Determining Gradients for Neural SDEs

Andrew Stonehouse, Andy Sun, Petru Ciur, Shaharyar Siddiqui, Stephen Kowalski

MFM 703

McMaster University

December 16$^{\text{th}}$, 2024

Introduction and Motivation

Stochastic Differential Equations (SDEs) are a powerful tool for modeling stochastic processes in a variety of fields. In a financial context, they provide the underlying framework for the Black–Scholes equation, which is one of the most fundamental models in all of financial mathematics. Beyond just option pricing, SDEs are used to understand the price dynamics of most asset classes in general, also being used in the fixed income and foreign exchange sectors [1]. The growing body of research on SDEs has been critical to bringing about the fast-paced, technology-enabled, data-driven financial world that we know today. The development of Neural Stochastic Differential Equations (Neural SDEs) has expanded on the strong modeling capabilities of SDEs by leveraging the universal approximation capacity of neural networks. At a high level, Neural SDEs can learn complex dynamics from data associated with a particular stochastic process. To do this, the drift and diffusion terms of an SDE are parametrized as neural networks, allowing Neural SDEs to approximate any temporal stochastic process (subject to some constraints) [2]. As a result, Neural SDEs are extremely powerful modelling tools that can be used in a variety of scenarios, from simulation to forecasting and generative modeling.

Neural SDEs combine the benefits of recurrent neural networks (RNNs) and SDEs. RNNs have shown an aptitude for function approximation and have good memory efficiency. The drift and diffusion terms of an SDE are included explicitly in the model, which can be used to define the mean and variance of future states. Most SDEs that involve Brownian Motion (a type of continuous stochastic process) also satisfy typical regularity and continuity constraints, such as Lipschitz continuity [3]. This combination between RNNs and SDEs makes Neural SDEs well-suited for modeling any temporal data that has inherent randomness. As previously mentioned, they have a variety of applications in financial modeling, such as asset price prediction and risk assessments. However, the unique structure of Neural SDEs also comes with considerable computational challenges, particularly in the training process where gradients are computed through an SDE solver.

Unlike most standard neural networks, in which backpropagation relies on simply applying the chain rule, Neural SDEs require backpropagation through an entire stochastic path. This involves solving a backward-in-time SDE called an adjoint SDE that ultimately yields the necessary gradients for optimization [3]. However, this process is computationally expensive and prone to numerical inaccuracies, which has made Neural SDEs relatively unappealing for practical implementations in the past.

The computational cost arises from the twofold requirement of solving both the forward and backward SDEs, which is far more complicated than doing this for deterministic differential equations. First, the forward SDE must be solved to compute the model's predictions for the gradients. Then, to refine this prediction and get closer to the theoretically correct gradients, the adjoint SDE must be solved backward in time, effectively doubling the computational workload. Each time step in both directions requires evaluating the previously mentioned neural networks that parameterize the drift and diffusion terms, adding even more work to the process. For high-dimensional data or complex models, this computational burden is exacerbated. As a result, making training Neural SDEs without proper optimization all but infeasible. In addition to computational cost, numerical truncation errors are a significant challenge. Both SDE solvers rely on numerical methods to approximate continuous paths using discrete time steps. Larger time steps can reduce overall computation time but leads to greater truncation errors, yielding inaccurate gradients. Conversely, smaller time steps reduce truncation errors but drastically increase computational time. When the truncation errors in the forward and backward passes of the solver are not equal, the accuracy problem worsens, which leads to poor gradient computations and inhibits model convergence.

These challenges are particularly impactful when Neural SDEs are trained as Variational Autoencoders (VAEs) or Generative Adversarial Networks (GANs) [3]. VAEs require gradients to be accurate in order to optimize representation of the latent space, and any inaccuracies can lower the quality of reconstructions and generative samples. Similarly, GANs are prone to unstable training dynamics to begin with because of the adversarial relationship between the generator and discriminator, the min-max objective that they try to optimize. Inaccurate gradients can further destabilize GAN training, sometimes resulting in non-convergence

or other issues.

If these speed and accuracy issues are addressed appropriately, Neural SDEs could become much more viable for real-world applications. Kidger et al.'s *paper Efficient and Accurate Gradients for Neural SDEs* introduces innovative techniques to overcome these challenges, including the Reversible Heun Method and Brownian Intervals [2]. The former is the first algebraically reversible SDE solver, ensuring consistent truncation errors between forward and backward passes. Its implementation effectively eliminates numerical gradient errors and increases computational speed by a factor of 1.87 compared to standard SDE solvers [2]. Additionally, Brownian Intervals are a very efficient and scalable technique for sampling Brownian motion. They also allow for the reconstruction of particular paths of Brownian motion, making them a powerful tool for determining Neural SDE gradients. Overall, these contributions make Neural SDEs much more practical for tasks such as simulation, forecasting, and financial modeling, which require both high accuracy and computational efficiency.

Theoretical Considerations

Neural Stochastic Differential Equations are a powerful new approach to modeling time-series data that combine the strengths of neural networks and stochastic differential equations. This allows them to learn complex temporal dynamics, especially in situations involving uncertainty and irregular time intervals, where traditional methods like RNNs often struggle. Neural SDEs excel because they inherently incorporate randomness, enabling them to model systems with noise or unpredictable elements, and they can handle data with irregular time steps.

At the heart of Neural SDEs lies the concept of a latent stochastic process, which describes the evolution of a system's hidden state over time. This process is parameterized by a neural network, allowing it to learn non*linear* dynamics from data. The neural network guides the trajectory of the stochastic process, capturing the underlying patterns and dependencies in the observed data.

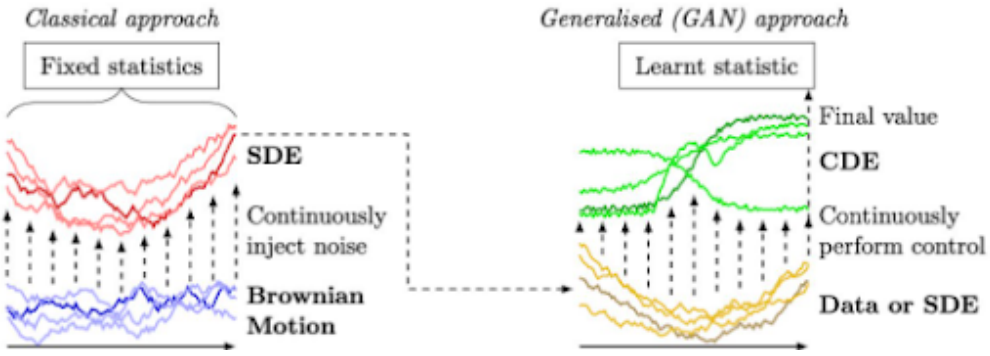

Figure 1: Difference between classical approach and SDE-GAN approach

SDE-GANs take this concept further by integrating Neural SDEs into the generative adversarial network (GAN) framework. In this setting, the Neural SDE acts as the generator, producing synthetic time-series data by sampling from the learned stochastic process (Figure 1). The discriminator, another neural network, then attempts to distinguish between the generated data and real data. This adversarial process drives both networks to improve, with the generator learning to produce increasingly realistic samples and the discriminator becoming more adept at identifying subtle differences. This interplay ultimately leads to a powerful generative model capable of capturing the complex dynamics and stochasticity of real-world time-series data.

However, training these models presents challenges, particularly in efficiently computing gradients for

optimization. Recent breakthroughs have addressed these challenges. The Brownian Interval, enables efficient sampling of Brownian motion, which is crucial for SDE simulations, resulting in faster query times and reduced memory usage. Furthermore, gradient-free training techniques have been developed for SDE-based Generative GANs, avoiding the computational cost and instability associated with traditional methods.

Training Neural SDEs involves simulating Brownian motion, which is like tracking the random movement of a particle. Often, we need to reconstruct specific portions of this particle's path, similar to knowing its starting and ending points and figuring out the route it took in between. This is where the concept of a Brownian bridge comes in. However, storing the entire path can be memory-intensive, especially for complex simulations.

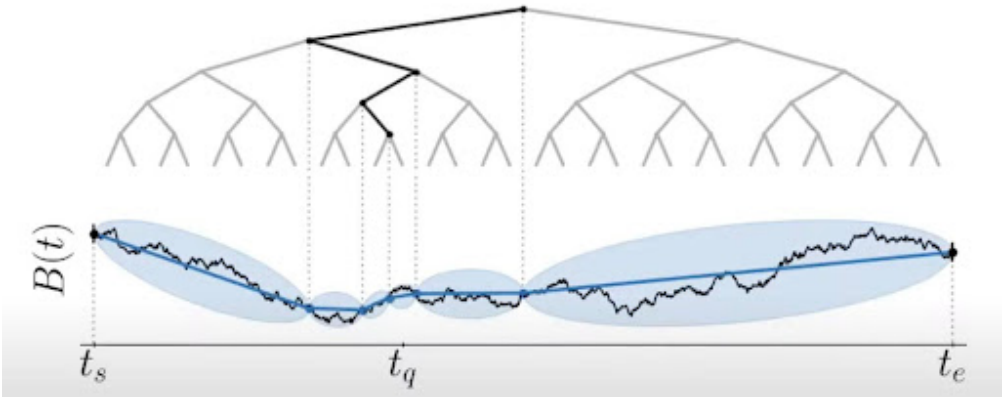

Figure 2: Binary Tree for Brownian Motion path

The Brownian Interval offers an elegant solution. It's a data structure that acts like a map of the Brownian motion path, organized as a binary tree (Figure 2). Instead of storing every point, it stores key intervals and random seeds, allowing for efficient reconstruction of any part of the path on demand using a formula called Lévy's Brownian bridge. This approach is analogous to having a map with key landmarks and directions, allowing you to reconstruct any route without needing to remember every single turn. Furthermore, the Brownian Interval includes an LRU cache, similar to a short-term memory, to quickly access recently used parts of the path. This makes it significantly faster and more memory-efficient than previous methods like the Virtual Brownian Tree. In benchmarks, it has shown up to 10.6x faster query times while maintaining exactness in reconstructing the Brownian motion.

Next, we will dig into the reversible Heun method. **Heun method** is sort of an improved Euler's method. It is a novel numerical solver specifically designed for SDEs. What sets it apart is its algebraic reversibility, a property that allows for precise computation of gradients by carefully matching the truncation errors of the forward and backward passes during the solution process. This design minimizes numerical errors in gradient calculations.

Technical Implementation

Stock Market Prediction Using Neural SDE and Model Stacking

Our approach for predicting stock market movements was rooted in modeling the evolution of prices as a Brownian motion with drift and volatility. We formulated the market price as a Stochastic Differential Equation (SDE), where the drift term captured the underlying trend, while the diffusion term modeled the inherent random fluctuations in financial markets.

To achieve this, we developed a robust pipeline in Python, combining advanced deep learning techniques with a principled stochastic modeling approach. The key steps are outlined below, following the structure of the accompanying Jupyter Notebook:

Methodology

1. Data Preparation

- We fetched historical daily closing prices of the S&P 500 using financial libraries.
- Log returns were computed to quantify the percentage changes in prices, aligning with the SDE-based formulation.
- The log return data was converted into tensors for neural network compatibility.
- Scaling the returns to the [0,1] range ensured numerical stability during training.

2. Neural SDE Model Definition

- We defined a custom SDE class with the following components:
  - **Drift function** f: Captures the long-term trend.
  - **Diffusion function** g: Models the random noise.
- This formulation allowed the model to learn and represent the drift and volatility dynamics from historical data.

3. Training Configuration

- **Optimizer**: Adam optimizer initialized with a learning rate of 0.001.
- **Loss Function**: Mean Squared Error (MSE) to minimize the discrepancy between simulated and actual values.
- **Training Duration**: 3000 epochs with incremental timesteps for solving the SDE.
- **Solver**: Used an SDE solver to simulate paths of returns over time.

4. Training Process

- **Batch Sampling**: Multiple batches of paths were generated during training.
- **Loss Calculation**: The solver simulated paths of returns, and the predicted final states were compared against actual final states to compute the loss.
- **Checkpointing**: The best models were saved incrementally to ensure stable convergence.

5. Predictions and Testing

To enhance robustness and reduce variance, predictions were delivered as the average of multiple simulated paths. Specifically, for each input, we simulated 10 potential future paths of stock prices. The mean of the final values from these paths served as the final prediction, leveraging a bagging-like approach for improved stability and reduced bias.

a. Training Loss

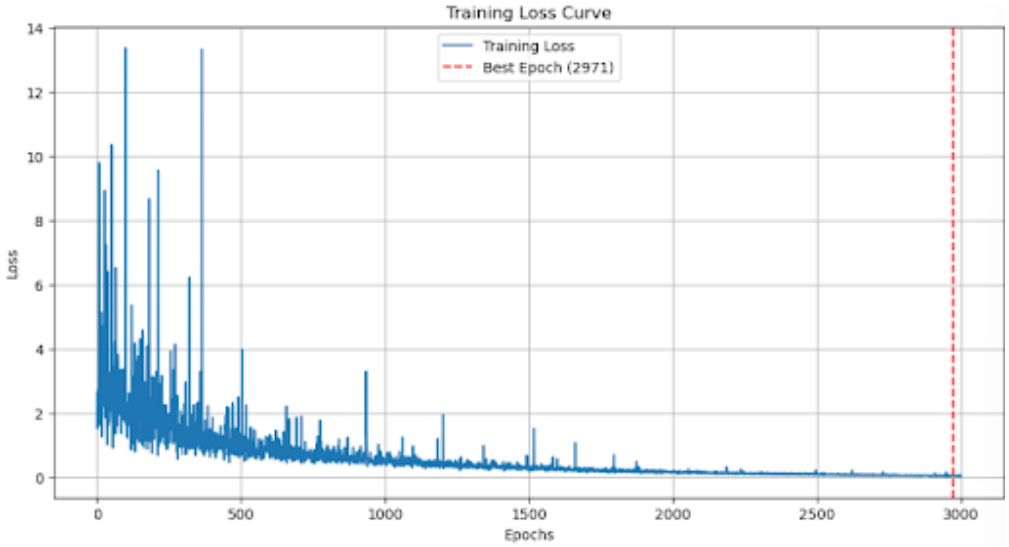

Figure 3: Binary Tree for Brownian Motion path

The training process effectively minimized the loss, demonstrating the model's ability to learn drift and volatility dynamics.

b. Predictions on Unseen Data

The model produced predictions that captured the general market trend but exhibited some degree of scaling error. While the averaged predictions improved robustness, residual noise remained.

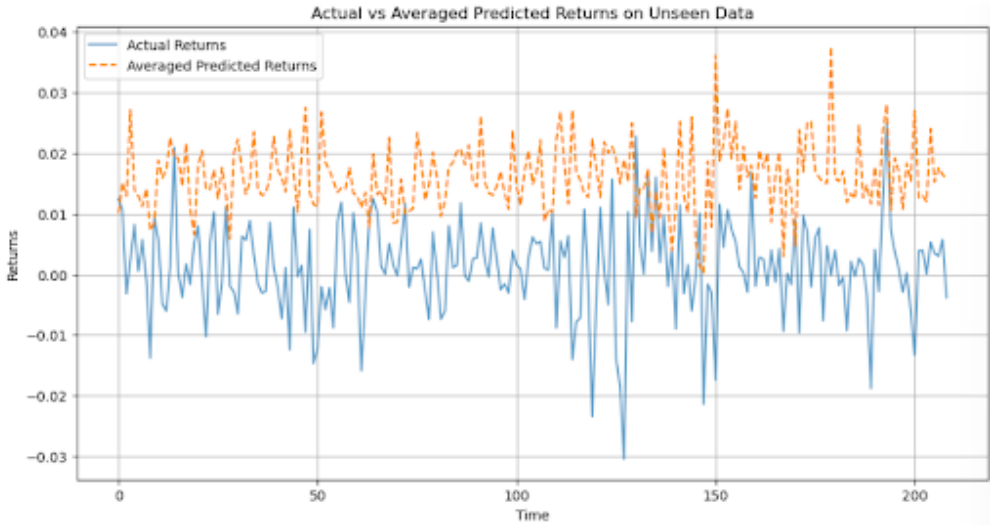

Figure 4: Binary Tree for Brownian Motion path

While the Neural SDE model showed promise, its predictions tended to decay over time, failing to capture the latest market dynamics. To address this limitation, we explored online learning as a natural progression.

a. Online Learning

In the online learning setup, the model was retrained step-by-step using the latest available data, allowing it to adapt to emerging trends. However, despite its conceptual appeal, this approach yielded only marginal improvements.

b. Model Stacking: Regression

Upon further analysis, we observed that scaling the model's predictions could align the forecasts more closely with the actual trends. This insight led us to implement a linear regression model to map the raw predictions to the actual returns:

- **Training**: The regression model's weights were learned on a separate validation set to prevent overfitting.
- **Final Predictions**: Predictions were evaluated on unseen data, achieving a notable improvement in accuracy.

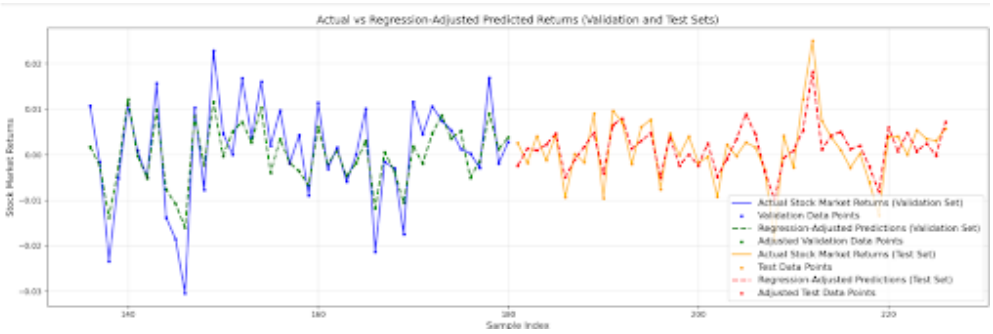

Figure 5: Binary Tree for Brownian Motion path

Drawbacks and Limitations

Solving a Neural SDE using the reversible Heun method involves sampling of Brownian Motion. While sampling Brownian motion is relatively straightforward, there arises some computational difficulties during the process. Ensuring reversibility requires the sampling of Brownian motion to be consistent in both the backwards and forwards passes. Additionally, during the backwards pass, it may be necessary to sample Brownian motion at points that were not sampled in the forward pass, adding further complexity. To address these challenges, techniques such as Levy Brownian Bridges and Brownian intervals are implemented, however their implementation can be challenging. These techniques rely on the use of specialized algorithms such as binary trees, LRU caches, and random seeds. Furthermore, as dimensionality increases or as time steps decrease, computational costs rise significantly.

Improvements

One improvement which can be implemented into the model to better predict movements in the S&P 500 would be to include a seasonal component within the SDE. This can be done by modifying the original SDE

in the following way:

$$dX_t = (\mu(t, X_t) + a\sin(zt) + b\cos(zt))dt + \sigma(t, X_t)dW_t$$

The seasonal components in the equation above, represented by sine and cosine, help to better model predictable oscillations, enhancing the SDE's ability to better capture the actual behaviour of the stock. Stock prices have a tendency to follow periodic movements to some extent which can be driven by market conditions, economic factors, and investor behaviour. Over longer periods of time, these seasonal trends are easier to observe. By the universal approximation theorem, neural networks are able to approximate periodic functions. However, by incorporating these seasonal components into the SDE, the neural network is able to adjust the weights accordingly for these periodic functions instead of approximating them. This allows for improved accuracy in the SDE's ability to forecast future stock prices such as the S&P 500.

Conclusion

In comparison to other traditional models, using Brownian intervals provides a faster and more efficient algorithm for predicting stock prices like the S&P 500. A key advantage to using Brownian intervals is its ability to only use a fixed amount of memory for the cache. This allows the algorithm to be more capable in handling larger time series datasets. The time complexity of a virtual Brownian motion tree is $O(\log(1/\varepsilon))$ where ε represents a low tolerance level. On the other hand, the time complexity of a Brownian interval is $O(\log(1/s))$, where $s > \varepsilon$. This slight variation allows for Brownian intervals to be more efficient when dealing with larger data sets. As a result, Brownian intervals provide a good starting point in predicting stock prices like the S&P 500.

References

[1] P. Gierjatowicz, M. Sabate-Vidales, D. Siska, L. Szpruch, and **Z. Zuric**, "Robust Pricing and Hedging via Neural SDEs," Jul. 08, 2020, *Social Science Research Network, Rochester, NY*: 3646241. doi: 10.2139/ssrn.3646241.

[2] P. Kidger, J. Foster, X. (Chen) Li, and T. Lyons, "Efficient and Accurate Gradients for Neural SDEs," in *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 2021, pp. 18747–18761. Accessed: Dec. 16, 2024. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2021/hash/9ba196c7a6e89eaf00954de80fc1b224-Abstract.html.

[3] P. Kidger, J. Foster, X. Li, and T. J. Lyons, "Neural SDEs as Infinite-Dimensional GANs," in *Proceedings of the 38th International Conference on Machine Learning*, PMLR, Jul. 2021, pp. 5453–5463. Accessed: Dec. 16, 2024. [Online]. Available: https://proceedings.mlr.press/v139/kidger21b.html.

Convergence Proof of the Reversible Heun Method

Theorem

Let $\{z_n\}$ denote the numerical solution to the Stratonovich SDE

$$dZ_t = \mu(t, Z_t)\, dt + \sigma(t, Z_t) \circ dW_t,$$

with initial condition $Z_0 = z_0$, obtained by the Reversible Heun method using a constant step size Δt. Assume that the functions μ and σ satisfy the following conditions:

1. **Lipschitz Continuity**: There exists a constant $L > 0$ such that for all $t \in [0, T]$ and all $z, z' \in \mathbb{R}^d$,

$$\|\mu(t, z) - \mu(t, z')\| + \|\sigma(t, z) - \sigma(t, z')\| \leq L\|z - z'\|.$$

2. **Boundedness**: There exists a constant $K > 0$ such that for all $t \in [0, T]$ and all $z \in \mathbb{R}^d$,

$$\|\mu(t, z)\| + \|\sigma(t, z)\| \leq K.$$

Then, there exists a constant $C > 0$ such that

$$\mathbb{E}\left[\|z_N - Z_T\|^2\right] \leq C\Delta t,$$

for sufficiently small Δt, where $N = T/\Delta t$.

Proof

We aim to prove that the expected mean-square error between the numerical solution z_N and the exact solution Z_T satisfies the inequality $\mathbb{E}\left[\|z_N - Z_T\|^2\right] \leq C\Delta t$.

Step 1: Define the Global Error

Let $e_n = z_n - Z_{t_n}$ denote the global error at time $t_n = n\Delta t$. Our goal is to derive a bound for $\mathbb{E}\left[\|e_n\|^2\right]$.

Step 2: Write the Numerical Scheme

The Reversible Heun method updates the numerical solution as follows:

1. **Predictor Step (Forward Step)**:

$$z_n^* = z_n + \mu_n \Delta t + \sigma_n \Delta W_n,$$

   where $\mu_n = \mu(t_n, z_n)$, $\sigma_n = \sigma(t_n, z_n)$, and $\Delta W_n = W_{t_{n+1}} - W_{t_n}$.

2. **Corrector Step (Symmetric Update)**:

$$z_{n+1} = z_n + \frac{1}{2}\left(\mu_n + \mu_{n+1}\right)\Delta t + \frac{1}{2}\left(\sigma_n + \sigma_{n+1}\right)\Delta W_n,$$

   where $\mu_{n+1} = \mu(t_{n+1}, z_{n+1})$, $\sigma_{n+1} = \sigma(t_{n+1}, z_{n+1})$.

Step 3: Express the Exact Solution Increment

The exact solution satisfies:

$$Z_{t_{n+1}} = Z_{t_n} + \int_{t_n}^{t_{n+1}} \mu(s, Z_s)\, ds + \int_{t_n}^{t_{n+1}} \sigma(s, Z_s) \circ dW_s.$$

We can approximate the integrals using the midpoint rule for small Δt:

$$\int_{t_n}^{t_{n+1}} \mu(s, Z_s)\, ds = \mu(t_n, Z_{t_n})\Delta t + R_n^{\mu},$$

$$\int_{t_n}^{t_{n+1}} \sigma(s, Z_s) \circ dW_s = \sigma(t_n, Z_{t_n})\Delta W_n + R_n^{\sigma},$$

where R_n^{μ} and R_n^{σ} are remainder terms satisfying $\|R_n^{\mu}\| \leq C\Delta t^2$ and $\mathbb{E}\left[\|R_n^{\sigma}\|^2\right] \leq C\Delta t^2$.

Step 4: Derive the Error Recursion

The global error can be written as:

$$e_{n+1} = e_n + (z_{n+1} - z_n) - \left(Z_{t_{n+1}} - Z_{t_n}\right).$$

Substituting the numerical and exact increments, we get:

$$\begin{aligned} e_{n+1} = e_n &+ \frac{1}{2}\left[\mu_n + \mu_{n+1}\right]\Delta t + \frac{1}{2}\left[\sigma_n + \sigma_{n+1}\right]\Delta W_n \\ &- \left[\mu(t_n, Z_{t_n})\Delta t + \sigma(t_n, Z_{t_n})\Delta W_n + R_n^{\mu} + R_n^{\sigma}\right]. \end{aligned}$$

Step 5: Simplify the Error Expression

Using the definitions of μ_n, σ_n, and the Lipschitz properties, we have:

$$\begin{aligned} e_{n+1} = e_n &+ \frac{1}{2}\left[\mu_n - \mu(t_n, Z_{t_n}) + \mu_{n+1} - \mu(t_n, Z_{t_n})\right]\Delta t \\ &+ \frac{1}{2}\left[\sigma_n - \sigma(t_n, Z_{t_n}) + \sigma_{n+1} - \sigma(t_n, Z_{t_n})\right]\Delta W_n - \left(R_n^{\mu} + R_n^{\sigma}\right). \end{aligned}$$

Let's denote the local truncation error as:

$$\delta_{n+1} = \frac{1}{2}\left[\mu_n - \mu(t_n, Z_{t_n}) + \mu_{n+1} - \mu(t_n, Z_{t_n})\right]\Delta t + \frac{1}{2}\left[\sigma_n - \sigma(t_n, Z_{t_n}) + \sigma_{n+1} - \sigma(t_n, Z_{t_n})\right]\Delta W_n - (R_n^{\mu} + R_n^{\sigma})$$

Thus,

$$e_{n+1} = e_n + \delta_{n+1}.$$

Step 6: Bounding the Local Truncation Error

Using the Lipschitz condition, we have:

$$\|\mu_n - \mu(t_n, Z_{t_n})\| \leq L\|e_n\|,$$

$$\|\mu_{n+1} - \mu(t_n, Z_{t_n})\| \leq L\left(\|e_{n+1}\| + K\Delta t\right),$$

since $z_{n+1} = z_n + \mathcal{O}(\Delta t)$.

Similarly for σ. Therefore,

$$\|\delta_{n+1}\| \leq C\left(\|e_n\| + \|e_{n+1}\|\right)\Delta t + C\Delta t^2.$$

However, since e_{n+1} depends on δ_{n+1}, we can rearrange terms to get:

$$\|\delta_{n+1}\| \leq C\|e_n\|\Delta t + C\Delta t^2.$$

Step 7: Bounding the Expected Error

Now, consider the squared norm of the error:

$$\|e_{n+1}\|^2 = \|e_n\|^2 + 2\langle e_n, \delta_{n+1}\rangle + \|\delta_{n+1}\|^2.$$

Taking expectations and using the Cauchy-Schwarz inequality:

$$\mathbb{E}\left[\|e_{n+1}\|^2\right] \leq \mathbb{E}\left[\|e_n\|^2\right] + 2\mathbb{E}\left[\|e_n\|\|\delta_{n+1}\|\right] + \mathbb{E}\left[\|\delta_{n+1}\|^2\right].$$

Using the bounds from Step 6:

$$\mathbb{E}\left[\|e_{n+1}\|^2\right] \leq \mathbb{E}\left[\|e_n\|^2\right] + 2C\Delta t\mathbb{E}\left[\|e_n\|^2\right] + C\Delta t^2\mathbb{E}\left[\|e_n\|\right] + C\Delta t^3.$$

Step 8: Applying Gronwall's Inequality

Assuming that $\mathbb{E}\left[\|e_n\|\right] \leq \left(\mathbb{E}\left[\|e_n\|^2\right]\right)^{1/2}$, we have:

$$\mathbb{E}\left[\|e_{n+1}\|^2\right] \leq (1 + 2C\Delta t)\,\mathbb{E}\left[\|e_n\|^2\right] + C\Delta t^3.$$

Applying this recursively and noting that $\mathbb{E}\left[\|e_0\|^2\right] = 0$, we get:

$$\mathbb{E}\left[\|e_N\|^2\right] \leq C\Delta t^2 \sum_{k=0}^{N-1} (1 + 2C\Delta t)^{N-k-1}.$$

For small Δt, we can approximate $(1 + 2C\Delta t)^N \approx e^{2CT}$. Therefore,

$$\mathbb{E}\left[\|e_N\|^2\right] \leq C\Delta t^2 \left(\frac{e^{2CT} - 1}{2C\Delta t}\right) = C'\Delta t,$$

where C' is a constant depending on T and C.

Conclusion

We have shown that:

$$\mathbb{E}\left[\|z_N - Z_T\|^2\right] = \mathbb{E}\left[\|e_N\|^2\right] \leq C'\Delta t.$$

Thus, the Reversible Heun method converges with strong order 0.5 for Stratonovich SDEs.