Scalable Gradients and Variational Inference for
Stochastic Differential Equations

Xuechen Li
Google Research

Ting-Kam Leonard Wong
University of Toronto

Ricky T. Q. Chen
University of Toronto, Vector Institute

David Duvenaud
University of Toronto, Vector Institute

Abstract
We derive reverse-mode (or adjoint) automatic differentiation for solutions of stochastic
differential equations (SDEs), allowing time-efficient and constant-memory computation of
pathwise gradients, a continuous-time analogue of the reparameterization trick. Specifically,
we construct a backward SDE whose solution is the gradient and provide conditions under
which numerical solutions converge. We also combine our stochastic adjoint approach with
a stochastic variational inference scheme for continuous-time SDE models, allowing us to
learn distributions over functions using stochastic gradient descent. Our latent SDE model
achieves competitive performance compared to existing approaches on time series modeling.

1. Introduction
Deterministic dynamical systems can often be modeled by ordinary differential equations
(ODEs). For training, a memory-efficient implementation of the adjoint sensitivity method
(Chen et al., 2018) effectively computes gradients through ODE solutions with constant
memory cost. Stochastic differential equations (SDEs) are a generalization of ODEs which
incorporate instantaneous noise into their dynamics (Arnold, 1974; Øksendal, 2003). They
are a natural fit for modeling phenomena governed by small and unobserved interactions.

In this paper, we generalize the adjoint method to dynamics defined by SDEs resulting in
an approach which we call the stochastic adjoint sensitivity method. Building on theoretical
advances by Kunita (2019), we derive a memory-efficient adjoint method whereby we
simultaneously reconstruct the original trajectory and evaluate the gradients by solving a
backward SDE (in the sense of Kunita (2019)) whose formulation we detail in Section 3.
Computationally, in order to retrace the original trajectory during the backward pass, we need
to reuse noise samples generated in the forward pass. In Section 4, we give an algorithm that
allows arbitrarily-precise querying of a Brownian motion realization at any time point, while
only storing a single random seed. Overall, this results in a constant-memory algorithm that
approximates the gradient arbitrarily well as step size reduces by computing vector-Jacobian

© X. Li, T.-K.L. Wong, R.T.Q. Chen & D. Duvenaud.
products a constant number of times per-iteration. See Table 2 for a comparison of our method against previous approaches in terms of asymptotic time and memory complexity.

We incorporate SDEs into a stochastic variational inference framework, whereby we efficiently compute likelihood ratios and backpropagate through the evidence lower bound using our adjoint approach. This effectively generalizes existing model families such as latent ODEs (Rubanova et al., 2019) and deep Kalman filters (Krishnan et al., 2017).

2. Background

We review works on learning ODEs and SDEs. We refer the reader to Appendix B on background for stochastic flows (Kunita, 2019) and (backward) Stratonovich integrals.

2.1. Adjoint Sensitivity Method

The adjoint sensitivity method is an efficient approach to solve optimization problems by considering the dual form (Pontryagin, 2018). Chen et al. (2018) recently applied this idea to obtain gradients with respect to parameters of a neural network defining an ODE. The method is scalable due to its memory-efficiency, as intermediate computations need not be cached as in regular backpropagation (Rumelhart et al., 1988).

2.2. Neural Stochastic Differential Equation

Recent works have considered SDEs whose drift and diffusion functions are defined by neural networks (Tzen and Raginsky, 2019a,b; Liu et al., 2019; Jia and Benson, 2019). Consider a filtered probability space \((\Omega, \mathcal{F}, \{\mathcal{F}_t\}_{t \in \mathbb{T}}, P)\) on which an \(m\)-dimensional adapted Wiener process \(\{W_t\}_{t \in \mathbb{T}}\) is defined. An Itô SDE defines a stochastic process \(\{Z_t\}_{t \in \mathbb{T}}\) by

\[
Z_T = z_0 + \int_0^T b(Z_t, t) \, dt + \sum_{i=1}^m \int_0^T \sigma_i(Z_t, t) \, dW_t^{(i)},
\]

where \(z_0 \in \mathbb{R}^d\) is a deterministic starting value, and \(b : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d\) and \(\sigma_i : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d\) are the drift and diffusion functions, respectively. Here, the second integral on the right hand side of (1) is the Itô stochastic integral (Øksendal, 2003). When the coefficients are globally Lipschitz in both the state and time components, there exists a unique strong solution to the SDE (Øksendal, 2003). Therefore, one can consider coefficients defined by neural networks that have smooth activation functions (e.g. \(\tanh\)) of the form \(b(z, t, \theta)\) and \(\sigma(z, t, \theta)\). This results in a model known as the neural SDE.

3. Sensitivities via Stochastic Adjoint

We derive a backward Stratonovich SDE for what we call the stochastic adjoint process. A direct implication of this is a gradient computation algorithm that works by solving a set of dynamics in reverse time and relies on vector-Jacobian products without storing intermediate computation. Recall from Appendix B.3, \(\Phi_{s,t}(z) := Z_t^{s,z}\) is the solution at time \(t\) when the process is started at \(z\) at time \(s\), and its inverse is defined as \(\Psi_{s,t}(z) := \Phi_{s,t}^{-1}(z)\).

Consider \(A_{s,t}(z) = \nabla(\mathcal{L}(\Phi_{s,t}(z)))\), where \(\mathcal{L}\) is a scalar loss function. The chain rule gives \(A_{s,t}(z) = \nabla \mathcal{L}(\Phi_{s,t}(z)) \nabla \Phi_{s,t}(z)\). Let \(\tilde{A}_{s,t}(z) := A_{s,t}(\Psi_{s,t}(z)) = \nabla \mathcal{L}(z) \nabla \Phi_{s,t}(\Psi_{s,t}(z)) = \ldots\)
\[ \nabla L(z)K_{s,t}(z). \] Note that \( A_{s,t}(z) = \tilde{A}_{s,t}(\Phi_{s,t}(z)). \) Since \( \nabla L(z) \) is constant, we see that \( (\tilde{A}_{s,t}(z), \tilde{\Psi}_{s,t}(z)) \) satisfies the following backward SDE system by Lemma C.1 (cf. Appendix C)

\[
\begin{align*}
\tilde{A}_{s,t}(z) &= \nabla L(z) + \int_s^t \nabla b(\tilde{\Psi}_{r,t}(z), r) \, \mathrm{d}r + \sum_{i=1}^m \int_s^t \nabla \sigma_i(\tilde{\Psi}_{r,T}(r), u) \, \mathrm{d}W_r^i, \\
\tilde{\Psi}_{s,t}(z) &= z - \int_s^t b(\tilde{\Psi}_{r,t}(z), r) \, \mathrm{d}r - \sum_{i=1}^m \int_s^t \sigma_i(\tilde{\Psi}_{r,t}(z), r) \, \mathrm{d}W_r^i.
\end{align*}
\]

(2)

Since (11) can be viewed as a single SDE (with smooth coefficients) for an augmented state, \( \tilde{A}_{s,T}(z) \) also has a unique strong solution. Therefore, for \( t = 0 \), we may write

\[ \tilde{A}_0(z) = F(z, W), \]

where \( W = \{ W_t \}_{0 \leq t \leq T} \) denotes the path of the Brownian motion and \( F : \mathbb{R}^d \times C([0,1], \mathbb{R}^m) \rightarrow \mathbb{R}^d \) is a deterministic measurable function (the Itô map) (Rogers and Williams, 2000, Definition 10.9). The next theorem follows immediately from (3) and the definition of \( F \).

**Theorem 3.1:**
For \( P \)-almost all \( \omega \in \Omega \), \( A_{0,T}(z) = \tilde{A}_0(z) = F(G(z, W, W), W) \), where \( G(z, W) = Z^0_{T,z} \).

The theorem is a consequence of \( A_{0,T}(z) = \tilde{A}_0(\Phi_{0,T}(z)) = \tilde{A}_0(\tilde{Z}_{T,z}) \) and (3). This implies we may solve the dynamics (11) starting from the end state of the forward solve \( \tilde{Z}_{T,z} \) to obtain the gradient of the loss with respect to the starting value \( z \). To obtain the gradient with respect to the parameters, we augment the original state with parameters. Algorithm 1 summarizes this assuming access to a black-box solver SDESolve. See details in Appendix C.

**Algorithm 1 Stratonovich SDE Adjoint Computation**

**Input:** parameters \( \theta \), start time \( t_0 \), stop time \( t_1 \), final state \( z_{t_1} \), loss gradient \( \partial L / \partial z_{t_1} \).

**Input:** drift \( b(z, t, \theta) \), diffusion \( \sigma(z, t, \theta) \), Wiener process sample \( w(t) \).

\[
s_{t_1} = [z_{t_1}, \partial L / \partial z_{t_1}, 0_p] \\
def \text{augb}([z_t, a_t^\top], \theta): \quad \text{return} \quad -b(z_t, -t, \theta), a_t^\top \partial b / \partial z, a_t^\top \partial b / \partial \theta \\
def \text{augc}([z_t, a_t^\top], \theta): \quad \text{return} \quad -\sigma_i(z_t, -t, \theta), a_t^\top \partial \sigma_i / \partial z, a_t^\top \partial \sigma_i / \partial \theta \\
def \text{augw} \quad \text{return} \quad -w(-t), -w(-t), -w(-t) \\
[\partial L / \partial z_{t_0}, \partial L / \partial \theta] = \text{SDESolve} \left( s_{t_1}, \text{augb}, \text{augc}_1, \ldots, \text{augc}_m, \text{augw}, -t_1, -t_0 \right) \\
\text{return} \quad [\partial L / \partial z_{t_0}, \partial L / \partial \theta]
\]

### 4. Seeded Brownian Tree

We present a data structure that allows arbitrarily-precise query of the sample path of the Wiener process given a global random seed based on the Brownian tree construction. The data structure facilitates the adjoint method such that we can ensure the noise

![Brownian tree](image_url)
sample in the backward solve is the same as in the forward solve using a split pseudorandom random number generator (PRNG). We present the procedure in Algorithm D.2 and details in Appendix D.

5. Latent SDE

Consider the SDEs

\begin{align*}
    d\tilde{Z}_t &= h_0(\tilde{Z}_t, t) + \sigma(\tilde{Z}_t, t) \, d\tilde{W}_t, \quad \tilde{Z}_0 = z \in \mathbb{R}^d, \\
    dZ_t &= h_1(Z_t, t) + \sigma(Z_t, t) \, dW_t, \quad Z_0 = z \in \mathbb{R}^d,
\end{align*}

where \( h_0, h_1 \in \mathbb{R}^d \times \mathbb{R} \rightarrow \mathbb{R}^d \) and \( \sigma \in \mathbb{R}^d \times \mathbb{R} \rightarrow \mathbb{R}^{d \times m} \) are Lipschitz in both arguments. Suppose (4) and (5) define the prior and posterior processes, respectively. Additionally, assume there is a function \( u : \mathbb{R}^d \times \mathbb{R} \rightarrow \mathbb{R}^m \) such that \( \sigma(x, t)u(x, t) = h_1(x, t) - h_0(x, t) \) for all \( x \in \mathbb{R}^d \) and \( t \in \mathbb{R} \). Then, the variational free energy (Opper, 2019) can be written as

\[ L_{VI} = \mathbb{E}\left[ \frac{1}{2} \int_0^T |u(Z_t, t)|^2 \, dt + \int_0^T u(Z_t, t)^T \, dW_t - \sum_{i=1}^N \log p(y_t | z_{t_i}) \right], \]

where the expectation is taken over the distribution of the posterior process defined by (5), and \( y_1, \ldots, y_N \) are observations at times \( t_1, \ldots, t_N \), respectively. To compute the gradient with respect to parameters, we need only augment the forward equation with an extra variable whose drift function returns \( \frac{1}{2} |u(Z_t, t)|^2 \) and diffusion function is 0. In this case, the backward adjoint dynamics can be derived analogously using (11). Appendix E includes details.

6. Experiments

We verify our theory by comparing the gradients obtained by our stochastic adjoint framework against analytically derived gradients for chosen test problems with closed-form solutions. We then fit latent SDE models with our framework on two synthetic datasets and a real dataset, verifying that the variational inference framework promotes learning a generative model of time series. Due to space constraint, we refer the reader to Appendix F for results on numerical studies and Appendix N for results on synthetic data. We present only the results on the motion capture dataset here.

6.1. Motion Capture Dataset

We experiment on a dataset extracted from the CMU motion capture library. We use the dataset adopted by Gan et al. (2015) which consists of 23 walking sequences of subject number 35 that is partitioned into 16 training, 3 validation, and 4 test sequences. We include the settings in Appendix O and report the test MSE here following Yıldız et al. (2019).

References

Table 1: Test MSE; 95% confidence interval averaged over 50 samples. † from Yildiz et al. (2019).

<table>
<thead>
<tr>
<th>Method</th>
<th>Test MSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>npODE Heinonen et al. (2018)</td>
<td>22.96†</td>
</tr>
<tr>
<td>NeuralODE Chen et al. (2018)</td>
<td>22.49 ± 0.88†</td>
</tr>
<tr>
<td>ODE2VAE Yildiz et al. (2019)</td>
<td>10.06 ± 1.4†</td>
</tr>
<tr>
<td>ODE2VAE-KL Yildiz et al. (2019)</td>
<td>8.09 ± 1.95†</td>
</tr>
<tr>
<td>Latent ODE Chen et al. (2018); Rubanova et al. (2019)</td>
<td>5.98 ± 0.28</td>
</tr>
<tr>
<td>Latent SDE (this work)</td>
<td><strong>4.03 ± 0.20</strong></td>
</tr>
</tbody>
</table>


Appendix A. Asymptotic Complexity Comparison

Table 2: Asymptotic time complexity of differentiation. \( L \) is the number of steps used in the solve, and \( D \) is the size of the state. Both memory and time are expressed in units of the cost of evaluating the drift and diffusion functions once each.

<table>
<thead>
<tr>
<th>Method</th>
<th>Memory</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forward pathwise (Yang and Kushner, 1991)</td>
<td>( \mathcal{O}(1) )</td>
<td>( \mathcal{O}(LD) )</td>
</tr>
<tr>
<td>Backprop through solver’s ops. (Giles and Glasserman, 2006)</td>
<td>( \mathcal{O}(L) )</td>
<td>( \mathcal{O}(L) )</td>
</tr>
<tr>
<td>Stochastic adjoint (ours)</td>
<td>( \mathcal{O}(1) )</td>
<td>( \mathcal{O}(L) )</td>
</tr>
</tbody>
</table>

Appendix B. Additional Background

**Notation.** For a fixed terminal time \( T > 0 \), we denote by \( \mathbb{T} = [0, T] \subseteq \mathbb{R} \) the time horizon. Let \( C^\infty \) be the class of infinitely differentiable functions from \( \mathbb{R}^d \) to itself. Let \( C^{p,q} \) be the class of functions from \( \mathbb{R}^d \times \mathbb{T} \) to \( \mathbb{R}^d \) that are \( p \) and \( q \) times continuously differentiable in the first and second component, respectively. Let \( C^{p,q}_b \subseteq C^{p,q} \) be the subclass with bounded derivatives of all possible orders. For a positive integer \( m \), we adopt the shorthand \( [m] = \{1, 2, \ldots, m\} \). We denote the Euclidean norm of a vector \( v \) by \( |v| \). For an \( m \)-dimensional Wiener process \( W_t \), we denote the \( i \)th component of \( W_t \) by \( W_t^{(i)} \). For the diffusion functions \( \sigma_1, \ldots, \sigma_m \), we will also write \( \sigma : \mathbb{R}^d \rightarrow \mathbb{R}^{d \times m} \) as the matrix-valued function obtained by stacking the component functions \( \sigma_i \) in a columnwise fashion, and index its \( j \)th row and \( i \)th column by \( \sigma_{j,i} \).

**B.1. Recent works on Neural SDEs**

Among recent work on neural SDEs, none has enabled an efficient training framework. In particular, Tzen and Raginsky (2019a); Liu et al. (2019) considered computing the gradient by simulating the forward dynamics of an explicit Jacobian matrix the size either the squared number of parameters or the number of parameters times the number of states, building on the pathwise approach (Gobet and Munos, 2005; Yang and Kushner, 1991). By contrast, the approach we present only requires evaluating vector-Jacobian products a constant number of times with respect to the number of parameters and states, which has the same asymptotic time cost as evaluating the drift and diffusion functions, and can be done automatically by modern machine learning libraries (Maclaurin et al., 2015; Paszke et al., 2017; Abadi et al., 2016; Frostig et al., 2018).

**B.2. Backward Stratonovich Integral**

Our stochastic adjoint sensitivity method involves stochastic processes running forward and backward in time. The Stratonovich stochastic integral, due to its symmetry, gives nice expressions for the backward dynamics and so is more convenient for our purpose. Our results can be applied straightforwardly to Itô SDEs as well using a conversion result (see e.g. (Platen, 1999, Sec. 2)).
Following the treatment of Kunita (Kunita, 2019), we introduce the forward and backward Stratonovich integrals. Let \( F_{s,t} \) be a two-sided filtration, where \( F_{s,t} \) is the \( \sigma \)-algebra generated by \( \{ W_v - W_u : s \leq u \leq v \leq t \} \) for all \( s, t \in \mathbb{T} \) such that \( s \leq t \). For a continuous semimartingale \( \{ Y_t \}_{t \in \mathbb{T}} \) adapted to the forward filtration \( \{ F_{0,t} \}_{t \in \mathbb{T}} \), the Stratonovich stochastic integral is defined as

\[
\int_0^T Y_t \circ dW_t = \lim_{|\Pi| \to 0} \sum_{k=1}^{N} \frac{1}{2} (Y_{t_k} + Y_{t_{k-1}}) (W_{t_k} - W_{t_{k-1}}),
\]

where \( \Pi = \{ 0 = t_0 < \cdots < t_N = T \} \) is a partition of the interval \( \mathbb{T} = [0, T] \), \( |\Pi| = \max k t_k - t_{k-1} \) denotes the size of largest segment of the partition, and the limit is to be interpreted in the \( L^2 \) sense. The Itô integral uses instead the left endpoint \( Y_{t_k} \) rather than the average. In general, the Itô and Stratonovich integrals differ by a term of finite variation.

To define the backward Stratonovich integral, we consider the backward Wiener process \( \tilde{W}_t \) defined as \( \tilde{W}_t = W_T - W_t \) for all \( t \in \mathbb{T} \) that is adapted to the backward filtration \( \{ F_{t,T} \}_{t \in \mathbb{T}} \). For a continuous semimartingale \( \tilde{Y}_t \) adapted to the backward filtration, we define

\[
\int_s^T \tilde{Y}_t \circ d\tilde{W}_t = \lim_{|\Pi| \to 0} \sum_{k=1}^{N} \frac{1}{2} (\tilde{Y}_{t_k} + \tilde{Y}_{t_{k-1}}) (\tilde{W}_{t_k} - \tilde{W}_{t_{k-1}}),
\]

where \( \Pi = \{ 0 = t_N < \cdots < t_0 = T \} \) is a partition, and the limit is again in the \( L^2 \) sense.

### B.3. Stochastic Flow of Diffeomorphisms

It is well known that an ODE defines a flow of diffeomorphisms (Arnold, 1978). Here we consider the stochastic analogue for the Stratonovich SDE

\[
Z_T = z_0 + \int_0^T b(Z_t, t) \, dt + \sum_{i=1}^m \int_0^T \sigma_i(Z_t, t) \circ dW_t^{(i)}. \tag{7}
\]

Throughout the paper, we assume \( b, \sigma \) are of class \( C_b^{\infty,1} \), so that the SDE has a unique strong solution. Let \( \Phi_{s,t}(z) := Z_t^{s,z} \) be the solution at time \( t \) when the process is started at \( z \) at time \( s \). Given a realization of the Wiener process, this defines a collection \( \mathcal{S} = \{ \Phi_{s,t} \}_{s \leq t, t \in \mathbb{T}} \) of continuous maps from \( \mathbb{R}^d \) to itself.

The following theorem shows that these maps are diffeomorphisms and that they satisfy backward SDEs.

**Theorem B.1 (Thm. 3.7.1 (Kunita, 2019)):**

(i) With probability 1, the collection \( \mathcal{S} = \{ \Phi_{s,t} \}_{s \leq t, t \in \mathbb{T}} \) satisfies the flow property

\[
\Phi_{s,t}(z) = \Phi_{s,u}(\Phi_{u,t}(z)), \quad s \leq u \leq t, \quad z \in \mathbb{R}^d.
\]

Moreover, each \( \Phi_{s,t} \) is a smooth diffeomorphism from \( \mathbb{R}^d \) to itself. We thus call \( \mathcal{S} \) the stochastic flow of diffeomorphisms generated by the SDE (7).
(ii) The backward flow $\Psi_{s,t} := \Phi_{s,t}^{-1}$ satisfies the backward SDE:

$$\Psi_{s,t}(z) = z - \int_s^t b(\Psi_{u,t}(z), u) \, du - \sum_{i=1}^m \int_s^t \sigma_i(\Psi_{u,t}(z), u) \circ dB_{u}^{(i)},$$

for all $z \in \mathbb{R}^d$ and $s, t \in \mathbb{T}$ such that $s \leq t$ a.s.

Note that the coefficients in (7) and (8) differ by only a negative sign. This symmetry is due to our use of the Stratonovich integral (see Figure 2).

Figure 2: Negating the drift and diffusion functions for an Itô SDE and simulating backwards from the end state gives the wrong solution. Negating the drift and diffusion functions for the converted Stratonovich SDE, however, gives the correct path when simulated in reverse time.

Appendix C. Details on Stochastic Adjoint Derivation and Implementation

We present our main contribution, i.e. the stochastic analog of the adjoint sensitivity method for SDEs. We use (8) to derive another backward Stratonovich SDE for what we call the stochastic adjoint process. The direct implication of this is a gradient computation algorithm that works by solving a set of dynamics in reverse time and relies on vector-Jacobian products without storing intermediate computation produced in the forward pass.

C.1. Stochastic Adjoint Process

The goal is to derive the stochastic adjoint process $\{\partial L/\partial Z_t\}_{t \in \mathbb{T}}$ that can be simulated by evaluating only vector-Jacobian products, where $L = L(Z_T)$ is a scalar loss of the terminal state $Z_T$. The main theoretical result is Theorem 3.1. We first derive a backward SDE for the process $\{\partial Z_T/\partial Z_t\}_{t \in \mathbb{T}}$, assuming that $Z_t = \Psi_{t,T}(Z_T)$ for a deterministic $Z_T \in \mathbb{R}^d$ that does not depend on the realized Wiener process. We then extend to the case where $Z_T = \Phi_{0,T}(z_0)$ for a deterministic $z_0 \in \mathbb{R}^d$. In the latter case, the resulting value cannot be interpreted as the solution to a backward SDE anymore due to loss of adaptiveness; instead we will formulate the result using the Itô map (Rogers and Williams, 2000). Finally, we extend the state of $Z$ to include parameters and obtain the gradient with respect to them.

We first derive the SDE for the Jacobian matrix of the backward flow.
Lemma C.1 (Dynamics of $\partial Z_T / \partial t$):

Consider the stochastic flow generated by the backward SDE (8) as in Theorem B.1(ii). Let $J_s(z) := \nabla \Psi_{s,T}(z)$, then it satisfies the backward SDE

$$J_{s,t}(z) = I_d - \int_s^t \nabla b(\Psi_{r,t}(z), r) J_{r,t}(z) \, dr - \sum_{i=1}^m \int_s^t \nabla \sigma_i(\Psi_{r,t}(z), r) J_{r,t}(z) \circ dW_r^{(i)},$$

for all $s \leq t$ and $x \in \mathbb{R}^d$ a.s. Furthermore, let $K_{s,t}(z) = J_{s,t}(z)^{-1}$, we have

$$K_{s,t}(z) = I_d + \int_s^t K_{r,t}(z) \nabla b(\Psi_{r,t}(z), r) \, dr + \sum_{i=1}^m \int_s^t K_{r,t}(z) \nabla \sigma_i(\Psi_{r,t}(z), r) \circ dW_r^{(i)},$$

for all $s \leq t$ and $x \in \mathbb{R}^d$ a.s.

The proof included in Appendix I relies on Itô’s lemma in the Stratonovich form (Kunita, 2019, Theorem 2.4.1). This lemma considers only the case where the endpoint $z$ is fixed and deterministic.

Now we compose the state process (represented by the flow) and the loss function $\mathcal{L}$. Consider $A_{s,t}(z) = \nabla (\mathcal{L}(\Phi_{s,t}(z)))$. The chain rule gives $A_{s,t}(z) = \nabla \mathcal{L}(\Phi_{s,t}(z)) \nabla \Phi_{s,t}(z)$. Let

$$\tilde{A}_{s,t}(z) := A_{s,t}(\Psi_{s,t}(z)) = \nabla \mathcal{L}(\Phi_{s,t}(z)) \nabla \Phi_{s,t}(z) = \nabla \mathcal{L}(z) K_{s,t}(z).$$

Note that $A_{s,t}(z) = \tilde{A}_{s,t}(\Phi_{s,t}(z))$. Since $\nabla \mathcal{L}(z)$ is constant, we see that $(\tilde{A}_{s,t}(z), \Psi_{s,t}(z))$ satisfies the backward SDE system

$$\tilde{A}_{s,t}(z) = \nabla \mathcal{L}(z) + \int_s^t \nabla b(\Psi_{r,t}(z), r)^\top \tilde{A}_{r,t}(z) \, dr + \sum_{i=1}^m \int_s^t \nabla \sigma_i(\Psi_{r,t}(r), u)^\top \tilde{A}_{r,t}(z) \circ dW_r^{(i)},$$

$$\Psi_{s,t}(z) = z - \int_s^t b(\Psi_{r,t}(z), r) \, dr - \sum_{i=1}^m \int_s^t \sigma_i(\Psi_{r,t}(z), r) \circ dW_r^{(i)}.$$

(11)

Writing $\tilde{X}_s = (\tilde{A}_{s,T}(z)^\top, \tilde{\Psi}_{s,T}(z)^\top)^\top$ as the augmented process, the system (11) is a backward Stratonovich SDE of the form

$$\tilde{X}_s(z) = A(z) + \int_s^T B(\tilde{X}_r(z), r) \, dr + \sum_{i=1}^m \int_s^T S_i(\tilde{X}_r(z), r) \circ dW_r^{(i)},$$

where $B, S_i \in C_b^{\infty,1}$. As a result (11) has a unique strong solution.

Without loss of generality, assume $t = 0$. Since (11) admits a strong solution, we may write

$$\tilde{A}_0(z) = F(z, W),$$

where $W = \{W_t\}_{0 \leq t \leq T}$ denotes the path of the Brownian motion and

$$F : \mathbb{R}^d \times C([0, 1], \mathbb{R}^m) \to \mathbb{R}^d$$

is a deterministic measurable function (the Itô map) (Rogers and Williams, 2000, Definition 10.9). Intuitively, $F$ can be thought as an algorithm that computes the solution to the backward SDE (11) given the position $z$ at time $T$ and the realized Brownian path. Similarly, we let $G$ be the solution map for the forward flow (7). Immediately, we arrive at Theorem 3.1.
C.2. Numerical Approximation

In practice, we compute solutions to SDEs with numerical solvers \(F_h\) and \(G_h\), where \(h = T/N\) denotes the mesh size of a fixed grid\(^1\). The approximate algorithm thus outputs \(F_h(G_h(z,W),W)\). The following theorem provides sufficient conditions for convergence.

**Theorem C.2:**
Suppose the schemes \(F_h\) and \(G_h\) satisfy the following conditions: (i) \(F_h(z,W) \to F(z,W)\) and \(G_h(z,W) \to G(z,W)\) converge to 0 in probability as \(h \to 0\), and (ii) for any \(M > 0\), we have \(\sup |z| \leq M |F_h(z,W) - F(z,W)| \to 0\) in probability as \(h \to 0\). Then, for any starting point \(z\) of the forward flow, we have

\[
F_h(G_h(z,W),W) \to F(G(z,W),W) = A_{0,T}(z)
\]

in probability as \(h \to 0\).

For details and the proof see Appendix J. Usual schemes such as the Euler-Maruyama and Milstein method satisfy condition (i). Indeed, they converge pathwise (i.e. almost surely) with explicit rates for any fixed starting point (Kloeden and Neuenkirch, 2007). While condition (ii) is rather strong, we note that the SDEs considered here have smooth coefficients and thus the solutions enjoy nice regularity properties in the starting position. Therefore, it is reasonable to expect that the corresponding numerical schemes to also behave nicely as a function of both the mesh size and the starting position. To the best of our knowledge this property is not considered at all in the literature on numerical methods for SDEs (where the initial position is fixed), but is crucial in the proof of Theorem C.2. Detailed analysis for specific schemes is beyond the scope of this paper and is left for future research.

C.3. The Algorithm

So far we have derived the gradient of the loss with respect to the initial state. We can extend these results to give gradients with respect to parameters of the drift and diffusion functions by treating them as an additional part of the state whose dynamics has zero drift and diffusion. We summarize this in Algorithm 1\(^2\), assuming access to a numerical solver \texttt{SDESolve}. Note for the Euler-Maruyama scheme, the most costly terms to compute \(a_t^\top \partial b/\partial \theta\) and \(a_t^\top \partial \sigma_i/\partial \theta\) can be evaluated by calling \texttt{vjp}(\(a_t,b,\theta\)) and \texttt{vjp}(\(a_t,\sigma_i,\theta\)), respectively.

In principle, we can simulate the forward and backward adjoint dynamics with any high-order solver of choice. However, in practice, to obtain a strong numerical solution\(^3\) with order beyond \(1/2\), we need to simulate multiple integrals of the Wiener process such as \(\int_0^t \int_0^s dW_u^{(i)} dW_s^{(j)}\) for \(i,j \in [m], i \neq j\). These random variables are difficult to simulate exactly and costly to approximate using truncated infinite series (Wiktorsson et al., 2001).

Note that even though the backward SDE for the stochastic adjoint does not have diagonal noise, it satisfies a commutativity property (Rößler, 2004) when the SDE of the original

---

1. We may also use adaptive solvers (Ilie et al., 2015).
2. We use row vector notation here.
3. A numerical scheme is of strong order \(p\) if \(E[|X_T - X_{N\eta}|] \leq C\eta^p\) for all \(T > 0\), where \(X_t\) and \(X_{N\eta}\) are respectively the coupled true solution and numerical solution, \(N\) and \(\eta\) are respectively the iteration index and step size such that \(N\eta = T\), and \(C\) is independent of \(\eta\).
dynamics has diagonal noise. In this case, we can safely adopt certain numerical schemes of strong order 1.0 (e.g. Milstein (Milstein, 1994) and stochastic Runge-Kutta (Rößler, 2010)) without approximating multiple integrals or the Lévy area during simulation. We verify this formally in Appendix K.

C.4. Software and Implementation
We have implemented several SDE solvers in PyTorch (Paszke et al., 2017) which include Euler-Maruyama, Milstein, and stochastic Runge-Kutta schemes with adaptive time-stepping using a PI controller (Ilie et al., 2015). In addition, following torchdiffeq (Chen et al., 2018), we have created a user-friendly subclass of torch.autograd.Function that facilitates gradient computation using our stochastic adjoint framework when the neural SDE is implemented as a subclass of torch.nn.Module. We include a short code snippet covering the main idea of the stochastic adjoint in Appendix L and plan to release all code after the double-blind reviewing process.

Appendix D. Details on Seeded Brownian Tree
The formulation of the adjoint ensures it can be numerically integrated by merely evaluating dynamics cheaply defined by vector-Jacobian products, as opposed to whole Jacobians. However, the backward-in-time nature also introduces the additional difficulty that the same Wiener process sample path in the forward pass has to be queried again during the backward pass. Naïvely storing Brownian motion increments and related quantities (e.g. Lévy area approximations) not only implies a large memory consumption but also disables using adaptive time-stepping numerical integrators, where the evaluation timestamps in the backward pass may be different from those in the forward pass.

To overcome this issue, we combine Brownian trees with splittable Pseudorandom number generators (PRNGs) and obtain a data structure that allows querying values of the Wiener process path at arbitrary times with logarithmic time cost with respect to some error tolerance.

D.1. Brownian Bridge and Brownian Tree
Lévy’s Brownian bridge (Revuz and Yor, 2013) states that given a start time $t_s$ and end time $t_e$ along with their respective Wiener process values $w_s$ and $w_e$, the marginal of the process at time $t \in (t_s, t_e)$ is a normal distribution:

$$N \left( \frac{(t_e - t)w_s + (t - t_s)w_e}{t_e - t_s}, \frac{(t_e - t)(t - t_s)}{t_e - t_s}I_d \right).$$

We can recursively apply this formula to evaluate the process at the midpoint of any two distinct timestamps where the values are already known. Constructing the whole sample path of a Wiener process in this manner results in what is known as the Brownian tree (Gaines and Lyons, 1997).
D.2. Combining Brownian Tree with Splittable PRNG

We assume access to a splittable PRNG (Claessen and Pałka, 2013), which has an operation \texttt{split} that deterministically generates two (or more) keys\(^4\) using an existing key. In addition, we assume access to an operation \texttt{BrownianBridge} which samples from (12) given a key. To obtain the Wiener process value at a specific time, the \textit{seeded Brownian tree} works by recursively sampling according to the Brownian tree with keys split from those of parent nodes, assuming the values at some initial and terminal times are known. The algorithm terminates when the current time under consideration is within a certain error tolerance of the desired time. We outline the full procedure in Algorithm D.2.

\begin{algorithm}[h]
\caption{Seeded Brownian Tree}
\begin{algorithmic}
\State \textbf{Input:} seed \(s\), query \(t\), tolerance \(\epsilon\), start time \(t_s\), start state \(w_s\), end time \(t_e\), end state \(w_e\).
\State \(t_{\text{mid}} = (t_s + t_e)/2\); \(w_{\text{mid}} = \texttt{BrownianBridge}(t_s, w_s, t_e, w_e, t_{\text{mid}}, s)\)
\While{\(|t - t_{\text{mid}}| > \epsilon\)}
\State \(s_l, s_r = \texttt{split}(s)\) \Comment{Generates two keys corresponding to left and right subtrees.}
\If{\(t < t_{\text{mid}}\)}
\State \(t_e, x_e, s = t_{\text{mid}}, w_{\text{mid}}, s_l\)
\Else
\State \(t_s, x_s, s = t_{\text{mid}}, w_{\text{mid}}, s_r\)
\EndIf
\State \(t_{\text{mid}} = (t_s + t_e)/2\); \(w_{\text{mid}} = \texttt{BrownianBridge}(t_s, w_s, t_e, w_e, t_{\text{mid}}, s)\)
\EndWhile
\State \textbf{return} \(t_{\text{mid}}\)
\end{algorithmic}
\end{algorithm}

This algorithm has constant memory cost. For fixed-step-size solvers, the tolerance that the tree will be queried at will scale as \(1/L\), where \(L\) is the number of steps in the solver. Thus the complexity per-step will scale as \(\log L\).

Appendix E. Stochastic Adjoint for Latent SDE

Note that the variational free energy (6) can be derived from Girsanov’s change of measure theorem (Opper, 2019). To efficiently Monte Carlo estimate this quantity and its gradient, we simplify the equation by noting that for a one-dimensional process \(\{V_t\}_{t \in \mathbb{T}}\) adapted to the filtration generated by a one-dimensional Wiener process \(\{W_t\}_{t \in \mathbb{T}}\), if Novikov’s condition (Øksendal, 2003) is satisfied, then the process defined by the Itô integral \(\int_0^t V_s \, dW_s\) is a Martingale (Øksendal, 2003). Hence, \(\mathbb{E}\left[\int_0^T u(Z_t, t)^T \, dW_t\right] = 0\), and

\[\mathcal{L}_{\text{VI}} = \mathbb{E}\left[\frac{1}{2} \int_0^T |u(Z_t, t)|^2 \, dt - \sum_{i=1}^N \log p(y_{t_i} | z_{t_i})\right].\]

To Monte Carlo simulate the quantity in the forward pass along with the original dynamics, we need only extend the original augmented state with an extra variable \(L_t\) such that the

\(^4\) We consider PRNGs that are based on hashing keys to values which are the desired random numbers.
new drift and diffusion functions for the new augmented state $Y_t = (Z_t^T, \theta^T, L_t)^T$ are

$$f(x,t) = \begin{pmatrix} b(z,t) \\ 0_p \\ \frac{1}{2} |u(z,t)|^2 \end{pmatrix} \in \mathbb{R}^{d+p+1}, \quad g_i(x,t) = \begin{pmatrix} \sigma_i(z,t) \\ 0_p \\ 0 \end{pmatrix} \in \mathbb{R}^{d+p+1}, \quad i \in [m].$$

By (11), the backward SDEs of the adjoint processes become

$$A_z^x = A_z^x + \int_t^T \left( \frac{\partial b(z,s)}{\partial z} \right)_{z=Z_s}^T A_z^x + \frac{1}{2} \frac{\partial \|u(z,s)\|^2}{\partial z} \bigg|_{z=Z_s} A_z^x \, dt + \sum_{i=1}^m \int_t^T \frac{\partial \sigma_i(z,s)}{\partial z} \bigg|_{z=Z_s} A_z^x \circ dW_s^{(i)},$$

$$A_\theta^x = A_\theta^x + \int_t^T \left( \frac{\partial b(z,s)}{\partial \theta} \right)_{z=Z_s}^T A_z^x + \frac{1}{2} \frac{\partial \|u(z,s)\|^2}{\partial \theta} \bigg|_{z=Z_s} A_z^x \, dt + \sum_{i=1}^m \int_t^T \frac{\partial \sigma_i(z,s)}{\partial \theta} \bigg|_{z=Z_s} A_z^x \circ dW_s^{(i)},$$

$$A_L^x = A_L^x.$$  \hspace{1cm} (13)

In this case, neither do we need to actually simulate the backward SDE of the extra variable nor do we need to simulate its adjoint. Moreover, when considered as a single system for the augmented adjoint state, the diffusion function of the backward SDE (13) satisfies the commutativity property (17).

**Appendix F. Numerical Studies**

We consider three carefully designed test problems (examples 1-3 (Rackauckas and Nie, 2017); details in Appendix M) all of which have closed-form solutions. We compare the gradient computed from simulating our stochastic adjoint process using the Milstein scheme against the gradient evaluated by analytically solving the equations. Figure F (a) shows that for test example 1, the error between the adjoint gradient and analytical gradient decreases as the fixed step size decreases.

One phenomenon not covered by our theory is that the error can be indeed be controlled by the adaptive solver. This is shown by the fact that for all three test problems, the mean-square error across dimensions tends to be smaller as the absolute tolerance is reduced (see Figure F (c, f, j)). However, we note that the Number of Function Evaluations (NFEs) tends to be much larger than that in the ODE case (Chen et al., 2018), which is expected given the inherent roughness of Brownian motion paths.

**Appendix G. Related Work**

**Sensitivity Analysis for SDEs.** Gradient computation is closely related to sensitivity analysis. Computing gradients with respect to parameters of vector fields of an SDE has been extensively studied in the stochastic control literature (Kushner and Dupuis, 2013). In particular, for low dimensional problems, this is done effectively using dynamic programming (Baxter and Bartlett, 2001) and finite differences (Glasserman and Yao, 1992; L’Ecuyer and Perron, 1994). However, both approaches scale poorly with the dimensionality of the parameter vector.

Analogous to REINFORCE (or score-function estimator) (Williams, 1992; Kleijnen and Rubinstein, 1996; Glynn, 1990), Yang and Kushner (1991) considered deriving the gradient
as $\nabla \mathbb{E}[\mathcal{L}(Z_T)] = \mathbb{E}[\mathcal{L}(Z_T)H]$ for some random variable $H$. However, $H$ usually depends on the density of $Z_T$ with respect to the Lebesgue measure which can be difficult to compute. Gobet and Munos (2005) extended this approach by weakening a non-degeneracy condition using Mallianvin calculus.

Closely related to the current submission is the pathwise method (Yang and Kushner, 1991), which is the continuous-time analog of the reparameterization trick (Kingma and Welling, 2013; Rezende et al., 2014). Existing methods in this regime (Tzen and Raginsky, 2019a; Gobet and Munos, 2005; Liu et al., 2019) all require simulating a forward SDE where each step requires computing entire Jacobian matrices. This computational cost is prohibitive for high-dimensional systems with a large number of parameters.

Based on the Euler discretization, Giles and Glasserman (2006) considered storing the intermediate values and performing reverse-mode automatic differentiation. They named this method the adjoint approach, which, by modern standards, is a form of “backpropagation

---

5. The random variable $H$ is not unique.
through the operations of a numerical solver”. We comment that this approach, despite widely adopted in the field of finance for calibrating market models (Giles and Glasserman, 2006), has high memory cost, and relies on a fixed step size Euler-Maruyama discretization. This approach was used by (Hegde et al., 2019) to parameterize the drift and diffusion of an SDE using Gaussian processes.

**Backward SDEs.** Our backward SDE for the stochastic adjoint process relies on the notion of backward SDEs by Kunita (2019) which is based on two-sided filtrations. This is different from the more traditional notion of backward SDEs where only a single filtration is defined (Peng, 1990; Pardoux and Peng, 1992). Based on the latter notion, forward-backward SDEs (FBSDEs) have been proposed to solve the stochastic optimal control problem (Peng and Wu, 1999). However, simulating FBSDEs is costly due to the need to estimate conditional expectations in the backward pass. Estimating conditional expectations, however, is a direct consequence of the appearance of an auxiliary process from the Martingale representation theorem (Pardoux and Peng, 1992).

**Appendix H. Discussion**
We presented a stochastic analog of the adjoint sensitivity method to compute gradients through solutions of SDEs. In contrast to existing approaches, this method has nearly the same time and memory complexity as simply solving the SDE. We showed how our stochastic adjoint framework can incorporate a gradient-based stochastic variational inference scheme for training latent SDEs.

Our method opens up a broad set of opportunities for fitting any differentiable SDE model, such as the mutation and selection parameters in Wright-Fisher models (Ewens, 2012), derivative models in finance, or infinitely-deep Bayesian neural networks (Peluchetti and Favaro, 2019). In addition, the latent SDE model enabled by our framework can be extended to include structure, domain knowledge, and stationarity constraints (Ma et al., 2015) in the prior process.

**Appendix I. Proof of Theorem C.1**

**Proof** [Proof of Theorem C.1] We have \( J_{s,t}(z) = \nabla \tilde{\Psi}_{s,t}(z) \), where \( \tilde{\Psi}_{s,t}(z) \) is defined in (8). Now we take the gradient with respect to \( z \) on both sides. The solution is differentiable with respect to \( z \) and we may differentiate under the stochastic integral (Kunita, 2019, Proposition 2.4.3). Theorem 3.4.3 Kunita (2019) is sufficient for the regularity conditions required. Since \( K_{s,t}(z) = J_{s,t}(z)^{-1} \), applying the Stratonovich version of Itô’s formula to (9), we have (10).

**Appendix J. Proof of Theorem C.2**

**Proof** [Proof of Theorem C.2] We want to bound the difference

\[
|F(G(z, W_\cdot), W_\cdot) - F_h(G_h(z, W_\cdot), W_\cdot)| \\
\leq |F(G(z, W_\cdot), W_\cdot) - F(G_h(z, W_\cdot), W_\cdot)| + |F(G_h(z, W_\cdot), W_\cdot) - F_h(G_h(z, W_\cdot), W_\cdot)| \\
=: I_1 + I_2.
\]
For notational convenience we suppress $z$ and $W$.

Bounding $I_1$. Let $\epsilon > 0$ be given. Since $G_h \to G$ in probability, there exist $M_1 > 0$ and $h_0 > 0$ such that

$$
P(\|G\| > M_1) < \epsilon, \quad P(\|G_h\| > 2M_1) < \epsilon, \quad \text{for all } h \leq h_0.
$$

Since the SDE defines a stochastic flow of diffeomorphisms, there exists a finite random variable $C_2$ such that $\sup_{\|z\| \leq 2M_1} \|\nabla_x F\| \leq C_2$, and there exists $M_2 > 0$ such that $P(\|C_2\| > M_2) < \epsilon$. Given $M_2$, there exists $h_1 > 0$ such that $P(\|G_h - G\| > \epsilon M_2) < \epsilon$, for all $h \leq h_1$.

Now suppose $h \leq \min\{h_0, h_1\}$. Then, by the union bound, with probability at least $1 - 4\epsilon$, we have

$$
\|G\| \leq M_1, \quad \|G_h\| \leq 2M_1, \quad \|C_2\| \leq M_2, \quad \|G - G_h\| \leq \epsilon M_2
$$

On this event, we have

$$
I_1 = \|F(G) - F(G_h)\| \leq C_2\|G - G_h\| \leq M_2 \frac{\epsilon}{M_2} = \epsilon.
$$

Thus, we have shown that $I_1$ converges to 0 in probability as $h \to 0$.

Bounding $I_2$. The idea is similar. By condition (ii), we have

$$
\lim_{h \to 0} \sup_{\|z_T\| \leq M} |F_h(z_T) - F(z_T)| = 0
$$

in probability. Using this and condition (i), for given $\epsilon > 0$, there exist $M > 0$ and $h_0 > 0$ such that for $h \geq h_0$, we have

$$
\|G_h\| \leq M \quad \text{and} \quad \sup_{\|z_T\| \leq M} |F_h(z_T) - F(z_T)| < \epsilon
$$

with probability at least $1 - \epsilon$. On this event, we have

$$
|F(G_h) - F_h(G_h)| \leq \sup_{\|z_T\| \leq M} |F_h(z_T) - F(z_T)| < \epsilon.
$$

Thus $I_2$ also converges to 0 in probability.

Appendix K. Stochastic Adjoint has Commutative Noise when Original SDE has Diagonal Noise

Recall the Stratonovich SDE (7) with drift and diffusion functions $b, \sigma_1, \ldots, \sigma_m \in \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$ governed by a set of parameters $\theta \in \mathbb{R}^p$. Consider the augmented state composed of the original state and parameters $Y_t = (Z_t^\top, \theta^\top)^\top$. The augmented state satisfies a Stratonovich SDE with the drift function $f(y, t) = (b(z, t)^\top, 0_p^\top)^\top$ and diffusion functions
Thus, we can write out the backward SDEs for the adjoint processes of the state and property (Rößler, 2004) if notation slightly to let
backward SDE (8). We write the overall state as can be written as:
\[ A_t^y = A_T^y + \int_t^T \nabla f(Y_s, s)^\top A_s^y \, ds + \sum_{i=1}^m \int_t^T \nabla g_i(Y_s, s)^\top A_s^y \circ \mathcal{d}W_s^{(i)}. \]
By definitions of \( f \) and \( g_i \), the Jacobian matrices \( \nabla f(x, s) \) and \( \nabla g_i(x, s) \) can be written as:
\[ \nabla f(y, s) = \begin{pmatrix} \frac{\partial b(z, s)}{\partial z} & 0_{d \times p} \\ 0_{p \times d} & 0_{p \times p} \end{pmatrix} \in \mathbb{R}^{(d+p) \times (d+p)}, \quad \nabla g_i(y, s) = \begin{pmatrix} \frac{\partial \sigma_i(z, s)}{\partial z} & 0_{d \times p} \\ 0_{p \times d} & 0_{p \times p} \end{pmatrix} \in \mathbb{R}^{(d+p) \times (d+p)}. \]
Thus, we can write out the backward SDEs for the adjoint processes of the state and parameters separately:
\[ A_t^z = A_T^z + \int_t^T \frac{\partial b(z, s)}{\partial z} |_{z=Z_s} A_s^z \, ds + \sum_{i=1}^m \int_t^T \frac{\partial \sigma_i(z, s)}{\partial z} |_{z=Z_s} A_s^z \circ \mathcal{d}W_s^{(i)}, \quad (14) \]
\[ A_t^\theta = A_T^\theta + \int_t^T \frac{\partial b(z, s)}{\partial \theta} |_{z=Z_s} A_s^\theta \, ds + \sum_{i=1}^m \int_t^T \frac{\partial \sigma_i(z, s)}{\partial \theta} |_{z=Z_s} A_s^\theta \circ \mathcal{d}W_s^{(i)}. \]
Now assume the original SDE has diagonal noise. Then, \( m = d \) and Jacobian matrix \( \nabla \sigma_i(z) \) can be written as:
\[ \frac{\partial \sigma_i(z)}{\partial z} = \begin{pmatrix} 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & \cdots & 0 & \frac{\partial \sigma_i(z)}{\partial z_i} & \cdots & 0 \\ 0 & \cdots & 0 & 0 & \cdots & 0 \end{pmatrix}. \quad (15) \]
Consider the adjoint process for the augmented state along with the backward flow of the backward SDE (8). We write the overall state as \( X_t = (Z_t^T, (A_t^z)^T, (A_t^\theta)^T)^T \), where we abuse notation slightly to let \( \{Z_t\}_{t \in \mathbb{T}} \) denote the backward flow process. Then, by (14) and (15), \( \{X_t\}_{t \in \mathbb{T}} \) satisfies a backward SDE with a diffusion function that can be written as:
\[ G(x) = \begin{pmatrix} -\sigma_{1,1}(z_1) & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & -\sigma_{d,d}(z_d) \\ \frac{\partial \sigma_{1,1}(z_1)}{\partial z_1} a_1^z & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & \frac{\partial \sigma_{d,d}(z_d)}{\partial z_d} a_d^z \\ \frac{\partial \sigma_{1,1}(z_1)}{\partial \theta_1} a_1^z & \cdots & \cdots & \frac{\partial \sigma_{d,d}(z_d)}{\partial \theta_d} a_d^z \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \frac{\partial \sigma_{1,1}(z_1)}{\partial \theta_p} a_1^z & \cdots & \cdots & \frac{\partial \sigma_{d,d}(z_d)}{\partial \theta_p} a_d^z \end{pmatrix} \in \mathbb{R}^{(2d+p) \times d}. \quad (16) \]
Recall, for an SDE with diffusion function \( \Sigma(x) \in \mathbb{R}^{d \times m} \), it is said to satisfy the commutativity property (Rößler, 2004) if
\[ \sum_{i=1}^d \Sigma_{i,j_2}(x) \frac{\partial \Sigma_{k,j_1}(x)}{\partial x_i} = \sum_{i=1}^d \Sigma_{i,j_1}(x) \frac{\partial \Sigma_{k,j_2}(x)}{\partial x_i}, \quad (17) \]
for all \( j_1, j_2 \in [m] \) and \( k \in [d] \). When an SDE has commutative noise, the computationally intensive double Itô integrals (and the Lévy areas) need not be simulated by having the numerical scheme take advantage of the following property of iterated integrals (Ilie et al., 2015):

\[
\int_t^s \int_u^s dW_r^{(i)} dW_u^{(j)} + \int_t^s \int_u^s dW_r^{(j)} dW_u^{(i)} = \Delta W^{(i)} \Delta W^{(j)},
\]

where the Brownian motion increment \( \Delta W^{(i)} = W_t^{(i)} - W_s^{(i)} \) for \( i \in [m] \) can be easily sampled.

To see that the diffusion function (16) indeed satisfies the commutativity condition (17), we consider several cases:

- \( k = 1, \ldots, d \): Both LHS and RHS are zero unless \( j_1 = j_2 = k \), since for \( \Sigma_{i,j_2}(x) \frac{\partial \Sigma_{k,j_1}(x)}{\partial x_i} \) to be non-zero, \( i = j_1 = j_2 = k \).

- \( k = d + 1 \ldots, 2d \): Similar to the case above.

- \( k = 2d + 1 \ldots, 2d + p \): Write \( k = 2d + l \), where \( l \in [p] \). Both LHS and RHS are zero unless \( j_1 = j_2 = l \), since for \( \Sigma_{i,j_2}(x) \frac{\partial \Sigma_{k,j_1}(x)}{\partial x_i} \) to be non-zero \( i = l \) or \( i = d + l \) and \( j_1 = j_2 = l \).

Since in all scenarios, LHS = RHS, we conclude that the commutativity condition holds.

Finally, we comment that the Milstein scheme for the stochastic adjoint of diagonal noise SDEs can be implemented such that during each iteration of the backward solve, vjp is only called a number of times constant with respect to the dimensionality of the original SDE.

### Appendix L. Stochastic Adjoint Implementation

We include the core code of our stochastic adjoint framework, assuming access to a callable Brownian motion \( \text{bm} \), an Euler-Maruyama integrator \text{ito_int_diag} for diagonal noise SDEs, and several other helper functions whose purposes can be inferred from their names.

class _SdeintAdjointMethod ( torch . autograd . Function ) :
    @staticmethod
    def forward ( ctx , *args ) :
        \( (y0, f, g, ts, flat_params_f, flat_params_g, dt, bm) = ( \)
        \( \text{ctx} . f, \text{ctx} . g, \text{ctx} . dt, \text{ctx} . bm = f, g, dt, bm \)

        def g_prod ( t , y , noise ) :
            g_eval = g ( t = t , y = y )
            g_prod_eval = tuple (
                g_eval_i , noise_i for g_eval_i , noise_i in _zip ( g_eval , noise ))
            return g_prod_eval

        with torch . no_grad () :
            ans = ito_int_diag ( f , g_prod , y0 , ts , dt , bm)
        ctx . save_for_backward ( ts , flat_params_f , flat_params_g , *ans)

    def backward ( ctx , *grads ) :
        \( (\text{grads}[0], \text{grads}[1], \text{grads}[2], \text{grads}[3], \text{grads}[4], \text{grads}[5]) = ( \)
        \( \text{ctx} . f , \text{ctx} . g , \text{ctx} . dt , \text{ctx} . bm \)

        def g_prod ( t , y , noise ) :
            g_eval = g ( t = t , y = y )
            g_prod_eval = tuple (
                g_eval_i , noise_i for g_eval_i , noise_i in _zip ( g_eval , noise ))
            return g_prod_eval

        with torch . no_grad () :
            ans = ito_int_diag ( f , g_prod , y0 , ts , dt , bm)
        ctx . save_for_backward ( ts , flat_params_f , flat_params_g , *ans)
return ans

@staticmethod
def backward(ctx, *grad_outputs):
    ts, flat_params_f, flat_params_g, *ans = ctx.saved_tensors
    f, g, dt, bm = ctx.f, ctx.g, ctx.dt, ctx.bm
    f_params, g_params = tuple(f.parameters()), tuple(g.parameters())
    n_tensors = len(ans)

    def aug_f(t, y_aug):
        y, adj_y = y_aug[:n_tensors], y_aug[n_tensors:2*n_tensors]

        with torch.enable_grad():
            y = tuple(y_.detach().requires_grad_(True) for y_ in y)
            adj_y = tuple(adj_y_.detach() for adj_y_ in adj_y)

            g_eval = g(t=-t, y=y)
            gdg = torch.autograd.grad(
                outputs=g_eval,
                inputs=y,
                grad_outputs=g_eval,
                retain_graph=True, create_graph=True)
            f_eval = f(t=-t, y=y)
            f_eval = _sequence_subtract(gdg, f_eval)  # -f + gdg.

            vjp_y_and_params = torch.autograd.grad(
                outputs=f_eval,
                inputs=y + f_params + g_params,
                grad_outputs=tuple(-adj_y_ for adj_y_ in adj_y),
                retain_graph=True, allow_unused=True)
            vjp_y = vjp_y_and_params[:n_tensors]
            vjp_f = vjp_y_and_params[-len(f_params + g_params):-len(g_params)]
            vjp_g = vjp_y_and_params[-len(g_params):]

            vjp_y = tuple(
                torch.zeros_like(y_) if vjp_y_ is None
                else vjp_y_ for vjp_y_, y_ in zip(vjp_y, y))

        adj_times_dgdx = torch.autograd.grad(
            outputs=g_eval,
            inputs=y,
            grad_outputs=adj_y,
            create_graph=True)

        extra_vjp_y_and_params = torch.autograd.grad(
            outputs=g_eval,
            inputs=y + f_params + g_params,
            grad_outputs=adj_times_dgdx,
            allow_unused=True)
        extra_vjp_y = extra_vjp_y_and_params[:n_tensors]
        extra_vjp_f = extra_vjp_y_and_params[-len(f_params + g_params):-len(g_params)]
        extra_vjp_g = extra_vjp_y_and_params[-len(g_params):]

        extra_vjp_y = tuple(
            torch.zeros_like(y_) if extra_vjp_y_ is None
            else extra_vjp_y_ for extra_vjp_y_, y_ in zip(extra_vjp_y, y))

    return ans
Scalable Gradients and Variational Inference for SDEs

```python
vjp_y = _sequence_add(vjp_y, extra_vjp_y)
vjp_f = vjp_f + extra_vjp_f
vjp_g = vjp_g + extra_vjp_g

return (*f_eval, *vjp_y, vjp_f, vjp_g)
def aug_g_prod(t, y_aug, noise):
y, adj_y = y_aug[:n_tensors], y_aug[n_tensors:2 * n_tensors]

with torch.enable_grad():
y = tuple(y_.detach().requires_grad_(True) for y_ in y)
adj_y = tuple(adj_y_.detach() for adj_y_ in adj_y)

g_eval = tuple(-g_ for g_ in g(t=-t, y=y))
vjp_y_and_params = torch.autograd.grad(
    outputs=g_eval,
    inputs=y + f_params + g_params,
    grad_outputs=tuple(-noise_ * adj_y_ for noise_, adj_y_ in zip(noise, adj_y)),
    allow_unused=True)
vjp_y = vjp_y_and_params[:n_tensors]
vjp_f = vjp_y_and_params[-len(f_params + g_params): -len(g_params)]
vjp_g = vjp_y_and_params[-len(g_params):]

vjp_y = tuple(
torch.zeros_like(y_) if vjp_y_ is None
else vjp_y_ for vjp_y_, y_ in zip(vjp_y, y)
)
g_prod_eval = _sequence_multiply(g_eval, noise)

return (*g_prod_eval, *vjp_y, vjp_f, vjp_g)
def aug_bm(t):
    return tuple(-bmi for bmi in bm(-t))

T = ans[0].size(0)
with torch.no_grad():
    adj_y = tuple(grad_outputs_[-1] for grad_outputs_ in grad_outputs)
adj_params_f = torch.zeros_like(flat_params_f)
adj_params_g = torch.zeros_like(flat_params_g)

for i in range(T - 1, 0, -1):
    ans_i = tuple(ans_[i] for ans_ in ans)
aug_y0 = (*ans_i, *adj_y, adj_params_f, adj_params_g)
aug_ans = ito_int_diag(
    f=aug_f, g_prod=aug_g_prod, y0=aug_y0,
    ts=torch.tensor([-ts[i], -ts[i - 1]]).to(ts),
    dt=dt, bm=aug_bm)
adj_y = aug_ans[n_tensors:2 * n_tensors]
adj_params_f, adj_params_g = aug_ans[-2], aug_ans[-1]

# Take the result at the end time.
adj_y = tuple(adj_y_[1] for adj_y_ in adj_y)
adj_params_f, adj_params_g = adj_params_f[1], adj_params_g[1]
```

23
# Accumulate gradients at intermediate points.
adj_y = _sequence_add(
    adj_y, tuple(grad_outputs_[i - 1] for grad_outputs_ in grad_outputs)
)

return (*adj_y, None, None, None, adj_params_f, adj_params_g, None, None)

Appendix M. Test Problems

In the following, $\alpha, \beta$, and $p$ are parameters of SDEs, and $x_0$ is a fixed initial value.

Example 1.

$$dX_t = \alpha X_t \, dt + \beta X_t \, dW_t, \quad X_0 = x_0.$$  

Analytical solution:

$$X_t = X_0 e^{(\beta - \alpha^2)t + \alpha W_t}.$$  

Example 2.

$$dX_t = -p^2 \sin(X_t) \cos^3(X_t) \, dt + p \cos^2(X_t) \, dW_t, \quad X_0 = x_0$$

Analytical solution:

$$X_t = \arctan(p W_t + \tan(X_0)).$$  

Example 3.

$$dX_t = \left(-\frac{\beta}{\sqrt{1 + t}} - \frac{1}{2(1 + t)} X_t\right) \, dt + \frac{\alpha \beta}{\sqrt{1 + t}} \, dW_t, \quad X_0 = x_0$$

Analytical solution:

$$X_t = \frac{1}{\sqrt{1 + t}} X_0 + \frac{\beta}{\sqrt{1 + t}} (t + \alpha W_t).$$

In each numerical experiment, we duplicate the equation 10 times to obtain a system of SDEs where each dimension had their own parameter values sampled from the standard Gaussian distribution and then passed through a sigmoid to ensure positivity. Moreover, we also sample the initial value for each dimension from a Gaussian distribution.

Appendix N. Synthetic Datasets

We consider training latent SDE models with our adjoint framework to recover (1) a 1D Geometric Brownian motion, and (2) a 3D stochastic Lorenz attractor process. The main objective is to verify that the learned posterior is able to reconstruct the training data, and the learned prior exhibit stochastic behavior. We jointly optimize the variational free energy (6) with respect to parameters of the prior and posterior distributions at the initial
latent state $z_0$, the prior and posterior drift, the diffusion function, the encoder, and the decoder. We include the details of dataset and architecture in Appendix N.1.

For the stochastic Lorenz attractor, not only is the model able to reconstruct the data well, but also the learned prior process can produce bimodal samples in both data and latent space. This is showcased in the last row of Figure 4, where once the initial position sampled from the learned prior distribution is fixed, the latent and data space samples cluster around two modes. Note that this cannot be achieved by a latent ODE, where trajectories are determined once their initial latent state is determined.

Figure 4: Additional visualizations of learned posterior and prior dynamics on the synthetic stochastic Lorenz attractor dataset. First row displays the true data and posterior reconstructions. Second row displays samples with initial latent state for each trajectory is sampled independently. Third row displays samples with initial latent state sampled and fixed to be the same for different trajectories.

See Figure 4 for additional visualization on the synthetic Lorenz attractor dataset. See Figure 5 for visualization on the synthetic geometric Brownian motion dataset. We comment that for the second example, the posterior reconstructs the data well, and the prior process exhibit behavior of the data. However, from the third row, we can observe that the prior process is learned such that most of the uncertainty is account for in the initial latent state. We leave the investigation of more interpretable prior process for future work.

N.1. Synthetic Datasets Configuration

N.1.1. Geometric Brownian Motion

Consider a geometric Brownian motion SDE:

$$dX_t = \mu X_t \, dt + \sigma X_t \, dW_t, \quad X_0 = x_0.$$
Scalable Gradients and Variational Inference for SDEs

Figure 5: Visualizations of learned posterior and prior dynamics on the synthetic geometric Brownian motion dataset. First row displays the true data and posterior reconstructions. Orange contour covers 95% of 512 samples. Second row displays samples with initial latent state for each trajectory is sampled independently. Third row displays samples with initial latent state sampled and fixed to be the same for different trajectories.

We use $\mu = 1$, $\sigma = 0.5$, and $x_0 = 0.1 + \epsilon$ as the ground-truth model, where $\epsilon \sim N(0,0.03^2)$. We sample 1024 time series, each of which is observed at intervals of 0.02 from time 0 to time 1. We corrupt this data using Gaussian noise with mean zero and standard deviation 0.01.

To recover the dynamics, we use a GRU-based Cho et al. (2014) latent SDE model where the GRU has 1 layer and 100 hidden units, the prior and posterior drift functions are MLPs with 1 hidden layer of 100 units, and the diffusion function is an MLP with 1 hidden layer of 100 hidden units and the sigmoid activation applied at the end. The drift function in the posterior is time-inhomogenous in the sense that it takes in a context vector of size 1 at each observation that is output by the GRU from running backwards after processing all future observations. The decoder is a linear mapping from a 4 dimensional latent space to observation space. For all nonlinearities, we use the softplus function. We fix the observation model to be Gaussian with noise standard deviation 0.01.

We optimize the model jointly with respect to the parameters of a Gaussian distribution for initial latent state distribution, the prior and posterior drift functions, the diffusion function, the GRU encoder, and the decoder. We use a fixed discretization with step size of 0.01 in both the forward and backward pass. We use the Adam optimizer Kingma and Ba (2014) with an initial learning rate of 0.01 that is decay by a factor of 0.999 after each iteration. We use a linear KL annealing schedule over the first 50 iterations.
N.1.2. Stochastic Lorenz Attractor

Consider a stochastic Lorenz attractor SDE with diagonal noise:

\[ dX_t = \sigma (Y_t - X_t) \, dt + \alpha_x \, dW_t, \quad X_0 = x_0, \]

\[ dY_t = (X_t (\rho - Z_t) - Y_t) \, dt + \alpha_y \, dW_t, \quad Y_0 = y_0, \]

\[ dZ_t = (X_t Y_t - \beta Z_t) \, dt + \alpha_z \, dW_t, \quad Z_0 = z_0. \]

We use \( \sigma = 10, \rho = 28, \beta = 8/3, (\alpha_x, \alpha_y, \alpha_z) = (1, 1, 1, 1), \) and \((x_0, y_0, z_0)\) sampled from the standard Gaussian distribution as the ground-truth model. We sample 1024 time series, each of which is observed at intervals of 0.025 from time 0 to time 1. We normalize these samples by their mean and standard deviation across each dimension and corrupt this data by Gaussian noise with mean zero and standard deviation 0.01.

We use the same architecture and training procedure for the latent SDE model as in the geometric Brownian motion section, except that the diffusion function consists of four small neural networks, each for a single dimension of the latent SDE.

Appendix O. Settings for Learning from Motion Capture Dataset

We follow the preprocessing used by Wang et al. (2007). Following Yildiz et al. (2019), we use a fully connected network to encode the first three observations of each sequence and thereafter predicted the remaining sequence. Note the current choice of encoder is for comparing fairly to models in the existing literature, and it may be extended to be a recurrent or attention model Vaswani et al. (2017) to enhance performance. The overall architecture is described in Appendix O and is similar to that of ODE\(^2\)VAE Yıldız et al. (2019) with a similar number of parameters. We also use a fixed step size that is 1/5 of smallest interval between any two observations Yıldız et al. (2019).

We train latent ODE and latent SDE models with the Adam optimizer Kingma and Ba (2014) and its default hyperparameter settings, with an initial learning rate of 0.01 that is exponentially decayed with rate 0.999 during each iteration. We perform validation over the number of training iterations, KL penalty Higgins et al. (2017), and KL annealing schedule. All models were trained for at most 400 iterations, where we start to observe severe overfitting for most model instances.

We use a latent SDE model with an MLP encoder which takes in the first three frames and outputs the mean and log-variance of the variational distribution of the initial latent state and a context vector. The decoder has a similar architecture as that for the ODE\(^2\)VAE model Yıldız et al. (2019) and projects the 6-dimensional latent state into the 50-dimensional observation space. The posterior drift function takes in a 3-dimensional context vector output by the encoder and the current state and time, whereas the prior drift only takes in the current state and time. The diffusion function is composed of multiple small neural nets, each producing a scalar for the corresponding dimension such that the posterior SDE has diagonal noise. We comment that the overall parameter count of our model (11605) is smaller than that of ODE\(^2\)VAE for the same task (12157).

The latent ODE baseline was implemented with a similar architecture, except is does not have the diffusion and prior drift components, and its vector field defining the ODE does not
Figure 6: Architecture specifics for the latent SDE model used to train on the mocap dataset. First row from left to right are the encoder and decoder. Second row from left to right are the prior drift, posterior drift, and diffusion functions.

take in a context vector. Therefore, the model has slightly fewer parameters (10573) than the latent SDE model. See Figure 6 for overall details of the architecture.

The main hyperparameter we tuned was the coefficient for reweighting the KL. For both the latent ODE and SDE, we considered training the model with a reweighting coefficient in \{1, 0.1, 0.01, 0.001\}, either with or without a linear KL annealing schedule that increased from 0 to the prescribed value over the first 200 iterations of training.