Efficient Solutions of High Dimensional Semilinear Parabolic Partial Differential Equations with Neural Stochastic Differential Equations and Differentiable Programming

Abstract

Common stochastic models of financial assets and heterogeneous agents give rise to nonlinear partial differential equations (PDEs) such as nonlinear formulations of Black–Scholes and Hamilton–Jacobi–Bellman equations. The dimensionality of the PDEs is equal to the number of assets or agents, meaning that one can quickly arrive at realistic scenarios with PDEs of hundreds or thousands of spatial dimensions. In this manuscript, we develop a method to use neural stochastic differential equations for solving a class of high dimensional nonlinear parabolic partial differential equations. Through a differentiable programming approach, we demonstrate the ability to estimate the solution through a backwards stochastic differential equation (BSDE) by training a pair of neural networks. By directly utilizing the existing DifferentialEquations.jl library of stochastic differential equation (SDE) solvers, our approach efficiently handles many different scenarios by utilizing high weak order estimation, adaptive time stepping, handling of stiff models, etc. Together, this demonstrates that a differentiable programming approach with neural SDEs can provide an efficient yet general approach for solving high dimensional PDEs.

Preliminary work. Under review by AISTATS 2020. Do not distribute.

CJH: This papers reads to me like it could be a good submission to AISTATS 2020 (Palermo, Italy in June) whose CFP closes early Oct. Relevant topical areas: time series, deep learning, applications to finance. It's already longer than what would be expected as a workshop paper, and it doesn't quite fit the topics of Data, Fairness, Explainability, Trustworthiness, and Privacy that my workshop will be on. I've updated the paper to use the AISTATS template.

1 Introduction

High dimensional partial differential equations (PDEs) with terminating conditions form the foundation of many financial and macroeconomic models. For example, the fair valuation of a portfolio of European options with a fractional value upon default is given a nonlinear Black–Scholes equation. Similarly, the transition dynamics of a model of macroeconomic heterogeneity towards a new steady-state can be a high-dimensional PDE. In many financial scenarios, such as cases uncertain volatility or in cases with risk control, the solution to optimal pricing of stochastic financial assets is governed by a nonlinear PDE known—in particular, a Hamilton–Jacobi–Bellman (HJB) equation. In these applications, the dimensionality of the partial differential equation is the number of sources of heterogeneity for consumers in a macroeconomic model, the number of sources of shocks to a financial asset, the possible sources of revenue for an equity valuation problem, etc. However, high dimensional PDEs cannot be efficiently solved through traditional techniques, such as finite difference or finite element methods, since the memory required to store the state is often exponential in the size of the dimension. For example, consider if individuals or assets are differentiated by n characteristics. With a finite difference grid of $[0, 1]^n$ with spacing $\Delta x = 0.1$ in each direction has a size of 10^n points, making it impractical but all but small sizes of n while still being a very coarse approximation.

Thus to handle these cases, alternative approaches based on stochastic differential equations are utilized. The famous Feynman–Kac theorem (Kac, 1949; Feynman and Hibbs, 1965) demonstrates how the solution to a backwards diffusion-advection equation can be approximated using the expectation of a stochastic differential equation (Øksendal, 2003), giving a method with dimension-independent memory scaling. However, this theorem only applies to a set of quasilinear PDEs. Non-trivial degrees of heterogeneity and control, such as investment decisions by consumers or the endogenous choice of when to execute an option, lead to HJBEs and do not fall under the class of methods handled by this technique. Extensive literature exists on Backwards Stochastic Differential Equations (BSDEs), which give a more general set of estimatable parabolic PDEs (Pardoux, 1995; Pardoux and Tang, 1999; El Karoui et al., 1997; Gobet, 2016), but the development of efficient numerical schemes for the full class of equations has been difficult.

Groundbreaking work by Han et al. (2018) led to a method which utilized deep learning to estimate the non-analytic nature of terms in the resulting BSDEs, resulting in an efficient method for point estimates of semilinear partial differential equations. Their methodology utilized a neural network structure developed from a discretization of a stochastic differential equation which can be interpreted as a form of a deep latent Gaussian model

cite a bunch

. However, this architecture suffers from a few drawbacks:

1. It utilizes a separate neural network per time step, increasing the memory and computational costs required to solve the equation.
2. The neural network is derived from an Euler–Maruyama discretization of the forward stochastic differential equation. This is known to be an inefficient method (Kloeden and Platen, 1999), converging to the solution of the SDE with $\mathcal{O}(\sqrt{\Delta t})$.
3. Discretizing the stochastic differential equation into a deep latent Gaussian model results in a fixed time step discretization. In many cases, differential equations can be solved much more efficiently through adaptive time stepping (Hairer et al., 1993; Rackauckas and Nie, 2017a, 2018).
4. The explicit discretization that was chosen is unstable if the SDE is sufficiently stiff

   cite

   . To handle cases where the resulting SDE is stiff, intricate discretizations like (semi-)implicit schemes (Kloeden and Platen, 1999; Gilsing and Shardlow, 2007; Wang et al., 2012; Beyn et al., 2015; Jiang et al., 2016; Yao and Gan, 2018) and explicit-stabilized methods based on Chebyschev discretizations (Abdulle and Li, 2008; Abdulle and Cirilli, 2008; Abdulle et al., 2013, 2017; Komori and Burrage, 2012) need to be employed.

To combat these issues, we develop an efficient high dimensional partial differential equation solver using neural SDEs, an extension of neural or latent ordinary differential equations (Gao et al., 2008; Boker et al., 2004; Chen et al., 2018) to allow for stochasticity and were first showcased as an efficient computational method for automatic learning of dynamical models(Rackauckas et al., 2019)[1]. By utilizing the neural SDE formulation with differentiable programming (Innes et al., 2019) over the DifferentialEquations.jl numerical stochastic differential equation solvers (Rackauckas and Nie, 2017b), we demonstrate that the ability to make use of highly optimized, high order stiffly stable methods with adaptive time stepping to solve a resulting double neural network neural SDE to approximate solutions to the high dimensional partial differential equation.

2 Approximating High Dimensional Semilinear PDEs with Neural SDEs

We consider the class of semilinear parabolic PDEs, in finite time $t \in [0, T]$ and d-dimensional space $x \in \mathbb{R}^d$, that have the form

$$\begin{aligned}\frac{\partial u}{\partial t}(t,x) &+ \frac{1}{2}\operatorname{Tr}\left(\sigma\sigma^T(t,x)\left(\operatorname{Hess}_x u\right)(t,x)\right)\\ &+ \nabla u(t,x)\cdot\mu(t,x)\\ &+ f\left(t,x,u(t,x),\sigma^T(t,x)\nabla u(t,x)\right) = 0,\end{aligned} \tag{1}$$

with a terminal condition $u(T,x) = g(x)$. In this equation, Tr is the trace of a matrix, σ^T is the transpose of σ, ∇u is the gradient of u, and $\operatorname{Hess}_x u$ is the Hessian of u with respect to x. Furthermore, μ is a vector-valued function, σ is a $d \times d$ matrix-valued function and f is a nonlinear function. We assume that μ, σ, and f are known. We wish to find the solution at initial time, $t = 0$, at some starting point, $x = \zeta$.

Let W_t be a Brownian motion and take X_t to be the solution to the stochastic differential equation

$$dX_t = \mu(t, X_t)dt + \sigma(t, X_t)dW_t \tag{2}$$

with initial condition $X(0) = \zeta$. Previous work

[1]Follow-up studies have shown that neural stochastic differential equations are the limiting process of deep latent Gaussian models (Tzen and Raginsky, 2019; Peluchetti and Favaro, 2019) and may be more robust to generalizing than neural ODEs (Liu et al., 2019).

CJH: citation needed?

has shown that the solution to Equation 1 satisfies the following BSDE:

$$
\begin{aligned}
u(t,X_t) - u(0,\zeta) = \\
&- \int_0^t f(s, X_s, u(s, X_s), \sigma^T(s, X_s)\nabla u(s, X_s))ds \\
&+ \int_0^t \left[\nabla u(s, X_s)\right]^T \sigma(s, X_s) dW_s,
\end{aligned}
\tag{3}
$$

with terminating condition $g(X_T) = u(X_T, W_T)$. Notice that we can combine Equations 2 and 3 into a system of $d+1$ SDEs:

$$
\begin{aligned}
dX_t =& \mu(t, X_t)dt + \sigma(t, X_t)dW_t, \\
dU_t =& f(t, X_t, U_t, \sigma^T(t, X_t)\nabla u(t, X_t))dt \\
&+ \left[\sigma^T(t, X_t)\nabla u(t, X_t)\right]^T dW_t,
\end{aligned}
\tag{4}
$$

where $U_t = u(t, X_t)$. Since X_0, μ, σ, and f are known from the choice of model, the remaining unknown portions are the functional $\sigma^T(t, X_t)\nabla u(t, X_t)$ and initial condition $U(0) = u(0, \zeta)$, the latter being the point estimate solution to the PDE.

To solve this problem, we approximate both unknown quantities by universal function approximators, such as deep neural networks.

CJH: I reversed the order of the approximation equations

Introduce the approximations

$$
\begin{aligned}
\sigma^T(t, X_t)\nabla u(t, X_t) &\approx \mathrm{NN}_1(t, X_t|\theta_1), \\
u(0, X_0) &\approx \mathrm{NN}_2(X_0|\theta_2),
\end{aligned}
\tag{5}
$$

where NN_1 and NN_2 are two universal function approximators to the unknown functions parameterized by θ_1 and θ_2 respectively. Therefore we can rewrite Equation 4 as a system of SDEs with embedded neural networks, a mixed neural SDE, of the following form:

$$
\begin{aligned}
dX_t &= \mu(t, X_t)dt + \sigma(t, X_t)dW_t, \\
dU_t &= f(t, X_t, U_t, \mathrm{NN}_1(t, X_t))dt + \left[\mathrm{NN}_1(t, X_t)\right]^T dW_t,
\end{aligned}
\tag{6}
$$

with initial condition $(X_0, U_0) = (X_0, \mathrm{NN}_2(X_0))$. To be a solution of the PDE, the approximation must satisfy the terminating condition, and thus we define our loss to be the expected difference between the approximating solution and the required terminating condition:

$$
l(\theta_1, \theta_2|X_T, U_T) = \mathbb{E}\left[\|g(X_T) - U_T\|\right]. \tag{7}
$$

Finding the parameters (θ_1, θ_2) which minimize this loss function thus give rise to a BSDE which solves the PDE, and thus $\mathrm{NN}_2(X_0|\theta_2)$ is the solution to the PDE.

3 Training neural SDEs with Differentiable Programming

In order to efficiently handle the set of applications which can arise, it is essential to this approach that there are many choices for possible SDE integrators. For example, when μ has high separation of timescales, mixing fast and slow processes, a phenomena known as stiffness arises in the resulting stochastic differential equation which makes common explicit methods inefficient while implicit or stabilized methods become more efficient (Kloeden and Platen, 1999). Meanwhile, if stiffness is not present, then explicit methods are much more efficient since implicit methods and stabilized methods require extra computations like the inversion of a Jacobian in order to receive a larger yet stability region which is only useful in the stiff case. In addition, higher weak order methods can more efficiently converge to the expected value, allowing for less steps to more efficiently integrate the time interval to the requested tolerance. Likewise, adaptive timestepping has been shown to give many orders of efficiency gains in cases with multistable dynamical behavior (Rackauckas and Nie, 2018). Thus an efficient neural SDE system for high dimensional PDEs requires the ability to choose between many different SDE solver methods which are all highly optimized for different scenarios. For this we chose to utilize the DifferentialEquations.jl (Rackauckas and Nie, 2017b) library since it's the only open source library with optimized methods in many of these categories.

To train the neural stochastic differential equations, we utilize a differentiable programming approach to backpropagate through the stochastic differential equation solvers. Approaches which utilize forward sensitivities, like as derived in Tzen and Raginsky (2019), or done via forward-mode automatic differentiation like in Revels et al. (2016), were found to be inefficient since the computational cost scales with the number of parameters in the neural SDE. While an adjoint relationship has been derived for the solution of a stochastic differential equation via Malliavan calculus (Gobet and Munos, 2005), it is difficult to implement efficiently, especially when utilizing adaptive time stepping since the reverse calculation would require doing a search at each step for properly taking values from the Brownian bridge distribution for the reverse of W_t (Rackauckas and Nie, 2017a). These principles suggested that a differentiable programming approach based on reverse-mode automatic differentiation would be the most efficient method for calculating the gradient of the loss's expectation with respect to the neural network parameters. To achieve this end, we used the Tracker.jl (Innes, 2018b) tape-based and the Zygote.jl (Innes et al., 2019; Innes, 2018a) source-to-source automatic differentiation (AD) systems, which can perform reverse mode AD on Julia codes, of which the DifferentialEquations.jl SDE solvers are a pure-Julia implementation.

Thus the training loop proceeded as follows:

```
Data: Initial values of $\theta_1$ and $\theta_2$
Convergence criteria, e.g. $l(\theta_1, \theta_2) < l^*$ or maximum number of iterations reached.
Result: Trained parameters $\theta_1$ and $\theta_2$.
while not converged do
    Solve the initial value problem of Equation 6 using the current value of the parameters $\theta_1$ and $\theta_2$, obtaining the solution $(X_t, U_t)$ over $t \in [0, T]$.
    Calculate the loss $l(\theta_1, \theta_2 | X_T, U_T)$ of Equation 7 at the final time point $(X_T, U_T)$.
    Calculate derivatives of the loss function, $\partial l / \partial \theta_i$, by backpropagation using a reverse-mode automatic differentiation (AD) system.
    Choose the next set of parameters $(\theta_1, \theta_2)$ using gradient descent.
end
```

Algorithm 1: Training loop

4 Examples and Benchmarks

CJH: There is a unifying theme in these examples that can work if you are willing to frame it all in terms of finance applications. The theme is what to do when you have a choice between one risky investment that can fluctuate, and one risk-free investment that won't (like a savings account with guaranteed and fixed interest). The problem of how to hedge the risk when you can use a particular kind of option (European put/call) leads to Black-Scholes, which gives you the "correct" price for this financial instrument. The problem of how to allocate your funds to invest (how much of the risky and risk-free assets to buy and hold) to maximize expected value leads to Merton's portfolio problem, which is a particular kind of optimal control problem.
Black-Scholes is an important equation not just for it being a stochastic model for a market, but also the fact that it relates a few ideas that were not generally well appreciated to be equivalent: (i) geometric (log-) Brownian motion with deterministic drift is effectively the heat equation with drift and decay, (ii) conservation of energy in the heat equation is related to the existence of an equivalent martingale measure (EMM), i.e. the price fully bakes in all quantifiable risk, and therefore there no opportunity for arbitrage (create money by moving it around in a closed loop).
So perhaps you can tie 4.1 to 4.2 by saying that more explicitly, that the simple heat equation is a prelude to Black-Scholes by omitting drift, and even then you get some pretty "weird" issues like stiffness coming up.
Then you can transition from 4.2 to 4.3 because they are related problems in the same market setting (one cash account and one stock account effectively).

4.1 High Dimensional Stiff Semilinear Heat Equation

In this problem, the problem to solve is:

$$\frac{\partial u}{\partial t}(t,x) = \Delta u(t,x) - \alpha u(t,x), \tag{8}$$

with terminating condition $g(x) = \|x\|^2$. This PDE can be cast in the form of Equation 1 by setting:

$$\begin{aligned} \mu &= 0 \\ \sigma &= I \\ f &= -\alpha U_t \end{aligned} \tag{9}$$

with $t \in [0, T = 2]$ and $X_0 = (8, ..., 8) \in R^{100}$.

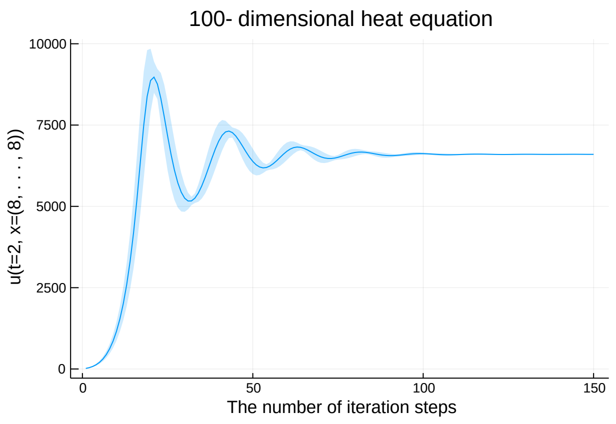

Figure 1: Caption

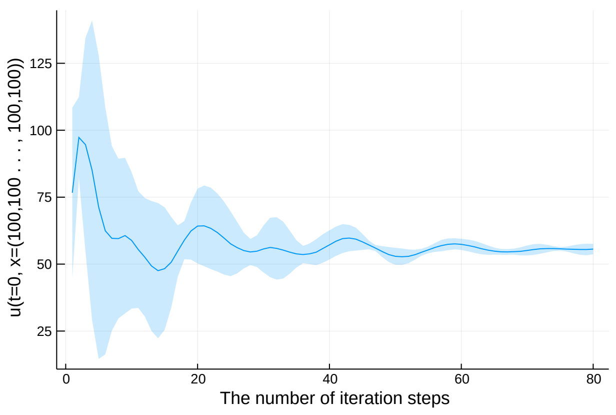

Figure 2: Caption

Thus the resulting SDE is a linear SDE with eigenvalues 0 and α. For sufficiently large α, this equation will be numerically stiff and thus explicit discretizations, such as the Euler–Maruyama discretization as chosen in Han et al. (2018), will not be efficient. To showcase this, we solved this problem with $\alpha = 100$ with both the Euler–Maruyama method and SROCKEM, an SROCK stabilization of the Euler–Maruyama method (Abdulle and Li, 2008). Figure 1 demonstrates that both of the numerical methods converge to the correct solution. However, with EM we required a $dt = ...$ in order to ensure stability, while SROCKEM was stable with $dt = ...$ Thus the former took nearly XXX seconds to solve the equation, whereas SROCKEM took XXX seconds. This highlights how the generality of the neural SDE approach allows for more efficient methods to be more readily employed.

KZ: Update the script to solve with EM and SROCKEM and get the benchmark values & numbers, along with benchmark times

4.2 Higher Order Stepping for Nonlinear Black–Scholes

Next consider the nonlinear Black–Scholes Equation with default risk.

$$\frac{\partial u}{\partial t}(t,x) + \bar{\mu}\cdot\nabla u(t,x) + \frac{\bar{\sigma}^2}{2}\sum_{i=1}^{d}|x_i|^2\frac{\partial^2 u}{\partial {x_i}^2}(t,x) \\ -(1-\delta)Q(u(t,x))u(t,x) - Ru(t,x) = 0 \tag{10}$$

with terminating condition $g(x) = \min_i x_i$ for $x = (x_1, ..., x_{100}) \in R^{100}$, where $\delta \in [0,1)$, R is the interest rate of the risk-free asset, and Q is a piecewise linear function of the current value with three regions ($v^h < v^l, \gamma^h > \gamma^l$),

$$Q(y) = \mathbb{1}_{(-\infty,v^h)}(y)\gamma^h + \mathbb{1}_{[v^l,\infty)}(y)\gamma^l \\ + \mathbb{1}_{[v^h,v^l]}(y)\left[\frac{(\gamma^h-\gamma^l)}{(v^h-v^l)}(y-v^h)+\gamma^h\right]. \tag{11}$$

This PDE can be cast into the form of Equation 1 by setting:

$$\begin{aligned} \mu &= \overline{\mu}X_t \\ \sigma &= \overline{\sigma}\,\mathrm{diag}(X_t) \\ f &= -(1-\delta)Q(u(t,x))u(t,x) - Ru(t,x) \end{aligned} \tag{12}$$

We choose following parameters for solve: $\delta = 2/3$, R = 0.02, $\overline{\mu} = 0.02$, $\overline{\sigma} = 0.2$, $v^h = 50$, $v^l = 70$, $\gamma^h = 0.2$, $\gamma^l = 0.02$, with $t \in [0,T]$, T = 1 and $X_0 = (100, ..., 100) \in R^{100}$.

To highlight an advantage of the neural SDE approach, we solve this equation using a higher order SDE integrator from the DifferentialEquations.jl library. Here we employ the Milstein method (Mil'shtejn, 1975). Because $\left[\sigma^T(t,X_t)\nabla u(t,X_t)\right]^T$ exists in the diffusion term, there is no way to guarantee that the noise term satisfies the commutativity property, and thus a general non-commutative noise approximation to the iterated integrals must be used. In normal uses of multidimensional SDEs, this iterated integral calculation is expensive, taking $\mathcal{O}(\sqrt{\Delta t})$ random numbers, even with the specialized tail correction due to Wiktorsson (2001); Gilsing and Shardlow (2007). However, the off-diagonal noise calculation does not require calculations of the drift and diffusion terms, and thus as the neural network grows in complexity this term stays constant. The total effect is a reduced number of neural network evaluations and thus greater efficiency. Figure 2 shows that the Milstein approximation is able to converge.

KZ: Solve with RKMil_General and EM to generate a convergence plot, and then solve RKMil_General and a higher time step (or adaptivity?) to show it's faster

4.3 Adaptive Time Stepping with Hamilton–Jacobi–Bellman

Next consider the classical linear-quadratic Gaussian (LQG) control problem in 100 dimensions

$$dX_t = 2\sqrt{\lambda}c_t dt + \sqrt{2}dW_t \tag{13}$$

with $t \in [0, T]$, $X_0 = x$, and with a cost function

$$C(c_t) = \mathbb{E}\left[\int_0^T \|c_t\|^2 dt + g(X_t)\right] \tag{14}$$

where X_t is the state we wish to control, λ is the strength of the control, and c_t is the control process. To minimize the control, the Hamilton–Jacobi–Bellman equation:

$$\frac{\partial u}{\partial t}(t,x) + \Delta u(t,x) - \lambda\|\nabla u(t,x)\|^2 = 0 \tag{15}$$

has a solution $u(t, x)$ which at $t = 0$ represents the optimal cost of starting from x. For an arbitrary g, one can show with Itô's formula that

$$u(t,x) = -\frac{1}{\lambda}\ln\left(\mathbb{E}\left[\exp\left(-\lambda g(x + \sqrt{2}W_{T-t}\right)\right]\right) \tag{16}$$

where $g(x) = u(0, X_0) = \ln(0.5(1 + \|x\|^2))$ is the terminal condition.

CJH: Initial condition? time $t = 0$ is probably not terminal. Or is this meant to be a solution that is periodic in time?

This PDE can be rewritten into the canonical form of Equation 1 by setting:

$$\begin{aligned} \mu &= 0, \\ \sigma &= \overline{\sigma} I, \\ f &= -\alpha \left\|\sigma^T(s, X_s)\nabla u(s, X_s))\right\|^2, \end{aligned} \tag{17}$$

where $\overline{\sigma} = \sqrt{2}$, T = 1 and $X_0 = (0, ..., 0) \in R^{100}$.

In previous methods, a user-chosen Δt was required in order to solve the equation. In this case, we will utilize the LambaEM method from DifferentialEquations.jl which utilizes the techniques from Lamba (2003); Rackauckas and Nie (2017a) to allow for adaptive time stepping with Euler–Maruyama. Figure 3 showcases that the automated form is able to accurately solve the equation.

KZ: Solve with LambaEM

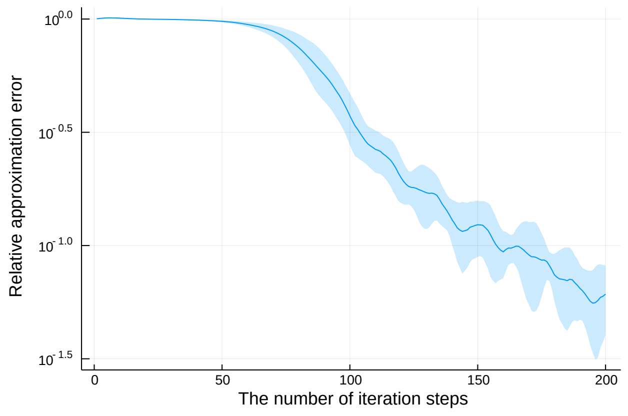

Figure 3: Caption

5 Discussion

In this manuscript we have developed a system for efficiently solving high dimensional partial differential equations through neural stochastic differential equations. By utilizing differentiable programming over the DifferentialEquations.jl library, we can effectively handle a large class of possible model choices all within one open source software implementation. While implementing each of these discretization choices specifically for the purpose of this manuscript would have been an astronomically infeasible amount of manpower, this technique has allowed us to directly compose with and utilize the develop efforts of an existing numerical SDE solver library. This technique highlights how ecosystem-wide differentiable programming which allows for direct use of existing packages may be a required element of scientific computing ecosystems in the near future in order to scale the complexity of the methods without having to continuously repeat development efforts. It's clear how other neural network based differential equation solver methods, such as the discrete physics-informed neural networks (PINNs) (Raissi et al., 2017), can be similarly generalized by utilizing neural differential equation architectures for the time stepping. This showcases how neural differential equation toolboxes could be a central tool for high efficiency deep learning enhanced physical simulators.

References

Abdulle, A., Almuslimani, I., and Vilmart, G. (2017). Optimal explicit stabilized integrator of weak order one for stiff and ergodic stochastic differential equations. arXiv:1707.08145 [math.NA].

Abdulle, A. and Cirilli, S. (2008). S-ROCK: Chebyshev methods for stiff stochastic differential equations. *SIAM Journal on Scientific Computing*, 30(2):997–1014.

Abdulle, A. and Li, T. (2008). S-ROCK methods for stiff Itô SDEs. *Communications in Mathematical Sciences*, 6(4):845–868.

Abdulle, A., Vilmart, G., and Zygalakis, K. (2013). Weak second order explicit stabilized methods for stiff stochastic differential equations. *SIAM Journal on Scientific Computing*, 35(4):1792–1814.

Beyn, W.-J., Isaak, E., and Kruse, R. (2015). Stochastic C-stability and B-consistency of explicit and implicit Milstein-type schemes. arXiv:1512.06905 [math.NA].

Boker, S. M., Neale, M. C., and Rausch, J. R. (2004). Latent differential equation modeling with multivariate multi-occasion indicators. In van Montfort, K., Oud, J., and Satorra, A., editors, *Recent Developments on Structural Equation Models*, volume 19 of *Mathematical Modelling: Theory and Applications*, pages 151–174, Dordrecht. Springer Science+Business Media.

Chen, R. T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. (2018). Neural ordinary differential equations. arXiv:1806.07366 [cs.LG].

El Karoui, N., Peng, S., and Quenez, M. C. (1997). Backward stochastic differential equations in finance. *Mathematical Finance*, 7(1):1–71.

Feynman, R. P. and Hibbs, A. R. (1965). *Quantum mechanics and path integrals*. International series in pure and applied physics. McGraw–Hill, New York.

Gao, P., Honkela, A., Rattray, M., and Lawrence, N. D. (2008). Gaussian process modelling of latent chemical species: applications to inferring transcription factor activities. *Bioinformatics*, 24(16):i70–i75.

Gilsing, H. and Shardlow, T. (2007). SDELab: A package for solving stochastic differential equations in MATLAB. *Journal of Computational and Applied Mathematics*, 205(2):1002–1018. Special issue on evolutionary problems.

Gobet, E. (2016). *Monte-Carlo Methods and Stochastic Processes: From Linear to Non-Linear*. Chapman & Hall/CRC.

Gobet, E. and Munos, R. (2005). Sensitivity analysis using Itô–Malliavin calculus and martingales, and application to stochastic optimal control. *SIAM Journal on Control and Optimization*, 43(5):1676–1713.

Hairer, E., Nørsett, S. P., and Wanner, G. (1993). *Solving Ordinary Differential Equations I: Nonstiff Problems*, volume 8 of *Springer Series in Computational Mathematics*. Springer-Verlag, Berlin, Heidelberg, 2 edition.

Han, J., Jentzen, A., and E, W. (2018). Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences*, 115(34):8505–8510.

Innes, M. (2018a). Don't unroll adjoint: Differentiating SSA-form programs. arXiv:1810.07951 [cs.PL].

Innes, M. (2018b). Flux: Elegant machine learning with Julia. *Journal of Open Source Software*.

Innes, M., Edelman, A., Fischer, K., Rackauckas, C., Saba, E., Shah, V. B., and Tebbutt, W. (2019). A differentiable programming system to bridge machine learning and scientific computing. arXiv:1907.07587 [cs.LG].

Jiang, F., Zong, X., Yue, C., and Huang, C. (2016). Double-implicit and split two-step Milstein schemes for stochastic differential equations. *International Journal of Computer Mathematics*, 93(12):1987–2011.

Kac, M. (1949). On distributions of certain Wiener functionals. *Transactions of the American Mathematical Society*, 65:1–13.

Kloeden, P. and Platen, E. (1999). *The Numerical Solution of Stochastic Differential Equations*, volume 23 of *Applications of Mathematics*. Springer-Verlag, Berlin Heidelberg.

Komori, Y. and Burrage, K. (2012). Weak second order S-ROCK methods for Stratonovich stochastic differential equations. *Journal of Computational and Applied Mathematics*, 236(11):2895–2908.

Lamba, H. (2003). An adaptive timestepping algorithm for stochastic differential equations. *Journal of Computational and Applied Mathematics*, 161(2):417–430.

Liu, X., Xiao, T., Si, S., Cao, Q., Kumar, S., and Hsieh, C.-J. (2019). Neural SDE: stabilizing neural ODE networks with stochastic noise. arXiv:1906.02355 [cs.LG].

Mil'shtejn, G. N. (1975). Approximate integration of stochastic differential equations. *Theory of Probability & Its Applications*, 19(3):557–562.

Øksendal, B. (2003). *Stochastic Differential Equations: An Introduction with Applications*. Hochschultext / Universitext. Springer.

Pardoux, E. (1995). Backward stochastic differential equations and applications. In Chatterji, S. D., editor, *Proceedings of the International Congress of Mathematicians*, pages 1502–1510, Basel. Birkhäuser.

Pardoux, E. and Tang, S. (1999). Forward-backward stochastic differential equations and quasilinear parabolic pdes.

Peluchetti, S. and Favaro, S. (2019). Neural stochastic differential equations. arXiv:1905.11065 [stat.ML].

Rackauckas, C., Innes, M., Ma, Y., Bettencourt, J., White, L., and Dixit, V. (2019). DiffEqFlux.jl

- a Julia library for neural differential equations. arXiv:1902.02376 [cs.LG].

Rackauckas, C. and Nie, Q. (2017a). 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.

Rackauckas, C. and Nie, Q. (2017b). DifferentialEquations.jl – a performant and feature-rich ecosystem for solving differential equations in Julia. *The Journal of Open Source Software*, 5(1).

Rackauckas, C. and Nie, Q. (2018). Stability-Optimized High Order Methods and Stiffness Detection for Pathwise Stiff Stochastic Differential Equations. arXiv:1804.04344 [math.NA].

Raissi, M., Perdikaris, P., and Karniadakis, G. E. (2017). Physics informed deep learning (Part I): Data-driven solutions of nonlinear partial differential equations. arXiv:1711.10561 [cs.AI].

Revels, J., Lubin, M., and Papamarkou, T. (2016). Forward-mode automatic differentiation in Julia. arXiv:1607.07892 [cs.MS]. Presented at the *AD2016 7th International Conference on Algorithmic Differentiation*.

Tzen, B. and Raginsky, M. (2019). Neural stochastic differential equations: Deep latent Gaussian models in the diffusion limit. arXiv:1905.09883 [cs.LG].

Wang, X., Gan, S., and Wang, D. (2012). A family of fully implicit Milstein methods for stiff stochastic differential equations with multiplicative noise. *BIT Numerical Mathematics*, 52:741–772.

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

Yao, J. and Gan, S. (2018). Stability of the drift-implicit and double-implicit Milstein schemes for nonlinear SDEs. *Applied Mathematics and Computation*, 339:294–301.