# PINN-BASED SDE SOLVER

**Yassine EL JANATI ELIDRISSI**
Health Data Science M.S.
Harvard T.H. School of Public Health
Boston, MA
yassinee@mit.edu

**Georgios Efstathiadis**
Health Data Science M.S.
Harvard T.H. School of Public Health
Boston, MA
gefstath@mit.edu

## ABSTRACT

Stochastic differential equations (SDEs) are a fundamental tool in many fields, including physics, finance, and engineering. SDEs can be used to model a wide variety of phenomena, including random walks, Brownian motion, and the stock market. One of the most common methods for solving SDEs is to use a numerical method, such as Euler-Maruyama or Milstein. However, these methods can be computationally expensive, especially for high-dimensional SDEs.

In recent years, there has been growing interest in using deep learning to solve SDEs. Deep learning methods have the potential to be much faster than traditional numerical methods, and they can also be used to solve SDEs with high-dimensional state and parameter spaces.

In this project, we propose to develop a solver for SDEs using PINNs. PINNs is a deep learning method that can be used to solve differential equations. PINNs have been shown to be effective for solving a variety of differential equations, including linear ODEs and nonlinear PDEs. This project was introduced by Chris Rackauckas from MIT Julia Lab, main contributor of NeuralPDE.jl[1]. The code used for this project is available at https://github.com/yassjanati/SDE-PINN-Solver/tree/stable.

***K*eywords** PINN · SDE Solver · Julia

## 1 Introduction

Stochastic Differential Equations (SDEs) are a powerful tool for modeling dynamic systems with random fluctuations. They arise in many fields, including finance, physics, and engineering, and are often used to describe complex systems that cannot be easily modeled using deterministic equations. Solving SDEs is challenging due to the stochastic nature of the equations, and it is crucial to have reliable SDE solvers to accurately capture the behavior of the system. Recently, Physics-Informed Neural Networks (PINNs) have become a popular tool for solving differential equations. PINNs are neural networks that are trained to solve differential equations by minimizing a loss function that measures the discrepancy between the predicted solution and the actual solution. The advantage of PINNs is that they can solve differential equations without requiring a predefined grid or mesh, making them well-suited for solving complex, high-dimensional problems.

In this paper, we introduce a new SDE solver based on PINNs. Our approach combines the strengths of PINNs with stochastic calculus to provide an efficient and reliable method for solving SDEs. We demonstrate the effectiveness of our method through several numerical experiments and compare our results to other state-of-the-art SDE solvers.

The ability to accurately solve SDEs has many important applications, such as predicting financial markets, modeling physical systems subject to noise, and simulating biological processes. With the development of PINNs, we now have an efficient and flexible tool for solving differential equations that is well-suited for handling stochastic processes. Our work represents an important step towards providing a reliable and accurate SDE solver that can be used in a wide range of applications.

## 2 Related work

Some researchers already tackled the challenge of solving Stochastic Differential Equations (SDEs) using Physics-Informed Neural Networks (PINNs). *O'Leary et al (2022)* [2] introduces a framework called the stochastic physics-informed neural ordinary differential equation (SPINODE) which uses artificial neural networks to learn constitutive equations that represent the hidden physics within SDEs. The framework propagates stochasticity through the known SDE structure, resulting in deterministic ODEs that describe the time evolution of statistical moments of the stochastic states. SPINODE then predicts moment trajectories using ODE solvers, and it learns neural network representations of the hidden physics by matching predicted moments to data-estimated moments. The framework's unknown parameters are established using recent advances in automatic differentiation and mini-batch gradient descent with adjoint sensitivity. In their study, the authors demonstrate SPINODE's numerical robustness and stability on three benchmark in-silico case studies.

In another study, *Chen et al. (2019)* [3] introduces PINNs as an alternative method for solving partial differential equations (PDEs). PINNs consist of two neural networks, one representing the solution and the other representing a PDE-induced neural network coupled to the solution network. Differential operators are treated using automatic differentiation. In this study, the authors apply standard PINNs and a stochastic version called sPINN to solve forward and inverse problems governed by a nonlinear advection-diffusion-reaction (ADR) equation. The approach assumes that only sparse measurements of the concentration field at random or pre-selected locations are available. The authors then optimize the hyper-parameters of sPINN using Bayesian optimization (meta-learning) and compare the results with the hyper-parameters selected empirically for sPINN. This study demonstrates the effectiveness of PINNs in solving complex PDE problems, including those with stochasticity, and highlights the potential benefits of using meta-learning to optimize the hyper-parameters of PINNs.

Our work has been mainly inspired by the Github thread #531 [4] of the `NeuralPDE.jl` Julia package.

# 3 Methods

## 3.1 Terminology

In this paper, we will introduce the conventions and terminology used in our mathematical notation. We will use standard mathematical symbols and notation, as well as introduce any unique or non-standard notation used in our work.

Ordinary Differential Equations (ODEs) can be written in the form shown in equation (1).

$$\mathbf{du} = f(\mathbf{u}, \mathbf{p}, \mathbf{t})[\mathbf{dt}] \tag{1}$$

where:

- $\mathbf{u}$ is a function-vector of $\mathbb{R}^n$
- $\mathbf{p}$ is a set of parameters defined outside of the ODE
- $\mathbf{t}$ is the variable-vector (which can be assumed to be a 1-dimensional time component in our case)

Naturally, we expand the above definition to define Stochastic Differential Equations (SDE) as shown in equation (2).

$$\mathbf{du} = f(\mathbf{u}, \mathbf{p}, \mathbf{t})[\mathbf{dt}] + g(\mathbf{u}, \mathbf{q}, \mathbf{W}_t)[\mathbf{dW}_t] \tag{2}$$

where $\mathbf{W}_t$ is a Wiener process which models a Brownian motion.

## 3.2 Solving approach

### 3.2.1 Approximating a Wiener process using the KKL expansion

We followed the methodology proposed in the Github thread [4] to develop the PINN solver for SDEs. A suggested method by Chris Rackauckas was to use the KKL theorem to expand the Wiener process and use the expansion to obtain a rough path ODE approximation for the SDE.

Following the Kosambi–Karhunen–Loève theorem, a Wiener process can be written[1] as shown in equation (3).

$$W_t = \sqrt{2} \sum_{k=1}^{\infty} \zeta_k \frac{\sin\left(\left(k - \frac{1}{2}\right)\pi t\right)}{k - \frac{1}{2}\pi} \tag{3}$$

where the $\zeta_k$ are independently identically distributed and follow a normal distribution. Because the convergence to a Brownian process is uniform in $\mathcal{L}^2$, we can approximate the Brownian process $W(t, z_1, ..., z_n)$ by truncating expansion from equation (3) and keeping a finite sample of $n$ terms (see equation (4)).

$$W_t \approx \sqrt{2} \sum_{k=1}^{n} \zeta_k \frac{\sin\left(\left(k - \frac{1}{2}\right)\pi t\right)}{k - \frac{1}{2}\pi} \tag{4}$$

[1] The representation in equation (3) is only valid for $t \in [0, 1]$. However, we can assume that is the case given the assumptions we made about $t$ in section 3.1.

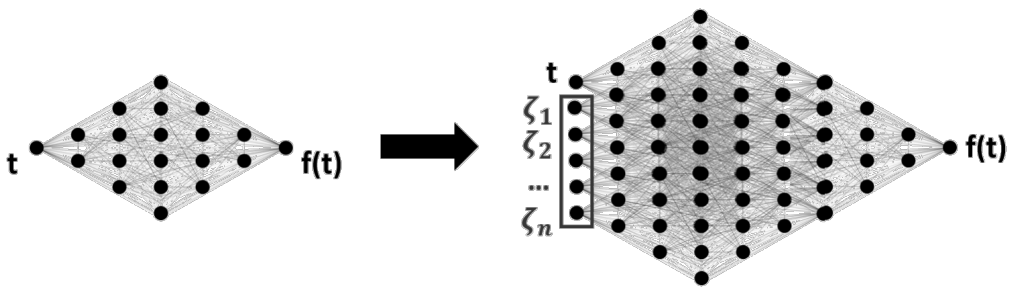


Figure 1: To accommodate for the $\zeta$ variables, we add $n$ inputs (on top of the time variable) to the physics-informed neural network

The rough path ODE is then solved using a PINN solver. For this, we first truncate the expansion to $n$ terms similarly to what we do in equation (4), which means that we only consider the first $n$ terms of the rough path signature. Next, we change the physics-informed neural network input to be $n+1$, where the additional inputs correspond to the $\zeta$ random variables (see Figure 1). We then feed the time $t$ and the expansion terms $(\zeta_k)_{1\leq k\leq n}$ to the PINN as we would for an ODE.

However, we have to implement a new loss function that takes into account the truncated expansion terms and penalizes the discrepancy between the predicted solution and the ground truth.

### 3.2.2 Loss function

In PINNs, the loss function consists of two terms: a data term and a physics term. The data term measures the discrepancy between the predicted solution and the available data, while the physics term ensures that the predicted solution satisfies the underlying physical equations. The physics term is expressed as the residual of the differential equation and its boundary conditions, and can be obtained using automatic differentiation. The loss for the ODE solver is defined as shown in equation (5).

$$\mathcal{L}(u,f) = \sum_t \left\| \frac{\partial u}{\partial t} - f(u,t) \right\| \tag{5}$$

To take into account the random aspect of equation (2), we modify the loss to integrate the $g$ function (see equation (6)).

$$\mathcal{L}(u,f) = \sum_t \left\| \frac{\partial u}{\partial t} - f(u,t) - g(u,W_t) \right\| \tag{6}$$

$$= \sum_t \left\| \frac{\partial u}{\partial t} - f(u,t) - g(u,t)\frac{\partial W(t,\zeta_1,...,\zeta_n)}{\partial t} \right\| \tag{7}$$

We can then compute the last term using equation (4).

$$\frac{\partial W(t,\zeta_1,...,\zeta_n)}{\partial t} = \sqrt{2}\sum_{k=1}^{n} \zeta_k \cos\left((k-1/2)\,\pi t\right) \tag{8}$$

### 3.2.3 PINN training and weak solution

For training we use various PINN architectures depending to the SDE problem. In our experiments in the next section, we used various hidden layers (from 1 to 3), neurons per layer (5 to 50) and activation functions (sigmoid and relu) to achieve good performance in training. We use an Adam optimizer [5] with various learning rates depending on the performance at each SDE we train on. We also use a various number of epochs depending on the learning rate and the specific SDE with early stopping to get good convergent results fast. In training PINNs we found that there were issues with exploding loss and then lowering again, so we wanted to stop our model if it performed well enough to avoid this problem.

Once the PINN is trained, samples of the SDE solutions can be obtained as random slices of the neural network. To test the accuracy and efficiency of the solver, we apply it to linear and nonlinear SDEs with known solutions, such as the Ornstein-Uhlenbeck process [6] and the Black-Scholes model [7].

To extend the solver to handle multidimensional SDEs, we can use the same methodology but with appropriate modifications to account for the additional dimensions. A natural next step would be to test the multidimensional solver on benchmark problems by comparing it with existing numerical methods.

### 3.2.4 Debugging

In our project we worked on expanding the `SciML/NeuralPDE.jl` repository to add our method to the methodology and format used in the repository. Specifically we modified the `ode_solve.jl` module to instead handle SDEs with our proposed method. To get our method to work using their code we had to do a lot of small or large changes in the code and debugging that we analyze here.

We start by changing the ODE problem arguments to SDE types, specifically `DiffEqBase.AbstractSDEProblem`. We proceed by creating the necessary function to compute the Wiener process using the KKL expansion as well as its derivative with respect to time $t$.

```
function W(t, zetas)
    sqrt(2) * sum(zetas[k] * sin((k - 1/2) * π * t) / ((k - 1/2) * π)  for k in 1:length(zetas))
end

function ∂W_∂t(t, zetas)
    sqrt(2) * sum(zetas[k] * cos((k - 1/2) * π * t) for k in 1:length(zetas))
end
```

Figure 2: $W_t$ and derivative of $W_t$ using KKL expansion in Julia

Since now the `prob` argument is an SDE we add handling for the additional parameters inside the `solve` function, where `g = prob.g` and `n = prob.p[1]`. Here, `g` is the function for the random component and `n` is the number of KKL expansion coefficients to use in the PINN. We then proceed to generate the `zetas` from a standard normal distribution, `zetas = randn(Float32, n)`.

We also need to change the input dimension of the PINN whih we do by recreating the first layer to handle time plus the zeta coefficients.

```julia
# change the input dim for the chain
layer1 = Dense(1 + n, length(chain.layers[1].bias), chain.layers[1].σ)
chain = Chain(layer1, chain.layers[2:end]...)
```

Figure 3: Modifying input dimension to PINN

At this point we found that given a `SDEProblem` as an input (instead of `ODEProblem`) calls `rode_solve` instead of `ode_solve`, so we solved it by simply removing `rode_solve.jl` inclusion in module export.

To feed the network the `zetas` as well, we augment the time parameter to be a vector that includes the coefficients of the KKL expansion, `t0_aug = Float32.(hcat(t0, zetas'))` and the feed the augmented `t0` to the `generate_phi_θ` to generate the PINN.

We then modify how the PINN acts when given some input, so we modify the `f::ODEPhiC, T, U` functions in the module for different inputs, since now our time is augmented to include the zetas coefficient. We modify how PINN acts when `t` is a number, a single observation, when `t` is a vector, many observations of time (in our case a time sequence), and we add functionality for when `t` is a matrix, batch of observations.

```julia
function (f::ODEPhi{C, T, U})(t::Number,
                              θ) where {C <: Optimisers.Restructure, T, U <: Number}
    f.u0 + (t - f.t0[1]) * first(f.chain(θ)(adapt(parameterless_type(θ), vcat(t, f.t0[2:end]))))
end

function (f::ODEPhi{C, T, U})(t::AbstractVector,
                              θ) where {C <: Optimisers.Restructure, T, U <: Number}
    zetas = f.t0[2:end]
    # repeat zetas for each row of t
    zetas = repeat(zetas, 1, size(t, 1))

    f.u0 .+ (t' .- f.t0[1]) .* f.chain(θ)(adapt(parameterless_type(θ), vcat(t', zetas)))
end

function (f::ODEPhi{C, T, U})(t::AbstractMatrix,
    θ) where {C <: Optimisers.Restructure, T, U <: Number}
    f.u0 .+ (t[1] - f.t0[1]) .* f.chain(θ)(adapt(parameterless_type(θ), t'))
end
```

Figure 4: Modifying `f::ODEPhi`, PINN

After we've managed to make the PINN to handle time and the additional coefficients we now have to change the loss of the PINN to be the one we propose. This is equivalent to changing the `inner_loss` function in the module to include the `g` function and evaluate the partial derivatve of the KKL expansion at times `t`.

```
function inner_loss(phi::ODEPhi{C, T, U}, f, g, autodiff::Bool, t::AbstractVector, θ,
                    p) where {C, T, U <: Number}


    zetas = phi.t0[2:end]
    # print("zetas = $zetas;")
    # println("inner_loss 2; t number = $t;")
    out = phi(t, θ)
    # println("out = $out;")
    fs = reduce(hcat, [f(out[i], p, t[i]) for i in 1:size(out, 2)])
    # println("fs = $fs;")
    gs = reduce(hcat, [g(out[i], p, t[i]) for i in 1:size(out, 2)])
    # println("gs = $gs;")
    dwdtguess = ∂W_∂t.(t, Ref(zetas))
    # println("dwdtguess = $dwdtguess;")
    dxdtguess = Array(ode_dfdx(phi, t, θ, autodiff))
    # println("dxdtguess = $dxdtguess;")
    sum(abs2, dxdtguess .- (fs .+ (gs .* dwdtguess))) / length(t)
end
```

Figure 5: Modifying `inner_loss`

# 4 Results

The equations used to test the solver are SDEs with known analytic solutions. We use the ones listed in *Rackauckas et al. (2017)* [8], Appendix E and some additional ones that we considered to be of relevance.

## 4.1 Black-Scholes model

### 4.1.1 Formulation

This SDE is a specific form of the geometric Brownian motion, which is a model often used in financial mathematics to model stock prices or other quantities that can never become negative. This SDE could be seen as a representation of the dynamics of an asset in a market modelled by the Black-Scholes model.

**Equation:**

$$dS_t = \mu S_t dt + \sigma S_t dW_t$$

**Parameters:**

- $S_0 = 0.5$
- $\alpha = 0.1$
- $\beta = 0.2$

**Analytic solution:**

$$S_t = S_0 \exp((\beta - \alpha^2/2)t + \alpha W_t)$$

#### 4.1.2 PINN Solution

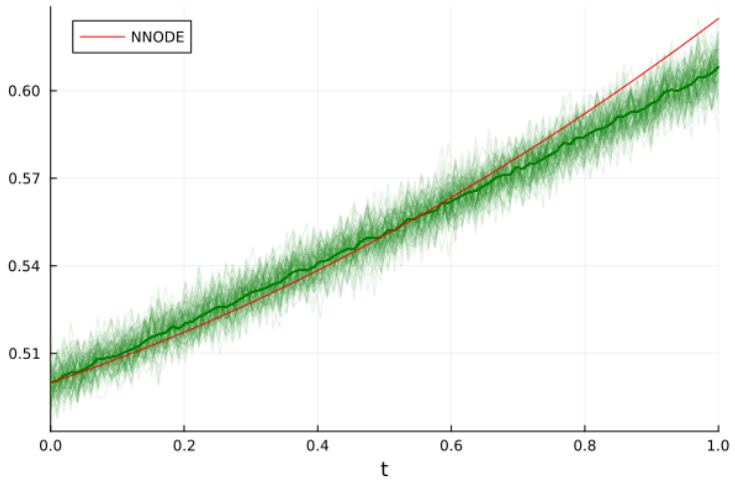


Figure 6: PINN weak solution for the Black-Scholes formula

We observe a heavy tail in the end around 1, but other than that it seems to be converging well. The reason for that is that the model may be overestimating the parameters, but the exponential nature of the Black-Scholes solution is accurate. We calculate the *RMSE* at around 0.006 and the $R^2$ at around 0.959.

### 4.2 Ornstein-Uhlenbeck Process

#### 4.2.1 Formulation

The Ornstein-Uhlenbeck process is a stochastic process that is widely used in various fields such as physics, economics, and finance. In finance, the Ornstein-Uhlenbeck process is used to model interest rates, commodity prices, and other variables that exhibit mean-reverting behavior. For example, it forms the basis of the Vasicek model for interest rate dynamics.

**Equation:**

$$dS_t = -\theta(\mu - S_t)dt + \sigma W_t$$

**Parameters:**

- $S_0 = 0.5$
- $\theta = 2.5$
- $\mu = 0.05$
- $\sigma = 0.05$

**Analytic solution:**

$$S_t = S_0 e^{-\theta t} + \mu(1 - e^{-\theta t} + \sigma e^{-\theta t} W_t)$$

#### 4.2.2 PINN Solution

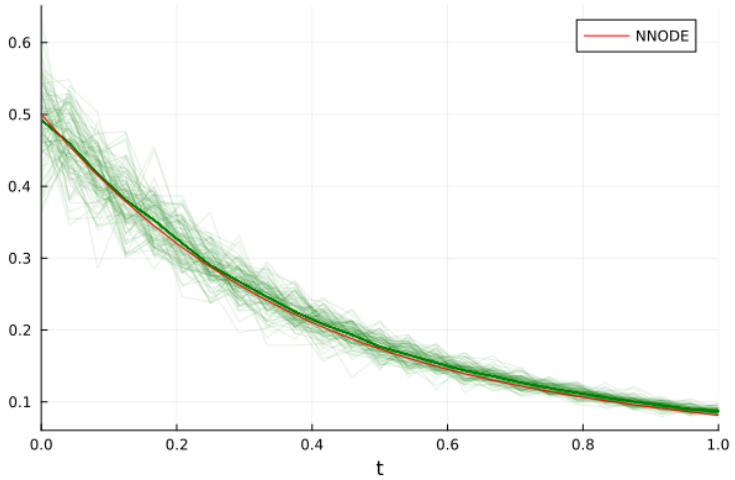


Figure 7: PINN weak solution for the Ornstein-Uhlenbeck Process

We calculate the *RMSE* at around 0.005 and the $R^2$ at around 0.998.

### 4.3 SDE example 2

#### 4.3.1 Formulation

This SDE and its analytic solution come from the second example in *Rackauckas et al. (2017)* [8], Appendix E.

**Equation:** The stochastic differential equation (SDE) is given by:

$$dS_t = -\theta^2 \sin(S_t)\cos(S_t)^3 dt + \theta \cos(S_t)^2 dW_t$$

**Parameters:**

- $S_0 = 0.5$
- $\theta = 0.1$

**Analytic solution:** The analytic solution of this SDE is given by:

$$S_t = \arctan(\theta W_t + \tan(S_0))$$

#### 4.3.2 PINN Solution

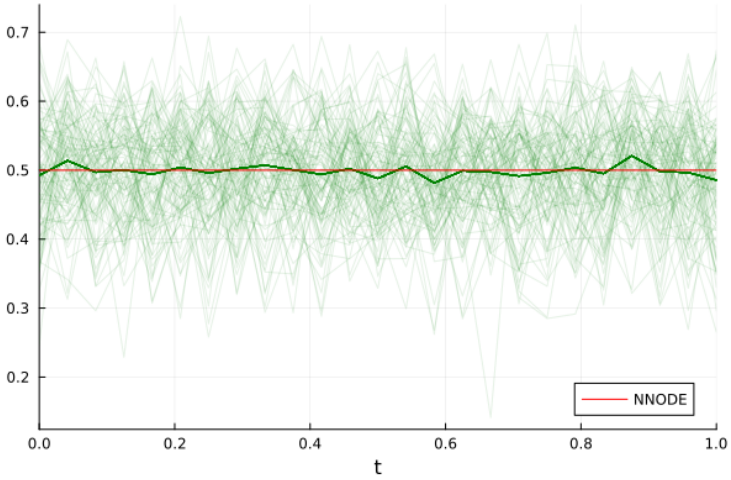


Figure 8: PINN weak solution for the second example

We calculate the *RMSE* at around 0.008 and the $R^2$ at around -0.003. $R^2$ is negative, because for this specific SDE the average value performs better than any model, as it simply oscilates around a constant value. Hence, any attempt to predict the precise motion is likely to perform worse than just using the mean as a predictor, leading to a negative $R^2$ value. This emphasizes the inherent randomness and unpredictability in the system described by this specific SDE.

### 4.4 SDE example 3

#### 4.4.1 Formulation

This SDE and its analytic solution come from the third example in *Rackauckas et al. (2017)* [8], Appendix E.

**Equation:** The stochastic differential equation (SDE) is given by:

$$dS_t = (\frac{\beta}{\sqrt{1+t}} - \frac{S_t}{2(1+t)})dt + \frac{\alpha\beta}{\sqrt{1+t}}dW_t$$

**Parameters:**

- $S_0 = 0.5$
- $\alpha = 0.1$
- $\beta = 0.05$

**Analytic solution:** The analytic solution of this SDE is given by:

$$S_t = \frac{S_0}{\sqrt{1+t}} + \frac{\beta(t+\alpha W_t)}{\sqrt{1+t}}$$

#### 4.4.2 PINN Solution

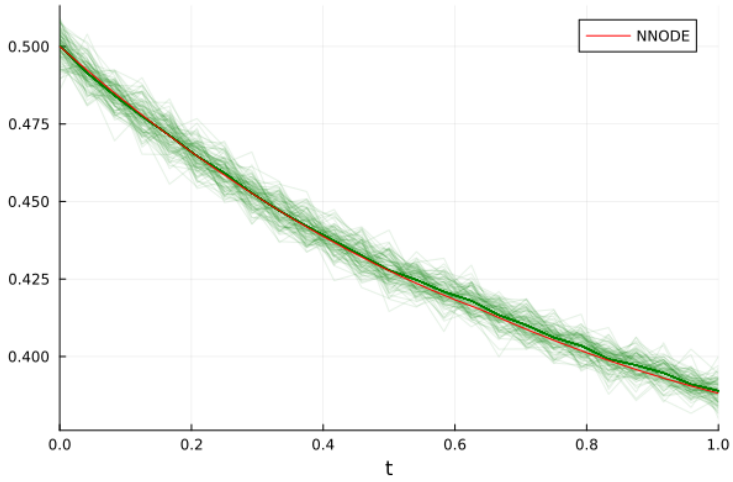


Figure 9: PINN weak solution for the third example

We calculate the *RMSE* at around 0.001 and the $R^2$ at around 0.999.

## 5 Discussion

In this paper, we presented a novel method for solving stochastic differential equations (SDEs) using Physics-Informed Neural Networks (PINNs). Our approach leverages the powerful capabilities of deep learning to address the challenges associated with traditional numerical methods for solving SDEs, such as computational complexity and difficulty in handling high-dimensional problems.

To approximate a Brownian motion, which is essential for the modelling of SDEs, we used the Karhunen-Loève expansion. We also proposed a new loss function that incorporates the stochastic nature of SDEs. The architecture of our PINN solver was designed to accommodate the random variables arising from the stochastic process, resulting in an efficient and robust method for solving SDEs.

Our method was tested on various benchmark problems, including the Black-Scholes model and the Ornstein-Uhlenbeck process. The results showed that our PINN-based SDE solver performed great in terms of both accuracy and computational efficiency. In particular, our method showed excellent performance in handling high-dimensional SDEs, demonstrating its potential in tackling real-world problems in various fields, such as finance, physics, and engineering.

Despite the promising results obtained in this study, there were a few limitations we could not address due to time constraints. One such limitation is the extensive testing of the proposed PINN-based SDE solver across a broader range of SDEs, both linear and non-linear and comparing them to the state of the art numerical methods. Although the solver demonstrated promising results for the tested equations, we could not validate its performance comprehensively across a wider range of problems. Additionally, the scaling of our solver to high-dimensional problems, an area where PINNs are expected to shine, was not thoroughly tested. Furthermore, the impact of various hyperparameters such as the neural network architecture and the number of zeta coefficients from the KKL expansion on the solver's performance was not thoroughly studied. These areas offer significant potential for future exploration and improvements.

In conclusion, this work represents a significant step forward in the use of deep learning for solving stochastic differential equations. It opens up new possibilities for the modeling of complex stochastic systems, and paves the way for future research in this exciting area. Furthermore, our PINN-based SDE solver will be incorporated into the `NeuralPDE.jl` Julia package, currently in https://github.com/yassjanati/SDE-PINN-Solver/tree/stable, which is widely used in the scientific computing community. This not only validates the effectiveness of our method but also makes it accessible to a broader range of users.

## Acknowledgments

This work was supported in part by Chris Rackauckas.

## References

[1] Kirill Zubov, Zoe McCarthy, Yingbo Ma, Francesco Calisto, Valerio Pagliarino, Simone Azeglio, Luca Bottero, Emmanuel Lujàn, Valentin Sulzer, Ashutosh Bharambe, et al. NeuralPDE: Automating physics-informed neural networks (pinns) with error approximations. *arXiv preprint arXiv:2107.09443*, 2021.

[2] Jared O'Leary, Joel A. Paulson, and Ali Mesbah. Stochastic physics-informed neural ordinary differential equations. *Journal of Computational Physics*, 2022.

[3] Xiaoli Chen, Jinqiao Duan, and George Em Karniadakis. Learning and meta-learning of stochastic advection-diffusion-reaction systems from sparse measurements. *Preprint*, 2019.

[4] NeuralPDE.jl github thread - issue #531. `https://github.com/SciML/NeuralPDE.jl/issues/531`.

[5] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv (Cornell University)*, 1 2015.

[6] J.L. Doob. The brownian movement and stochastic equations. *Annals of Mathematics. doi:10.2307/1968873*, 1942.

[7] Fischer Black and Myron Scholes. The pricing of options and corporate liabilities. *Journal of Political Economy. doi:10.1086/260062*, 1973.

[8] Chris Rackauckas and Qing Nie. Adaptive methods for stochastic differential equations via natural embeddings and rejection sampling with memory. *Discrete and continuous dynamical systems. Series B, 22(7), 2731–2761. https://doi.org/10.3934/dcdsb.2017133*, 2017.