Partial Differential Equations in Applied Mathematics 6 (2022) 100457

Contents lists available at ScienceDirect

Partial Differential Equations in Applied Mathematics

journal homepage: www.elsevier.com/locate/padiff

A test of backward stochastic differential equations solver for solving semilinear parabolic differential equations in 1D and 2D[☆]

Evan Davis[a], Guangming Yao[b,*], Elizabeth Javor[c], Kalani Rubasinghe[b], Luis Antonio Topete Galván[d]

[a] Department of Mathematics, University of Wisconsin, Madison, 53706, WI, USA
[b] Department of Mathematics, Clarkson University, 8 Clarkson Ave, Potsdam, 13699, NY, USA
[c] Department of Mathematics, Rochester Institute of Technology, Rochester, 14623-5600, NY, USA
[d] Universidad Autónoma del Estado de Hidalgo, Kilómetro 4.5 carretera Pachuca-Hidalgo, Pachuca, Hidalgo, 42074, Mexico

ARTICLE INFO

Keywords:
Semilinear PDE
Backward stochastic PDE
Euler method
Deep learning
Parabolic
Black–scholes equations
Allen–Cahn equations
Diffusion–reaction equations
Radial basis functions
Picard method

ABSTRACT

Backward stochastic differential equation solver was first introduced by Han et al in 2017. A semilinear parabolic partial differential equation is converted into a stochastic differential equation, and then solved by the backward stochastic differential equation (BSDE) solver. The BSDE solver uses the backward Euler scheme for time discretization and stochastic process and neural network to learn the derivative functions at each time step. The algorithm is ideal for solving high-dimensional PDEs, especially in dimensions higher than 3-dimension problems, whereas the traditional numerical techniques fail to produce any simulations. We modified the BSDE solver so that is works for one-dimensional problems as well. The focus of this paper is to understand how the BSDE solver works in comparison with the traditional numerical techniques in low dimensional spaces (1D and 2D). We examined the BSDE solver in terms of accuracy, efficiency and convergence. Through five classical differential equations, we discovered that the solver works for low dimensional spaces, as accurate as it can be in high dimensional spaces. It is more accurate than the radial basis function collocation method reported in literature and the results by the Picard method. However, the BSDE solver is time consuming. This however, can be solved by parallel computing if needed.

1. Introduction

Partial differential equations (PDEs) are among the fundamental tools used to model physical phenomena in sciences and engineering.[1] Traditional analytical or numerical methods are sometimes inadequate to use in realistic problems due to increase in dimensions and complexity.[2–5] Additionally, numerical stochastic differential equations (SDE)[6–9] are very successful for modeling stochastic phenomena such as those in finance, biology, mechanical engineering, population and etc. Methods for solving SDE including finite difference method,[10,11] finite element method,[12] meshfree methods,[13] Bernoulli wavelet method,[14] hybrid methods,[15,16] and many others. Furthermore, over the last decade, deep learning algorithms have gained a lot of interests in dealing with high-dimensional problems in many fields. It has effectively been used in solving problems with complicated geometries,[17] many-electron Schrodinger equations,[18] high-dimensional forward–backward stochastic differential equations,[19] computational fluid dynamics,[20] and etc.

In this paper, we are interested in examining the *semilinear parabolic partial differential equations (PDEs)* of the following form:

$$\frac{\partial u}{\partial t}(t,x) + \frac{1}{2}\mathrm{Tr}\left[\sigma\sigma^T(t,x)\mathbf{H}_x u(t,x)\right] + \nabla u(t,x)\cdot\mu(t,x) + f(t,x,u,\sigma^T\nabla u) = 0 \tag{1.1}$$

where $x \in \mathbb{R}^d, t \in \mathbb{R}$, subject to terminal condition

$$u(T,x) = g(x), \tag{1.2}$$

where the function $g(x)$ is a given real-valued function defined on $\mathbb{R}^d$. Note that Tr denotes the trace of matrices, ∇u is the gradient, $\mathbf{H}_x u(t,x)$ is the Hessian matrix, $\sigma(t,x) : \mathbb{R}\times\mathbb{R}^d \to M^{d\times d}(\mathbb{R})$ is a known matrix-valued function, $\mu(t,x) : \mathbb{R}\times\mathbb{R}^d \to \mathbb{R}^d$ is a known vector-valued function, and f is a known nonlinear function. The goal is to obtain $u(0,\xi)$ for some fixed $\xi \in \mathbb{R}^d$.

Backward stochastic differential equation solver was first introduced in Ref. 21 in 2017 to solve such semilinear parabolic PDEs in high-dimensional spaces as high as hundreds. Instead of setting up

☆ This work was supported in part by National Security Agency, USA grant H98230-21-1-0336 and H89230-22-1-0008.
* Corresponding author.
E-mail address: gyao@clarkson.edu (G. Yao).

https://doi.org/10.1016/j.padiff.2022.100457
Received 23 January 2022; Received in revised form 12 June 2022; Accepted 19 October 2022

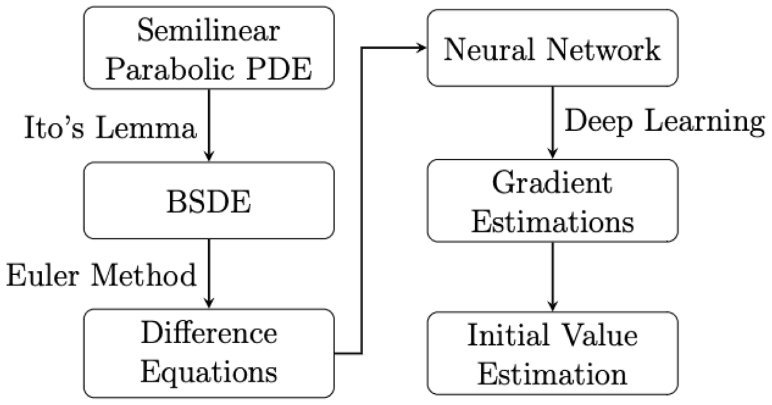

Fig. 1. Flowchart of the BSDE solver algorithm for solving semilinear parabolic PDEs.

initial value problems, as is commonly done in traditional numerical methods literature, problem with terminal conditions is considered since it enables to make connections with BSDEs. To be able to handle such high-dimension, the PDEs are converted into SDEs, and then solved by the backward stochastic differential equation (BSDE) solver. The BSDE solver uses the backward Euler scheme for time discretization and stochastic process and neural network to learn the derivative functions at each time step. Fig. 1 illustrates the steps involved in the BSDE solver. The solver is programmed in PYTHON using TENSORFLOW by Han et al. in 2017, which can be downloaded from GitHub.

The focus of our paper is to examine how the BSDE solver works in low-dimensional spaces (1D, 2D) and compare its performance with traditional numerical techniques. The original BSDE solver found to be faulty for 1D problems. We modified the solver so it works for 1D problems. These numerical experiments include classical benchmark differential equation problems, such as 1D and 2D heat equation with a terminal condition, a diffusion–reaction equation in 2D and higher dimensions up to 100 dimensions, nonlinear Black–Scholes equations in 2D, and an Allen–Cahn Equation in 2D. We discovered that the solver works for low dimensional spaces, as accurate as it can be in high dimensional spaces. As the solver can only approximate solution at a single point at a time makes it inefficient when we are interested in solutions in a region. Note that authors claimed that the solver can be improved to solve PDEs over a region. Furthermore, the method is analyzed and showed to be convergences for high-dimensional forward–backward stochastic differential equations in Ref. 19.

2. Itô's Lemma and the BSDE

For completeness, we will briefly review the algorithm used in the BSDE solver in this section. First, we need to define an Itô process. An Itô process is a type of stochastic process described by Kiyoshi Itô. It is expressed as the sum of the integral of a process over time and the integral of another process over a Brownian motion. Those processes are the base of Stochastic integration, which therefore are widely used in stochastic calculus.

Definition 1. For a vector $\mu \in L^1$ and a matrix $\sigma \in L^2$, $X(t)$ is an Itô process[22] if

$$X(t) = X_0 + \int_0^t \mu ds + \int_0^r \sigma dW. \tag{2.1}$$

This implies that

$$dX_t = \mu dt + \sigma dW_t. \tag{2.2}$$

Definition 2. Let $\{W_t\}_{t\in[0,T]}$ be a d-dimensional Brownian motion. We define the d-dimensional stochastic processes, $\{X_t\}_{t\in[0,T]}, \{Y_t\}_{t\in[0,T]}$, $\{Z_t\}_{t\in[0,T]}$ as follows[23]:

$$X_t = \xi + \int_0^t \mu ds + \int_0^t \sigma dW, \tag{2.3}$$

$$Y_t = g(X_T) + \int_t^T f(s, X_s, Y_s, Z_s) ds - \int_t^T (Z_t)^T dW_s \tag{2.4}$$

$$Z_t = \sigma^T(t, X_t)\nabla u(t, X_t). \tag{2.5}$$

Theorem 1 (*Itô's Lemma*[22]).
For a vector μ and matrix σ, let $X_t = (X_t^1, X_t^2, \dots, X_t^d)^T$ be a vector of Itô processes such that $dX_t = \mu\, dt + \sigma\, dW_t$. Then

$$\begin{aligned} du(t, X_t) &= \frac{\partial u}{\partial t} dt + (\nabla u)^T\, dX_t + \frac{1}{2} (dX_t)^T (\mathbf{H}_x u)\, dX_t, \\ &= \left\{ \frac{\partial u}{\partial t} + (\nabla u)^T \mu + \frac{1}{2} \operatorname{Tr}\left[\sigma^T \sigma \left(\mathbf{H}_x u\right)\right] \right\} dt + (\nabla u)^T \sigma\, dW_t \end{aligned} \tag{2.6}$$

where ∇u is the gradient of u w.r.t. x, $\mathbf{H}_x u$ is the Hessian matrix of u w.r.t. x, and Tr is the trace operator.

If $u(t,x)$ be any twice differentiable scalar function of two real variables t and x, for an Itô drift–diffusion process

$$dX_t = \mu\, dt + \sigma\, dW_t, \tag{2.7}$$

one has

$$du(t, X_t) = \left(\frac{\partial u}{\partial t} + \mu \frac{\partial u}{\partial x} + \frac{\sigma^2}{2} \frac{\partial^2 u}{\partial x^2} \right) dt + \sigma \frac{\partial u}{\partial x} dW_t. \tag{2.8}$$

This immediately implies that $u(t, X_t)$ is itself an Itô drift–diffusion process. We seek to show that the solution u to the semilinear parabolic PDEs (1.1)–(1.2) can lead us to a solution of a stochastic differential equation (SDE) and vise versa.

Theorem 2. *The semilinear parabolic PDEs (1.1)–(1.2) has a solution $u(t,x)$ if and only if $u(t, X_t)$ satisfies the following Backward SDE (BSDE)*

$$\begin{aligned} &u(t, X_t) - u(0, X_0) \\ &= - \int_0^t f(s, X_s, u(s, X_s), \sigma^T(s, X_s)\nabla u(s, X_s)) ds \\ &\quad + \int_0^t [\nabla u(s, X_s)]^T \sigma(s, X_s) dW_s. \end{aligned} \tag{2.9}$$

where X_t is defined in (2.3).

Proof. For our simplicity, we rewrite (2.9) as follows:

$$u(t, X_t) - u(0, X_0) = - \int_0^t f ds + \int_0^t [\nabla u(s, X_s)]^T \sigma(s, X_s) dW_s. \tag{2.10}$$

We can also reformulate (2.4) as follows:

$$dY_t = -f(t, X_t, Y_t, Z_t) dt + (Z_s)^T dW \tag{2.11}$$

In addition, by Itô's Lemma, we have that

$$d(u(t,X_t)) = \left\{u_t + \nabla u\cdot\mu + \frac{1}{2}Tr(\sigma\sigma^T\mathbf{H}_x u)\right\}dt + [\nabla u]^T\sigma dW. \tag{2.12}$$

- If $u(t,x)$ is a solution to the semilinear parabolic PDE (1.1)–(1.2), by Eq. (1.1), $u_t + \nabla u\cdot\mu + \frac{1}{2}Tr(\sigma\sigma^T\mathbf{H}_x u) = -f(t,X_t,Y_t,Z_t)$. Thus,

$$d(u(t,X_t)) = -f(t,X_t,Y_t,Z_t)dt + [\nabla u]^T\sigma dW. \tag{2.13}$$

Note the definitions of Y_t and Z_t by Eqs. (2.4) and (2.5) or (2.11),

$$d(u(t,X_t)) = -f(t,X_t,Y_t,Z_t)dt + (Z_t)^T dW = dY_t, \tag{2.14}$$

Thus,

$$\begin{aligned} u(t,X_t) = Y_t &= g(X_T) + \int_t^T f(s,X_s,Y_s,Z_s)ds - \int_t^T (Z_t)^T dW_s,\\ u(0,X_0) = Y_0 &= g(X_T) + \int_0^T f(s,X_s,Y_s,Z_s)ds - \int_0^T (Z_t)^T dW_s. \end{aligned} \tag{2.15}$$

Therefore,

$$\begin{aligned} &u(t,X_t) - u(0,X_0)\\ &= -\int_0^t f(s,X_s,u(s,X_s),\sigma^T(s,X_s)\nabla u(s,X_s))ds\\ &\quad + \int_0^t [\nabla u(s,X_s)]^T\sigma(s,X_s)dW_s\\ &= -\int_0^t f ds + \int_0^t (Z_t)^T dW_s \end{aligned}$$

This is the BSDE (2.9).

- If $u(t,X_t)$ is a solution of the BSDE (2.9), then

$$\begin{aligned} u(t,X_t) &= u(0,X_0) - \int_0^t f ds + \int_0^t (Z_t)^T dW_s,\\ du(t,X_t) &= -f dt + (Z_t)^T dW. \end{aligned} \tag{2.16}$$

Thus, by Eq. (2.4) we have that

$$\begin{aligned} u(T,X_t) &= u(0,X_0) - \int_0^T f ds + \int_0^T (Z_t)^T dW_s\\ &= u(0,X_0) + g(X_T) - Y_0 = g(X_T). \end{aligned} \tag{2.17}$$

Thus, $u(T,x) = g(x)$. On the other hand, recall Itô's lemma, combining Eqs. (2.12) and (2.16), we have that

$$\begin{aligned} &\left\{u_t + \nabla u\cdot\mu + \frac{1}{2}Tr(\sigma\sigma^T\mathbf{H}_x u)\right\}dt + [\nabla u]^T\sigma dW = -f dt + (Z_t)^T dW,\\ &u_t + \nabla u\cdot\mu + \frac{1}{2}Tr(\sigma\sigma^T\mathbf{H}_x u) = -f,\\ &u_t + \nabla u\cdot\mu + \frac{1}{2}Tr(\sigma\sigma^T\mathbf{H}_x u) + f = 0. \end{aligned} \tag{2.18}$$

Thus, we have a solution to the PDE (1.1)–(1.2), $u(t,x)$. □

3. Numerical solutions to the BSDE

The BSDE solver buit by Han et al. in Ref. 18 uses a simple explicit Euler scheme to discretize the time space for the BSDE, and then use the deep learning method to approximate derivatives in spatial variables during each time step. In this section, we briefly introduce the numerical algorithm.

3.1. Euler's method for time discretization

First, we will apply a temporal discretization using the explicit Euler scheme on (2.3)–(2.5). Let $\Delta t_n = t_{n+1} - t_n$, and $\Delta W_n = W_{t_{n+1}} - W_{t_n}$, $n = 0,1,\dots,N$, and $t_0 = 0, t_N = T$. With the time discretization, Eq. (2.3) can be discretized as

$$X_{t_{n+1}} - X_{t_n} \approx \mu(t_n, X_{t_n})\Delta t_n + \sigma(t_n, X_{t_n})\Delta W_n, \tag{3.1}$$

The choices in Eq. (3.1) is the explicit Euler's method. The local truncation error by such time discretization is approximately proportional to the square of time step size. We chose to use small time steps to ensure accuracy in the numerical experiments, so the effect of neural network training on the derivative approximations can be studied. With the time discretization, (2.9) becomes

$$\begin{aligned} u(t_{n+1}, X_{t_{n+1}}) - u(t_n, X_{t_n}) \approx &-f(t_n, X_{t_n}, u(t_n, X_{t_n}), \sigma^T(t_n, X_{t_n}))\nabla u(t_n, X_{t_n})\Delta t_n\\ &+[\nabla u(t_n, X_{t_n})]^T\sigma(t_n, X_{t_n})\Delta W_n. \end{aligned} \tag{3.2}$$

Given this temporal discretization, the path X_{t_n}, $0 \le n \le N$ can be easily sampled using (3.1).

3.2. Neural network for spatial derivative approximations

Next key step is to approximate the function

$$x \mapsto \sigma^T(t,x)\nabla u(t,x) \tag{3.3}$$

at each time step $t = t_n$ by a multilayer feedforward network under the assumption that $u(t,x)$ is know by given terminal condition. Denote

$$(\sigma^T\nabla u)(t_n, X_{t_n}) \approx (\sigma^T\nabla u)(t_n, X_{t_n}; \theta_n)$$

for $n = 1,\dots,N-1$, and θ_n refers to neural network parameter. The loss function is defined based on the squared approximation error associated to the terminal condition of the BSDE:

$$l(\theta) = \mathbb{E}[|g(X_{t_N}) - u(t_N, X_{t_N})|^2] \tag{3.4}$$

Optimization methods such as Stochastic Gradient Descent (SGD) algorithm or Adam optimizer[24] can be used to minimize $l(\theta)$ over θ. Once the maximum number of iterations has occurred, a final estimate of the initial value is obtained. We refer readers to[21] for detailed description on the training of the neural networks in the algorithm.

The traditional numerical methods represent functions using polynomials or other basis functions, leads to high complexity. In contrast, the BSDE algorithm uses compositions of simple functions in the neural network which leads to less computational cost. This made the numerical approximation to higher dimensional PDEs possible whereas the traditional numerical techniques unable to produce any solution within reasonable time and storage spaces in computers.

4. Numerical experiments

In this section, we extended the solver so that it works for one-dimensional problems. Additionally, a comparison of the BSDE solver with traditional numerical techniques in low dimensional spaces is presented through five classical differential equations benchmark problems. There are 1D and 2D heat equation with a terminal condition, a diffusion–reaction equation in 2D and higher dimensions up to 100 dimensions, nonlinear Black–Scholes equations in 2D, and an Allen–Cahn Equation in 2D.

- In Examples 1 and 2, we were able to derive an analytical solution to the 1D and 2D heat equation. So that the performance of the BSDE solver is examined by comparing numerical solution with the analytical solution.
- In Example 3, we examined a diffusion–reaction equations in 2D and dimensions up to 100. We also compared the performance of the solver with a radial basis function collocation method reported in Ref. 25.
- In Example 4, the focus is a Black–Scholes equation where the performance of the solver is closely experimented by considering

Table 1
Example 1: Comparison of analytical and approximated solutions at different spatial locations: $x = -1, 0, 1, 2,$ and 3. Absolute errors and relative errors are shown in the table, where $T = 1.0$, number of iterations in the deep learning network is 1000, and the number of time steps is 30.

x	BSDE	$u(T,x)$	Absolute error	Relative error
−1	2.554×10^{-2}	2.275×10^{-2}	3.394×10^{-4}	0.123
0	0.147	0.159	1.137×10^{-2}	7.163×10^{-2}
1	0.482	0.5	1.825×10^{-2}	3.788×10^{-2}
2	0.800	0.841	4.174×10^{-2}	5.22×10^{-2}
3	0.988	0.977	1.091×10^{-2}	1.104×10^{-2}

different locations in the domain, number of iterations in the neural network and number of iterations in time discretization.

- In Example 5, an Allen–Cahn equation in 2D is considered.

In all the numerical experiments, spatial domains have been discretized and approximate solutions at several random points have been obtained using the BSDE solver. Due to the stochastic property of the solver, the solver is tested 5 times on all spatial points of interests. Averages of 5 runs are reported in tables and standard deviations are also presented in figures. All simulations are performed on a MacBook Pro with a 2.4 GHz Quad-Core Intel Core i5 processor and 16 GB memory.

Example 1. We consider the following 1D heat equation with terminal condition[26]

$$\begin{aligned} &u_t(\tilde{t}, x) + \frac{1}{2}\frac{\partial u}{\partial x}(t,x) = 0, \qquad \forall\ (t,x) \in [0,T) \times \mathbb{R} \\ &u(T,x) = \chi_{[1,\infty)}(x), \qquad \forall\ x \in \mathbb{R}, \end{aligned} \tag{4.1}$$

where $\chi_{[1,\infty)}$ is an indicator function of $[1,\infty)$ and $\chi_{[1,\infty)}(x) = \begin{cases} 1, & x \geq 1, \\ 0, & otherwise \end{cases}$. It can be shown that there exists at most one strong-viscosity solution to the equation above. The solution is given by

$$u(t,x) = 1 - \Phi\left(\frac{1-x}{\sqrt{T-t}}\right), \qquad (t,x) \in [0,T] \times \mathbb{R}, \tag{4.2}$$

where $\Phi(z) = \int_{-\infty}^{z} \frac{1}{\sqrt{2\pi}} e^{-z^2/2}\ dz$.

Table 1 shows a comparison between analytical solution and the approximated solutions at different spatial locations: $x = -1, 0, 1, 2,$ and 3. Absolute errors and relative errors are shown, where $T = 1.0$, number of iterations in deep learning network is 1000, and the number of time steps is 30.

Left of Fig. 2 shows the profile of the approximated solution at $t = 0, x = 0$ as a function of number of iterations. When the number of iterations higher than 200, the approximated solutions started converge to the true solution. The line represents the average values over 5 runs when the number of iteration is fixed, and the shaded blue region around the line demonstrates the standard deviation of the approximation for that iteration. Right of Fig. 2 shows the profile of the approximated solution versus the analytical solution as a function of x when $t = 0$.

Example 2. We consider the following 2D heat equation with terminal condition

$$\begin{aligned} &u_t(\tilde{t}, \tilde{x}) + \frac{1}{2}\Delta u(\tilde{t}, \tilde{x}) = 0, \qquad \forall\ (\tilde{t}, \tilde{x} = (\tilde{x}_1, \tilde{x}_2)) \in [0,T) \times \mathbb{R}^2 \\ &u(T, \tilde{x}) = \chi_{[1,\infty)\times[1,\infty)}(\tilde{x}), \qquad \forall\ \tilde{x} \in \mathbb{R}^2. \end{aligned} \tag{4.3}$$

We will first derive the analytical solution of the system above. A change of variables is performed to transform a terminal condition problems into an initial value problem. Let $x_1 = \tilde{x}_1 - 1, x_2 = \tilde{x}_2 - 1, t = T - \tilde{t}$, then the system above becomes

$$\begin{aligned} &u_t(t,x) = \frac{1}{2}\Delta u(t,x), \qquad \forall\ (t,x) \in [0,T) \times \mathbb{R}^2 \\ &u(0,x) = \chi_{[1,\infty)\times[1,\infty)}(x), \qquad \forall\ x \in \mathbb{R}^2. \end{aligned} \tag{4.4}$$

Table 2
Example 2: Comparison of approximated solution by the BSDE solver and the analytical solution. Absolute errors and relative errors are shown in the table, where $T = 1.0$, number of iteration in time is $N = 1000$, and 30 time steps are used.

(x_1, x_2)	BSDE	$u(T, x_1, x_2)$	Absolute error	Relative error
(−1,−1)	8.570×10^{-4}	5.176×10^{-4}	3.394×10^{-4}	0.656
(−1,0)	3.423×10^{-3}	3.609×10^{-3}	1.860×10^{-4}	5.155×10^{-2}
(−1,1)	1.019×10^{-2}	1.138×10^{-2}	1.188×10^{-3}	0.104
(0,0)	2.547×10^{-2}	2.517×10^{-2}	2.936×10^{-4}	1.166×10^{-2}
(0,1)	8.240×10^{-2}	7.933×10^{-2}	3.070×10^{-3}	3.870×10^{-2}
(0,2)	0.129	0.133	4.177×10^{-3}	3.129×10^{-2}
(0,3)	0.153	0.155	2.340×10^{-3}	1.509×10^{-2}
(1,1)	0.249	0.25	9.973×10^{-4}	3.989×10^{-3}
(1,2)	0.422	0.421	8.990×10^{-4}	2.137×10^{-3}
(1,3)	0.490	0.487	1.252×10^{-3}	2.562×10^{3}
(2,2)	0.707	0.708	9.257×10^{-4}	1.308×10^{-3}
(2,3)	0.820	0.822	1.767×10^{-3}	2.149×10^{-3}
(3,3)	0.956	0.955	1.104×10^{-3}	1.156×10^{-3}

The solution to the initial value problem[27] is

$$u(t,x) = \iint g(k,h)\frac{1}{2\pi t}e^{-[(x-k)^2+(y-h)^2]/2t}\ dh\ dk. \tag{4.5}$$

Imposing the initial condition $g(x)$ and applying Fubini's Theorem, we arrive at:

$$u(t,x) = \frac{1}{2\pi t}\left(\int_0^\infty e^{-(x_1-h)^2/2t} dh\right)\left(\int_0^\infty e^{-(x_2-k)^2/2t} dk\right).$$

Let $z = \dfrac{h - x_1}{\sqrt{2t}}$ and $w = \dfrac{k - x_2}{\sqrt{2t}}$. Note that $\int_0^z e^{-t^2} dt = \frac{\sqrt{\pi}}{2} erf(z)$.

The solution can be rewritten as follows

$$\begin{aligned} u(t,x) &= \iint g(k,h)\frac{1}{2\pi t}e^{-[(x-k)^2+(y-h)^2]/2t}\ dhdk \\ &= \frac{1}{\pi}\int_{-x_1/\sqrt{2t}}^{\infty} e^{-z^2} dz \int_{-x_2/\sqrt{2t}}^{\infty} e^{-w^2} dw \\ &= \frac{1}{\pi}\left(\int_0^\infty e^{-z^2} dz + \int_0^{x_1/\sqrt{2t}} e^{-z^2} dz\right) \\ &\quad \times \left(\int_0^\infty e^{-w^2} dw + \int_0^{x_2/\sqrt{2t}} e^{-w^2} dw\right) \\ &= \frac{1}{\pi}\left[\frac{\sqrt{\pi}}{2} + \frac{\sqrt{\pi}}{2} erf\left(\frac{x_1}{\sqrt{2t}}\right)\right]\left[\frac{\sqrt{\pi}}{2} + \frac{\sqrt{\pi}}{2} erf\left(\frac{x_2}{\sqrt{2t}}\right)\right] \\ &= \left[\frac{1}{2} + \frac{1}{2} erf\left(\frac{x_1}{\sqrt{2t}}\right)\right]\left[\frac{1}{2} + \frac{1}{2} erf\left(\frac{x_2}{\sqrt{2t}}\right)\right] \\ &= \frac{1}{2}\left[1 + erf\left(\frac{x_1}{\sqrt{2}\sqrt{t}}\right)\right]\frac{1}{2}\left[1 + erf\left(\frac{x_2}{\sqrt{2}\sqrt{t}}\right)\right] \\ &= \Phi\left(\frac{x_1}{\sqrt{t}}\right)\Phi\left(\frac{x_2}{\sqrt{t}}\right). \end{aligned} \tag{4.6}$$

where $\Phi(x) = \frac{1}{2}\left[1 + erf\left(\frac{x}{\sqrt{2}}\right)\right]$. Substituting in the original variables, we arrive at the solution to the original terminal condition problem:

$$u(\tilde{t}, \tilde{x}) = \Phi\left(\frac{\tilde{x}_1 - 1}{\sqrt{T - \tilde{t}}}\right)\Phi\left(\frac{\tilde{x}_2 - 1}{\sqrt{T - \tilde{t}}}\right). \tag{4.7}$$

The profile of our analytical solution is shown in Fig. 3 on the left. On the right of Fig. 3, it shows the absolute error surface when $T = 1, N = 1000$ and 30 time steps are used. Detailed comparison of approximated solution by the BSDE solver and the analytical solution computed on several locations of the domain can be found in Table 2.

Left of Fig. 4 shows the absolute error calculated at $(0,0)$ and on the right, it shows the profile of approximation solution at $(0,0)$ against the number of iterations with standard deviation shown as the shaded region around the curves.

Compare to the 1D heat equation in Example 1, the approximated solution in 2D heat equation experienced less deviation from the mean

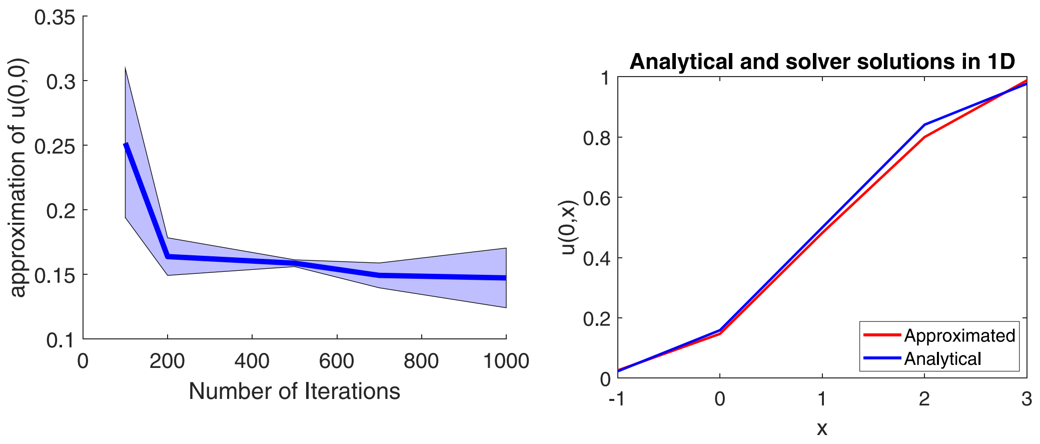

Fig. 2. Example 1: Left is the profile of the approximated solution at $t = 0, x = 0$ as a function of number of iterations; The line represents the average values over 5 runs when the number of iteration is fixed, and the shaded blue region around the line demonstrates the standard deviation of the approximation for that iteration. Right is the analytical solution vs. approximated solution as a function of x at $t = 0$.

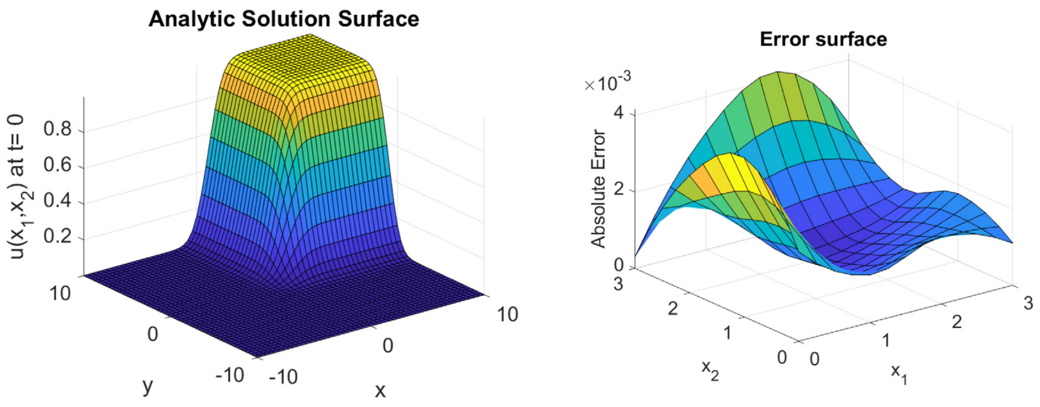

Fig. 3. Example 2: on the left, it shows a profile of the analytical solution; on the right, it shows a profile of absolute error surface when $T = 1.0$, $N = 1000$ and 30 time steps are used.

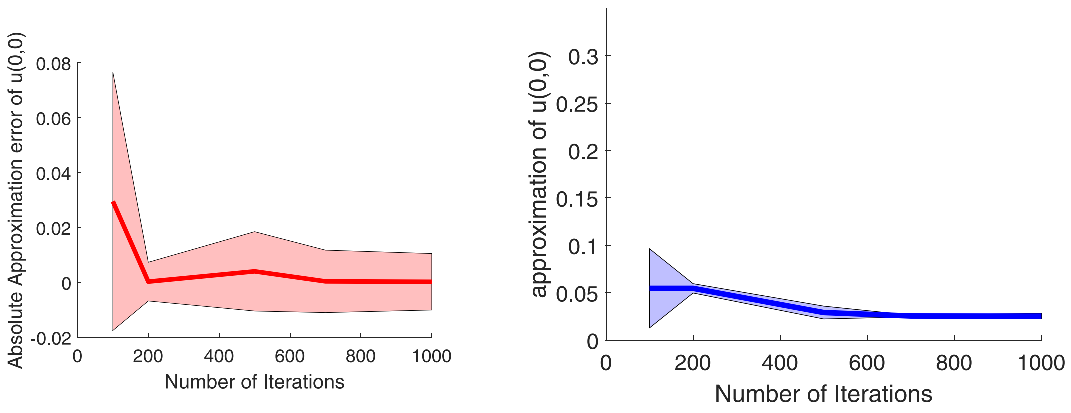

Fig. 4. Example 2: Left: Absolute error at $(0, 0)$. Right: Approximate solution at $(0, 0)$ with standard deviations shown as the shaded region around the mean curves.

value than in 1D. This indicates the solver performs more stable as the dimension gets higher.

Example 3. In this example, we consider the following 2D problem:

$$u_t(t,x) = 0.2\Delta u(t,x) + 0.1u(t,x) \tag{4.8}$$

subject to the initial condition: $u(0,x) = \cos(x_1) + \sin(x_2)$. This problem has an exact analytical solution:

$$u(t,x) = \exp(-0.1t)\left[\cos(x_1) + \sin(x_2)\right]. \tag{4.9}$$

By considering the time reversal $t \to T - t$ for some $T > 0$, we can obtain an equation as expressed in form of (1.1).

$$u_t(t,x) + 0.2\Delta u(t,x) + 0.1u(t,x) = 0. \tag{4.10}$$

This matches the semilinear parabolic form with $\sigma = \sqrt{0.4}\ I_2$, $\mu = 0$, and $f(t,x) = 0.1u(t,x)$, where I_2 is an identity matrix of size 2×2. We have the terminal condition $g(x) = u(T,x) = \cos(x_1) + \sin(x_2)$.

In our numerical experiments, a 6×6 grid of points were tested over the domain $[0, 2\pi] \times [0, 2\pi]$ using total of 5000 iterations in neural network with a learning rate of 0.01 for the first half and 0.0002 for

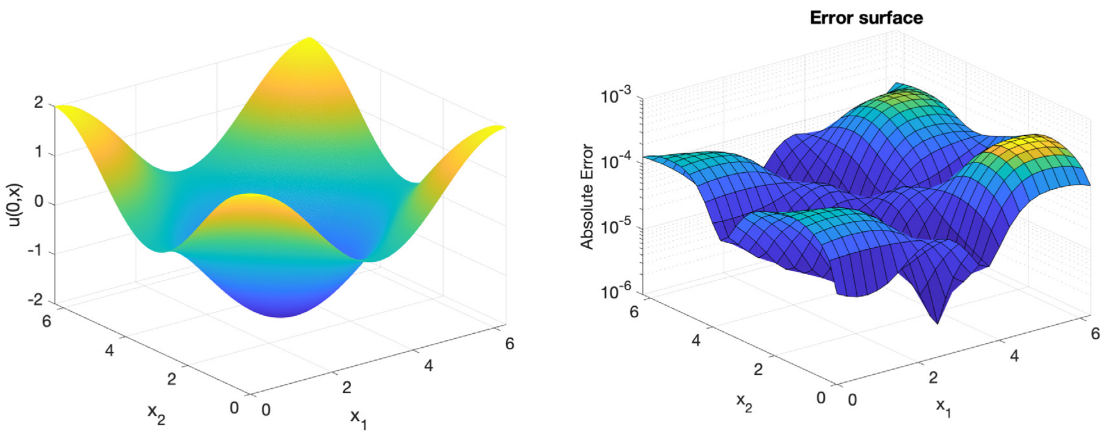

Fig. 5. Example 3: The exact solution profile on the left and the absolute error profile on the right.

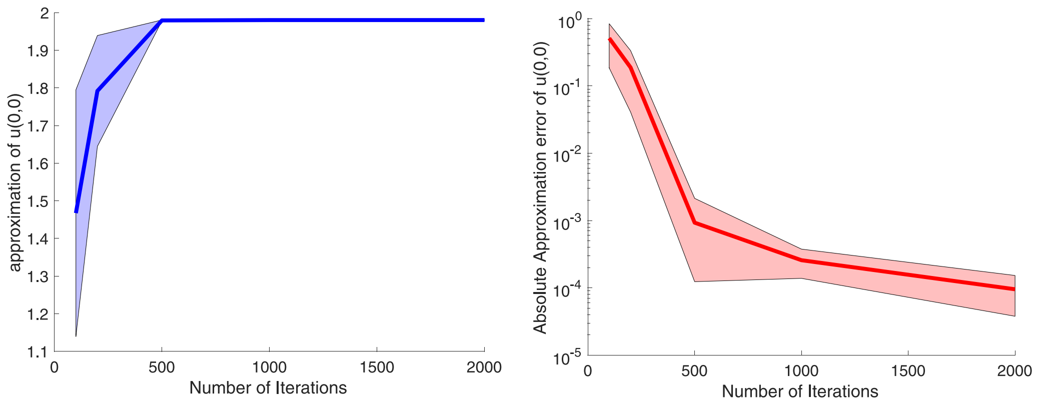

Fig. 6. Example 3: Left: The approximate value of $u(0,0)$ at different numbers of iterations of the Deep BSDE solver and Right: The absolute errors of the Deep BSDE solver at various numbers of iterations.

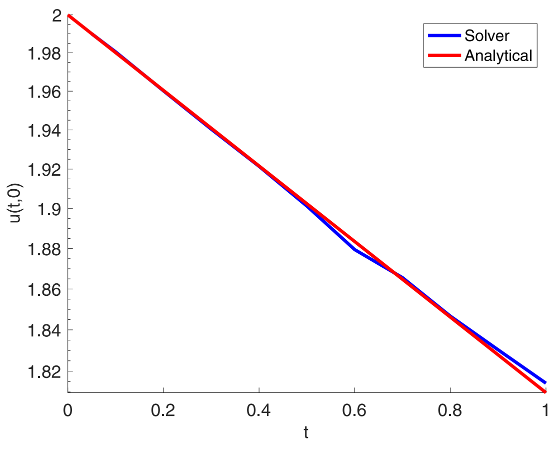

Fig. 7. Example 3: The value of $u(t,0)$ for various times t. The BSDE solver results are in blue, while the exact solution is in red.. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

the second half. In two separate rounds of testing, 10 and 100 step sizes were used with $T = 0.1$.

The surface of the exact solution in 2D and the surface of the absolute error in 2D obtained by the BSDE solver are shown on the left and right of Fig. 5, respectively. To ascertain the impact of the number of iterations used, five independent runs were completed using several total iterations. The average approximation solution and approximation errors at $u(t = 0, x = 0)$ are plotted in Fig. 6. The shaded regions around the lines demonstrate the standard deviation of the approximation and the error for the particular iteration. It is clear from Fig. 6, as the number of iterations increases, the approximate solution converges to an asymptotic value equal to the analytical solution and absolute error decreases with increasing number of iterations. Time profile of the 2D solution was obtained by running the solver with different end conditions, which can be found in Fig. 7 when the number of iteration in the deep learning network is sufficient.

In Table 3, we compare the performance of the BSDE solver with RBFCM in Ref. 28. The accuracy of the results from BSDE solver did not improve substantially when the time step size was reduced, while the time required to perform the approximation increased greatly. Compared to the RBFCM, the BSDE solver with the aforementioned parameters produces results more accurate than the results obtained by RBFCM. It should be noted that the computational time is high, requiring several minutes per point of interest when 100 times steps are used. Even when 10 time steps are used, the computation time is still substantial.

To further evaluate the performance of the Deep BSDE solver, the previous equation was considered in d dimensions with d being 5, 10, 20, 50, and 100:

$$u_t(t,x) = 0.2\Delta u(t,x) + 0.1u(t,x)$$

Subject to $u(0,x) = g(x) = \sum_{i=1}^{d}\cos(x_i)$. This has a solution:

$$u(t,x) = \exp(-0.1t)\sum_{i=1}^{d}\cos(x_i).$$

Table 4 shows the approximated solutions at the origin for various d. For each value of d, the approximations are calculated by averaging

Table 3
Example 3: Comparison of BSDE solution with different time step sizes and with RBFCM presented in Ref. 28. Decreasing the step size by a factor of 10 had minimal impact on the error while increasing time greatly.

Δt	BSDE Solver			RBFCM[28]	
	L_∞	L_{rms}	CPU time	L_∞	L_{rms}
$\Delta t = 10^{-2}$	3.002×10^{-4}	1.598×10^{-4}	38	8.31×10^{-3}	2.45×10^{-3}
$\Delta t = 10^{-3}$	3.456×10^{-4}	1.340×10^{-4}	408.5	3.33×10^{-3}	2.45×10^{-3}

Table 4
Example 3: The approximated solutions at the origin, errors and CPU time for various $d = 5, 10, 20, 50$ and 100, where $T = 0.1$. 3000 iterations are used in the deep learning network, learning rate is set to be 0.5 for the first half of the iterations to speed up computational time, and it was set to be 0.0001 for the next half of the iterations.

d	5	10	20	50	100	200	500
Y_0 (BSDE)	4.951	9.900	19.80	49.50	99.00	198.0	479.0
Y_0 (exact)	4.951	9.901	19.80	49.50	99.01	198.0	495.0
Absolute error	$2.63E-4$	$4.82E-4$	$2.70E-3$	$2.30E-3$	$6.74E-4$	$5.47E-4$	$1.60E1$
Relative error	0.0053%	0.0049%	0.0136%	0.0046%	0.0007%	0.0003%	3.23%
CPU Time	50.8	51.4	53.4	61.4	75.8	109.1	220

10 independent trials. In running the trials, a total time of $T = 0.1$ was considered, split up into 20 time steps. A total of 3000 iterations were considered for each trial with a learning rate of 0.5 for the first half and 0.0001 for the second half. The BSDE solver is fairly accurate despite the dimension of the problem. However, the relative error when $d = 500$ is slightly higher than when the number of dimensions are small.

Example 4. Let us consider a model of pricing with default risk with domain $[0,T] \times \mathbb{R}^2$. Let R be the interest rate of the risk-free asset and Q be an intensity function defined in three regions $(v^h < v^l, \gamma^h > \gamma^l)$, which are defined by :

$$Q(y) = 1_{(-\infty, v^h)}(y)\gamma^h + 1_{[v^l,\infty)}(y)\gamma^l + 1_{[v^h,v^l)}(y)\left(\frac{\gamma^h - \gamma^l}{v^h - v^l}(y - v^h) + \gamma^h\right), \tag{4.11}$$

$$f(t,x,u(t,x),\sigma^T(t,x)\nabla u(t,x)) = -(1-\delta)Q(u(t,x))u(t,x) - Ru(t,x), \tag{4.12}$$

for any $\delta \in (0,1]$, where f is the value process. The nonlinear Black–Scholes equation then can be formed as follows

$$\frac{\partial u}{\partial t} + \mu x \nabla u(t,x) + \frac{\sigma^2}{2}\left(|x_1|^2 \frac{\partial^2 u}{\partial x_1^2}(t,x) + |x_2|^2 \frac{\partial^2 u}{\partial x_2^2}(t,x)\right)$$
$$- (1-\delta)Q(u(t,x))u(t,x) - Ru(t,x) = 0. \tag{4.13}$$

In mathematics finance, Black and Scholes[29] is a partial differential equation that governs the price evolution of a European call or European put options. Finding the analytic closed-form solution of the Black–Scholes equation is not easy. Therefore, it is necessary to approximate the solutions using numerical methods. The finite difference methods (FDM) such as the operator splitting method[30] and multigrid method[31] are very popular to approximate the solution of the BS equations.[32] In 2018, Grohs[33] showed that artificial neural networks overcome the curse of dimensionality in the numerical approximation of Black–Scholes partial differential equations.

To verify the accuracy of the BSDE solver in lower dimension, we approximate the initial value $u(t = 0, (x_1, x_2))$, with $(x_1, x_2) \in [0, 100]^2$, using the BSDE solver and compare with the approximations computed means of the multilevel Picard method.[34] We set all the known parameters as follows:

$$T = 1, \delta = \frac{2}{3}, R = 0.02, \mu = 0.02, \sigma = 0.2,$$
$$v^h = 50, v^l = 70, \gamma^h = 0.2, \gamma^l = 0.02, \tag{4.14}$$

and terminal condition $g(x) = \min\{x_1, x_2\}$. Thus, the nonlinear Black–Scholes equation is simplified as:

$$\frac{\partial u}{\partial t} + 0.02x\nabla u(t,x) + 0.02\left(|x_1|^2 \frac{\partial^2 u}{\partial x_1^2}(t,x) + |x_2|^2 \frac{\partial^2 u}{\partial x_2^2}(t,x)\right)$$

Table 5
Example 4: Approximation of $u(t = 0, x = (x_1, x_2))$ at different values of x_1 and x_2.

$x_2 \backslash x_1$	0	20	40	60	80	100
0	2.41×10^{-4}	0.0632	9.67×10^{-3}	2.24×10^{-4}	0.0183	0.0075
20	0.03	16.596	18.811	18.835	18.176	18.722
40	3.44×10^{-3}	18.583	33.199	37.194	37.43	37.472
60	4.25×10^{-3}	18.599	36.805	50.412	56.108	57.428
80	3.3942×10^{-2}	18.605	37.234	56.083	70.096	76.269
100	3.6747×10^{-2}	18.606	37.274	57.537	76.224	88.16

$$- \frac{1}{3}Q(u(t,x))u(t,x) - 0.02u(t,x) = 0. \tag{4.15}$$

Table 5 shows the approximations of $u(t = 0)$ computed on different locations of the domain and left of Fig. 9 shows the corresponding solution profile. According to the Fig. 8, solver seems to converge to a solution when we let it work higher number of iterations. Moreover, we compute the approximations using multilevel Picard method for the exact same (x_1, x_2) as in Table 5 and present them on the Table 6. Accuracy of the BSDE solver in lower dimensions is evident from the errors computed in Table 6 and absolute error profile on the right of Fig. 9.

From the left of Fig. 10 it is clear that solutions from the BSDE solver converge for any given point (x_1, x_2) if the number of iteration in Neural Network is high enough. The right of Fig. 10 shows that BSDE solver can obtain approximation which is accurate up to fourth order by increasing the number of iterations in neural network. The approximation solutions of $u(t = 0, x = (100, 100))$ were reported at different time steps for number of iterations 1000 and 8000 in Table 7. The approximations vary significantly as the number of time steps changes when the number of iteration in network is not high enough to achieve convergence in the solution. The not explicitly known "exact" solution at $t = 0$, $x = (100, 100)$ has been approximately computed using the multilevel Picard method[35]: $u(t = 0, x = (100, 100)) \approx 88.16$.

Example 5. We consider the following Allen–Cahn Equation in 2D on the domain $[0, 10] \times [0, 10]$:

$$u_t(t,x) = \Delta u(t,x) + u(t,x) - u(t,x)^3 \tag{4.16}$$

subject to the initial condition: $u(0,x) = \frac{1}{2+0.4\|x\|^2}$.

By considering the time reversal $t \to T - t$ for some $T > 0$, we can obtain an equation as expressed in (1.1):

$$u_t(t,x) + \Delta u(t,x) + u(t,x) - u(t,x)^3 = 0. \tag{4.17}$$

This matches the semilinear parabolic form with $\sigma = \sqrt{2}I_2$, $\mu = 0$, and $f(t,x) = u(t,x) - u(t,x)^3$.

Table 6
Example 4: Relative and absolute errors of $u(t=0,x=(x_1,x_2))$, for each $x_1,x_2 \in \{0,20,40,60,80,100\}$, compared with approximations made with multilevel Picard method values.

$x_2\backslash x_1$		0	20	40	60	80	100
0	$Y_0(Picard)$	0	0	0	0	0	0
	Relative error						
	Absolute error	2.41×10^{-4}	0.0632	9.67×10^{-3}	2.24×10^{-4}	0.0183	0.0075
20	$Y_0(Picard)$	0	16.5963	18.8110	18.8354	18.8356	18.8354
	Relative error		0.01%	0.58%	0.64%	0.64%	0.6%
	Absolute error	0.03	.002137	0.1095	0.121	0.1199	0.1133
40	$Y_0(Picard)$	0	18.5834	33.1986	37.1937	37.611	37.6367
	Relative error		0.61%	0.02%	0.53%	0.48%	0.44%
	Absolute error	3.44×10^{-3}	0.1142	.0054	0.1962	0.1813	0.1643
60	$Y_0(Picard)$	0	18.5994	36.8051	50.4123	56.2028	57.6156
	Relative error		0.63%	0.63%	0.04%	0.17%	0.33%
	Absolute error	4.25×10^{-3}	0.1165	0.2324	0.0187	0.0944	0.1873
80	$Y_0(Picard)$	0	18.6048	37.2338	56.0825	70.1898	76.3015
	Relative error		0.6%	0.49%	0.02%	0.13%	0.04%
	Absolute error	0.0339	0.1118	0.1821	0.0131	0.0935	0.0321
100	$Y_0(Picard)$	0	18.6059	37.2738	57.569	76.1515	88.16
	Relative error		0.6%	0.45%	0.06%	0.09%	0.0001%
	Absolute error	0.03674	0.1109	0.1674	0.0318	0.072	9.83×10^{-5}

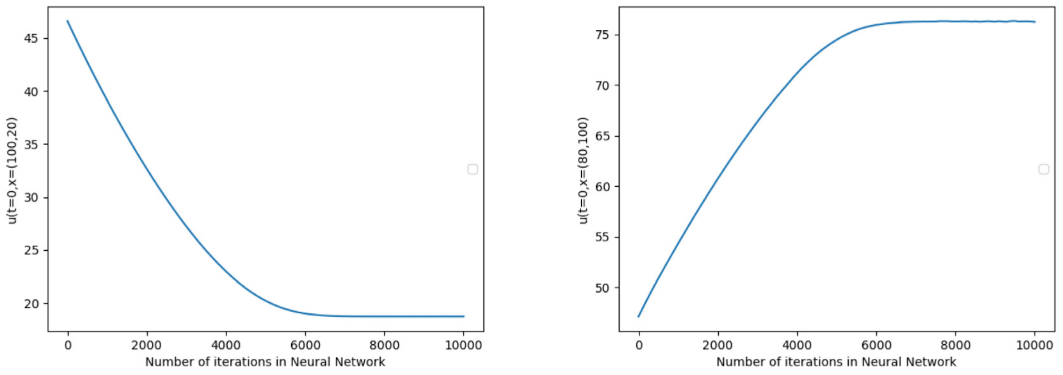

Fig. 8. Example 4: Approximations of $u(t=0)$ at $(100,20)$ and $(80,100)$ using the BSDE solver for Black–Scholes equation against number of iterations in neural network.

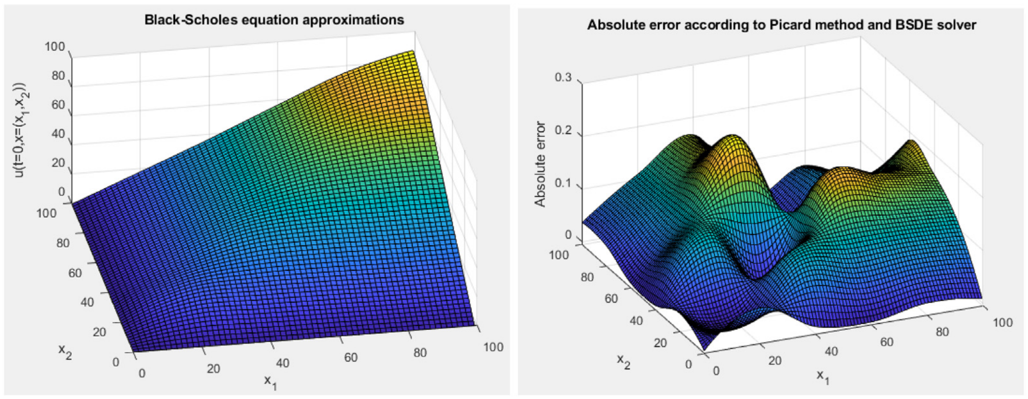

Fig. 9. Example 4: Left: Approximate solution $u(t=0,x=(x_1,x_2))$ profile based on Table 5, Right:Absolute error compared to the approximation of u obtained by multilevel Picard method.

Table 7
Example 4: Comparison of approximations of $u(t=0,x=(100,100))$ as a mean of 5 independent runs at different time steps for number of iterations 1000 and 8000.

# of time steps	10	20	40	60	100
1000 Iterations	35.165	37.725	37.605	35.864	36.781
8000 Iterations	87.631	88.057	87.269	87.180	87.530

The Allen–Cahn equation was first introduced by Allen and Cahn in Ref. 36 to describe the motion of anti-phase boundaries in crystalline solids. Then it has been widely used in many fields.[36–39] Numerical methods[40] including Finite difference method,[41] the reduced order method[42,43] spectral method[40,44] and reproducing kernel method,[45] radial basis function collocation method,[46] and many others have investigated.

In this example, the Allen–Cahn Equation was explored using both the BSDE solver and the RBF solver. Solution surfaces for the two methods are shown in Fig. 11. Fig. 12 shows the absolute error obtained by the BSDE solver when using RBF solver as the comparison solutions. The error is viewed from two different angles.

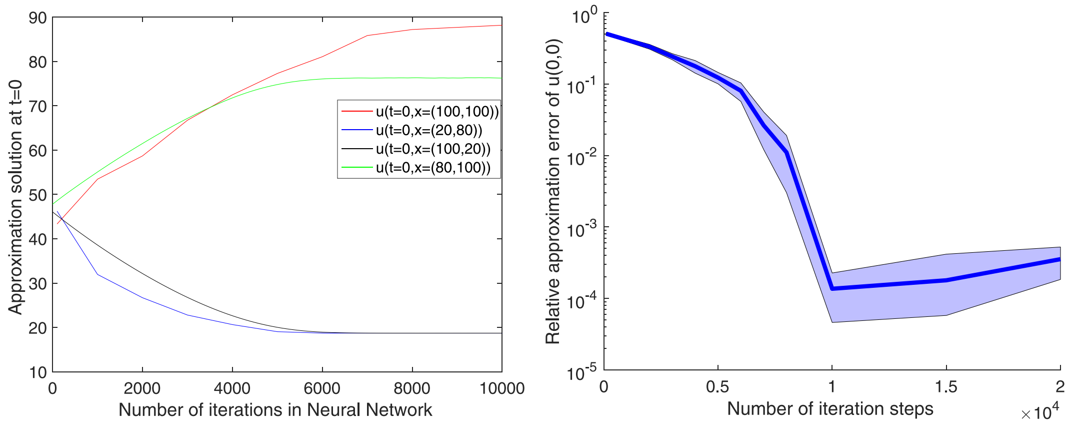

Fig. 10. Example 4: Left shows an approximation of u at $t = 0$ as a mean of 5 independent runs against the number of iterations in network for different points in the domain. Right shows the relative approximation error of $u(t = 0, x = (100, 100))$ against the number of iterations in network. The shaded area depicts the mean ±the SD of the relative error for five different runs.

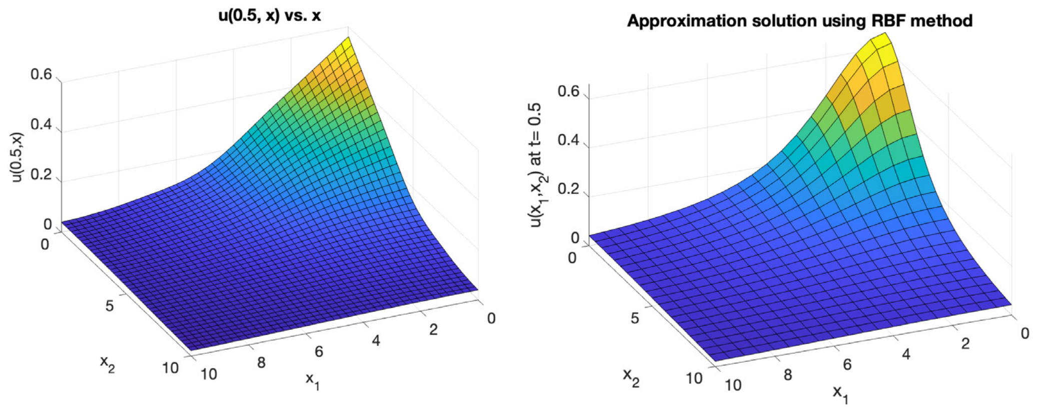

Fig. 11. Example 5: Approximation solution of $u(x_1, x_2)$ at $t = 0.5$ in a square domain using BSDE method on the left and using RBF method on the right.

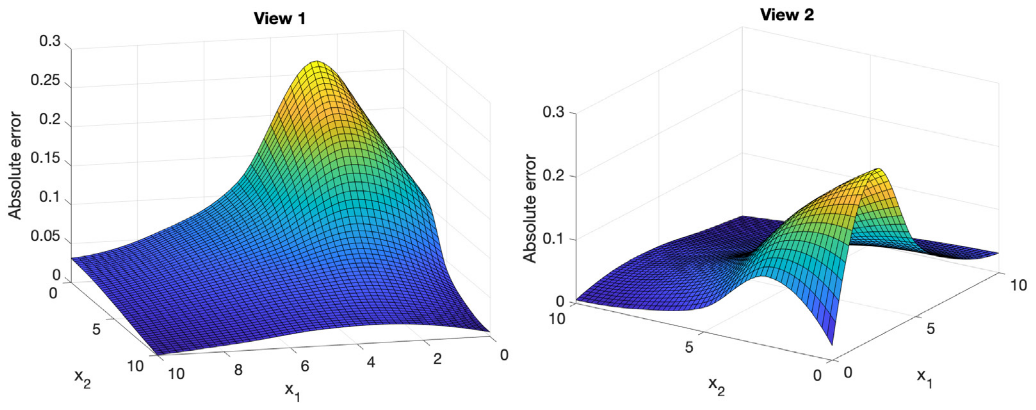

Fig. 12. Example 5: The approximation error of u at $t = 0.5$ in two different views.

5. Conclusions

Backward stochastic differential equation solver solves semilinear parabolic partial differential equations by converting them into stochastic differential equations . This algorithm is especially powerful for solving high-dimensional PDEs, which traditional numerical techniques cannot handle.

This paper modified the BSDE solver so it can be used for one-dimensional problems. Furthermore, we compared the BSDE solver with traditional numerical techniques in low dimensional spaces including 1D, 2D. Through classical differential equations, we discovered that the solver works well for low dimensional problems, as accurate as it can be in high dimensional spaces. However, the BSDE solver might take much longer time than the traditional methods, considering that it only handles one point at a time and the approximation should be computed by means of few independent runs of the algorithm.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

1. Evans LC. *Partial Differential Equations, Vol. 19*. American Mathematical Society; 2010.
2. Morton KW, Mayers DF. *Numerical Solution of Partial Differential Equations: An Introduction*. Cambridge University Press; 2005.
3. Lapidus L, Pinder GF. *Numerical Solution of Partial Differential Equations in Science and Engineering*. John Wiley & Sons; 2011.
4. Farshid M, Shadi R, Nasrin S. Application of combination schemes based on radial basis functions and finite difference to solve stochastic coupled nonlinear time fractional sine–Gordon equations. *Comput Appl Math*. 2021;41(10).
5. Cheng L. KdV-type Wronskian rational solutions to the (4+ 1)-dimensional Fokas equation. *Partial Differ Equ Appl Math*. 2022;5:100222.
6. Kloeden PE, Platen E. Stochastic differential equations. In: *Numerical Solution of Stochastic Differential Equations*. Springer; 1992:103–160.
7. Platen E. An introduction to numerical methods for stochastic differential equations. *Acta Numer*. 1999;8:197–246.
8. Burrage K, Burrage P, Mitsui T. Numerical solutions of stochastic differential equations–implementation and stability issues. *J Comput Appl Math*. 2000;125(1–2):171–182.
9. Higham DJ. An algorithmic introduction to numerical simulation of stochastic differential equations. *SIAM Rev*. 2001;43(3):525–546.
10. Mirzaee F, Sayevand K, Rezaei S, Samadyar N. Finite difference and spline approximation for solving fractional stochastic advection-diffusion equation. *Iran J Sci Technol Trans A Sci*. 2021;45(2):607–617.
11. Mirzaee F, Samadyar N. Implicit meshless method to solve 2D fractional stochastic Tricomi-type equation defined on irregular domain occurring in fractal transonic flow. *Numer Methods Partial Differential Equations*. 2021;37(2):1781–1799.
12. Yan Y. Galerkin finite element methods for stochastic parabolic partial differential equations. *SIAM J Numer Anal*. 2005;43(4):1363–1384.
13. Mirzaee F, Rezaei S, Samadyar N. Solving one-dimensional nonlinear stochastic Sine-Gordon equation with a new meshfree technique. *Int J Numer Modell: Electron Netw*. 2020;34.
14. Mirzaee F, Samadyar N. Application of Bernoulli wavelet method for estimating a solution of linear stochastic Itô-Volterra integral equations. *Multidiscip Model Mater Struct*. 2018;15(3):575–598.
15. Mirzaee F, Samadyar N. Combination of finite difference method and meshless method based on radial basis functions to solve fractional stochastic advection–diffusion equations. *Eng Comput*. 2020;36(4):1673–1686.
16. Samadyar N, Ordokhani Y, Mirzaee F. Hybrid Taylor and block-pulse functions operational matrix algorithm and its application to obtain the approximate solution of stochastic evolution equation driven by fractional Brownian motion. *Commun Nonlinear Sci Numer Simul*. 2020;90:105346.
17. Berg J, Nyström K. A unified deep artificial neural network approach to partial differential equations in complex geometries. *Neurocomputing*. 2018;317:28–41.
18. Han J, Zhang L, Weinan E. Solving many-electron Schrödinger equation using deep neural networks. *J Comput Phys*. 2019;399:108929.
19. Han J, Long J. Convergence of the deep BSDE method for coupled FBSDEs. *Probab Uncertain Quant Risk*. 2020;5(1):1–33.
20. Lye KO, Mishra S, Ray D. Deep learning observables in computational fluid dynamics. *J Comput Phys*. 2020;410:109339.
21. Weinan E, Han J, Jentzen A. Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. *Commun Math Stat*. 2017;5(4):349–380.
22. Øksendal B. *Stochastic Differential Equations: An Introduction with Applications, Vol. 82*. 2000.
23. Pardoux É, Peng S. Backward stochastic differential equations and quasilinear parabolic partial differential equations. In: *Stochastic Partial Differential Equations and their Applications*. Springer; 1992:200–217.
24. Kingma DP, Ba J. A method for stochastic optimization. In: *CAnon. International-Conferenceon Learning Representations*. SanDego: ICLR, 2015 abs/1412.6980.
25. Yao G. An improved localized method of approximate particular solutions for solving elliptic PDEs. *Comput Math Appl*. 2016;71(1):171–184.
26. Cosso A, Russo F. Strong-viscosity solutions: Classical and path-dependent PDEs. *Osaka J Math*. 2019;56(2):323–373.
27. Haberman R. *Elementary Applied Partial Differential Equations, Vol. 987*. Prentice Hall Englewood Cliffs, NJ; 1983.
28. Yao G. A comparative study of global and local meshless methods for diffusion-reaction equation. *Comput Model Eng Sci(CMES)*. 2010;59(2):127–154.
29. Black F, Scholes M. The pricing of options and corporate liabilities. In: *World Scientific Reference on Contingent Claims Analysis in Corporate Finance: Volume 1: Foundations of CCA and Equity Valuation*. World Scientific; 2019:3–21.
30. Ikonen S, Toivanen J. Operator splitting methods for American option pricing. *Appl Math Lett*. 2004;17(7):809–814.
31. Ramage A, von Sydow L. A multigrid preconditioner for an adaptive Black-Scholes solver. *BIT Numer Math*. 2011;51(1):217–233.
32. Duffy DJ. *Finite Difference Methods in Financial Engineering: A Partial Differential Equation Approach*. John Wiley & Sons; 2013.
33. Grohs P, Hornung F, Jentzen A, Von Wurstemberger P. A proof that artificial neural networks overcome the curse of dimensionality in the numerical approximation of Black-Scholes partial differential equations. 2018 arXiv, abs/1809.02362.
34. Hutzenthaler M, Jentzen A, Kruse T, et al. On multilevel Picard numerical approximations for high-dimensional nonlinear parabolic partial differential equations and high-dimensional nonlinear backward stochastic differential equations. *J Sci Comput*. 2019;79(3):1534–1571.
35. Weinan E, Hutzenthaler M, Jentzen A, Kruse T. On multilevel picard numerical approximations for high-dimensional nonlinear parabolic partial differential equations and high-dimensional nonlinear backward stochastic differential equations. *J Sci Comput*. 2019;79(3):1534–1571.
36. Allen SM, Cahn JW. A microscopic theory for antiphase boundary motion and its application to antiphase domain coarsening. *Acta Metall*. 1979;27(6):1085–1095.
37. Li Y, Jeong D, Choi J-i, Lee S, Kim J. Fast local image inpainting based on the Allen–Cahn model. *Digit Signal Process*. 2015;37:65–74.
38. Cheng M, Warren JA. An efficient algorithm for solving the phase field crystal model. *J Comput Phys*. 2008;227(12):6241–6248.
39. Ward MJ. Metastable bubble solutions for the Allen-Cahn equation with mass conservation. *SIAM J Appl Math*. 1996;56(5):1247–1279.
40. Shen J, Yang X. Numerical approximations of Allen-Cahn and Cahn-Hilliard equations. *Discrete Contin Dyn Syst*. 2010;28(4):1669–1691.
41. Kim J, Jeong D, Yang S-D, Choi Y. A finite difference method for a conservative Allen–Cahn equation on non-flat surfaces. *J Comput Phys*. 2017;334:170–181.
42. Song H, Jiang L, Li Q. A reduced order method for Allen–Cahn equations. *J Comput Appl Math*. 2016;292:213–229.
43. Abbaszadeh M, Dehghan M. A reduced order finite difference method for solving space-fractional reaction-diffusion systems: The Gray-Scott model. *Eur Phys J Plus*. 2019;134(12):1–15.
44. Lee HG, Lee J-Y. A semi-analytical Fourier spectral method for the Allen–Cahn equation. *Comput Math Appl*. 2014;68(3):174–184.
45. Niu J, Xu M, Yao G. An efficient reproducing kernel method for solving the Allen–Cahn equation. *Appl Math Lett*. 2019;89:78–84.
46. Hon Y-C, Mao X-Z. A radial basis function method for solving options pricing model. *Financ Eng*. 1999;81(1):31–49.