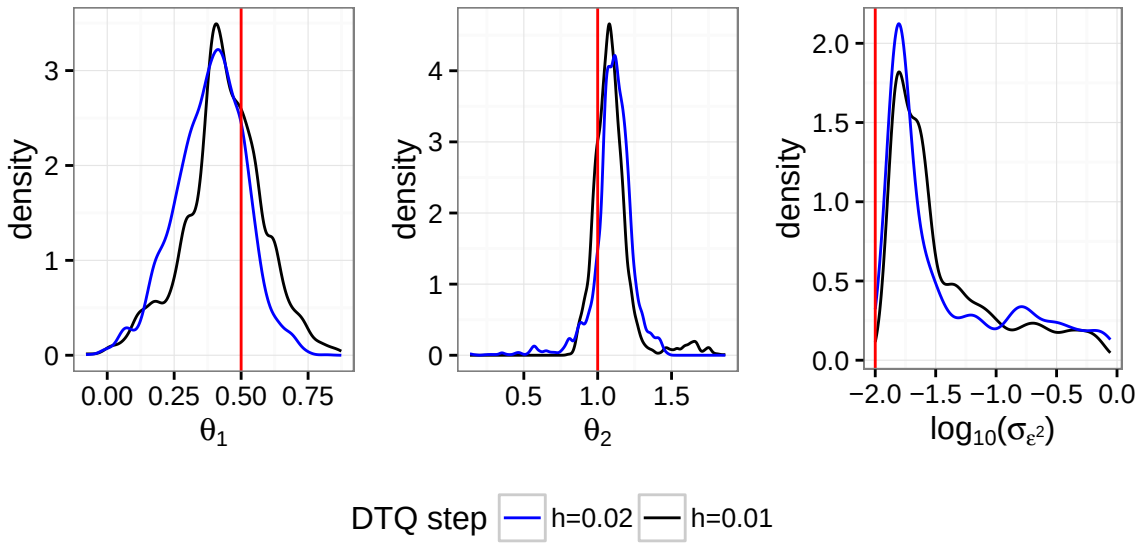


Figure 4: Posterior densities for non-equispaced time series

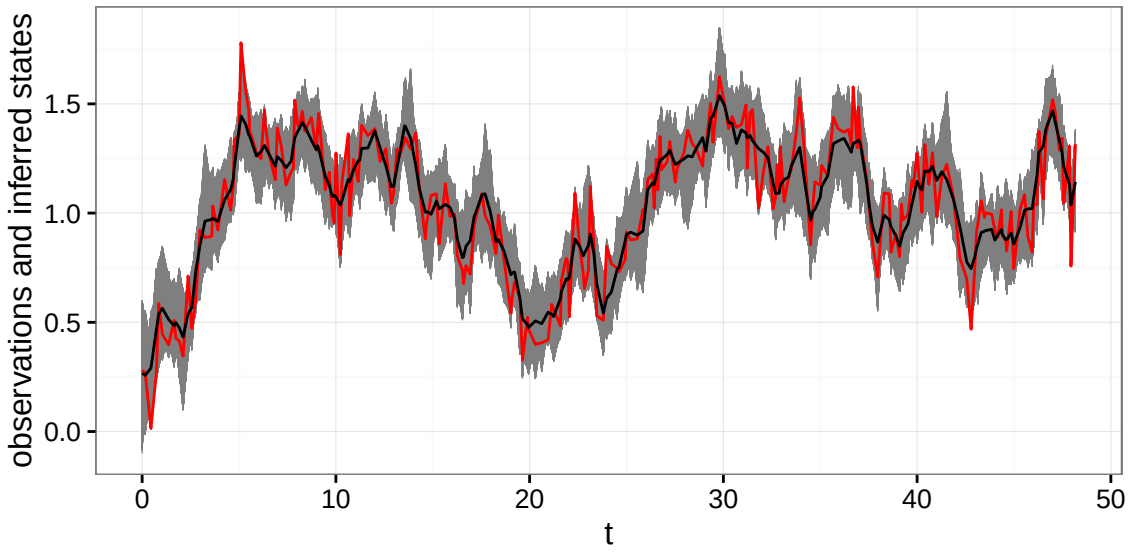


Figure 5: Observations and inferred states

## References

Yacine Aït-Sahalia. Maximum likelihood estimation of discretely sampled diffusions: A closed-form approximation approach. *Econometrica*, 70(1):223–262, 2002.

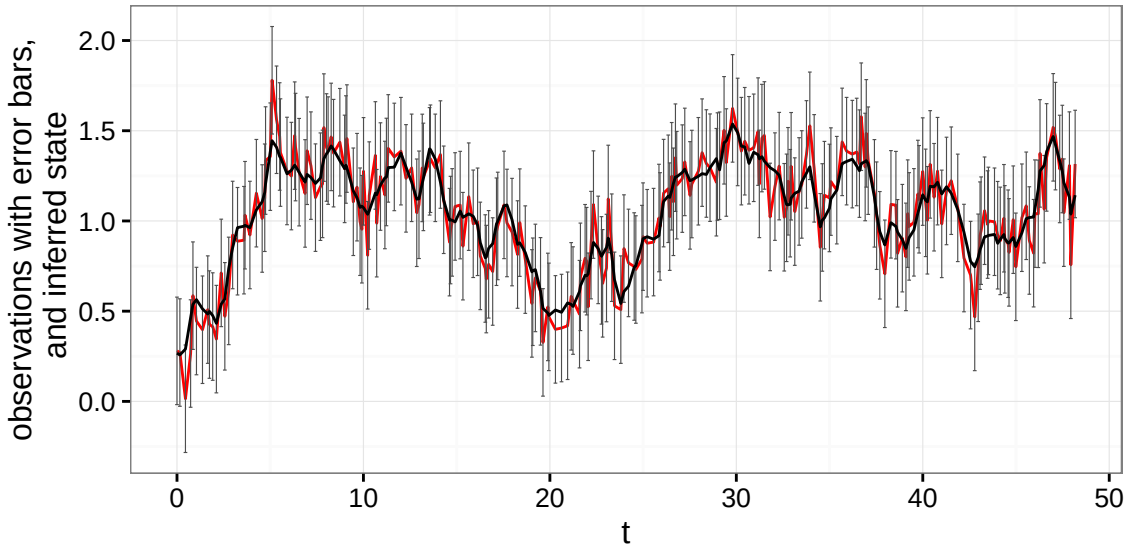


Figure 6: Observations and mean inferred state with error bars. The error bars are computed by adding/subtracting the mean inferred value of $\sigma_\epsilon$ to the observation series $\mathbf{y}$.

C. Archambeau, D. Cornford, M. Opper, and J. Shawe-Taylor. Gaussian process approximations of stochastic differential equations. *Journal of Machine Learning Research: Workshop and Conference Proceedings*, 1:1–16, 2007a.

C. Archambeau, M. Opper, Y. Shen, D. Cornford, and J. Shawe-Taylor. Variational inference for diffusion processes. In *Advances in Neural Information Processing Systems 20*, pages 17–24, 2007b.

Harish S. Bhat and R. W. M. A. Madushani. Density tracking by quadrature for stochastic differential equations. Preprint, 2016.

S. Donnet and A. Samson. Parametric inference for mixed models defined by stochastic differential equations. *ESAIM: Probability and Statistics*, 12:196–218, 2008.

O. Elerian, S. Chib, and N. Shephard. Likelihood inference for discretely observed nonlinear diffusions. *Econometrica*, 69(4):959–993, 2001.

C. Fuchs. *Inference for Diffusion Processes: With Applications in Life Sciences*. Springer, Berlin, 2013.

A. Golightly and D. J. Wilkinson. On Bayesian inference for stochastic kinetic models using diffusion approximations. Statistics Preprint STA04, 4, University of Newcastle, U.K., 2004.

A. Golightly and D. J. Wilkinson. Bayesian inference for stochastic kinetic models using a diffusion approximation. *Biometrics*, 61:781–788, 2005.

S. M. Iacus. *Simulation and Inference for Stochastic Differential Equations: With R Examples*. Springer Series in Statistics. Springer, New York, 2009.

Asger Roer Pedersen. A new approach to maximum likelihood estimation for stochastic differential equations based on discrete observations. *Scandinavian Journal of Statistics*, 22(1):55–71, 1995.

U. Picchini. Inference for SDE models via approximate Bayesian computation. *Journal of Computational and Graphical Statistics*, 23(4):1080–1100, 2014.

G. O. Roberts and O. Stramer. On inference for partially observed nonlinear diffusion models using the Metropolis-Hastings algorithm. *Biometrika*, 88(3):603–621, 2001.

A. Ruttor, P. Batz, and M. Opper. Approximate Gaussian process inference for the drift function in stochastic differential equations. In *Advances in Neural Information Processing Systems 26*, pages 2040–2048, 2013.

P. Santa-Clara. Simulated likelihood estimation of diffusions with an application to the short term interest rate. Working Paper 12-97, UCLA Anderson School of Management, 1997.

S. Särkkä, J. Hartikainen, I. S. Mbalawata, and H. Haario. Posterior inference on parameters of stochastic differential equations via non-linear Gaussian filtering and adaptive MCMC. *Statistics and Computing*, 25:427–437, 2015.

H. Sørensen. Parametric inference for diffusion processes observed at discrete points in time: a survey. *International Statistical Review*, 72(3):337–354, 2004.

M. D. Vrettas, M. Opper, and D. Cornford. Variational mean-field algorithm for efficient inference in large systems of stochastic differential equations. *Physical Review E*, 91 (012148), 2015.