Efficient and Accurate Gradients for Neural SDEs

Patrick Kidger [1] **James Foster** [1] **Xuechen Li** [2] **Terry Lyons** [1]

[1] University of Oxford; The Alan Turing Insitute [2] Stanford
{kidger, foster, tlyons}@maths.ox.ac.uk
lxuechen@cs.toronto.edu

Abstract

Neural SDEs combine many of the best qualities of both RNNs and SDEs: memory efficient training, high-capacity function approximation, and strong priors on model space. This makes them a natural choice for modelling many types of temporal dynamics. Training a Neural SDE (either as a VAE or as a GAN) requires backpropagating through an SDE solve. This may be done by solving a backwards-in-time SDE whose solution is the desired parameter gradients. However, this has previously suffered from severe speed and accuracy issues, due to high computational cost and numerical truncation errors. Here, we overcome these issues through several technical innovations. First, we introduce the *reversible Heun method*. This is a new SDE solver that is *algebraically reversible*: eliminating numerical gradient errors, and the first such solver of which we are aware. Moreover it requires half as many function evaluations as comparable solvers, giving up to a $1.98\times$ speedup. Second, we introduce the *Brownian Interval*: a new, fast, memory efficient, and exact way of sampling *and reconstructing* Brownian motion. With this we obtain up to a $10.6\times$ speed improvement over previous techniques, which in contrast are both approximate and relatively slow. Third, when specifically training Neural SDEs as GANs (Kidger et al. 2021), we demonstrate how SDE-GANs may be trained through careful weight clipping and choice of activation function. This reduces computational cost (giving up to a $1.87\times$ speedup) and removes the numerical truncation errors associated with gradient penalty. Altogether, we outperform the state-of-the-art by substantial margins, with respect to training speed, and with respect to classification, prediction, and MMD test metrics. We have contributed implementations of all of our techniques to the `torchsde` library to help facilitate their adoption.

1 Introduction

Stochastic differential equations Stochastic differential equations have seen widespread use in the mathematical modelling of random phenomena, such as particle systems [1], financial markets [2], population dynamics [3], and genetics [4]. Featuring inherent randomness, then in modern machine learning parlance SDEs are generative models.

Such models have typically been constructed theoretically, and are usually relatively simple. For example the Black–Scholes equation, widely used to model asset prices in financial markets, has only two scalar parameters: a fixed drift and a fixed diffusion [5].

Neural stochastic differential equations Neural stochastic differential equations offer a shift in this paradigm. By parameterising the drift and diffusion of an SDE as neural networks, then modelling capacity is greatly increased, and theoretically arbitrary SDEs may be approximated. (By the universal approximation theorem for neural networks [6, 7].) Several authors have now studied or introduced Neural SDEs; [8, 9, 10, 11, 12, 13, 14, 15, 16] amongst others.

35th Conference on Neural Information Processing Systems (NeurIPS 2021).

Connections to recurrent neural networks A numerically discretised (Neural) SDE may be interpreted as an RNN (featuring a residual connection), whose input is random noise – Brownian motion – and whose output is a generated sample. Subject to a suitable loss function between distributions, such as the KL divergence [15] or Wasserstein distance [16], this may then simply be backpropagated through in the usual way.

Generative time series models SDEs are naturally random. In modern machine learning parlance they are thus generative models. As such we treat Neural SDEs as *generative time series models*.

The (recurrent) neural network-like structure offers high-capacity function approximation, whilst the SDE-like structure offers strong priors on model space, memory efficiency, and deep theoretical connections to a well-understood literature. Relative to the classical SDE literature, Neural SDEs have essentially unprecedented modelling capacity.

(Generative) time series models are of classical interest, with forecasting models such as Holt–Winters [17, 18], ARMA [19] and so on. It has also attracted much recent interest with (besides Neural SDEs) the development of models such as Time Series GAN [20], Latent ODEs [21], GRU-ODE-Bayes [22], ODE^{2}VAE [23], CTFPs [24], Neural ODE Processes [25] and Neural Jump ODEs [26].

1.1 Contributions

We study backpropagation through SDE solvers, in particular to train Neural SDEs, via continuous adjoint methods. We introduce several technical innovations to improve both model performance and the speed of training: in particular to reduce numerical gradient errors to almost zero.

First, we introduce the *reversible Heun method*: a new SDE solver, constructed to be *algebraically reversible*. By matching the truncation errors of the forward and backward passes, the gradients computed via continuous adjoint method are precisely those of the numerical discretisation of the forward pass. This overcomes the typical greatest limitation of continuous adjoint methods – and to the best of our knowledge, is the first algebraically reversible SDE solver to have been developed.

After that, we introduce the *Brownian Interval* as a new way of sampling *and reconstructing* Brownian motion. It is fast, memory efficient and exact. It has an average (modal) time complexity of $\mathcal{O}(1)$, and consumes only $\mathcal{O}(1)$ GPU memory. This is contrast to previous techniques requiring either $\mathcal{O}(T)$ memory, or a choice of approximation error $\varepsilon \ll 1$ and then a time complexity of $\mathcal{O}(\log(1/\varepsilon))$.

Finally, we demonstrate how the Lipschitz condition for the discriminator of an SDE-GAN may be imposed without gradient penalties – instead using careful clipping and the LipSwish activation function – so as to overcome their previous incompatibility with continuous adjoint methods.

Overall, multiple technical innovations provide substantial improvements over the state-of-the-art with respect to training speed, and with respect to classification, prediction, and MMD test metrics.

2 Background

2.1 Neural SDE construction

Certain minimal amount of structure Following Kidger et al. [16], we construct Neural SDEs with a certain minimal amount of structure. Let $T > 0$ be fixed and suppose we wish to model a path-valued random variable $Y_{\text{true}} \colon [0, T] \to \mathbb{R}^y$. The size of y is the dimensionality of the data.[1]

Let $W \colon [0, T] \to \mathbb{R}^w$ be a w-dimensional Brownian motion, and let $V \sim \mathcal{N}(0, I_v,)$ be drawn from a v-dimensional standard multivariate normal. The values w, v are hyperparameters describing the size of the noise. Let

$$\zeta_\theta \colon \mathbb{R}^v \to \mathbb{R}^x, \quad \mu_\theta \colon [0, T] \times \mathbb{R}^x \to \mathbb{R}^x, \quad \sigma_\theta \colon [0, T] \times \mathbb{R}^x \to \mathbb{R}^{x \times w}, \quad \ell_\theta \colon \mathbb{R}^x \to \mathbb{R}^y,$$

where ζ_θ, μ_θ and σ_θ are neural networks and ℓ_θ is affine. Collectively these are parameterised by θ. The dimension x is a hyperparameter describing the size of the hidden state.

[1] In practice we will typically observe some discretised time series sampled from Y_{true}. For ease of presentation we will neglect this detail for now and will return to it in Section 2.3.

We consider Neural SDEs as models of the form

$$X_0 = \zeta_\theta(V), \qquad \mathrm{d}X_t = \mu_\theta(t, X_t)\,\mathrm{d}t + \sigma_\theta(t, X_t) \circ \mathrm{d}W_t, \qquad Y_t = \ell_\theta(X_t), \tag{1}$$

for $t \in [0, T]$, with $X \colon [0, T] \to \mathbb{R}^x$ the (strong) solution to the SDE.[2] The solution X is guaranteed to exist given mild conditions: that μ_θ, σ_θ are Lipschitz, and that $\mathbb{E}_V\left[\zeta_\theta(V)^2\right] < \infty$.

We seek to train this model such that $Y \overset{\mathrm{d}}{\approx} Y_{\text{true}}$. That is to say, the model Y should have approximately the same distribution as the target Y_{true}, for some notion of approximate. (For example, to be similar with respect to the Wasserstein distance).

RNNs as discretised SDEs The minimal amount of structure is chosen to parallel RNNs. The solution X may be interpreted as hidden state, and the ℓ_θ maps the hidden state to the output of the model. In Appendix A we provide sample PyTorch [27] code computing a discretised SDE according to the Euler–Maruyama method. The result is an RNN consuming random noise as input.

2.2 Training criteria for Neural SDEs

Equation (1) produces a random variable $Y \colon [0, T] \to \mathbb{R}^y$ implicitly depending on parameters θ. This model must still be fit to data. This may be done by optimising a distance between the probability distributions (laws) for Y and Y_{true}.

SDE-GANs The Wasserstein distance may be used by constructing a discriminator and training adversarially, as in Kidger et al. [16]. Let $F_\phi(Y) = m_\phi \cdot H_T$, where

$$H_0 = \xi_\phi(Y_0), \qquad \mathrm{d}H_t = f_\phi(t, H_t)\,\mathrm{d}t + g_\phi(t, H_t) \circ \mathrm{d}Y_t, \tag{2}$$

for suitable neural networks ξ_ϕ, f_ϕ, g_ϕ and vector m_ϕ. This is a deterministic function of the generated sample Y. Here $\cdot$ denotes a dot product. They then train with respect to

$$\min_\theta \max_\phi \left(\mathbb{E}_Y\left[F_\phi(Y)\right] - \mathbb{E}_{Y_{\text{true}}}\left[F_\phi(Y_{\text{true}})\right]\right). \tag{3}$$

See Appendix B for additional details on this approach, and in particular how it generalises the classical approach to fitting (calibrating) SDEs.

Latent SDEs Li et al. [15] instead optimise a KL divergence. This consists of constructing an auxiliary process $\widehat{X}$ with drift ν_ϕ parameterised by ϕ, and optimising an expression of the form

$$\min_{\theta,\phi} \mathbb{E}_{W, Y_{\text{true}}}\left[\int_0^T (Y_{\text{true},t} - \ell_\theta(\widehat{X}_t))^2 + \frac{1}{2}\left\|(\sigma_\theta(t, \widehat{X}_t))^{-1}(\mu_\theta(t, \widehat{X}_t) - \nu_\phi(t, \widehat{X}_t, Y_{\text{true}}))\right\|_2^2 \mathrm{d}t\right]. \tag{4}$$

The full construction is moderately technical; see Appendix B for further details.

2.3 Discretised observations

Observations of Y_{true} are typically a discrete time series, rather than a true continuous-time path. This is not a serious hurdle. If training an SDE-GAN, then equation (2) may be evaluated on an interpolation Y_{true} of the observed data. If training a Latent SDE, then ν_ϕ in equation (4) may depend explicitly on the discretised Y_{true}.

2.4 Backpropagation through SDE solves

Whether the loss for our generated sample Y is produced via a Latent SDE or via the discriminator of an SDE-GAN, it is still required to backpropagate from the loss to the parameters θ, ϕ.

Here we use *the continuous adjoint method*. Also known as simply 'the adjoint method', or 'optimise-then-discretise', this has recently attracted much attention in the modern literature on neural differential equations. This exploits the reversibility of a differential equation: as with invertible neural

[2] The notation '$\circ\,\mathrm{d}W_t$' denotes Stratonovich integration. Itô is less efficient; see Appendix C.

networks [28], intermediate computations such as X_t for $t < T$ are reconstructed from output computations, so that they do not need to be held in memory.

Given some Stratonovich SDE

$$\mathrm{d}Z_t = \mu(t, Z_t)\,\mathrm{d}t + \sigma(t, Z_t) \circ \mathrm{d}W_t \quad \text{for } t \in [0, T], \tag{5}$$

and a loss $L \colon \mathbb{R}^z \to \mathbb{R}$ on its terminal value Z_T, then the adjoint process $A_t = {}^{\mathrm{d}L(Z_T)}/_{dZ_t} \in \mathbb{R}^z$ is a (strong) solution to

$$\mathrm{d}A_t^i = -A_t^j \frac{\partial \mu^j}{\partial Z^i}(t, Z_t)\,\mathrm{d}t - A_t^j \frac{\partial \sigma^{j,k}}{\partial Z^i}(t, Z_t) \circ \mathrm{d}W_t^k, \tag{6}$$

which in particular uses the same Brownian motion W as on the forward pass. Equations (5) and (6) may be combined into a single SDE and solved backwards-in-time[3] from $t = T$ to $t = 0$, starting from $Z_T = Z_T$ (computed on the forward pass of equation (5)) and $A_T = {}^{L(Z_T)}/_{dZ_T}$. Then $A_0 = {}^{\mathrm{d}L(Z_T)}/_{Z_0}$ is the desired backpropagated gradient.

Note that we assumed here that the loss L acts only on Z_T, not all of Z. This is not an issue in practice. In both equations (3) and (4), the loss is an integral. As such it may be computed as part of Z in a single SDE solve. This outputs a value at time T, the operation L may simply extract this value from Z_T, and then backpropagation may proceeed as described here.

The main issue is that the two numerical approximations to Z_t, computed in the forward and backward passes of equation (5), are different. This means that the Z_t used as an input in equation (6) has some discrepancy from the forward calculation, and the gradients A_0 suffer some error as a result. (Often exacerbating an already tricky training procedure, such as the adversarial training of SDE-GANs.)

See Appendix C for further discussion on how an SDE solve may be backpropagated through.

2.5 Alternate constructions

There are other uses for Neural SDEs, beyond our scope here. For example Song et al. [30] combine SDEs with score-matching, and Xu et al. [31] use SDEs to represent Bayesian uncertainty over parameters. The techniques introduced in this paper will apply to any backpropagation through an SDE solve.

3 Reversible Heun method

We introduce a new SDE solver, which we refer to as the *reversible Heun method*. Its key property is algebraic reversibility; moreover to the best of our knowledge it is the first SDE solver to exhibit this property.

To fix notation, we consider solving the Stratonovich SDE

$$\mathrm{d}Z_t = \mu(t, Z_t)\,\mathrm{d}t + \sigma(t, Z_t) \circ \mathrm{d}W_t, \tag{7}$$

with known initial condition Z_0.

Solver We begin by selecting a step size Δt, and initialising $t_0 = 0$, $z_0 = \widehat{z}_0 = Z_0$, $\mu_0 = \mu(0, Z_0)$ and $\sigma_0 = \sigma(0, Z_0)$. Let W denote a single sample path of Brownian motion. It is important that the same sample be used for both the forward and backward passes of the algorithm; computationally this may be accomplished by taking W to be a Brownian Interval, which we will introduce in Section 4.

Algorithm 1: Forward pass

Input: $t_n, z_n, \widehat{z}_n, \mu_n, \sigma_n, \Delta t, W$

$$\begin{aligned}
t_{n+1} &= t_n + \Delta t \\
\Delta W_n &= W_{t_{n+1}} - W_{t_n} \\
\widehat{z}_{n+1} &= 2z_n - \widehat{z}_n + \mu_n \Delta t + \sigma_n \Delta W_n \\
\mu_{n+1} &= \mu(t_{n+1}, \widehat{z}_{n+1}) \\
\sigma_{n+1} &= \sigma(t_{n+1}, \widehat{z}_{n+1}) \\
z_{n+1} &= z_n + \frac{1}{2}(\mu_n + \mu_{n+1})\Delta t \\
&\qquad + \frac{1}{2}(\sigma_n + \sigma_{n+1})\Delta W_n
\end{aligned}$$

Output: $t_{n+1}, z_{n+1}, \widehat{z}_{n+1}, \mu_{n+1}, \sigma_{n+1}$

We then iterate Algorithm 1. Suppose $T = N\Delta t$ so that $z_N, \widehat{z}_N, \mu_N, \sigma_N$ are the final output. Then $z_N \approx Z_T$ is returned, whilst $z_N, \widehat{z}_N, \mu_N, \sigma_N$ are all retained for the backward pass.

Nothing else need be saved in memory for the backward pass: in particular no intermediate computations, as would otherwise be typical.

[3] Li et al. [15] give rigorous meaning to this via two-sided filtrations; for the reader familiar with rough path theory then Kidger et al. [29, Appendix A] also give a pathwise interpretation. The reader familiar with neither of these should feel free to intuitively treat Stratonovich (but not Itô) SDEs like ODEs.

Algebraic reversibility The key advantage of the reversible Heun method, and the motivating reason for its use alongside continuous-time adjoint methods, is that it is algebraically reversible. That is, it is possible to reconstruct $(z_n, \widehat{z}_n, \mu_n, \sigma_n)$ from $(z_{n+1}, \widehat{z}_{n+1}, \mu_{n+1}, \sigma_{n+1})$ in closed form. (And without a fixed-point iteration.)

This crucial property will mean that it is possible to backpropagate through the SDE solve, such that the gradients obtained via the continuous adjoint method (equation (6)) *exactly match* the (discretise-then-optimise) gradients obtained by autodifferentiating the numerically discretised forward pass.

In doing so, one of the greatest limitations of continuous adjoint methods is overcome.

To the best of our knowledge, the reversible Heun method is the first algebraically reversible SDE solver.

Computational efficiency A further advantage of the reversible Heun method is computational efficiency. The method requires only a single function evaluation (of both the drift and diffusion) per step. This is in contrast to other Stratonovich solvers (such as the midpoint method or regular Heun's method), which require two function evaluations per step.

Algorithm 2: Backward pass

Input: $t_{n+1}, z_{n+1}, \widehat{z}_{n+1}, \mu_{n+1}, \sigma_{n+1}, \Delta t, W,$
$\frac{\partial L(Z_T)}{\partial z_{n+1}}, \frac{\partial L(Z_T)}{\partial \widehat{z}_{n+1}}, \frac{\partial L(Z_T)}{\partial \mu_{n+1}}, \frac{\partial L(Z_T)}{\partial \sigma_{n+1}}$

Reverse step

$$\begin{aligned} t_n &= t_{n+1} - \Delta t \\ \Delta W_n &= W_{t_{n+1}} - W_{t_n} \\ \widehat{z}_n &= 2z_{n+1} - \widehat{z}_{n+1} - \mu_{n+1}\Delta t - \sigma_{n+1}\Delta W_n \\ \mu_n &= \mu(t_n, \widehat{z}_n) \\ \sigma_n &= \sigma(t_n, \widehat{z}_n) \\ z_n &= z_{n+1} - \frac{1}{2}(\mu_n + \mu_{n+1})\Delta t \\ &\quad - \frac{1}{2}(\sigma_n + \sigma_{n+1})\Delta W_n \end{aligned}$$

Local forward

$$z_{n+1}, \widehat{z}_{n+1}, \mu_{n+1}, \sigma_{n+1} = \text{Forward}(t_n, z_n, \widehat{z}_n, \mu_n, \sigma_n, \Delta t, W)$$

Local backward

$$\frac{\partial L(Z_T)}{\partial(z_n, \widehat{z}_n, \mu_n, \sigma_n)} = \frac{\partial L(Z_T)}{\partial(z_{n+1}, \widehat{z}_{n+1}, \mu_{n+1}, \sigma_{n+1})} \cdot \frac{\partial(z_{n+1}, \widehat{z}_{n+1}, \mu_{n+1}, \sigma_{n+1})}{\partial(z_n, \widehat{z}_n, \mu_n, \sigma_n)}$$

Output: $t_n, z_n, \widehat{z}_n, \mu_n, \sigma_n,$
$\frac{\partial L(Z_T)}{\partial z_n}, \frac{\partial L(Z_T)}{\partial \widehat{z}_n}, \frac{\partial L(Z_T)}{\partial \mu_n}, \frac{\partial L(Z_T)}{\partial \sigma_n}$

Convergence of the solver When applied to the Stratonovich SDE (7), the reversible Heun method exhibits strong convergence of order 0.5; the same as the usual Heun's method.

Theorem. *Let $\{z_n\}$ denote the numerical solution of (7) obtained by Algorithm 1 with a constant step size Δt and assume sufficient regularity of μ and σ. Then there exists a constant $C > 0$ so that*

$$\mathbb{E}\Big[\big\|z_N - Z_T\big\|_2\Big] \leq C\sqrt{\Delta t}\,,$$

for small Δt. That is, strong convergence of order 0.5. If σ is constant, then this improves to order 1.

The key idea in the proof is to consider two adjacent steps of the SDE solver. Then the update $\widehat{z}_n \mapsto \widehat{z}_{n+2}$ becomes a step of a midpoint method, whilst $z_n \mapsto z_{n+1}$ is similar to Heun's method. This makes it possible to show that $\{\widehat{z}_n\}$ and $\{z_n\}$ stay close together: $\mathbb{E}\big[\|z_n - \widehat{z}_n\|_2^4\big] \sim O(\Delta t^2)$. With this $\mathbb{L}_4$ bound on $z - \widehat{z}$, we can then apply standard lines of argument from the numerical SDE literature. Chaining together local mean squared error estimates, we obtain $\mathbb{E}\big[\|z_N - Z_T\|_2^2\big] \sim O(\Delta t)$.

See Appendix D for the full proof. We additionally consider stability in the ODE setting. Whilst the method is not A-stable, we do show it has the same absolute stability region for a linear test equation as the (reversible) asynchronous leapfrog integrator proposed for Neural ODEs in Zhuang et al. [32].

Precise gradients The backward pass is shown in Algorithm 2. As the same numerical solution $\{z_n\}$ is recovered on both the forward and backward passes – exhibiting the same truncation errors – then the computed gradients are precisely the (discretise-then-optimise) gradients of the numerical discretisation of the forward pass.

Each $\partial L(Z_T)/\partial z_n \approx A_{n\Delta t}$, where A is the adjoint variable of equation (6).

This is unlike the case of solving equation (6) via standard numerical techniques, for which small or adaptive step sizes are necessary to obtain useful gradients [15].

Table 1: SDE-GAN on weights dataset; Latent SDE on air quality dataset. Mean $\pm$ standard deviation averaged over three runs.

Dataset, Solver	Label classification accuracy (%)	MMD ($\times 10^{-2}$)	Training time
Weights, Midpoint	—	4.38 ± 0.67	5.12 ± 0.01 days
Weights, Reversible Heun	—	$\mathbf{1.75 \pm 0.3}$	$\mathbf{2.59 \pm 0.05}$ days
Air quality, Midpoint	46.3 ± 5.1	0.591 ± 0.206	5.58 ± 0.54 hours
Air quality, Reversible Heun	$\mathbf{49.2 \pm 0.02}$	$\mathbf{0.472 \pm 0.290}$	$\mathbf{4.47 \pm 0.31}$ hours

3.1 Experiments

We validate the empirical performance of the reversible Heun method. For space, we present abbreviated details and results here. See Appendix F for details of the hyperparameter optimisation procedure, test metric definitions, and so on, and for further results on additional datasets and additional metrics.

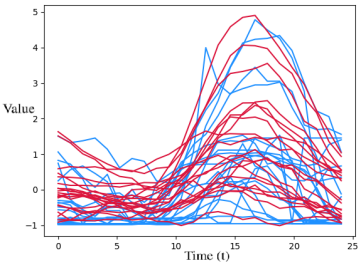

Figure 1: Samples (red) from Latent SDE on O_3 ozone channel of air quality dataset (blue).

Versus midpoint We begin by comparing the reversible Heun method with the midpoint method, which also converges to the Stratonovich solution. We train an SDE-GAN on a dataset of weight trajectories evolving under stochastic gradient descent, and train a Latent SDE on a dataset of air quality over Beijing.

See Table 1. Due to the reduced number of vector field evaluations, we find that training speed roughly doubles ($1.98\times$) on the weights dataset, whilst its numerically precise gradients substantially improve the test metrics (comparing generated samples to a held-out test set). Similar behaviour is observed on the air quality dataset, with substantial test metric improvements and a training speed improvement of $1.25\times$.

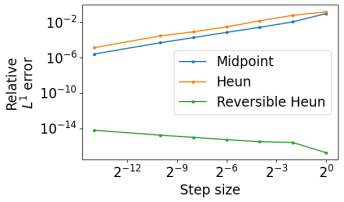

Figure 2: Relative error in gradient calculation.

Samples We verify that samples from a model using reversible Heun resemble that of the original dataset: in Figure 1 we show the Latent SDE on the ozone concentration over Beijing.

Gradient error We investigate the numerical error made in solving (6), compared to the (discretise-then-optimise) gradients of the numerically discretised forward pass. We fix a test problem (differentiating a small Neural SDE) and vary the step size and solver; see Figure 2. The error made in standard solvers is very large (but does at least decrease with step size). The reversible Heun method produces results accurate to floating point error, unattainable by any standard solver.

4 Brownian Interval

Numerically solving an SDE, via the reversible Heun method or via any other numerical solver, requires sampling Brownian motion: this is the input W in Algorithms 1 and 2.

Brownian bridges Mathematically, sampling Brownian motion is straightforward. A fixed-step numerical solver may simply sample independent increments during its time stepping. An adaptive solver (which may reject steps) may use Lévy's Brownian bridge [33] formula to generate increments with the appropriate correlations: letting $W_{a,b} = W_b - W_a \in \mathbb{R}^w$, then for $u < s < t$,

$$W_{u,s} | W_{u,t} = \mathcal{N}\Big(\frac{s-u}{t-u} W_{u,t} \, , \, \frac{(t-s)(s-u)}{t-u} I_w \Big). \tag{8}$$

Brownian reconstruction However, there are computational difficulties. On the backward pass, the same Brownian sample as the forward pass must be used, and potentially at locations other than were measured on the forward pass [15].

Time and memory efficiency The simple but memory intensive approach would be to store every sample made on the forward pass, and then on the backward pass reuse these samples, or sample Brownian noise according to equation (8), as appropriate.

Li et al. [15] instead offer a memory-efficient but time-intensive approach, by introducing the 'Virtual Brownian Tree'. This approximates the real line by a tree of dyadic points. Samples are approximate, and demand deep (slow) traversals of the tree.

Binary tree of (interval, seed) pairs In response to this, we introduce the 'Brownian Interval', which offers memory efficiency, exact samples, and fast query times, all at once. The Brownian Interval is built around a binary tree, each node of which is an interval and a random seed.

The tree starts as a stump consisting of the global interval $[0, T]$ and a randomly generated random seed. New leaf nodes are created as observations of the sample are made. For example, making a first query at $[s, t] \subseteq [0, T]$ (an operation that returns $W_{s,t}$) produces the binary tree shown in Figure 3a. Algorithm 4 in Appendix E gives the formal definition of this procedure. Making a subsequent query at $[u, v]$ with $u < s < v < t$ produces Figure 3b. Using a splittable PRNG [34, 35], each child node has a random seed deterministically produced from the seed of its parent.

The tree is thus designed to completely encode the conditional statistics of a sample of Brownian motion: $W_{s,t}, W_{t,u}$ are completely specified by $t, [s, u], W_{s,u}$, equation (8), and the random seed for $[s, u]$.

In principle this now gives a way to compute $W_{s,t}$; calculating $W_{s,u}$ recursively. Naïvely this would be very slow – recursing to the root on every query – which we cover by augmenting the binary tree structure with a fixed-size Least Recently Used (LRU) cache on the computed increments $W_{s,t}$.

Algorithm 3: Sampling the Brownian Interval

Type: Let *Node* denote a 5-tuple consisting of an interval, a seed, and three optional *Node*s, corresponding to the parent node, and two child nodes, respectively. (Optional as the root has no parent and leaves have no children.)

State: Binary tree with elements of type *Node*, with root $\widehat{I} = ([0, T], \widehat{s}, *, \widehat{I}_{\texttt{left}}, \widehat{I}_{\texttt{right}})$. A *Node* $\widehat{J}$.

Input: Interval $[s, t] \subseteq [0, T]$

```
# The returned 'nodes' is a list of Nodes whose
# intervals partition [s, t]. Practically speaking
# this will usually have only one or two elements.
# Ĵ is a hint for where in the tree we are.
nodes = traverse(Ĵ, [s, t])

def sample(I : Node):
    if I is Î then
        return 𝒩(0, T) sampled with seed ŝ.
    Let I = ([a, b], s, I_parent, I_left, I_right)
    Let I_parent = ([a_p, b_p], s_p, I_pp, I_lp, I_rp)
    W_parent = sample(I_parent)
    if I_i is I_rp then
        W_left = bridge(a_p, b_p, a, W_parent, s)
        return W_parent − W_left
    else
        return bridge(a_p, b_p, b, W_parent, s)
sample = LRUCache(sample)

Ĵ ← nodes[−1]
W_{s,t} = Σ_{I∈nodes} sample(I)
```

Output: $W_{s,t}$

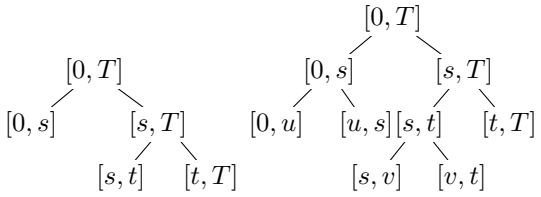

Figure 3: Binary tree of intervals.

See Algorithm 3, where `bridge` denotes equation (8). The operation `traverse` traverses the binary tree to find or create the list of nodes whose disjoint union is the interval of interest, and is defined explicitly as Algorithm 4 in Appendix E.

Additionally see Appendix E for various technical considerations and extensions to this algorithm.

Advantages of the Brownian Interval The LRU cache ensures that queries have an average-case (modal) time complexity of only $\mathcal{O}(1)$: in SDE solvers, subsequent queries are typically close to (and thus conditional on) previous queries. Even given cache misses all the way up the tree, the worst-case time complexity will only be $\mathcal{O}(\log(1/s))$ in the average step size s of the SDE solver. This is in contrast to the Virtual Brownian Tree, which has an (average or worst-case) time complexity of $\mathcal{O}(\log(1/\varepsilon))$ in the approximation error $\varepsilon \ll s$.

Table 2: V. Brownian Tree against Brownian Interval on speed benchmarks, over 100 subintervals.

	SDE solve, speed (seconds)			Doubly sequential access, speed (seconds)		
	Size=1	Size=2560	Size=32768	Size=1	Size=2560	Size=32768
V. B. Tree	1.6×10^{0}	2.0×10^{0}	5.00×10^{2}	2.4×10^{-1}	3.9×10^{-1}	2.9×10^{0}
B. Interval	$\mathbf{8.2\times10^{-1}}$	$\mathbf{1.3\times10^{0}}$	$\mathbf{4.7\times10^{1}}$	$\mathbf{5.0\times10^{-2}}$	$\mathbf{8.0\times10^{-2}}$	$\mathbf{3.5\times10^{-1}}$

Meanwhile the (GPU) memory cost is only $\mathcal{O}(1)$, corresponding to the fixed and constant size of the LRU cache. There is the small additional cost of storing the tree structure itself, but this is held in CPU memory, which for practical purposes is essentially infinite. This is in contrast to simply holding all the Brownian samples in memory, which has a memory cost of $\mathcal{O}(T)$.

Finally, queries are exact because the tree aligns with the query points. This is contrast to the Virtual Brownian Tree, which only produces samples up to some discretisation of the real line at resolution ε.

4.1 Experiments

We benchmark the performance of the Brownian Interval against the Virtual Brownian Tree considered in Li et al. [15]. We include benchmarks corresponding to varying batch sizes, number of sample intervals, and access patterns. For space, just a subset of results are shown. Precise experimental details and further results are available in Appendix F.

See Table 2. We see that the Brownian Interval is uniformly faster than the Virtual Brownian Tree, ranging from $1.5\times$ faster on smaller problems to $10.6\times$ faster on larger problems. Moreover these speed gains are despite the Brownian Interval being written in Python, whilst the Virtual Brownian Tree is carefully optimised and written in C++.

5 Training SDE-GANs without gradient penalty

Kidger et al. [16] train SDEs as GANs, as discussed in Section 2.2, using a neural CDE as a discriminator as in equation (2). They found that only gradient penalty [36] was suitable to enforce the Lipschitz condition, given the recurrent structure of the discriminator.

However gradient penalty requires calculating second derivatives (a 'double-backward'). This complicates the use of continuous adjoint methods: the double-continuous-adjoint introduces substantial truncation error; sufficient to obstruct training and requiring small step sizes to resolve.

Here we overcome this limitation, and moreover do so independently of the possibility of obtaining exact double-gradients via the reversible Heun method. For simplicity we now assume throughout that our discriminator vector fields f_ϕ, g_ϕ are MLPs, which is also the choice we make in practice.

Lipschitz constant one The key point is that the vector fields f_ϕ, g_ϕ of the discriminator must not only be Lipschitz, but must have Lipschitz constant at most one.

Given vector fields with Lipschitz constant λ, then the recurrent structure of the discriminator means that the Lipschitz constant of the overall discriminator will be $\mathcal{O}\left(\lambda^T\right)$. Ensuring $\lambda \approx 1$ with $\lambda \leq 1$ thus enforces that the overall discriminator is Lipschitz with a reasonable Lipschitz constant.

Hard constraint The exponential size of $\mathcal{O}\left(\lambda^T\right)$ means that λ only slightly greater than one is still insufficient for stable training. We found that this ruled out enforcing $\lambda \leq 1$ via soft constraints, via either spectral normalisation [37] or gradient penalty across just vector field evaluations.

Clipping The first part of enforcing this Lipschitz constraint is careful clipping. Considering each linear operation from $\mathbb{R}^a \to \mathbb{R}^b$ as a matrix in $A \in \mathbb{R}^{a\times b}$, then after each gradient update we clip its entries to the region $[-1/b, 1/b]$. Given $x \in \mathbb{R}^a$, then this enforces $\|Ax\|_\infty \leq \|x\|_\infty$.

LipSwish activation functions Next we must pick an activation function with Lipschitz constant at most one. It should additionally be at least twice continuously differentiable to ensure convergence of the numerical SDE solver (Appendix D). In particular this rules out the ReLU.

Table 3: SDE-GAN on OU dataset. Mean $\pm$ standard deviation averaged over three runs.

Solver	Test Metrics			
	Real/fake classification accuracy (%)	Prediction loss	MMD ($\times 10^{-1}$)	Training time (hours)
Midpoint w/ gradient penalty	98.2 ± 2.4	2.71 ± 1.03	2.58 ± 1.81	55.0 ± 27.7
Midpoint w/ clipping	93.9 ± 6.9	1.65 ± 0.17	1.03 ± 0.10	32.5 ± 12.1
Reversible Heun w/ clipping	$\mathbf{67.7 \pm 1.1}$	$\mathbf{1.38 \pm 0.06}$	$\mathbf{0.45 \pm 0.22}$	$\mathbf{29.4 \pm 8.9}$

There remain several admissible choices; we use the LipSwish activation function introduced by Chen et al. [38], defined as $\rho(x) = 0.909\, x \,\mathrm{sigmoid}(x)$. This was carefully constructed to have Lipschitz constant one, and to be smooth. Moreover the SiLU activation function [39, 40, 41] from which it is derived has been reported as an empirically strong choice.

The overall vector fields f_ϕ, g_ϕ of the discriminator consist of linear operations (which are constrained by clipping), adding biases (an operation with Lipschitz constant one), and activation functions (taken to be LipSwish). Thus the Lipschitz constant of the overall vector field is at most one, as desired.

5.1 Experiments

We test the SDE-GAN on a dataset of time-varying Ornstein–Uhlenbeck samples. For space only a subset of results are shown; see Appendix F for further details of the dataset, optimiser, and so on.

See Table 3 for the results. We see that the test metrics substantially improve with clipping, over gradient penalty (which struggles due to numerical errors in the double adjoint). The lack of double backward additionally implies a computational speed-up. This reduced training time from 55 hours to just 33 hours. Switching to reversible Heun additionally and substantially improves the test metrics, and further reduced training time to 29 hours; a speed improvement of $1.87\times$.

6 Discussion

6.1 Available implementation in `torchsde`

To facilitate the use of the techniques introduced here – in particular without requiring a technical background in numerical SDEs – we have contributed implementations of both the reversible Heun method and the Brownian Interval to the open-source `torchsde` [42] package. (In which the Brownian Interval has already become the default choice, due to its speed.)

6.2 Limitations

The reversible Heun method, Brownian Interval, and training of SDE-GANs via clipping, all appear to be strict improvements over previous techniques. Across our experiments we have observed no limitations relative to previous techniques.

6.3 Ethical statement

No significant negative societal impacts are anticipated as a result of this work. A positive environmental impact is anticipated, due to the reduction in compute costs implied by the techniques introduced. See Appendix G for a more in-depth discussion.

7 Conclusion

We have introduced several improvements over the previous state-of-the-art for Neural SDEs, with respect to both training speed and test metrics. This has been accomplished through several novel technical innovations, including a first-of-its-kind algebraically reversible SDE solver; a fast, exact, and memory efficient way of sampling and reconstructing Brownian motion; and the development of SDE-GANs via careful clipping and choice of activation function.

Acknowledgments and Disclosure of Funding

PK was supported by the EPSRC grant EP/L015811/1. PK, JF, TL were supported by the Alan Turing Institute under the EPSRC grant EP/N510129/1. PK thanks Chris Rackauckas for discussions related to the reversible Heun method.

References

[1] W. T. Coffey, Y. P. Kalmykov, and J. T. Waldron. *The Langevin Equation: With Applications to Stochastic Problems in Physics, Chemistry and Electrical Engineering*. World Scientifc, 2012.

[2] S. E. Shreve. *Stochastic Calculus for Finance II: Continuous-Time Models*. Springer Science & Business Media, 2004.

[3] M. Arató. A famous nonlinear stochastic equation (Lotka–Volterra model with diffusion). *Mathematical and Computer Modelling*, 38(7–9):709–726, 2003.

[4] K.-C. Chen, T.-Y. Wang, H.-H. Tseng, C.-Y. F. Huang, and C.-Y. Kao. A stochastic differential equation model for quantifying transcriptional regulatory network in saccharomyces cerevisiae. *Bioinformatics*, 21(12):2883–2890, 2005.

[5] F. Black and M. Scholes. The Pricing of Options and Corporate Liabilities. *Journal of Political Economy*, 81(3):637–654, 1973.

[6] A. Pinkus. Approximation theory of the MLP model in neural networks. *Acta Numerica*, 8: 143–195, 1999.

[7] P. Kidger and T. Lyons. Universal Approximation with Deep Narrow Networks. In *Proceedings of the 33rd Conference on Learning Theory*, pages 2306–2327, 2020.

[8] B. Tzen and M. Raginsky. Neural Stochastic Differential Equations: Deep Latent Gaussian Models in the Diffusion Limit. *arXiv:1905.09883*, 2019.

[9] B. Tzen and M. Raginsky. Theoretical guarantees for sampling and inference in generative models with latent diffusions. In *Proceedings of the 32rd Conference on Learning Theory*, pages 3084–3114, 2019.

[10] J. Jia and A. Benson. Neural Jump Stochastic Differential Equations. In *Advances in Neural Information Processing Systems 32*, pages 9847–9858, 2019.

[11] X. Liu, T. Xiao, S. Si, Q. Cao, S. Kumar, and C.-J. Hsieh. Neural SDE: Stabilizing Neural ODE Networks with Stochastic Noise. *arXiv:1906.02355*, 2019.

[12] L. Kong, J. Sun, and C. Zhang. SDE-Net: Equipping Deep Neural Networks with Uncertainty Estimates. In *Proceedings of the 37th International Conference on Machine Learning*, pages 5405–5415, 2020.

[13] L. Hodgkinson, C. van der Heide, F. Roosta, and M. Mahoney. Stochastic Normalizing Flows. *arXiv:2002.09547*, 2020.

[14] P. Gierjatowicz, M. Sabate-Vidales, D. Šiška, L. Szpruch, and Ž. Žurič. Robust Pricing and Hedging via Neural SDEs. *arXiv:2007.04154*, 2020.

[15] X. Li, T.-K. L. Wong, R. T. Q. Chen, and D. Duvenaud. Scalable Gradients and Variational Inference for Stochastic Differential Equations. *AISTATS*, 2020.

[16] P. Kidger, J. Foster, X. Li, H. Oberhauser, and T. Lyons. Neural SDEs as Infinite-Dimensional GANs. In *Proceedings of the 38th International Conference on Machine Learning*, 2021.

[17] C. Holt. Forecasting seasonals and trends by exponentially weighted moving averages. *ONR Research Memorandum, Carnegie Institute of Technology*, 52, 1957.

[18] P. Winters. Forecasting sales by exponentially weighted moving averages. *Management Science*, 6:324–342, 1960.

[19] E. J. Hannan and J. Rissanen. Recursive Estimation of Mixed Autoregressive-Moving Average Order. *Biometrika*, 69:81–94, 1982.

[20] J. Yoon, D. Jarrett, and M. van der Schaar. Time-series Generative Adversarial Networks. In *Advances in Neural Information Processing Systems 32*, 2019.

[21] Y. Rubanova, R. T. Q. Chen, and D. Duvenaud. Latent Ordinary Differential Equations for Irregularly-Sampled Time Series. In *Advances in Neural Information Processing Systems 32*, pages 5320–5330, 2019.

[22] E. De Brouwer, J. Simm, A. Arany, and Y. Moreau. GRU-ODE-Bayes: Continuous Modeling of Sporadically-Observed Time Series. In *Advances in Neural Information Processing Systems 32*, pages 7379–7390, 2019.

[23] C. Yildiz, M. Heinonen, and H. Lahdesmaki. ODE2VAE: Deep generative second order ODEs with Bayesian neural networks. In *Advances in Neural Information Processing Systems 32*, 2019.

[24] R. Deng, B. Chang, M. Brubaker, G. Mori, and A. Lehrmann. Modeling Continuous Stochastic Processes with Dynamic Normalizing Flows. In *Advances in Neural Information Processing Systems 33*, pages 7805–7815, 2020.

[25] A. Norcliffe, C. Bodnar, B. Day, J. Moss, and P. Liò. Neural ODE Processes. In *International Conference on Learning Representations*, 2021.

[26] C. Herrera, F. Krach, and J. Teichmann. Neural Jump Ordinary Differential Equations: Consistent Continuous-Time Prediction and Filtering. In *International Conference on Learning Representations*, 2021.

[27] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035, 2019.

[28] J. Behrmann, W. Grathwohl, R. T. Q. Chen, D. Duvenaud, and J.-H. Jacobsen. Invertible Residual Networks. In *Proceedings of the 36th International Conference on Machine Learning*, pages 573–582, 2019.

[29] P. Kidger, J. Foster, X. Li, H. Oberhauser, and T. Lyons. Neural SDEs Made Easy: SDEs are Infinite-Dimensional GANs. *OpenReview*, 2020.

[30] Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole. Score-Based Generative Modeling through Stochastic Differential Equations. In *International Conference on Learning Representations*, 2021.

[31] W. Xu, R. T. Q. Chen, X. Li, and D. Duvenaud. Infinitely Deep Bayesian Neural Networks with Stochastic Differential Equations. *arXiv:2102.06559*, 2021.

[32] J. Zhuang, N. C. Dvornek, S. Tatikonda, and J. S. Duncan. MALI: A memory efficient and reverse accurate integrator for Neural ODEs. In *International Conference on Learning Representations*, 2021.

[33] D. Revuz and M. Yor. *Continuous martingales and Brownian motion*, volume 293. Springer Science & Business Media, 2013.

[34] J. K. Salmon, M. A. Moraes, R. O. Dror, and D. E. Shaw. Parallel random numbers: as easy as 1, 2, 3. *Proc. High Performance Computing, Networking, Storage and Analysis*, pages 1–12, 2011.

[35] K. Claessen and M. Pałka. Splittable pseudorandom number generators using cryptographic hashing. *ACM SIGPLAN Notices*, 48:47–58, 2013.

[36] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville. Improved Training of Wasserstein GANs. In *Advances in Neural Information Processing Systems 30*, pages 5767–5777, 2017.

[37] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida. Spectral Normalization for Generative Adversarial Networks. In *International Conference on Learning Representations*, 2018.

[38] R. T. Q. Chen, J. Behrmann, D. Duvenaud, and J.-H. Jacobsen. Residual Flows for Invertible Generative Modeling. In *Advances in Neural Information Processing Systems 32*, 2019.

[39] D. Hendrycks and K. Gimpel. Gaussian Error Linear Units (GELUs). *arXiv:1606.08415*, 2016.

[40] S. Elfwing, E. Uchibe, and K. Doya. Sigmoid-Weighted Linear Units for Neural Network Function Approximation in Reinforcement Learning. *arXiv:1702.03118*, 2017.

[41] P. Ramachandran, B. Zoph, and Q. Le. Swish: a Self-Gated Activation Function. *arXiv:1710.05941*, 2017.

[42] X. Li. `torchsde`, 2019. https://github.com/google-research/torchsde.

[43] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud. Neural Ordinary Differential Equations. In *Advances in Neural Information Processing Systems 31*, 2018.

[44] M. E. Sander, P. Ablin, M. Blondel, and G. Peyré. Momentum Residual Neural Networks. *arXiv:2102.07870*, 2021.

[45] P. Kidger, J. Morrill, J. Foster, and T. Lyons. Neural Controlled Differential Equations for Irregular Time Series. In *Advances in Neural Information Processing Systems 33*, pages 6696–6707, 2020.

[46] J. Morrill, C. Salvi, P. Kidger, J. Foster, and T. Lyons. Neural Rough Differential Equations for Long Time Series. In *Proceedings of the 38th International Conference on Machine Learning*, 2021.

[47] B. Chang, M. Chen, E. Haber, and E. H. Chi. AntisymmetricRNN: A dynamical system view on recurrent neural networks. *International Conference on Learning Representations*, 2019.

[48] T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, and T. Aila. Analyzing and Improving the Image Quality of StyleGAN. In *Proc. CVPR*, 2020.

[49] P. Kidger. `torchtyping`, 2021. https://github.com/patrick-kidger/torchtyping.

[50] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein Generative Adversarial Networks. In *Proceedings of the 34th International Conference on Machine Learning*, pages 214–223, 2017.

[51] M. Giles and P. Glasserman. Smoking adjoints: Fast Monte Carlo greeks. *Risk*, 19:88–92, 2006.

[52] R. S. Hamilton. The inverse function theorem of Nash and Moser. *Bulletin of the American Mathematical Society*, 7(1):65–222, 1982.

[53] P. E. Kloeden and E. Platen. *Numerical Solution of Stochastic Differential Equations*. Springer, 1992.

[54] J. Foster, H. Oberhauser, and T. Lyons. An optimal polynomial approximation of Brownian motion. *SIAM Journal on Numerical Analysis*, 58(3):1393–1421, 2020.

[55] B. Leimkuhler, C. Matthews, and M. V. Tretyakov. On the long-time integration of stochastic gradient systems. *Proceedings of the Royal Society A*, 470(2170), 2014.

[56] L. F. Shampine. Stability of the leapfrog/midpoint method. *Applied Mathematics and Computation*, 208(1):293–298, 2009.

[57] A. Griewank. Achieving logarithmic growth of temporal and spatial complexity in reverse automatic differentiation. *Optimization Methods and Software*, 1(1):35–54, 1992.

[58] J. Gaines and T. Lyons. Variable step size control in the numerical solution of stochastic differential equations. *SIAM Journal on Applied Mathematics*, 57(5):1455–1484, 1997.

[59] A. Rößler. Runge-Kutta Methods for the Strong Approximation of Solutions of Stochastic Differential Equations. *SIAM Journal on Numerical Analysis*, 48(3):922–952, 2010.

[60] A. Dickinson. Optimal Approximation of the Second Iterated Integral of Brownian Motion. *Stochastic Analysis and Applications*, 25(5):1109–1128, 2007.

[61] J. Gaines and T. Lyons. Random Generation of Stochastic Area Integrals. *SIAM Journal on Applied Mathematics*, 54(4):1132–1146, 1994.

[62] A. Davie. KMT theory applied to approximations of SDE. In *Stochastic Analysis and Applications 2014*, pages 185–201. Springer, 2014.

[63] G. Flint and T. Lyons. Pathwise approximation of SDEs by coupling piecewise abelian rough paths. *arXiv:1505.01298*, 2015.

[64] J. Foster. *Numerical approximations for stochastic differential equations*. PhD thesis, University of Oxford, 2020.

[65] M. Wiktorsson. Joint characteristic function and simultaneous simulation of iterated Itô integrals for multiple independent Brownian motions. *Annals of Applied Probability*, 11(2):470–487, 2001.

[66] J. Mrongowius and A. Rößler. On the Approximation and Simulation of Iterated Stochastic Integrals and the Corresponding Lévy Areas in Terms of a Multidimensional Brownian Motion. *arXiv:2101.09542*, 2021.

[67] J. Foster and K. Habermann. Brownian bridge expansions for Lévy area approximations and particular values of the Riemann zeta function. *arXiv:2102.10095*, 2021.

[68] A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf, and A. Smola. A Kernel Two-Sample Test. *Journal of Machine Learning Research*, 13(1):723–773, 2013.

[69] F. Király and H. Oberhauser. Kernels for sequentially ordered data. *Journal of Machine Learning Research*, 20(31):1–45, 2019.

[70] C. Toth and H. Oberhauser. Bayesian Learning from Sequential Data using Gaussian Processes with Signature Covariances. In *Proceedings of the 37th International Conference on Machine Learning*, pages 9548–9560, 2020.

[71] P. Bonnier, P. Kidger, I. Perez-Arribas, C. Salvi, and T. Lyons. Deep Signature Transforms. In *Advances in Neural Information Processing Systems 32*, 2019.

[72] C. Salvi, T. Cass, J. Foster, T. Lyons, and W. Yang. The Signature Kernel is the solution of a Goursat PDE. *arXiv:2006.14794*, 2021.

[73] M. Lemercier, C. Salvi, T. Cass, E. V. Bonilla, T. Damoulas, and T. Lyons. SigGPDE: Scaling Sparse Gaussian Processes on Sequential Data. In *Proceedings of the 38th International Conference on Machine Learning*, 2021.

[74] P. Kidger and T. Lyons. Signatory: differentiable computations of the signature and logsignature transforms, on both CPU and GPU. In *International Conference on Learning Representations*, 2021. `https://github.com/patrick-kidger/signatory`.

[75] J. Morrill, A. Fermanian, P. Kidger, and T. Lyons. A generalised signature method for multivariate time series feature extraction. *arXiv:2006.00873*, 2020.

[76] Y. Li, K. Swersky, and R. Zemel. Generative Moment Matching Networks. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 1718–1727, 2015.

[77] C.-L. Li, W.-C. Chang, Y. Cheng, Y. Yang, and B. Poczos. MMD GAN: Towards Deeper Understanding of Moment Matching Network. In *Advances in Neural Information Processing Systems 30*, pages 2203–2213, 2017.

[78] P. Kidger. `torchcde`, 2020. `https://github.com/patrick-kidger/torchcde`.

[79] R. T. Q. Chen. torchdiffeq, 2018. `https://github.com/rtqichen/torchdiffeq`.

[80] Ax. `https://ax.dev`.

[81] D. Kingma and J. Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

[82] M. D. Zeiler. ADADELTA: An Adaptive Learning Rate Method. *arXiv:1212.5701*, 2012.

[83] Y. Yazıcı, C.-S. Foo, S. Winkler, K.-H. Yap, G. Piliouras, and V. Chandrasekhar. The Unusual Effectiveness of Averaging in GAN training. In *International Conference on Learning Representations*, 2019.

[84] P. Izmailov, D. Podoprikhin, T. Garipov, D. Vetrov, and A. G. Wilson. Averaging Weights Leads to Wider Optima and Better Generalization. *Conference on Uncertainty in Artificial Intelligence*, 2018.

[85] D. Dua and C. Graff. UCI Machine Learning Repository, 2017. URL `http://archive.ics.uci.edu/ml`.

[86] S. Zhang, B. Guo, A. Dong, J. He, Z. Xu, and S. X. Chen. Cautionary tales on air-quality improvement in Beijing. *Proceedings of the Royal Society A*, 473(2205), 2017.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?
   [Yes] The abstract reflects the paper's contributions and scope.
   (b) Did you describe the limitations of your work?
   [Yes] See Section 6.2. We have found every contribution to be a strict improvement on previous work, without introducing any further limitations.
   (c) Did you discuss any potential negative societal impacts of your work?
   [Yes] This is discussed in Section 6.3 and in further detail in Appendix G.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them?
   [Yes] We have read the ethics review guidelines available at `https://nips.cc/public/EthicsGuidelines` and believe our paper conforms to them.
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results?
   [Yes] Full theoretical details are provided in Appendix D.
   (b) Did you include complete proofs of all theoretical results?
   [Yes] Complete proofs of convergence of the SDE solver are provided in Appendix D.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)?
   [Yes] All code is attached as supplementary material.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)?
   [Yes] Full training details (including data splits, hyperparameter optimisation, and so on) are available in Appendix F.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)?
   [Yes] Every experiment is run multiple times, and mean and standard deviations of the test metrics reported.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)?
   [Yes] Compute resources and computed time are specified in Appendix F.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators?
   [Yes] Citations are included in-line.
   (b) Did you mention the license of the assets?
   [Yes] See Appendix G.
   (c) Did you include any new assets either in the supplemental material or as a URL?
   [Yes] Assuming that the contributions of this paper count as assets, then these are made available in the supplementary material.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating?
   [Yes] See Appendix G.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content?
   [Yes] See Appendix G.
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable?
   [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable?
[N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation?
[N/A]

```python
from torch import randn_like
from torch.nn import Module
from torchtyping import TensorType

def euler_maruyama_solve(y0        : TensorType["batch", 1],
                         dt        : TensorType[()],  # scalar
                         num_steps: int,
                         drift     : Module,
                         diffusion: Module
                         ) ->       list[TensorType["batch", 1]]:
    ys = [y0]
    for _ in range(num_steps):
        ys.append(ys[-1] + drift(ys[-1]) * dt
                  + diffusion(ys[-1]) * randn_like(ys[-1]) * dt.sqrt())
    return ys
```

Figure 4: Euler–Maruyama method in PyTorch. Type annotations (Python 3.9+) are included and `torchtyping` [49] used to indicate shapes of tensors.

A RNNs as discretised SDEs

Consider the autonomous one-dimensional Itô SDE

$$\mathrm{d}Y_t = \mu(Y_t)\,\mathrm{d}t + \sigma(Y_t)\,\mathrm{d}W_t,$$

with $Y_t, \mu(Y_t), \sigma(Y_t), W_t \in \mathbb{R}$. Then its numerical Euler–Maruyama discretisation is

$$Y_{n+1} = Y_n + \mu(Y_n)\Delta t + \sigma(Y_n)\Delta W_n,$$

where Δt is some fixed time step and $W_n \sim \mathcal{N}(0, \Delta t)$. This may be implemented in very few lines of PyTorch code – see Figure 4 – and subject to a suitable loss function between distributions, such as the KL divergence [15] or Wasserstein distance [16], simply backpropagated through in the usual way.

In this way we see that a discretised SDE is simply an RNN consuming random noise.

Neural networks as discretised Neural Differential Equations In passing we note that this is a common occurrence: many popular neural network architectures may be interpreted as discretised differential equations.

Residual networks are discretisations of ODEs [43, 44]. RNNs are discretised controlled differential equations [45, 46, 47].

StyleGAN2 and denoising diffusion probabilistic models are both essentially discretised SDEs [30, 48].

Many invertible neural networks resemble discretised differential equations; for example using an explicit Euler method on the forward pass, and recovering the intermediate computations via the implicit Euler method on the backward pass [28, 38].

B Training criteria for Neural SDEs

SDEs as GANs One classical way of fitting (non-neural) SDEs is to pick some prespecified functions of interest $F_1, \ldots, F_N$, and then ask that $\mathbb{E}_Y\left[F_i(Y)\right] \approx \mathbb{E}_{Y_{\text{true}}}\left[F_i(Y_{\text{true}})\right]$ for all i. For example this may be done by optimising

$$\min_\theta \sum_{i=1}^N \left(\mathbb{E}_Y\left[F_i(Y)\right] - \mathbb{E}_{Y_{\text{true}}}\left[F_i(Y_{\text{true}})\right]\right)^2.$$

This ensures that the model and the data behave the same with respect to the functions F_i. (Known as either 'witness functions' or 'payoff functions' depending on the field.)

Kidger et al. [16] generalise this by replacing $F_1, \ldots F_N$ with a parameterised function F_ϕ – a neural network with parameters ϕ – and training adversarially:

$$\min_\theta \max_\phi \left(\mathbb{E}_Y \left[F_\phi(Y) \right] - \mathbb{E}_{Y_{\text{true}}} \left[F_\phi(Y_{\text{true}}) \right] \right). \tag{9}$$

Thus making a connection to the GAN literature.

In principle F_ϕ could be parameterised as any neural network capable of operating on the path-valued Y. There is a natural choice: use a Neural CDE [45]. This is a differential equation capable of acting on path-valued inputs. This means letting the discriminator be $F_\phi(Y) = m_\phi \cdot H_T$, where

$$H_0 = \xi_\phi(Y_0), \qquad \mathrm{d}H_t = f_\phi(t, H_t)\,\mathrm{d}t + g_\phi(t, H_t) \circ \mathrm{d}Y_t,$$

for suitable neural networks ξ_ϕ, f_ϕ, g_ϕ and vector m_ϕ. This is a deterministic function of the generated sample Y. Here $\cdot$ denotes a dot product.

Adding regularisation to control the derivative of F_ϕ ensures that equation (3) corresponds to the dual formulation of the Wasserstein distance, so that Y is capable of perfectly matching Y_{true} given enough data, training time, and model capacity [50].

In our experience, this approach produces models with a very high modelling capacity, but which are somewhat involved to train – GANs being notoriously hard to train stably.

Latent SDEs Li et al. [15] have an alternate approach. Let

$$\xi_\phi \colon \mathbb{R}^y \to \mathbb{R}^x \times \mathbb{R}^x, \qquad \nu_\phi \colon [0,T] \times \mathbb{R}^x \times \{[0,T] \to \mathbb{R}^y\} \to \mathbb{R}^x, \tag{10}$$

be (Lipschitz) neural networks[4] parameterised by ϕ. Let $(m, s) = \xi_\phi(Y_{\text{true},0})$, let $\widehat{V} \sim \mathcal{N}(m, sI_v,)$, and let

$$\widehat{X}_0 = \zeta_\theta(\widehat{V}), \qquad \mathrm{d}\widehat{X}_t = \nu_\phi(t, \widehat{X}_t, Y_{\text{true}})\,\mathrm{d}t + \sigma_\theta(t, \widehat{X}_t) \circ \mathrm{d}W_t, \qquad \widehat{Y}_t = \ell_\theta(\widehat{X}_t).$$

Note that $\widehat{X}$ is a random variable *over SDEs*, as Y_{true} is still a random variable.[5]

They show that the KL divergence

$$\mathrm{KL}\left(\widehat{X} \middle\| X\right) = \mathbb{E}_W \int_0^T \frac{1}{2} \left\| (\sigma_\theta(t, \widehat{X}_t))^{-1} (\mu_\theta(t, \widehat{X}_t) - \nu_\phi(t, \widehat{X}_t, Y_{\text{true}})) \right\|_2^2 \mathrm{d}t,$$

and optimise

$$\min_{\theta,\phi} \mathbb{E}_{Y_{\text{true}}} \left[(\widehat{Y}_0 - Y_{\text{true},0})^2 + \mathrm{KL}\left(\widehat{V} \middle\| V\right) + \mathbb{E}_W \int_0^T (\widehat{Y}_t - Y_{\text{true},t})^2 \,\mathrm{d}t + \mathrm{KL}\left(\widehat{X} \middle\| X\right) \right].$$

This may be derived as an evidence lower-bound (ELBO). The first two terms are simply a VAE for generating Y_0, with latent V. The third term and fourth term are a VAE for generating Y, by autoencoding Y_{true} to $\widehat{Y}$, and then fitting X to $\widehat{X}$.

In our experience, this produces less expressive models than the SDE-GAN approach; however the model is easier to train, due to the lack of adversarial training.[6]

C Adjoints for SDEs

Recall that we wish to backpropagate from our generated sample Y to the parameters θ, ϕ. Here we provide a more complete overview of the options for how this may be done.

[4] We do not discuss the regularity of elements of $\{[0,T] \to \mathbb{R}^y\}$ as in practice we will have discrete observations; ν_ϕ is thus commonly parameterised as $\nu_\phi(t, \widehat{X}_t, Y_{\text{true}}) = \nu_\phi^1(t, \widehat{X}_t, \nu_\phi^2(Y_{\text{true}}|_{[t,T]}))$, where ν_ϕ^1 is for example an MLP, and ν_ϕ^2 is an RNN.

[5] So that $\widehat{V}, \widehat{X}, \widehat{Y}$ might be better denoted $\widehat{V}|Y_{\text{true},0}$, $\widehat{X}|Y_{\text{true}}$, $\widehat{Y}|Y_{\text{true}}$ to reflect their dependence on Y_{true}; we elide this for simplicity of notation.

[6] For ease of presentation this section features a slight abuse of notation: writing the KL divergence between random variables rather than probability distributions. It is also a slight specialisation of Li et al. [15], who allow losses other than the L^2 loss between data and sample.

Discretise-then-optimise One way is to simply backpropagate through the internals of every numerical solver. (Also known as 'discretise-then-optimise'.) However this requires $\mathcal{O}(HT)$ memory, where H denotes the amount of memory used to evaluate and backpropagate through each neural network once.

Optimise-then-discretise The continuous adjoint method,[7] also known as "optimise-then-discretise", instead exploits the reversibility of a differential equation. This means that intermediate computations such as X_t for $t < T$ are reconstructed from output computations, and do not need to be held in memory.

Recall equations (5) and (6): given some Stratonovich SDE

$$\mathrm{d}Z_t = \mu(t, Z_t)\,\mathrm{d}t + \sigma(t, Z_t) \circ \mathrm{d}W_t \quad \text{for } t \in [0, T],$$

and a loss $L\colon \mathbb{R}^z \to \mathbb{R}$ on its terminal value Z_T, then the adjoint process $A_t = {}^{\mathrm{d}L(Z_T)}/_{dZ_t} \in \mathbb{R}^z$ is a (strong) solution to

$$\mathrm{d}A_t^i = -A_t^j \frac{\partial \mu^j}{\partial Z^i}(t, Z_t)\,\mathrm{d}t - A_t^j \frac{\partial \sigma^{j,k}}{\partial Z^i}(t, Z_t) \circ \mathrm{d}W_t^k,$$

which in particular uses the same Brownian motion W as on the forward pass. These may be solved backwards-in-time from $t = T$ to $t = 0$, starting from $Z_T = Z_T$ (computed on the forward pass) and $A_T = {}^{L(Z_T)}/_{dZ_T}$. Then $A_0 = {}^{\mathrm{d}L(Z_T)}/_{Z_0}$ is the desired backpropagated gradient.

The main advantage of the continuous adjoint method is that it reduces the memory footprint to only $\mathcal{O}(H + T)$. $\mathcal{O}(H)$ to compute each vector-Jacobian product (${A^j \partial \mu^i}/{\partial Z^j}$ and ${A^j \partial \sigma^{i,k}}/{\partial Z^j}$), and $\mathcal{O}(T)$ to hold the batch of training data.

The main disadvantage (unless using the reversible Heun method) is that the two numerical approximations to Z_t, computed in the forward and backward passes of equation (5), are different. This means that the Z_t used as an input in equation (6) does not perfectly match what is used in the forward calculation, and the gradients A_0 suffer some error as a result. This slows and worsens training. (Often exacerbating an already tricky training procedure, such as the adversarial training of SDE-GANs.)

Itô versus Stratonovich Note that backpropagation through an Itô SDE may be performed by first adding a $-({\sigma^{j,k} \partial \sigma^{i,k}}/{\partial Z^j})(t, Z_t)/2$ correction term to μ in equation (5), which converts it to a Stratonovich SDE, and then applying equation (6).

This additional computational cost – including double autodifferentiation to compute derivatives of the correction term – means that we prefer to use Stratonovich SDEs throughout.

D Error analysis of the reversible Heun method

D.1 Notation and definitions

In this section, we present some of the notation, definitions and assumptions used in our error analysis.

Throughout, we use $\|\cdot\|_2$ to denote the standard Euclidean norm on $\mathbb{R}^n$ and $\langle \cdot, \cdot \rangle$ will be the associated inner product with $\langle x, x \rangle = \|x\|_2^2$. We use the same notation to denote norms for tensors.

For $k \geq 1$, a k-tensor on $\mathbb{R}^n$ is simply an element of the n^k-dimensional space $\mathbb{R}^{n \times \cdots \times n}$ and can be interpreted as a multilinear map into $\mathbb{R}$ with k arguments from $\mathbb{R}^n$. Hence for a k-tensor T on $\mathbb{R}^n$ (with $k \geq 2$) and a vector $v \in \mathbb{R}^n$, we shall define $Tv := T(v, \cdots)$ as a $(k-1)$-tensor on $\mathbb{R}^n$.

Therefore, we can define the operator norm of T recursively as

$$\|T\|_{\mathrm{op}} := \begin{cases} \|T\|_2, & \text{if } T \in \mathbb{R}^n, \\ \sup\limits_{\substack{v \in \mathbb{R}^n, \\ \|v\|_2 \leq 1}} \|Tv\|_{\mathrm{op}}, & \text{if } T \text{ is a } k\text{-tensor on } \mathbb{R}^n \text{ with } k \geq 2. \end{cases} \tag{11}$$

[7] Frequently abbreviated to simply 'adjoint method' in the modern literature, although this term is ambiguous as it is also used to refer to backpropagation-through-the-solver in other literature [51].

With a slight abuse of notation, we shall write $\|\cdot\|_2$ instead of $\|\cdot\|_{\text{op}}$ for all k-tensors.

In addition, we denote the standard tensor product by $\otimes$. So for a k_1-tensor x and k_2-tensor y on $\mathbb{R}^n$, $x \otimes y$ is a (k_1+k_2)-tensor on $\mathbb{R}^n$. Moreover, it is straightforward to show that $\|x \otimes y\|_2 \le \|x\|_2 \|y\|_2$ and for a k-tensor T on $\mathbb{R}^n$, we have that $Tv := T(v_1, \cdots, v_k)$ for all $v = v_1 \otimes \cdots \otimes v_k \in (\mathbb{R}^n)^{\otimes k}$.

We suppose that $\big(\Omega, \mathcal{F}, \mathbb{P}; \{\mathcal{F}_t\}_{t\ge 0}\big)$ is a filtered probability space carrying a standard d-dimensional Brownian motion. We consider the following Stratonovich SDE over the finite time horizon $[0,T]$,

$$\mathrm{d}y_t = f(t, y_t)\,\mathrm{d}t + g(t, y_t) \circ \mathrm{d}W_t\,, \tag{12}$$

where $y = \{y_t\}_{t\in[0,T]}$ is a continuous $\mathbb{R}^e$-valued stochastic process and $f : [0,T] \times \mathbb{R}^e \to \mathbb{R}^e$, $g : [0,T] \times \mathbb{R}^e \to \mathbb{R}^{e\times d}$ are functions. Without loss of generality, we rewrite (12) as an autonomous SDE by letting $t \mapsto t$ be a coordinate of y . This simplifies notation and results in the following SDE:

$$\mathrm{d}y_t = f(y_t)\,\mathrm{d}t + g(y_t) \circ \mathrm{d}W_t\,, \tag{13}$$

We shall assume that f, g are bounded and twice continuously differentiable with bounded derivatives,

$$\|D^k f\|_\infty := \sup_{x\in\mathbb{R}^e} \big\|D^k f(x)\big\|_2 < \infty, \qquad \|D^k g\|_\infty := \sup_{x\in\mathbb{R}^e} \big\|D^k g(x)\big\|_2 < \infty, \tag{14}$$

for $k = 0, 1, 2$, where $\|\cdot\|_2$ denotes the Euclidean operator norm on k-tensors given by (11).

For $N \ge 1$, we will compute numerical SDE solutions on $[0,T]$ using a constant step size $h = \frac{T}{N}$. That is, numerical solutions are obtained at times $\{t_n\}_{0\le n\le N}$ with $t_n := nh$ for $n \in \{0, 1, \cdots, N\}$.

For $0 \le n \le N$, we denote increments of Brownian motion by $W_n := W_{t_{n+1}} - W_{t_n} \sim \mathcal{N}(0, \mathrm{I}_d h)$, where I_d is the $d \times d$ identity matrix. We use $h_{\max} > 0$ to denote an upper bound for the step size h.

Given a random vector X, taking its values in $\mathbb{R}^n$, we define the $\mathbb{L}_p$ norm of X for $p \ge 1$ as

$$\|X\|_{\mathbb{L}_p} := \mathbb{E}\big[\|X\|_2^p\big]^{\frac{1}{p}}, \tag{15}$$

Similarly, for a stochastic process $\{X_t\}$, the $\mathcal{F}_{t_n}$-conditional $\mathbb{L}_p$ norm of X_t is

$$\|X_t\|_{\mathbb{L}_p^n} := \mathbb{E}_n\big[\|X_t\|_2^p\big]^{\frac{1}{p}} = \mathbb{E}\big[\|X_t\|_2^p \,\big|\, \mathcal{F}_{t_n}\big]^{\frac{1}{p}}, \tag{16}$$

for $t \ge t_n$.

We say that $x(h) = O(h^\gamma)$ if there exists a constant $C > 0$, depending on $h_{\max}$ but not h, such that $|x(h)| \le Ch^\gamma$ for $h \in (0, h_{\max}]$. We also use big-O notation for estimating quantities in $\mathbb{L}_p$ and $\mathbb{L}_p^n$.

We use $\|\mathcal{N}(0, \mathrm{I}_d)\|_{\mathbb{L}_p}$ to denote the $\mathbb{L}_p$ norm of a standard d-dimensional normal random vector. Thus we have $\|\mathcal{N}(0, \mathrm{I}_d)\|_{\mathbb{L}_2} = \sqrt{d}$, $\|\mathcal{N}(0, \mathrm{I}_d)\|_{\mathbb{L}_4} = (d^2 + 2d)^{\frac{1}{4}}$ and $\|W_n\|_{\mathbb{L}_p} = \|\mathcal{N}(0, \mathrm{I}_d)\|_{\mathbb{L}_p} h^{\frac{1}{2}}$.

We recall the reversible Heun method given by Algorithm 1.

Definition D.1 (Reversible Heun method)**.** *For $N \ge 1$, we construct a numerical solution $\{Y_n, Z_n\}_{0\le n\le N}$ for the SDE (13) by setting $Y_0 = Z_0 = y_0$ and, for each $n \in \{0, 1, \cdots, N-1\}$, defining (Y_{n+1}, Z_{n+1}) from (Y_n, Z_n) as*

$$Z_{n+1} := 2Y_n - Z_n + f\big(Z_n\big)h + g\big(Z_n\big)W_n\,, \tag{17}$$

$$Y_{n+1} := Y_n + \frac{1}{2}\big(f\big(Z_n\big) + f\big(Z_{n+1}\big)\big)h + \frac{1}{2}\big(g\big(Z_n\big) + g\big(Z_{n+1}\big)\big)W_n\,, \tag{18}$$

where $W_n := W_{t_{n+1}} - W_{t_n} \sim \mathcal{N}(0, \mathrm{I}_d h)$ is an increment of a d-dimensional Brownian motion W. The collection $\{Y_n\}_{0\le n\le N}$ is the numerical approximation to the true solution.

Remark D.2. Whilst it is not used in our analysis, the above numerical method is time-reversible as

$$Z_n = 2Y_{n+1} - Z_{n+1} - f\big(Z_{n+1}\big)h - g\big(Z_{n+1}\big)W_n\,,$$
$$Y_n = Y_{n+1} - \frac{1}{2}\big(f\big(Z_{n+1}\big) + f\big(Z_n\big)\big)h - \frac{1}{2}\big(g\big(Z_{n+1}\big) + g\big(Z_n\big)\big)W_n\,.$$

To simplify notation, we shall define another numerical solution $\{\widetilde{Z}_n\}_{0\le n\le N}$ with $\widetilde{Z}_n := 2Y_n - Z_n$.

Our analysis is outlined as follows. In Section D.2, we show that when h is sufficiently small, $\|Y_n - Z_n\|_{\mathbb{L}_4} = O\big(\sqrt{h}\big)$. The choice of $\mathbb{L}_4$ instead of $\mathbb{L}_2$ is important in subsequent error estimates. In Section D.3, we show that the reversible Heun method converges to the Stratonovich solution of SDE (13) in the $\mathbb{L}_2$-norm at a rate of $O\big(\sqrt{h}\big)$. This matches the convergence rate of standard SDE solvers such as the Heun and midpoint methods. In Section D.4, we consider the case where g is constant (i.e. additive noise) and show that the $\mathbb{L}_2$ convergence rate of our method becomes $O(h)$. We also give numerical evidence that the the reversible Heun method can achieve second order weak convergence for SDEs with additive noise. In Section D.5, we consider the stability of the reversible Heun method in the ODE setting. We show that it has the same absolute stability region for a linear test equation as the (reversible) asynchronous leapfrog integrator proposed for Neural ODEs in [32].

D.2 Approximation error between components of the reversible Heun method

The key idea underlying our analysis is to consider two steps of the numerical method, which gives,

$$Z_{n+2} := Z_n + 2f\big(Z_{n+1}\big)h + g\big(Z_{n+1}\big)\big(W_n + W_{n+1}\big), \tag{19}$$

$$\begin{aligned} Y_{n+2} := Y_n &+ \frac{1}{2}\big(f\big(Z_n\big) + 2f\big(Z_{n+1}\big) + f\big(Z_{n+2}\big)\big)h \\ &+ \frac{1}{2}\big(g\big(Z_n\big) + g\big(Z_{n+1}\big)\big)W_n + \frac{1}{2}\big(g\big(Z_{n+1}\big) + g\big(Z_{n+2}\big)\big)W_{n+1}\,. \end{aligned} \tag{20}$$

Thus Z is propagated by a midpoint method and Y is propagated by a trapezoidal rule / Heun method. To prove that $\{Z_n\}$ and $\{Y_n\}$ are close together when h is small, we shall use the Taylor expansions:

Theorem D.3 (Taylor expansions of vector fields)**.** *Let F be a bounded and twice continuously differentiable function on $\mathbb{R}^e$ with bounded derivatives (i.e. we can set $F = f$ or g). Then for $n \geq 0$,*

$$\begin{aligned} F\big(Z_{n+1}\big) &= F\big(\widetilde{Z}_n\big) + F'\big(\widetilde{Z}_n\big)\big(Z_{n+1} - \widetilde{Z}_n\big) + R_n^{F,1}, \\ F\big(Z_{n+2}\big) &= F\big(Z_n\big) + F'\big(Z_n\big)\big(Z_{n+2} - Z_n\big) + R_n^{F,2}, \end{aligned}$$

where the remainder terms $R_n^{F,1}$ and $R_n^{F,2}$ satisfy the following estimates for any fixed $p \geq 1$,

$$\begin{aligned} \big\|R_n^{F,1}\big\|_{\mathbb{L}_p^n} &= O\big(h\big), \\ \big\|R_n^{F,2}\big\|_{\mathbb{L}_p^n} &= O\big(h\big). \end{aligned}$$

Proof. By Taylor's theorem with integral remainder [52, Theorem 3.5.6], the $R_n^{F,i}$ terms are given by

$$\begin{aligned} R_n^{F,1} &= \int_0^1 (1-t)F''\big(\widetilde{Z}_n + t\big(Z_{n+1} - \widetilde{Z}_n\big)\big)\,\mathrm{d}t\,\big(Z_{n+1} - \widetilde{Z}_n\big)^{\otimes 2}, \\ R_n^{F,2} &= \int_0^1 (1-t)F''\big(Z_n + t\big(Z_{n+2} - Z_n\big)\big)\,\mathrm{d}t\,\big(Z_{n+2} - Z_n\big)^{\otimes 2}. \end{aligned}$$

We first note that for $\mathbb{R}^e$-valued random vectors X_1 and X_2, we have

$$\begin{aligned} \left\| \int_0^1 (1-t)F''\big(X_1 + tX_2\big)\,\mathrm{d}t\,X_2^{\otimes 2}\right\|_{\mathbb{L}_p^n} &= \mathbb{E}_n\left[\left\| \int_0^1 (1-t)F''\big(X_1 + tX_2\big)\,\mathrm{d}t\,X_2^{\otimes 2}\right\|_2^p\right]^{\frac{1}{p}} \\ &\leq \mathbb{E}_n\left[\left\| \int_0^1 (1-t)^{k-1}F''\big(X_1 + tX_2\big)\,\mathrm{d}t\right\|_2^p \big\|X_2^{\otimes 2}\big\|_2^p\right]^{\frac{1}{p}} \\ &\leq \mathbb{E}_n\left[\int_0^1 \Big\|(1-t)F''\big(X_1 + tX_2\big)\Big\|_2^p \mathrm{d}t\,\big\|X_2\big\|_2^{2p}\right]^{\frac{1}{p}} \\ &\leq \big(p+1\big)^{-\frac{1}{p}}\|F''\|_\infty\big\|X_2\big\|_{\mathbb{L}_{2p}^n}^2, \end{aligned}$$

where the inequality $\big\| \int_0^1 \cdot \,\mathrm{d}t \big\|_2^p \le \int_0^1 \|\cdot\|_2^p \,\mathrm{d}t$ follows by Jensen's inequality and the convexity of $x \mapsto x^p$. Thus, it is enough to estimate $Z_{n+1} - \widetilde{Z}_n$ and $Z_{n+2} - Z_n$ using Minkowski's inequality.

$$\begin{aligned}
\big\|Z_{n+1} - \widetilde{Z}_n\big\|_{\mathbb{L}^n_{2p}}^2 &= \big\|f(Z_n)h + g(Z_n)W_n\big\|_{\mathbb{L}^n_{2p}}^2 \\
&\le 2\big\|f(Z_n)h\big\|_{\mathbb{L}^n_{2p}}^2 + 2\big\|g(Z_n)W_n\big\|_{\mathbb{L}^n_{2p}}^2 \\
&\le 2\|f\|_\infty^2 h^2 + 2\|g\|_\infty^2 \|\mathcal{N}(0,\mathrm{I}_d)\|_{\mathbb{L}_{2p}}^2 h\,, \\
\big\|Z_{n+2} - Z_n\big\|_{\mathbb{L}^n_{2p}}^2 &= \big\|2f\big(Z_{n+1}\big)h + g\big(Z_{n+1}\big)\big(W_n + W_{n+1}\big)\big\|_{\mathbb{L}^n_{2p}}^2 \\
&\le 2\big\|2f\big(Z_{n+1}\big)h\big\|_{\mathbb{L}^n_{2p}}^2 + 2\big\|g\big(Z_{n+1}\big)\big(W_n + W_{n+1}\big)\big\|_{\mathbb{L}^n_{2p}}^2 \\
&\le 4\|f\|_\infty^2 h^2 + 4\|g\|_\infty^2 \|\mathcal{N}(0,\mathrm{I}_d)\|_{\mathbb{L}_{2p}}^2 h,
\end{aligned}$$

where we also used the inequality $(a+b)^2 \le 2a^2 + 2b^2$. The result now follows from the above. □

With Theorem D.3, it is straightforward to derive a Taylor expansion for the difference $Y_{n+2} - Z_{n+2}$.

Theorem D.4. *For $n \in \{0, 1, \cdots, N-2\}$, the difference $Y_{n+2} - Z_{n+2}$ can be expanded as*

$$\begin{aligned}
Y_{n+2} - Z_{n+2} = Y_n - Z_n &+ \Big(f(Z_n) - f\big(\widetilde{Z}_n\big)\Big)h + \frac{1}{2}\Big(g(Z_n) - g\big(\widetilde{Z}_n\big)\Big)\big(W_n + W_{n+\frac{1}{2}}\big) \\
&+ \frac{1}{2}\Big(g'(Z_n)\big(g\big(\widetilde{Z}_n\big)\big(W_n + W_{n+1}\big)\big)\Big)W_{n+1} \\
&- \frac{1}{2}\Big(g'\big(\widetilde{Z}_n\big)\big(g(Z_n)W_n\big)\Big)\big(W_n + W_{n+1}\big) + R_n^Z\,,
\end{aligned} \tag{21}$$

where the remainder term R_n^Z satisfies $\big\|R_n^Z\big\|_{\mathbb{L}_2^n} = O\big(h^{\frac{3}{2}}\big)$.

Proof. Expanding the components (17), (18) of the reversible Heun method with Theorem D.3 gives

$$\begin{aligned}
&\big(Y_{n+2} - Z_{n+2}\big) - \big(Y_n - Z_n\big) \\
&\quad= \frac{1}{2}f(Z_n)h + \frac{1}{2}f(Z_{n+2})h - f\big(Z_{n+1}\big)h \\
&\qquad+ \frac{1}{2}g(Z_n)W_n + \frac{1}{2}g(Z_{n+2})W_{n+1} - \frac{1}{2}g\big(Z_{n+1}\big)\big(W_n + W_{n+1}\big) \\
&\quad= \frac{1}{2}f(Z_n)h + \frac{1}{2}g(Z_n)W_n + \frac{1}{2}\Big(f(Z_n) + f'(Z_n)\big(Z_{n+2} - Z_n\big) + R_n^{f,2}\Big)h \\
&\qquad- \Big(f\big(\widetilde{Z}_n\big) + f'\big(\widetilde{Z}_n\big)\big(Z_{n+1} - \widetilde{Z}_n\big) + R_n^{f,1}\Big)h \\
&\qquad+ \frac{1}{2}\Big(g(Z_n) + g'(Z_n)\big(Z_{n+2} - Z_n\big) + R_n^{g,2}\Big)W_{n+1} \\
&\qquad- \frac{1}{2}\Big(g\big(\widetilde{Z}_n\big) + g'\big(\widetilde{Z}_n\big)\big(Z_{n+1} - \widetilde{Z}_n\big) + R_n^{g,1}\Big)\big(W_n + W_{n+1}\big).
\end{aligned}$$

Since $R_n^{f,1}, R_n^{f,2}, R_n^{g,1}, R_n^{g,2} \sim O(h)$, which was shown in Theorem D.3, the above simplifies to

$$\begin{aligned}
&\big(Y_{n+2} - Z_{n+2}\big) - \big(Y_n - Z_n\big) \\
&\quad= \Big(f(Z_n) - f\big(\widetilde{Z}_n\big)\Big)h + \frac{1}{2}\Big(g(Z_n) - g\big(\widetilde{Z}_n\big)\Big)\big(W_n + W_{n+\frac{1}{2}}\big) \\
&\qquad+ \frac{1}{2}\Big(f'(Z_n)\big(Z_{n+2} - Z_n\big)\Big)h - \Big(f'\big(\widetilde{Z}_n\big)\big(Z_{n+1} - \widetilde{Z}_n\big)\Big)h \\
&\qquad+ \frac{1}{2}\Big(g'(Z_n)\big(Z_{n+2} - Z_n\big)\Big)W_{n+1} - \frac{1}{2}\Big(g'\big(\widetilde{Z}_n\big)\big(Z_{n+1} - \widetilde{Z}_n\big)\Big)\big(W_n + W_{n+1}\big) + O\big(h^{\frac{3}{2}}\big).
\end{aligned}$$

Substituting the formulae (17) and (19) for Z_{n+1} and Z_{n+2} respectively produces

$$\begin{aligned}
&\big(Y_{n+2}-Z_{n+2}\big)-\big(Y_n-Z_n\big)\\
&\quad=\Big(f\big(Z_n\big)-f\big(\widetilde{Z}_n\big)\Big)h+\frac{1}{2}\Big(g\big(Z_n\big)-g\big(\widetilde{Z}_n\big)\Big)\big(W_n+W_{n+\frac{1}{2}}\big)\\
&\qquad+\frac{1}{2}\Big(f'\big(Z_n\big)\big(2f\big(Z_{n+1}\big)h+g\big(Z_{n+1}\big)\big(W_n+W_{n+1}\big)\big)\Big)h\\
&\qquad-\Big(f'\big(\widetilde{Z}_n\big)\big(f\big(Z_n\big)h+g\big(Z_n\big)W_n\big)\Big)h\\
&\qquad+\frac{1}{2}\Big(g'\big(Z_n\big)\big(2f\big(Z_{n+1}\big)h+g\big(Z_{n+1}\big)\big(W_n+W_{n+1}\big)\big)\Big)W_{n+1}\\
&\qquad-\frac{1}{2}\Big(g'\big(\widetilde{Z}_n\big)\big(f\big(Z_n\big)h+g\big(Z_n\big)W_n\big)\Big)\big(W_n+W_{n+1}\big)+O\big(h^{\frac{3}{2}}\big).
\end{aligned}$$

As $f\big(Y_{n+1}\big)$ and $g\big(Y_{n+1}\big)$ are bounded by $\|f\|_\infty$ and $\|g\|_\infty$, collecting the $O\big(h^{\frac{3}{2}}\big)$ terms yields

$$\begin{aligned}
\big(Y_{n+2}-Z_{n+2}\big)-\big(Y_n-Z_n\big)&=\Big(f\big(Z_n\big)-f\big(\widetilde{Z}_n\big)\Big)h+\frac{1}{2}\Big(g\big(Z_n\big)-g\big(\widetilde{Z}_n\big)\Big)\big(W_n+W_{n+\frac{1}{2}}\big)\\
&\quad+\frac{1}{2}\Big(g'\big(Z_n\big)\big(g\big(Z_{n+1}\big)\big(W_n+W_{n+1}\big)\big)\Big)W_{n+1}\\
&\quad-\frac{1}{2}\Big(g'\big(\widetilde{Z}_n\big)\big(g\big(Z_n\big)W_n\big)\Big)\big(W_n+W_{n+1}\big)+O\big(h^{\frac{3}{2}}\big).
\end{aligned}$$

We note that it is direct consequence of Theorem D.3 that $g\big(Z_{n+1}\big)=g\big(\widetilde{Z}_n\big)+O\big(\sqrt{h}\big)$. Thus,

$$\begin{aligned}
\big(Y_{n+2}-Z_{n+2}\big)-\big(Y_n-Z_n\big)&=\Big(f\big(Z_n\big)-f\big(\widetilde{Z}_n\big)\Big)h+\frac{1}{2}\Big(g\big(Z_n\big)-g\big(\widetilde{Z}_n\big)\Big)\big(W_n+W_{n+\frac{1}{2}}\big)\\
&\quad+\frac{1}{2}\Big(g'\big(Z_n\big)\big(g\big(\widetilde{Z}_n\big)\big(W_n+W_{n+1}\big)\big)\Big)W_{n+1}\\
&\quad-\frac{1}{2}\Big(g'\big(\widetilde{Z}_n\big)\big(g\big(Z_n\big)W_n\big)\Big)\big(W_n+W_{n+1}\big)+O\big(h^{\frac{3}{2}}\big),
\end{aligned}$$

which gives the desired result. □

Using the expansion (21), we shall derive estimates for the $\mathbb{L}_4$-norm of the difference $Y_{n+2}-Z_{n+2}$.

Theorem D.5 (Local bound for $Y-Z$)**.** *Let $h_{\max}>0$ be fixed. Then there exist constants $c_1,c_2>0$ such that for $n\in\{0,1,\cdots,N-2\}$,*

$$\big\|Y_{n+2}-Z_{n+2}\big\|_{\mathbb{L}_4^n}^4\le e^{c_1h}\big\|Y_n-Z_n\big\|_2^4+c_2h^3,$$

provided $h\le h_{\max}$.

Proof. To begin, we will expand $\big\|Y_{n+2}-Z_{n+2}\big\|_{\mathbb{L}_4^n}^4$ as

$$\begin{aligned}
&\big\|Y_{n+2}-Z_{n+2}\big\|_{\mathbb{L}_4^n}^4\\
&\quad=\mathbb{E}_n\Big[\Big(\big\|Y_n-Z_n\big\|_2^2+2\big\langle Y_n-Z_n,\big(Y_{n+2}-Z_{n+2}\big)-\big(Y_n-Z_n\big)\big\rangle\\
&\qquad\qquad+\big\|\big(Y_{n+2}-Z_{n+2}\big)-\big(Y_n-Z_n\big)\big\|_2^2\Big)^2\Big]\\
&\quad=\big\|Y_n-Z_n\big\|_2^4+4\,\mathbb{E}_n\Big[\big\|Y_n-Z_n\big\|_2^2\big\langle Y_n-Z_n,\big(Y_{n+2}-Z_{n+2}\big)-\big(Y_n-Z_n\big)\big\rangle\Big]\\
&\qquad+4\,\mathbb{E}_n\Big[\big\langle Y_n-Z_n,\big(Y_{n+2}-Z_{n+2}\big)-\big(Y_n-Z_n\big)\big\rangle^2\Big]\\
&\qquad+4\,\mathbb{E}_n\Big[\big\langle Y_n-Z_n,\big(Y_{n+2}-Z_{n+2}\big)-\big(Y_n-Z_n\big)\big\rangle\big\|\big(Y_{n+2}-Z_{n+2}\big)-\big(Y_n-Z_n\big)\big\|_2^2\Big]\\
&\qquad+2\,\mathbb{E}_n\Big[\big\|Y_n-Z_n\big\|_2^2\big\|\big(Y_{n+2}-Z_{n+2}\big)-\big(Y_n-Z_n\big)\big\|_2^2\Big]\\
&\qquad+\big\|\big(Y_{n+2}-Z_{n+2}\big)-\big(Y_n-Z_n\big)\big\|_{\mathbb{L}_4^n}^4.
\end{aligned}$$

Using the Cauchy-Schwarz inequality along with the fact that $\|\cdot\|_{\mathbb{L}_2^n} \le \|\cdot\|_{\mathbb{L}_4^n}$, we have

$$\begin{aligned}
&\big\|Y_{n+2} - Z_{n+2}\big\|_{\mathbb{L}_4^n}^4 \\
&\quad\le \big\|Y_n - Z_n\big\|_2^4 + 4\big\|Y_n - Z_n\big\|_2^2\Big\langle Y_n - Z_n, \mathbb{E}_n\big[\big(Y_{n+2} - Z_{n+2}\big) - \big(Y_n - Z_n\big)\big]\Big\rangle \\
&\qquad + 6\big\|Y_n - Z_n\big\|_2^2\big\|\big(Y_{n+2} - Z_{n+2}\big) - \big(Y_n - Z_n\big)\big\|_{\mathbb{L}_2^n}^2 \\
&\qquad + 4\big\|Y_n - Z_n\big\|_2\big\|\big(Y_{n+2} - Z_{n+2}\big) - \big(Y_n - Z_n\big)\big\|_{\mathbb{L}_4^n}^3 \\
&\qquad + \big\|\big(Y_{n+2} - Z_{n+2}\big) - \big(Y_n - Z_n\big)\big\|_{\mathbb{L}_4^n}^4.
\end{aligned} \tag{22}$$

By Theorem D.4, the second term can be expanded as

$$\begin{aligned}
&2\,\mathbb{E}_n\Big[\big\langle Y_n - Z_n, \big(Y_{n+2} - Z_{n+2}\big) - \big(Y_n - Z_n\big)\big\rangle\Big] \\
&\quad= 2\Big\langle Y_n - Z_n, \mathbb{E}_n\Big[\big(Y_{n+2} - Z_{n+2}\big) - \big(Y_n - Z_n\big)\Big]\Big\rangle \\
&\quad= 2\Big\langle Y_n - Z_n, \mathbb{E}_n\Big[\Big(f(Z_n) - f(\widetilde{Z}_n)\Big)h + \frac{1}{2}\Big(g(Z_n) - g(\widetilde{Z}_n)\Big)\big(W_n + W_{n+\frac{1}{2}}\big)\Big]\Big\rangle \\
&\qquad + 2\Big\langle Y_n - Z_n, \frac{1}{2}\mathbb{E}_n\Big[\Big(g'(Z_n)\big(g(\widetilde{Z}_n)\big(W_n + W_{n+1}\big)\big)\Big)W_{n+1}\Big]\Big\rangle \\
&\qquad - 2\Big\langle Y_n - Z_n, \frac{1}{2}\mathbb{E}_n\Big[\Big(g'(\widetilde{Z}_n)\big(g(Z_n)W_n\big)\Big)\big(W_n + W_{n+1}\big) + R_n^Z\Big]\Big\rangle \\
&\quad= 2\big\langle Y_n - Z_n, \big(f(Z_n) - f(\widetilde{Z}_n)\big)\big\rangle h + \big\langle Y_n - Z_n, \big(g'(Z_n)g(\widetilde{Z}_n) - g'(Z_n)g(Z_n)\big)\mathrm{I}_d\big\rangle h \\
&\qquad + \big\langle Y_n - Z_n, \big(g'(Z_n)g(Z_n) - g'(\widetilde{Z}_n)g(Z_n)\big)\mathrm{I}_d\big\rangle h + 2\big\langle Y_n - Z_n, \mathbb{E}_n\big[R_n^Z\big]\big\rangle.
\end{aligned}$$

The above can be estimated using the Cauchy-Schartz inequality and Young's inequality as

$$\begin{aligned}
&\Big|2\,\mathbb{E}_n\Big[\big\langle Y_n - Z_n, \big(Y_{n+2} - Z_{n+2}\big) - \big(Y_n - Z_n\big)\big\rangle\Big]\Big| \\
&\quad\le 2\big\|Y_n - Z_n\big\|_2\big\|f(Z_n) - f(\widetilde{Z}_n)\big\|_2 h + \big\|Y_n - Z_n\big\|_2\big\|\big(g'(Z_n)g(\widetilde{Z}_n) - g'(Z_n)g(Z_n)\big)\mathrm{I}_d\big\|_2 h \\
&\qquad + \big\|Y_n - Z_n\big\|_2\big\|\big(g'(Z_n)g(Z_n) - g'(\widetilde{Z}_n)g(Z_n)\big)\mathrm{I}_d\big\|_2 h + 2\big\|Y_n - Z_n\big\|_2\big\|\mathbb{E}_n\big[R_n^Z\big]\big\|_2 \\
&\quad\le 2\|f'\|_\infty\big\|Y_n - Z_n\big\|_2\big\|Z_n - \widetilde{Z}_n\big\|_2 h + \|g'\|_\infty^2\big\|Y_n - Z_n\big\|_2\|Z_n - \widetilde{Z}_n\big\|_2 dh \\
&\qquad + \big\|g''\big\|_\infty\|g\|_\infty\big\|Y_n - Z_n\big\|_2\|Z_n - \widetilde{Z}_n\big\|_2 dh + 2\big\|Y_n - Z_n\big\|_2 h^{\frac{1}{2}}\big\|\mathbb{E}_n\big[R_n^Z\big]\big\|_2 h^{-\frac{1}{2}} \\
&\quad\le \big(4\|f'\|_\infty + 2\|g'\|_\infty^2 d + 2\big\|g''\big\|_\infty\|g\|_\infty d + 1\big)\big\|Y_n - Z_n\big\|_2^2 h + \frac{1}{2}\big\|\mathbb{E}_n\big[R_n^Z\big]\big\|_2^2 h^{-1}.
\end{aligned}$$

By Jensen's inequality and Theorem D.4, we have that $\big\|\mathbb{E}_n\big[R_n^Z\big]\big\|_2^2 \le \big\|R_n^Z\big\|_{\mathbb{L}_2^n}^2 = O(h^3)$. Therefore

$$\Big|\,\mathbb{E}_n\Big[\big\langle Y_n - Z_n, \big(Y_{n+2} - Z_{n+2}\big) - \big(Y_n - Z_n\big)\big\rangle\Big]\Big| \le O(h)\cdot\big\|Y_n - Z_n\big\|_2^2 + O(h^2)\,.$$

The final term in (22) can be estimated using Minkowski's inequality as

$$\begin{aligned}
&\Big\|\big(Y_{n+2} - Z_{n+2}\big) - \big(Y_n - Z_n\big)\Big\|_{\mathbb{L}_p^n} \\
&\quad= \Big\|\Big(f(Z_n) - f(\widetilde{Z}_n)\Big)h + \frac{1}{2}\Big(g(Z_n) - g(\widetilde{Z}_n)\Big)\big(W_n + W_{n+\frac{1}{2}}\big) \\
&\qquad + \frac{1}{2}\Big(g'(Z_n)\big(g(\widetilde{Z}_n)\big(W_n + W_{n+1}\big)\big)\Big)W_{n+1} \\
&\qquad - \frac{1}{2}\Big(g'(\widetilde{Z}_n)\big(g(Z_n)W_n\big)\Big)\big(W_n + W_{n+1}\big) + R_n^Z\Big\|_{\mathbb{L}_p^n} \\
&\quad\le O\big(\sqrt{h}\,\big)\cdot\big\|Y_n - Z_n\big\|_2 + O(h).
\end{aligned}$$

Since $(a+b)^k \le 2^{k-1}(a^k+b^k)$ for $a, b \ge 0$ (which follows from Jensen's inequality), this gives

$$\left\|\big(Y_{n+2}-Z_{n+2}\big)-\big(Y_n-Z_n\big)\right\|_{\mathbb{L}_p^n}^k \le O\big(h^{\frac{1}{2}k}\big)\cdot\big\|Y_n-Z_n\big\|_2^k + O\big(h^k\big).$$

Hence the estimate (22) becomes

$$\begin{aligned}\big\|Y_{n+2}-Z_{n+2}\big\|_{\mathbb{L}_4^n}^4 \le \big\|Y_n-Z_n\big\|_2^4 &+ 4\big\|Y_n-Z_n\big\|_2^2\big(O(h)\cdot\big\|Y_n-Z_n\big\|_2^2+O(h^2)\big)\\ &+ 6\big\|Y_n-Z_n\big\|_2^2\big(O(h)\cdot\big\|Y_n-Z_n\big\|_2^2+O\big(h^2\big)\big)\\ &+ 4\big\|Y_n-Z_n\big\|_2\big(O\big(h^{\frac{3}{2}}\big)\cdot\big\|Y_n-Z_n\big\|_2^3+O\big(h^3\big)\big)\\ &+ O\big(h^2\big)\cdot\big\|Y_n-Z_n\big\|_2^4+O\big(h^4\big).\end{aligned} \tag{23}$$

By Young's inequality, we can further estimate the terms which do not contain $\big\|Y_n-Z_n\big\|_2^4$ as

$$\begin{aligned}\big\|Y_n-Z_n\big\|_2^2\cdot O(h^2) &= \big\|Y_n-Z_n\big\|_2^2 h^{\frac{1}{2}}\cdot O\big(h^{\frac{3}{2}}\big)\\ &\le \frac{1}{2}\big\|Y_n-Z_n\big\|_2^4 h + O\big(h^3\big),\\ \big\|Y_n-Z_n\big\|_2\cdot O(h^3) &= \big\|Y_n-Z_n\big\|_2 h\cdot O\big(h^2\big)\\ &\le \frac{1}{2}\big\|Y_n-Z_n\big\|_2^2 h^2 + O\big(h^4\big)\\ &\le \frac{1}{4}\big\|Y_n-Z_n\big\|_2^4 h + h^3 + O\big(h^4\big).\end{aligned}$$

Finally, by applying the above two estimates to the inequality (23), we arrive at

$$\big\|Y_{n+2}-Z_{n+2}\big\|_{\mathbb{L}_4^n}^4 \le \big(1+O(h)\big)\big\|Y_n-Z_n\big\|_{\mathbb{L}_4^n}^4 + O\big(h^3\big),$$

which gives the desired result. □

We can now prove that Y and Z become close to each other at a rate of $O\big(\sqrt{h}\,\big)$ in the $\mathbb{L}_4$-norm.

Theorem D.6 (Global bound for $Y-Z$)**.** *Let $h_{\max}>0$ be fixed. There exist a constant $C_1>0$ such that for all $n\in\{0,1,\cdots,N\}$,*

$$\big\|Y_n-Z_n\big\|_{\mathbb{L}_4} \le C_1\sqrt{h}\,,$$

provided $h\le h_{\max}$.

Proof. By the tower property of expectations, it follows from Theorem D.5 that

$$\big\|Y_{n+2}-Z_{n+2}\big\|_{\mathbb{L}_4}^4 \le e^{c_1 h}\big\|Y_n-Z_n\big\|_{\mathbb{L}_4}^4 + c_2 h^3,$$

which, along with the fact that $Y_0=Z_0$, implies that

$$\big\|Y_{2n}-Z_{2n}\big\|_{\mathbb{L}_4}^4 \le c_2\sum_{k=0}^{n-1} e^{c_1 kh}h^2 = c_2\,\frac{e^{c_1 nh}-1}{e^{c_1 h}-1}h^2 \le c_2\,\frac{e^{\frac{1}{2}c_1 T}-1}{e^{c_1 h}-1}h^3 = O\big(h^2\big).$$

It is now straightforward to estimate the $\mathbb{L}_4$-norm of $Y_{2n+1}-Z_{2n+1}$ as

$$\begin{aligned}&\big\|Y_{2n+1}-Z_{2n+1}\big\|_{\mathbb{L}_4}\\ &\quad\le \big\|Y_{2n}-Z_{2n}\big\|_{\mathbb{L}_4} + \big\|\big(Y_{2n+1}-Z_{2n+1}\big)-\big(Y_{2n}-Z_{2n}\big)\big\|_{\mathbb{L}_4}\\ &\quad\le 2\big\|Y_{2n}-Z_{2n}\big\|_{\mathbb{L}_4} + \frac{1}{2}\big\|\big(f\big(Z_{2n+1}\big)-f\big(Z_{2n}\big)\big)h + \big(g\big(Z_{2n+1}\big)-g\big(Z_{2n}\big)\big)W_{2n}\big\|_{\mathbb{L}_4}\\ &\quad= O\big(\sqrt{h}\,\big),\end{aligned}$$

which gives the desired result. □

D.3 Strong convergence of the reversible Heun method

To begin, we will Taylor expand the terms in the $Y_n \mapsto Y_{n+1}$ update of the reversible Heun method.

Theorem D.7 (Further Taylor expansions for the vector fields)**.** *Let F be a bounded and twice continuously differentiable function on $\mathbb{R}^e$ with bounded derivatives. Then for $n \geq 0$,*

$$\begin{aligned} F\big(Z_n\big) &= F\big(Y_n\big) + F'\big(Y_n\big)\big(Z_n - Y_n\big) + R_n^{F,3}, \\ F\big(Z_{n+1}\big) &= F\big(Y_n\big) + F'\big(Y_n\big)\big(Y_n - Z_n\big) + F'\big(Y_n\big)\big(g\big(Y_n\big)W_n\big) + R_n^{F,4}, \end{aligned}$$

where the remainder terms $R_n^{F,3}$ and $R_n^{F,4}$ satisfy the following estimates,

$$\begin{aligned} \big\|R_n^{F,3}\big\|_{\mathbb{L}_2} &= O\big(h\big), \\ \big\|R_n^{F,4}\big\|_{\mathbb{L}_2} &= O\big(h\big), \\ \big\|R_n^{F,3}W_n\big\|_{\mathbb{L}_2} &= O\big(h^{\frac{3}{2}}\big), \\ \big\|R_n^{F,4}W_n\big\|_{\mathbb{L}_2} &= O\big(h^{\frac{3}{2}}\big). \end{aligned}$$

Proof. By Taylor's theorem with integral remainder [52, Theorem 3.5.6], we have that for $k \in \{0, 1\}$,

$$\begin{aligned} F\big(Z_{n+k}\big) = F\big(Y_n\big) &+ F'\big(Y_n\big)\big(Z_{n+k} - Y_n\big) \\ &+ \int_0^1 (1-t)F''\big(Y_n + t\big(Z_{n+k} - Y_n\big)\big)\,\mathrm{d}t\,\big(Z_{n+k} - Y_n\big)^{\otimes 2}. \end{aligned}$$

By the same argument used in the proof of Theorem D.3, we can estimate the remainder term as

$$\left\| \int_0^1 (1-t)F''\big(Y_n + t\big(Z_{n+k} - Y_n\big)\big)\,\mathrm{d}t\,\big(Z_{n+k} - Y_n\big)^{\otimes 2} \right\|_{\mathbb{L}_2^n}^2 \leq \frac{1}{3}\|F''\|_\infty^2 \big\|Z_{n+k} - Y_n\big\|_{\mathbb{L}_4^n}^4,$$

Using the inequality $(a+b)^4 \leq (2a^2 + 2b^2)^2 \leq 8a^4 + 8b^4$, we have

$$\begin{aligned} \big\|Z_n - Y_n\big\|_{\mathbb{L}_4^n}^4 &= \big\|Y_n - Z_n\big\|_2^4, \\ \big\|Z_{n+1} - Y_n\big\|_{\mathbb{L}_4^n}^4 &= \big\|Y_n - Z_n + f\big(Z_n\big)h + g\big(Z_n\big)W_n\big\|_{\mathbb{L}_4^n}^4 \\ &\leq 8\big\|Y_n - Z_n\big\|_2^4 + 8\big\|f\big(Z_n\big)h + g\big(Z_n\big)W_n\big\|_{\mathbb{L}_4^n}^4 \\ &\leq 8\big\|Y_n - Z_n\big\|_2^4 + 64\big\|f\big(Z_n\big)h\big\|_{\mathbb{L}_4^n}^4 + 64\big\|g\big(Z_n\big)W_n\big\|_{\mathbb{L}_4^n}^2 \\ &\leq 8\big\|Y_n - Z_n\big\|_2^4 + 64\|f\|_\infty^4 h^4 + 64\|g\|_\infty^4 \|\mathcal{N}(0, \mathrm{I}_d)\|_{\mathbb{L}_4}^4 h^2. \end{aligned}$$

Therefore, by the tower property of expectations, for $k \in \{0, 1\}$,

$$\begin{aligned} \left\| \int_0^1 (1-t)F''\big(Y_n + t\big(Z_{n+k} - Y_n\big)\big)\,\mathrm{d}t\,\big(Z_{n+k} - Y_n\big)^{\otimes 2} \right\|_{\mathbb{L}_2}^2 &\leq \frac{1}{3}\|F''\|_\infty^2 \mathbb{E}\Big[\big\|Z_{n+k} - Y_n\big\|_{\mathbb{L}_4^n}^4\Big] \\ &= O\Big(\big\|Y_n - Z_n\big\|_{\mathbb{L}_4}^4 + h^2\Big) \\ &= O\big(h^2\big), \end{aligned}$$

by the $O\big(\sqrt{h}\,\big)$ global bound on $\|Y_n - Z_n\|_{\mathbb{L}_4}$ in Theorem D.6.

Similarly, for $k \in \{0, 1\}$,

$$\begin{aligned}
&\left\| \int_0^1 (1-t)\, F''\big(Y_n + t\big(Z_{n+k} - Y_n\big)\big)\, \mathrm{d}t\, \big(Z_{n+k} - Y_n\big)^{\otimes 2}\, W_n \right\|_{\mathbb{L}_2}^2 \\
&\quad = \mathbb{E}\left[\left\| \int_0^1 (1-t)\, F''\big(Y_n + t\big(Z_{n+k} - Y_n\big)\big)\, \mathrm{d}t\, \big(Z_{n+k} - Y_n\big)^{\otimes 2}\, W_n \right\|_{\mathbb{L}_2^n}^2 \right] \\
&\quad \le \mathbb{E}\left[\left\| \int_0^1 (1-t)\, F''\big(Y_n + t\big(Z_{n+k} - Y_n\big)\big)\, \mathrm{d}t \right\|_2^2 \left\| \big(Z_{n+k} - Y_n\big)^{\otimes 2} \right\|_2^2 \big\|W_n\big\|_2^2 \right] \\
&\quad \le \mathbb{E}\left[\int_0^1 \left\| (1-t)\, F''\big(Y_n + t\big(Z_{n+k} - Y_n\big)\big) \right\|_2^2 \mathrm{d}t\, \big\|Z_{n+k} - Y_n\big\|_2^4 \big\|W_n\big\|_2^2 \right] \\
&\quad \le \frac{1}{3} \|F''\|_\infty^2 \mathbb{E}\Big[\big\|Z_{n+k} - Y_n\big\|_2^4 \big\|W_n\big\|_2^2 \Big].
\end{aligned}$$

We consider the $k = 0$ and $k = 1$ cases separately. If $k = 0$, then by the tower property, we have

$$\begin{aligned}
&\left\| \int_0^1 (1-t) F''\big(Y_n + t\big(Z_n - Y_n\big)\big)\, \mathrm{d}t\, \big(Z_n - Y_n\big)^{\otimes 2}\, W_n \right\|_{\mathbb{L}_2}^2 \\
&\quad \le \frac{1}{3} \|F''\|_\infty^2\, \mathbb{E}\Big[\mathbb{E}_n\Big[\big\|Z_n - Y_n\big\|_2^4 \big\|W_n\big\|_2^2 \Big] \Big] \\
&\quad = \frac{1}{3} \|F''\|_\infty^2\, \mathbb{E}\Big[\big\|Z_n - Y_n\big\|_2^4\, \mathbb{E}_n\Big[\big\|W_n\big\|_2^2 \Big] \Big] \\
&\quad = O\big(h^3\big).
\end{aligned}$$

Slightly more care should be taken when $k = 1$ since $Z_{n+1} - Y_n$ is not independent of W_n.

$$\begin{aligned}
&\left\| \int_0^1 (1-t)\, F''\big(Y_n + t\big(Z_{n+1} - Y_n\big)\big)\, \mathrm{d}t\, \big(Z_{n+1} - Y_n\big)^{\otimes 2}\, W_n \right\|_{\mathbb{L}_2}^2 \\
&\quad \le \frac{1}{3} \|F''\|_\infty^2\, \mathbb{E}\Big[\mathbb{E}_n\Big[\big\|Y_n - Z_n + f\big(Z_n\big)h + g\big(Z_n\big)W_n\big\|_2^4 \big\|W_n\big\|_2^2 \Big] \Big] \\
&\quad \le \frac{1}{3} \|F''\|_\infty^2\, \mathbb{E}\Big[\mathbb{E}_n\Big[8\big\|Y_n - Z_n\big\|_2^4 \big\|W_n\big\|_2^2 + 8\big\|f\big(Z_n\big)h + g\big(Z_n\big)W_n\big\|_2^4 \big\|W_n\big\|_2^2 \Big] \Big] \\
&\quad \le \frac{8}{3} \|F''\|_\infty^2\, \mathbb{E}\Big[\big\|Y_n - Z_n\big\|_2^4\, \mathbb{E}_n\Big[\big\|W_n\big\|_2^2 \Big] + \mathbb{E}_n\Big[\big\|f\big(Z_n\big)h + g\big(Z_n\big)W_n\big\|_2^4 \big\|W_n\big\|_2^2 \Big] \Big] \\
&\quad = O\big(h^3\big).
\end{aligned}$$

Finally, by Theorem D.6 and the Lipschitz continuity of g, $F'\big(Y_n\big)\big(Z_{n+1} - Y_n\big)$ can be expanded as

$$\begin{aligned}
&F'\big(Y_n\big)\big(Z_{n+1} - Y_n\big) \\
&\quad = F'\big(Y_n\big)\big(Y_n - Z_n\big) + F'\big(Y_n\big)\big(g\big(Y_n\big)W_n\big) + F'\big(Y_n\big)\big(f\big(Z_n\big)h + \big(g\big(Y_n\big) - g\big(Z_n\big)\big)W_n\big) \\
&\quad = F'\big(Y_n\big)\big(Y_n - Z_n\big) + F'\big(Y_n\big)\big(g\big(Y_n\big)W_n\big) + O(h).
\end{aligned}$$

The result follows from the above estimates. □

We are now in a position to compute the local Taylor expansion of the numerical approximation Y.

Theorem D.8 (Taylor expansion of the reversible Heun method)**.** *For $n \in \{0, 1, \cdots, N-1\}$,*

$$Y_{n+1} = Y_n + f\big(Y_n\big)h + g\big(Y_n\big)W_n + \frac{1}{2} g'\big(Y_n\big)\big(g\big(Y_n\big)W_n\big)W_n + R_n^Y,$$

where the remainder term satisfies $\|R_n^Y\|_{\mathbb{L}_2} = O\big(h^{\frac{3}{2}}\big)$.

Proof. By Theorem D.7, we can expand Y_{n+1} as

$$\begin{aligned}
Y_{n+1} &= Y_n + \frac{1}{2}\big(f(Z_n) + f(Z_{n+1})\big)h + \frac{1}{2}\big(g(Z_n) + g(Z_{n+1})\big)W_n \\
&= Y_n + \frac{1}{2}\big(f(Y_n) + f'(Y_n)(Z_n - Y_n) + R_n^{f,3}\big)h \\
&\quad + \frac{1}{2}\big(f(Y_n) + f'(Y_n)(Y_n - Z_n) + f'(Y_n)\big(g(Y_n)W_n\big) + R_n^{f,4}\big)h \\
&\quad + \frac{1}{2}\big(g(Y_n) + g'(Y_n)(Z_n - Y_n) + R_n^{g,3}\big)W_n \\
&\quad + \frac{1}{2}\big(g(Y_n) + g'(Y_n)(Y_n - Z_n) + g'(Y_n)\big(g(Y_n)W_n\big) + R_n^{g,4}\big)W_n \\
&= Y_n + f(Y_n)h + g(Y_n)W_n + \frac{1}{2}g'(Y_n)\big(g(Y_n)W_n\big)W_n \\
&\quad + \frac{1}{2}f'(Y_n)\big(g(Y_n)W_n\big)h + \frac{1}{2}\big(R_n^{f,3} + R_n^{f,4}\big)h + \frac{1}{2}\big(R_n^{g,3} + R_n^{g,4}\big)W_n\,.
\end{aligned}$$

The result now follows as the bottom line is clearly $O\big(h^{\frac{3}{2}}\big)$ by Theorem D.7. □

Just as in the numerical analysis of ODE solvers, we also compute Taylor expansions for the solution. In our setting, we consider the following stochastic Taylor expansion for the Stratonovich SDE (13).

Theorem D.9 (Stratonovich–Taylor expansion, [53, Proposition 5.10.1])**.** *For $n \in \{0, 1, \cdots, N-1\}$,*

$$y_{(n+1)h} = y_{nh} + f(y_{nh})h + g(y_{nh})W_n + g'(y_{nh})g(y_{nh})\mathbb{W}_n + R_n^y\,,$$

where $\mathbb{W}_n$ denotes the second iterated integral of Brownian motion, that is the $d \times d$ matrix given by

$$\mathbb{W}_n := \int_{nh}^{(n+1)h} (W_t - W_{nh}) \otimes \circ\, \mathrm{d}W_t\,,$$

and the remainder term satisfies $\|R_n^y\|_{\mathbb{L}_2} = O\big(h^{\frac{3}{2}}\big)$.

Using the above theorems, we can now obtain a Taylor expansion for the difference $Y_{n+1} - y_{(n+1)h}$.

Theorem D.10. *For $n \in \{0, 1, \cdots, N-1\}$, , the difference $Y_{n+2} - Z_{n+2}$ can be expanded as*

$$\begin{aligned}
Y_{n+1} - y_{(n+1)h} &= Y_n - y_{nh} + \big(f(Y_n) - f(y_{nh})\big)h + \big(g(Y_n) - g(y_{nh})\big)W_n \\
&\quad + \frac{1}{2}\Big(g'(Y_n)g(Y_n) - g'(y_{nh})g(y_{nh})\Big)W_n^{\otimes 2} \\
&\quad - g'(y_{nh})g(y_{nh})\Big(\mathbb{W}_n - \frac{1}{2}W_n^{\otimes 2}\Big) + R_n\,,
\end{aligned}$$

where the remainder term satisfies $\|R_n\|_{\mathbb{L}_2} = O\big(h^{\frac{3}{2}}\big)$.

Proof. Expanding $Y_{n+1} - y_{(n+1)h}$ using Theorem D.8 and Theorem D.9 gives

$$\begin{aligned}
Y_{n+1} - y_{(n+1)h} &= Y_{n+1} - y_{nh} - f(y_{nh})h - g(y_{nh})W_n - g'(y_{nh})g(y_{nh})\mathbb{W}_n - R_n^y \\
&= Y_n - y_{nh} + f(Y_n)h + g(Y_n)W_n + \frac{1}{2}\,g'(Y_n)\big(g(Y_n)W_n\big)W_n \\
&\quad - f(y_{nh})h - g(y_{nh})W_n - g'(y_{nh})g(y_{nh})\mathbb{W}_n + R_n^Y - R_n^y\,,
\end{aligned}$$

where the remainder term R_n satisfies $\|R_n\|_{\mathbb{L}_2} \le \|R_n^Y\|_{\mathbb{L}_2} + \|R_n^y\|_{\mathbb{L}_2} = O\big(h^{\frac{3}{2}}\big)$. □

Having derived Taylor expansions for the approximation and solution processes, we will establish the main results of the section (namely, local and global error estimates for the reversible Heun method).

Theorem D.11 (Local error estimate for the reversible Heun method)**.** *Let $h_{\max} > 0$ be fixed. Then there exist constants $c_3, c_4 > 0$ such that for all $n \in \{0, 1, \cdots, N-1\}$,*

$$\big\|Y_{n+1} - y_{(n+1)h}\big\|_{\mathbb{L}_2}^2 \le e^{c_3 h}\big\|Y_n - y_{nh}\big\|_{\mathbb{L}_2}^2 + c_4 h^2, \tag{24}$$

provided $h \le h_{\max}$.

Proof. Expanding the left hand side of (24) and applying the tower property of expectations yields

$$\begin{aligned}\big\|Y_{n+1} - y_{(n+1)h}\big\|_{\mathbb{L}_2}^2 &= \big\|Y_n - y_{nh}\big\|_{\mathbb{L}_2}^2 + 2\,\mathbb{E}\Big[\big\langle Y_n - y_{nh}\,, \big(Y_{n+1} - y_{(n+1)h}\big) - \big(Y_n - y_{nh}\big)\big\rangle\Big] \\ &\quad + \big\|\big(Y_{n+1} - y_{(n+1)h}\big) - \big(Y_n - y_{nh}\big)\big\|_{\mathbb{L}_2}^2 \\ &= \big\|Y_n - y_{nh}\big\|_{\mathbb{L}_2}^2 + 2\,\mathbb{E}\Big[\big\langle Y_n - y_{nh}\,, \mathbb{E}_n\big[\big(Y_{n+1} - y_{(n+1)h}\big) - \big(Y_n - y_{nh}\big)\big]\big\rangle\Big] \\ &\quad + \big\|\big(Y_{n+1} - y_{(n+1)h}\big) - \big(Y_n - y_{nh}\big)\big\|_{\mathbb{L}_2}^2.\end{aligned}$$

A simple application of the Cauchy-Schwarz inequality then gives

$$\begin{aligned}\big\|Y_{n+1} - y_{(n+1)h}\big\|_{\mathbb{L}_2}^2 \le \big\|Y_n - y_{nh}\big\|_{\mathbb{L}_2}^2 &+ \big\|\big(Y_{n+1} - y_{(n+1)h}\big) - \big(Y_n - y_{nh}\big)\big\|_{\mathbb{L}_2}^2 \\ &+ 2\,\mathbb{E}\Big[\big\|Y_n - y_{nh}\big\|_2 \big\|\mathbb{E}_n\big[\big(Y_{n+1} - y_{(n+1)h}\big) - \big(Y_n - y_{nh}\big)\big]\big\|_2\Big].\end{aligned} \tag{25}$$

To further estimate the above, we note that $\mathbb{W}_n$ and $\frac{1}{2}W_n^{\otimes 2}$ have the same expectation as

$$\begin{aligned}\mathbb{E}_n\big[\mathbb{W}_n\big] &= \mathbb{E}_n\bigg[\int_{nh}^{(n+1)h} \big(W_t - W_{nh}\big) \otimes \circ\, \mathrm{d}W_t\bigg] \\ &= \mathbb{E}_n\bigg[\int_{nh}^{(n+1)h} \big(W_t - W_{nh}\big) \otimes \mathrm{d}W_t + \frac{1}{2}\mathrm{I}_d h\bigg] \\ &= \frac{1}{2}\mathrm{I}_d h,\end{aligned}$$

where the second lines follws by the Itô–Stratonovich correction.

This gives the required $O(h)$ cancellation when we expand the final term in (25) using Theorem D.10.

$$\begin{aligned}&\big\|\mathbb{E}_n\big[\big(Y_{n+1} - y_{(n+1)h}\big) - \big(Y_n - y_{nh}\big)\big]\big\|_2 \\ &\quad= \Big\|\big(f\big(Y_n\big) - f\big(y_{nh}\big)\big)h + \frac{1}{2}\Big(g'\big(Y_n\big)g\big(Y_n\big) - g'\big(y_{nh}\big)g\big(y_{nh}\big)\Big)\mathrm{I}_d h + \mathbb{E}_n\big[R_n\big]\Big\|_2 \\ &\quad\le \|f'\|_\infty \|Y_n - y_{nh}\|_2\, h + \frac{1}{2}\big(\|g'\|_\infty^2 + \|g''\|_\infty \|g\|_\infty\big)\|Y_n - y_{nh}\|_2\, dh + \big\|\mathbb{E}_n\big[R_n\big]\big\|_2.\end{aligned}$$

Similar to the proof of Theorem D.5, we use Young's inequality to estimate remainder terms.

$$\begin{aligned}&\mathbb{E}\Big[\big\|Y_n - y_{nh}\big\|_2 \big\|\mathbb{E}_n\big[\big(Y_{n+1} - y_{(n+1)h}\big) - \big(Y_n - y_{nh}\big)\big]\big\|_2\Big] \\ &\quad\le \big\|Y_n - y_{nh}\big\|_{\mathbb{L}_2}^2 \cdot O(h) + \mathbb{E}\Big[\big\|Y_n - y_{nh}\big\|_2 h^{\frac{1}{2}} \big\|\mathbb{E}_n\big[R_n\big]\big\|_2 h^{-\frac{1}{2}}\Big] \\ &\quad\le \big\|Y_n - y_{nh}\big\|_{\mathbb{L}_2}^2 \cdot O(h) + \frac{1}{2}\,\mathbb{E}\Big[\big\|Y_n - y_{nh}\big\|_2^2\, h + \big\|\mathbb{E}_n\big[R_n\big]\big\|_2^2\, h^{-1}\Big] \\ &\quad\le \big\|Y_n - y_{nh}\big\|_{\mathbb{L}_2}^2 \cdot O(h) + O\big(h^2\big),\end{aligned}$$

where the final line is a consequence of Jensen's inequality as $\mathbb{E}\big[\|\mathbb{E}_n[R_n]\|_2^2\big] \le \|R_n\|_{\mathbb{L}_2}^2 = O\big(h^3\big)$.

It is straightforward to estimate the second term in (25) using Minkowski's inequality as

$$\begin{aligned}\big\|\big(Y_{n+1} - y_{(n+1)h}\big) - \big(Y_n - y_{nh}\big)\big\|_{\mathbb{L}_2^n} &= \Big\|\big(f\big(Y_n\big) - f\big(y_{nh}\big)\big)h + \big(g\big(Y_n\big) - g\big(y_{nh}\big)\big)W_n \\ &\quad + \frac{1}{2}\Big(g'\big(Y_n\big)g\big(Y_n\big) - g'\big(y_{nh}\big)g\big(y_{nh}\big)\Big)W_n^{\otimes 2} \\ &\quad - g'\big(y_{nh}\big)g\big(y_{nh}\big)\Big(\mathbb{W}_n - \frac{1}{2}W_n^{\otimes 2}\Big) + R_n\Big\|_{\mathbb{L}_2^n} \\ &\le \|f'\|_\infty \big\|Y_n - y_{nh}\big\|_2 h + \|g'\|_\infty \big\|Y_n - y_{nh}\big\|_2 \sqrt{d}\, h^{\frac{1}{2}} \\ &\quad + \frac{1}{2}\big(\|g'\|_\infty^2 + \|g''\|_\infty \|g\|_\infty\big)\big\|Y_n - y_{nh}\big\|_2 \|\mathcal{N}(0, \mathrm{I}_d)\|_{\mathbb{L}_4}^2 h \\ &\quad + \|g'\|_\infty \|g\|_\infty \Big\|\mathbb{W}_n - \frac{1}{2}W_n^{\otimes 2}\Big\|_{\mathbb{L}_2^n} + \|R_n\|_{\mathbb{L}_2^n} \\ &\le \big\|Y_n - y_{nh}\big\|_2 \cdot O\big(\sqrt{h}\big) + \big\|Y_n - Z_n\big\|_2 \cdot O\big(\sqrt{h}\big) + O(h).\end{aligned}$$

Therefore, by the $\mathbb{L}_4$-bound on $Y_n - Z_n$ given by Theorem D.6 and Young's inequality, we have

$$\big\|\big(Y_{n+1} - y_{(n+1)h}\big) - \big(Y_n - y_{nh}\big)\big\|_{\mathbb{L}_2}^2 \leq \big\|Y_n - y_{nh}\big\|_{\mathbb{L}_2}^2 \cdot O(h) + O\big(h^2\big).$$

Putting this all together, the inequality (25) becomes

$$\big\|Y_{n+1} - y_{(n+1)h}\big\|_{\mathbb{L}_2}^2 \leq \big(1 + O(h)\big)\big\|Y_n - y_{nh}\big\|_{\mathbb{L}_2}^2 + O\big(h^2\big),$$

and the result follows. □

Just as before, we can immediately obtain a global error estimate by chaining together local estimates.

Theorem D.12 (Global error estimate for the reversible Heun method)**.** *Let $h_{\max} > 0$ be fixed. Then there exists a constant $C_2 > 0$ such that for all $n \in \{0, 1, \cdots, N\}$,*

$$\big\|Y_n - y_{nh}\big\|_{\mathbb{L}_2} \leq C_2\sqrt{h}\,,$$

provided $h \leq h_{\max}$.

Proof. Since $Y_0 = Z_0$, it follows from Theorem D.11 that

$$\big\|Y_n - y_{nh}\big\|_{\mathbb{L}_2}^2 \leq c_4 \sum_{k=0}^{n-1} e^{c_3 kh} h^2 = c_4\,\frac{e^{c_3 nh} - 1}{e^{c_3 h} - 1} h^2 \leq c_4\,\frac{e^{c_3 T} - 1}{e^{c_3 h} - 1} h^2 = O(h),$$

which gives the desired result. □

D.4 The reversible Heun method in the additive noise setting

We now replace the vector field g in the Stratonovich SDE (13) with a fixed matrix $\sigma \in \mathbb{R}^{e \times d}$ to give

$$\mathrm{d}y_t = f(y_t)\,\mathrm{d}t + \sigma\,\mathrm{d}W_t\,. \tag{26}$$

Unsurprisingly, this simplifies the analysis and gives an $O(h)$ strong convergence rate for the method.

Theorem D.13 (Taylor expansion of Y when the SDE's noise is additive)**.** *For $n \in \{0, 1 \cdots, N-1\}$,*

$$Y_{n+1} = Y_n + f\big(Y_n\big)h + \sigma W_n + \frac{1}{2} f'\big(Y_n\big)\big(\sigma W_n\big)h + \widetilde{R}_n^Y,$$

where the remainder term satisfies $\|\widetilde{R}_n^Y\|_{\mathbb{L}_2} = O(h^2)$.

Proof. Expanding $f\big(Z_n\big)$ and $f\big(Z_{n+1}\big)$ at Y_n using Taylor's theorem [52, Theorem 3.5.6] yields

$$\begin{aligned} Y_{n+1} &= Y_n + \frac{1}{2}\big(f\big(Z_n\big) + f\big(Z_{n+1}\big)\big)h + \sigma W_n \\ &= Y_n + f\big(Y_n\big)h + \sigma W_n + \frac{1}{2} f'\big(Y_n\big)\big(\sigma W_n\big)h + \widetilde{R}_n^Y\,, \end{aligned}$$

where $\widetilde{R}_n^Y$ is given by

$$\begin{aligned} \widetilde{R}_n^Y := \frac{1}{2} f'\big(Y_n\big) f\big(Z_n\big)h^2 &+ \frac{1}{2}\int_0^1 (1-t) f''\big(Y_n + t\big(Z_n - Y_n\big)\big)\,\mathrm{d}t\,\big(Z_n - Y_n\big)^{\otimes 2} h \\ &+ \frac{1}{2}\int_0^1 (1-t) f''\big(Y_n + t\big(Z_{n+1} - Y_n\big)\big)\,\mathrm{d}t\,\big(Z_{n+1} - Y_n\big)^{\otimes 2} h\,. \end{aligned}$$

Similar to the proofs of Theorems D.3 and D.7, we can estimate this remainder term as

$$\begin{aligned} \|\widetilde{R}_n^Y\|_{\mathbb{L}_2}^2 &\leq 2\Big\|\frac{1}{2} f'\big(Y_n\big) f\big(Z_n\big)h^2\Big\|_{\mathbb{L}_2}^2 + 2\Big\|\widetilde{R}_n - \frac{1}{2} f'\big(Y_n\big) f\big(Z_n\big)h^2\Big\|_{\mathbb{L}_2}^2 \\ &\leq \frac{1}{2}\|f'\|_\infty^2 \|f\|_\infty^2 h^4 + \mathbb{E}\left[\left\|\int_0^1 (1-t) f''\big(Y_n + t\big(Z_n - Y_n\big)\big)\,\mathrm{d}t\,\big(Z_n - Y_n\big)^{\otimes 2}\right\|_{\mathbb{L}_2^n}^2\right] h^2 \\ &\quad + \mathbb{E}\left[\left\|\int_0^1 (1-t) f''\big(Y_n + t\big(Z_{n+1} - Y_n\big)\big)\,\mathrm{d}t\,\big(Z_{n+1} - Y_n\big)^{\otimes 2}\right\|_{\mathbb{L}_2^n}^2\right] h^2 \\ &\leq \frac{1}{2}\|f'\|_\infty^2 \|f\|_\infty^2 h^4 + \mathbb{E}\left[\frac{1}{3}\|f''\|_\infty^2 \big\|Z_n - Y_n\big\|_2^4\right] h^2 + \mathbb{E}\left[\frac{1}{3}\|f''\|_\infty^2 \big\|Z_{n+1} - Y_n\big\|_{\mathbb{L}_4^n}^4\right] h^2 \\ &\leq \frac{1}{2}\|f'\|_\infty^2 \|f\|_\infty^2 h^4 + 3\|f''\|_\infty^2 \big\|Z_n - Y_n\big\|_{\mathbb{L}_2}^4 h^2 + \frac{8}{3}\|f''\|_\infty^2 \mathbb{E}\Big[\big\|f\big(Z_n\big)h + \sigma W_n\big\|_{\mathbb{L}_4^n}^4\Big] h^2. \end{aligned}$$

The result now follows by the $\mathbb{L}_4$-bound on $Y_n - Z_n$ given by Theorem D.6. □

Likewise, the additive noise SDE (26) admits a simpler Taylor expansion than the general SDE (13).

Theorem D.14 (Stochastic Taylor expansion for additive noise SDEs)**.** *For $n \in \{0, 1, \cdots, N-1\}$,*

$$y_{(n+1)h} = y_{nh} + f\big(y_{nh}\big)h + \sigma W_n + f'\big(y_{nh}\big)\sigma J_n + \widetilde{R}_n^y\,,$$

where J_n denotes the time integral of Brownian motion over the interval $[nh, (n+1)h]$, that is

$$J_n := \int_{nh}^{(n+1)h} \big(W_t - W_{nh}\big)\,\mathrm{d}t\,,$$

and the remainder term satisfies $\|\widetilde{R}_n^y\|_{\mathbb{L}_2} = O\big(h^2\big)$.

Proof. As σ is constant, the terms involving second and third iterated integrals of W do not appear. Therefore the result follows from more general expansions, such as [53, Proposition 5.10.1]. □

To simplify the error analysis, we note the following lemma.

Lemma D.15. *For each $n \in \{0, 1, \cdots, N-1\}$, we define the random vector $H_n := \frac{1}{h}J_n - \frac{1}{2}W_n$. Then H_n is independent of W_n and $H_n \sim \mathcal{N}\big(0, \frac{1}{12}\mathrm{I}_d h\big)$.*

Proof. For the $d = 1$ case, the lemma was shown in [54, Definition 3.5]. When $d > 1$, the result is still straightforward as each coordinate of W is an independent one-dimensional Brownian motion. □

Using the same arguments as before, we can obtain error estimates for reversible Heun method.

Theorem D.16 (Local error estimate for the reversible Heun method in the additive noise setting)**.** *Let $h_{\max} > 0$ be fixed. Then there exist constants $c_5, c_6 > 0$ such that for $n \in \{0, 1, \cdots, N-1\}$,*

$$\big\|Y_{n+1} - y_{(n+1)h}\big\|_{\mathbb{L}_2}^2 \le e^{c_5 h}\big\|Y_n - y_{nh}\big\|_{\mathbb{L}_2}^2 + c_6 h^3, \tag{27}$$

provided $h \le h_{\max}$.

Proof. By Theorems D.13 and D.14 along with Lemma D.15, we have

$$\begin{aligned}
Y_{n+1} - y_{(n+1)h} &= Y_n + f\big(Y_n\big)h + \sigma W_n + \frac{1}{2}f'\big(Y_n\big)\big(\sigma W_n\big)h + \widetilde{R}_n^Y \\
&\quad - y_{nh} - f\big(y_{nh}\big)h - \sigma W_n - f'\big(y_{nh}\big)\sigma\Big(\frac{1}{2}W_n h + H_n h\Big) - \widetilde{R}_n^y \\
&= Y_n - y_{nh} + \big(f\big(Y_n\big) - f\big(y_{nh}\big)\big)h + \frac{1}{2}\big(f'\big(Y_n\big) - f'\big(y_{nh}\big)\big)\big(\sigma W_n\big)h \\
&\quad - f'\big(y_{nh}\big)\big(\sigma H_n\big)h + \widetilde{R}_n^Y - \widetilde{R}_n^y\,.
\end{aligned}$$

The result now follows using exactly the same arguments as in the proof of Theorem D.11. □

Theorem D.17 (Global error estimate for the reversible Heun method in the additive noise setting)**.** *Let $h_{\max} > 0$ be fixed. Then there exists a constant $C_3 > 0$ such that for all $n \in \{0, 1, \cdots, N\}$,*

$$\big\|Y_n - y_{nh}\big\|_{\mathbb{L}_2} \le C_3\, h,$$

provided $h \le h_{\max}$.

Proof. Since $Y_0 = Z_0$, it follows from Theorem D.16 that

$$\big\|Y_n - y_{nh}\big\|_{\mathbb{L}_2}^2 \le c_6 \sum_{k=0}^{n-1} e^{c_5 kh} h^3 = c_6\,\frac{e^{c_5 nh} - 1}{e^{c_5 h} - 1}h^3 \le c_6\,\frac{e^{c_5 T} - 1}{e^{c_5 h} - 1}h^3 = O\big(h^2\big),$$

which gives the desired result. □

It is known that Heun's method achieves second order weak convergence for additive noise SDEs [55]. This can make Heun's method more appealing for SDE simulation than other two-stage methods, such as the standard midpoint method – which is first order weak convergent. Whilst understanding the weak convergence of the reversible Heun method is a topic for future work, we present numerical evidence that it has similar convergence properties as Heun's method for SDEs with additive noise.

We apply the standard and reversible Heun methods to the following scalar anharmonic oscillator:

$$\mathrm{d}y_t = \sin(y_t)\,\mathrm{d}t + \mathrm{d}W_t\,, \tag{28}$$

with $y_0 = 1$, and compute the following error estimates by standard Monte Carlo simulation:

$$\begin{aligned} S_N &:= \sqrt{\mathbb{E}\big[\,\big|Y_N - Y_T^{\text{fine}}\big|\,\big]}, \\ E_N &:= \big|\mathbb{E}\big[Y_N\big] - \mathbb{E}\big[Y_T^{\text{fine}}\big]\big|, \\ V_N &:= \big|\mathbb{E}\big[Y_N^2\big] - \mathbb{E}\big[\big(Y_T^{\text{fine}}\big)^2\,\big]\big|, \end{aligned}$$

where $\{Y_n\}$ denotes a numerical solution of the SDE (28) obtained with step size $h = \frac{T}{N}$ and Y_T^{fine} is an approximation of y_Y obtained by applying Heun's method to (28) with a finer step size of $\frac{1}{10}h$. Both Y_N and Y_T^{fine} are obtained using the same Brownian sample paths and the time horizon is $T = 1$.

The results of this simple numerical experiment are presented in Figures 5 and 6. From the graphs, we observe that the standard and reversible Heun methods exhibit very similar convergence rates (strong order 1.0 and weak order 2.0).

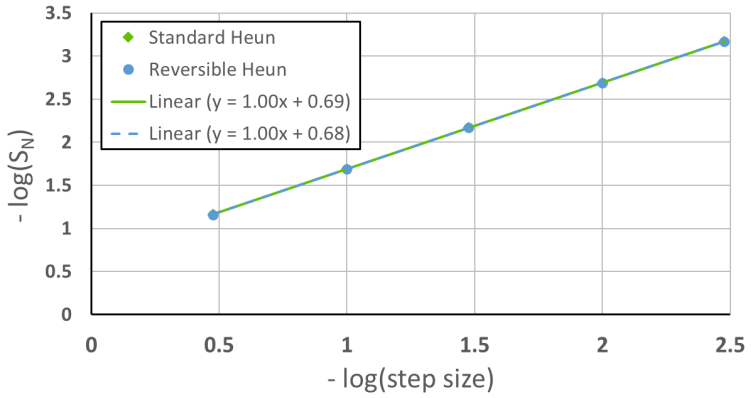

Figure 5: Log-log plot for the strong error estimator S_N computed with 10^7 Brownian sample paths.

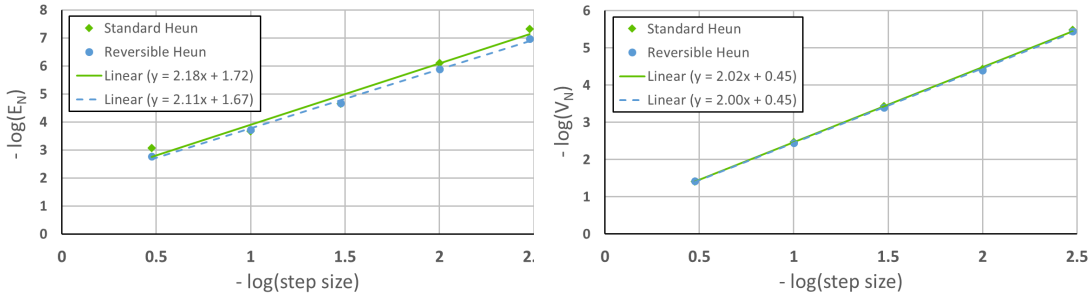

Figure 6: Log-log plots for the weak error estimators computed with 10^7 Brownian sample paths.

D.5 Stability properties of the reversible Heun method in the ODE setting

In this section, we present a stability result for the reversible Heun method when applied to an ODE,

$$y' = f(t, y). \tag{29}$$

Just as for the error analysis, it will be helpful to consider two steps of the reversible Heun method. In particular, the updates for the Z component of the numerical solution satisfy

$$Z_{n+2} = Z_n + 2f\big(t_{n+1}\,, Z_{n+1}\big)h, \tag{30}$$

with the second value of Z being computed using a standard Euler step as $Z_1 := Z_0 + f\big(t_0\,, Z_0\big)h$. That is, $\{Z_n\}$ is precisely the numerical solution obtained by the leapfrog/midpoint method, see [56]. The absolute stability region of this ODE solver is well-known and given below.

Theorem D.18 (Stability region of leapfrog/midpoint method [56, Section 2])**.** *Suppose that we apply the leapfrog/midpoint method to obtain a numerical solution $\{Z_n\}_{n\geq 0}$ for the linear test equation*

$$y' = \lambda y, \tag{31}$$

where $\lambda \in \mathbb{C}$ with $\mathrm{Re}(\lambda) \leq 0$ and $y_0 \neq 0$. Then $|Z_n|$ is bounded for all n if and only if $\lambda h \in [-i, i]$.

Using similar techniques, it is straightforward to extend this result to the reversible Heun method.

Theorem D.19 (Stability region of the reversible Heun method)**.** *Suppose that we apply the reversible Heun method to obtain a pair of numerical solutions $\{Y_n\}_{n\geq 0}\,, \{Z_n\}_{n\geq 0}$ for the linear test equation*

$$y' = \lambda y,$$

where $\lambda \in \mathbb{C}$ with $\mathrm{Re}(\lambda) \leq 0$ and $y_0 \neq 0$. Then $\{Y_n, Z_n\}_{n\geq 0}$ is bounded if and only if $\lambda h \in [-i, i]$.

Proof. By Theorem D.18, it is enough to show that $|Y_n|$ is bounded for all $n \geq 0$ when $\lambda h \in [-i, i]$. It follows from the difference equation (30) and the formula for Z_1 that

$$Z_n = \alpha\eta_1^n + \beta\eta_2^n\,,$$

where the constants $\alpha, \beta, \eta_1, \eta_2 \in \mathbb{C}$ are given by

$$\begin{aligned}
\alpha &:= \frac{1}{2}y_0\left(1 + \frac{1}{\sqrt{1+\lambda^2h^2}}\right),\\
\beta &:= \frac{1}{2}y_0\left(1 - \frac{1}{\sqrt{1+\lambda^2h^2}}\right),\\
\eta_1 &:= \lambda h + \sqrt{1+\lambda^2h^2}\,,\\
\eta_2 &:= \lambda h - \sqrt{1+\lambda^2h^2}\,.
\end{aligned}$$

For each $k \geq 0$, we have $Y_{k+1} = Y_k + \frac{1}{2}\lambda\big(Z_k + Z_{k+1}\big)h$ and so we can explicitly compute Y_n as

$$\begin{aligned}
Y_n &= y_0 + \frac{1}{2}\lambda h\sum_{k=0}^{n-1}\big(Z_k + Z_{k+1}\big)\\
&= y_0 + \frac{1}{2}\lambda h\alpha\big(1+\eta_1\big)\sum_{k=0}^{n-1}\eta_1^k + \frac{1}{2}\lambda h\beta\big(1+\eta_2\big)\sum_{k=0}^{n-1}\eta_2^k\\
&= y_0 + \frac{1}{2}\lambda h\alpha\left(\frac{1+\eta_1}{1-\eta_1}\right)\big(1-\eta_1^n\big) + \frac{1}{2}\lambda h\beta\left(\frac{1+\eta_2}{1-\eta_2}\right)\big(1-\eta_2^n\big).
\end{aligned} \tag{32}$$

Since $\lambda h \in [-i, i]$, we have $\eta_1 = \lambda h + \sqrt{1-|\lambda h|^2}$ and $\eta_2 = \lambda h - \sqrt{1-|\lambda h|^2}$, which implies that

$$|\eta_i|^2 = |\lambda h|^2 + \big(1 - |\lambda h|^2\big) = 1 \quad \text{and} \quad \mathrm{Im}(\eta_i) = |\lambda h|\,,$$

for both $i \in \{1, 2\}$. When $\lambda \neq 0$, it follows that $\eta_1, \eta_2 \in \{z \in \mathbb{C} : |z| = 1\} \setminus \{1\}$ and thus by (32), $|Y_n|$ is bounded for all $n \geq 0$. On the other hand, when $\lambda = 0$, we have $Y_n = y_0$ for all $n \geq 0$. □

Remark D.20. The reversible Heun method is not A-stable for ODEs as that would require $|\eta_i| < 1$.

Remark D.21. The domain $\{\lambda \in \mathbb{C} : \lambda h \in [-i, i]\}$ is also the absolute stability region for the (reversible) asynchronous leapfrog integrator proposed for Neural ODEs in Zhuang et al. [32].

E Sampling Brownian motion

E.1 Algorithm

We begin by providing the complete traversal and splitting algorithm needed to find or create all intervals in the Brownian Interval, as in Section 4. See Algorithm 4.

Here, *List* is an ordered data structure that can be appended to, and iterated over sequentially. For example a linked list would suffice. We let `split_seed` denote a splittable PRNG as in Salmon et al. [34], Claessen and Pałka [35]. We use $*$ to denote an unfilled part of the data structure, equivalent to `None` in Python or a null pointer in C/C++; in particular this is used as a placeholder for the (nonexistent) children of leaf nodes. We use $=$ to denote the creation of a new local variable, and $\leftarrow$ to denote in-place modification of a variable.

E.2 Discussion

The function `traverse` is a depth-first tree search for locating an interval within a binary tree. The search may split into multiple (potentially parallelisable) searches if the target interval crosses the intervals of multiple existing leaf nodes. If the search's target is not found then additional nodes are created as needed.

Sections 4 and E.1 now between them define the algorithm in technical detail.

There are some further technical considerations worth mentioning. Recall that the context we are explicitly considering is when sampling Brownian motion to solve an SDE forwards in time, then the adjoint backwards in time, and then discarding the Brownian motion. This motivates several of the choices here.

Small intervals First, the access patterns of SDE solvers are quite specific. Queries will be over relatively small intervals: the step that the solver is making. This means that the list of nodes populated by `traverse` is typically small. In our experiments we observed it usually only consisting of a single element; occasionally two. In contrast if the Brownian Interval has built up a reasonable tree of previous queries, and was then queried over $[0, s]$ for $s \gg 0$, then a long (inefficient) list would be returned. It is the fact that SDE solvers do not make such queries that means this is acceptable.

Search hints: starting from $\widehat{J}$ Moreover, the queries are either just ahead (fixed-step solvers; accepted steps of adaptive-step solvers) or just before (rejected steps of adaptive-step solvers) previous queries. Thus in Algorithm 3, we keep track of the most recent node $\widehat{J}$, so that we begin `traverse` near to the correct location. This is what ensures the modal time complexity is only $\mathcal{O}(1)$, and not $\mathcal{O}(\log(1/s))$ in the average step size s, which for example would be the case if searching commenced from the root on every query.

LRU cache The fact that queries are often close to one another is also what makes the strategy of using an LRU (least recently used) cache work. Most queries will correspond to a node that have a recently-computed parent in the cache.

Backward pass The queries are broadly made left-to-right (on the forward pass), and then right-to-left (on the backward pass). (Other than the occasional rejected adaptive step.)

Left to its own devices, the forward pass will thus build up a highly imbalanced binary tree. At any one time, the LRU cache will contain only nodes whose intervals are a subset of some contiguous subinterval $[s, t]$ of the query space $[0, T]$. Letting n be the number of queries on the forward pass, then this means that the backward pass will consume $\mathcal{O}\left(n^2\right)$ time – each time the backward pass moves past s, then queries will miss the LRU cache, and a full recomputation to the root will be triggered, costing $\mathcal{O}(n)$. This will then hold only nodes whose intervals are subets of some contiguous subinterval $[u, s]$: once we move past u then this $\mathcal{O}(n)$ procedure is repeated, $\mathcal{O}(n)$ times. This is clearly undesirable.

This is precisely analogous to the classical problem of optimal recomputation for performing back-propagation, whereby a dependency graph is constructed, certain values are checkpointed, and a minimal amount of recomputation is desired; see Griewank [57].

Algorithm 4: Definition of `traverse`

def `bisect`*(I : Node, x : $\mathbb{R}$)*:
    # Only called on leaf nodes
    Let $I = ([a, b], s, I_{\texttt{parent}}, *, *)$
    $s_{\texttt{left}}, s_{\texttt{right}} = \texttt{split\_seed}(s)$
    $I_{\texttt{left}} = ([a, x], s_{\texttt{left}}, J, *, *)$
    $I_{\texttt{right}} = ([x, b], s_{\texttt{right}}, J, *, *)$
    $I \leftarrow ([a, b], s, I_{\texttt{parent}}, I_{\texttt{left}}, I_{\texttt{right}})$
    return

def `traverse_impl`*(I : Node, $[c, d]$: Interval,* nodes *: List[Node])*:
    Let $I = ([a, b], s, I_{\texttt{parent}}, I_{\texttt{left}}, I_{\texttt{right}})$

    # Outside our jurisdiction - pass to our parent
    if $c < a$ or $d > b$ **then**
        `traverse_impl`($I_{\texttt{parent}}, [c, d]$, nodes)
        return
    # It's I that is sought. Add I to the list and return.
    if $c = a$ and $d = b$ **then**
        nodes.append(I)
        return
    # Check if I is a leaf or not.
    if I_{left} is $*$ **then**
        # I is a leaf
        if $a = c$ **then**
            # If the start points align then create children and add on the left child.
            # (Which is created in `bisect`.)
            `bisect`(I, d)
            nodes.append($I_{\texttt{left}}$) # nodes is passed by reference
            return
        # Otherwise create children and pass on to our right child.
        # (Which is created in `bisect`.)
        `bisect`(I, c)
        `traverse_impl`($I_{\texttt{right}}, [c, d]$, nodes)
        return
    else
        # I is not a leaf.
        Let $I_{\texttt{left}} = ([a, m], s_{\texttt{left}}, I, I_{ll}, I_{lr})$
        if $d \leq m$ **then**
            # Strictly our left child's problem.
            `traverse_impl`($I_{\texttt{left}}, [c, d]$, nodes)
            return
        if $c \geq m$ **then**
            # Strictly our right child's problem.
            `traverse_impl`($I_{\texttt{right}}, [c, d]$, nodes)
            return
        # A problem for both of our children.
        `traverse_impl`($I_{\texttt{left}}, [c, m]$, nodes)
        `traverse_impl`($I_{\texttt{right}}, [m, d]$, nodes)
        return

def `traverse`*(I : Node, $[c, d]$: Interval)*:
    Let nodes be an empty List.
    `traverse_impl`(I, $[c, d]$, nodes)
    return nodes

In principle the same solution may be applied: apply a snapshotting procedure in which specific extra nodes are held in the cache. This is a perfectly acceptable solution, but implementing it requires some additional engineering effort, carefully determining which nodes to augment the cache with.

Fortunately, we have an advantage that Griewank [57] does not: we have some control over the dependency structure between the nodes, as we are free to prespecify any dependency structure we like. That is, we do not have to start the binary tree as just a stump. We may exploit this to produce an easier solution.

Given some estimate ν of the average step size of the SDE solver (which may be fixed and known if using a fixed step size solver), a size of the LRU cache L, and *before a user makes any queries*, then we simply make some queries of our own. These queries correspond to the intervals $[0, T/2], [T/2, T], [0, T/4], [T/4, T/2], \ldots$, so as to create a dyadic tree, such that the smallest intervals (the final ones in this sequence) are of size not more than νL. (In practice we use $\frac{4}{5}\nu L$ as an additional safety factor.)

Letting $[s, t]$ be some interval at the bottom of this dyadic tree, where $t \approx s + \frac{4}{5}\nu L$, then we are capable of holding every node within this interval in the LRU cache. Once we move past s on the backward pass, then we may in turn hold the entire previous subinterval $[u, s]$ in the LRU cache, and in particular the values of the nodes whose intervals lie within $[u, s]$ may be computed in only logarithmic time, due to the dyadic tree structure.

This is now analogous to the Virtual Brownian Tree of Li et al. [15], Gaines and Lyons [58]. (Up to the use of intervals rather than points.) If desired, this approach may be loosely interpreted as placing a Brownian Interval on every leaf of a shallow Virtual Brownian Tree.

Recursion errors We find that for some problems, the recursive computations of `traverse` (and in principle also `sample`, but this is less of an issue due to the LRU cache) can occasionally grow very deep. In particular this occurs when crossing the midpoint of the pre-specified tree: for this particular query, the traversal must ascend the tree to the root, and then descend all the way down again. As such `traverse` should be implemented with trampolining and/or tail recursion to avoid maximum depth recursion errors.

CPU vs GPU memory We describe this algorithm as requiring only constant memory. To be more precise, the algorithm requires only constant GPU memory, corresponding to the fixed size of the LRU cache. As the Brownian Interval receives queries then its internal tree tracking dependencies will grow, and CPU memory will increase. For deep learning models, GPU memory is usually the limiting (and so more relevant) factor.

Stochastic integrals What we have not discussed so far is the numerical simulation of integrals such as $\mathbb{W}_{s,t} = \int_s^t W_{s,r} \circ \mathrm{d}W_r$ and $H_{s,t} = \frac{1}{t-s}\int_s^t \left(W_{s,r} - \left(\frac{r-s}{t-s}\right)W_{s,t}\right) \mathrm{d}r$ which are used in higher order SDEs solvers (for example, the Runge-Kutta methods in [59] and the log-ODE method in [54]). Just like increments $W_{s,t}$, these integrals fit nicely into an interval-based data structure.

In general, simulating the pair $(W_{s,t}, \mathbb{W}_{s,t})$ is known to be a difficult problem [60], and exact algorithms are only known when W is one or two dimensional [61]. However, the approximation proposed in [62] and further developed in [63, 64] constitutes a simple and computable solution. Their approach is to generate

$$\widetilde{\mathbb{W}}_{s,t} := \frac{1}{2}W_{s,t} \otimes W_{s,t} + H_{s,t} \otimes W_{s,t} - W_{s,t} \otimes H_{s,t} + \lambda_{s,t},$$

where $\lambda_{s,t}$ is an anti-symmetric matrix with independent entries $\lambda_{s,t}^{i,j} \sim \mathcal{N}(0, \frac{1}{12}(t-s)^2), i < j$.

In these works, the authors input the pairs $\{(W_{t_n,t_{n+1}}, \widetilde{\mathbb{W}}_{t_n,t_{n+1}})\}_{0 \leq n \leq N-1}$ into a SDE solver (the Milstein and log-ODE methods respectively) and prove that the resulting approximation achieves a 2-Wasserstein convergence rate close to $O(1/N)$, where N is the number of steps. In particular, this approach is efficient and avoids the use of costly Lévy area approximations, such as in [65, 66, 67].

F Experimental Details and Further Results

F.1 Metrics

Several metrics were used to evaluate final model performance of the trained Latent SDEs and SDE-GANs.

Real/fake classification A classifier was trained to distinguish real from generated data. This is trained by taking an 80%/20% split of the test data, training the classifier on the 80%, and evaluating its performance on the 20%. This produces a classification accuracy as a performance metric.

We parameterise the classifier as a Neural CDE [45], whose vector field is an MLP with two hidden layers each of width 32. The evolving hidden state is also of width 32. A final classification result is given by applying a learnt linear readout to the final hidden state, which produces a scalar. A sigmoid is then applied and this is trained with binary cross-entropy.

It is trained for 5000 steps using Adam with a learning rate of 10^{-4} and a batch size of 1024.

Smaller accuracies – indicating inability to distinguish real from generated data – are better.

Label classification (train-on-synthetic-test-on-real) Some datasets (in particular the air quality dataset) have labelled classes associated with each sample time series. For these datasets, a classifier was trained on the generated data – possible as every model we train is trained conditional on the class label as an input – and then evaluated on the real test data. This produces a classification accuracy as a performance metric.

We parameterise the classifier as a Neural CDE, with the same architecture as before. A final classification result is given by applying a learnt linear readout to the final hidden state, which produces a vector of unnormalised class probabilities. These are normalised with a softmax and trained using cross-entropy.

It is trained for 5000 steps using Adam with a learning rate of 10^{-4} and a batch size of 1024.

Larger accuracies – indicating similarity of real and generated data – are better.

Prediction (train-on-synthetic-test-on-real) A sequence-to-sequence model is trained to perform time series forecasting: given the first 80% of a time series, can the latter 20% be predicted. This is trained on the generated data, and then evaluated on the real test data. This produces a regression loss as a performance metric.

We parameterise the predictor as a sequence-to-sequence Neural CDE / Neural ODE pair. The Neural CDE is as before. The Neural ODE has a vector field which is an MLP of two hidden layers, each of width 32. Its evolving hidden state is also of width 32. An evolving prediction is given by applying a learnt linear readout to the evolving hidden state, which produces a time series of predictions. These are trained using an L^2 loss.

It is trained for 5000 steps using Adam with a learning rate of 10^{-4} and a batch size of 1024.

Smaller losses – indicating similarity of real and generated data – are better.

Maximum mean discrepancy Maximum mean discrepancies [68] can be used to compute a (semi)distance between probability distributions. Given some set $\mathcal{X}$, a fixed feature map $\psi \colon \mathcal{X} \to \mathbb{R}^m$, a norm $\|\cdot\|$ on $\mathbb{R}^m$, and two probability distributions $\mathbb{P}$ and $\mathbb{Q}$ on $\mathcal{X}$, this is defined as

$$\|\mathbb{E}_{P\sim\mathbb{P}}\left[\psi(P)\right] - \mathbb{E}_{Q\sim\mathbb{Q}}\left[\psi(Q)\right]\|.$$

In practice $\mathbb{P}$ corresponds to the true distribution, of which we observe samples of data, and $\mathbb{Q}$ corresponds to the law of the generator, from which we may sample arbitrarily many times. Given N empirical samples P_i from the true distribution, and M generated samples Q_i from the generator, we may thus approximate the MMD distance via

$$\left\|\frac{1}{N}\sum_{i=1}^{N}\psi(P_i) - \frac{1}{M}\sum_{i=1}^{M}\psi(Q_i)\right\|.$$

In our case, $\mathcal{X}$ corresponds to the observed time series, and we use a depth-5 signature transform as the feature map [69, 70]. Similarly, the untruncated signature can be used as a feature map [71, 72, 73, 74, 75].

(Note that MMDs may also be used as differentiable optimisation metrics provided ψ is differentiable [76, 77]. A mistake we have seen 'in the wild' for training SDEs is to choose a feature map that is overly simplistic, such as taking ψ to be the marginal mean and variance at all times. Such a feature map would fail to capture time-varying correlations; for example W and $t \mapsto W(0)\sqrt{t}$, where W is a Brownian motion, would be equivalent under this feature map.)

Smaller values – indicating similarity of real and generated data – are better.

F.2 Common details

The following details are in common to all experiments.

Libraries used PyTorch was used as an autodifferentiable tensor framework [27].

SDEs were solved using `torchsde` [42]. CDEs were solved using `torchcde` [78]. ODEs were solved using `torchdiffeq` [79]. Signatures were computed using Signatory [74].

Tensors had their shapes annotated using the torchtyping [49] library, which helped to enforce correctness of the implementation.

Hyperparameter optimisation was performed using the Ax library [80].

Numerical methods SDEs were solved using either the reversible Heun method or the midpoint method (as per the experiment), and trained using continuous adjoint methods.

The CDE used in the discriminator of an SDE-GAN was solved using either the reversible Heun method, or the midpoint method, in common with the choice made in the generator, and trained using continuous adjoint methods.

The ODEs solved for the train-on-synthetic-test-on-real prediction metric used the midpoint method, and were trained using discretise-then-optimise backpropagation. The CDEs solved for the various evaluation metrics used the midpoint method, and trained using discretise-then-optimise backpropagation. (These were essentially arbitrary choices – we merely needed to fix some choices throughout to ensure a fair comparison.)

Normalisation Every dataset is normalised so that its initial value (at time $= 0$) has mean zero and unit variance. (That is, calculate mean and variance statistics of just the initial values in the dataset, and then normalise every element of the dataset using these statistics.)

We speculate that normalising based on the initial condition produces better results than calculating mean and variance statistics over the whole trajectory, as the rest of the trajectory cannot easily be learnt unless its initial condition is well learnt first. We did not perform a thorough investigation of this topic, merely finding that this worked well enough on the problems considered here.

The times at which observations were made were normalised to have mean zero and unit range. (This is of relevance to the modelling, as the generated samples must be made over the same timespan: that is the integration variable t must correspond to some parameterisation of the time at which data is actually observed.)

Dataset splits We used 70% of the data for training, 15% for validation and hyperparameter optimisation, and 15% for testing.

Optimiser The batch size is always taken to be 1024. The number of training steps varies by experiment, see below.

We use Adam [81] to train every Latent SDE.

Following Kidger et al. [16] we use Adadelta [82] to train every SDE-GAN.

Stochastic weight averaging When training SDE-GANs, we take a Cesàro mean over the latter 50% of the training steps, of the generator's weights, to produce the final trained model. Known as 'stochastic weight averaging' this often slightly improves GAN training [83, 84].

Architectures Each of $\zeta_\theta, \mu_\theta, \sigma_\theta, \xi_\phi, f_\phi, g_\phi$ were parameterised as MLPs. (From equations (1), (2), (10).)

Following Li et al. [15], then ν_ϕ (of equation (10)) was parameterised as an MLP composed with a GRU.

In brief, that is $\nu_\phi(t, \widehat{X}_t, Y_{\text{true}}) = \nu_\phi^1(t, \widehat{X}_t, \nu_\phi^2(Y_{\text{true}}|_{[t,T]}))$, where ν_ϕ^1 is an MLP, and ν_ϕ^2 is a GRU run backwards-in-time from T to t over whatever discretisation of $Y_{\text{true}}|_{[t,T]}$ is observed.

In all cases, for simplicity, the LipSwish activation function was used throughout.

Hyperparameter optimisation Hyperparameter optimisation used the default optimisation strategy provided by Ax (initial quasirandom Sobol sampling followed by Bayesian optimisation). The optimisation metric was the MMD evaluation metric, due to the speed at which it can be computed relative to the other optimisation metrics (which require training an auxiliary model).

The Latent SDE model was hyperoptimised on every dataset. To ensure we do not bias results in our favour, the hyperoptimised models used the midpoint method, not the reversible Heun method, throughout, and was optimised using discretise-then-optimise backpropagation.

The SDE-GAN models then used the same hyperparameters where applicable, for example on learning rate, neural network size, and so on. This fixes mosts of the hyperparameters. A few extra hyperparameters were then chosen manually.

First, the size of the initial noise V and the dimensionality of the Brownian motion W were arbitrarily fixed at 10.

Second, we adjusted the initialisation strategy for the parameters of the SDE. For each dataset, we picked some constants

$$\alpha > 0, \quad \beta > 0, \tag{33}$$

and multiplied the parameters θ at initialisation by either α or β, depending on whether the parameters were used to generate the initial condition (ζ_θ), or were used in the vector fields of the SDE ($\mu_\theta, \sigma_\theta, \ell_\theta$), respectively. This helped to ensure that the SDE had a good initialisation, and therefore took fewer steps to train. The values α, β were chosen by manually plotting generated samples from an untrained model against real data. (A less naïve approach would be nice.)

The one exception is the Ornstein–Uhlenbeck dataset, for which the SDE-GAN had ζ_θ, μ_θ, $\sigma_\theta, \xi_\phi, f_\phi, g_\phi$ parameterised as MLPs with one hidden layer of width 32, and had an evolving hidden state X of size 32; these figures were chosen as being known to work based on early experiments.

Compute resources Experiments were performed on an internal GPU cluster. Each experiment used only a single GPU at a time. Amount of compute time varied depending on the experiment – some took a few hours, some took a few days. Precise times given in the tables of results.

(Moreover some would likely have taken a few weeks without the algorithmic speed improvements introduced in this paper. Recall that each baseline experiment used the improvements introduced in the other sections of this paper, simply to produce tractable training times. For example the Brownian Interval was used throughout.)

GPU types varied between GeForce RTX 2080 Ti, Quadro GP100, and A100s.

F.3 Weights dataset

Technical details We consider a dataset of weights of a small convolutional network, as it is trained to classify MNIST, with training using stochastic gradient descent, as in Kidger et al. [16]. The network was trained 10 times, and all weight trajectories, across all runs, were aggregated together to form a dataset of univariate time series.

Each time series is 50 elements long, corresponding to how the weights change over 50 epochs.

Table 4: SDE-GAN on weights dataset. Mean $\pm$ standard deviation averaged over three runs.

Solver	Test Metrics			
	Real/fake classification accuracy (%)	Prediction loss	MMD ($\times 10^{-2}$)	Training time (days)
Midpoint	77.0 ± 6.9	0.303 ± 0.369	4.38 ± 0.67	5.12 ± 0.01
Reversible Heun	$\mathbf{75.5 \pm 0.01}$	$\mathbf{0.068 \pm 0.037}$	$\mathbf{1.75 \pm 0.3}$	$\mathbf{2.59 \pm 0.05}$

We train an SDE-GAN on this dataset. We train the generator and discriminator for 80 000 steps each.

Each MLP $(\zeta_\theta, \mu_\theta, \sigma_\theta, \xi_\phi, f_\phi, g_\phi)$ is parameterised as having two hidden layers each of width 67. The evolving hidden states X, H are each of size 62.

The parameters of ζ_θ, ξ_ϕ have a learning rate of 4.9×10^{-3}. The parameters of $\mu_\theta, \sigma_\theta, f_\phi, g_\phi$ have a learning rate of 1.3×10^{-3}.

The initialisation scaling parameters α and β (of equation (33)) were selected to be 4.5 and 0.25 respectively.

Results We compare the reversible Heun method to the midpoint method on the weights dataset, by training an SDE-GAN.

We compute three test metrics: classification of real versus generated data, forecasting via train-on-synthetic-test-on-real, and a maximum mean discrepancy. We additionally report training time. See Table 4.

First and most notably, we see a dramatic reduction in training time: the training time of the reversible Heun method is roughly half that of the midpoint method. This corresponds to the reduction in vector field evaluations of the reversible Heun method.

We additionally see better performance on the test metrics, as compared to the midpoint method. This corresponds to the calculation of numerically precise gradients via the reversible Heun method.

We believe that these test metrics could be further improved (in particular the classification accuracy) given further training time.

The Brownian Interval (Section 4) is used to sample Brownian noise, and the SDE-GAN is trained using clipping as in Section 5. (Without either of which the baseline experiments would have taken infeasibly long to run.)

F.4 Air quality dataset

Technical details This is a dataset of air quality samples over Beijing, as they vary over the course of a day. Each time series is 24 elements long, corresponding to a single hour each day. We consider specifically the PM2.5 particulate matter concentration, and ozone concentration, to produce a dataset of bivariate time series. In particular the ozone channel was selected as displaying obvious non-autonomous behaviour: the latter half of the time series often includes a peak.

This dataset is available via the UCI machine learning repository [85, 86].

Each time series has a label, corresponding to which of 12 different locations the measurements were made at.

We train a Latent SDE on this dataset. We train for 40 000 steps.

Each MLP $(\zeta_\theta, \mu_\theta, \sigma_\theta, \xi_\phi, \nu^1_\phi)$ is parameterised as having a single hidden layer of width 84. The evolving hidden state X is of size 63.

ν^2_ϕ was parameterised as GRU with hidden size 84, whose final hidden state has a learnt affine map applied, to produce vector of size 60.

Table 5: Latent SDE on air quality dataset. Mean ± standard deviation averaged over three runs.

Solver	Test Metrics				
	Real/fake classification accuracy (%)	Label classification accuracy (%)	Prediction loss	MMD ($\times 10^{-3}$)	Training time (hours)
Midpoint	**92.3 ± 0.02**	46.3 ± 5.1	**0.281 ± 0.009**	5.91 ± 2.06	5.58 ± 0.54
Reversible Heun	96.7 ± 0.01	**49.2 ± 0.02**	0.314 ± 0.005	**4.72 ± 2.90**	**4.47 ± 0.31**

The parameters of ζ_θ have learning rate 1.1×10^{-4}. The parameters of $\mu_\theta, \sigma_\theta, \xi_\phi, \nu_\phi^1, \nu_\phi^2$ have learning rate 1.9×10^{-5}.

The initialisation scaling parameters α and β (of equation (33)) were selected to be 2 and 1 respectively.

Results We compare the reversible Heun method to the midpoint method on the air quality dataset, by training a Latent SDE.

We compute four test metrics: classification of real versus generated data, classification via train-on-synthetic-test-on-real, forecasting via train-on-synthetic-test-on-real, and a maximum mean discrepancy. We additionally report training time. See Table 5.

Here, the most important metric is again the improvement in training time: whilst less dramatic than the previous SDE-GAN experiment, it is still a speed improvement of $1.2\times$.

Performance on the test metrics varies between the solvers, without a clear pattern.

It is worth noting the apparently poor real/fake classification accuracies obtained using either solver. This is typical of Latent SDEs in general, in particular as opposed to SDE-GANs. Whilst Latent SDEs are substantially quicker to train, they tend to produce less convincing samples.

F.5 Gradient error analysis

We investigate the error made in the gradient calculation using continuous adjoints. We consider using the midpoint method, Heun's method, and the reversible Heun method, and vary the step size in decreasing powers of two.

Our test problem is to compute $\mathrm{d}X_1/\mathrm{d}X_0$ and $\mathrm{d}X_1/\mathrm{d}\theta$ over a batch of 32 samples of

$$\mathrm{d}X_t = f_\theta(t, X_t)\,\mathrm{d}t + g_\theta(t, X_t) \circ \mathrm{d}W_t,$$

where $X_t, f_\theta(t, X_t) \in \mathbb{R}^{32}, W_t \in \mathbb{R}^{16}, g_\theta(t, X_t) \in \mathbb{R}^{32,16}$ and f_θ, g_θ are feedforward neural networks with a single hidden layer of width 8, and LipSwish activation function. Additionally f_θ has a $\tanh$ as a final nonlinearity, and g_θ has a sigmoid as final nonlinearity.

We compare the gradients computed via optimise-then-discretise against discretise-then-optimise, and plot the relative L^1 error. Letting $\delta_{\mathrm{o-d}} \in \mathbb{R}^M$ and $\delta_{\mathrm{d-o}} \in \mathbb{R}^M$ denote these gradients (with M equal to the size of X_0 plus the size of θ), then this is defined as

$$\frac{\sum_{i=1}^M \left|\delta_{\mathrm{o-d}}^i - \delta_{\mathrm{d-o}}^i\right|}{\max\{\sum_{i=1}^M \left|\delta_{\mathrm{o-d}}^i\right|, \sum_{i=1}^M \left|\delta_{\mathrm{d-o}}^i\right|\}}.$$

Numerical values are shown in Table 6. Results are plotted graphically in the main text.

F.6 Brownian benchmarks

We benchmark the Brownian Interval against the Virtual Brownian Tree across several benchmarks.

Access benchmarks We subdivide the interval $[0, 1]$ into a disjoint union of equal-sized intervals. Across each interval we then place a query for a small Brownian increment.

We consider subdividing $[0, 1]$ into different numbers of subintervals (and so of different sizes): either 10, 100 or 1000 subintervals.

We consider several access patterns.

Sequential access: this involves querying every interval in order, from 0 to 1. Every interval is queried precisely once. This simulates an SDE solve from 0 to 1.

Doubly sequential access: this involves querying every interval in order, from 0 to 1, and then querying them again in reverse order, from 1 to 0. Every interval is queried precisely twice. This simulates an SDE solve from 0 to 1, followed by a backpropagation via the continuous adjoint method, from 1 to 0.

Random access: this involves querying every interval precisely once, in a random order.

We consider several batch sizes: either a single Brownian simulation, a typical batch size of 2560 simultaneous Brownian simulations (corresponding to a batch of size 256, each with a vector of 10 Brownian motions), and a large batch size of 32768 simultaneous Brownian simulations.

For every such combination we report metrics for the Brownian Interval and the Virtual Brownian Tree.

For every such combination we run 32 repeats. The reported metric is the fastest (minimum time) over these repeats.[8]

See Tables 7, 8, and 9.

The Brownian Interval is consistently and substantially faster than the Virtual Brownian Tree, across all access patterns, batch sizes, and number of subintervals.

On the doubly sequential access benchmark (emulating the SDE solve and backpropagation typical in practice), we see that total speed-ups vary from a factor of $2.89\times$ to a factor of $13.5\times$. That is, at minimum, speed is roughly tripled. Potentially it is improved by over an order of magnitude. Typical values are speed-ups of 6–$8\times$.

SDE solve benchmarks We now benchmark the Brownian Interval against the Virtual Brownian Tree on the actual task of solving and backpropagating through an SDE.

As before we consider several numbers of subintervals, several batch sizes, and run 32 repeats and take the fastest time.

Our test SDE is an Itô SDE with diagonal noise:

$$\mathrm{d}X_t^i = \tanh(A^{i,j}X_t^j)\,\mathrm{d}t + \delta^{i,k}\delta^{i,l}\tanh(B^{k,j}X_t^j)\,\mathrm{d}W_t^l,$$

where either $X_t, W_t \in \mathbb{R}^1$, $X_t, W_t \in \mathbb{R}^{10}$, or $X_t, W_t \in \mathbb{R}^{16}$, corresponding to the small, medium and large batch sizes considered in the previous set of benchmarks. Likewise $A, B \in \mathbb{R}^{1\times1}$, $A, B \in \mathbb{R}^{10\times10}$, or $A, B \in \mathbb{R}^{16\times16}$ are random matrices. $\tanh$ is applied component-wise.

For $i = 10, 100, 1000$ subintervals, we calculate a forward pass for $[0, 1]$ using the Euler–Maruyama method, to calculate $X_{j/(i-1)}$ for $j \in \{0, \ldots, i-1\}$. We then backpropagate from the vector $(X_{j/(i-1)})_j$ to X_0, using the continuous adjoint method.

See Table 10.

We once again see that the Brownian Interval is uniformly and substantially faster than the Virtual Brownian Tree, in all regimes. On smaller problems (batch size equal to 1 or 2560), then it is typically twice as fast; at worst it is $1.89\times$ as fast. Meanwhile on larger problems (batch size equal to 32768), then it is typically ten times as fast.

These benchmarks include the realistic overheads involved in solving an SDE (such as evaluating its vector fields), and represent typical speed-ups from using the Brownian Interval over the Virtual Brownian Tree.

[8]Not the mean. Errors in speed benchmarks are one-sided, and so the minimum time represents the least noisy measurement.

Table 6: Relative L^1 error on the gradient analysis test problem.

	Step size			
Method	2^0	2^{-2}	2^{-4}	2^{-6}
Midpoint	9.4×10^{-2}	1.1×10^{-2}	2.6×10^{-3}	7.2×10^{-4}
Heun	1.4×10^{-1}	5.9×10^{-2}	1.4×10^{-2}	2.9×10^{-3}
Reversible Heun	$\mathbf{1.9 \times 10^{-17}}$	$\mathbf{2.7 \times 10^{-16}}$	$\mathbf{3.3 \times 10^{-16}}$	$\mathbf{5.7 \times 10^{-16}}$

2^{-8}	2^{-10}	2^{-14}
1.8×10^{-4}	4.8×10^{-5}	2.5×10^{-6}
8.2×10^{-4}	2.8×10^{-4}	1.3×10^{-5}
$\mathbf{1.0 \times 10^{-15}}$	$\mathbf{1.8 \times 10^{-15}}$	$\mathbf{6.5 \times 10^{-15}}$

Table 7: Speed on the sequential access benchmark. Minimum over 32 runs.

	Speed (seconds)	
Batch size, subinterval number	Virtual Brownian Tree	Brownian Interval
1, 10	8.08×10^{-3}	$\mathbf{2.59 \times 10^{-3}}$
1, 100	7.84×10^{-2}	$\mathbf{3.12 \times 10^{-2}}$
1, 1000	7.96×10^{-1}	$\mathbf{3.27 \times 10^{-1}}$
2560, 10	1.90×10^{-2}	$\mathbf{4.13 \times 10^{-3}}$
2560, 100	1.94×10^{-1}	$\mathbf{4.95 \times 10^{-2}}$
2560, 1000	1.93×10^{0}	$\mathbf{5.15 \times 10^{-1}}$
32768, 10	1.16×10^{-1}	$\mathbf{1.71 \times 10^{-2}}$
32768, 100	1.40×10^{0}	$\mathbf{2.02 \times 10^{-1}}$
32768, 1000	1.43×10^{1}	$\mathbf{2.13 \times 10^{0}}$

Table 8: Speed on the doubly sequential access benchmark. Minimum over 32 runs.

	Speed (seconds)	
Batch size, subinterval number	Virtual Brownian Tree	Brownian Interval
1, 10	2.02×10^{-2}	$\mathbf{2.79 \times 10^{-3}}$
1, 100	2.42×10^{-1}	$\mathbf{4.96 \times 10^{-2}}$
1, 1000	1.67×10^{0}	$\mathbf{5.79 \times 10^{-1}}$
2560, 10	3.23×10^{-2}	$\mathbf{4.31 \times 10^{-3}}$
2560, 100	3.91×10^{-1}	$\mathbf{8.05 \times 10^{-2}}$
2560, 1000	4.01×10^{0}	$\mathbf{9.56 \times 10^{-1}}$
32768, 10	2.30×10^{-1}	$\mathbf{1.70 \times 10^{-2}}$
32768, 100	2.92×10^{0}	$\mathbf{3.49 \times 10^{-1}}$
32768, 1000	2.91×10^{1}	$\mathbf{4.36 \times 10^{0}}$

Table 9: Speed on the random access benchmark. Minimum over 32 runs.

	Speed (seconds)	
Batch size, subinterval number	Virtual Brownian Tree	Brownian Interval
1, 10	1.53×10^{-2}	$\mathbf{2.56 \times 10^{-3}}$
1, 100	1.09×10^{-1}	$\mathbf{5.17 \times 10^{-2}}$
1, 1000	8.52×10^{-1}	$\mathbf{1.02 \times 10^{0}}$
2560, 10	1.50×10^{-2}	$\mathbf{3.56 \times 10^{-3}}$
2560, 100	1.86×10^{-1}	$\mathbf{8.96 \times 10^{-2}}$
2560, 1000	1.99×10^{0}	$\mathbf{1.80 \times 10^{0}}$
32768, 10	1.18×10^{-1}	$\mathbf{1.60 \times 10^{-2}}$
32768, 100	1.47×10^{0}	$\mathbf{3.63 \times 10^{-1}}$
32768, 1000	1.44×10^{1}	$\mathbf{9.08 \times 10^{0}}$

Table 10: Speed on the sequential access benchmark. Minimum over 32 runs.

	Speed (seconds)	
Batch size, subinterval number	Virtual Brownian Tree	Brownian Interval
1, 10	1.42×10^{-1}	$\mathbf{7.18 \times 10^{-2}}$
1, 100	1.55×10^{0}	$\mathbf{8.16 \times 10^{-1}}$
1, 1000	1.61×10^{1}	$\mathbf{8.52 \times 10^{0}}$
2560, 10	2.61×10^{-1}	$\mathbf{1.13 \times 10^{-1}}$
2560, 100	3.00×10^{0}	$\mathbf{1.31 \times 10^{0}}$
2560, 1000	3.00×10^{1}	$\mathbf{1.31 \times 10^{1}}$
32768, 10	4.34×10^{1}	$\mathbf{4.66 \times 10^{0}}$
32768, 100	4.99×10^{2}	$\mathbf{4.68 \times 10^{1}}$
32768, 1000	1.54×10^{3}	$\mathbf{4.74 \times 10^{2}}$

F.7 Time-dependent Ornstein–Uhlenbeck dataset

Technical details This is a dataset of univariate samples of length 32 from the time-dependent Ornstein–Uhlenbeck process

$$\mathrm{d}Y_{\mathrm{true},t} = (\rho t - \kappa Y_{\mathrm{true},t})\,\mathrm{d}t + \chi\,\mathrm{d}W_t,$$

with $\rho = 0.02, \kappa = 0.1$ and $\chi = 0.4$ and $t \in [0, 31]$

We train an SDE-GAN on this dataset. We train the generator for 20 000 steps and the discriminator for 100 000 steps.

Each MLP $(\zeta_\theta, \mu_\theta, \sigma_\theta, \xi_\phi, f_\phi, g_\phi)$ is parameterised as having a single hidden layer of width 32. The evolving hidden states X, H are each of size 32.

The parameters of ζ_θ, ξ_ϕ have a learning rate of 1.6×10^{-3}. The parameters of $\mu_\theta, \sigma_\theta, f_\phi, g_\phi$ have a learning rate of 2.0×10^{-4}.

The initialisation scaling parameters α and β (of equation (33)) were selected to be 5 and 0.5 respectively.

Results We compare training SDE-GANs by careful clipping (as in Section 5) to using gradient penalty (as in Kidger et al. [16]) on the OU dataset.

As our implementation of the reversible Heun method does not support a double backward, we provide two comparisons of interest: reversible Heun method with clipping against midpoint with gradient penalty, and midpoint with clipping against midpoint with gradient penalty.

Table 11: SDE-GAN on OU dataset. Mean $\pm$ standard deviation averaged over three runs.

Solver	Test Metrics			Training time (hours)
	Real/fake classification accuracy (%)	Prediction loss	MMD ($\times 10^{-1}$)	
Midpoint with gradient penalty	98.2 ± 2.4	2.71 ± 1.03	2.58 ± 1.81	55.0 ± 27.7
Midpoint with clipping	93.9 ± 6.9	1.65 ± 0.17	1.03 ± 0.10	32.5 ± 12.1
Reversible Heun with clipping	$\mathbf{67.7 \pm 1.1}$	$\mathbf{1.38 \pm 0.06}$	$\mathbf{0.45 \pm 0.22}$	$\mathbf{29.4 \pm 8.9}$

We compute three test metrics: classification of real versus generated data, forecasting via train-on-synthetic-test-on-real, and a maximum mean discrepancy. We additionally report training time. See Table 11.

We see that reversible Heun with clipping dominates midpoint with clipping, which in turn dominates midpoint with gradient penalty, across all metrics.

As per Kidger et al. [16], the poor performance of gradient penalty is due in part to the numerical errors of a double adjoint. Switching to midpoint with clipping produces substantially better test metrics. It additionally improves training speed by $1.41\times$.

Switching from midpoint with clipping to reversible Heun with clipping then produces another substantial boost to the test metrics – most notably the real-versus-fake classification accuracy. It also improves training speed by another $1.09\times$.

G Ethical statement

SDEs are already a widely used modelling paradigm, primarily in fields such as finance and science. In this regard they are a tried-and-tested mathematical tool, with, to the best of the authors knowledge, no significant ethical concerns attached.

As this paper extends this existing methodology, then broadly speaking we expect the same to be true.

Expected applications We anticipate the results of this paper as having applications to finance and the sciences. For example, to model the movement of asset prices, or to model predator-prey interactions.

As such no significant negative societal impacts are anticipated.

Environmental impacts The primary contributions of this paper are speed improvements to existing methodologies. As such we anticipate a positive environmental impact from this paper, due to a reduction in the compute resources necessary to train model.

Dataset content The data we are using contains no personally identifiable or offensive content.

Dataset provenance All data used has been made publicly available, for example via the UCI machine learning repository [85]. To the best of our knowledge this availability was voluntary, and so we believe the use of the data to be ethical and without licensing issues.