

UC Merced

UC Merced Electronic Theses and Dissertations

Title

Learning governing equations for stochastic dynamical systems

Permalink

<https://escholarship.org/uc/item/4zj533b9>

Author

Rawat, Shagun

Publication Date

2018

Copyright Information

This work is made available under the terms of a Creative Commons Attribution License, available at <https://creativecommons.org/licenses/by/4.0/>

Peer reviewed|Thesis/dissertation

Learning Governing Equations for Stochastic Dynamical Systems

by

SHAGUN RAWAT

A dissertation submitted to the
UNIVERSITY OF CALIFORNIA, MERCED
in partial fulfillment of the requirements for
the degree of DOCTOR OF PHILOSOPHY.



UNIVERSITY OF CALIFORNIA, MERCED

2018

UNIVERSITY OF CALIFORNIA, MERCED
Graduate Division

This is to certify that I have examined a copy of a dissertation by

Shagun Rawat

and found it satisfactory in all respects, and that any and all revisions
required by the examining committee have been made.

Faculty Advisor:

Professor Harish S. Bhat

Committee Members:

Professor Noémi Petra

Professor Roummel F. Marcia

Applied Mathematics Graduate Chair:

Professor Harish S. Bhat

Date

ABSTRACT OF THE DISSERTATION

Learning Governing Equations for Stochastic Dynamical Systems

by

Shagun Rawat

University of California, Merced, 2018

Professor Harish S. Bhat, Chair

In this dissertation, we present our work on automating discovery of governing equations for stochastic dynamical systems from noisy, vector-valued time series. By discovery, we mean learning both a drift vector field and a diffusion matrix for an Itô stochastic differential equation (SDE) in \mathbb{R}^d . In particular, we develop, test, and compare numerical methods for the computation of likelihoods for SDE models. We focus on likelihood computation as it is intractable with no closed form solution in most cases. Thus it forms the bottleneck for both the frequentist and Bayesian methods for inference of stochastic systems.

In the first part of the dissertation, we develop an iterative algorithm using expectation maximization (EM) combined with data augmentation using diffusion bridge sampling. We focus on nonparametric models for high-dimensional SDEs in the low-data, high-noise regime. To our knowledge, this is the most general EM approach to learning an SDE with multidimensional drift vector field and diffusion matrix. Data augmentation has a two-fold advantage; the expectation of log likelihood in the E step reduces to summation and the optimization in the M step reduces to a batch-wise least-squares problem.

In the second part of the dissertation, we consider the problem of Bayesian filtering and inference for lower-dimensional parametric SDE models in the low-data, high-noise regime. Our goal is to be able to infer the model parameters (the inference problem) and the true states of the processes (the filtering problem). We develop a numerical approximation for the likelihood of the SDE using an innovative density tracking by quadrature (DTQ) method. The posterior can be deterministically tracked, as it evolves between each time interval, through a temporal and spatial grid. We focus on generating accurate estimates of the likelihood function, which allows accurate maximum likelihood estimation (MLE) and maximum a posteriori (MAP) estimates, and for vanilla Monte Carlo samplers to explore the distribution.

Learning Governing Equations for Stochastic Dynamical Systems

Copyright 2018
by
Shagun Rawat

ACKNOWLEDGMENTS

First and foremost, I would like to thank my advisor, Prof. Harish Bhat, without whose guidance this work would not have been possible. He has been patient while explaining new concepts and diligent in helping me master them. His encouragement helped me be unafraid to explore new research avenues and take it in exciting directions. He taught me far more than technical knowledge and I am truly grateful for the experience.

I would also like to thank my committee members, Prof. Roummel Marcia and Prof. Noémi Petra for their guidance and support through the process, as well as the time and effort they spent to serve as my committee members.

I would like to thank the Applied Mathematics Graduate Program for the Summer 2016 USAP fellowship, Summer 2016 travel fellowship and the Spring 2018 GSR support. I would also like to acknowledge the NSF Grant (*DMS-1723272*) which helped partially fund my research for Spring 2018. I gratefully acknowledge computational time on the MERCED cluster; MERCED is supported by the National Science Foundation (Grant No. *ACI-1429783*).

I had the opportunity to learn from and work with some incredible professors. I am grateful to Prof. Boaz Ilan, Prof. Mayya Tokman, Prof. Juan Meza, Prof. Shilpa Khatri and Prof. Miguel Carreira-Perpiñán. I am especially thankful to Prof. Tokman for taking time out after lectures to help me navigate graduate school when I was at my lowest. The combined effort of these teachers provided me with a strong base to build my research on.

I greatly cherish the time I spent with some amazing friends and peers at Merced. Christine Hoffman, Som Sarang, Mathew Jian, Rafay Qureshi, Li-Hsuan Huang, and the many more friends in Applied Mathematics group and the UC Merced research community at large gave me a strong sense of community and helped me thrive even during the testing times of my graduate life. I consider myself lucky to have forged friendships that endured throughout my PhD journey and will so beyond.

Now to the family that I left back home! My brother, Bhagawat, has been a guiding light for my academic journey, sharing his experiences, from which I had plenty to learn. I can not thank my parents – Geeta and Bharat – enough for their hard work and dedication towards my education. I will never be able to fully comprehend the struggles you faced to raise me, but I do hope that I made you proud!

Finally, I owe special gratitude to Suril Shah for taking on so many roles in my life; for being my partner, best friend, critique, motivator, mentor, and for his unwavering support in everything I do. No matter how challenging the journey has been, no day was a bad day, because of you. And to the lovely Shah family – Mr. Rakesh, Mrs. Sangita and Rahil – you have added so much richness to my life. Thank you for being more supportive, loving and hilarious than what I could have ever hoped for!

Contents

Abstract	iii
Acknowledgment	v
Contents	vii
List of Figures	ix
List of Tables	xv
1 Introduction	1
1.1 Bridge EM Introduction	3
1.1.1 Summary	4
1.1.2 Related Work	4
1.2 DTQ Introduction	5
1.2.1 Summary	5
1.2.2 Related Work	6
1.3 Outline	6
2 EM via Diffusion Bridge	9
2.1 Problem Setup	9
2.1.1 Parameterization	9
2.1.2 Data	10
2.1.3 Diffusion Bridge	11
2.1.4 Expectation Maximization (EM)	13
2.2 Experiments	14
2.2.1 Experiment 1: Varying Number of Time Series	18
2.2.2 Experiment 2: Varying Length of Time Series	21
2.2.3 Experiment 3: Varying Noise Strength	24
2.2.4 Experiment 4: Varying Data Augmentation	26
2.3 Discussion	28
3 Density Tracking by Quadrature	29
3.1 Statistical Model	29
3.2 DTQ method	30
3.2.1 Likelihood Computation	32
3.2.2 Gradient Computation	34

3.3	Two-dimensional Coupled SDE	35
3.4	Results	38
3.4.1	Linear SDE (Ornstein-Uhlenbeck process)	38
3.4.2	Nonlinear SDE (Double Well Potential)	40
3.4.3	Generic Polynomial Drift and Diffusion Functions	42
3.4.4	Coupled SDEs	43
3.5	Discussion	52
4	SDE Filtering	53
4.1	Introduction	53
4.2	Statistical Method	55
4.2.1	Inference Problem	55
4.2.2	Metropolis Algorithm	56
4.3	Scalable Implementation	57
4.3.1	Scala/Breeze	57
4.3.2	Spark	59
4.4	Results	60
4.4.1	Equispaced Time Series	61
4.4.2	Non-equispaced Time Series	62
4.4.3	Scaling	64
4.5	Discussion	65
	References	67
	Appendices	73
A	Derivation of DTQ method	75
A.1	Likelihood Computation	75
A.2	Gradient Computation	77
A.3	Multiple sample paths	80
A.3.1	Likelihood Computation	80
A.3.2	Gradient Computation	81
A.4	Adjoint based Gradient Computation	81

List of Figures

2.1	Illustration of 1000 Brownian bridge sample paths, post burnin, in 1D (left) and 2D (right) from a fixed initial to final point over a time interval. The blue dots represent the observed data. The grey curves represent the Brownian bridge sample paths between the two observed points and the red line represents the mean of the sample paths.	11
2.2	On the left, we plot 10 time series observed 21 times over the time interval $[0,10]$. On the right, we plot 100 Brownian bridge paths between each observed data point for the solid blue series in the left plot. The blue dots represent the observed data. The grey curves represent the Brownian bridge sample paths between the initial and final observed data point for each interval. The red line represents the mean of the Brownian bridge path samples.	12
2.3	From left to right, top to bottom, we plot 10 sample paths for the 1D, 2D, 3D and 4D systems where the initial condition is normally distributed $\mathcal{N}(0,2)$. The initial time $t = 0$ and final time $t = 10$, with 100 time points observed over the interval. The plots are of the observations of the x components versus time.	17
2.4	As we increase the number S of time series used to learn the drift, the estimated drift more closely approximates the ground truth. From top to bottom, left to right, we have plotted estimated and true drifts for the 1D, 2D, 3D and 4D systems. For the 4D system, we have plotted the X_1, X_2 and X_3 components, keeping the X_0 component constant.	18
2.5	As we increase the number S of time series used to learn the drift, the Frobenius norm error between estimated and true drifts—see (2.15)—decreases significantly. In the first column, from top to bottom, we have plotted results for the 1D, 2D, 3D and 4D systems, with varying threshold from 0.5 to 0.01. The recall and F1 score, on the other hand, increases significantly as we increase the number S of time series. The effect of thresholding is more evident for classification error metrics as a threshold of 0.05 shows much better results for recall compared to the no thresholding case. In the second and third column, from top to bottom, we have plotted recall and F1 score for the 1D, 2D, 3D and 4D systems, respectively.	19

2.6	As we increase the length L of each time series used for learning, the L^2 regression error between estimated and true drifts—see (2.15)—decreases significantly. Recall and F1 score increase with increasing length, L , of time series. As in Experiment 1, applying a hard threshold of $\lambda = 0.05$ helps in improving the classification errors without increasing the regression errors significantly. From left to right, top to bottom, we have plotted the L^2 error, recall and F1 score for 1D, 2D, 3D and 4D systems, respectively.	21
2.7	We plot true and estimated drifts for the 1D, 2D, 3D and 4D systems as a function of increasing time series length L . The last three components of the vector field for the 4D system are plotted as in Figure 2.4. The results show that randomization of observation times compensates for a small value of L , enabling accurate estimation.	23
2.8	We plot true and estimated drifts for the 1D, 2D, 3D and 4D systems as a function of varying noise strength, γ . As in Figure 2.7, the last three components of the vector field for the 4D system are plotted. The results show that estimation is accurate even for large noise strength values and the error goes to zero with the noise strength.	23
2.9	Varying the noise strength in simulated data alters the quality of estimated drift coefficients, quantified using the L^2 error (2.15), recall (2.18) and F1 score (2.19).	24
2.10	As we increase the length F of the diffusion bridge interleaving observed data points, the quality of estimated drifts improves considerably. From left to right, top to bottom, we have plotted the Frobenius errors (2.15), recall (2.18) and F1 score (2.19) between true and estimated coefficients, for the 1D, 2D, 3D and 4D systems.	26
2.11	As in the previous experiments, the estimated drift functions lie close to the true drift function. We find that introducing diffusion bridge samples helps estimating the drift function more accurately.	27
3.1	Illustration of a sample path with discrete model observations x_0, \dots, x_M at times t_0, \dots, t_M in red. Our goal is to infer the parameters of the model using these observations.	29
3.2	Illustration of the temporal grid between a time interval (t_m, t_{m+1}) . We take $\{\tau_i\}_{i=0}^n$ to be the temporal grid such that $\tau_0 = t_m$, $\tau_n = t_{m+1}$, and $h = (t_{m+1} - t_m)/n > 0$. The observed value is constrained by the data point, thus $x_m = X_{\tau_0} = X_{t_m}$	30
3.3	Illustration of the truncated spatial grid, $\{z_j\}_{j=-L}^L$, for each discretized time point in the interval (t_m, t_{m+1}) . The estimated probability at any given point, $p(\tilde{x}_i)$, is represented by a vector, \mathbf{q}_i such that the j -th component of \mathbf{q}_i is $q_i^j = p(\tilde{x}_i = z_j)$	31
3.4	Illustration of the first step of DTQ, propagation of probability from a Dirac-delta function at τ_0 to a vector over the spatial grid at τ_1	32
3.5	Illustration of the forward propagation of probabilities as matrix-vector multiplications, $\mathbf{q}_{n-1} = \mathbf{A}^{n-2} \mathbf{q}_1$	33

3.6	Illustration of the last step of the DTQ algorithm over one time interval. Since the data point at final time τ_n is an observed value, this is a vector to Dirac-delta conversion.	33
3.7	Diagram illustrating motion of runner and chaser. At any instant of time, the chaser's velocity vector points toward the runner's current position. . . .	36
3.8	Illustration of sample paths generated for Ornstein-Uhlenbeck process for the same initial conditions. The plot on the left is the deterministic (ODE) version with no additive noise. The plot on the right is the stochastic (SDE) version with noise.	39
3.9	Illustration of sample paths generated for the double well potential process with same initial conditions. The plot on the left is the deterministic (ODE) version with no additive noise. The plot on the right is the stochastic (SDE) version with noise.	40
3.10	Illustration of the sample path generated by an LC circuit. The dependent variables $X_{1,t}$ and $X_{2,t}$ represent, respectively, the current and voltage of the circuit at time t	43
3.11	We plot kernel density estimates of posterior densities $p(\theta_1^2 \mathbf{x})$. We use simulated data with $\Delta t = 0.04$, generated as described above. Each posterior density corresponds to a finer DTQ step h . As we take a finer DTQ step (i.e., as h decreases), the posterior mode approaches the true value indicated by the solid vertical line at $1/L = 2\pi$	45
3.12	We plot the probability density for DTQ using a naïve MH-sampler and POMP using an adaptive MH-sampler, for varying intermediate time steps h . The blue, red and black curves are the posterior distributions for $h = \Delta t/2, \Delta t/4$, and $\Delta t/8$ respectively. The solid black line is the value of the true parameter, and the dotted black line represents the MAP estimate.	46
3.13	We plot the estimated likelihood surfaces to get an insight on the approximation of the likelihood, $\ell(\theta)$ for DTQ and POMP. The plots are on the θ_2 vs θ_3 grid using 100 intervals and varying intermediate time steps ($h = \Delta t/2, \Delta t/8$ and $\Delta t/16$). The plots show that the DTQ method produces a smoother approximation of the likelihood as compared to the particle filter.	47
3.14	We plot simulated data for the runner and chaser on the standard basketball court of dimensions 94ft. \times 50 ft. The plot on the left creates the runner's trajectory as $y^r = 5\log(x^r)$. The chaser's speed is $\gamma(t) = 1 + 2t$ and we consider 10 points along the trajectory. For the plot on the right, we consider 50 points on both the trajectories. The runner's trajectory in this case is randomized. $y^r = ((x^r)^2)/100 + (x^r/20) + \mathcal{N}(\mu = 0, \sigma = 2)$	49
3.15	As the number of DTQ steps increase ($h = \Delta t, \Delta t/2, \Delta t/3, \Delta t/4$), the L^2 norm error between the estimated and the true parameters decrease, for $\Delta t = 0.1, 0.2$ and 0.4	50
3.16	The agreement between the black curve (mean of simulated stochastic pursuit trajectories using MAP estimated parameters) and the red curve (chaser's trajectory) shows that the stochastic pursuit model is appropriate. The runner's trajectory is given in blue.	51

- 3.17 For the fast break tracking data described in the text, we plot the MAP estimate of the chaser's speed $\gamma(t)$ in black. Note that the inferred speed differs greatly from the mean speed across the entire trajectory, plotted as a horizontal red line. 51
- 4.1 In order to implement the matrix-vector multiplication in (4.7) 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 (4.7) makes efficient use of Scala, Breeze, and the Intel MKL. For more details, see the description in Section 4.3.1. 58
- 4.2 We use Spark to parallelize the computation of the likelihood (3.2). 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 \vec{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. 60
- 4.3 Illustration of sample paths generated for Ornstein-Uhlenbeck process in the deterministic (left), stochastic (center) and measurement noise (right) setups with the same initial conditions. 61
- 4.4 Posterior densities for the inference/filtering problem with equispaced time series (\mathbf{t}, \mathbf{y}) . Each density is calculated on the basis of 9900 post-burn-in Metropolis samples computed using the indicated value of the internal DTQ time step parameter h . Overall, we see reasonable agreement between the ground truth values (indicated by red vertical lines) and the posterior densities. 62
- 4.5 Posterior densities for the inference/filtering problem with non-equispaced time series (\mathbf{t}, \mathbf{y}) . Each density is calculated on the basis of 9900 post-burn-in Metropolis samples computed using the indicated value of the internal DTQ time step parameter h . Overall, we see reasonable agreement between the ground truth values (indicated by red vertical lines) and the posterior densities. 63
- 4.6 We plot the observations (in red) together with each of the samples of the state series \mathbf{x} . Each such sample is a grey curve, and the mean of all such grey curves is plotted in black. We refer to the black curve as the mean inferred state series. 63

- 4.7 We plot the observations (in red) together with the mean inferred state series (in black). The error bars (grey) are computed by adding/subtracting the mean inferred value of σ_ϵ to/from the observation series \mathbf{y} . Note that the mean inferred state is typically within one σ_ϵ of the corresponding observation. 64
- 4.8 **Left panel:** For each indicated value of L , we have generated a time series of length L , run our inference/filtering code, and recorded the amount of time T required to generate 1000 Metropolis samples of the posterior. We fit lines to $\log T$ as a function of $\log L$ —both the lines and the original data are plotted on log-transformed axes. The slopes of the lines are less than 1, consistent with $O(L)$ temporal scaling. **Right panel:** For a non-equispaced time series of length 2501, we ran our code with ν Spark processors where $\nu \in \{3, 6, 12, 24\}$. We recorded T , the time required to generate 10 Metropolis samples of the posterior. We fit lines to $\log T$ as a function of $\log \nu$ —both the lines and the original data are plotted on log-transformed axes. The slopes of the lines are close to -0.5 , suggesting $O(\nu^{-1/2})$ scaling. 65

List of Tables

2.1	As the maximum allowed degree of Hermite polynomials (M) increase, the number of parameters (\tilde{M}) indicating the multi-index also increase.	10
2.2	Results for average compute time (in seconds) per EM iteration for varying number of time series S . As we increase the number of time series used to learn the drift, the average compute time increases for the 1D, 2D, 3D and 4D systems. The time taken for each EM iteration also increases with the dimensions of the system.	20
2.3	Results for average acceptance percentage for Metropolis-Hastings sampler for varying number of time series S . The acceptance rate decreases as the dimensionality and complexity of the system increases. We suspect that the nonlinearity of the 3D damped Duffing system causes the acceptance rate to be lower than that of the linear 4D system.	20
2.4	Results for number of EM iterations required to converge for varying number of time series S . A threshold of 0.01, 0.05, 0.1 and 0.1 is selected for the 1D, 2D, 3D and 4D systems respectively. If a single time series observation is used for training, then either the relative error in estimated $\tilde{\beta}$ can not be reduced below the threshold (as marked by *), or requires considerably higher number of iterations to become less than the threshold.	20
2.5	Results for average compute time (in seconds) per EM iteration for varying length of time series L . As in Experiment 1 (Section 2.2.1), longer time series implies an increase in the time required to compute one step of the EM algorithm. The average compute time also increases with increasing dimensionality of the system.	22
2.6	Results for average acceptance percentage for Metropolis-Hastings sampler as we increase the length of time series L . As in Experiment 1 (Section 2.2.1), an increase in L results in higher acceptance percentage for the sampler. The acceptance rate decreases with increase in dimensionality and complexity of the system.	22
2.7	Results for number of EM iterations required to converge for varying length of time series L . As in Experiment 1, a threshold of 0.01, 0.05, 0.1 and 0.1 is set for 1D, 2D, 3D and 4D systems respectively. Unlike Experiment 1 (Section 2.2.1), the number of iterations does not change considerably with the length of time series.	22

2.8	Results for average compute time (in seconds) per EM iteration for varying levels of noise strength γ . The time required for EM iterations does not vary with varying noise strength, but it increases as the complexity of the system increases.	25
2.9	Results for average acceptance percentage for Metropolis-Hasting sampler for varying noise strength γ . The acceptance percentage decreases with decrease in γ . As the system becomes more deterministic, it becomes harder to generate Brownian bridge sample paths between the observed data points. The acceptance percentage also decreases with increased complexity of the system.	25
2.10	Results for number of EM iterations required to converge. As in Experiment 1 and 2 (Section 2.2.1, 2.2.2) a threshold of 0.01, 0.05, 0.1 and 0.1 are set for the 1D, 2D, 3D and 4D systems respectively. The number of iterations decrease, in general, with a decrease in the noise strength. In a few cases the relative error between $\tilde{\beta}$ iterates does not reduce below the specified threshold (as marked by *). It is important to note here that the relative error does not reduce below the threshold, but on inspecting the estimated $\tilde{\beta}$ parameter, the estimated value is close to the true value and the relative error is still small.	25
2.11	Results for average compute time (in seconds) per EM iteration for varying amount of data augmentation. As the Brownian bridge is created explicitly using the discretized version of (2.9), increasing the amount of data augmentation does not cause significant increase in the compute time. The time required to compute each EM iteration increases with an increase in the dimensionality of the system.	27
2.12	Results for average acceptance rate for Metropolis-Hastings sampler for varying amount of data augmentation, F . For $F = 1$, no diffusion bridge has been created and thus the acceptance probability is 1. The algorithm in this case reduces to solving a least squares using only the observed time series. As we increase data augmentation, the acceptance probability decreases as it becomes more difficult to create a bridge between the observed values. The acceptance probability also decreases with an increase in the dimensionality and complexity of the system.	28
2.13	Results for number of EM iterations required to converge. As in the previous experiments (Section 2.2.1, 2.2.2, 2.2.3) we consider a threshold of 0.01, 0.05, 0.1 and 0.1 for the 1D, 2D, 3D and 4D systems respectively. The number of EM iterations does not vary significantly with varying amount of data augmentation F	28
3.1	Results for Case 1. Using either 300 or 100 sample paths produced by Euler-Maruyama simulation with time step $\xi = 10^{-4}$, we study the effect of reducing h , the internal DTQ time step.	39
3.2	Results for Case 1. Using either 300 or 100 sample paths produced by Euler-Maruyama simulation with time step $\xi = 10^{-6}$, we study the effect of reducing h , DTQ's internal time step.	40

3.3	Results for Case 2. We study a collection of problems involving different true θ values and different initial guesses θ_0	41
3.4	Results for Case 2. We study the effect of decreasing h , keeping all other parameters fixed.	41
3.5	Results for Case 2. We compare spatial grid laws $k = h^{0.75}$ and $k = h$	42
3.6	Results for Case 2. We examine the effect of increasing the number of sample paths in the data set, keeping all other parameters fixed.	42
3.7	Results for Case 3. We perform inference using model (3.31), which has a higher-dimensional parameter space than (3.30), the model used to generate the data.	43
3.8	Comparison of the mean and standard deviation of inferred parameters, θ_1, θ_2 and θ_3 , for DTQ and POMP, for varying intermediate time steps, $h = \Delta t, \Delta t/2, \Delta t/4$ and $\Delta t/8$	47
3.9	Comparing difference in mean and modes for the estimated parameters from the true parameters for POMP and DTQ methods.	48

Chapter 1

Introduction

Traditional mathematical modeling in the sciences and engineering often has as its goal the development of equations of motion that describe observed phenomena. Classically, these equations of motion took the form of deterministic systems of ordinary or partial differential equations (ODE or PDE, respectively). Especially in systems of contemporary interest in biology and finance where intrinsic noise must be modeled, we find SDEs used instead of the deterministic versions. The popularity of stochastic differential equations (SDEs) for dynamical systems appears to be for many reasons. They provide a flexible framework for modeling steady-state, transient and oscillatory behavior. Solutions of SDEs follow the strong Markov property which allows for scalable computation. They allow physical interpretation of stochastic dynamics, as a counterpart to deterministic modeling using ODEs.

The models presented here have been extensively used for understanding continuous-time phenomena in many scientific fields. An indicative list of applications includes finance and economics (Cox *et al.* (2005); Merton (1976)), biology (Bressloff (2014)), chemistry (Van Kampen (1992)), ecology (Lande *et al.* (2003)), genetics (Shiga (1981)), neuroscience (Lynch and Houghton (2015)) and political analysis (Cobb (1981)).

Recent years have seen a surge of interest in using data to automate discovery of ODE, PDE, and SDE models. Still, these models are often built from first principles or from empirical observations, after which the model's predictions (obtained, for instance, by numerical simulation) are compared against observed data. In modern applications such as climate science and neuroscience, the dynamics are only partially known and difficult to model. Advances in machine learning approaches complement traditional modeling efforts, using available data to constrain the space of plausible models, and shortening the feedback loop linking model development to prediction and comparison to real observations.

We posit two additional reasons to develop algorithms to learn SDE models. First, SDE models—including the models considered here—have the capacity to model highly non-linear, coupled stochastic systems, including systems whose equilibria are non-Gaussian and/or multimodal. Second, SDE models often allow for interpretability. Especially if the terms on the right-hand side of the SDE are expressed in terms of commonly used functions (such as polynomials), we can obtain a qualitative understanding of how the system's variables influence, regulate, and/or mediate one other.

In our work, we explore methods for learning SDE models, using time series data, in an

interpretable form. We consider dynamics driven by an Itô stochastic differential equation

$$d\mathbf{X}_t = \mathbf{f}(\mathbf{X}_t)dt + \mathbf{g}(\mathbf{X}_t)d\mathbf{W}_t, \quad \mathbf{X}_{t_0} = \mathbf{X}_0 \quad (1.1)$$

where \mathbf{X}_t is a stochastic process taking values in \mathbb{R}^d . For rigorous definitions of Brownian motion and SDE, see Bhattacharya and Waymire (2009); Øksendal (2003). Let $t_i, 0 \leq i \leq N$, be the times at which \mathbf{X}_t is observed, and let t_0 be the initial time. \mathbf{W}_t is the Brownian motion in \mathbb{R}^d , also known as the Wiener process. Then $d\mathbf{W}_t$ is an infinitesimal increment of Brownian motion; we think of $d\mathbf{W}_t$ as a multivariate Gaussian random variable with mean vector $\mathbf{0}$ and covariance matrix $\mathbf{I}dt$. The functions \mathbf{f} and \mathbf{g} are called the drift and diffusion functions, respectively. The drift $\mathbf{f}: \mathbb{R}^d \rightarrow \mathbb{R}^d$ is a vector field and the diffusion function $\mathbf{g}: \mathbb{R}^d \rightarrow \mathbb{R}^{d \times d}$ is matrix-valued.

We focus on approximating the likelihood, which is the bottleneck for inference and filtering problems in both the frequentist and Bayesian methods. The drift and diffusion functions can be parametrized as $\mathbf{f}(\mathbf{X}_t; \boldsymbol{\theta})$ and $\mathbf{g}(\mathbf{X}_t; \boldsymbol{\theta})$. The nonparametric estimation problem of finding \mathbf{f} and \mathbf{g} from data is infinite-dimensional. To finite-dimensionalize the problem, we represent each component of \mathbf{f} as a linear combination of M basis functions:

$$f_i(\mathbf{X}_t) = \sum_{m=0}^M \theta_{i,m} \phi_m(\mathbf{X}_t) \quad (1.2)$$

Once $\boldsymbol{\theta}$ is specified, we can compute the drift and diffusion. A time series, $\boldsymbol{\xi}$, consists of N sample observations, \mathbf{x} , of the stochastic process \mathbf{X} :

$$\boldsymbol{\xi} = (\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_N). \quad (1.3)$$

Our goal is to use a collection of S such time series $\Xi = (\boldsymbol{\xi}^{(0)}, \boldsymbol{\xi}^{(1)}, \dots, \boldsymbol{\xi}^{(S)})$ to estimate the drift and diffusion parameters, $\boldsymbol{\theta}$, i.e., the posterior distribution, $p(\boldsymbol{\theta} | \Xi)$. Using Bayes' theorem, this can be defined as

$$p(\boldsymbol{\theta} | \Xi) = \frac{p(\Xi | \boldsymbol{\theta}) p(\boldsymbol{\theta})}{p(\Xi)}, \quad (1.4)$$

where $p(\boldsymbol{\theta})$ is the prior belief about the parameters. Since the observed data remains fixed, its probability, $p(\Xi)$, remains fixed and is seen as a normalization constant. The quantity of importance is the likelihood, $p(\Xi | \boldsymbol{\theta})$, which is the probability of observed data given a particular parameter vector:

$$L(\boldsymbol{\theta}) = p(\Xi | \boldsymbol{\theta}). \quad (1.5)$$

A frequentist method to estimate $\boldsymbol{\theta}$ is to find the value of $\boldsymbol{\theta}$ that maximizes $L(\boldsymbol{\theta})$, also known as maximum likelihood estimation (MLE). Bayesian methods focus on the posterior $p(\boldsymbol{\theta} | \Xi)$. To maximize the posterior, we need to maximize $L(\boldsymbol{\theta}) \times p(\boldsymbol{\theta})$. Thus both frequentist and Bayesian methods require the likelihood function.

Although typically intractable, the transition density has various representations which suggest different approaches for its approximation. First, it can be expressed in various ways as an expectation, and these expressions lend themselves to Monte Carlo approximation. These methods are largely stochastic in nature as they focus on approximating probabilities and generating samples from these approximated models. Second, the tran-

sition density is approximated using linearization techniques, series expansions, path integration, finite difference methods or numerical solutions to the Fokker-Planck equation — see Hurn *et al.* (2007). These methods provide solutions which are deterministic in nature.

We present two methods in this dissertation, developed to generate interpretable SDE models via the two approaches mentioned above. In Section 1.1 and subsequently in Chapter 2, we introduce a Monte Carlo approximation algorithm we developed using diffusion bridge simulation. In Section 1.2, and subsequently in Chapters 3 and 4, we introduce an algorithm based on density approximation.

1.1 SDE Model Discovery using Expectation Maximization (EM) via Bridge Sampling

In Chapter 2, we develop an algorithm to learn SDE models from high-dimensional time series. Our approach relies on the expectation maximization (EM) algorithm (Dempster *et al.* (1977)) along with data augmentation using diffusion bridge sampling (Roberts and Stramer (2001)).

EM is an iterative method to find maximum likelihood (MLE) or maximum a posteriori (MAP) estimates of parameters in statistical models, where the model depends on unobserved latent variables. The EM iteration alternates between performing an expectation (*E*) step, which creates a function for the expectation of the log likelihood evaluated using the current estimate for the parameters, and a maximization (*M*) step, which computes parameters maximizing the expected log likelihood found in the *E* step.

In the absence of high-frequency data, simulation of diffusion bridges plays a fundamental role in computing the *E* step for discretely sampled diffusion processes, by producing time series observed at a higher frequency than the original data. In missing data problems, data augmentation via diffusion bridge sampling takes advantage of the fact that the resulting optimization of the expected log likelihood in the *M* step, of the complete data, reduces to a least squares problem.

Diffusion bridge simulation is a highly non-trivial problem as the diffusion bridge is an infinite-dimensional random variable. It is the process obtained by conditioning a diffusion to start and finish at specific values at two consecutive times, $t_0 < t_1$. Simulating a strong solution of the diffusion bridge corresponds to jointly constructing X and W , while simulating a weak solution only asks for simulating X according to the probability law implied by (1.1). At a fundamental level, simulation of unconditional diffusion is difficult since the transition distribution of the diffusion is intractable. The diffusion bridge, however, solves an SDE whose drift is intractable, hence even approximate simulation using density approximating methods as described above, is infeasible (Papaspiliopoulos and Roberts (2012)). This leads to the use of Monte Carlo methods for simulating diffusion bridges.

It was previously thought impossible to simulate diffusion bridges by means of simple procedures. A rejection sampler that tries to hit the prescribed end-point for the bridge will have an excessively high rejection probability. We use a Markov Chain Monte Carlo (MCMC) diffusion bridge sampler similar to those proposed by Roberts and Stramer (2001); Bladt *et al.* (2014). These samplers achieve reasonable acceptance rates by trying to hit a sample path rather than a point.

1.1.1 Summary

To briefly introduce our EM with bridge sampling approach, we describe the four main steps in the process:

1. For each time interval $[t_i, t_{i+1}]$, generate an ensemble of diffusion bridge sample paths, Ψ , conditioned on the initial value, x_{t_i} , and final value, $x_{t_{i+1}}$, in the interval.
2. Numerically approximate the Girsanov likelihood of the proposal. Iterate through a MCMC algorithm to accept/reject the proposal using the ratio of the likelihoods of the current iterate and the proposal.
3. Approximate the expected likelihood, $p(\Xi|\theta)$, using the complete data likelihood, $p(\Xi, \Psi|\theta)$, for the observed data Ξ and augmented data Ψ .
4. To maximize the expected likelihood, solve the batch-wise least squares problem iteratively, alternating between the drift and diffusion parameters

In our current work, we parameterize the drift vector field using tensor products of Hermite polynomials, enabling the model to capture highly nonlinear and/or coupled dynamics. Please note that the method itself only requires the drift and diffusion functions to be sufficiently smooth for the Girsanov theorem to hold (Rogers and Williams (1994)). The functions can also be expressed in terms of other basis functions or, in general, an additive model of a dictionary of user-defined functions.

Through experiments on systems with dimensions one through four, we show that this EM approach enables accurate estimation for multiple time series with possibly irregular observation times. We study how the EM method performs as a function of the noise level in the data, the volume of data, and the amount of data augmentation performed.

The current limitation of our work lies in the scalability of diffusion bridge sampling. It becomes increasingly difficult to create a diffusion bridge sampler with reasonable acceptance rate as the dimension increases. The sampler also has low sampling rate when the diffusion coefficient \mathbf{g} vanishes and the resulting system becomes a completely deterministic process. The loss of stochasticity is inconsistent with the diffusion bridge formulation.

1.1.2 Related Work

As mentioned, differential equation discovery problems have attracted considerable recent interest. A variety of methods have been developed to learn ODE (Brunton *et al.* (2016); Schön *et al.* (2018); Chen *et al.* (2017b); Tran and Ward (2017); Schaeffer *et al.* (2017); Schaeffer (2017); Quade *et al.* (2018)) as well as PDE (Schaeffer *et al.* (2013); Raissi *et al.* (2017); Rudy *et al.* (2017); Raissi and Karniadakis (2018)). Recently, there has been work on the stochastic counterpart by (Boninsegna *et al.* (2017)), that focuses on high-frequency stochastic data. Unlike many of these works, we do not focus on model selection through cross-validation and/or regularization; if needed, our methods can be combined with model selection procedures developed in the ODE context (Mangan *et al.* (2016, 2017)).

Many other techniques explored in the statistical literature focus on scalar SDE (Nicoulaou (2007); Müller *et al.* (2010); Verzelin *et al.* (2012); Bhat and Madushani (2016)). Prior EM approaches were restricted to one-dimensional SDE (Ghahramani and Roweis (1999)),

or used a Gaussian process approximation, linear drift approximation, and approximate maximization (Ruttor *et al.* (2013)).

In the literature, variational Bayesian methods are another class of well-known SDE learning methods that have been tested on high-dimensional problems (Vrettas *et al.* 2015). These methods use approximations consisting of linear SDE with time-varying coefficients (Archambeau *et al.* 2008), kernel density estimates (Batz *et al.* 2016), or Gaussian processes (Batz *et al.* 2017). In contrast, we parameterize the drift vector field using tensor products of Hermite polynomials; as mentioned above, the resulting SDE has much higher capacity for nonlinear expression than linear and/or Gaussian process models.

1.2 Scalable inference and filtering using Density Tracking by Quadrature (DTQ)

In Chapters 3 and 4, we present our work on parameter inference and filtering for SDEs, respectively. Our approach relies on the theory of Markov diffusion processes. The justification of using this theory lies in the fact that if the drift and diffusion functions are sufficiently regular, the SDE (1.1) has a unique strong solution for each initial condition, and this solution is a time-homogeneous strong Markov process (Rogers and Williams (1994)). In our work, we develop a convergent numerical method to approximate the transition density, $p(\Xi, t)$, using temporal and spatial discretization, thus the name, density tracking by quadrature (DTQ).

1.2.1 Summary

To briefly introduce DTQ, we describe the three main steps in the derivation:

1. Discretize the SDE (1.1) in time using a time-integrator
2. Interpret the time-discretized equations as a discrete-time Markov chain with density \tilde{p} .
3. Write the Chapman-Kolmogorov equations for the time-evolution of \tilde{p} . Discretize both the Chapman-Kolmogorov equation and \tilde{p} in space to compute the discrete-space density approximation \hat{p} .

In our current work, we use the explicit Euler-Maruyama method in step 1 and the trapezoidal rule for numerical quadrature in step 3. Note that the above steps constitute a general framework for numerical density computation. With different choices of time integrator and quadrature rules, variations of the DTQ method can be derived for improved performance. The DTQ method is provably $\mathcal{O}(h)$ convergent, where $h > 0$ denotes the temporal step size (Bhat and Madushani (2016)). The convergence of $\hat{p} \rightarrow \tilde{p}$ is exponential in h , while the $\tilde{p} \rightarrow p$ convergence is of order 1 (Bally and Talay (1996)). Thus the convergence of the time integrator dictates the convergence of the DTQ method. Many higher-order schemes exist based on the Itô-Taylor expansion, implicit methods and split-step methods (Kloeden and Platen (1992)), which can be used for improved convergence.

To study the properties of our algorithm, we run a series of preliminary tests involving both inference and filtering models. We show that our algorithm is capable of accurate

inference, and that its performance depends in a logical way on problem and algorithm parameters.

1.2.2 Related Work

The problem of estimating solutions of SDEs and the underlying likelihood has been a demanding challenge for several decades because of intractability of the likelihood. A thorough review of past work on this problem is contained in Sørensen (2004); Iacus (2009); Fuchs (2013); Chen (2003); Hurn *et al.* (2007). Here we focus on past work that is particularly relevant to our approach.

Methodology similar to DTQ has previously appeared in literature as numerical path integration (NPI). Early efforts of using a path integration method for numerically estimating the probability density of a stochastic process have been focused on solving the Fokker-Planck equation as it provides analogies between quantum mechanics and nonequilibrium thermodynamics (Zambrini and Yasue (1980)). The numerical path integration scheme proposed by Wehner and Wolfer (1983) uses the midpoint rule for numerical quadrature and a short-term propagator computed by Wissel (1979) for the time evolution. The method has achieved accurate results for long-term evolution and extreme response statistics on a variety of problems in, e.g., nonlinear mechanics and finance — see Næss and Johnsen (1993); Linetsky (1997); Yu *et al.* (1997); Rosa-Clot and Taddei (2002); Skaug and Næss (2007); Chen *et al.* (2017a).

The transition density of the Fokker-Planck equation can be approximated analytically using orthogonal polynomials (Cukier *et al.* (1973)), closed-form approximations (Ait-Sahalia (2002)) and eigenfunction expansions (Schenzle and Brand (1979)), and numerically integrated using Monte Carlo methods (Kikuchi *et al.* (1991)) and finite element schemes (Di Paola and Sofi (2002)). The Chapman-Kolmogorov equation that is at the center of our approach has appeared in Pedersen (1995); Santa-Clara (1997). In these works, the right-hand side of the Chapman-Kolmogorov equation is interpreted as an expected value computed using Monte Carlo methods. Cai (2002) has used a similar approach in the context of nonlinear autoregressive time series model. The Bayesian approach to the intractability of the likelihood function focuses on introducing auxiliary points to create a high-frequency dataset (Fuchs (2013)).

1.3 Outline

The dissertation is structured in the following way. In Chapter 2, we focus on nonparametric drift estimation. In Section 2.1.3, we describe the diffusion bridge sampling computation, followed by the Expectation Maximization steps in Section 2.1.4. To study the properties of our algorithm, we run a series of experiments with 1D, 2D, 3D and 4D systems in Section 2.2. We record the error in the estimated and true parameters by varying the number of time series used, the length of each time series, the level of noise strength and the amount of data augmentation. In Section 2.3, we discuss future work for non-constant diffusion matrices and exploring regularization to induce sparsity.

In Chapter 3, we propose the DTQ method. In Section 3.2.1 and 3.2.2, we derive the likelihood and gradient computations, respectively, as simple vector-matrix and matrix-matrix multiplications. In Section 3.3 we extend the scalar model to a two-dimensional

coupled system and derive the likelihood and gradient, as a generalization of the computations in Section 3.2. In Section 3.4, we present the results for parameter inference for the Ornstein-Uhlenbeck process (linear), double well potential (nonlinear) and a generic polynomial drift and diffusion function. We record the estimated parameters and the number of iterations vs RMS error for varying temporal and spatial grid spacing. In Section 3.4.4, we further present our results for the coupled system, using two cases: an electrical oscillator, and a pursuit model using basketball player tracking data. We also provide a comparison study between a particle filtering approach (pomp) and DTQ to highlight strengths and weaknesses of both methods. In Section 3.5, we discuss future improvements for scalable implementation of the DTQ method in the high noise regime. The work from this Chapter has been published (Bhat *et al.* (2015, 2016a)).

In Chapter 4, we extend our work on DTQ to infer both state and system parameters from time series data. This problem is known as filtering, since the observed variables have measurement noise and this noise is filtered out to recover the true system variable observations. In Section 4.2, we introduce the measurement noise model and the associated inference problem in the Bayesian setup. DTQ proves to be a natural extension for the likelihood computation as the bottleneck for both the models is the same. In Section 4.2.2, we enumerate the steps for an MCMC sampler which samples from the posterior. As the inferred parameters scale linearly with the length of the time series, we provide a scalable implementation of our method on the Scala/Breeze platform, the first of its kind in Chapter 4.3. In Chapter 4.4, we report the results of our method for inferred state and system parameters for the Ornstein-Uhlenbeck process for equispaced and non-equispaced time series. In Section 4.4.3, we conduct tests to explore the relationship between running time and the length of observation series. In Chapter 4.5, we discuss future work in the direction of batch filtering and adjoint-based DTQ method for faster gradient computation. The work presented in this Chapter has been published (Bhat *et al.* (2016b)).

Chapter 2

SDE Model Discovery via Bridge Sampling

In this chapter, we develop an algorithm to learn SDE models from high-dimensional time series. To our knowledge, this is the most general expectation maximization (EM) approach to learning an SDE with multidimensional drift vector field and diagonal diffusion matrix. Prior EM approaches were restricted to one-dimensional SDE (Ghahramani and Roweis (1999)), or used a Gaussian process approximation, linear drift approximation, and approximate maximization (Ruttor *et al.* (2013)). To develop our method, we use diffusion bridge sampling as in van der Meulen *et al.* (2014, 2017), which focused on Bayesian non-parametric methods for SDE in \mathbb{R}^1 . After augmenting the data using bridge sampling, we are left with a least-squares problem, generalizing the work of Brunton *et al.* (2016) from the ODE to the SDE context.

2.1 Problem Setup

Let W_t denote Brownian motion in \mathbb{R}^d —informally, an increment dW_t of this process has a multivariate normal distribution with zero mean vector and covariance matrix Idt . Let X_t denote an \mathbb{R}^d -valued stochastic process that evolves according to the Itô SDE

$$dX_t = f(X_t)dt + \Gamma dW_t. \quad (2.1)$$

For rigorous definitions of Brownian motion and SDE, see Bhattacharya and Waymire (2009); Øksendal (2003). The nonlinear vector field $f : \Omega \subset \mathbb{R}^d \rightarrow \mathbb{R}^d$ is the *drift* function, and the $d \times d$ matrix Γ is the *diffusion* matrix. To reduce the number of model parameters, we assume $\Gamma = \text{diag} \gamma$.

Our goal is to develop an algorithm that accurately estimates the functional form of f and the vector γ from time series data.

2.1.1 Parameterization

We parameterize f using Hermite polynomials. The n -th Hermite polynomial takes the form

$$H_n(x) = (\sqrt{2\pi}n!)^{-1/2}(-1)^n e^{x^2/2} \frac{d^n}{dx^n} e^{-x^2/2} \quad (2.2)$$

Let $\langle f, g \rangle_w = \int_{\mathbb{R}} f(x)g(x) \exp(-x^2/2) dx$ denote a weighted L^2 inner product. Then, $\langle H_i, H_j \rangle_w = \delta_{ij}$, i.e., the Hermite polynomials are orthonormal with respect to the weighted inner product. In fact, with respect to this inner product, the Hermite polynomials form an orthonormal basis of $L_w^2(\mathbb{R}) = \{f \mid \langle f, f \rangle_w < \infty\}$.

Now let $\alpha = (\alpha_1, \dots, \alpha_d) \in \mathbb{Z}_+^d$ denote a multi-index. We use the notation $|\alpha| = \sum_j \alpha_j$ and $x^\alpha = \prod_j (x_j)^{\alpha_j}$ for $x = (x_1, \dots, x_d) \in \mathbb{R}^d$. For $x \in \mathbb{R}^d$ and a multi-index α , we also define

$$H_\alpha(x) = \prod_{j=1}^d H_{\alpha_j}(x_j). \quad (2.3)$$

We write $f(x) = (f_1(x), \dots, f_d(x))$ and then parameterize each component

$$f_j(x) = \sum_{m=0}^M \sum_{|\alpha|=m} \beta_\alpha^j H_\alpha(x). \quad (2.4)$$

We see that the maximum degree of $H_\alpha(x)$ is $|\alpha|$. Hence we think of the double sum in (2.4) as first summing over degrees and then summing over all terms with a fixed maximum degree. We say maximum degree because, for instance, $H_2(z) = (z^2 - 1)/(2\sqrt{2\pi})^{1/2}$ contains both degree 2 and degree 0 terms.

dim (d)	Max degree of Hermite polynomial (M)						
	0	1	2	3	4	5	6
1	1	2	3	4	5	6	7
2	1	3	6	10	15	21	28
3	1	4	10	20	35	56	84
4	1	5	15	35	70	126	210
5	1	6	21	56	126	252	462
6	1	7	28	84	210	462	924

Table 2.1: As the maximum allowed degree of Hermite polynomials (M) increase, the number of parameters (\widetilde{M}) indicating the multi-index also increase.

There are $\binom{m+d-1}{d-1}$ possibilities for a d -dimensional multi-index α such that $|\alpha| = m$. Summing this from $m = 0$ to M , there are $\widetilde{M} = \binom{M+d}{d}$ total multi-indices in the double sum in (2.4). Let (i) denote the i -th multi-index according to some ordering. Then we can write

$$f_j(x) = \sum_{i=1}^{\widetilde{M}} \beta_{(i)}^j H_{(i)}(x). \quad (2.5)$$

Essentially, we parameterize f using tensor products of Hermite polynomials.

2.1.2 Data

We consider our data $\mathbf{x} = \{x_j\}_{j=0}^L$ to be direct observations of X_t at discrete points in time $\mathbf{t} = \{t_j\}_{t=0}^L$. Note that these time points do not need to be equispaced. In the derivation

that follows, we will consider the data (\mathbf{t}, \mathbf{x}) to be one time series. Later, we indicate how our methods generalize naturally to multiple time series, i.e., repeated observations of the same system.

To achieve our estimation goal, we apply expectation maximization (EM). We regard \mathbf{x} as the incomplete data. Let $\Delta t = \max_j(t_j - t_{j-1})$ be the maximum inter-observation spacing. We think of the missing data \mathbf{z} as data collected at a time scale $h \ll \Delta t$ fine enough such that the transition density of (2.1) is approximately Gaussian. To see how this works, let $\mathcal{N}(\mu, \Sigma)$ denote a multivariate normal with mean vector μ and covariance matrix Σ . Now discretize (2.1) in time via the Euler-Maruyama method with time step $h > 0$; the result is

$$\tilde{X}_{n+1} = \tilde{X}_n + f(\tilde{X}_n)h + h^{1/2}\Gamma Z_{n+1}, \quad (2.6)$$

where $Z_{n+1} \sim \mathcal{N}(0, I)$ is a standard multivariate normal, independent of X_n . This implies that

$$(\tilde{X}_{n+1} | \tilde{X}_n = v) \sim \mathcal{N}(v + f(v)h, h\Gamma^2). \quad (2.7)$$

As h decreases, $\tilde{X}_{n+1} | \tilde{X}_n = v$ —a Gaussian approximation—will converge to the true transition density $X_{(n+1)h} | X_{nh} = v$, where X_t refers to the solution of (2.1).

2.1.3 Diffusion Bridge

To augment or complete the data, we employ diffusion bridge sampling, using a Markov chain Monte Carlo (MCMC) method that goes back to Roberts and Stramer (2001); Paspaliopoulos *et al.* (2013). Let us describe our version here. We suppose our current estimate of $\theta = (\beta, \gamma)$ is given. Define the diffusion bridge process to be (2.1) conditioned on both the initial value x_i at time t_i , and the final value x_{i+1} at time t_{i+1} . The goal is to generate sample paths of this diffusion bridge. By a sample path, we mean $F - 1$ new samples $\{z_{i,j}\}_{j=1}^{F-1}$ at times $t_i + jh$ with $h = (t_{i+1} - t_i)/F$.

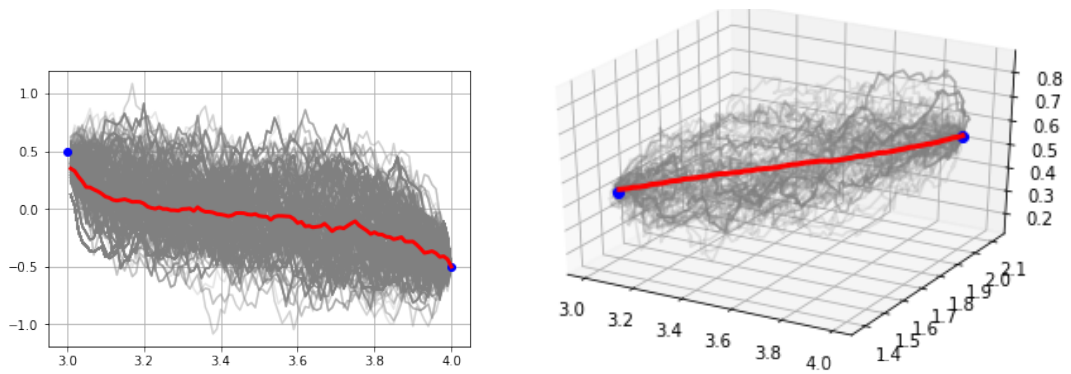


Figure 2.1: Illustration of 1000 Brownian bridge sample paths, post burnin, in 1D (left) and 2D (right) from a fixed initial to final point over a time interval. The blue dots represent the observed data. The grey curves represent the Brownian bridge sample paths between the two observed points and the red line represents the mean of the sample paths.

To generate such a path, we start by drawing a sample from a Brownian bridge with

the same diffusion as (2.1). That is, we sample from the SDE

$$d\widehat{X}_t = \Gamma dW_t \quad (2.8)$$

conditioned on $\widehat{X}_{t_i} = x_i$ and $\widehat{X}_{t_{i+1}} = x_{i+1}$. This Brownian bridge can be described explicitly

$$\widehat{X}_t = \Gamma(W_t - W_{t_i}) + x_i - \frac{t - t_i}{t_{i+1} - t_i}(\Gamma(W_{t_{i+1}} - W_{t_i}) + x_i - x_{i+1}) \quad (2.9)$$

Here $W_0 = 0$ (almost surely), and $W_t - W_s \sim \mathcal{N}(0, (t-s)I)$ for $t > s \geq 0$.

Let \mathbb{P} denote the law of the diffusion bridge process, and let \mathbb{Q} denote the law of the Brownian bridge (2.9). Using Girsanov's theorem (Papaspiliopoulos and Roberts (2012)), we can show that

$$\frac{d\mathbb{P}}{d\mathbb{Q}} = C \exp\left(\int_{t_i}^{t_{i+1}} f(\widehat{X}_s)^T \Gamma^{-2} d\widehat{X}_s - \frac{1}{2} \int_{t_i}^{t_{i+1}} f(\widehat{X}_s)^T \Gamma^{-2} f(\widehat{X}_s) ds\right), \quad (2.10)$$

where the constant C depends only on x_i and x_{i+1} . The left-hand side is a Radon-Nikodym derivative, equivalent to a density or likelihood; the ratio of two such likelihoods is the accept/reject ratio in the Metropolis algorithm (Stuart (2010)).

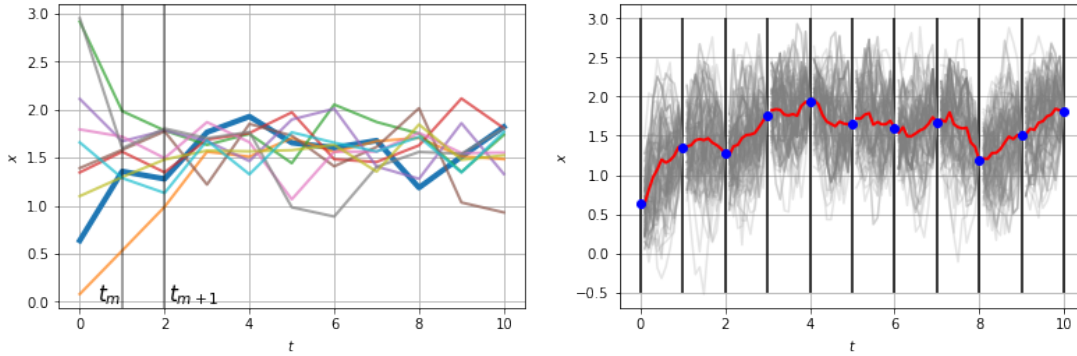


Figure 2.2: On the left, we plot 10 time series observed 21 times over the time interval $[0, 10]$. On the right, we plot 100 Brownian bridge paths between each observed data point for the solid blue series in the left plot. The blue dots represent the observed data. The grey curves represent the Brownian bridge sample paths between the initial and final observed data point for each interval. The red line represents the mean of the Brownian bridge path samples.

Putting the above pieces together yields the following Metropolis algorithm to generate diffusion bridge sample paths. Fix $F \geq 2$ and $i \in \{0, \dots, L-1\}$. Assume we have stored the previous Metropolis step, i.e., a path $\mathbf{z}^{(\ell)} = \{z_{i,j}^{(\ell)}\}_{j=1}^{F-1}$.

1. Use (2.9) to generate samples of \widehat{X}_t at times $t_i + jh$, for $j = 1, 2, \dots, F-1$ and $h = (t_{i+1} - t_i)/F$. This is the proposal $\mathbf{z}^* = \{z_{i,j}^*\}_{j=1}^{F-1}$.
2. Numerically approximate the integrals in (2.10) to compute the likelihood of the pro-

posal. Specifically, we compute

$$\frac{p(\mathbf{z}^*)}{C} = \sum_{j=0}^{F-1} f(z_{i,j}^*)^T \Gamma^{-2} (z_{i,j+1}^* - z_{i,j}^*) - \frac{h}{4} \sum_{j=0}^{F-1} \left[f(z_{i,j}^*)^T \Gamma^{-2} f(z_{i,j}^*) + f(z_{i,j+1}^*)^T \Gamma^{-2} f(z_{i,j+1}^*) \right]$$

We have discretized the stochastic $d\widehat{X}_s$ integral using Itô's definition, and we have discretized the ordinary ds integral using the trapezoidal rule.

3. Accept the proposal with probability $p(\mathbf{z}^*)/p(\mathbf{z}^{(\ell)})$ —note the factors of C cancel. If the proposal is accepted, then set $\mathbf{z}^{(\ell+1)} = \mathbf{z}^*$. Else set $\mathbf{z}^{(\ell+1)} = \mathbf{z}^{(\ell)}$.

We initialize this MCMC algorithm with a Brownian bridge path and use post-burn-in steps as the diffusion bridge samples we seek.

2.1.4 Expectation Maximization (EM)

Let us now give details to justify the intuition expressed above, that employing the diffusion bridge to augment the data on a fine scale will enable estimation. Let $\mathbf{z}^{(r)} = \{z_{i,j}^{(r)}\}_{j=1}^{F-1}$ be the r -th diffusion bridge sample path. We interleave this sampled data together with the observed data \mathbf{x} to create the completed time series

$$\mathbf{y}^{(r)} = \{y_j^{(r)}\}_{j=1}^N,$$

where $N = LF + 1$. By interleaving, we mean that $y_{1+iF}^{(r)} = x_i$ for $i = 0, 1, \dots, L$, and that $y_{1+j+iF}^{(r)} = z_{i,j}$ for $j = 1, 2, \dots, F-1$ and $i = 0, 1, \dots, L-1$. With this notation, we can more easily express the EM algorithm. Let us assume that we currently have access to $\boldsymbol{\theta}^{(k)}$, our estimate of the parameters after k iterations. If $k = 0$, we set $\boldsymbol{\theta}^{(0)}$ equal to an initial guess. Then we follow two steps:

1. For the expectation (E) step, we first generate an ensemble of R diffusion bridge sample paths. Interleaving as above, this yields R completed time series $\mathbf{y}^{(r)}$ for $r = 1, \dots, R$. In what follows, we will use an average over this ensemble to approximate the expected value. Let h_j denote the elapsed time between observations y_j and y_{j+1} . Using the completed data, the temporal discretization (2.6) of the SDE, the Markov property, and property (2.7), we have:

$$\begin{aligned} Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(k)}) &= \mathbb{E}_{\mathbf{z}|\mathbf{x}, \boldsymbol{\theta}^{(k)}} [\log p(\mathbf{x}, \mathbf{z} | \boldsymbol{\theta})] \\ &\approx \frac{1}{R} \sum_{r=1}^R \log p(\mathbf{y}^{(r)} | \boldsymbol{\theta}) \\ &= \frac{1}{R} \sum_{r=1}^R \sum_{n=1}^{N-1} \log p(y_{n+1}^{(r)} | y_n^{(r)}, \boldsymbol{\theta}) \\ &= -\frac{1}{R} \sum_{r=1}^R \sum_{n=1}^{N-1} \left[\sum_{j=1}^d \frac{1}{2} \log(2\pi h_n \gamma_j^2) + \frac{1}{2h_n} \left\| \Gamma^{-1} \left(y_{n+1}^{(r)} - y_n^{(r)} - h_n \sum_{\ell=1}^{\widetilde{M}} \beta_{(\ell)} H_{(\ell)}(y_n^{(r)}) \right) \right\|_2^2 \right]. \end{aligned} \tag{2.11}$$

2. For the maximization (M) step, we carry out:

$$\boldsymbol{\theta}^{(k+1)} = \arg \max_{\boldsymbol{\theta}} Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(k)})$$

Note that $y_j^{(r)} \in \mathbb{R}^d$ —we denote the i -th component by $y_j^{(r),i}$. We find $\beta^{(k+1)}$ by solving $\mathcal{M}\beta = \rho$ where \mathcal{M} is the $\widetilde{M} \times \widetilde{M}$ matrix

$$\mathcal{M}_{k,\ell} = \frac{1}{R} \sum_{r=1}^R \sum_{n=1}^{N-1} h_n H_{(k)}(y_n^{(r)}) H_{(\ell)}(y_n^{(r)}), \quad (2.12)$$

and ρ is the $\widetilde{M} \times d$ matrix

$$\rho_{k,i} = \frac{1}{R} \sum_{r=1}^R \sum_{n=1}^{N-1} H_{(k)}(y_n^{(r)}) (y_{n+1}^{(r),i} - y_n^{(r),i}). \quad (2.13)$$

We find $\gamma^{(k+1)}$ by computing

$$\gamma_i^2 = \frac{1}{R(N-1)} \sum_{r=1}^R \sum_{n=1}^{N-1} h_n^{-1} (y_{n+1}^{(r),i} - y_n^{(r),i} - h_n \sum_{\ell=1}^{\widetilde{M}} \beta_{(\ell)}^i H_{(\ell)}(y_n^{(r)}))^2. \quad (2.14)$$

Here $\beta_{(\ell)}^i$ denotes the ℓ -th row and i -th column of the $\beta^{(k+1)}$ matrix. We then set $\theta^{(k+1)} = (\beta^{(k+1)}, \gamma^{(k+1)})$.

We iterate the above two steps until $\|\theta^{(k+1)} - \theta^{(k)}\| / \|\theta^{(k)}\| < \delta$ for some tolerance $\delta > 0$.

When the data consists of multiple time series $\{\mathbf{t}^{(i)}, \mathbf{x}^{(i)}\}_{i=1}^S$, everything scales accordingly. For instance, we create an ensemble of R diffusion bridge samples for each of the S time series. If we index the resulting completed time series appropriately, we simply replace R by RS in (2.12), (2.13), and (2.14) and keep everything else the same.

There are three sources of error in the above algorithm. The first relates to replacing the expectation by a sample average; the induced error should, by the law of large numbers, decrease as $R^{-1/2}$. The second stems from the approximate nature of the computed diffusion bridge samples—as indicated above, we use numerical integration to approximate the Girsanov likelihood. The third source of error is in using the Gaussian transition density to approximate the true transition density of the SDE. Both the second and third sources of error vanish in the $F \rightarrow \infty$ limit (Kloeden and Platen (2011)).

2.2 Experiments

We present a series of increasingly higher-dimensional experiments with synthetic data. To generate this data, we start with a known stochastic dynamical system of the form (2.1). Using Euler-Maruyama time stepping starting from a randomly chosen initial condition, we march forward in time from $t = 0$ to a final time $t = 10$.

In all examples, we step forward internally at a time step of $h = 0.0001$, but for the purposes of estimation, we only use data sampled every 0.1 units of time, discarding 99.9% of the simulated trajectory. We use a fine internal time step to reduce, to the extent possible, numerical error in the simulated data. We save the data on a coarse time scale to test the proposed EM algorithm.

To study how the EM method performs as a function of noise strength, data volume, and data augmentation, we perform four sets of experiments. When we run EM, we randomly generate the initial guess $\beta^{(0)} \sim \mathcal{N}(\mu = 0, \sigma^2 = 0.5)$. We set the EM tolerance parameter $\delta = 0.01$. The only regularization we include is to threshold β —values less than 0.01 in

absolute value are reset to zero. Finally, in the MCMC diffusion bridge sampler, we use 10 burn-in steps and then create an ensemble of size $R = 100$.

To quantify the error between the estimated $\tilde{\beta}$ and the true β , we use the Frobenius norm as the regression error metric. Since drift function is represented as an additive model of Hermite functions, the error reduces to L^2 error in the respective coefficients:

$$\varepsilon = \|f_{\text{true}} - f_{\text{estimated}}\|_2 = \sqrt{\sum_i \|\beta_{(i)} - \tilde{\beta}_{(i)}\|^2}. \quad (2.15)$$

We also define classification metrics which quantifies the number of active and inactive polynomial terms correctly identified. Most physical systems have only a few relevant terms that define the dynamics, making the governing equations sparse in a high-dimensional function space. We use hard thresholding to investigate the reasonable amount of thresholding, λ , for meaningful classification tasks in different scenarios:

$$\tilde{\beta} = (|\tilde{\beta}| - \lambda)_+ \quad (2.16)$$

To define the classification metrics, we define four intermediate terms. These terms focus on if the entries have been identified correctly as zero/non-zero:

	$\tilde{\beta}_{(i)} = 0$	$\tilde{\beta}_{(i)} \neq 0$
$\beta_{(i)} = 0$	True Positive (TP)	False Negative (FN)
$\beta_{(i)} \neq 0$	False Positive (FP)	True Negative (TN)

Using these definitions, we compute the classification error using the following metrics:

- Precision is the proportion of positive identifications that are correct, i.e., out of all the polynomial terms rejected by the estimated system, how many do not contribute in the true system:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} = \sum_i \frac{\beta_{(i)} = 0 \wedge \tilde{\beta}_{(i)} = 0}{\tilde{\beta}_{(i)} = 0}. \quad (2.17)$$

- Recall is the proportion of actual positives correctly identified, i.e., out of all the inactive terms in the true system, how many have been estimated correctly as inactive:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} = \sum_i \frac{\beta_{(i)} = 0 \wedge \tilde{\beta}_{(i)} = 0}{\beta_{(i)} = 0}. \quad (2.18)$$

- F-measure or balanced F1-score combines precision and recall using the harmonic mean:

$$\text{F1 score} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}. \quad (2.19)$$

The $\tilde{\beta}$ coefficients are the Hermite coefficients of the estimated drift vector field f . For each example system, we compute the true Hermite coefficients β by multiplying the true ordinary polynomial coefficients by a change-of-basis matrix that is easily computed by

solving the system of equations for the equivalent conversion. We provide the calculations for a general 1D system with the maximum degree $M = 3$. The coefficients of the ordinary polynomial terms are represented as B .

$$\begin{aligned} B_{(0)} + B_{(1)}x + B_{(2)}x^2 + B_{(3)}x^3 &= \beta_{(0)}H_{(0)}(x) + \beta_{(1)}H_{(1)}(x) + \beta_{(2)}H_{(2)}(x) + \beta_{(3)}H_{(3)}(x) \\ &= (\beta_{(0)}H_0 - \beta_{(2)}H_2) + (\beta_{(1)} - 3\beta_{(3)}H_3)x + (\beta_{(2)}H_2)x^2 + (\beta_{(3)}H_3)x^3 \end{aligned}$$

This results in a transformation matrix from Hermite space to ordinary polynomial space as $H\beta = B$ and from ordinary polynomial space to Hermite space as $\beta = H^{-1}B$:

$$\begin{bmatrix} H_0 & & & & \\ & H_1 & & & \\ & & H_2 & & \\ & & & H_3 & \\ & & & & & H_3 \end{bmatrix} \begin{bmatrix} \beta_{(0)} \\ \beta_{(1)} \\ \beta_{(2)} \\ \beta_{(3)} \end{bmatrix} = \begin{bmatrix} B_{(0)} \\ B_{(1)} \\ B_{(2)} \\ B_{(3)} \end{bmatrix}$$

This system can be generalized to a d -dimensional system and H can be represented as an $\widetilde{M} \times \widetilde{M}$ matrix. We provide the transformation matrix for the 2D system to illustrate how this works:

$$\begin{bmatrix} H_0^2 & & & & & & & & \\ & H_0H_1 & & & & & & & \\ & & H_0H_1 & & & & & & \\ & & & H_0H_2 & & & & & \\ & & & & H_1^2 & & & & \\ & & & & & H_0H_2 & & & \\ & & & & & & H_0H_3 & & \\ & & & & & & & H_1H_2 & \\ & & & & & & & & H_1H_2 & \\ & & & & & & & & & H_0H_3 \end{bmatrix} \begin{bmatrix} \beta_{(0,0)} \\ \beta_{(1,0)} \\ \beta_{(0,1)} \\ \beta_{(2,0)} \\ \beta_{(1,1)} \\ \beta_{(0,2)} \\ \beta_{(3,0)} \\ \beta_{(2,1)} \\ \beta_{(1,2)} \\ \beta_{(0,3)} \end{bmatrix} = \begin{bmatrix} B_{(0,0)} \\ B_{(1,0)} \\ B_{(0,1)} \\ B_{(2,0)} \\ B_{(1,1)} \\ B_{(0,2)} \\ B_{(3,0)} \\ B_{(2,1)} \\ B_{(1,2)} \\ B_{(0,3)} \end{bmatrix}$$

We test the method using stochastic systems in dimensions $d = 1, 2, 3$ and 4. In what follows, we call these systems as 1D, 2D, 3D and 4D. We plot the observed data in the form of 10 time series used for learning. The initial conditions are drawn from a Gaussian random distribution. Each time series has initial time = 0 and final time = 10, and 100 time points are observed over the interval. The additive noise is set at 0.05 for each case.

- In 1D, we use: $dX_t = (1 + X_t - X_t^2)dt + \gamma dW_t$.
- In 2D, we use a stochastic Duffing oscillator with no damping or driving:

$$\begin{aligned} dX_{0,t} &= X_{1,t}dt + \gamma_0 dW_{0,t} \\ dX_{1,t} &= (-X_{0,t} - X_{0,t}^3)dt + \gamma_1 dW_{1,t} \end{aligned}$$

- In 3D, we consider the stochastic, damped, driven Duffing oscillator:

$$\begin{aligned} dX_{0,t} &= X_{1,t}dt + \gamma_0 dW_{0,t} \\ dX_{1,t} &= (X_{0,t} - X_{0,t}^3 - 0.3X_{1,t} + 0.5 \cos(X_{2,t}))dt + \gamma_1 dW_{1,t} \\ dX_{2,t} &= 1.2dt + \gamma_2 dW_{2,t} \end{aligned}$$

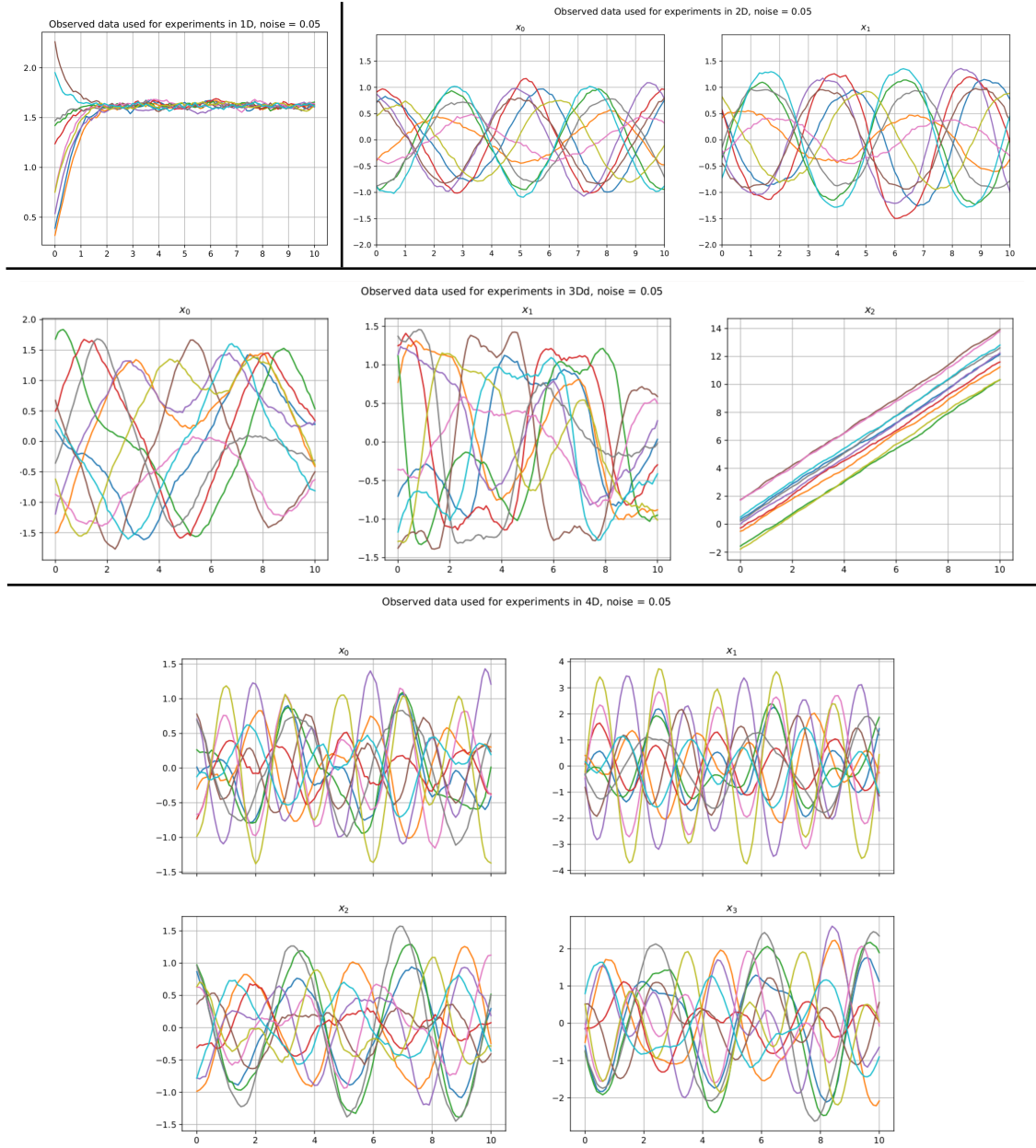


Figure 2.3: From left to right, top to bottom, we plot 10 sample paths for the 1D, 2D, 3D and 4D systems where the initial condition is normally distributed $\mathcal{N}(0,2)$. The initial time $t = 0$ and final time $t = 10$, with 100 time points observed over the interval. The plots are of the observations of the x components versus time.

- For the 4D case, we consider the coupled spring system with two masses:

$$\begin{aligned} dX_{0,t} &= X_{1,t}dt + \gamma_0 dW_{0,t} \\ dX_{1,t} &= -5X_{0,t} - 3.5(X_{0,t} - X_{2,t})dt + \gamma_1 dW_{1,t} \\ dX_{2,t} &= X_{3,t}dt + \gamma_2 dW_{2,t} \\ dX_{3,t} &= -2.33(X_{2,t} - X_{0,t}) - 2X_{2,t}dt + \gamma_3 dW_{3,t} \end{aligned}$$

2.2.1 Experiment 1: Varying Number of Time Series

Here we vary data volume by stepping the number S of time series from $S = 1$ to $S = 10$. Each time series has length $L + 1 = 101$. As plotted in Figure 2.4, the approximation of the true drift function gets better with increasing number of time series S for the 1D, 2D, 3D and 4D systems, respectively. The regression and classification error results, as plotted in Figure 2.5, show that increasing S leads to much better estimates of β . Hard thresholding is useful to improve classification error metrics. As a rule of thumb, the results indicate that at least $S \geq 4$ time series are needed for accurate estimation.

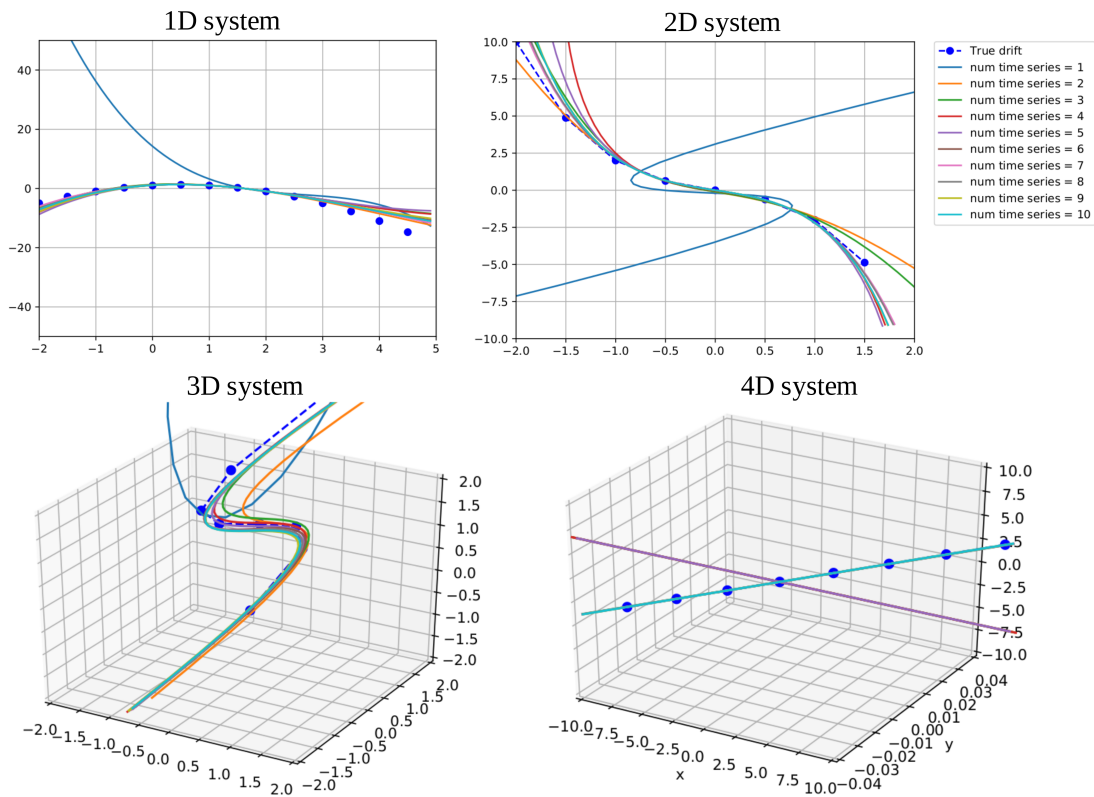


Figure 2.4: As we increase the number S of time series used to learn the drift, the estimated drift more closely approximates the ground truth. From top to bottom, left to right, we have plotted estimated and true drifts for the 1D, 2D, 3D and 4D systems. For the 4D system, we have plotted the X_1, X_2 and X_3 components, keeping the X_0 component constant.

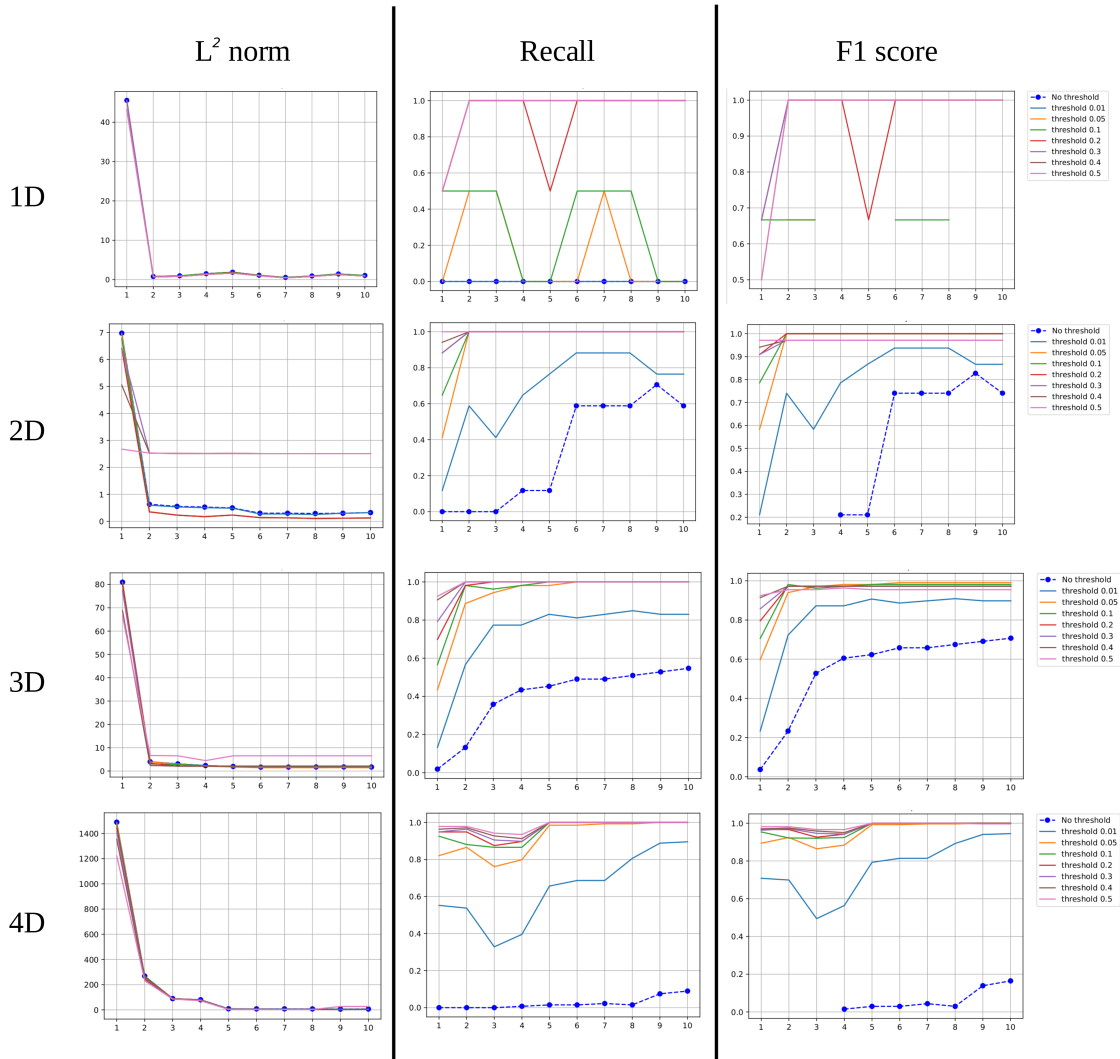


Figure 2.5: As we increase the number S of time series used to learn the drift, the Frobenius norm error between estimated and true drifts—see (2.15)—decreases significantly. In the first column, from top to bottom, we have plotted results for the 1D, 2D, 3D and 4D systems, with varying threshold from 0.5 to 0.01. The recall and F1 score, on the other hand, increases significantly as we increase the number S of time series. The effect of thresholding is more evident for classification error metrics as a threshold of 0.05 shows much better results for recall compared to the no thresholding case. In the second and third column, from top to bottom, we have plotted recall and F1 score for the 1D, 2D, 3D and 4D systems, respectively.

$\frac{S}{\text{System}}$	1	2	3	4	5	6	7	8	9	10
1D	0.84	1.26	1.84	2.18	2.88	3.27	3.90	4.38	4.88	5.25
2D	0.97	1.74	2.54	3.28	3.88	4.58	5.38	6.25	6.77	7.85
3D	1.49	2.75	3.92	5.22	6.46	7.49	8.63	10.46	11.76	13.06
4D	2.65	5.62	7.36	9.76	12.08	14.11	16.22	20.11	22.59	23.14

Table 2.2: Results for average compute time (in seconds) per EM iteration for varying number of time series S . As we increase the number of time series used to learn the drift, the average compute time increases for the 1D, 2D, 3D and 4D systems. The time taken for each EM iteration also increases with the dimensions of the system.

$\frac{S}{\text{System}}$	1	2	3	4	5	6	7	8	9	10
1D	72.91	80.23	79.22	79.62	81.07	80.97	79.43	79.42	78.74	79.33
2D	13.88	17.43	25.57	22.33	18.12	29.46	19.46	4.16	12.15	28.32
3D	2.19	3.22	5.06	4.12	4.34	4.05	4.20	3.61	4.12	4.08
4D	3.51	2.92	12.01	14.50	11.81	11.09	11.86	13.65	14.37	11.08

Table 2.3: Results for average acceptance percentage for Metropolis-Hastings sampler for varying number of time series S . The acceptance rate decreases as the dimensionality and complexity of the system increases. We suspect that the nonlinearity of the 3D damped Duffing system causes the acceptance rate to be lower than that of the linear 4D system.

$\frac{S}{\text{System}}$	1	2	3	4	5	6	7	8	9	10
1D	4	4	3	3	3	3	4	3	3	3
2D	4	3	3	3	3	3	3	43	6	3
3D	501*	3	3	3	3	3	3	6	3	5
4D	26	59	3	3	3	3	2	2	2	2

Table 2.4: Results for number of EM iterations required to converge for varying number of time series S . A threshold of 0.01, 0.05, 0.1 and 0.1 is selected for the 1D, 2D, 3D and 4D systems respectively. If a single time series observation is used for training, then either the relative error in estimated $\hat{\beta}$ can not be reduced below the threshold (as marked by *), or requires considerably higher number of iterations to become less than the threshold.

2.2.2 Experiment 2: Varying Length of Time Series

Here we vary data volume by stepping the length $L + 1$ of the time series from $L + 1 = 11$ to $L + 1 = 101$, keeping the number of time series fixed at $S = 10$. Also note that in this experiment, observation times strictly between the initial and final times are chosen randomly. In Figure 2.4, we have plotted the estimated and true drifts for the 1D, 2D, 3D and 4D systems, respectively. In Figure 2.6, we have plotted the regression (2.15) and classification error (2.18, 2.19) metrics for the systems. Comparing with Experiment 1, we see that randomization of the observation times improves estimation. That is, even with $L + 1 = 11$ data points per time series, we obtain accurate estimates. On the other hand, since the points are chosen randomly, the L^2 error does not decrease monotonically and is dependent on the time points selected.

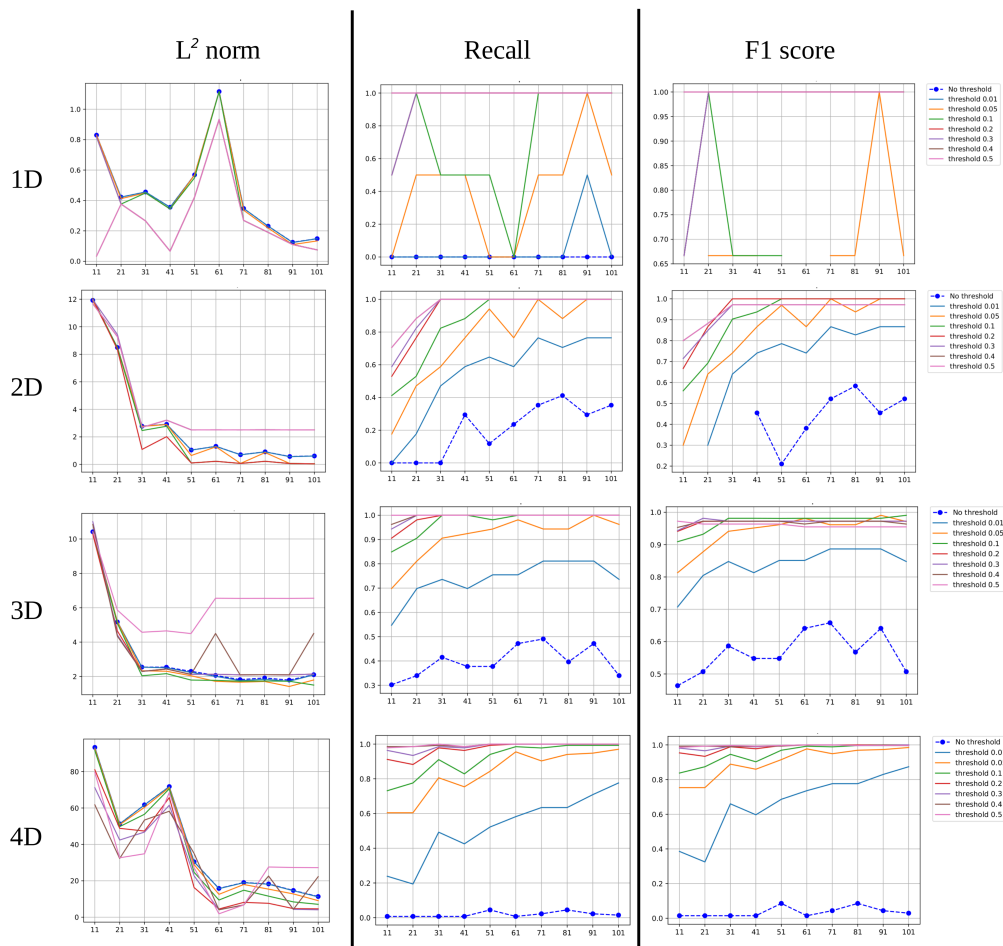


Figure 2.6: As we increase the length L of each time series used for learning, the L^2 regression error between estimated and true drifts—see (2.15)—decreases significantly. Recall and F1 score increase with increasing length, L , of time series. As in Experiment 1, applying a hard threshold of $\lambda = 0.05$ helps in improving the classification errors without increasing the regression errors significantly. From left to right, top to bottom, we have plotted the L^2 error, recall and F1 score for 1D, 2D, 3D and 4D systems, respectively.

$\frac{L}{\text{System}}$	11	21	31	41	51	61	71	81	91	101
1D	0.82	1.50	2.12	2.32	2.84	3.23	3.97	4.45	5.06	5.99
2D	1.04	1.74	2.42	3.14	3.91	4.78	5.61	6.08	6.78	7.68
3D	1.55	2.79	3.98	5.43	6.46	7.89	8.63	10.18	11.80	12.87
4D	2.85	4.80	7.41	10.21	12.31	14.22	16.57	21.23	22.16	23.94

Table 2.5: Results for average compute time (in seconds) per EM iteration for varying length of time series L . As in Experiment 1 (Section 2.2.1), longer time series implies an increase in the time required to compute one step of the EM algorithm. The average compute time also increases with increasing dimensionality of the system.

$\frac{L}{\text{System}}$	11	21	31	41	51	61	71	81	91	101
1D	56.02	64.31	71.96	67.38	74.73	76.42	76.88	83.32	83.36	83.22
2D	4.54	8.96	15.10	23.09	27.72	23.42	27.01	30.38	34.24	35.80
3D	10.21	4.59	4.09	8.04	7.42	10.85	9.65	12.21	13.80	12.33
4D	8.18	5.95	8.35	9.51	16.18	14.16	19.51	18.16	20.76	23.03

Table 2.6: Results for average acceptance percentage for Metropolis-Hastings sampler as we increase the length of time series L . As in Experiment 1 (Section 2.2.1), an increase in L results in higher acceptance percentage for the sampler. The acceptance rate decreases with increase in dimensionality and complexity of the system.

$\frac{L}{\text{System}}$	11	21	31	41	51	61	71	81	91	101
1D	3	3	3	3	3	3	3	3	3	3
2D	4	4	3	3	3	3	3	3	3	3
3D	4	4	3	3	3	4	3	3	5	4
4D	2	2	2	2	2	2	2	2	2	2

Table 2.7: Results for number of EM iterations required to converge for varying length of time series L . As in Experiment 1, a threshold of 0.01, 0.05, 0.1 and 0.1 is set for 1D, 2D, 3D and 4D systems respectively. Unlike Experiment 1 (Section 2.2.1), the number of iterations does not change considerably with the length of time series.

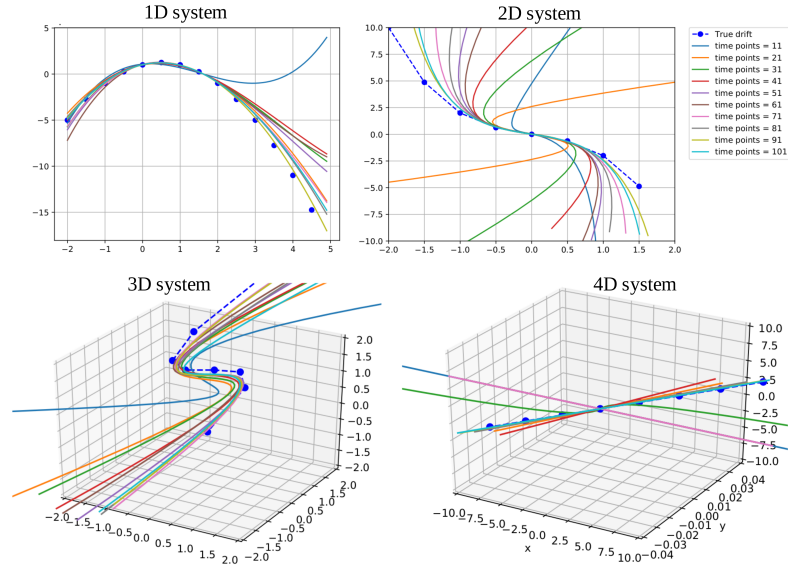


Figure 2.7: We plot true and estimated drifts for the 1D, 2D, 3D and 4D systems as a function of increasing time series length L . The last three components of the vector field for the 4D system are plotted as in Figure 2.4. The results show that randomization of observation times compensates for a small value of L , enabling accurate estimation.

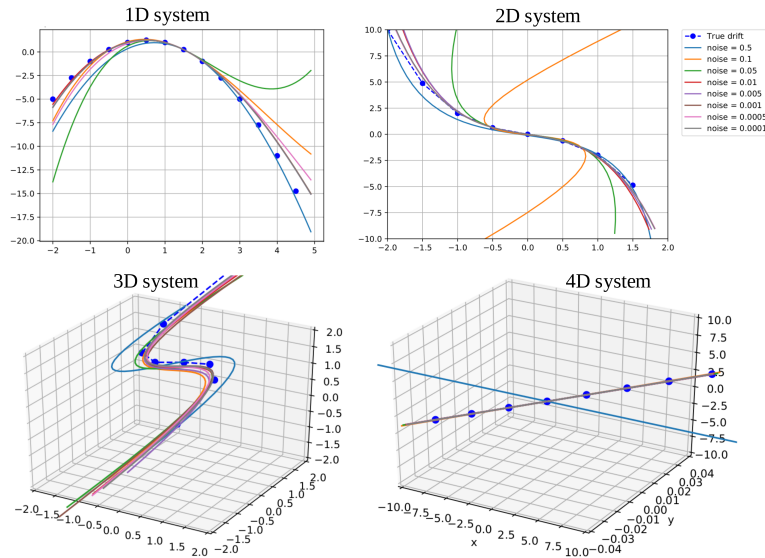


Figure 2.8: We plot true and estimated drifts for the 1D, 2D, 3D and 4D systems as a function of varying noise strength, γ . As in Figure 2.7, the last three components of the vector field for the 4D system are plotted. The results show that estimation is accurate even for large noise strength values and the error goes to zero with the noise strength.

2.2.3 Experiment 3: Varying Noise Strength

Here we vary the noise strength γ , stepping from 0.5 to 0.0001 while keeping other parameters constant. Specifically, we take $S = 10$ time series each of length $L + 1 = 101$. In Figure 2.9, we have plotted regression and classification errors for all systems. As in the previous experiments, the estimated and true drift functions are close—see Figure 2.8. There is an increase in the L^2 error when the maximum noise strength is close to the drift coefficients. For all systems, as the noise strength goes to zero, the error drops close to zero. The effect of thresholding is more pronounced in the 3D system. Many non-zero components are present in the 3D system, thus excessive thresholding results in a drastic increase of the L^2 error. As in the previous experiments, $\lambda = 0.05$, is a reasonable threshold which greatly improves classification without drastically increasing the L^2 error.

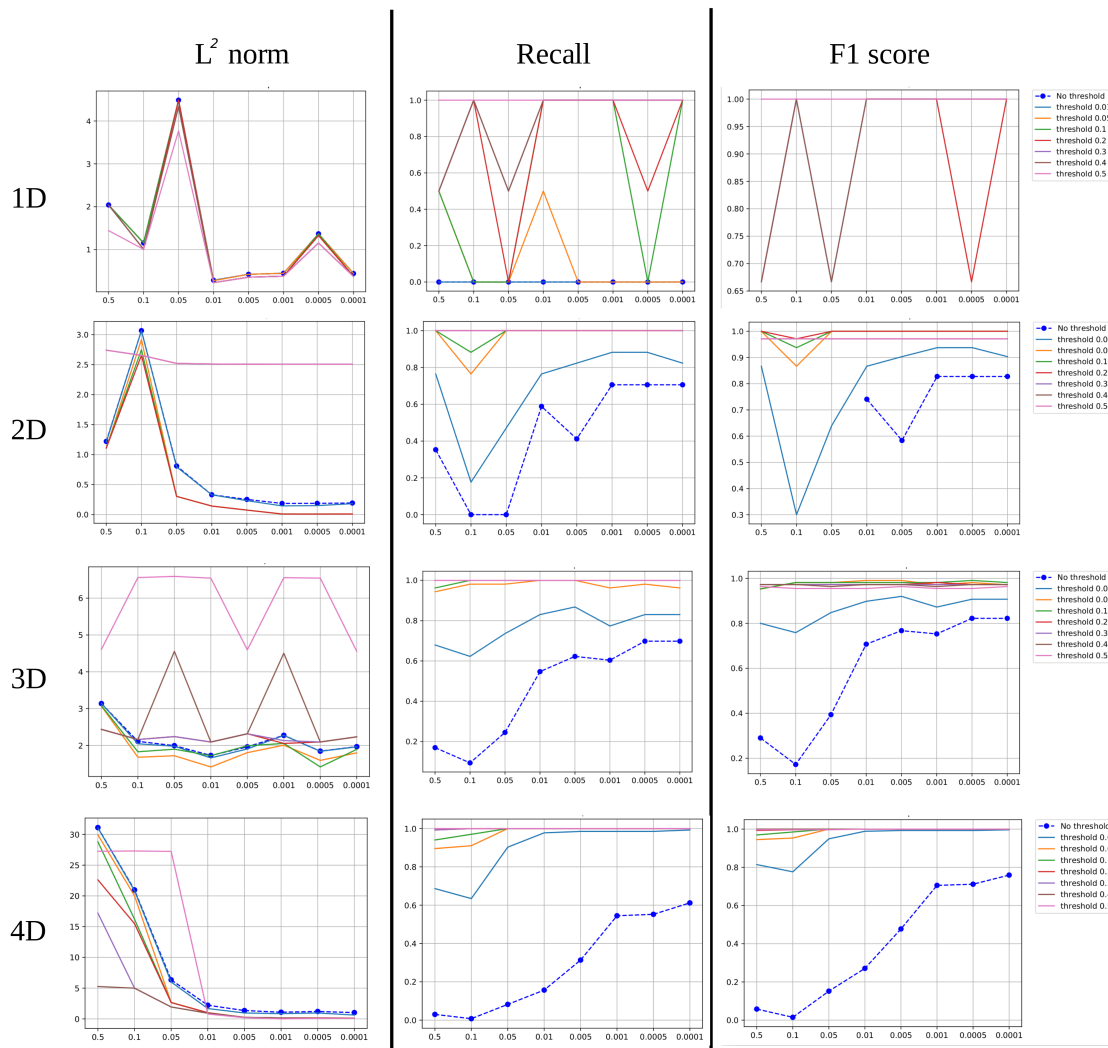


Figure 2.9: Varying the noise strength in simulated data alters the quality of estimated drift coefficients, quantified using the L^2 error (2.15), recall (2.18) and F1 score (2.19).

$\frac{\gamma}{\text{System}}$	0.5	0.1	0.05	0.01	0.005	0.001	0.0005	0.0001
1D	5.57	5.37	5.55	5.66	5.74	5.28	6.43	6.04
2D	7.78	7.57	7.65	8.05	7.86	7.41	7.51	7.66
3D	12.84	12.58	12.71	12.38	13.51	12.81	12.21	12.11
4D	24.07	23.86	24.12	23.62	23.16	24.19	23.58	26.39

Table 2.8: Results for average compute time (in seconds) per EM iteration for varying levels of noise strength γ . The time required for EM iterations does not vary with varying noise strength, but it increases as the complexity of the system increases.

$\frac{\gamma}{\text{System}}$	0.5	0.1	0.05	0.01	0.005	0.001	0.0005	0.0001
1D	87.91	86.28	86.72	79.42	76.49	70.90	67.02	65.06
2D	38.18	46.12	31.27	24.21	2.49	7.76	2.47	2.29
3D	28.99	34.78	17.06	3.74	3.06	2.34	2.42	2.22
4D	15.39	27.22	18.51	12.28	4.08	2.46	2.24	2.36

Table 2.9: Results for average acceptance percentage for Metropolis-Hasting sampler for varying noise strength γ . The acceptance percentage decreases with decrease in γ . As the system becomes more deterministic, it becomes harder to generate Brownian bridge sample paths between the observed data points. The acceptance percentage also decreases with increased complexity of the system.

$\frac{\gamma}{\text{System}}$	0.5	0.1	0.05	0.01	0.005	0.001	0.0005	0.0001
1D	15	4	4	4	3	3	3	2
2D	5	6	4	3	501*	2	2	2
3D	501*	5	7	3	501*	2	2	2
4D	3	3	3	2	2	2	2	2

Table 2.10: Results for number of EM iterations required to converge. As in Experiment 1 and 2 (Section 2.2.1, 2.2.2) a threshold of 0.01, 0.05, 0.1 and 0.1 are set for the 1D, 2D, 3D and 4D systems respectively. The number of iterations decrease, in general, with a decrease in the noise strength. In a few cases the relative error between $\hat{\beta}$ iterates does not reduce below the specified threshold (as marked by *). It is important to note here that the relative error does not reduce below the threshold, but on inspecting the estimated $\hat{\beta}$ parameter, the estimated value is close to the true value and the relative error is still small.

2.2.4 Experiment 4: Varying Data Augmentation

We start with $S = 1$ time series with $L + 1 = 51$ points each. Here we vary the number of interleaved diffusion bridge samples: $F = 1, \dots, 10$. For $F = 1$, no diffusion bridge is created; the likelihood is computed by applying the Gaussian transition density directly to the observed data. The results, plotted in Figures 2.10 and 2.11, show that increased data augmentation dramatically improves the quality of estimated drifts.

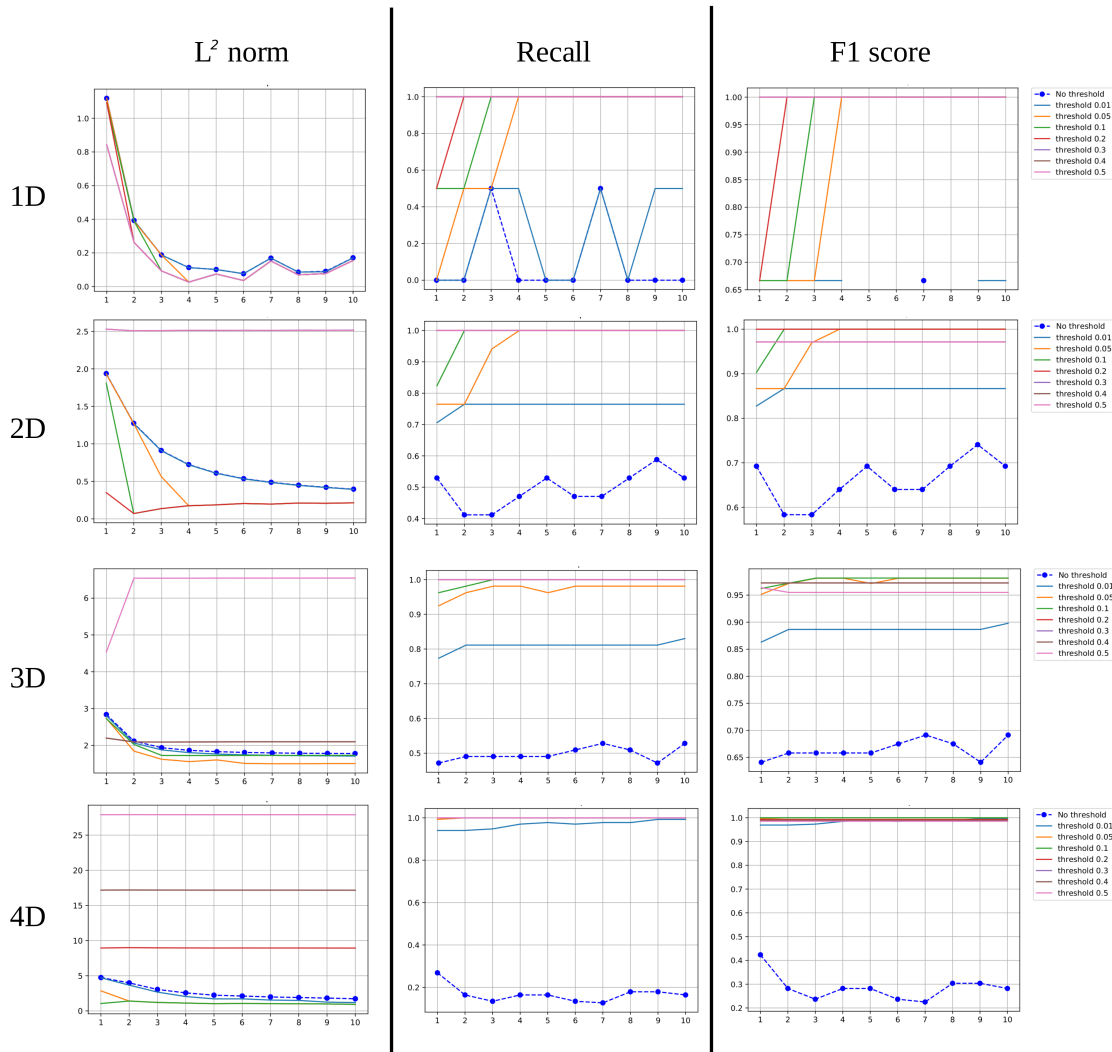


Figure 2.10: As we increase the length F of the diffusion bridge interleaving observed data points, the quality of estimated drifts improves considerably. From left to right, top to bottom, we have plotted the Frobenius errors (2.15), recall (2.18) and F1 score (2.19) between true and estimated coefficients, for the 1D, 2D, 3D and 4D systems.

We have not plotted results for the other scarce data regime where we have $S = 10$ time series with $L = 11$ points each. In this regime, data augmentation enables highly accurate estimation for all the systems as well.

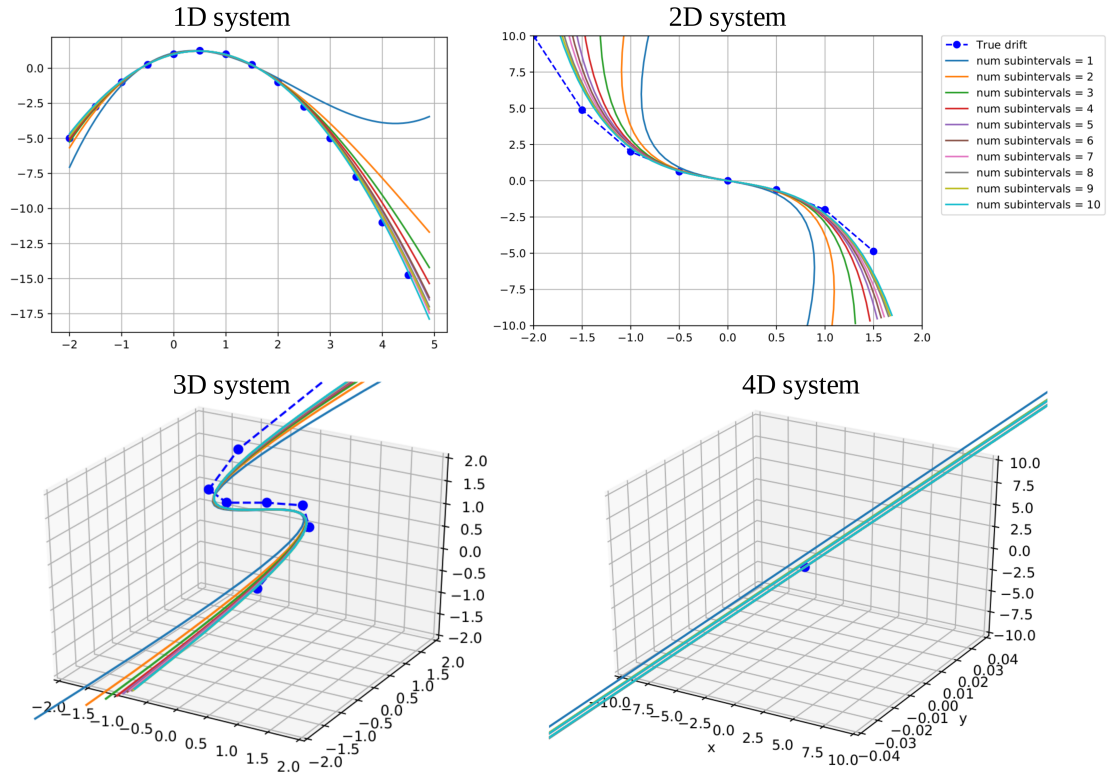


Figure 2.11: As in the previous experiments, the estimated drift functions lie close to the true drift function. We find that introducing diffusion bridge samples helps estimating the drift function more accurately.

$\frac{F}{\text{System}}$	1	2	3	4	5	6	7	8	9	10
1D	0.59	0.54	0.54	0.54	1.00	0.57	0.58	0.57	0.85	0.55
2D	0.65	0.57	0.58	0.57	0.57	0.57	0.57	0.62	0.56	0.57
3D	6.51	9.58	6.29	6.55	6.46	6.82	6.47	6.36	6.69	6.59
4D	24.08	24.34	23.94	23.98	24.93	25.65	23.99	23.17	25.64	24.54

Table 2.11: Results for average compute time (in seconds) per EM iteration for varying amount of data augmentation. As the Brownian bridge is created explicitly using the discretized version of (2.9), increasing the amount of data augmentation does not cause significant increase in the compute time. The time required to compute each EM iteration increases with an increase in the dimensionality of the system.

$\frac{F}{\text{System}}$	1	2	3	4	5	6	7	8	9	10
1D	100	75.04	67.08	61.78	61.23	58.71	55.22	53.57	52.58	49.52
2D	100	13.17	9.04	6.54	4.75	4.35	4.11	4.94	2.87	4.02
3D	100	6.07	3.20	2.82	2.74	2.54	2.48	2.27	2.51	2.41
4D	100	25.69	19.22	13.69	11.63	7.81	6.88	5.83	4.10	4.04

Table 2.12: Results for average acceptance rate for Metropolis-Hastings sampler for varying amount of data augmentation, F . For $F = 1$, no diffusion bridge has been created and thus the acceptance probability is 1. The algorithm in this case reduces to solving a least squares using only the observed time series. As we increase data augmentation, the acceptance probability decreases as it becomes more difficult to create a bridge between the observed values. The acceptance probability also decreases with an increase in the dimensionality and complexity of the system.

$\frac{F}{\text{System}}$	1	2	3	4	5	6	7	8	9	10
1D	2	3	3	3	3	3	3	3	3	3
2D	2	8	5	7	6	8	8	4	9	6
3D	2	3	3	3	3	3	9	3	3	3
4D	2	2	2	2	2	2	2	2	2	2

Table 2.13: Results for number of EM iterations required to converge. As in the previous experiments (Section 2.2.1, 2.2.2, 2.2.3) we consider a threshold of 0.01, 0.05, 0.1 and 0.1 for the 1D, 2D, 3D and 4D systems respectively. The number of EM iterations does not vary significantly with varying amount of data augmentation F .

2.3 Discussion

We have developed an EM algorithm for estimation of drift functions and diffusion matrices for SDE. We have demonstrated the conditions under which the algorithm succeeds in estimating SDE. Specifically, our tests show that with enough data volume and data augmentation, the EM algorithm produces highly accurate results. In future work, we seek to further test our method on high-dimensional, nonlinear problems, problems with non-constant diffusion matrices, and real experimental data. As we move to higher-dimensional problems, we will also explore regularization and model selection techniques.

Chapter 3

Parameter Inference for SDEs via Density Tracking by Quadrature

3.1 Statistical Model

The fundamental model considered in this section is

$$dX_t = f(X_t; \boldsymbol{\theta}) dt + g(X_t; \boldsymbol{\theta}) dW_t, \quad X_{t_0} = x_0 \quad (3.1)$$

where $X = (X_t)_{t \geq 0}$ is a scalar stochastic process, $\boldsymbol{\theta} \in \mathbb{R}^N$ is a vector of parameters, and $W = (W_t)_{t \geq 0}$ is standard Brownian motion. Here f and g are referred to, respectively, as the drift and diffusion functions. They are assumed to be known in parametric form. Suppose we have collected data $\mathbf{x} = (x_0, x_1, \dots, x_M)$ as shown in Figure 3.1. For the current model, we assume that all observations are complete, i.e. there are no missing observations. Based on this data, we would like to infer $\boldsymbol{\theta}$. The parameter space for inference in this case is \mathbb{R}^N .

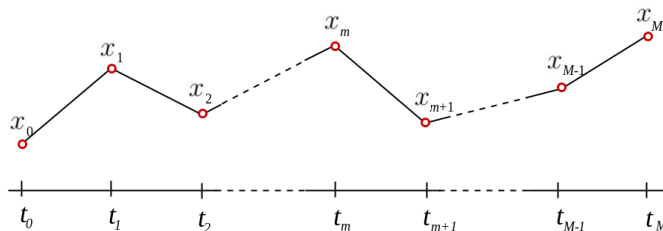


Figure 3.1: Illustration of a sample path with discrete model observations x_0, \dots, x_M at times t_0, \dots, t_M in red. Our goal is to infer the parameters of the model using these observations.

One way to carry out this inference is through numerical maximization of the likelihood function $p(\mathbf{x}|\boldsymbol{\theta})$. Alternatively, we can view this problem as a Bayesian inference problem, and our goal is to sample from the posterior. The posterior density of the parameter vector given the observations is $p(\boldsymbol{\theta}|\mathbf{x}) \propto p(\mathbf{x}|\boldsymbol{\theta}) p(\boldsymbol{\theta})$, where $p(\mathbf{x}|\boldsymbol{\theta})$ is the likelihood and $p(\boldsymbol{\theta})$ is the prior. This shows the need for a method to compute the likelihood function, which we derive in Section 3.2.

3.2 DTQ method

Here we describe how to compute the likelihood $p(\mathbf{x}|\boldsymbol{\theta})$ under the fundamental model (3.1). Our first step is to apply a Markov property satisfied by (3.1): the random variable $X_{t_{m+1}}$, given X_{t_m} , is conditionally independent of all random variables $X_{t_{m-k}}$ for $k \geq 1$. With this property, the likelihood factors:

$$L(\boldsymbol{\theta}) = p(\mathbf{x}|\boldsymbol{\theta}) = p(x_0|\boldsymbol{\theta}) \prod_{m=0}^{M-1} p(x_{m+1}|x_m;\boldsymbol{\theta}) \quad (3.2)$$

Each term in the product can be interpreted as follows: we start the SDE (3.1) with the initial condition $X_{t_m} = x_m$ and fixed parameter vector $\boldsymbol{\theta}$. We then solve for the probability density function (pdf) of $X_{t_{m+1}}$, and evaluate that pdf at x_{m+1} . By following these steps, we have calculated $p(x_{m+1}|x_m;\boldsymbol{\theta})$.

We now outline a convergent method to compute the aforementioned pdf (Bhat and Madushani (2016)). 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 (3.1) via the Euler-Maruyama discretization. When describing this discretization, we specialize to the case where we seek $p(x_{m+1}|x_m,\boldsymbol{\theta})$. That is, we take $\{\tau_i\}_{i=0}^n$ to be a temporal grid such that $\tau_0 = t_m$, $\tau_n = t_{m+1}$, and $h = (t_{m+1} - t_m)/n > 0$. Then, for $i = 1, 2, \dots, n$, we have $\tau_i = t_m + ih$. On this temporal grid, the Euler-Maruyama discretization of (3.1) is:

$$\tilde{x}_{i+1} = \tilde{x}_i + f(\tilde{x}_i;\boldsymbol{\theta})h + g(\tilde{x}_i;\boldsymbol{\theta})\sqrt{h}Z_{i+1} \quad (3.3)$$

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

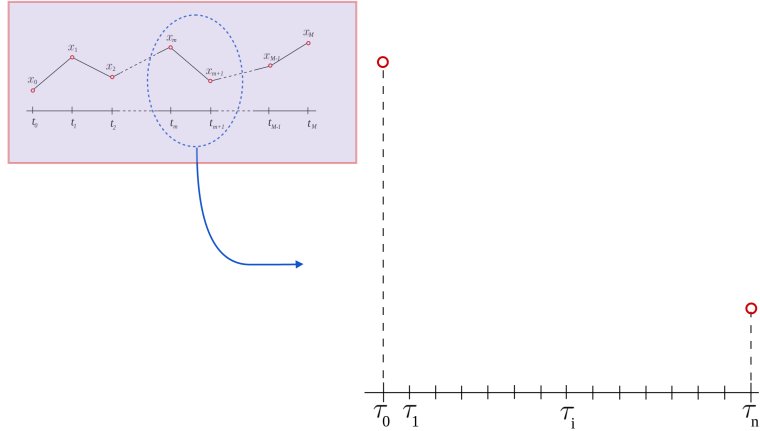


Figure 3.2: Illustration of the temporal grid between a time interval (t_m, t_{m+1}) . We take $\{\tau_i\}_{i=0}^n$ to be the temporal grid such that $\tau_0 = t_m$, $\tau_n = t_{m+1}$, and $h = (t_{m+1} - t_m)/n > 0$. The observed value is constrained by the data point, thus $x_m = X_{\tau_0} = X_{t_m}$

The next step in deriving the DTQ method is to write down the Chapman-Kolmogorov equation corresponding to (3.3). Let $p(\tilde{x}_i)$ denote the pdf of \tilde{x}_i given the initial condition $\tilde{x}_0 = x_m$. Then purely based on the laws of probability we can write:

$$p(\tilde{x}_{i+1}) = \int_{\tilde{x}_i} p(\tilde{x}_{i+1} | \tilde{x}_i) p(\tilde{x}_i) d\tilde{x}_i. \quad (3.4)$$

Inspecting (3.3), we see that conditional on \tilde{x}_i , the pdf of \tilde{x}_{i+1} is Gaussian with mean $\tilde{x}_i + f(\tilde{x}_i; \boldsymbol{\theta})h$ and variance $g^2(\tilde{x}_i; \boldsymbol{\theta})h$. Let us define the function

$$G_{\boldsymbol{\theta}}^h(a, b) = \frac{1}{\sqrt{2\pi g^2(b; \boldsymbol{\theta})h}} \exp\left(-\frac{(a - b - f(b; \boldsymbol{\theta})h)^2}{2g^2(b; \boldsymbol{\theta})h}\right). \quad (3.5)$$

Then (3.4) becomes

$$p(\tilde{x}_{i+1}) = \int_{\tilde{x}_i} G_{\boldsymbol{\theta}}^h(\tilde{x}_{i+1}, \tilde{x}_i) p(\tilde{x}_i) d\tilde{x}_i. \quad (3.6)$$

The last step is to spatially discretize the pdfs 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{q}_i such that the j -th component of \mathbf{q}_i is $q_i^j = p(\tilde{x}_i = z_j)$. We then apply the trapezoidal rule to (3.6), resulting in

$$q_{i+1}^{j'} = \sum_j k G_{\boldsymbol{\theta}}^h(z_{j'}, z_j) q_i^j. \quad (3.7)$$

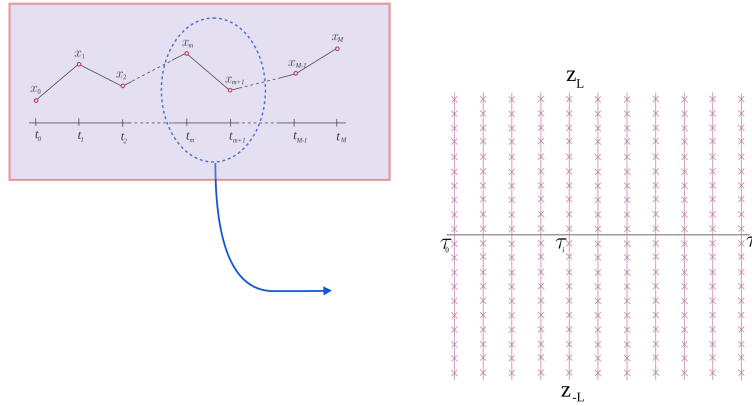


Figure 3.3: Illustration of the truncated spatial grid, $\{z_j\}_{j=-L}^L$, for each discretized time point in the interval (t_m, t_{m+1}) . The estimated probability at any given point, $p(\tilde{x}_i)$, is represented by a vector, \mathbf{q}_i such that the j -th component of \mathbf{q}_i is $q_i^j = p(\tilde{x}_i = z_j)$.

We think of $kG_{\boldsymbol{\theta}}^h(z_{j'}, z_j)$ as the (j', j) element of a matrix \mathbf{A} . In this way, the above formula reduces to repeated matrix-vector multiplication.

$$\mathbf{q}_{i+1} = \mathbf{A} \mathbf{q}_i. \quad (3.8)$$

In practice, we evaluate these sums on a finite subset of \mathbb{Z} . Therefore, we truncate the

spatial domain. Let $L > 0$ be an integer. We take both $j, j' \in \{-L, \dots, 0, \dots, L\}$; this means that each vector \mathbf{q}_i has dimension $2L + 1$.

3.2.1 Likelihood Computation

Now that we have a method to compute the required intermediate pdfs, here is how we compute $p(x_{m+1} | x_m, \boldsymbol{\theta})$:

1. We know that $\tilde{x}_0 = x_m$, so we define $p(\tilde{x}_0) = \delta(\tilde{x}_0 - x_m)$. Then keeping x_m fixed and evaluating the right-hand side with \tilde{x}_1 equal to each point in the spatial grid $\{z_j\}_{j=-L}^L$, we obtain \mathbf{q}_1 with dimension $2L + 1$:

$$\mathbf{q}_1 = p(\tilde{x}_1) = G_{\boldsymbol{\theta}}^h(\tilde{x}_1 = \{z_j\}_{j=-L}^L, x_m). \quad (3.9)$$

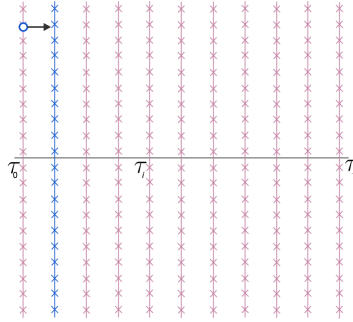


Figure 3.4: Illustration of the first step of DTQ, propagation of probability from a Dirac-delta function at τ_0 to a vector over the spatial grid at τ_1 .

2. With \mathbf{q}_1 in hand, multiplication by the matrix \mathbf{A} corresponds to stepping forward in time by h , i.e.,

$$\mathbf{q}_2 = \mathbf{A}\mathbf{q}_1.$$

We iterate (3.7) a total of $n - 2$ times to step forward to $i = n - 1$. This takes us to

$$\mathbf{q}_{n-1} = \mathbf{A}^{n-2} \mathbf{q}_1. \quad (3.10)$$

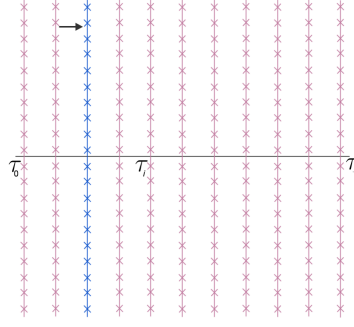


Figure 3.5: Illustration of the forward propagation of probabilities as matrix-vector multiplications, $\mathbf{q}_{n-1} = \mathbf{A}^{n-2} \mathbf{q}_1$.

3. Finally, noting that x_{m+1} is a known data point, let us define a $2L + 1$ dimensional vector Γ_{n-1} by keeping x_{m+1} fixed and evaluating on the spatial grid $\{z_j\}_{j=-L}^{j=L}$ for each point \tilde{x}_{n-1} :

$$\Gamma_{n-1} = kG_{\boldsymbol{\theta}}^h(x_{m+1}, \tilde{x}_{n-1} = \{z_j\}_{j=-L}^{j=L}). \quad (3.11)$$

Then we have that the density of \tilde{x}_n evaluated at the data point x_{m+1} is

$$p_{\tilde{x}_{m+1}}(x_{m+1} | \tilde{X}_m = x_m; \boldsymbol{\theta}) = \Gamma_{n-1}^\top A^{n-2} \mathbf{q}_1. \quad (3.12)$$

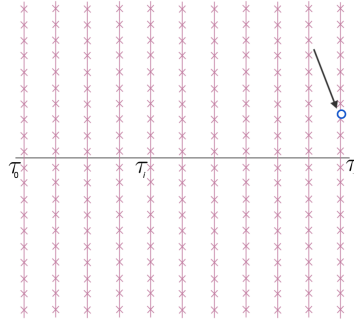


Figure 3.6: Illustration of the last step of the DTQ algorithm over one time interval. Since the data point at final time τ_n is an observed value, this is a vector to Dirac-delta conversion.

This computation is done for individual intervals (t_m, t_{m+1}) . As shown above, the vectors \mathbf{q}_1 and Γ_{n-1} depend on the spatial grid and the observation values x_m and x_{m+1} , respectively. They do not depend on the temporal grid. Thus for the general case, given a spatial grid for the DTQ method, we have the first vector $\mathbf{q}_m := \mathbf{q}_1$ and the final vector $\Gamma_{m+1} := \Gamma_{n-1}$.

We insert this computation into (3.2) to obtain the expression for the likelihood function using the DTQ method,

$$L(\boldsymbol{\theta}) \approx p(x_0 | \boldsymbol{\theta}) \prod_{m=0}^{M-1} [\boldsymbol{\Gamma}_{m+1}]^\top A^{n-2} \mathbf{q}_m. \quad (3.13)$$

The log likelihood used for numerical optimization procedures is

$$\mathcal{L}(\boldsymbol{\theta}) = \log L(\boldsymbol{\theta}) \approx \log p(x_0 | \boldsymbol{\theta}) + \sum_{m=0}^{M-1} [\log \boldsymbol{\Gamma}_{m+1}]^\top A^{n-2} \mathbf{q}_m. \quad (3.14)$$

3.2.2 Gradient Computation

Next, we compute the gradient of the log likelihood with respect to $\boldsymbol{\theta}$. This gradient is an important ingredient for numerical optimization procedures. The complete derivation of the gradients is similar to the derivation of the likelihood function and is given in the Appendix (A.1). We start with the component-wise gradient of the log likelihood $\mathcal{L}(\boldsymbol{\theta})$ given by (3.2). We factor the likelihood and evaluate the gradients of the transition density on the discretized spatial grid iteratively.

$$\begin{aligned} \frac{\partial}{\partial \boldsymbol{\theta}_\ell} \mathcal{L}(\boldsymbol{\theta}) &= \frac{\partial}{\partial \boldsymbol{\theta}_\ell} \log \left[p(x_0 | \boldsymbol{\theta}) \prod_{m=0}^{M-1} p(x_{m+1} | x_m; \boldsymbol{\theta}) \right] \\ &= \frac{\partial}{\partial \boldsymbol{\theta}_\ell} \log p(x_0 | \boldsymbol{\theta}) + \sum_{m=0}^{M-1} \frac{\partial}{\partial \boldsymbol{\theta}_\ell} \log p(x_{m+1} | x_m; \boldsymbol{\theta}) \\ &\approx \frac{\partial}{\partial \boldsymbol{\theta}_\ell} \log p(x_0 | \boldsymbol{\theta}) + \sum_{m=0}^{M-1} \frac{1}{p(x_{m+1} | x_m; \boldsymbol{\theta})} \frac{\partial}{\partial \boldsymbol{\theta}_\ell} p(x_{m+1} | x_m; \boldsymbol{\theta}). \end{aligned} \quad (3.15)$$

Here, we give the algorithm used to compute $\nabla p(x_{m+1} | x_m; \boldsymbol{\theta})$, the gradient of the transition density:

1. We begin with the initial condition, $p(\tilde{x}_0) = \delta(\tilde{x}_0 - x_m)$ and get the ℓ -th component of the gradient

$$\mathbf{q}_{1,\ell} = \frac{\partial \mathbf{q}_1}{\partial \boldsymbol{\theta}_\ell} = \frac{\partial}{\partial \boldsymbol{\theta}_\ell} G_{\boldsymbol{\theta}}^h(\tilde{x}_1 = \{z_j\}_{j=-L}^{j=L}, x_m). \quad (3.16)$$

2. We then define the $(z_{j'}, z_j)$ -th element of the matrix \mathbf{A}_ℓ on the spatial grid with $j', j \in \{-L, \dots, 0, \dots, L\}$ by

$$\mathbf{A}_\ell = \frac{\partial \mathbf{A}}{\partial \boldsymbol{\theta}_\ell} = k \frac{\partial}{\partial \boldsymbol{\theta}_\ell} G_{\boldsymbol{\theta}}^h(z_{j'}, z_j). \quad (3.17)$$

Using (3.16) and (3.17), we iteratively define the time evolution equation of the gradient as

$$\mathbf{q}_{i+1,\ell} = \mathbf{A}_\ell \mathbf{q}_i + \mathbf{A} \mathbf{q}_{i,\ell}. \quad (3.18)$$

3. Similarly, we define the gradient of the final time point, $\boldsymbol{\Gamma}_{n-1}$ vector, as

$$\boldsymbol{\Gamma}_{n-1,\ell} = \frac{\partial \boldsymbol{\Gamma}_{n-1}}{\partial \boldsymbol{\theta}_\ell} = k \frac{\partial}{\partial \boldsymbol{\theta}_\ell} G_{\boldsymbol{\theta}}^h(x_{m+1}, \tilde{x}_{n-1} = \{z_j\}_{j=-L}^{j=L}). \quad (3.19)$$

We finish with

$$\frac{\partial}{\partial \boldsymbol{\theta}_\ell} p_{\tilde{X}_{m+1}}(x_{m+1} | \tilde{X}_m = x_m; \boldsymbol{\theta}) \approx [\boldsymbol{\Gamma}_{n-1, \ell}]^\top \mathbf{q}_{n-1} + [\boldsymbol{\Gamma}_{n-1}]^\top \mathbf{q}_{n-1, \ell} \quad (3.20)$$

where both \mathbf{q}_i and $\mathbf{q}_{i, \ell}$ are iteratively computed by matrix-vector multiplications using (3.8) and (3.18). As stated previously, this computation is done for individual intervals (t_m, t_{m+1}) . The vectors $\mathbf{q}_1, \mathbf{q}_{1, \ell}$ only depend on the initial data point x_m , and the vectors $\boldsymbol{\Gamma}_{n-1}, \boldsymbol{\Gamma}_{n-1, \ell}$ depend on the final time point x_{m+1} . In the general computation, we thus define $\mathbf{q}_m := \mathbf{q}_1, \boldsymbol{\Gamma}_{m+1} := \boldsymbol{\Gamma}_{n-1}$ and $\boldsymbol{\Gamma}_{m+1, \ell} := \boldsymbol{\Gamma}_{n-1, \ell}$. Using this together with (3.18) in (3.15), we obtain

$$\frac{\partial}{\partial \boldsymbol{\theta}_\ell} \mathcal{L}(\boldsymbol{\theta}) \approx \frac{\partial}{\partial \boldsymbol{\theta}_\ell} \log p(x_0 | \boldsymbol{\theta}) + \sum_{m=0}^{M-1} \frac{1}{[\boldsymbol{\Gamma}_{m+1}]^\top \mathbf{A}^{n-2} \mathbf{q}_m} [\boldsymbol{\Gamma}_{m+1, \ell}]^\top \mathbf{q}_{n-1} + [\boldsymbol{\Gamma}_{m+1}]^\top \mathbf{q}_{n-1, \ell}. \quad (3.21)$$

The full derivation of the DTQ method is in Appendix A. The derivations of the likelihood and its gradient are in Appendixes A.1 and A.2, respectively. We also provide a derivation of the DTQ likelihood and gradient with multiple sample paths in Appendixes A.3.1 and A.3.2, respectively. We also provide a more efficient adjoint based gradient computation in Appendix A.4. Above, we only included the crucial calculations required to implement the DTQ method, for the sake of clarity.

3.3 Two-dimensional Coupled SDE

Here we deal with coupled SDE of the general form

$$dX_{1,t} = f_1(\mathbf{X}_t; \boldsymbol{\theta}) dt + g_1(\mathbf{X}_t; \boldsymbol{\theta}) dW_{1,t} \quad (3.22a)$$

$$dX_{2,t} = f_2(\mathbf{X}_t; \boldsymbol{\theta}) dt + g_2(\mathbf{X}_t; \boldsymbol{\theta}) dW_{2,t}, \quad (3.22b)$$

where $\mathbf{X}_t = (X_{1,t}, X_{2,t})$ is a two-dimensional stochastic process. $W_{1,t}$ and $W_{2,t}$ denote two independent Wiener processes with $W_{1,0} = W_{2,0} = \mathbf{0}$ almost surely. For $a = 1, 2$, we refer to f_a and g_a as, respectively, drift and diffusion functions. Both drift and diffusion functions may depend on a parameter vector $\boldsymbol{\theta} \in \mathbb{R}^N$.

As in the fundamental model (3.1), our goal is to infer the parameter vector $\boldsymbol{\theta}$ from direct observations of \mathbf{X}_t . Suppose that at a sequence of times $0 = t_0 < t_1 < \dots < t_M = T$, we have observations $\mathbf{x} = \mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_M$. Here $\mathbf{x}_m = (x_{1,m}, x_{2,m})$ is a sample of \mathbf{X}_{t_m} .

For this case, we consider a real-world system. We create a pursuit model to infer a stochastic model of a chaser's pursuit of a runner. This is motivated by "fast-break" situations in a basketball game, where an offensive player races towards the basket in an attempt to score before the defensive team has time to set up their defense. Let the *runner* be the player (on offense) who has the ball and is running toward the basket. Let the *chaser* be the player (on defense) who is trying to prevent the runner from scoring. Let the current spatial coordinates of the runner and chaser be, respectively, $(x^r(t), y^r(t))$ and $(x^c(t), y^c(t))$.



Figure 3.7: Diagram illustrating motion of runner and chaser. At any instant of time, the chaser's velocity vector points toward the runner's current position.

Consider the diagram in Figure 3.7. Since the chaser is moving towards the runner, the velocity vector of the chaser points toward the runner's current position. Let $\vec{\phi} = (x^r - x^c, y^r - y^c)$. Then the unit vector that points toward the runner from the chaser is $\phi/\|\phi\|$. The velocity of the chaser, (\dot{x}^c, \dot{y}^c) , can thus be given as

$$(\dot{x}^c, \dot{y}^c) = \gamma(t) \frac{\phi}{\|\phi\|}, \quad (3.23)$$

where $\gamma(t) = \|(\dot{x}^c, \dot{y}^c)\|$, the instantaneous speed of the chaser. Note that (3.23) is a coupled system of nonlinear ordinary differential equations known as the pursuit model—classically, one assumes that $\gamma(t)$ and $(x^r(t), y^r(t))$ are given, in which case one typically solves an initial-value problem for $(x^c(t), y^c(t))$. To generalize the classical model to the real data context considered here, we multiply both sides of (3.23) by dt and then add noise to each component:

$$d(x^c, y^c) = \gamma(t) \frac{\vec{\phi}}{\|\vec{\phi}\|} dt + (v_1 dW_{1,t}, v_2 dW_{2,t}). \quad (3.24)$$

For the stochastic pursuit model (3.24), we take $\mathbf{X}_t = (x^c(t), y^c(t))$. We treat $\gamma(t)$ as piecewise constant. Each constant value of $\gamma(t)$ is one component of the parameter vector θ ; the final two values of this vector are v_1 and v_2 . Finally, if we treat $(x^r(t), y^r(t))$ as given, then we can identify the time-dependent drift functions f_1 and f_2 as the two components of $\gamma(t)\phi/\|\phi\|$. Given time-discrete observations of (x^c, y^c) and (x^r, y^r) , how do we infer $\gamma(t)$ along with v_1 and v_2 ? We now provide a method to answer this question.

Following the method mentioned in Section 3.2.1, we discretize the model using the Euler-Maruyama discretization as

$$\tilde{x}_{1,i+1} = \tilde{x}_{1,i} + f_1(t_i, \tilde{\mathbf{x}}_i; \theta) h + g_1(t_i, \tilde{\mathbf{x}}_i; \theta) \sqrt{h} Z_{1,i+1} \quad (3.25a)$$

$$\tilde{x}_{2,i+1} = \tilde{x}_{2,i} + f_2(t_i, \tilde{\mathbf{x}}_i; \theta) h + g_2(t_i, \tilde{\mathbf{x}}_i; \theta) \sqrt{h} Z_{2,i+1}. \quad (3.25b)$$

The $Z_{1,i}$ and $Z_{2,i}$ are independent and identically distributed random variables, normally distributed with mean 0 and variance 1, i.e., $Z_i \sim \mathcal{N}(0, 1)$. If we let $\tilde{p}_i(x_1, x_2 | \theta)$ denote the joint probability density function of $X_{1,i}$ and $X_{2,i}$ given θ , then the Chapman-Kolmogorov

equation associated with (3.25) is

$$\tilde{p}_{i+1}(x_1, x_2 | \boldsymbol{\theta}) = \int_{\mathbf{y} \in \mathbb{R}^2} G_{\boldsymbol{\theta}}^h(x_1, x_2, y_1, y_2, t_i; \boldsymbol{\theta}) \tilde{p}_i(y_1, y_2 | \boldsymbol{\theta}) d\mathbf{y}, \quad (3.26)$$

where

$$\begin{aligned} G_{\boldsymbol{\theta}}^h(x_1, x_2, y_1, y_2, t_i; \boldsymbol{\theta}) &= \tilde{p}_{i+1|i}(x_1, x_2 | y_1, y_2; \boldsymbol{\theta}) \\ &= \frac{1}{\sqrt{2\pi\sigma_1^2}} \exp\left[-\frac{(x_1 - \mu_1)^2}{(2\sigma_1^2)}\right] \frac{1}{\sqrt{2\pi\sigma_2^2}} \exp\left[-\frac{(x_2 - \mu_2)^2}{(2\sigma_2^2)}\right] \end{aligned}$$

Here

$$\begin{aligned} \mu_1 &= y_1 + f_1(t_i, y_1, y_2; \boldsymbol{\theta}) h, \\ \mu_2 &= y_2 + f_2(t_i, y_1, y_2; \boldsymbol{\theta}) h, \\ \sigma_1^2 &= g_1^2(t_i, y_1, y_2; \boldsymbol{\theta}) h, \\ \sigma_2^2 &= g_2^2(t_i, y_1, y_2; \boldsymbol{\theta}) h. \end{aligned}$$

That is, $G_{\boldsymbol{\theta}}^h(x_1, x_2, y_1, y_2, t_i; \boldsymbol{\theta})$ is the conditional density of $X_{1,i+1}$ and $X_{2,i+1}$ given $X_{1,i} = y_1$, $X_{2,i} = y_2$ and $\boldsymbol{\theta} = \boldsymbol{\theta}$, evaluated at the point (x_1, x_2) . The fact that the conditional density is a product of normal distributions with means μ_1, μ_2 and variances σ_1^2, σ_2^2 can be shown using (3.25) together with the fact that $X_{1,i+1}$ and $X_{2,i+1}$ are conditionally independent given $X_{1,i}$ and $X_{2,i}$. This conditional independence is a direct consequence of having two independent random variables $Z_{1,i}$ and $Z_{2,i}$ in (3.25).

The crux of the DTQ method is to apply quadrature to (3.26) to evolve an initial density forward in time. Consider a spatial grid with fixed spacing $k > 0$ and grid points $x_1^r = rk$, $x_2^s = sk$, $y_1^{r'} = r'k$, and $y_2^{s'} = s'k$. Then we apply the trapezoidal rule in both the y_1 and y_2 variables to obtain:

$$\hat{p}_{i+1}(x_1^r, x_2^s; \boldsymbol{\theta}) = k^2 \sum_{r'=-\infty}^{\infty} \sum_{s'=-\infty}^{\infty} G_{\boldsymbol{\theta}}^h(x_1^r, x_2^s, y_1^{r'}, y_2^{s'}, t_i; \boldsymbol{\theta}) \hat{p}_i(y_1^{r'}, y_2^{s'}; \boldsymbol{\theta}). \quad (3.27)$$

It is unnecessary to sum over all of \mathbb{Z}^2 . We know that a two-dimensional Gaussian decays to zero far from its mean. Since the mean (μ_1, μ_2) is approximately (y_1, y_2) , we sum only from $y_1 = x_1 - \zeta k$ to $y_1 = x_1 + \zeta k$ and similarly for y_2 :

$$\hat{p}_{i+1}(x_1^r, x_2^s; \boldsymbol{\theta}) = k^2 \sum_{r'=r-\zeta}^{r+\zeta} \sum_{s'=s-\zeta}^{s+\zeta} G_{\boldsymbol{\theta}}^h(x_1^r, x_2^s, y_1^{r'}, y_2^{s'}, t_i; \boldsymbol{\theta}) \hat{p}_i(y_1^{r'}, y_2^{s'}; \boldsymbol{\theta}). \quad (3.28)$$

We choose ζ manually to ensure the accuracy of the computation. We now have our method to evaluate $\tilde{p}(\mathbf{x}_{m+1} | \mathbf{x}_m, \boldsymbol{\theta})$. Let us take $i = 0$ in (3.28) to correspond to the time t_m . We start with the deterministic initial condition $\mathbf{X}_0 = \mathbf{x}_m$, corresponding to the density $\tilde{p}_0(\mathbf{x}) = \delta(\mathbf{x} - \mathbf{x}_m)$. Inserting this point mass into (3.26), we obtain a Gaussian density for $\tilde{p}_1(\mathbf{x})$. For each $r, s \in [-y_L/k, y_L/k]$, we set

$$\hat{p}_1(x_1^r, x_2^s; \boldsymbol{\theta}) \approx \tilde{p}_1(x_1^r, x_2^s; \boldsymbol{\theta}).$$

Now that we have \hat{p}_1 , we use (3.28) repeatedly to compute \hat{p}_2 , \hat{p}_3 , and so on until we reach \hat{p}_n . The object \hat{p}_n is then a spatially discrete approximation of the transition density from time t_m to time $t_m + nh = t_{m+1}$. For this last density, instead of evaluating it on the spatial grid used by the trapezoidal rule, we evaluate the density at the data \mathbf{x}_{m+1} . This avoids interpolation. In this way, we compute a numerical approximation of $\tilde{p}(\mathbf{x}_{m+1}|\mathbf{x}_m, \boldsymbol{\theta})$, as required for the likelihood function.

3.4 Results

We now present numerical tests of our algorithm in three cases. For each case, we generate multiple sample paths using a specified SDE with known parameters. We use $\boldsymbol{\theta}$ to denote the true parameter vector. Using the data thus generated, we then use our method to produce $\hat{\boldsymbol{\theta}}$, our maximum likelihood estimate of the parameter vector. In the first two scenarios, the SDE we use for generating data coincides with the SDE used for inference. In the third scenario, we use a generic polynomial SDE for inference—this SDE includes as a special case the SDE used for generating data.

To test the performance of the algorithm, we generate data using the Euler-Maruyama approximation of the SDE. We step forward in time, starting from t_0 to a final time point $T > 0$. We use a step size of ξ , where $\xi = 10^{-4}$ unless specified otherwise. We retain the samples only at times $t = m\Delta t$ from $m = 0$ to $m = M$, where $M\Delta t = T$. For consistency during comparisons, we set $t_0 = 0$, $T = 25$, and $\Delta t = 1$.

3.4.1 Linear SDE (Ornstein-Uhlenbeck process)

We consider the SDE for the Ornstein-Uhlenbeck process with linear drift and constant diffusion terms:

$$dX_t = \theta_1(\theta_2 - X_t)dt + \theta_3 dW_t. \quad (3.29)$$

For the first set of experiments, the true parameter vector is $\boldsymbol{\theta} = (0.5, 0, 1)$. We start the optimizer with an initial condition $\boldsymbol{\theta}_0 = (1, 2, 0.5)$. We study how well we are able to infer the parameters as a function of DTQ's internal time step h and the number of sample paths. For this set of experiments, the spatial grid spacing k is set to $k = h^{0.75}$. In Table 3.1, we summarize this information together with the RMS (root-mean-square) error between the estimated and true parameter values. This is equivalent to the 2-norm error, $\|\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}\|_2$. We also record the number of iterations required for the optimizer to converge to the minimizer of the objective function, the negative log likelihood.

The method is not as sensitive to h as one might expect. Instead, what we find is that the error decreases when we decrease the number of sample paths. When we use only 100 sample paths, we obtain a qualitatively reasonable solution for all three components of $\boldsymbol{\theta}$, with θ_2 in particular identified up to machine precision.

To explore whether the above findings were peculiar to the way we generated the data, we conducted another series of tests starting with a true parameter vector of $\boldsymbol{\theta} = (0.5, 0.9, 1)$. The results are displayed in Table 3.2. This time, when we use the Euler-Maruyama method to generate data, we use an internal time step of $\xi = 10^{-6}$, retaining all other parameters described above. For the inference, we give the optimizer an initial guess of $\boldsymbol{\theta}_0 = (1, 0.5, 0.5)$. We again set the spatial grid spacing to $k = h^{0.75}$ and record the RMS error.

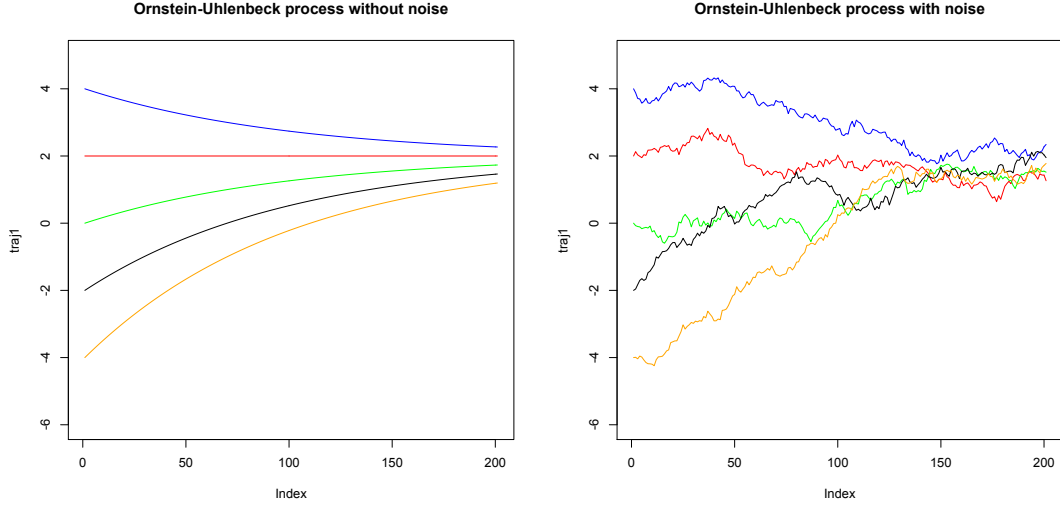


Figure 3.8: Illustration of sample paths generated for Ornstein-Uhlenbeck process for the same initial conditions. The plot on the left is the deterministic (ODE) version with no additive noise. The plot on the right is the stochastic (SDE) version with noise.

True θ	Initial θ_0	Estimated $\hat{\theta}$	Iterations	h	Paths	RMS Error
(0.5, 0, 1)	(1, 2, 0.5)	(1.020, 0, 1.404)	31	0.05	300	0.6597
(0.5, 0, 1)	(1, 2, 0.5)	(1.041, 0, 1.430)	30	0.02	300	0.6916
(0.5, 0, 1)	(1, 2, 0.5)	(1.048, 0, 1.438)	34	0.01	300	0.7028
(0.5, 0, 1)	(1, 2, 0.5)	(1.052, 0, 1.443)	34	0.005	300	0.7084
(0.5, 0, 1)	(1, 2, 0.5)	(1.054, 0, 1.445)	35	0.002	300	0.7119
(0.5, 0, 1)	(1, 2, 0.5)	(0.671, 0, 1.143)	31	0.01	100	0.2238
(0.5, 0, 1)	(1, 2, 0.5)	(0.673, 0, 1.146)	28	0.005	100	0.2264
(0.5, 0, 1)	(1, 2, 0.5)	(0.674, 0, 1.147)	26	0.002	100	0.2284

Table 3.1: Results for Case 1. Using either 300 or 100 sample paths produced by Euler-Maruyama simulation with time step $\xi = 10^{-4}$, we study the effect of reducing h , the internal DTQ time step.

The results from Table 3.2 show that if we increase the number of sample paths from 50 to 300, the error decreases dramatically. This leads us to our view that, for the present version of DTQ, the quality of the data is important. When we decrease the Euler-Maruyama time step from $\xi = 10^{-4}$ to $\xi = 10^{-6}$, we gain roughly one extra decimal place of accuracy in the sample paths. This leads DTQ towards higher quality estimates of the parameters in the Ornstein-Uhlenbeck model (3.29).

The performance of DTQ should increase as the number of sample paths increases. In this regard, we believe the results from Table 3.1 are an artifact of how the data was generated. We will see confirmation of this in the results below on a nonlinear SDE model.

Additionally, we note that Table 3.2 confirms that DTQ's results are relatively insensitive to decreasing h , the internal time step of DTQ. Note that the data set we use for the

experiments is collected at intervals of $\Delta t = 1$. We have found, in practice, that the choice $h = \Delta t/20$ is sufficient for inference. This is consistent with the results of Pedersen (1995), who chooses $h \approx \Delta t/25$.

True θ	Initial θ_0	Estimated $\hat{\theta}$	Iterations	h	Paths	RMS Error
(0.5, 0.9, 1)	(1, 0.5, 0.5)	(0.361, 0.968, 0.836)	39	0.050	50	0.2254
(0.5, 0.9, 1)	(1, 0.5, 0.5)	(0.362, 0.968, 0.839)	46	0.020	50	0.2226
(0.5, 0.9, 1)	(1, 0.5, 0.5)	(0.362, 0.968, 0.840)	42	0.010	50	0.2219
(0.5, 0.9, 1)	(1, 0.5, 0.5)	(0.362, 0.968, 0.841)	28	0.005	50	0.2212
(0.5, 0.9, 1)	(1, 0.5, 0.5)	(0.463, 0.885, 0.966)	45	0.050	300	0.05244
(0.5, 0.9, 1)	(1, 0.5, 0.5)	(0.466, 0.886, 0.973)	22	0.020	300	0.04561
(0.5, 0.9, 1)	(1, 0.5, 0.5)	(0.467, 0.886, 0.975)	22	0.010	300	0.04370
(0.5, 0.9, 1)	(1, 0.5, 0.5)	(0.468, 0.886, 0.976)	26	0.005	300	0.04237
(0.5, 0.9, 1)	(1, 0.5, 0.5)	(0.468, 0.886, 0.976)	20	0.002	300	0.04237

Table 3.2: Results for Case 1. Using either 300 or 100 sample paths produced by Euler-Maruyama simulation with time step $\xi = 10^{-6}$, we study the effect of reducing h , DTQ's internal time step.

3.4.2 Nonlinear SDE (Double Well Potential)

As our second example, we consider the following SDE with a nonlinear drift and constant diffusion term:

$$dX_t = \theta_1(\theta_2 - X_t^2)dt + \theta_3 dW_t. \quad (3.30)$$

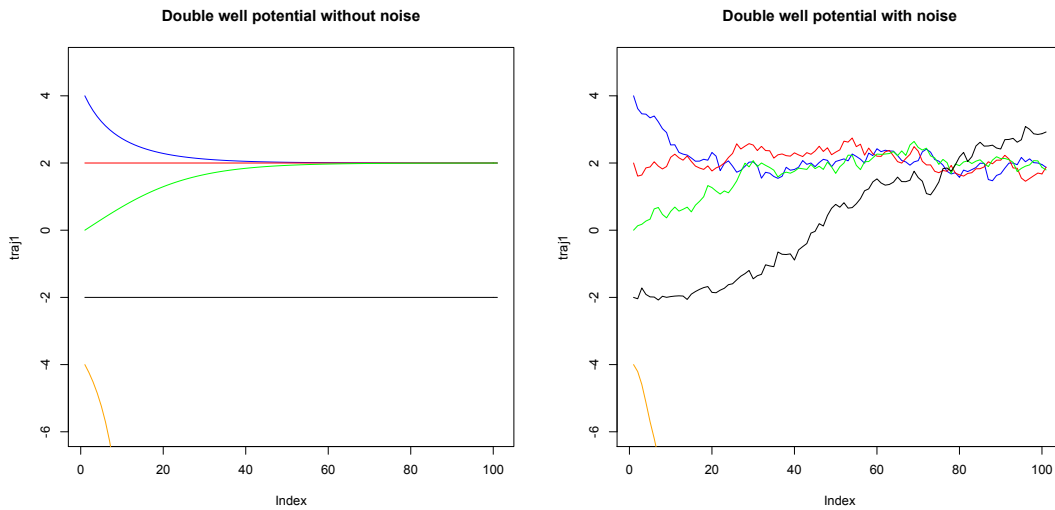


Figure 3.9: Illustration of sample paths generated for the double well potential process with same initial conditions. The plot on the left is the deterministic (ODE) version with no additive noise. The plot on the right is the stochastic (SDE) version with noise.

In Table 3.3, we show the results of an initial set of tests. In these tests, we vary both

the true parameter vector θ and the initial guess θ_0 that is given to the optimizer. For these tests, the data consists of 100 sample paths and the DTQ grid spacing is given by $k = h^{0.75}$. Note that even when θ_0 is far from θ , the estimated parameters $\hat{\theta}$ are close to θ . This trend holds for different values of θ . In fact, DTQ's RMS errors are quite low for all tests involving the nonlinear model (3.30).

True θ	Initial θ_0	Estimated $\hat{\theta}$	Iterations	h	RMS Error
(0.2, 1, 0.5)	(1, 1, 1)	(0.162, 0.886, 0.488)	37	0.05	0.06901
(0.4, 1, 0.5)	(1, 1, 1)	(0.629, 1.023, 0.618)	24	0.05	0.14965
(1, 4, 0.5)	(0.5, 0.5, 0.5)	(0.928, 3.990, 0.467)	50	0.01	0.04568
(1, 4, 0.5)	(2, 2, 1)	(0.925, 3.990, 0.430)	48	0.01	0.05935
(1, 4, 0.5)	(8, 8, 2)	(0.928, 3.990, 0.467)	47	0.01	0.04571

Table 3.3: Results for Case 2. We study a collection of problems involving different true θ values and different initial guesses θ_0 .

Next, in Table 3.4, we study the effect of decreasing DTQ's internal time step, h , when all other problem/algorithm parameters are kept fixed. For these tests, we set $\theta = (1, 4, 0.5)$, $\theta_0 = (2, 2, 1)$, and $k = h^{0.75}$. The data consists of 100 sample paths. The results show that it is possible to slightly reduce the RMS error by decreasing h , the internal time step. Based on these results, we see that there is no disadvantage incurred by using our method with $h = 0.05$; at this internal time step, the method runs very quickly in R.

True θ	Initial θ_0	Estimated $\hat{\theta}$	Iterations	h	RMS Error
(1, 4, 0.5)	(2, 2, 1)	(0.925046, 3.990012, 0.430020)	37	0.05	0.05948
(1, 4, 0.5)	(2, 2, 1)	(0.925311, 3.990029, 0.430068)	48	0.01	0.05935
(1, 4, 0.5)	(2, 2, 1)	(0.926930, 3.990418, 0.471400)	48	0.005	0.04563
(1, 4, 0.5)	(2, 2, 1)	(0.925808, 3.990464, 0.473724)	41	0.002	0.04577
(1, 4, 0.5)	(2, 2, 1)	(0.925433, 3.990480, 0.474493)	31	0.001	0.04583

Table 3.4: Results for Case 2. We study the effect of decreasing h , keeping all other parameters fixed.

In Table 3.5, we run a series of tests where each test is repeated twice, once with the spatial grid spacing set to $k = h^{0.75}$ and again with $k = h$. For these tests, we generate data with $\theta = (1, 4, 0.5)$. If we examine the first two rows of Table 3.5, what we see is that decreasing the spatial grid spacing has a significant, beneficial effect on the RMS error. What has happened here is that we have given the optimizer an initial guess where the third element of θ_0 is 0.1, a relatively small value. If we go back to the SDE (3.30), we see that this third element of θ_0 corresponds to the diffusion coefficient. When the diffusion coefficient is small, the Gaussian kernel G_θ^h becomes very narrow. This necessitates a finer spatial grid in order to resolve the kernel well enough to perform accurate quadrature. For the final four rows of Table 3.5, the third element of θ_0 is 1 and we do not observe as significant a reduction in RMS error when we refine the spatial grid.

True θ	Initial θ_0	Estimated $\hat{\theta}$	Iterations	k	Paths	RMS Error
(1, 4, .5)	(.5, .5, .1)	(0.100, 4.024, 0.100)	39	$h^{0.75}$	100	0.5688
(1, 4, .5)	(.5, .5, .1)	(1.035, 3.993, 0.499)	43	h	100	0.0205
(1, 4, 0.5)	(2, 2, 1)	(0.925, 3.990, 0.430)	48	$h^{0.75}$	100	0.0593
(1, 4, 0.5)	(2, 2, 1)	(0.955, 3.995, 0.481)	35	h	100	0.0283
(1, 4, 0.5)	(2, 2, 1)	(1.035, 3.993, 0.499)	75	$h^{0.75}$	300	0.0206
(1, 4, 0.5)	(2, 2, 1)	(1.022, 4.008, 0.497)	32	h	300	0.0138

Table 3.5: Results for Case 2. We compare spatial grid laws $k = h^{0.75}$ and $k = h$.

Finally, in Table 3.6, we study the effect of increasing the number of Euler-Maruyama sample paths in the data set that we feed into the inference algorithm. We keep all other algorithm and problem parameters fixed, with $\theta = (1, 4, 0.5)$, $\theta_0 = (2, 2, 1)$, $h = 0.01$, and $k = h^{0.75}$. The results show a steady improvement in the estimated $\hat{\theta}$ as the number of sample paths increase. The last row of Table 3.6 contains our best result for this inference problem with an RMS error less than 0.01.

True θ	Initial θ_0	Estimated $\hat{\theta}$	Iterations	Paths	RMS Error
(1, 4, 0.5)	(2, 2, 1)	(0.776, 4.060, 0.424)	100	2	0.1408
(1, 4, 0.5)	(2, 2, 1)	(0.899, 3.992, 0.510)	27	10	0.0583
(1, 4, 0.5)	(2, 2, 1)	(0.833, 4.018, 0.440)	35	50	0.1030
(1, 4, 0.5)	(2, 2, 1)	(0.925, 3.990, 0.430)	48	100	0.0593
(1, 4, 0.5)	(2, 2, 1)	(0.901, 4.007, 0.464)	33	200	0.0609
(1, 4, 0.5)	(2, 2, 1)	(1.035, 3.993, 0.499)	75	300	0.0206
(1, 4, 0.5)	(2, 2, 1)	(1.107, 3.994, 0.513)	43	400	0.0624
(1, 4, 0.5)	(2, 2, 1)	(0.988, 3.999, 0.489)	33	1000	0.0094

Table 3.6: Results for Case 2. We examine the effect of increasing the number of sample paths in the data set, keeping all other parameters fixed.

3.4.3 Generic Polynomial Drift and Diffusion Functions

For our third example, we reuse (3.30) to generate simulated data, but we use a more general model for the drift function, a generic cubic polynomial. In other words, for the purposes of inference, we use the SDE model

$$dX_t = (\theta_0 + \theta_1 X_t + \theta_2 X_t^2 + \theta_3 X_t^3)dt + \theta_4 dW_t. \quad (3.31)$$

We infer the parameters $\theta = (\theta_0, \theta_1, \theta_2, \theta_3, \theta_4)$ in the SDE (3.31) from the observations generated using the SDE (3.30) to see if we recover the correct form of the drift function. Ideally, DTQ will infer that θ_0 and θ_2 in (3.31) are zero.

In Table 3.7, we display our results for three values of h , the internal time step. We generate our data by simulating 100 sample paths of (3.30) with $\theta_1 = 0.2$, $\theta_2 = 4$, and $\theta_3 = 0.4$. Note that in terms of the inference model (3.31), this corresponds to $\theta = (0, 0.8, 0, -0.2, 0.4)$. For the initial guess, we use $\theta_0 = (0, 0, 0, 0, 0.5)$. In this particular set of tests, instead of using the BFGS algorithm described above, we use NLopt's method of moving asymptotes

(MMA) algorithm (Svanberg (2002)).

Overall, DTQ correctly identifies the qualitative form of the model. That is, we find that the first and third components of $\hat{\theta}$ are close to zero, and the remaining components of $\hat{\theta}$ are also close to their true values.

True θ	Initial θ_0	Estimated $\hat{\theta}$	Iter	h	RMSE
(0, .8, 0, -.2, .4)	(0, 0, 0, 0, .5)	(0.014, 0.619, -0.003, -0.154, 0.357)	69	0.005	0.0859
(0, .8, 0, -.2, .4)	(0, 0, 0, 0, .5)	(0.014, 0.867, -0.003, -0.217, 0.424)	57	0.002	0.0334
(0, .8, 0, -.2, .4)	(0, 0, 0, 0, .5)	(0.012, 0.766, -0.003, -0.192, 0.408)	89	0.001	0.0168

Table 3.7: Results for Case 3. We perform inference using model (3.31), which has a higher-dimensional parameter space than (3.30), the model used to generate the data.

3.4.4 Coupled SDEs

We implement the Metropolis algorithm in R. Inside the Metropolis algorithm, we evaluate the likelihood function using the DTQ method, which is implemented in C++ as an R package. Note that all code and data used in this work is available online (<https://github.com/hbhat4000/sdeinference>)—see the **Rdtq2d** and **pursuit2d** directories.

Electrical Oscillator

To test the method, we first consider the SDE

$$dX_{1,t} = -\frac{X_{2,t}}{L}dt + \frac{s_1^2}{L}dW_{1,t}, \quad dX_{2,t} = \frac{X_{1,t}}{C}dt + \frac{s_2^2}{C}dW_{2,t}. \quad (3.32)$$

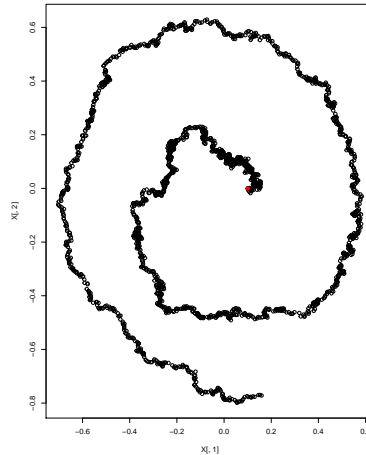


Figure 3.10: Illustration of the sample path generated by an LC circuit. The dependent variables $X_{1,t}$ and $X_{2,t}$ represent, respectively, the current and voltage of the circuit at time t .

This system describes a noisy electrical oscillator with one inductor (with inductance L) and one capacitor (with capacitance C) as shown in Figure 3.10. The dependent variables $X_{1,t}$ and $X_{2,t}$ represent, respectively, the current and voltage of the circuit at time t .

Our goal here is to test the performance of the algorithm using simulated data. To generate this data, we start with known values of the parameters: $L = C = (2\pi)^{-1}$ and $s_1 = s_2 = .4/\sqrt{2\pi}$. Using a fixed initial condition $(X_{1,0}, X_{2,0})$, we then use the Euler-Maruyama method to step (3.32) forward in time until a final time $T > 0$. When we carry out this time-stepping, we use a step size of 0.001 and then retain only those samples at times $t_m = m\Delta t$, from $m = 0$ to $m = M$, where $M\Delta t = T$. The simulated data is taken over two periods of the oscillator ($T = 2$) with a full resolution of $\Delta t = 0.01$. By, for example, taking every other row of this data set, we can obtain data with a resolution of $\Delta t = 0.02$.

Using the samples $\{\mathbf{x}_m\}_{m=0}^M$ thus constructed, we run the Metropolis algorithm. Because capacitance and inductance are physically constrained to be positive, we set $1/L = \theta_1^2$. For the tests presented here, we infer only θ_1 , keeping other parameters fixed at their known values. For θ_1 , we use a diffuse Gaussian prior with mean 0 and standard deviation 100. For the proposal distribution Z_{N+1} in the auxiliary Markov chain, we choose i.i.d. Gaussians with mean 0 and standard deviation 0.35.

When we run the Metropolis algorithm, we discard the first 100 samples and retain the next 1000 samples. For each value of Δt and the DTQ time step h , we compute both the mean of the samples of θ_1^2 and the mode of the kernel density estimate of θ_1^2 . We compare these values against the true value of the parameter $1/L = 2\pi$ and record the relative errors as, respectively, e_1 and e_2 :

Δt	h	e_1 (relative error of mean)	e_2 (relative error of mode)
0.04	0.04	6.1%	7.6%
0.04	0.02	0.54%	6.8%
0.04	0.01	5.1%	1.1%
0.02	0.02	12%	14%
0.02	0.01	4.9%	2.3%

When $h = \Delta t$, only one step of the method described in Section 3.3 is required to go from time t_m to t_{m+1} . This step does not use any quadrature at all—one merely evaluates (3.26) using a point mass for the density at time t_m . The resulting likelihood function is a product of Gaussians. On the other hand, when h is strictly less than Δt , we must use quadrature (i.e., the actual DTQ method) to step forward in time from t_m to t_{m+1} . Clearly, using the DTQ method to compute the likelihood yields more accurate posteriors than using a purely Gaussian likelihood.

To visualize the results, we present Figure 3.11. The true value of $1/L = 2\pi$ is indicated by the dashed black line. The posterior mode for $h = 0.01$ is indicated by the solid black line. The curves are kernel density estimates computed using the posterior samples described above.

Comparison of DTQ with POMP

In the following section, we present a detailed comparison between DTQ and POMP (King *et al.* (2016)). A partially observed Markov process (POMP) model consists of incomplete and noisy measurements of a latent, unobserved Markov process. These models are characterized by the transition density for the Markov process and the measurement density.

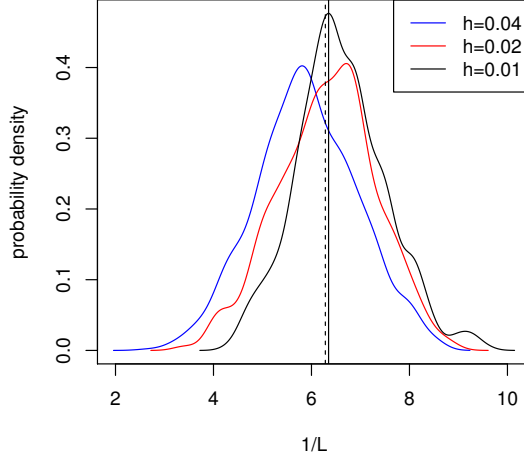


Figure 3.11: We plot kernel density estimates of posterior densities $p(\theta_1^2 | \mathbf{x})$. We use simulated data with $\Delta t = 0.04$, generated as described above. Each posterior density corresponds to a finer DTQ step h . As we take a finer DTQ step (i.e., as h decreases), the posterior mode approaches the true value indicated by the solid vertical line at $1/L = 2\pi$.

POMP facilitates inference of system parameters using time series data with frequentist and Bayesian methods.

Let θ be a p -dimensional parameter, $\theta \in \mathbb{R}^p$. Let $t_i \in T, i = 0, \dots, N$, be the times at which $X(t; \theta)$ is observed, and $t_0 \in T$ is the initial time. $X_i = X(t_i; \theta)$ represents the observed data at time point t_i and $X_{i:j} = (X_i, X_{i+1}, \dots, X_j)$ represents the observed data in the time interval $[t_i, t_j]$. The observations of the latent dynamical system $X_{0:N}$ are $Y_{1:N}$. The data, $y_{1:N}^* = (y_1^*, \dots, y_N^*)$, are modeled as a realization of the observation process and are known. The POMP model implies that the joint density of $X_{0:N}$ and $Y_{1:N}$ is given by $f_{X_{0:N}, Y_{1:N}}(x_{0:N}, y_{1:N}; \theta)$. This joint density is determined by the initial density $f_{X_0}(x_0; \theta)$, together with the conditional transition probability density, $f_{X_n | X_{n-1}}(x_n | x_{n-1}; \theta)$ and the measurement density $f_{Y_n | X_n}(y_n | x_n; \theta)$, for $1 \leq n \leq N$.

The log likelihood for a POMP model is $\ell(\theta) = \log f_{Y_{1:N}}(y_{1:N}^*; \theta)$, which can be written as a sum of conditional log likelihoods,

$$\ell(\theta) = \sum_{n=1}^N \ell_{n|1:n-1}(\theta), \quad (3.33)$$

where

$$\begin{aligned} \ell_{n|1:n-1}(\theta) &= \log f_{Y_n | Y_{1:n-1}}(y_n^* | y_{1:n-1}^*; \theta) \\ &= \log \int f_{Y_n | X_n}(y_n^* | x_n; \theta) f_{X_n | Y_{1:n-1}}(x_n | y_{1:n-1}^*; \theta) dx_n. \end{aligned}$$

POMP models compute the log likelihood $\ell(\theta)$ using a Sequential Monte Carlo (SMC, or particle filter) method. SMC builds up a representation of $f_{X_n | Y_{1:n-1}}(x_n | y_{1:n-1}^*; \theta)$ that can be used to obtain an estimate, $\hat{\ell}_{n|1:n-1}(\theta)$ of $\ell_{n|1:n-1}$, and hence an approximation $\hat{\ell}(\theta)$ of $\ell(\theta)$. The particle filter generates a swarm of particles representing $f_{X_n | Y_{1:n}}(x_n | y_{1:n}^*; \theta)$ which is analogous to the Monte Carlo samples, $X_{n,j}^F, j = 1 : J$.

For comparison, we use precisely the same data, priors, and initial conditions as above

in Section 3.4.4, to produce 2000 post-burn-in samples from the posterior using the particle filter. We infer $\theta_1 = 1/L = 1.0$, $\theta_2 = 1/C = 1.0$ and $\theta_3 = s_1^2/L = s_2^2/C = 4.0$. The time interval is set to $\Delta t = 0.1$ and inference is done for varying number of intermediate steps h taken. In Figure 3.13 we compare the posterior distributions and MAP estimates for DTQ and POMP. As shown, DTQ performs comparably to the particle filtering model for inference of θ_1 and θ_2 , the drift parameters. For the diffusion parameter, θ_3 , POMP model performs much better than DTQ.

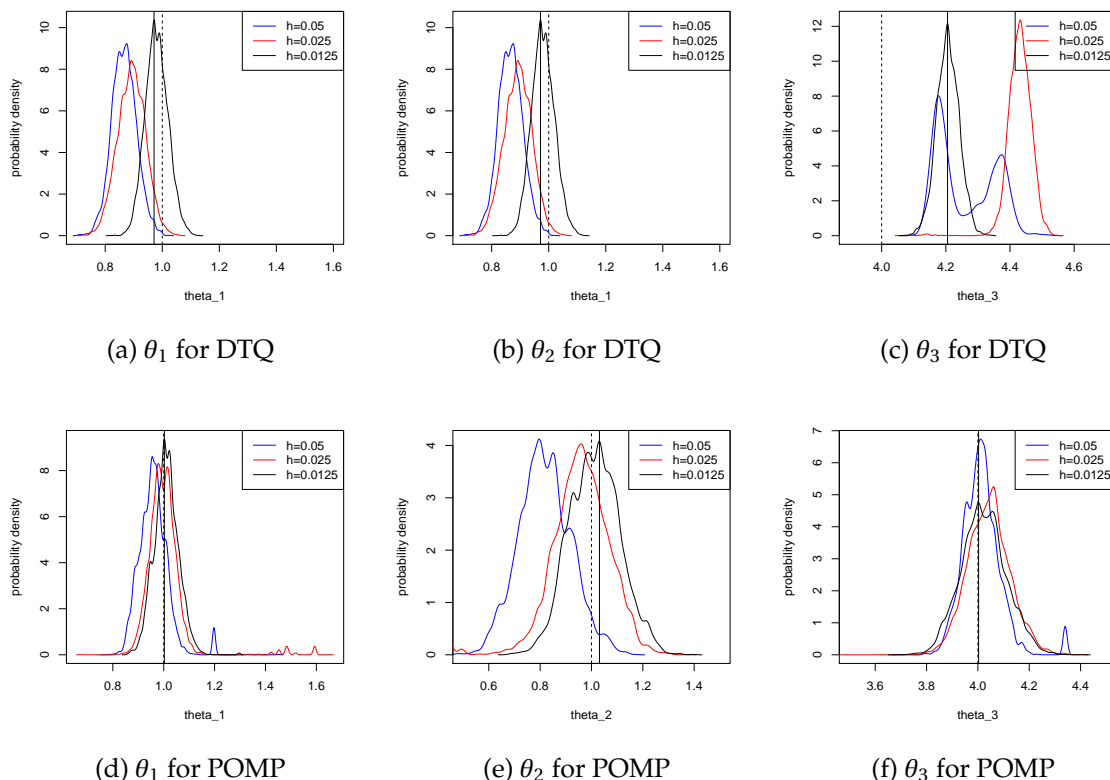


Figure 3.12: We plot the probability density for DTQ using a naïve MH-sampler and POMP using an adaptive MH-sampler, for varying intermediate time steps h . The blue, red and black curves are the posterior distributions for $h = \Delta t/2, \Delta t/4$, and $\Delta t/8$ respectively. The solid black line is the value of the true parameter, and the dotted black line represents the MAP estimate.

In Table 3.8 we further compare the mean and standard deviation of the inferred parameters from the true parameters. We show that introducing internal time steps improves the estimates as compared to the Eulerian estimate. Though the acceptance rate is lower for the sampler using POMP, the estimates are closer to the true value.

$\Delta t = 0.1$		Mean			RMS error	Std Dev			Accept rate (%)
		θ_1	θ_2	θ_3		θ_1	θ_2	θ_3	
$h = \Delta t$	Eulerian	0.747	0.906	3.072	0.557	0.062	0.076	0.031	29.55
$h = \Delta t/2$	pomp	0.960	0.809	4.014	0.113	0.058	0.123	0.076	10.99
	DTQ	0.866	1.305	4.260	0.244	0.043	0.082	0.092	28.53
$h = \Delta t/4$	pomp	1.004	0.954	3.992	0.027	0.083	0.125	0.367	16.41
	DTQ	0.892	1.157	4.430	0.271	0.048	0.069	0.037	25.43
$h = \Delta t/8$	pomp	1.01	1.01	4.02	0.015	0.048	0.098	0.087	17.14
	DTQ	0.98	1.17	4.21	0.155	0.039	0.077	0.035	23.87

Table 3.8: Comparison of the mean and standard deviation of inferred parameters, θ_1, θ_2 and θ_3 , for DTQ and POMP, for varying intermediate time steps, $h = \Delta t, \Delta t/2, \Delta t/4$ and $\Delta t/8$.

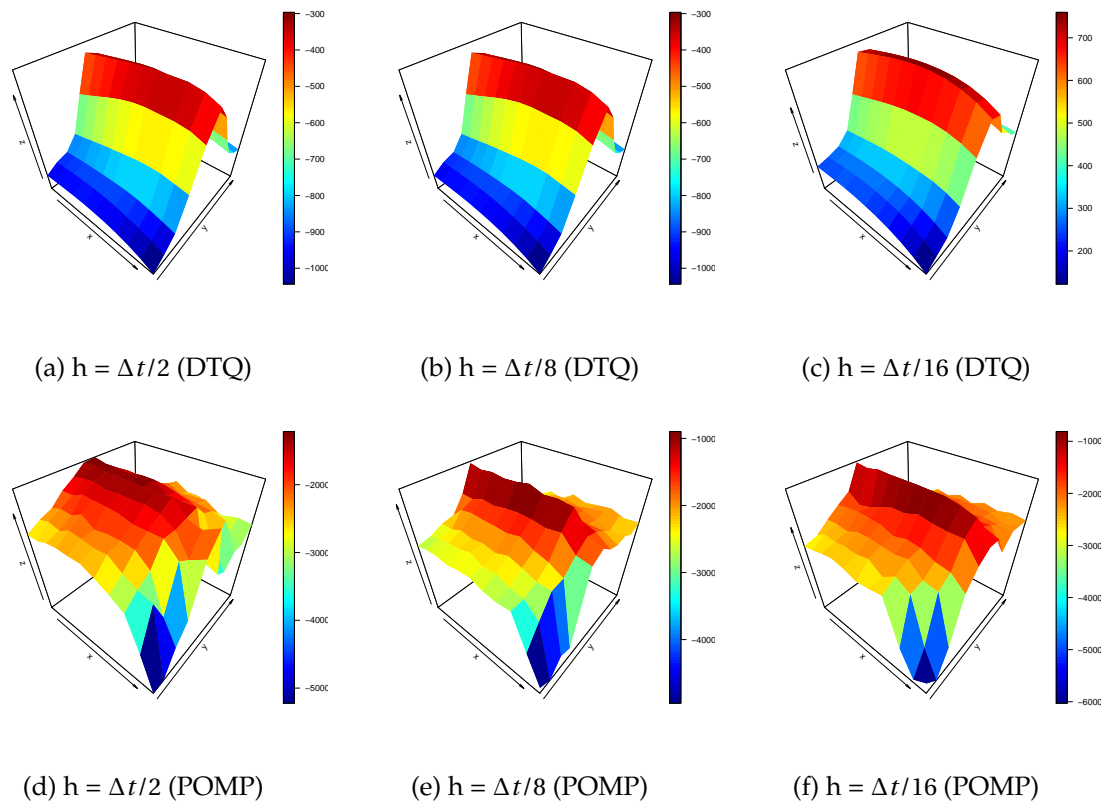


Figure 3.13: We plot the estimated likelihood surfaces to get an insight on the approximation of the likelihood, $\ell(\theta)$ for DTQ and POMP. The plots are on the θ_2 vs θ_3 grid using 100 intervals and varying intermediate time steps ($h = \Delta t/2, \Delta t/8$ and $\Delta t/16$). The plots show that the DTQ method produces a smoother approximation of the likelihood as compared to the particle filter.

$\Delta t = 0.1$	Difference in Mean		Difference in Mode	
	POMP	DTQ	POMP	DTQ
$h = \Delta t$	0.029	0.005	0.044	-0.014
$h = \Delta t / 2$	-0.373	-0.062	-0.014	0.025
$h = \Delta t / 3$	-0.235	-0.156	-0.034	-0.020
$h = \Delta t / 4$	-0.221	-0.086	-0.030	-0.048
$h = \Delta t / 5$	-0.028	-0.090	-0.066	-0.052
$h = \Delta t / 6$	-0.067	-0.058	-0.045	-0.044
$h = \Delta t / 7$	-0.181	-0.056	-0.039	-0.041
$h = \Delta t / 8$	-0.052	-0.063	-0.027	-0.041
$h = \Delta t / 9$	-0.273	-0.059	-0.041	-0.039
$h = \Delta t / 10$	-0.028	-0.095	-0.030	-0.035

Table 3.9: Comparing difference in mean and modes for the estimated parameters from the true parameters for POMP and DTQ methods.

The likelihood approximation plots in Figure 3.13 show that DTQ provides a smooth approximation of the likelihood function and thus a vanilla Metropolis-Hastings sampler can be used to sample the intermediate steps. In the case of POMP particle filter, the likelihood estimation is uneven with sharp changes. This makes the particle filters more sensitive to the initial condition, number of particles and the sampler. An adaptive Metropolis-Hastings sampler is needed with fine-tuned parameters to reach inference results comparable to those produced by DTQ. We believe that these results indicate that DTQ can be used an alternative to more established methods implemented in POMP, as it provides a more stable approximation of the likelihood function. This enables vanilla samplers to generate reasonable estimates of the parameters.

Basketball Tracking Data

Next, we test the method using the pursuit SDE (3.24). We set the runner's trajectory equal to a sinusoidal curve $y = \sin(\pi x)$ from $x = -1$ to $x = 1$. We assume the runner covers this trajectory over the time period $0 \leq t \leq 8$. The chaser's trajectory is simulated using the Euler-Maruyama method to step (3.24) forward in time from a fixed initial condition $\mathbf{X}_0 = (x_0^c, y_0^c)$. During the generation of the data, we use a step size of 10^{-4} . By downsampling this single time series, we generate time series with spacing $\Delta t = 0.4, 0.2$ and 0.1 .

For the test presented here, the values of the parameters are $v_1 = 0.15$, $v_2 = 0.1$, and

$$\gamma(t) = \begin{cases} \gamma_1 = 0.4 & \text{if } 0 \leq t < 4 \\ \gamma_2 = 1.0 & \text{if } 4 \leq t \leq 8. \end{cases}$$

Because we want all speeds and diffusion constants to be positive, we take $\gamma_i = e^{\theta_i}$ and $v_i = e^{\theta_{i+2}}$ for $i = 1, 2$. The priors for θ_1 and θ_2 are normal with variance one and mean equal to the log of the mean speed of the chaser computed over the chaser's entire trajectory. The priors for θ_3 and θ_4 are normal with mean $\log(0.4)$ and variance 1. We use mean zero Gaussian proposals for all components of $\boldsymbol{\theta}$. We choose the variances of these proposals so that the acceptance rate for all runs is near 30%.

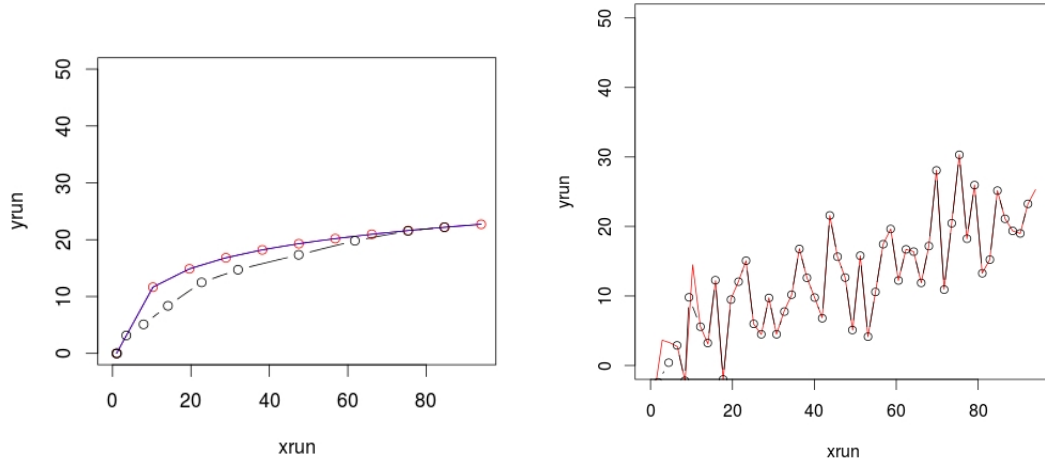


Figure 3.14: We plot simulated data for the runner and chaser on the standard basketball court of dimensions 94ft. \times 50 ft. The plot on the left creates the runner's trajectory as $y^r = 5\log(x^r)$. The chaser's speed is $\gamma(t) = 1 + 2t$ and we consider 10 points along the trajectory. For the plot on the right, we consider 50 points on both the trajectories. The runner's trajectory in this case is randomized. $y^r = ((x^r)^2)/100 + (x^r/20) + \mathcal{N}(\mu = 0, \sigma = 2)$.

Using the samples $\{\mathbf{x}_m\}_{m=0}^M$ thus constructed, we run the Metropolis algorithm with $h = \Delta t/i$ with $i = 1, 2, 3, 4$. For each choice of parameters Δt and h , we compute 10100 samples and discard the first 100. To compute the runner's trajectory at intermediate points, we use linear interpolation between times t_m and t_{m+1} . We tabulate the results below; each value of γ_1 represents the mean of e^{θ_1} over all Metropolis samples of θ_1 :

parameters	γ_1	γ_2	v_1	v_2
$\Delta t = 0.1; h = 0.1/1$	0.301	0.748	0.124	0.0886
$\Delta t = 0.1; h = 0.1/2$	0.311	0.956	0.124	0.0858
$\Delta t = 0.1; h = 0.1/3$	0.307	1.011	0.117	0.0805
$\Delta t = 0.1; h = 0.1/4$	0.308	1.025	0.120	0.0829
$\Delta t = 0.2; h = 0.2/1$	0.306	0.650	0.142	0.1146
$\Delta t = 0.2; h = 0.2/2$	0.310	0.877	0.137	0.1197
$\Delta t = 0.2; h = 0.2/3$	0.309	1.015	0.112	0.0844
$\Delta t = 0.2; h = 0.2/4$	0.304	1.019	0.111	0.0852
$\Delta t = 0.4; h = 0.4/1$	0.292	0.514	0.188	0.2010
$\Delta t = 0.4; h = 0.4/2$	0.312	0.960	0.177	0.1774
$\Delta t = 0.4; h = 0.4/3$	0.307	0.987	0.124	0.1447
$\Delta t = 0.4; h = 0.4/4$	0.303	1.014	0.145	0.1130

Overall, the results show that our algorithm produces mean posterior estimates that are reasonably close to the ground truth values. When the spacing of the data Δt is large, we see greater benefit from using the DTQ method. For instance, when $\Delta t = 0.4$, the mean estimates of γ_2 improve dramatically from 0.514 to 1.014 as we decrease h , i.e., as we take

more internal DTQ steps. Similar trends can be seen for v_1 and v_2 .

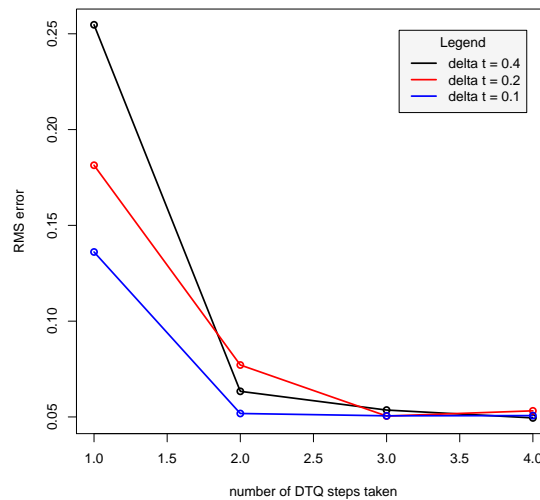


Figure 3.15: As the number of DTQ steps increase ($h = \Delta t, \Delta t/2, \Delta t/3, \Delta t/4$), the L^2 norm error between the estimated and the true parameters decrease, for $\Delta t = 0.1, 0.2$ and 0.4 .

We now turn to real tracking data taken from the game played between the Golden State Warriors and the Sacramento Kings on October 29, 2014. Reviewing this game, we found a fast break where Stephen Curry (of the Warriors) was the runner and Ramon Sessions (of the Kings) was the chaser. The entire fast break lasts 4.12 seconds. The spatial tracking data is recorded at intervals of 0.04 seconds, for a total of 104 observations. The tracking data uses the position on a court of dimension 94×50 . We have rescaled the data to lie in a square with center $(0, 0)$ and side length equal to one.

To parameterize the chaser's speed $\gamma(t)$, we have used a piecewise constant approximation with 8 equispaced pieces. Combined with the diffusion constants v_1 and v_2 , this yields a 10-dimensional parameter vector θ . As in the previous simulated data test, we set the true parameters γ_i and v_i to be the exponentials of the corresponding elements of the θ vector.

For the Metropolis sampler, the priors and proposals are higher-dimensional versions of those described in the simulated data test above. The main difference is that we now generate only 1000 post-burnin samples.

Using the Metropolis samples, we compute a kernel density estimate of each parameter. We then treat the mode of each computed density as the MAP (maximum a posteriori) estimate of the corresponding parameter. We then use the MAP estimates of the parameters in the pursuit SDE (3.24). We generate 100 sample paths of this SDE using the Euler-Maruyama method with time step 10^{-4} . As shown in Figure 3.16, the mean of these sample paths (plotted in black) agrees very well with the chaser's trajectory (plotted in red). This gives evidence that our stochastic pursuit system is an appropriate model for NBA fast breaks involving one runner and one chaser.

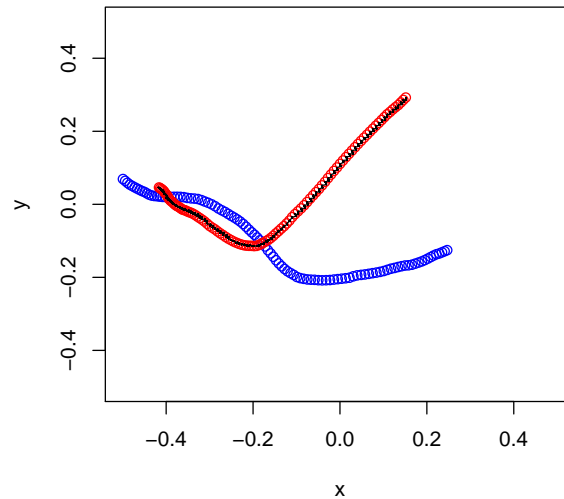


Figure 3.16: The agreement between the black curve (mean of simulated stochastic pursuit trajectories using MAP estimated parameters) and the red curve (chaser's trajectory) shows that the stochastic pursuit model is appropriate. The runner's trajectory is given in blue.

To visualize the insight provided by the model, we plot in Figure 3.17 the MAP estimated $\gamma(t)$ function over the time period of the fast break, $0 \leq t \leq 4.12$. The speed $\gamma(t)$ is the piecewise constant function plotted in black, while the mean speed computed directly from the data is given by a red horizontal line. The inferred speed shows that the chaser slows down dramatically approximately 1.5 seconds into the fast break. If one reviews the video footage of the play, this corresponds to the runner confusing the chaser and evading him.

Given our stochastic pursuit model's success in fitting the real data, in future work, we seek to apply the same methodology to a much larger sample of fast breaks. In this way, we can quantify a runner's ability to evade a chaser and/or a chaser's ability to stay near a runner who is actively trying to score.

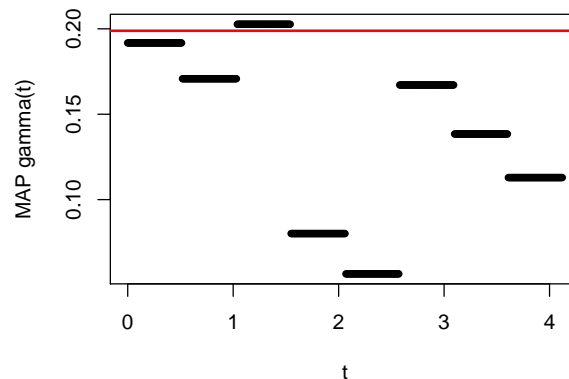


Figure 3.17: For the fast break tracking data described in the text, we plot the MAP estimate of the chaser's speed $\gamma(t)$ in black. Note that the inferred speed differs greatly from the mean speed across the entire trajectory, plotted as a horizontal red line.

3.5 Discussion

In this chapter, we have both derived and experimentally studied a new algorithm for parameter inference in stochastic differential equation models. The crux of the algorithm is to use quadrature to compute the transition densities required for the both the log likelihood function and its gradient.

The results in Section 3.4 clearly demonstrate several conditions under which DTQ performs well. In particular, we find empirical evidence justifying the approximations made in Section 4.2—especially (A.14) and (A.15), which have not been justified in prior theoretical work. Once the internal time step h is sufficiently small, further reduction of h does not significantly improve the quality of the inferred $\hat{\theta}$. We do find that certain algorithm parameters, such as the spatial grid spacing k , do need to be adjusted to handle scenarios such as very small diffusion coefficients.

We have seen that the primary challenge to be addressed is that the present version of DTQ, in order to produce highly accurate results, requires high-quality data. In the next chapter, we continue our work for measurement models that enable filtering of noisy observations. To create a scalable implementation of the DTQ method, we implement the algorithm in a parallel and distributed fashion using Apache Spark, allowing for time-dependent drift and diffusion coefficients and for data that is not equispaced in time.

Chapter 4

Scalable SDE Filtering and Inference

In this chapter, we consider the problem of Bayesian filtering and inference for time series data modeled as noisy, discrete-time observations of a stochastic differential equation (SDE) with undetermined parameters. We develop a Metropolis algorithm to sample from the high-dimensional joint posterior density of all SDE parameters and state time series. Our approach relies on an innovative density tracking by quadrature (DTQ) method to compute the likelihood of the SDE, the part of the posterior that requires the most computational effort to evaluate. As we show, the DTQ method lends itself to a natural implementation using Scala and Apache Spark, an open source framework for scalable data mining. We study the performance and scalability of our algorithm on filtering and inference problems for both regularly and irregularly spaced time series.

4.1 Introduction

In this chapter, we focus on the problem of Bayesian inference and filtering for time series. The time series consist of noisy observations of a process that satisfies a stochastic differential equation (SDE). The drift and diffusion terms of this SDE contain undetermined parameters. Our goal is to use the observations to infer both the time series of the SDEs actual state (the filtering problem) and the parameters in the drift/diffusion terms (the inference problem). While there are existing packages (e.g., for use with the R language for statistical computing) for solving either the inference problem or the filtering problem, we are not aware of an existing package that solves both problems simultaneously in a Bayesian fashion. The method and implementation described here is the first to make use of Apache Spark, an open source framework for scalable data processing and machine learning on computing clusters (<http://spark.apache.org>).

The problem we consider arises in many recent studies in neuroscience (Kneissler *et al.* (2015)) and systems biology (Sun *et al.* (2008)). In these fields, advances in measurement technology has led to large amounts of experimental data on biophysical processes. The data often consist of noisy and/or incomplete time series measurements of the system's state. Due to intrinsic noise on the cellular scale, these processes are often modeled by SDEs with undetermined parameters. The task is then to use the data to infer these parameters and the true states of the process.

To solve the Bayesian inference and filtering problem, we develop a Metropolis algo-

rithm to sample from the joint posterior density of the state series and the SDE parameters. A key bottleneck in the inference/filtering problem for SDEs, identified by many authors, is the evaluation of the conditional density of the state series given the parameters, i.e., the likelihood function for the SDE. The exact likelihood function for the SDE can only be computed in special cases when we can solve analytically for the SDEs transition density. 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 Chen (2003); Sørensen (2004); Iacus (2009); Barber *et al.* (2011); Fuchs (2013).

Consider the computation of the transition density $p_{X_{t_{j+1}}}(x_{j+1}|X_{t_j} = x_j, \theta)$. Here X_t stands for the state of a process that evolves forward in time via an SDE with parameter vector θ . We let x_j and x_{j+1} denote the true states of the system at times t_j and t_{j+1} . 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 $p(x, t_j) = \delta(x - x_j)$ up to time t_{j+1} . Then $p(x_{j+1}, T)$ will be a numerical approximation of the transition density. The Kolmogorov equation is a linear partial differential equation (PDE) with spatially-dependent coefficients.

Our approach is similar in that we also numerically track the density $p(x, t)$ without sampling. Instead of numerically solving a PDE, we track the density by applying quadrature to the Chapman-Kolmogorov equation associated with a time-discretization of the SDE (4.1a). We detail this density tracking by quadrature (DTQ) method in Section 4.2. As we show, the DTQ method enables one to break the computation of the likelihood into a sum of likelihoods involving consecutive pairs of observations (t_j, x_j) and (t_{j+1}, x_{j+1}) just as described above. For each such pair, the DTQ method computes the likelihood using iterated matrix multiplication. In Section 4.3, we show how these features of the DTQ method enable it to naturally take advantage of the efficiency and parallelism of Apache Spark and Scala.

The central contribution of this work is a DTQ-based Metropolis algorithm, and ensuing implementation in Scala and Spark, to sample from the joint posterior of the parameters and the state series given the observations. To clarify, we note that in our problem, the data consists of *noisy* observations of the process X_t : we let y_j denote the observation at time t_j . Equipped with the DTQ method, we develop a Metropolis algorithm to sample from the joint posterior $p(\theta, \mathbf{x}|\mathbf{y})$. Note that this is in contrast to numerous prior studies that treat the state series \mathbf{x} as a hidden or latent variable; in such approaches, the term “posterior” is used to denote $p(\theta|\mathbf{y})$. The `pomp` R package implements such an approach using particle filtering and particle Markov Chain Monte Carlo (King *et al.* (2016)). Our approach treats the state series \mathbf{x} as a parameter vector. This is necessary in situations where there is significant and/or interdependent uncertainty in both the parameters and the state series.

In Section 4.4, we describe experimental tests of our algorithm using both regularly and irregularly spaced time series. While there is further room for improvement, especially with regards to the classical Metropolis accept/reject step used here, the tests show that our current code does solve the Bayesian inference and filtering problem at a baseline acceptable level. The tests also establish the scalability of the algorithm, both as a function of the number of Spark processors and as a function of the length of the time series.

As far as computing the likelihood function of the SDE is concerned, the DTQ method is most similar to the methods 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 (4.1a). 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. Unless one generates sample paths conditional on both $X_{t_j} = x_j$ and $X_{t_{j+1}} = x_{j+1}$, this approach is problematic.

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

Other approaches that have been explored in the literature include likelihood-free methods such as Approximate Bayesian Computation (Picchini (2014)), variational methods (Archambeau *et al.* (2008); Vrettas *et al.* (2015)), and/or Gaussian processes (Archambeau *et al.* (2007); Ruttor *et al.* (2013)). We reserve for future work a detailed comparison of these methods to our method.

4.2 Statistical Method

The fundamental model considered in this chapter is

$$dX_t = f(X_t; \theta)dt + g(X_t; \theta)dW_t \quad (4.1a)$$

$$Y_t = X_t + \epsilon_t. \quad (4.1b)$$

The first part of the above system (4.1a) is a stochastic differential equation (SDE) driven by Brownian motion W_t . The second part (4.1b) models the observation process Y_t by the addition of noise ϵ_t to the state process X_t . In this work, we assume that ϵ_t is i.i.d. (independent and identically distributed) Gaussian with mean 0 and variance σ_ϵ^2 .

4.2.1 Inference Problem

Suppose that we have data of the form $\mathbf{y} = (y_0, \dots, y_L)$ where $y_j = Y_{t_j}$, the observation at time t_j . Here $t_0 < t_1 < \dots < t_L$ 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, $\mathbf{x} = (x_0, \dots, x_L)$. 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 σ_ϵ^2 of the noise term ϵ_t .

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

$$p(\mathbf{x}, \theta, \sigma_\epsilon^2 | \mathbf{y}) \propto p(\mathbf{y} | \mathbf{x}, \theta, \sigma_\epsilon^2) p(\mathbf{x}, \theta, \sigma_\epsilon^2). \quad (4.2)$$

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

It is clear from (4.1b) that \mathbf{y} is conditionally independent of θ given \mathbf{x} and σ_ϵ^2 . It is also clear from (4.1b) that the observation/state pair at time t_j is independent of all other observation/state pairs. Hence we can write

$$p(\mathbf{y}|\mathbf{x}, \theta, \sigma_\epsilon^2) = \prod_{j=0}^L p(y_{t_j} | x_j, \sigma_\epsilon^2). \quad (4.3)$$

Next, we examine the second term on the right-hand side of (4.2). It is clear that σ_ϵ^2 is independent of the other random variables, so we can write:

$$p(\mathbf{x}, \theta, \sigma_\epsilon^2) = p(\mathbf{x}, \theta) p(\sigma_\epsilon^2) = p(\mathbf{x}|\theta) p(\theta) p(\sigma_\epsilon^2). \quad (4.4)$$

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

$$p(\mathbf{x}, \theta, \sigma_\epsilon^2 | \mathbf{y}) \propto \left[\prod_{j=0}^L p(y_{t_j} | x_j, \sigma_\epsilon^2) \right] p(\mathbf{x}|\theta) p(\theta) p(\sigma_\epsilon^2). \quad (4.5)$$

From (4.1b), we have that $y_{t_j} | x_j, \sigma_\epsilon^2$ is Gaussian with mean x_j and variance σ_ϵ^2 . The terms $p(\theta)$ and $p(\sigma_\epsilon^2)$ are priors. The only other term, $p(\mathbf{x}|\theta)$, is the likelihood of θ under the model (4.1a). The detailed derivation of the likelihood computation using the DTQ method is provided in Section 3.2.1. The complete derivation of the DTQ method is given in Appendix A. The derivation of the likelihood computation is in Appendix A.1 and the gradient computation is in Appendix A.2.

4.2.2 Metropolis Algorithm

We return to the problem of sampling from the posterior (4.5). We give 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 (4.5). 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 , θ_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 old iterates with samples:

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

The log transformation ensures that $\sigma_\epsilon^2 > 0$.

- Calculate the ratio

$$\rho = \frac{p(\mathbf{x}_{i+1}^*, \theta_{i+1}^*, (\sigma_\epsilon^2)_{i+1}^* | \mathbf{y})}{p(\mathbf{x}_i, \theta_i, (\sigma_\epsilon^2)_i | \mathbf{y})}. \quad (4.6)$$

In practice, the denominator of this ratio has already been calculated at the previous step; only the numerator must be calculated. To compute the numerator, we use (4.5) together with the procedure from Section 3.2.1, including (3.9), (3.6), and (3.11).

- 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 $\mathbf{x}_{i+1} = \mathbf{x}_i$, $\theta_{i+1} = \theta_i$ and $(\sigma_\epsilon^2)_{i+1} = (\sigma_\epsilon^2)_i$

4.3 Scalable Implementation

There are two elements to our strategy of implementing the MCMC algorithm from Section 4.2 in a scalable fashion. The first aspect has to do with representing the main DTQ step (3.7) using Scala and Breeze. The second aspect has to do with using Apache Spark to evaluate each term in the product (3.2) 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.

4.3.1 Scala/Breeze

A typical approach in computational statistics to implement (3.7) 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 (3.7) can be written as

$$\mathbf{p}_{i+1} = \mathcal{G}\mathbf{p}_i. \quad (4.7)$$

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 (3.5), 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 ξ 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 (3.5) will be maximal. Similarly, we can conclude that for fixed a , when b is far from a , the exponential term in (3.5) will be negligible. We have focused on the exponential term under the assumption that the diffusion coefficient g , and hence the normalization term in (3.5), 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 $-M \leq j \leq M$ and $j' - \gamma \leq j \leq j' + \gamma$. We choose γ 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.

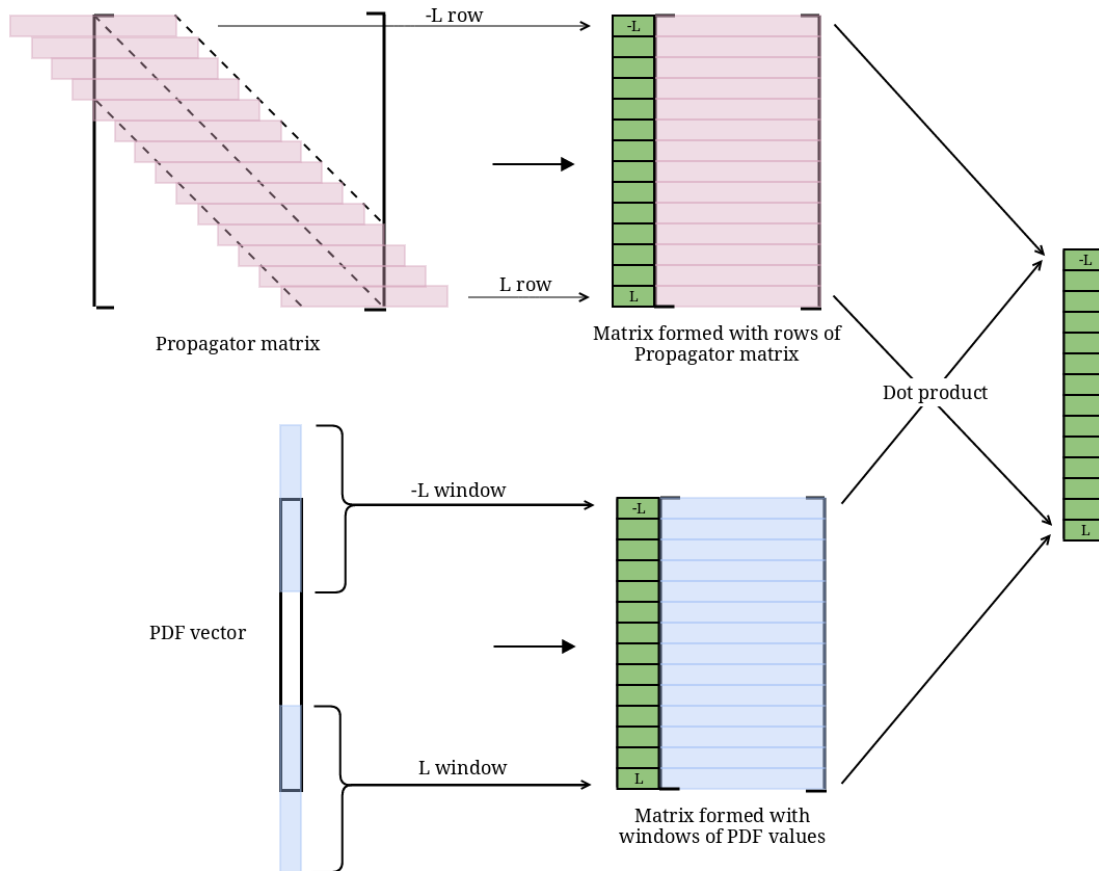


Figure 4.1: In order to implement the matrix-vector multiplication in (4.7) in a scalable way, we make use of the structure of the propagator matrix \mathcal{G} . Instead of computing all entries of this matrix, we compute and store only those entries that are close to the diagonal—the pink rectangles 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 (4.7) makes efficient use of Scala, Breeze, and the Intel MKL. For more details, see the description in Section 4.3.1.

The main DTQ step (3.7), as it is implemented in Scala using the window parameter γ , is represented graphically in Figure 4.1. 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 4.1. In the Metropolis algorithm given in Section 4.2.2, each time we must evaluate the numerator of ρ in (4.6), we must evaluate the likelihood function for a particular choice of \mathbf{x} and θ . For this choice of θ ,

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 4.1, occurs once per Metropolis step.

To implement (4.7), 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 $(p_i^{j-\gamma}, \dots, p_i^j, \dots, p_i^{j+\gamma})$. 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 4.1 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

$$\text{px} = \text{propagator dot allwins} \quad (4.8)$$

This line of code completes the implementation of (4.7). To iterate the procedure, we apply the windowing procedure—diagrammed in the lower half of Figure 4.1—to `px`, store the resulting collection in `allwins`, and then repeat (4.8).


The entire procedure diagrammed in Figure 4.1 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.

4.3.2 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 4.2. When we examine (3.2), 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 \vec{tx} in Figure 4.2, 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 θ 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.

$$\vec{tx} = \left[\begin{pmatrix} t_0 \\ x_0 \end{pmatrix}, \begin{pmatrix} t_1 \\ x_1 \end{pmatrix}, \dots, \begin{pmatrix} t_{M-1} \\ x_{M-1} \end{pmatrix}, \begin{pmatrix} t_M \\ x_M \end{pmatrix} \right]$$


$$\text{tslices} = \left\{ \left[\begin{pmatrix} t_0 \\ x_0 \end{pmatrix}, \begin{pmatrix} t_1 \\ x_1 \end{pmatrix} \right], \left[\begin{pmatrix} t_1 \\ x_1 \end{pmatrix}, \begin{pmatrix} t_2 \\ x_2 \end{pmatrix} \right], \dots, \left[\begin{pmatrix} t_{M-1} \\ x_{M-1} \end{pmatrix}, \begin{pmatrix} t_M \\ x_M \end{pmatrix} \right] \right\}$$

Figure 4.2: We use Spark to parallelize the computation of the likelihood (3.2). 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 \vec{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.

4.4 Results

We present results on artificial data sets. The model used to generate these data sets is the Ornstein-Uhlenbeck SDE together with an observation process:

$$dX_t = \theta_1(\theta_2 - X_t)dt + \theta_3 dW_t \quad (4.9)$$

$$Y_t = X_t + \epsilon_t. \quad (4.10)$$

Specifically, we apply the Euler-Maruyama discretization to (4.9) with a time step of $\kappa = 10^{-6}$. Suppose our temporal grid consists of $t_j = n(j)\kappa$, where $n(0) = 0$, and $n(j+1) > n(j)$. 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 2×10^5 , 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 κ), 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 $(x_0, x_1, \dots, x_L) =: \mathbf{x}$, and then perturb them via (4.10) to generate \mathbf{y} . In particular, we set $y_j = x_j + Z$ where Z is normally distributed with mean 0 and variance σ^2 .

The DTQ method described in Section 3.2.1 has four internal parameters: the time step h , the spatial grid spacing k , the spatial grid cutoff M , and the decay width γ 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 $\gamma = 25$.

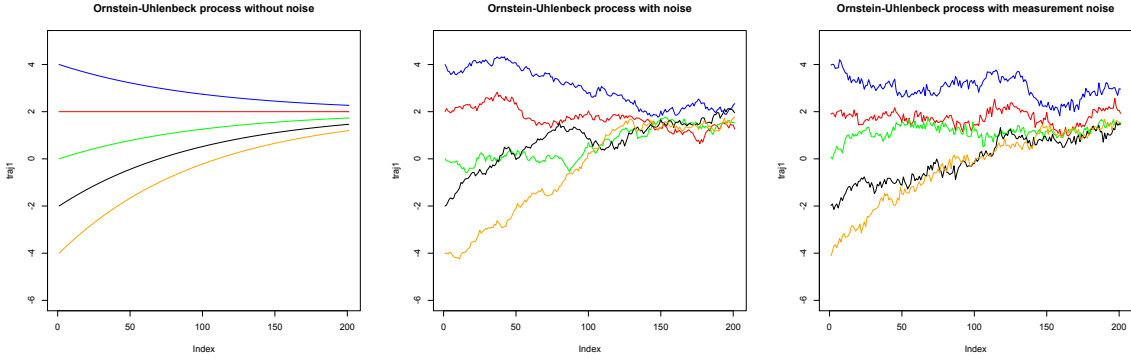


Figure 4.3: Illustration of sample paths generated for Ornstein-Uhlenbeck process in the deterministic (left), stochastic (center) and measurement noise (right) setups with the same initial conditions.

4.4.1 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 θ_1 and θ_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 θ_1 and θ_2 , we use Gaussian densities with respective parameters ($\mu = 0.5, \sigma = 1$) and ($\mu = 2, \sigma = 10$). For σ_ϵ^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_x is a collection of $L + 1$ independent Gaussians, each with parameters ($\mu = 0, \sigma = 0.02$), Z_θ consists of two independent Gaussians, each with parameters ($\mu = 0, \sigma = 0.05$), and Z_σ consists of a Gaussian with parameters ($\mu = 0, \sigma = 0.02$). These distributions have been chosen, via trial and error, to yield a Metropolis acceptance rate that is between 20 – 40% in all tests we have conducted.

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 (4.5). 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. Using the 9900 remaining samples, we plot the posterior densities for θ_1 , θ_2 and $\log_{10} \sigma_\epsilon^2$ in Figure 4.4. In this chapter, all density plots use a Gaussian kernel with the “nrd” (or normal reference rule) bandwidth Scott (2015). The density in blue (respectively, black) corresponds to the samples produced using the DTQ method with $h = 0.02$ (respectively, $h = 0.01$). The red vertical lines indicate the ground truth values for each parameter.

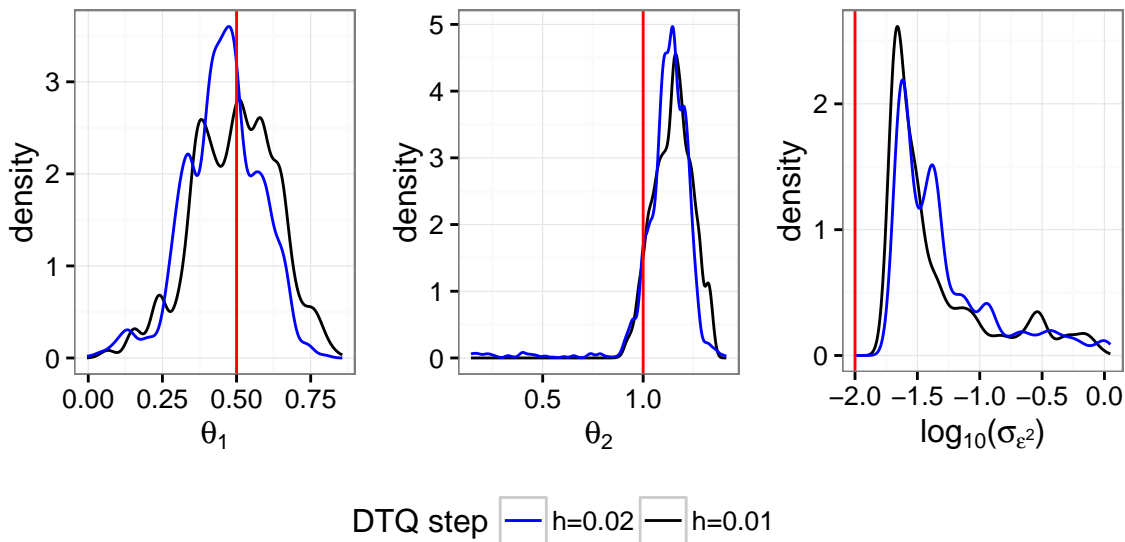


Figure 4.4: Posterior densities for the inference/filtering problem with equispaced time series (\mathbf{t}, \mathbf{y}) . Each density is calculated on the basis of 9900 post-burn-in Metropolis samples computed using the indicated value of the internal DTQ time step parameter h . Overall, we see reasonable agreement between the ground truth values (indicated by red vertical lines) and the posterior densities.

Overall, we see that the Metropolis algorithm does a reasonable job of inferring the parameters. For the Ornstein-Uhlenbeck model (4.9), Bayesian inference is non-trivial when the temporal spacing between observations is sufficiently large. Keeping in mind the logarithmic scale for the density of the third parameter, we still conclude that our method has the greatest room for improvement here. As we show below, however, the mean inferred value of σ_ϵ is consistent with the observation series \mathbf{y} and the mean inferred state series \mathbf{x} .

4.4.2 Non-equispaced Time Series

In this next set of experiments, we follow a nearly identical procedure to that described in Section 4.4.1. 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 4×10^4 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. We use the same priors and Metropolis initial conditions as in Section 4.4.1. We change the proposal distributions slightly. The parameters for $Z_{\mathbf{x}}$ and Z_σ are now $(\mu = 0, \sigma = 0.01)$, while for Z_θ the parameters are still $(\mu = 0, \sigma = 0.05)$. 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 (4.5). Again, we discard the first 100 samples as burn-in samples.

Using the samples thus obtained, and using the same procedure described in Section 4.4.1, we plot the posterior densities for θ_1 , θ_2 and $\log_{10} \sigma_\epsilon^2$ in Figure 4.5. Overall, as compared with Figure 4.4, we see improved agreement between the ground truth values and the posteriors for θ_2 and $\log_{10} \sigma_\epsilon^2$, while the posterior for θ_1 is still reasonably accurate.

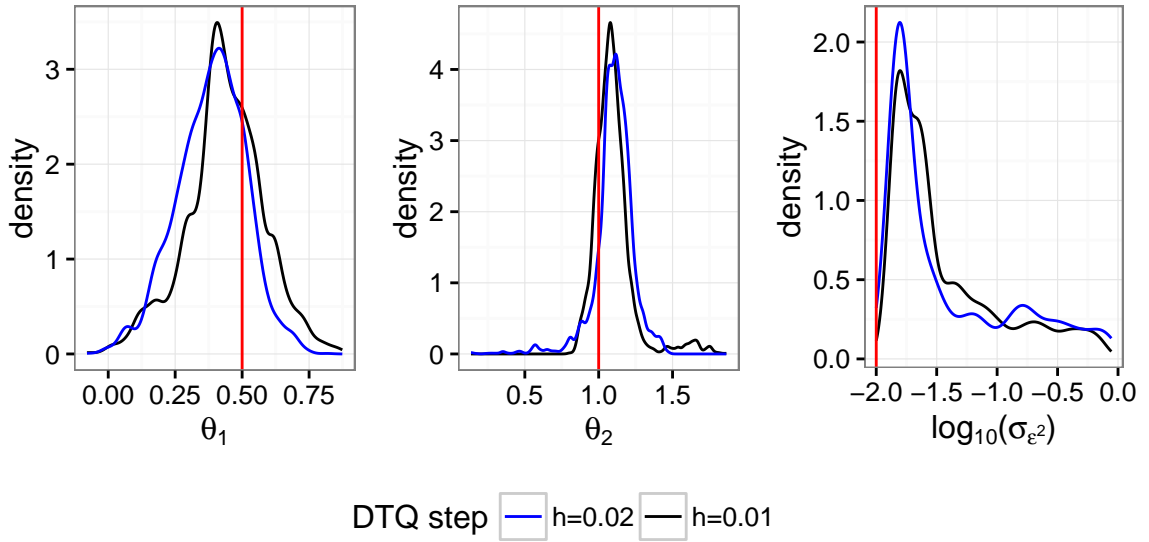


Figure 4.5: Posterior densities for the inference/filtering problem with non-equispaced time series (\mathbf{t}, \mathbf{y}) . Each density is calculated on the basis of 9900 post-burn-in Metropolis samples computed using the indicated value of the internal DTQ time step parameter h . Overall, we see reasonable agreement between the ground truth values (indicated by red vertical lines) and the posterior densities.

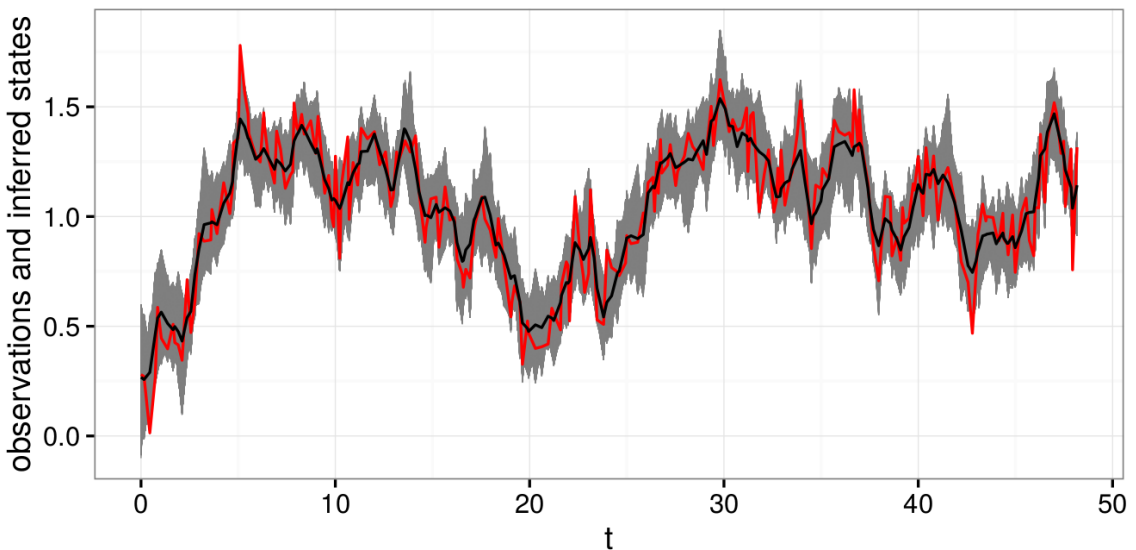


Figure 4.6: We plot the observations (in red) together with each of the samples of the state series \mathbf{x} . Each such sample is a grey curve, and the mean of all such grey curves is plotted in black. We refer to the black curve as the mean inferred state series.

Using precisely the same data, priors, and initial conditions, we used the particle MCMC method from `pomp` (King *et al.* (2016)) to produce 10000 post-burn-in samples from the posterior $p(\theta_1, \theta_2, \log \sigma_\epsilon | \mathbf{y})$. The mean of this posterior is $(0.00539, -0.765, -2.088)$. Only the final estimate is acceptable; the estimates for θ_1 and θ_2 are highly inaccurate. Here

we have given an initial guess for $x_0 = y_0 \approx 0.280$, exactly what we do in our method. If we instead provide a bit of external assistance to `pomp` and guess $x_0 = 0$, the true mean of the X_0 used to generate the artificial data, we obtain an excellent posterior mean of $(0.477, 0.973, -1.986)$. Though further testing is required, we believe these results indicate that our method may be a viable alternative to the more established methods implemented in `pomp`.

We turn to the filtering results, focusing on the post-burn-in samples of \mathbf{x} generated with DTQ parameter $h = 0.01$. In Figure 4.6, we plot the original non-equispaced observation series \mathbf{y} in red. We have plotted in grey each of the 9900 samples of the state series \mathbf{x} ; the mean of these samples is plotted in black. We see that the Metropolis sampler does indeed explore a number of different trajectories for the state series, and that the mean inferred state series corresponds to a smoothed version of the original observation series.

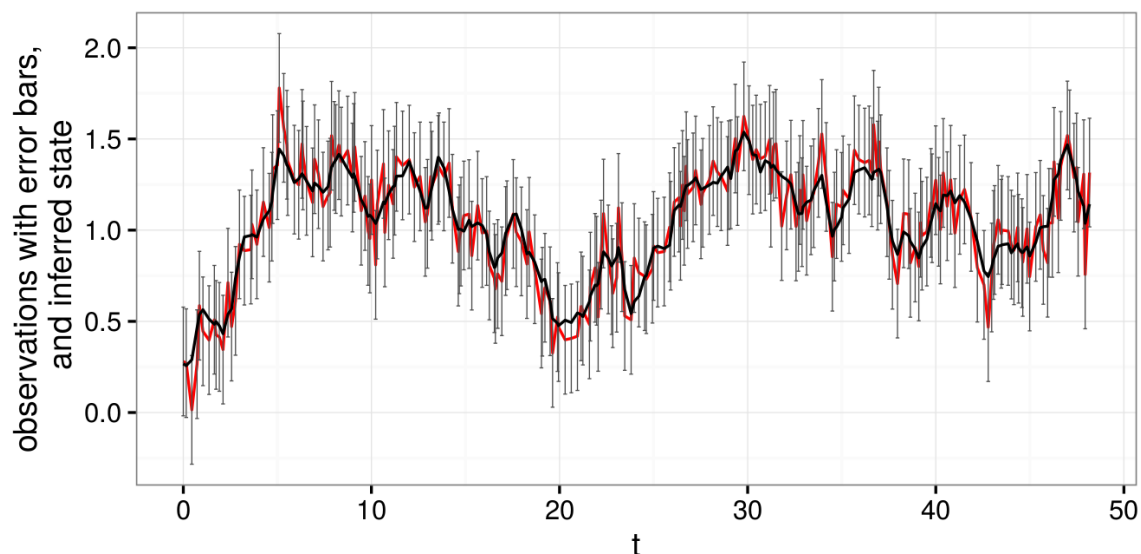


Figure 4.7: We plot the observations (in red) together with the mean inferred state series (in black). The error bars (grey) are computed by adding/subtracting the mean inferred value of σ_ϵ to/from the observation series \mathbf{y} . Note that the mean inferred state is typically within one σ_ϵ of the corresponding observation.

In Figure 4.7, we again plot the original non-equispaced observation series \mathbf{y} in red and the mean inferred state series \mathbf{x} in black. This time, however, we add/subtract the mean inferred value of σ_ϵ to \mathbf{y} to obtain error bars associated with each observation. These error bars are plotted in grey. The idea behind this plot is that if (4.1b) holds, then given the symmetry of the random variables ϵ_t , it should also be true that $X_t = Y_t + \epsilon'_t$, where ϵ'_t has the same distribution as ϵ_t . To be self-consistent, the observation series should, at least most of the time, lie within one (inferred) standard deviation σ_ϵ of the (inferred) state series. The plot in Figure 4.7 confirms that this is the case.

4.4.3 Scaling

We have conducted tests to explore the relationship between running time and L , the length of the observation series. For each $L \in \{124, 250, 500, 2500\}$, and for each $h \in \{0.02, 0.01\}$,

we have generated a time series of the indicated length, run our inference/filtering code, and recorded the amount of time T required to generate 1000 Metropolis samples of the posterior. The results are plotted in the left panel of Figure 4.8 with log-transformed axes. For each set of points, we fit a line of the form $\log T = \beta_0 + \beta_1 \log L$, which corresponds to the law $T = e^{\beta_0} L^{\beta_1}$. The solid lines in the left panel of Figure 4.8 correspond to this latter law, plotted on the same log-transformed axes. For the $h = 0.02$ line, we obtain $\beta_1 = 0.9092$; for the $h = 0.01$ line, we obtain $\beta_1 = 0.9639$. These results are consistent with $O(L)$ temporal scaling.

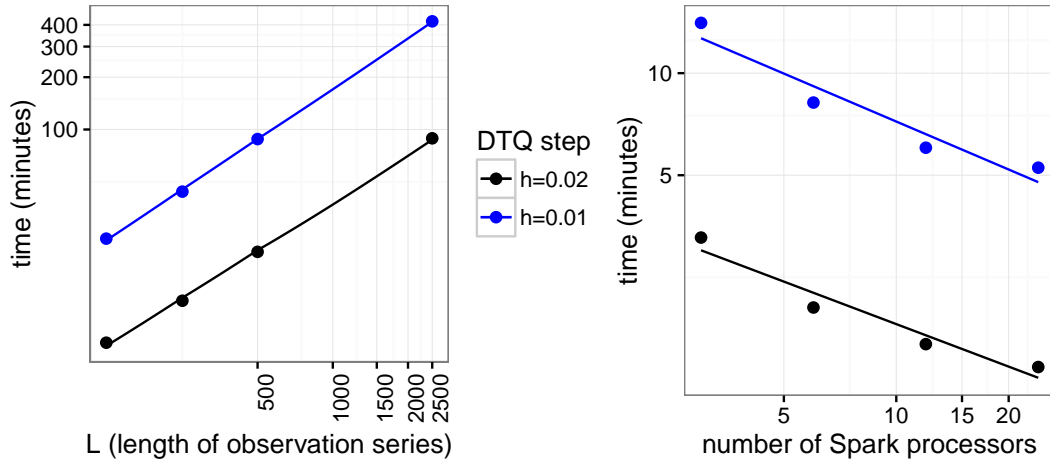


Figure 4.8: **Left panel:** For each indicated value of L , we have generated a time series of length L , run our inference/filtering code, and recorded the amount of time T required to generate 1000 Metropolis samples of the posterior. We fit lines to $\log T$ as a function of $\log L$ —both the lines and the original data are plotted on log-transformed axes. The slopes of the lines are less than 1, consistent with $O(L)$ temporal scaling. **Right panel:** For a non-equispaced time series of length 2501, we ran our code with ν Spark processors where $\nu \in \{3, 6, 12, 24\}$. We recorded T , the time required to generate 10 Metropolis samples of the posterior. We fit lines to $\log T$ as a function of $\log \nu$ —both the lines and the original data are plotted on log-transformed axes. The slopes of the lines are close to -0.5 , suggesting $O(\nu^{-1/2})$ scaling.

We have also explored how the running time of our code changes as we change the number ν of Spark processors. We begin with a non-equispaced time series of length 2501. For each $\nu \in \{3, 6, 12, 24\}$ and each $h \in \{0.02, 0.01\}$, we recorded the time T required to generate 10 Metropolis samples. For each set of points, we fit a line of the form $\log T = \beta_0 + \beta_1 \log \nu$. Both the raw data and lines are plotted on log-transformed axes in the right panel of Figure 4.8. For the $h = 0.02$ line, we obtain $\beta_1 = -0.4164$, while for the $h = 0.01$ line, we obtain $\beta_1 = -0.4699$. These results suggest $O(\nu^{-1/2})$ scaling.

4.5 Discussion

The results from Section 4.4 show that while decreasing the value of the DTQ internal step h does change the sampled values and the plotted densities, the change is not significant.

We believe much greater gains (in terms of agreement between the posterior modes and the ground truth values) would be achieved by improving both the proposal strategy and the vanilla Metropolis accept/reject step; such ideas have been pursued successfully by Fuchs (2013) and others, and we seek to implement these improvements in future work. For now, however, we conclude that the implementation described in this chapter does indeed perform Bayesian filtering and inference at a baseline acceptable level.

Aside from improving the Metropolis algorithm, we see three main areas for improvement. First, we have yet to test and tune our implementation on a large-scale, distributed Spark cluster. Second, we believe we can derive large gains in performance by adapting our algorithm to work in a streaming fashion. Specifically, instead of inferring the entire state series \mathbf{x} at once, as we currently do, we can proceed one step at a time through the temporal sequence of observations. Third, we have already begun to incorporate the DTQ method into an adjoint method suitable for computing gradients of the likelihood. This will enable us to apply techniques such as stochastic gradient descent or Hamiltonian Monte Carlo to our inference problem for either fast MLE/MAP point estimation or accelerated sampling from posteriors. These steps, and possibly others, will become necessary as we adapt our methods to multi-dimensional time series problems, a task we have already begun.

Bibliography

- Aït-Sahalia, Y. (2002). Maximum likelihood estimation of discretely sampled diffusions: A closed-form approximation approach. *Econometrica*, **70**(1), 223–262.
- Archambeau, C., Cornford, D., Opper, M., and Shawe-Taylor, J. S. (2007). Gaussian process approximations of stochastic differential equations. *Journal of machine learning research*, **1**, 1–16.
- Archambeau, C., Opper, M., Shen, Y., Cornford, D., and Shawe-Taylor, J. S. (2008). Variational inference for diffusion processes. In *Advances in Neural Information Processing Systems*, pages 17–24.
- Bally, V. and Talay, D. (1996). The law of the Euler scheme for stochastic differential equations. II. Convergence rate of the density. *Monte Carlo Methods and Applications*, **2**(2), 93–128.
- Barber, D., Cemgil, A. T., and Chiappa, S. (2011). *Bayesian Time Series Models*. Cambridge University Press.
- Batz, P., Ruttor, A., and Opper, M. (2016). Variational estimation of the drift for stochastic differential equations from the empirical density. *Journal of Statistical Mechanics: Theory and Experiment*, **2016**(8), 083404.
- Batz, P., Ruttor, A., and Opper, M. (2017). Approximate Bayes learning of stochastic differential equations. *arXiv preprint arXiv:1702.05390*.
- Bhat, H. S. and Madushani, R. W. M. A. (2016). Density tracking by quadrature for stochastic differential equations. *arXiv preprint arXiv:1610.09572*.
- Bhat, H. S. and Madushani, R. W. M. A. (2016). Nonparametric adjoint-based inference for stochastic differential equations. In *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 798–807.
- Bhat, H. S., Madushani, R., and Rawat, S. (2015). Parameter inference for stochastic differential equations with density tracking by quadrature. In *International Workshop on Simulation*, pages 99–113. Springer.
- Bhat, H. S., Madushani, R., and Rawat, S. (2016a). Bayesian inference of stochastic pursuit models from basketball tracking data. In *International Conference on Bayesian Statistics in Action*, pages 127–137. Springer.

- Bhat, H. S., Madushani, R., and Rawat, S. (2016b). Scalable SDE filtering and inference with Apache Spark. In *Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications*, pages 18–34.
- Bhattacharya, R. N. and Waymire, E. C. (2009). *Stochastic Processes with Applications*. SIAM.
- Bladt, M., Sørensen, M., *et al.* (2014). Simple simulation of diffusion bridges with application to likelihood inference for diffusions. *Bernoulli*, **20**(2), 645–675.
- Boninsegna, L., Nüske, F., and Clementi, C. (2017). Sparse learning of stochastic dynamic equations. *arXiv preprint arXiv:1712.02432*.
- Bressloff, P. C. (2014). *Stochastic Processes in Cell Biology*, volume 41. Springer.
- Brunton, S. L., Proctor, J. L., and Kutz, J. N. (2016). Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, **113**(15), 3932–3937.
- Cai, Y. (2002). Convergence theory of a numerical method for solving the Chapman–Kolmogorov equation. *SIAM Journal on Numerical Analysis*, **40**(6), 2337–2351.
- Chen, L., Jakobsen, E. R., and Næss, A. (2017a). On numerical density approximations of solutions of SDEs with unbounded coefficients. *Advances in Computational Mathematics*, pages 1–29.
- Chen, S., Shojaie, A., and Witten, D. M. (2017b). Network reconstruction from high-dimensional ordinary differential equations. *Journal of the American Statistical Association*, **112**(520), 1697–1707.
- Chen, Z. (2003). Bayesian filtering: From Kalman filters to particle filters, and beyond. *Statistics*, **182**(1), 1–69.
- Cobb, L. (1981). Stochastic differential equations for the social sciences. *Mathematical Frontiers of the Social and Policy Sciences*, Cobb, L., Thrall, M., eds, Westview Press, Boulder, co, pages 1–26.
- Cox, J. C., Ingersoll Jr, J. E., and Ross, S. A. (2005). A theory of the term structure of interest rates. In *Theory of Valuation*, pages 129–164. World Scientific.
- Cukier, R., Lakatos-Lindenberg, K., and Shuler, K. (1973). Orthogonal polynomial solutions of the Fokker-Planck equation. *Journal of Statistical Physics*, **9**(2), 137–144.
- Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (methodological)*, pages 1–38.
- Di Paola, M. and Sofi, A. (2002). Approximate solution of the Fokker–Planck–Kolmogorov equation. *Probabilistic Engineering Mechanics*, **17**(4), 369–384.
- Fuchs, C. (2013). *Inference for Diffusion Processes: With Applications in Life Sciences*. Springer Science & Business Media, Berlin.

- Ghahramani, Z. and Roweis, S. T. (1999). Learning nonlinear dynamical systems using an EM algorithm. *Advances in Neural Information Processing Systems (NIPS)*, pages 431–437.
- Hurn, A. S., Jeisman, J., and Lindsay, K. A. (2007). Seeing the wood for the trees: A critical evaluation of methods to estimate the parameters of stochastic differential equations. *Journal of Financial Econometrics*, **5**(3), 390–455.
- Iacus, S. M. (2009). *Simulation and Inference for Stochastic Differential Equations: With R Examples*. Springer Series in Statistics. Springer, New York.
- Kikuchi, K., Yoshida, M., Maekawa, T., and Watanabe, H. (1991). Metropolis Monte Carlo method as a numerical technique to solve the Fokker–Planck equation. *Chemical Physics Letters*, **185**(3-4), 335–338.
- King, A. A., Nguyen, D., and Ionides, E. L. (2016). Statistical inference for partially observed Markov processes via the R package pomp. *Journal of Statistical Software*, **69**(12), 1–43.
- Kloeden, P. E. and Platen, E. (1992). Higher-order implicit strong numerical schemes for stochastic differential equations. *Journal of Statistical Physics*, **66**(1-2), 283–314.
- Kloeden, P. E. and Platen, E. (2011). *Numerical Solution of Stochastic Differential Equations*. Springer Science & Business Media.
- Kneissler, J., Drugowitsch, J., Friston, K., and Butz, M. V. (2015). Simultaneous learning and filtering without delusions: A Bayes-optimal combination of predictive inference and adaptive filtering. *Frontiers in Computational Neuroscience*, **9**(47).
- Lande, R., Engen, S., and Sæther, B.-E. (2003). *Stochastic Population Dynamics in Ecology and Conservation*. Oxford University Press.
- Linetsky, V. (1997). The path integral approach to financial modeling and options pricing. *Computational Economics*, **11**(1-2), 129–163.
- Lynch, E. P. and Houghton, C. J. (2015). Parameter estimation of neuron models using in-vitro and in-vivo electrophysiological data. *Frontiers in Neuroinformatics*, **9**.
- Mangan, N. M., Brunton, S. L., Proctor, J. L., and Kutz, J. N. (2016). Inferring biological networks by sparse identification of nonlinear dynamics. *IEEE Transactions on Molecular, Biological and Multi-Scale Communications*, **2**(1), 52–63.
- Mangan, N. M., Kutz, J. N., Brunton, S. L., and Proctor, J. L. (2017). Model selection for dynamical systems via sparse regression and information criteria. *Proc. R. Soc. A*, **473**(2204), 20170009.
- Merton, R. C. (1976). Option pricing when underlying stock returns are discontinuous. *Journal of Financial Economics*, **3**(1-2), 125–144.
- Müller, H.-G., Yao, F., and others (2010). Empirical dynamics for longitudinal data. *The Annals of Statistics*, **38**(6), 3458–3486.

- Næss, A. and Johnsen, J. (1993). Response statistics of nonlinear, compliant offshore structures by the path integral solution method. *Probabilistic Engineering Mechanics*, **8**(2), 91–106.
- Nicolau, J. (2007). Nonparametric estimation of second-order stochastic differential equations. *Econometric Theory*, **23**(05), 880.
- Papaspiliopoulos, O. and Roberts, G. O. (2012). Importance sampling techniques for estimation of diffusion models. *Statistical Methods for Stochastic Differential Equations*, **124**, 311–340.
- Papaspiliopoulos, O., Roberts, G. O., and Stramer, O. (2013). Data augmentation for diffusions. *Journal of Computational and Graphical Statistics*, **22**(3), 665–688.
- Pedersen, A. R. (1995). A new approach to maximum likelihood estimation for stochastic differential equations based on discrete observations. *Scandinavian Journal of Statistics*, **22**(1), 55–71.
- Picchini, U. (2014). Inference for SDE models via approximate Bayesian computation. *Journal of Computational and Graphical Statistics*, **23**(4), 1080–1100.
- Quade, M., Abel, M., Kutz, J. N., and Brunton, S. L. (2018). Sparse identification of nonlinear dynamics for rapid model recovery. *arXiv preprint arXiv:1803.00894*.
- Raissi, M. and Karniadakis, G. E. (2018). Hidden physics models: Machine learning of nonlinear partial differential equations. *Journal of Computational Physics*, **357**, 125–141.
- Raissi, M., Perdikaris, P., and Karniadakis, G. E. (2017). Machine learning of linear differential equations using Gaussian processes. *Journal of Computational Physics*, **348**, 683–693.
- Roberts, G. O. and Stramer, O. (2001). On inference for partially observed nonlinear diffusion models using the Metropolis–Hastings algorithm. *Biometrika*, **88**(3), 603–621.
- Rogers, L. C. G. and Williams, D. (1994). *Diffusions, Markov Processes and Martingales: Volume 2, Itô Calculus*, volume 2. Cambridge University Press.
- Rosa-Clot, M. and Taddei, S. (2002). A path integral approach to derivative security pricing II: Numerical methods. *International Journal of Theoretical and Applied Finance*, **5**(02), 123–146.
- Rudy, S. H., Brunton, S. L., Proctor, J. L., and Kutz, J. N. (2017). Data-driven discovery of partial differential equations. *Science Advances*, **3**(4), e1602614.
- Ruttor, A., Batz, P., and Opper, M. (2013). Approximate Gaussian process inference for the drift function in stochastic differential equations. In *Advances in Neural Information Processing Systems*, pages 2040–2048.
- Santa-Clara, P. (1997). Simulated likelihood estimation of diffusions with an application to the short term interest rate. Working Paper 12-97, UCLA Anderson School of Management.

- Schaeffer, H. (2017). Learning partial differential equations via data discovery and sparse optimization. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Science*, **473**(2197), 20160446.
- Schaeffer, H., Caflisch, R., Hauck, C. D., and Osher, S. (2013). Sparse dynamics for partial differential equations. *Proceedings of the National Academy of Sciences*, **110**(17), 6634–6639.
- Schaeffer, H., Tran, G., and Ward, R. (2017). Extracting sparse high-dimensional dynamics from limited data. *arXiv:1707.08528 [math]*.
- Schenzle, A. and Brand, H. (1979). Multiplicative stochastic processes in statistical physics. *Physical Review A*, **20**(4), 1628.
- Schön, T. B., Svensson, A., Murray, L., and Lindsten, F. (2018). Probabilistic learning of nonlinear dynamical systems using sequential monte carlo. *Mechanical Systems and Signal Processing*, **104**, 866–883.
- Scott, D. W. (2015). *Multivariate Density Estimation: Theory, Practice, and Visualization*. Wiley, Hoboken, NJ, second edition.
- Shiga, T. (1981). Diffusion processes in population genetics. *Journal of Mathematics of Kyoto University*, **21**(1), 133–151.
- Skaug, C. and Næss, A. (2007). Fast and accurate pricing of discretely monitored barrier options by numerical path integration. *Computational Economics*, **30**(2), 143–151.
- Sørensen, H. (2004). Parametric inference for diffusion processes observed at discrete points in time: a survey. *International Statistical Review*, **72**(3), 337–354.
- Stuart, A. M. (2010). Inverse problems: A Bayesian perspective. *Acta Numerica*, **19**, 451–559.
- Sun, X., Jin, L., and Xiong, M. (2008). Extended Kalman filter for estimation of parameters in nonlinear state-space models of biochemical networks. *PLoS ONE*, **3**(11), 1–13.
- Svanberg, K. (2002). A class of globally convergent optimization methods based on conservative convex separable approximations. *SIAM Journal on Optimization*, **12**(2), 555–573.
- Tran, G. and Ward, R. (2017). Exact recovery of chaotic systems from highly corrupted data. *Multiscale Modeling & Simulation*, **15**(3), 1108–1129.
- van der Meulen, F., Schauer, M., and van Zanten, H. (2014). Reversible jump MCMC for nonparametric drift estimation for diffusion processes. *Computational Statistics & Data Analysis*, **71**, 615–632.
- van der Meulen, F., Schauer, M., and van Waaij, J. (2017). Adaptive nonparametric drift estimation for diffusion processes using Faber–Schauder expansions. *Statistical Inference for Stochastic Processes*, pages 1–26.
- Van Kampen, N. G. (1992). *Stochastic Processes in Physics and Chemistry*, volume 1. Elsevier.
- Verzelen, N., Tao, W., Müller, H.-G., and others (2012). Inferring stochastic dynamics from functional data. *Biometrika*, **99**(3), 533–550.

- Vrettas, M. D., Opper, M., and Cornford, D. (2015). Variational mean-field algorithm for efficient inference in large systems of stochastic differential equations. *Physical Review E*, **91**(1), 012148.
- Wehner, M. F. and Wolfer, W. (1983). Numerical evaluation of path-integral solutions to Fokker–Planck equations. *Physical Review A*, **27**(5), 2663.
- Wissel, C. (1979). Manifolds of equivalent path integral solutions of the Fokker–Planck equation. *Zeitschrift für Physik B Condensed Matter*, **35**(2), 185–191.
- Yu, J., Cai, G., and Lin, Y. (1997). A new path integration procedure based on Gauss–Legendre scheme. *International journal of non-linear mechanics*, **32**(4), 759–768.
- Zambrini, J.-C. and Yasue, K. (1980). Thermal mechanics: A quantum mechanical analogue of nonequilibrium statistical thermodynamics. *Annals of Physics*, **125**(1), 176–192.
- Øksendal, B. (2003). *Stochastic Differential Equations: An Introduction with Applications*. Universitext. Springer-Verlag, Berlin Heidelberg, 6 edition.

Appendices

Appendix A

Derivation of DTQ method

We begin by deriving our method and algorithm in the case where \mathbf{x} represents a scalar time series. Equivalently, we can say that $\nu = 1$ and the data consists of only one time-discretized sample path of (3.1). In Appendix A.1, we will show how to generalize this derivation to the case where there are $\nu > 1$ sample paths.

First we choose h , the internal time step, to be a small fraction of Δt . We set $hn = \Delta t$ where $n \in \mathbb{Z}$ and $n \geq 2$. Then:

$$\tilde{x}_{m+1/n} = \tilde{x}_m + f(\tilde{x}_m; \boldsymbol{\theta})h + g(\tilde{x}_m; \boldsymbol{\theta})\sqrt{h}Z_{m+1/n}. \quad (\text{A.1})$$

Here $\{Z_m\}$ is an i.i.d. family of Gaussian random variables with mean 0 and variance 1.

When the index m is an integer, the random variable \tilde{x}_m is intended to approximate X_{t_m} . When the index m is not an integer, \tilde{x}_m represents a random variable that *interpolates in time* between the random variables that have been sampled to give us our data.

A.1 Likelihood Computation

We are now in a position to compute the likelihood. Let us specify our notation. If A_1, \dots, A_M is a collection of random variables, then $p_{A_1, \dots, A_M}(z_1, \dots, z_M)$ denotes the joint probability density function of A_1, \dots, A_M . Conditional densities will be denoted similarly. Then the likelihood we seek to compute, the quantity we wrote as $p(\mathbf{x}|\boldsymbol{\theta})$ above, can be more accurately written as

$$p_{X_{t_M}, \dots, X_{t_0}}(x_M, \dots, x_0 | \boldsymbol{\theta}).$$

First let us use the fact that the SDE (3.1) is an Ito diffusion and therefore satisfies the strong Markov property (see Bhattacharya and Waymire (2009)). This enables us to write down our first expression for the likelihood function:

$$p_{X_{t_M}, \dots, X_{t_0}}(x_M, \dots, x_0 | \boldsymbol{\theta}) = \prod_{m=0}^{M-1} p_{X_{t_{m+1}}}(x_{m+1} | X_{t_m} = x_m; \boldsymbol{\theta}).$$

We now define the log likelihood function:

$$\mathcal{L}(\boldsymbol{\theta}) = \log p_{X_{t_M}, \dots, X_{t_0}}(x_M, \dots, x_0 | \boldsymbol{\theta}) = \sum_{m=0}^{M-1} \log p_{X_{t_{m+1}}}(x_{m+1} | X_{t_m} = x_m; \boldsymbol{\theta}). \quad (\text{A.2})$$

Now we introduce our first approximation:

$$p_{X_{t_m}} \approx p_{\tilde{x}_m}.$$

The idea is to approximate the density of X_{t_m} by the density of \tilde{x}_m . We can do the same for conditional densities, i.e.,

$$p_{X_{t_{m+1}}}(x_{m+1}|X_{t_m} = x_m; \boldsymbol{\theta}) \approx p_{\tilde{x}_{m+1}}(x_{m+1}|\tilde{x}_m = x_m; \boldsymbol{\theta}).$$

The convergence theory for the Euler-Maruyama method indicates that the L^1 norm of the difference between the left- and the right-hand sides is $\mathcal{O}(h)$ —see Bally and Talay (1996). With this approximation, we can write

$$\log p_{X_{t_M}, \dots, X_{t_0}}(x_M, \dots, x_0 | \boldsymbol{\theta}) \approx \sum_{m=0}^{M-1} \log p_{\tilde{x}_{m+1}}(x_{m+1} | \tilde{x}_m = x_m; \boldsymbol{\theta}).$$

Now we can use the density tracking by quadrature (DTQ) method to evaluate each transition density in the sum. The idea behind the DTQ method is to use quadrature to gradually evolve the density forward in time from time t_m to time t_{m+1} . For clarity in notation, we look only at the interval (t_m, t_{m+1}) . The first steps in the derivation are to introduce extra variables x_1, x_2, \dots, x_n , for each interval (t_m, t_{m+1}) , integrate them out, and then apply the Markov property recursively:

$$\begin{aligned} & p_{\tilde{x}_{m+1}}(x_{m+1} | \tilde{x}_m = x_m; \boldsymbol{\theta}) \\ &= \int_{x_{n-1}} \cdots \int_{x_1} p_{\tilde{x}_n, \tilde{x}_{n-1}, \dots, \tilde{x}_1}(x_n, x_{n-1}, \dots, x_1 | \tilde{x}_m = x_m; \boldsymbol{\theta}) dx_{n-1} \cdots dx_1 \\ &= \int_{x_{n-1}} \cdots \int_{x_1} \prod_{i=1}^n p_{\tilde{x}_i}(x_i | \tilde{x}_{i-1} = x_{i-1}; \boldsymbol{\theta}) dx_{n-1} \cdots dx_1. \end{aligned} \quad (\text{A.3})$$

The last equation is the Chapman-Kolmogorov equation for the Markov chain given by (A.1). Now let $G_{\boldsymbol{\theta}}^h(x, y)$ be the probability density function of a Gaussian random variable with mean $y + f(y; \boldsymbol{\theta})h$ and variance $g(y; \boldsymbol{\theta})^2 h$, evaluated at x . Then the crucial observation is that, for each $i \in \{1, \dots, n\}$,

$$p_{\tilde{x}_i}(x_i | \tilde{x}_{i-1} = x_{i-1}; \boldsymbol{\theta}) = G_{\boldsymbol{\theta}}^h(x_i, x_{i-1}). \quad (\text{A.4})$$

This follows from the discrete-time evolution equation for \tilde{x} . With these observations, the (A.3) simplifies to:

$$p_{\tilde{x}_{m+1}}(x_{m+1} | \tilde{x}_m = x_m; \boldsymbol{\theta}) = \int_{x_{n-1}} G_{\boldsymbol{\theta}}^h(x_n, x_{n-1}) \int_{x_{n-2}} \cdots \left[\int_{x_1} G_{\boldsymbol{\theta}}^h(x_2, x_1) G_{\boldsymbol{\theta}}^h(x_1, x_0) dx_1 \right] dx_2 \cdots dx_{n-1}. \quad (\text{A.5})$$

We constrain the approximation at the first time point x_0 to be equal to the actual observation, x_m . Also the density at the final time point, $t_n = t_{m+1}$, is evaluated at the observation x_{m+1} . Our next approximation is to evaluate the integrals by quadrature. We introduce the spatial grid spacing $k > 0$. We will use superscripts to denote spatial grid locations, so that,

for instance, $x_i^j = jk$ for all $j \in \mathbb{Z}$. Then, repeatedly applying the trapezoidal rule on the real line, we obtain

$$p_{\tilde{x}_{m+1}}(x_{m+1} | \tilde{x}_m = x_m; \boldsymbol{\theta}) \approx k \sum_{j_{n-1}} G_{\boldsymbol{\theta}}^h(x_{m+1}, x_{j_{n-1}}^{j_{n-1}}) \cdot k \sum_{j_{n-2}} G_{\boldsymbol{\theta}}^h(x_{j_{n-1}}^{j_{n-1}}, x_{j_{n-2}}^{j_{n-2}}) \cdots \cdot k \sum_{j_1} G_{\boldsymbol{\theta}}^h(x_2^{j_2}, x_1^{j_1}) G_{\boldsymbol{\theta}}^h(x_1^{j_1}, x_m).$$

In practice, we evaluate these sums on a finite subset of \mathbb{Z} . We think of

$$k G_{\boldsymbol{\theta}}^h(x_2^{j_2}, x_1^{j_1})$$

as the (j_2, j_1) element of a matrix \mathbf{A} . In this way, the above formula reduces to repeated matrix-vector multiplication. Specifically, let us define the j_1 -th element of the vector \mathbf{q}_1 by

$$\mathbf{q}_1^{j_1} = G_{\boldsymbol{\theta}}^h(x_1^{j_1}, x_m).$$

Then multiplication by the matrix \mathbf{A} corresponds to stepping forward in time by h , i.e.,

$$\mathbf{q}_2 = \mathbf{A} \mathbf{q}_1$$

and

$$\mathbf{q}_{n-1} = \mathbf{A}^{n-2} \mathbf{q}_1.$$

Finally, noting that x_{m+1} is a known data point, let us define the j_{n-1} -th element of the vector $\boldsymbol{\Gamma}_{n-1}$ by

$$\boldsymbol{\Gamma}_{n-1}^{j_{n-1}} = k G_{\boldsymbol{\theta}}^h(x_{m+1}, x_{j_{n-1}}^{j_{n-1}}).$$

Then we have

$$p_{\tilde{x}_{m+1}}(x_{m+1} | \tilde{x}_m = x_m; \boldsymbol{\theta}) \approx [\boldsymbol{\Gamma}_{n-1}]^\top \mathbf{A}^{n-2} \mathbf{q}_1, \quad (\text{A.6})$$

where $^\top$ denotes transpose. As noted above, the vector \mathbf{q}_1 only depends on the first observation x_m and the vector $\boldsymbol{\Gamma}_{n-1}$ only depends on the final observation x_{m+1} in the interval (t_m, t_{m+1}) . For the general case, we can thus define $\mathbf{q}_m := \mathbf{q}_1$ and $\boldsymbol{\Gamma}_{m+1} := \boldsymbol{\Gamma}_{n-1}$ for each interval. We insert this computation into (A.2) to obtain

$$\mathcal{L}(\boldsymbol{\theta}) \approx \sum_{m=0}^{M-1} \log [\boldsymbol{\Gamma}_{m+1}]^\top \mathbf{A}^{n-2} \mathbf{q}_m. \quad (\text{A.7})$$

A.2 Gradient Computation

Next, we compute the gradient of the log likelihood with respect to $\boldsymbol{\theta}$. This gradient is an important ingredient for numerical optimization procedures. We start with

$$\begin{aligned} \frac{\partial}{\partial \theta_\ell} \mathcal{L}(\boldsymbol{\theta}) &= \frac{\partial}{\partial \theta_\ell} \log p_{X_{t_M}, \dots, X_{t_0}}(x_M, \dots, x_0 | \boldsymbol{\theta}) \\ &\approx \sum_{m=0}^{M-1} \frac{1}{p_{\tilde{x}_{m+1}}(x_{m+1} | \tilde{x}_m = x_m; \boldsymbol{\theta})} \frac{\partial}{\partial \theta_\ell} p_{\tilde{x}_{m+1}}(x_{m+1} | \tilde{x}_m = x_m; \boldsymbol{\theta}). \end{aligned} \quad (\text{A.8})$$

The remaining derivative looks like this:

$$\begin{aligned} \frac{\partial}{\partial \theta_\ell} p_{\tilde{x}_{m+1}}(x_{m+1} | \tilde{x}_m = x_m; \boldsymbol{\theta}) &= \int_{x_{n-1}} \cdots \int_{x_1} \sum_{r=0}^{n-1} \left\{ \frac{\partial}{\partial \theta_\ell} p_{\tilde{x}_{r+1}}(x_{r+1} | \tilde{x}_r = x_r; \boldsymbol{\theta}) \right. \\ &\quad \left. \times \prod_{\substack{s=r \\ s=0}}^{n-1} p_{\tilde{x}_{s+1}}(x_{s+1} | \tilde{x}_s = x_s; \boldsymbol{\theta}) \right\} dx_{n-1} \cdots dx_1. \end{aligned}$$

Let us simplify the notation and derive an algorithm to compute this quantity. First, we peel off the $r = n - 1$ term in the sum to write:

$$\begin{aligned} \frac{\partial}{\partial \theta_\ell} p(x_{m+1} | x_m; \boldsymbol{\theta}) &= \int_{x_{n-1}} \cdots \int_{x_1} \left\{ \frac{\partial}{\partial \theta_\ell} p(x_n | x_{n-1}; \boldsymbol{\theta}) \prod_{s=0}^{n-2} p(x_{s+1} | x_s; \boldsymbol{\theta}) \right\} \\ &\quad + \left[p(x_n | x_{n-1}; \boldsymbol{\theta}) \sum_{r=0}^{n-2} \left(\frac{\partial}{\partial \theta_\ell} p(x_{r+1} | x_r; \boldsymbol{\theta}) \prod_{\substack{s=r \\ s=0}}^{n-2} p(x_{s+1} | x_s; \boldsymbol{\theta}) \right) \right] dx_{n-1} \cdots dx_1. \end{aligned}$$

Again, we can use the definition of $G_\boldsymbol{\theta}^h$ together with the crucial observation described above to simplify the above expression to:

$$\begin{aligned} \frac{\partial}{\partial \theta_\ell} p(x_{m+1} | x_m; \boldsymbol{\theta}) &= \int_{\mathbf{x} \in \mathbb{R}^{n-1}} \left\{ \frac{\partial}{\partial \theta_\ell} G_\boldsymbol{\theta}^h(x_n, x_{n-1}) \prod_{s=0}^{n-2} G_\boldsymbol{\theta}^h(x_{s+1}, x_s) \right\} \\ &\quad + \left[G_\boldsymbol{\theta}^h(x_n, x_{n-1}) \sum_{r=0}^{n-2} \left(\frac{\partial}{\partial \theta_\ell} G_\boldsymbol{\theta}^h(x_{r+1}, x_r) \prod_{\substack{s=r \\ s=0}}^{n-2} G_\boldsymbol{\theta}^h(x_{s+1}, x_s) \right) \right] d\mathbf{x}. \end{aligned}$$

Let us now discretize in space, again using the trapezoidal rule on the real line repeatedly just as we did earlier:

$$\begin{aligned} \frac{\partial}{\partial \theta_\ell} p(x_{m+1} | x_m; \boldsymbol{\theta}) &\approx k^{n-1} \sum_{j_{n-1}} \cdots \sum_{j_1} \left\{ \frac{\partial}{\partial \theta_\ell} G_\boldsymbol{\theta}^h(x_n, x_{n-1}^{j_{n-1}}) \prod_{s=0}^{n-2} G_\boldsymbol{\theta}^h(x_{s+1}^{j_{s+1}}, x_s^{j_s}) \right\} \\ &\quad + \left[G_\boldsymbol{\theta}^h(x_n, x_{n-1}^{j_{n-1}}) \sum_{r=0}^{n-2} \left(\frac{\partial}{\partial \theta_\ell} G_\boldsymbol{\theta}^h(x_{r+1}^{j_{r+1}}, x_r^{j_r}) \prod_{\substack{s=r \\ s=0}}^{n-2} G_\boldsymbol{\theta}^h(x_{s+1}^{j_{s+1}}, x_s^{j_s}) \right) \right]. \end{aligned}$$

In the above expression and in what follows, any instance of $x_m^{j_m}$ should be interpreted as simply x_m . Now let us push all summations over j_1, \dots, j_{n-2} inside to obtain

$$\begin{aligned} \frac{\partial}{\partial \theta_\ell} p(x_{m+1} | x_m; \boldsymbol{\theta}) &\approx k \sum_{j_{n-1}} \left\{ \frac{\partial}{\partial \theta_\ell} G_\boldsymbol{\theta}^h(x_n, x_{n-1}^{j_{n-1}}) \left(k^{n-2} \sum_{j_{n-2}} \cdots \sum_{j_1} \prod_{s=0}^{n-2} G_\boldsymbol{\theta}^h(x_{s+1}^{j_{s+1}}, x_s^{j_s}) \right) \right\} \\ &\quad + \left[G_\boldsymbol{\theta}^h(x_n, x_{n-1}^{j_{n-1}}) k^{n-2} \sum_{j_{n-2}} \cdots \sum_{j_1} \sum_{r=0}^{n-2} \left(\frac{\partial}{\partial \theta_\ell} G_\boldsymbol{\theta}^h(x_{r+1}^{j_{r+1}}, x_r^{j_r}) \prod_{\substack{s=r \\ s=0}}^{n-2} G_\boldsymbol{\theta}^h(x_{s+1}^{j_{s+1}}, x_s^{j_s}) \right) \right]. \end{aligned}$$

Now note that by our previous definitions, we have that

$$k^{n-2} \sum_{j_{n-2}} \cdots \sum_{j_1} \prod_{s=0}^{F-2} G_{\boldsymbol{\theta}}^h(x_{s+1}^{j_{s+1}}, x_s^{j_s}) = \mathbf{A}^{n-2} \mathbf{q}_1 = \mathbf{q}_{n-1}.$$

In an analogous fashion, let us define the j_{n-1} -th element of the gradient vector $\mathbf{q}_{n-1, \ell}$ by

$$\mathbf{q}_{n-1, \ell}^{j_{n-1}} = k^{n-2} \sum_{j_{n-2}} \cdots \sum_{j_1} \sum_{r=0}^{n-2} \left(\frac{\partial}{\partial \theta_{\ell}} G_{\boldsymbol{\theta}}^h(x_{r+1}^{j_{r+1}}, x_r^{j_r}) \prod_{\substack{s \neq r \\ s=0}}^{n-2} G_{\boldsymbol{\theta}}^h(x_{s+1}^{j_{s+1}}, x_s^{j_s}) \right). \quad (\text{A.9})$$

Let us define the j_{n-1} -th element of the vector $\boldsymbol{\Gamma}_{n-1, \ell}$ by

$$\boldsymbol{\Gamma}_{n-1, \ell}^{j_{n-1}} = k \frac{\partial}{\partial \theta_{\ell}} G_{\boldsymbol{\theta}}^h(x_{m+1}, x_{n-1}^{j_{n-1}}).$$

Using this together with our old definition of $\boldsymbol{\Gamma}_{n-1}$, we have

$$\frac{\partial}{\partial \theta_{\ell}} p_{\tilde{x}_{m+1}}(x_{m+1} | \tilde{x}_m = x_m; \boldsymbol{\theta}) \approx [\boldsymbol{\Gamma}_{n-1, \ell}]^{\top} \mathbf{q}_{n-1} + [\boldsymbol{\Gamma}_{n-1}]^{\top} \mathbf{q}_{n-1, \ell}.$$

Let us now define the (j_{r+1}, j_r) element of the matrix \mathbf{A}_{ℓ} by

$$\mathbf{A}_{\ell}^{j_{r+1}, j_r} = k \frac{\partial}{\partial \theta_{\ell}} G_{\boldsymbol{\theta}}^h(x_{r+1}^{j_{r+1}}, x_r^{j_r}).$$

Then let us return to (A.9). By peeling off the $r = n-2$ term, we can derive:

$$\mathbf{q}_{n-1, \ell} = \mathbf{A}_{\ell} \mathbf{q}_{n-2} + \mathbf{A} \mathbf{q}_{n-2, \ell}$$

where $\mathbf{q}_{n-2, \ell}$ is defined analogously to $\mathbf{q}_{n-1, \ell}$. It is clear that after a finite number of such manipulations, we will reach the $r = 0$ term. In this case, the product term will be empty (and hence equal 1), leaving us with only the derivative with respect to θ_{ℓ} of $G_{\boldsymbol{\theta}}^h(x_1^{j_1}, x_m)$. In this way, we may derive the following algorithm:

1. We begin with

$$\mathbf{q}_{1, \ell}^{j_1} = \frac{\partial}{\partial \theta_{\ell}} G_{\boldsymbol{\theta}}^h(x_1^{j_1}, x_m).$$

2. We then iteratively define, for $r = 1, \dots, n-2$,

$$\mathbf{q}_{r+1, \ell} = \mathbf{A}_{\ell} \mathbf{q}_r + \mathbf{A} \mathbf{q}_{r, \ell}. \quad (\text{A.10})$$

3. We finish with:

$$\frac{\partial}{\partial \theta_{\ell}} p_{\tilde{x}_{m+1}}(x_{m+1} | \tilde{x}_m = x_m; \boldsymbol{\theta}) \approx [\boldsymbol{\Gamma}_{m+1, \ell}]^{\top} \mathbf{q}_{n-1} + [\boldsymbol{\Gamma}_{m+1}]^{\top} \mathbf{q}_{n-1, \ell}.$$

Using this together with (A.6) in (A.8), we obtain

$$\frac{\partial}{\partial \theta_\ell} \mathcal{L}(\boldsymbol{\theta}) \approx \sum_{m=0}^{M-1} \frac{[\boldsymbol{\Gamma}_{m+1, \ell}]^\top \mathbf{q}_{n-1} + [\boldsymbol{\Gamma}_{m+1}]^\top \mathbf{q}_{n-1, \ell}}{[\boldsymbol{\Gamma}_{m+1}]^\top \mathbf{A}^{n-2} \mathbf{q}_1}. \quad (\text{A.11})$$

A.3 Multiple sample paths

Before proceeding, let us revisit (A.5). We can write

$$p_{\tilde{x}_1}(x_1 | \tilde{x}_m = x_m; \boldsymbol{\theta}) = G_{\boldsymbol{\theta}}^h(x_1, x_m) = \int_y G_{\boldsymbol{\theta}}^h(x_1, y) \delta(y - x_m) dy. \quad (\text{A.12})$$

The term on the right-hand side can be interpreted as evolving the initial density $p_{\tilde{x}_m}(y) = \delta(y - x_m)$ forward by h units of time. We note that conditioning on $\tilde{x}_m = x_m$ on the left-hand side leads to a Dirac delta initial density on the right-hand side. This will be an important ingredient in the algorithm that follows.

A.3.1 Likelihood Computation

Now let us reinterpret $\mathbf{x} = x_0, x_1, \dots, x_M$ as a sequence of vector-valued observations. For each $s = 1, 2, \dots, \nu$, the sequence $x_0^s, x_1^s, \dots, x_M^s$ is a scalar time series. With these changes, the derivation of the log likelihood from (A.2) to (A.3) holds without any changes.

The only real change is that (A.4) only holds for $i \in \{2, \dots, n\}$. When $i = 1$, the quantity that must be computed is:

$$p_{\tilde{x}_1}(x_1 | \tilde{x}_m = x_m; \boldsymbol{\theta}), \quad (\text{A.13})$$

where we have many samples $\{x_m^s\}_{s=1}^\nu$ of the random variable \tilde{x}_m . When $\nu > 1$, these samples can be used to estimate the density of \tilde{x}_m as follows:

$$p_{\tilde{x}_m}(y) \approx \frac{1}{\nu} \sum_{s=1}^{\nu} \delta(y - x_m^s). \quad (\text{A.14})$$

This approximation is a density estimate that corresponds to the spatial derivative of the empirical cumulative distribution function of the samples. By logic analogous to (A.12) and the above discussion, we can then evaluate (A.13) by

$$\begin{aligned} p_{\tilde{x}_1}(x_1 | \tilde{x}_m = x_m; \boldsymbol{\theta}) &= \int_y G_{\boldsymbol{\theta}}^h(x_1, y) p_{\tilde{x}_m}(y) dy \\ &\approx \frac{1}{\nu} \sum_{s=1}^{\nu} G_{\boldsymbol{\theta}}^h(x_1, x_m^s). \end{aligned} \quad (\text{A.15})$$

We make the approximation (A.14) so that the density along each sample path evolves with the same initial condition. Without such an approximation, the calculation (A.7) would have to be repeated ν times.

The calculation of the likelihood now proceeds just as in (A.3) with $G_{\boldsymbol{\theta}}^h(x_1, x_m)$ replaced by (A.15). We now redefine \mathbf{q}_1 such that its j_1 -th element is (A.15) evaluated at $x_1 = x_1^{j_1}$.

We also redefine $\mathbf{\Gamma}_{m+1} := \mathbf{\Gamma}_{n-1}$ to be a matrix whose (j_n, j_{n-1}) entry is

$$\mathbf{\Gamma}_{m+1}^{j_n, j_{n-1}} = k G_{\boldsymbol{\theta}}^h(x_{m+1}^{j_n}, x_{n-1}^{j_{n-1}}).$$

With these definitions, (A.6) becomes

$$p_{\tilde{x}_{m+1}}(x_{m+1} | \tilde{x}_m = x_m; \boldsymbol{\theta}) \approx \prod_{j_n=1}^v ([\mathbf{\Gamma}_{m+1}]^\top \mathbf{A}^{n-2} \mathbf{q}_m)_{j_n}, \quad (\text{A.16})$$

where $(\cdot)_s$ denotes the s -th component of the vector in parentheses. Similarly, (A.7) becomes

$$\mathcal{L}(\boldsymbol{\theta}) \approx \sum_{m=0}^{M-1} \sum_{j_n=1}^v \log([\mathbf{\Gamma}_{m+1}]^\top \mathbf{A}^{n-2} \mathbf{q}_m)_{j_n}. \quad (\text{A.17})$$

A.3.2 Gradient Computation

The derivation of the gradient of $\mathcal{L}(\boldsymbol{\theta})$, as given in Section A.3.1, now proceeds just as before with $G_{\boldsymbol{\theta}}^h(x_1, x_m)$ replaced by (A.15). The only changes required in the algorithm are, first, to redefine

$$\mathbf{q}_{1,\ell}^{j_1} = \frac{1}{v} \sum_{s=1}^v \frac{\partial}{\partial \theta_\ell} G_{\boldsymbol{\theta}}^h(x_1, x_m^s)$$

and, second, to redefine $\mathbf{\Gamma}_{m+1,\ell}$ as a matrix whose (j_n, j_{n-1}) entry is

$$\mathbf{\Gamma}_{m+1,\ell}^{j_n, j_{n-1}} = k \frac{\partial}{\partial \theta_\ell} G_{\boldsymbol{\theta}}^h(x_{m+1}^{j_n}, x_{n-1}^{j_{n-1}}).$$

With these changes, the gradient becomes

$$\frac{\partial}{\partial \theta_\ell} \mathcal{L}(\boldsymbol{\theta}) \approx \sum_{m=0}^{M-1} \sum_{j_n=1}^v \frac{([\mathbf{\Gamma}_{m+1,\ell}]^\top \mathbf{q}_{n-1} + [\mathbf{\Gamma}_{m+1}]^\top \mathbf{q}_{n-1,\ell})_{j_n}}{([\mathbf{\Gamma}_{m+1}]^\top \mathbf{A}^{n-2} \mathbf{q}_m)_{j_n}}, \quad (\text{A.18})$$

where \mathbf{q} is computed using (A.10) just as before.

A.4 Adjoint based Gradient Computation

In Section 3.2.2, we develop an algorithm for the direct computation of the gradient of the negative log likelihood function

$$-\log L(\boldsymbol{\theta}) = - \sum_{m=0}^{M-1} \log p(x_{m+1} | x_m; \boldsymbol{\theta}), \quad (\text{A.19})$$

where $p_{X_{t_{m+1}}}(x_{m+1} | X_{t_m} = x_m; \boldsymbol{\theta})$ is the conditional density of $X_{t_{m+1}} = x_{m+1}$ given $X_{t_m} = x_m$. We compute this conditional density using the DTQ method in (3.11):

$$p(x_{m+1} | x_m; \boldsymbol{\theta}) = [\mathbf{\Gamma}_{m+1}]^\top \mathbf{A}^{n-2} \mathbf{q}_m, \quad (\text{A.20})$$

where \mathbf{q}_m represents the first step, \mathbf{A}^{n-2} represents the next $n-2$ steps, and $\mathbf{\Gamma}_{m+1}^\top$ is the last step. Suppose that $\boldsymbol{\theta} \in \mathbb{R}^N$. Using this, we can obtain an evolution equation for $(\nabla p)_\ell, 1 \leq \ell \leq N$ using the direct method

$$\begin{aligned} \frac{\partial L(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}_\ell} &= \frac{\partial [\mathbf{\Gamma}_{m+1}]^\top}{\partial \boldsymbol{\theta}_\ell} \mathbf{A}^{n-2} \mathbf{q}_m + [\mathbf{\Gamma}_{m+1}]^\top \frac{\partial (\mathbf{A}^{n-2})}{\partial \boldsymbol{\theta}_\ell} \mathbf{q}_m + [\mathbf{\Gamma}_{m+1}]^\top \mathbf{A}^{n-2} \frac{\partial \mathbf{q}_m}{\partial \boldsymbol{\theta}_\ell} \\ &= [\mathbf{\Gamma}_{m+1, \ell}]^\top \mathbf{A}^{n-2} \mathbf{q}_m + [\mathbf{\Gamma}_{m+1}]^\top \mathbf{A}_\ell \mathbf{q}_m + [\mathbf{\Gamma}_{m+1}]^\top \mathbf{A}^{n-2} \mathbf{q}_{m, \ell}. \end{aligned} \quad (\text{A.21})$$

This is the direct method for computing the gradients. It requires computation of N different $(2L+1) \times (2L+1)$ matrices, \mathbf{A}_ℓ . To avoid this computation, we use the adjoint method developed by Bhat and Madushani (2016). Let us revisit the computation for the DTQ method. On a given inter-observation time interval (t_m, t_{m+1}) we consider a discretized temporal grid, $\tau_0 = t_m, \tau_n = t_{m+1}, \tau_i = t_m + ih$ for $0 \leq i \leq n$ and grid spacing $h = (t_{m+1} - t_m)/n$. We have the transition densities on the intermediate points in this interval as $\mathbf{q}_1, \dots, \mathbf{q}_{n-1}$ using (3.9), (3.10) and (3.11). We introduce M additional new variables, \mathbf{p}_m for $m = 1, \dots, M$, called *state variables*. They denote the transition density $p_{X_{t_{m+1}}}(z_j | X_{t_m} = x_m; \boldsymbol{\theta})$ at observation times t_m on a discretized spatial grid $\{z_j\}_{j=-L}^L$. The collection of all these state variables is $\mathbf{p} = (\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_M)$ for $m = 1, \dots, M$. These are the unknown transition densities that we want to compute. To help compute these densities, we introduce a function, ϕ , dependent solely on \mathbf{p} ,

$$\phi(\mathbf{p}) = \sum_{m=1}^M \phi(\mathbf{p}_m). \quad (\text{A.22})$$

This function plays the role of the negative likelihood function defined in (A.19). Our aim is to take a total derivative of the negative log likelihood function with respect to the $\boldsymbol{\theta}$ vector. We have removed the explicit dependence of the likelihood function on $\boldsymbol{\theta}$ using the ϕ function. The gradient we seek can thus be written with the differential operator, $D_{\boldsymbol{\theta}}$, as

$$D_{\boldsymbol{\theta}}[\phi(\mathbf{p})] = \sum_m D_{\boldsymbol{\theta}}[\phi_m(\mathbf{p}_m)] \quad (\text{A.23})$$

$$= \sum_m D_{\mathbf{p}}[\phi_m(\mathbf{p}_m)] \cdot D_{\boldsymbol{\theta}}[\mathbf{p}_m]. \quad (\text{A.24})$$

In the direct method for the gradient computation in (A.21), we see that $D_{\boldsymbol{\theta}}[\mathbf{p}_m]$ is the computationally intense part. To avoid this computation we introduce a function $\boldsymbol{\gamma}_m$ that implicitly relates \mathbf{p} to $\boldsymbol{\theta}$, the parameter vector such that

$$\boldsymbol{\gamma}_m(\mathbf{p}_m, \boldsymbol{\theta}) \equiv \mathbf{0} \implies D_{\boldsymbol{\theta}}[\boldsymbol{\gamma}_m(\mathbf{p}_m, \boldsymbol{\theta})] \equiv \mathbf{0} \quad (\text{A.25})$$

$$\implies \underbrace{D_{\mathbf{p}_m}[\boldsymbol{\gamma}_m(\mathbf{p}_m, \boldsymbol{\theta})]}_{\text{Matrix}} \cdot \underbrace{D_{\boldsymbol{\theta}}[\mathbf{p}_m]}_{\text{Vector}} + \underbrace{D_{\boldsymbol{\theta}}[\boldsymbol{\gamma}_m(\mathbf{p}_m, \boldsymbol{\theta})]}_{\text{Vector}} \equiv \mathbf{0}. \quad (\text{A.26})$$

We can solve this linear system to find the required $D_{\boldsymbol{\theta}}$ vector:

$$D_{\boldsymbol{\theta}} \mathbf{p}_m = -(D_{\mathbf{p}_m}[\boldsymbol{\gamma}_m(\mathbf{p}_m, \boldsymbol{\theta})])^{-1} D_{\boldsymbol{\theta}}[\boldsymbol{\gamma}_m(\mathbf{p}_m, \boldsymbol{\theta})]. \quad (\text{A.27})$$

This gives us the complete derivative $D_{\boldsymbol{\theta}}[\phi(\mathbf{p})]$ as

$$D_{\boldsymbol{\theta}}[\phi(\mathbf{p})] = - \sum_m D_{\mathbf{p}}[\phi_m(\mathbf{p}_m)] \cdot (D_{\mathbf{p}_m}[\boldsymbol{\gamma}_m(\mathbf{p}_m, \boldsymbol{\theta})])^{-1} D_{\boldsymbol{\theta}}[\boldsymbol{\gamma}_m(\mathbf{p}_m, \boldsymbol{\theta})]. \quad (\text{A.28})$$

We can rewrite this system using a set of Lagrange multipliers, $\{\boldsymbol{\lambda}_m\}_{m=1}^M$

$$\boldsymbol{\lambda}_m^\top = D_{\mathbf{p}}[\phi_m(\mathbf{p}_m)] \cdot (D_{\mathbf{p}_m}[\boldsymbol{\gamma}_m(\mathbf{p}_m, \boldsymbol{\theta})])^{-1}. \quad (\text{A.29})$$

We can now write the gradient equation and the adjoint equation respectively as,

$$D_{\boldsymbol{\theta}}[\phi(\mathbf{p})] = \sum_m \boldsymbol{\lambda}_m^\top D_{\boldsymbol{\theta}}[\boldsymbol{\gamma}_m(\mathbf{p}_m, \boldsymbol{\theta})], \quad (\text{A.30a})$$

$$(D_{\mathbf{p}_m}[\boldsymbol{\gamma}_m(\mathbf{p}_m, \boldsymbol{\theta})])^\top \boldsymbol{\lambda}_m = -(D_{\mathbf{p}_m}[\phi_m(\mathbf{p}_m, \boldsymbol{\theta})]). \quad (\text{A.30b})$$

Solving the linear system for the adjoint equation will help us solve for the required gradient.

Next, we present how the adjoint method is applicable with the computation of the DTQ method. The iterative step of the DTQ method $\mathbf{q}_{i+1} = \mathbf{A}\mathbf{q}_i$ can be written as a system of block matrices to compute $\mathbf{p}_{m+1} = (\mathbf{q}_1^\top, \dots, \mathbf{q}_{n-1}^\top)^\top$ as follows:

$$\mathbf{K}\mathbf{p}_{m+1} = \mathbf{v}_{m+1}, \quad (\text{A.31})$$

where $\mathbf{v}_{m+1} = (\mathbf{q}_1^\top, \dots, \mathbf{0})^\top$ and

$$\mathbf{K} = \begin{pmatrix} \mathbf{I} & & & & \mathbf{O} \\ -\mathbf{A} & \mathbf{I} & & & \\ & -\mathbf{A} & \mathbf{I} & & \\ & & \ddots & \ddots & \\ \mathbf{O} & & & -\mathbf{A} & \mathbf{I} \end{pmatrix}. \quad (\text{A.32})$$

Here $\mathbf{0}$ denotes a zero row vector of the same length as \mathbf{q}_i . Also, \mathbf{O} denotes a zero matrix and \mathbf{I} denotes an identity matrix, both with the same dimension as that of the matrix \mathbf{A} . Therefore, the block matrix \mathbf{K} has dimension $(2L+1)(n-1) \times (2L+1)(n-1)$. Now define $\mathbf{w}_{m+1} = (\mathbf{0}, \mathbf{0}, \dots, \mathbf{0}, \boldsymbol{\Gamma}_{n-1}^\top)^\top$, a vector of the same length as \mathbf{p}_{m+1} for the interval (t_m, t_{m+1}) . Then we can write the negative log likelihood in (A.19) as

$$-\mathcal{L}(\boldsymbol{\theta}) \approx -\sum_{m=0}^{M-1} \log(\mathbf{w}_{m+1}^\top \mathbf{p}_{m+1}). \quad (\text{A.33})$$

Let us define the Lagrangian \mathbf{L} as follows:

$$\mathbf{L}(\mathbf{p}_{m+1}, \boldsymbol{\theta}, \boldsymbol{\lambda}_{m+1}) = \sum_{m=0}^{M-1} \left[-\log(\mathbf{w}_{m+1}^\top \mathbf{p}_{m+1}) + \boldsymbol{\lambda}_{m+1}^\top (\mathbf{K}\mathbf{p}_{m+1} - \mathbf{v}_{m+1}) \right]. \quad (\text{A.34})$$

Here, \mathbf{p}_{m+1} is the state solution from (A.31) and $\boldsymbol{\lambda}_{m+1}$ is a vector of Lagrange multipliers. Now, by taking the variations of the Lagrangian with respect to $\boldsymbol{\lambda}_{m+1}$ and \mathbf{p}_{m+1} , we obtain the state equation,

$$\mathbf{L}_{\boldsymbol{\lambda}_{m+1}}(\mathbf{p}_{m+1}, \boldsymbol{\theta}, \boldsymbol{\lambda}_{m+1}) = \mathbf{K}\mathbf{p}_{m+1} - \mathbf{v}_{m+1} = 0, \quad (\text{A.35})$$

and the adjoint equation,

$$\mathbf{L}_{\mathbf{p}_{m+1}}(\mathbf{p}_{m+1}, \boldsymbol{\theta}, \boldsymbol{\lambda}_{m+1}) = -\frac{\mathbf{w}_{m+1}}{\mathbf{w}_{m+1}^\top \mathbf{p}_{m+1}} + \mathbf{K}^\top \boldsymbol{\lambda}_{m+1} = 0. \quad (\text{A.36})$$

In the above equations, we have used the subscript on \mathbf{L} to denote the variables with respect to which we have differentiated. We compute the gradient of the log likelihood by taking the derivative of the Lagrangian with respect to each θ_ℓ :

$$\mathbf{L}_{\theta_\ell}(\mathbf{p}_{m+1}, \boldsymbol{\theta}, \boldsymbol{\lambda}_{m+1}) = - \sum_{m=0}^{M-1} \left[\frac{\mathbf{p}_{m+1}^\top \frac{\partial \mathbf{w}_{m+1}}{\partial \theta_\ell}}{\mathbf{w}_{m+1}^\top \mathbf{p}_{m+1}} \right] + \sum_{m=0}^{M-1} \mathbf{Q}^\top \boldsymbol{\lambda}_{m+1} = - \frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \theta_\ell}, \quad (\text{A.37})$$

where

$$\mathbf{Q} = \frac{\partial \mathbf{K}}{\partial \theta_\ell} \mathbf{p}_{m+1} - \frac{\partial \mathbf{v}_{m+1}}{\partial \theta_\ell}. \quad (\text{A.38})$$

We rewrite (A.37) as

$$- \frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \theta_\ell} = - \sum_{m=0}^{M-1} \left[\frac{\mathbf{q}_{n-1}^\top \frac{\partial \boldsymbol{\Gamma}_{n-1}}{\partial \theta_\ell}}{\boldsymbol{\Gamma}_{n-1}^\top \mathbf{q}_{n-1}} \right] - \sum_{m=0}^{M-1} \left[\frac{\partial \mathbf{q}_1}{\partial \theta_\ell} \right]^\top \boldsymbol{\lambda}_1 - \sum_{m=0}^{M-1} \sum_{i=1}^{n-2} \left[\frac{\partial \mathbf{A}}{\partial \theta_\ell} \mathbf{q}_i \right]^\top \boldsymbol{\lambda}_{i+1}. \quad (\text{A.39})$$

Let $\boldsymbol{\lambda}_{m+1}^\top = (\lambda_1^\top, \lambda_2^\top, \dots, \lambda_{n-1}^\top)$. Then, the adjoint equation (A.36) can be written as the following temporally evolving system of equations, which can be solved backward in time for $i = 1, \dots, n-2$

$$\boldsymbol{\lambda}_{n-(i+1)} - \mathbf{K}^\top \boldsymbol{\lambda}_{n-i} = \mathbf{0}, \quad (\text{A.40a})$$

$$\boldsymbol{\lambda}_{n-1} = \frac{1}{\boldsymbol{\Gamma}_{n-1}^\top \mathbf{q}_{n-1}} \boldsymbol{\Gamma}_{n-1}. \quad (\text{A.40b})$$

The adjoint method to compute the gradient can now be summarized. For each fixed m , the procedure is as follows.

- 1 Given the unknown parameter vector $\boldsymbol{\theta}$, solve the state/forward problem equation (3.10) to find \mathbf{q}_{m+1} . This is the same as solving the time evolution system in (3.11).
- 2 Given $\boldsymbol{\theta}$ and \mathbf{q}_{m+1} , solve the adjoint equation (A.36) to find $\boldsymbol{\lambda}_{m+1}$. This is same as solving the time evolution system in (A.40a-A.40b).
- 3 With \mathbf{q}_{m+1} , $\boldsymbol{\lambda}_{m+1}$, use (A.39) to compute the gradient.

We follow this procedure for each Markovian piece of the likelihood $p(x_{m+1} | x_m; \boldsymbol{\theta})$. For each piece, we need n steps in time to solve for the state \mathbf{q} and n steps in time to solve for the adjoint $\boldsymbol{\lambda}$. In total, across the entire time series, we need $2Mn$ steps to compute the log likelihood and its gradient. In particular, note that the number of steps in time does not depend on the dimension of $\boldsymbol{\theta}$. This is in sharp contrast to a direct (i.e., non-adjoint) method computation of the gradient, in which one would have to perform a forward evolution in time (analogous to (3.10)) for each component of $\boldsymbol{\theta}$. Such a method would require $Mn(1 + |\boldsymbol{\theta}|)$ steps to compute the log likelihood and its gradient, where $|\boldsymbol{\theta}|$ is the dimension of $\boldsymbol{\theta}$.