Computational Economics
https://doi.org/10.1007/s10614-022-10279-x

A Deep Learning Based Numerical PDE Method for Option Pricing

Xiang Wang[1] · Jessica Li[2] · Jichun Li[3]

Accepted: 12 May 2022
© The Author(s), under exclusive licence to Springer Science+Business Media, LLC, part of Springer Nature 2022

Abstract
Proper pricing of options in the financial derivative market is crucial. For many options, it is often impossible to obtain analytical solutions to the Black–Scholes (BS) equation. Hence an accurate and fast numerical method is very beneficial for option pricing. In this paper, we use the Physics-Informed Neural Networks (PINNs) method recently developed by Raissi et al. (J Comput Phys 378:686–707, 2019) to solve the BS equation. Many experiments have been carried out for solving various option pricing models. Compared with traditional numerical methods, the PINNs based method is simple in implementation, but with comparable accuracy and computational speed, which illustrates a promising potential of deep neural networks for solving more complicated BS equations.

Keywords Option pricing · Black–Scholes equation · Deep learning · Neural networks

1 Introduction

A financial derivative, or simply derivative, is a financial instrument whose value depends on the values of other underlying assets such as common stocks, bonds, stock indexes, interest rates, etc. Option is one of the most popular derivative types

✉ Jichun Li
jichun.li@unlv.edu

Xiang Wang
wangxiang49@ncu.edu.cn

Jessica Li
jessica_li2@brown.edu

1 Department of Mathematical Sciences, and Institute of Mathematics and Interdisciplinary Sciences, Nanchang University, Nanchang, China

2 Program in Applied Mathematics and Computer Science, Brown University, Providence, RI 02912, USA

3 Department of Mathematical Sciences, University of Nevada, Las Vegas, NV 89154-4020, USA

Published online: 02 June 2022

and is used by hedgers to reduce the risk caused by potential future movements in a market variable and by speculators to bet on the future direction of a market variable. Obviously, derivative pricing is very important and has been widely studied in both financial industry and academia.

In 1973, Black and Scholes (1973) developed the famous Black–Scholes (BS) model for pricing financial derivatives. In 1997, they were awarded the Nobel Prize in Economics to honour their contributions to option pricing. Over the past several decades, many complicated option pricing models have been proposed. Due to their complexity, numerical methods have to be used for accurately pricing those options. Tree, Monte Carlo, and partial differential equation (PDE) are the three major methods in derivative pricing. Due to our expertise and interest, we focus on the PDE method. Over the years, various numerical methods have been proposed for option pricing. For example, a fourth-order compact finite difference scheme was proposed to solve a one-dimensional (1-D) nonlinear BS equation in Liao and Khaliq (2009), and further the scheme is proved to be unconditionally stable. A fourth-order compact finite difference scheme with adaptive node refinement was developed in Lee and Sun (2012) to solve the jump-diffusion model written as a partial integro-differential equation (PIDE). A superconvergent fitted finite volume method was proposed in Wang et al. (2015) to solve a degenerate nonlinear penalized BS equation used for European and American option pricing. A special kind of finite volume method was proposed in Koffi and Tambue (2020) to solve the BS model with two underlying assets. A radial basis function (cf. Ch.10, Li & Chen, 2019; Li & Nan, 2019) generated finite difference (RBF-FD) method was developed in Soleymani and Zhu (2021) to solve a stochastic volatility jump model written as a 2-D PIDE. A radial basis function coupled with partition of unity method was developed in Mollapourasl et al. (2019) for solving American options with stochastic volatility. A high-order finite difference method was developed in Dilloo and Tangman (2017) to solve various option pricing models such as 1-D nonlinear BS equation, Merton's jump-diffusion model and 2-D Heston's stochastic volatility model. In Lin and Zhu (2020), ADI-based predictor-corrector finite difference schemes were developed and analyzed for pricing the American-style convertible bonds with a Heston stochastic volatility model and a stochastic interest rate model, respectively. More backgrounds and references on option pricing through computational PDE method can be found in books such as (Higham, 2004; Zhu et al., 2004; Seydel, 2012).

Solving PDEs with a neural network has been considered previously in papers such as (Lagaris et al., 2000; Malek & Beidokhti, 2006)). But these papers estimate the neural network solutions on an a priori fixed mesh and mostly with just a single hidden layer. In recent years, a new class of techniques, called deep learning (Goodfellow et al., 2016), have emerged in machine learning by using deep neural networks and have been used to solve a variety of high-dimensional PDEs. (e.g., Han 2018, Sirignano & Spiliopoulos 2018). Here we are interested in the so-called Physics-Informed Neural Networks (PINNs) method (Raissi et al., 2019) recently developed by George Karniadarkis and his collaborators at Brown University, since it has been shown to be quite powerful and efficient in solving various PDEs such as Navier-Stokes equations, integro-differential equations, fractional differential

equations, and stochastic equations. Compared to the traditional numerical methods for solving PDEs (e.g., Li & Chen, 2019 and references therein) such as the finite difference method, the finite element method and finite volume method, the deep learning based numerical PDE method is a mesh-free method and has the potential for overcoming the curse of dimensionality. More details about PINNs can be found in their very recent review article (Lu et al., 2021).

The idea to use neural networks in finance was not new and many researches have been done since 1990s, see, e.g., Hutchinson et al. (1994) for supervised learning and (Barucci et al., 1997) for Galerkin method coupled with shallow neural networks. Recently, in Eskiizmirliler et al. (2021), the authors proposed to solve the one-dimensional BS equation for predicting the value of European call options by using a feed-forward neural network (with just a single hidden layer). In Arin and Ozbayoglu (2020), the authors proposed some hybrid deep neural network models for options pricing. Their method is classifier based, not PDE based. In Han et al. (2018), the authors reformulated the high-dimensional nonlinear BS equation into a backward stochastic differential equations (BSDEs) and approximate the gradient of the solution using deep neural networks. An example of 100-dimension problem was presented to show the capability of their deep BSDE method. More references on applications of neural networks in option pricing and hedging can be found in a recent literature review paper (Ruf & Wang, 2020).

The practical success of PINNs for solving PDEs by using deep neural networks has been very astonishing and encourages us to extend the power of PINNs to solving those PDEs appearing in option pricing. Our numerical results demonstrate that PINNs can be a powerful numerical PDE tool for option pricing in terms of simplicity in implementation, accuracy and speed. We believe that this opens up a host of possibilities in solving more complicated PDEs in economics and finance.

The rest of the paper is organized as follows. In Sect. 2, we present a brief introduction to deep neural networks and the PINN based method for solving PDEs. In Sect. 3, we demonstrate the capability and robustness of PINNs for solving the standard Black-Schole equation through several examples. In Sect. 4, we extend the application of PINNs to solve two-asset options problems. More numerical examples are presented to illustrate the generality and accuracy in solving more complicated option pricing models. Finally, we conclude the paper in Sect. 5.

2 Deep Neural Networks and Applications for Solving PDEs

There are many different types of neural networks developed in the past decades, such as convolutional neural network, the recurrent neural network, residual neural network (ResNet), radial basis function networks, and restricted Boltzmann machines etc. The feed-forward neural network (FNN) has been shown to be efficient for solving PDEs and is used in DeepXDE (Lu et al., 2021). Following Lu et al. (2021), let us denote $\mathcal{N}_L(\boldsymbol{x}) : \mathcal{R}^{d_{in}} \to \mathcal{R}^{d_{out}}$ for an L-layer neural network (i.e., an $(L-1)$-hidden layer neural network) with N_l neurons in the l-th layer. The FNN can be recursively defined as follows:

$$\begin{aligned} &\text{input layer:} && \mathcal{N}_0(\boldsymbol{x}) = \boldsymbol{x} \in \mathcal{R}^{d_{in}}, \\ &\text{hidden layers:} && \mathcal{N}_l(\boldsymbol{x}) = \sigma(W_l \mathcal{N}_{l-1}(\boldsymbol{x}) + \boldsymbol{b}_l) \in \mathcal{R}^{N_l},\ 1 \le l \le L-1, \\ &\text{output layer:} && \mathcal{N}_L(\boldsymbol{x}) = W_L \mathcal{N}_{L-1}(\boldsymbol{x}) + \boldsymbol{b}_L \in \mathcal{R}^{d_{out}}, \end{aligned} \tag{1}$$

where $W_l \in \mathcal{R}^{N_l \times N_{l-1}}$ and $\boldsymbol{b}_l \in \mathcal{R}^{N_l}$ denote the weight matrix and the bias vector in the lth layer, respectively, and σ denotes the activation function. Some popularly used activation functions are the hyperbolic tangent ($\tanh x$), the sigmoid ($\frac{1}{1+e^{-x}}$), and the rectified linear unit ($\mathrm{ReLU}(x) = \max\{x, 0\}$).

To descibe the Physics-Informed Neural Networks for solving PDEs, let us consider a general PDE with the solution $u(\boldsymbol{x})$ defined on a domain $\Omega \subset \mathcal{R}^d$:

$$f\left(\boldsymbol{x}; \frac{\partial u}{\partial x_1}, \cdots, \frac{\partial u}{\partial x_d}; \frac{\partial^2 u}{\partial x_1 \partial x_1}, \cdots, \frac{\partial^2 u}{\partial x_1 \partial x_d}; \cdots\right) = 0, \quad \boldsymbol{x} := (x_1, \cdots, x_d)' \in \Omega, \tag{2}$$

supplemented with proper boundary conditions

$$\mathcal{B}(u, \boldsymbol{x}) = 0 \quad \text{on} \quad \partial\Omega, \tag{3}$$

where $\mathcal{B}(u, \boldsymbol{x})$ can be Dirichlet, Neumann, Robin, or even periodic boundary condition. For time-dependent PDEs, we just treat the time variable t as a special component of $\boldsymbol{x}$, and Ω becomes a spatio-temporal domain. In this case, the initial condition is treated as a special type of boundary condition $\mathcal{B}(u, \boldsymbol{x})$.

The major steps of the PINNs algorithm for solving a PDE are as follows (cf. Lu et al. 2021):

Step 1 Construct a neural network $\hat{u}(\boldsymbol{x}; \theta)$ with parameters $\theta = \{W_l, \boldsymbol{b}_l\}_{1 \le l \le L}$.

Step 2 Specify the training data $\mathcal{T}_{pde}$ and $\mathcal{T}_{ib}$ of sizes N_{pde} and N_{ib}, respectively, to satisfy the physics imposed by the PDE and the boundary/initial conditions. Here $\mathcal{T}_{pde} \subset \Omega$ and $\mathcal{T}_{ib} \subset \partial\Omega$ are points inside the domain Ω and on the domain boundary $\partial\Omega$, respectively.

Step 3 Specify a loss function defined as a weighted sum of the L^2 errors for the PDE and the boundary/initial conditions:

$$\begin{aligned} \mathcal{L}(\theta) :=& \omega_{pde} \cdot \frac{1}{N_{pde}} \sum_{\boldsymbol{x} \in \mathcal{T}_{pde}} ||f\left(\boldsymbol{x}; \frac{\partial \hat{u}}{\partial x_1}, \cdots, \frac{\partial \hat{u}}{\partial x_d}; \frac{\partial^2 \hat{u}}{\partial x_1 \partial x_1}, \cdots, \frac{\partial^2 \hat{u}}{\partial x_1 \partial x_d}; \cdots\right)||^2_{L^2(\Omega)} \\ &+ \omega_{ib} \cdot \frac{1}{N_{ib}} \sum_{\boldsymbol{x} \in \mathcal{T}_{ib}} ||\mathcal{B}(\hat{u}; \boldsymbol{x})||^2_{L^2(\partial\Omega)}, \end{aligned} \tag{4}$$

where ω_{pde} and ω_{ib} are the weights. Those derivatives in the loss function (4) are evaluated by the automatic differentiation (AD) provided by Tensorflow (Abadi et al., 2016), one of the most popular and well documented open source libraries for machine learning computations.

Step 4 Train the neural network to search for the best parameters θ_* by minimizing the highly nonlinear loss function $\mathcal{L}(\theta)$. The commonly used method is the gradient based optimizers such as Adam and L-BFGS (cf. Ch.8, Goodfellow 2016).

3 Option Pricing with Black–Scholes Equation

We implement all option pricing simulations by using the DeepXDE package, which runs under "Colab" from Google Research on our Dell XPS Notebook (with Intel Core i5-1035G1 1.19 GHz CPU and 8 GB RAM) running Windows 10.

3.1 One-Dimensional Linear Case

We first consider solving the following standard Black–Scholes equation p.257, (Higham, 2004):

$$\frac{\partial V}{\partial \tau} - \frac{\sigma^2 S^2}{2}\frac{\partial^2 V}{\partial S^2} - rS\frac{\partial V}{\partial S} + rV = 0, \quad \text{on } (0,\infty)\times(0,T], \tag{5}$$

where parameters r and $\sigma > 0$ stand for the risk free interest rate and the volatility, respectively, $V(S,\tau)$ represents the fair value of a vanilla European option at time τ with asset value S at that time. Here T denotes the expiration date of the contract, $\tau = T - t$ denotes the time to maturity, and $t \in [0,T]$ is the instantaneous time. Hence the option payoff function becomes the initial condition for the PDE.

In this example, we solve the Black–Scholes equation (5) for an European put option subject to the initial condition

$$V(S,0) = \max(K - S, 0), \tag{6}$$

and the boundary condition

$$V(0,\tau) = Ke^{-r\tau}, \quad V(L,\tau) = 0, \tag{7}$$

where K is the strike price, and L is some suitably large value by truncating the original $(0,\infty)$ domain to $(0, L]$. The exact value of a European put option is given as Ch.8, (Higham, 2004):

$$V(S,\tau) = Ke^{-r\tau}(1 - N(d_2)) + (N(d_1) - 1)S, \tag{8}$$

where

$$d_1 = \frac{\log(S/K) + (r + 0.5\sigma^2)\tau}{\sigma\sqrt{\tau}}, \; d_2 = \frac{\log(S/K) + (r - 0.5\sigma^2)\tau}{\sigma\sqrt{\tau}},$$

and $N(z)$ is the cumulative distribution function of the normally distributed random variable z defined as follows:

$$N(z) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{z} e^{-\frac{\xi^2}{2}} d\xi. \tag{9}$$

We solve this model problem with parameters

$$K = 4,\ \sigma = 0.3,\ r = 0.03,\ T = 1,\ L = 10,\ N_x = 50,\ N_t = 500$$

which are exactly the same as those given in Higham (2004, p. 264), where several finite difference schemes are developed to solve this Black–Scholes model.

We use 2540 training residual points sampled inside the spatio-temporal domain, 80 training points sampled on the boundary, and 160 residual points for the initial conditions. The neural network depth is 3 and width is 20. The activation function is the hyperbolic tangent (tanh). We first use Adam for 100 iterations with learning rate 0.001, and then switch to L-BFGS. The L-BFGS does not require learning rate, and the neural network is trained until convergence. The train loss for 'Adam' is 3.47e-01, and 'train' took 1.66 seconds; while the train loss for 'L-BFGS' is 6.02e-05, and 'train' took 46.10 seconds.

In Fig. 1, we plot the numerical solution at the final time $\tau = T$, the corrsponding pointwise error (calculated as predicted solution - exact solution), the solution over the whole time [0, T], and the training history. The maximum pointwise error (i.e., numerical solution - exact solution) obtained by our scheme is 3.0e-03, which is compatiable with 1.5e-03 obtained by the FTCS scheme of Higham (2004, p. 260). Their FTCS scheme is built with forward Euler in time and second order in spatial approximation.

Of course, we can improve the accuracy by using more training points and wider and deeper neural networks. We resolve this model by doubling all training points (i.e, 5080 points inside the spatio-temporal domain, 160 points on the boundary, and 320 residual points for the initial conditions. The neural network depth is 5 and width is 30. The rest parameters are the same. Now we obtain the train loss for 'Adam' 2.58e-02, and 'train' took 10.28 seconds; while the train loss for 'L-BFGS' is 4.74e-06, and 'train' took 280.58 seconds.

In Fig. 2, we plot the numerical solution at the final time $\tau = T$, the corrsponding pointwise error, the solution over the whole time [0, T], and the training history. From Fig. 2, we see that the maximum pointwise error is reduced to 4.2e-04, which is smaller than 9.5178e-04 obtained in our previous work by a fourth-order difference scheme (cf. Example 2 of Wang, 2021), and much better than 1.5e-03 obtained by the FTCS scheme of (Higham, 2004, p. 260).

3.2 One-dimensional nonlinear case

Let us consider the nonlinear Black–Scholes equation (Lesmana et al., 2013):

$$\frac{\partial V}{\partial \tau} - \frac{1}{2}\sigma^2(\tau, S, \frac{\partial V}{\partial S}, \frac{\partial^2 V}{\partial S^2})S^2\frac{\partial^2 V}{\partial S^2} - rS\frac{\partial V}{\partial S} + rV = 0,\ \text{on } (0,\infty)\times(0,T], \tag{10}$$

where the volatility σ is a function of time τ, the underlying stock price S, the Delta $\frac{\partial V}{\partial S}$, and the Gamma $\frac{\partial^2 V}{\partial S^2}$.

Different nonlinear models have been proposed in recent years to accomodate the transaction costs arising in trading. For simplicity, we just consider the HWW transaction cost model (Wilmott et al., 1994), which assumes that the nonlinear volatility is given by

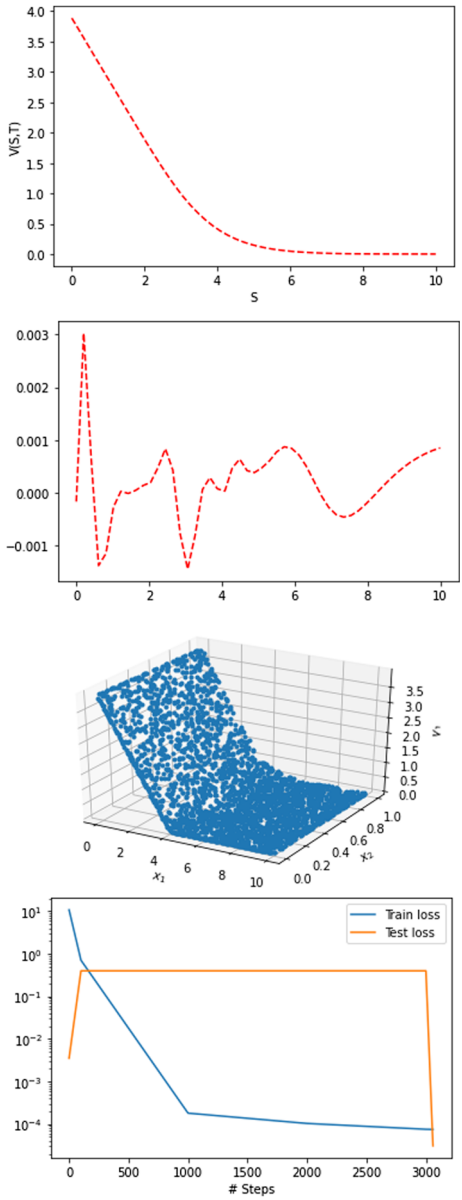

Fig. 1 Example 1. Plots of the numerical solution at the final time T (Top); the corresponding pointwise error at the final time T (Midde Top: horizonal axis is for S); the numerical solution over the whole time $[0, T]$ (Middle Bottom: x_1 is for S, and x_2 is for τ); the training history (Bottom)

$$\sigma^2 = \sigma_0^2\left(1 - K_\sigma \cdot sign\left(\frac{\partial^2 V}{\partial S^2}\right)\right), \quad \sigma^2 = \sigma_0^2\left(1 + K_\sigma \cdot sign\left(\frac{\partial^2 V}{\partial S^2}\right)\right), \tag{11}$$

for call and put options, respectively. Here we denote $sign(\cdot)$ for the standard sign function, the parameter $K_\sigma = \frac{\kappa}{\sigma_0}\sqrt{\frac{8}{\pi \delta t}}$, where δt is the fixed trading frequency, and κ denotes the transaction cost measure.

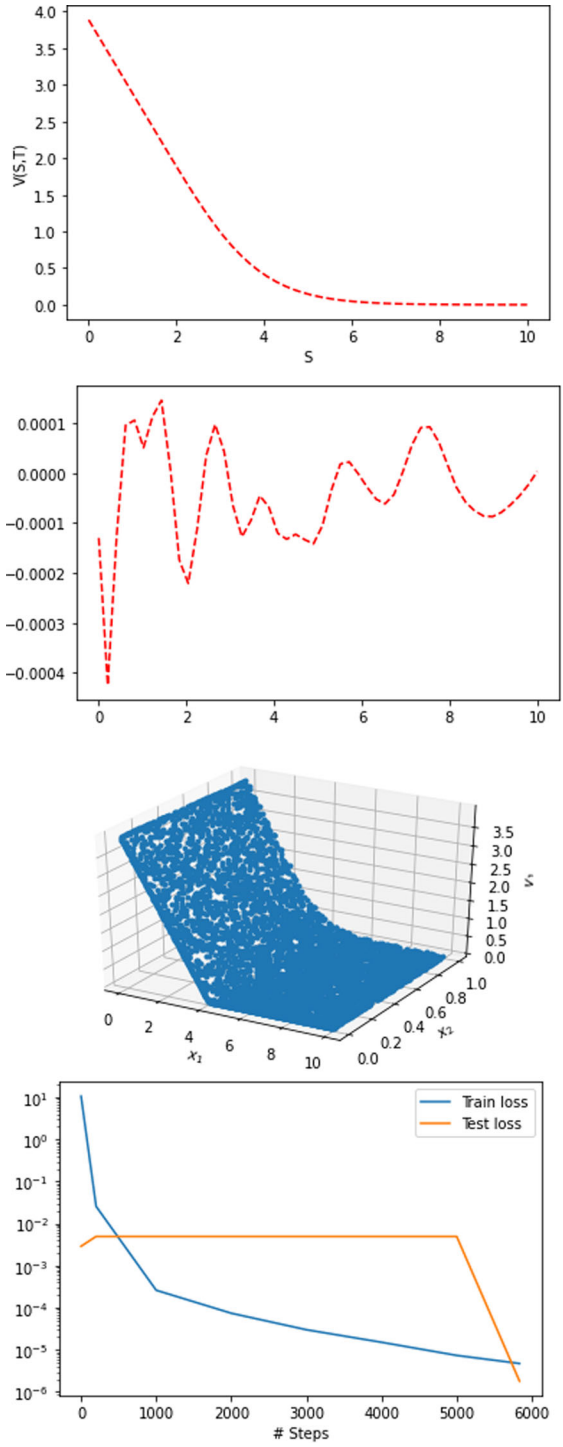

Fig. 2 Example 1 (Refined results). Plots of the numerical solution at the final time T (Top); the corresponding pointwise error at the final time T (Middle Top: horizonal axis is for S); the numerical solution over the whole time $[0, T]$ (Middle Bottom: x_1 is for S, and x_2 is for τ); the training history (Bottom)

First, we solve the nonlinear Black–Scholes equation (10) with HWW's put volatility with the same parameters as given in Test 2 of Lesmana et al. (2013):

$$r = 0.1,\ K = 40,\ T = 1,\ S \in [0, 80], \tag{12}$$

where the HWW parameters are $\sigma_0 = 0.2, \delta t = \frac{1}{12}$ (i.e., option trades monthly) and $\kappa = 0.02$.

We use 2540 training residual points sampled inside spatio-temporal domain, 80 training points sampled on the boundary. and 160 residual points for the initial conditions. The neural network depth is 3 and width is 20. Similar to last example, we first use Adam for 100 iterations with learn rate 0.001, and then switch to L-BFGS. For this example, the train loss for 'Adam' is 6.22e+02, and 'train' took 2.29 seconds; while the train loss for 'L-BFGS' is 2.14e-03, and 'train' took 96.78 seconds.

In Fig. 3, we present the numerical solution at final time T, the solution over the whole time [0, T], and the training history. Our numerical solution presented in Fig. 3 has no obvious difference from no different from Fig. 8 of Lesmana et al. (2013) and Fig. 4 of Wang et al. (2021) obtained by a high order finite difference scheme.

Next, we solve (10) with HWW's put volatility and the same parameters as Test 3 of (Lesmana et al. 2013):

$$r = 0.1,\ \sigma_0 = 0.2,\ E_1 = 30,\ E_2 = 40,\ E_3 = 50,\ T = 1,\ S \in [0, 80], \tag{13}$$

where E_1, E_2 and E_3 are the strike prices of the option. This example is set for the so-called Butterfly Spread option, which has the initial condition:

$$V(S, 0) = \max(S - E_1, 0) - 2\max(S - E_2, 0) + \max(S - E_3, 0), \tag{14}$$

and the boundary conditions

$$V(0, \tau) = V(80, \tau) = 0. \tag{15}$$

For this problem, we use 5000 training points sampled inside the spatio-temporal domain, 160 training points sampled on the boundary, and 320 residual points for the initial conditions. The neural network depth is 3 and width is 20. Similarly, we first use Adam for 100 iterations with learn rate 0.001, and then switch to L-BFGS. The train loss for 'Adam' is 6.80e+00, and 'train' took 2.14 seconds; while the train loss for 'L-BFGS' is 4.29e-02, and 'train' took 56.62 seconds.

We plot the numerical solution at final time T, the solution over the whole time [0, T], and the training history in Fig. 4. Our numerical solution presented in Fig. 4 has no visual difference from Test 3 of Lesmana et al. (2013) and Fig. 7 of Wang et al. (2021) obtained by the high order finite difference scheme.

4 Two-asset options

In this section, we consider the two-asset option pricing problem, which is governed by the following PDE (Seydel 2012, p. 250):

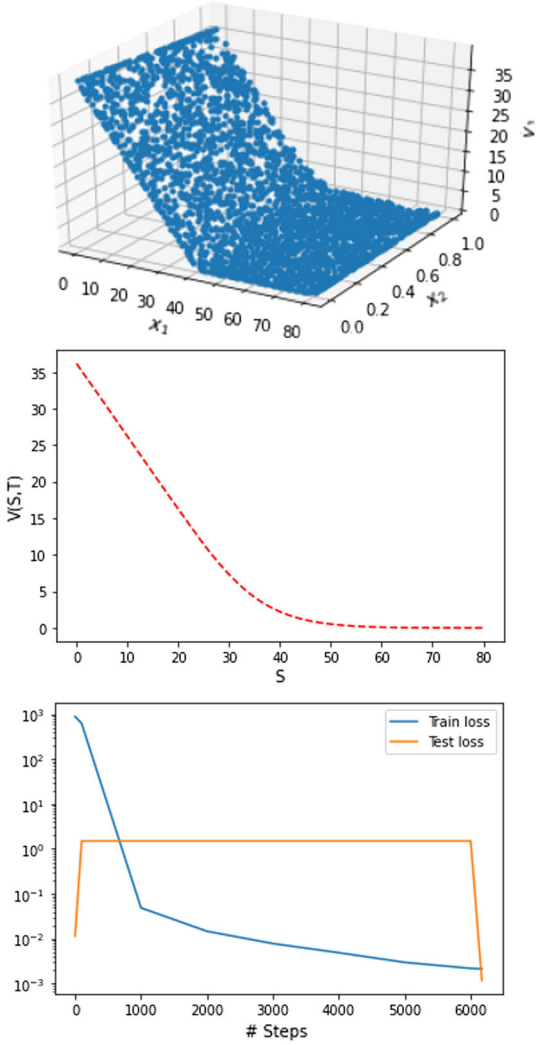

Fig. 3 Example 2. Plots of the numerical solution over the whole time $[0, T]$ (Top: x_1 is for S, and x_2 is for τ); the numerical solution at final time $\tau = T$ (Middle); the training history (Bottom)

$$\frac{\partial V}{\partial \tau} - \frac{\sigma_1^2 S_1^2}{2}\frac{\partial^2 V}{\partial S_1^2} - (r-\delta_1)S_1\frac{\partial V}{\partial S_1} + rV$$
$$- \frac{\sigma_2^2 S_2^2}{2}\frac{\partial^2 V}{\partial S_2^2} - (r-\delta_2)S_2\frac{\partial V}{\partial S_2} - \rho\sigma_1\sigma_2 S_1 S_2\frac{\partial^2 V}{\partial S_1 \partial S_2} = 0, \quad \text{on } (0,\infty)^2 \times (0,T], \tag{16}$$

where the option value $V(S_1, S_2, \tau)$ depends on the asset prices S_i $(i = 1, 2)$ and the time to maturity τ. The parameter r is the interest rate, σ_i and δ_i $(i = 1, 2)$ denote the volatilities and the dividend rates for the two assets, respectively, and ρ represents the correlation coeffcient between S_1 and S_2.

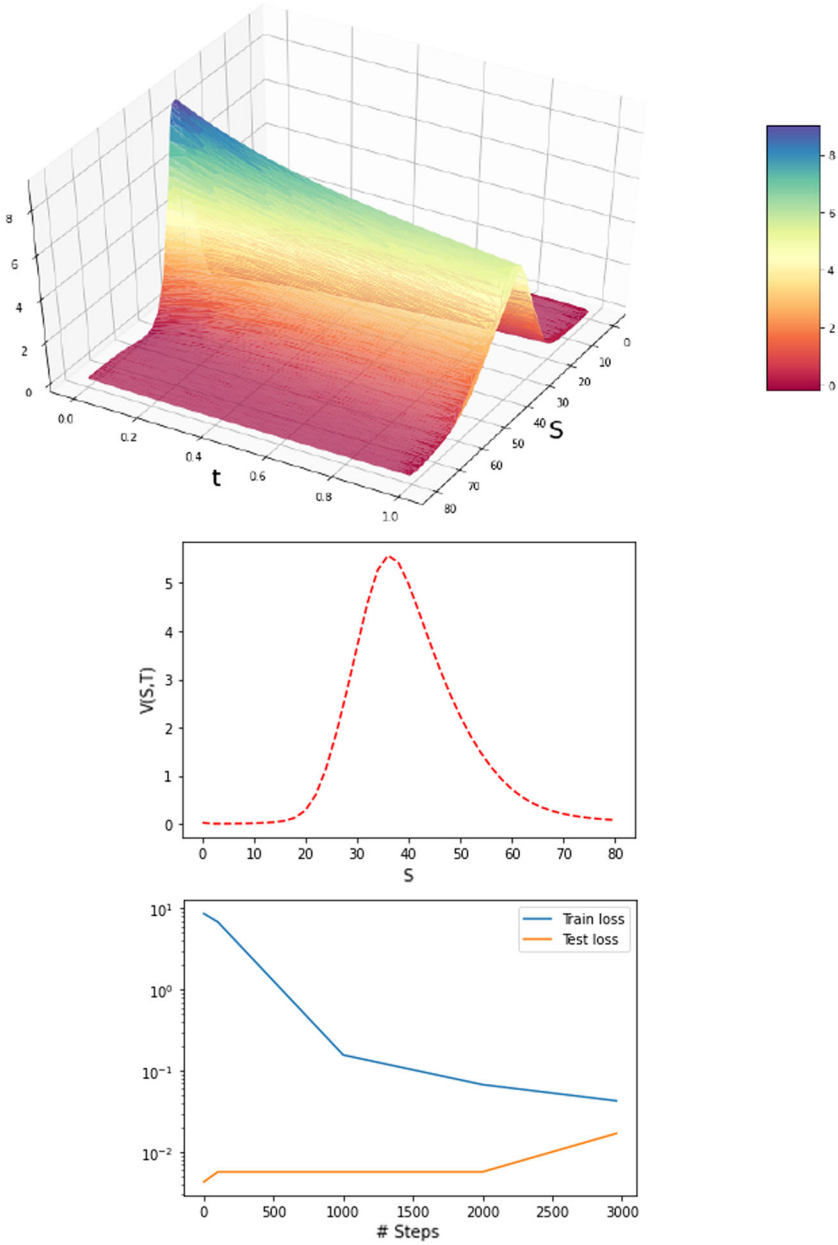

Fig. 4 Example 3. Plots of the numerical solution over the whole time $[0, T]$ (Top: x_1 is for S, and x_2 is for τ); the numerical solution at final time $\tau = T$ (Middle); the training history (Bottom)

4.1 Cash-or-nothing put option

For this two-asset cash-or-nothing put option, the payoff function is

$$V(S_1, S_2, 0) = \begin{cases} C, & \text{if } S_1 < K_1 \text{ and } S_2 < K_2, \\ 0, & \text{otherwise} \end{cases} \tag{17}$$

where $C > 0$ is a fixed cash amount. For comparison, we choose the same model parameters as Example 3.8 of Seydel (2012):

$$K_1 = K_2 = 5,\ T = 1, \sigma_1 = 0.2,\ \sigma_2 = 0.3,\ \rho = 0.3,\ r = 0.1,\ \delta_1 = \delta_2 = 0,\ C = 1.$$

The physical domain $0 < S_1, S_2 < \infty$ is truncated to a bounded rectangle: $0 \le S_1 \le x_{max} := 10$, and $0 \le S_2 \le y_{max} := 10$. Dirichlet boundary conditions $V = 0$ are imposed on $S_1 = x_{max}$ and $S_2 = y_{max}$. On $S_2 = 0$, the boundary condition is chosen as the one-dimensional European binary put, which is

$$V(S_1, 0, \tau) = Ce^{-r\tau}N(-d_1), \quad \text{where } d_1 = \frac{\log(S_1/K_1) + (r - \sigma^2/2)\tau}{\sigma\sqrt{\tau}},$$

where $N(z)$ is the cumulative distribution function defined in (9). By the same argument, the boundary condition on $S_1 = 0$ is given by

$$V(0, S_2, \tau) = Ce^{-r\tau}N(-d_2), \quad \text{where } d_2 = \frac{\log(S_2/K_2) + (r - \sigma^2/2)\tau}{\sigma\sqrt{\tau}}.$$

To solve this model, we use 20000 training residual points sampled inside the spatio-temporal domain, 2000 training points sampled on the boundary. and 1000 residual points for the initial conditions. The neural network depth is 5 and width is 20. Simlilar to previous examples, we use the hyperbolic tangent (tanh) as the activation function. As the optimizer, we first use 'Adam' for 1000 iterations with learning rate 0.001, and then switch to L-BFGS. The train loss we obtained for 'Adam' is 2.27e-02, and 'train' took 196.22 seconds; while the train loss fo r 'L-BFGS' is 1.91e-03, and 'train' took 1085.93 seconds.

The obtained numerical solution is plotted in Fig. 5, which looks no visual difference from Fig. 3.8 of Seydel (2012). To compare more carefully, we extract our numerical solution at point (5, 5), which is $V(5, 5, T) = 0.1756$ and is comparable to 0.174 given in Seydel (2012, p. 133).

4.2 Double knock-out barriers call option

We consider an European call on a two-asset basket with portfolio value $S_1 + S_2$, which leads to a payoff function

$$V(S_1, S_2, 0) = \max(S_1 + S_2 - K, 0), \tag{18}$$

where K denotes the strick price. For the double knock-out barriers case: down-and-out at L, and up-and-out at U, where L and U are the lower and upper barriers, respectively. In this case, the physical domain becomes a trapezoid, which is bounded by lines $S_1 + S_2 = L$, $S_1 + S_2 = U$, $S_1 = 0$ and $S_2 = 0$. This leads to the boundary condition $V(S_1, S_2, t) = 0$ on both boundaries $S_1 + S_2 = L$ and $S_1 + S_2 = U$. The boundary conditions on $S_1 = 0$ and $S_2 = 0$ can be obtained by

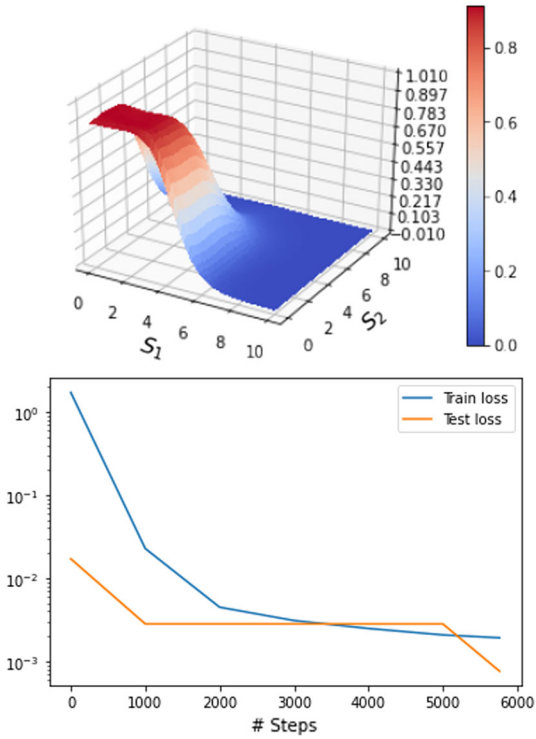

Fig. 5 Example 3. Plots of the numerical solution at the final time T (Top); the training history (Bottom)

setting $S_1 = 0$ and $S_2 = 0$ in (16), respectively. Each becomes a single-asset double knock-out call option problem governed by the one-dimensional Black–Scholes equation (5). Here we adopt an analytic formula derived by Pelsser (1997). Let us denote the drift $\mu = r - \frac{1}{2}\sigma^2$ and introduce the following notations

$$d = \log\frac{K}{L}, \quad l = \log\frac{U}{L}, \quad x = \log\frac{S}{L}, \quad \lambda_k = \frac{1}{2}\left(\frac{\mu^2}{\sigma^2} + \frac{k^2\pi^2\sigma^2}{l^2}\right),$$

then the value of the double knock-out call is given by Pelsser (1997, cf. (32)):

$$V_c(x,\tau) = e^{-r\tau}[(Q(x;1,l) - Q(x;1,d))L - (Q(x;0,l) - Q(x;0,d))K], \tag{19}$$

where the function $Q(x;\alpha,y)$ is

$$Q(x;\alpha,y) = \frac{2}{l}e^{\frac{\mu}{\sigma^2}(y-x)}e^{\alpha y}\sum_{k=1}^{\infty} e^{-\lambda_k\tau}\sin\frac{k\pi x}{l}\cdot\frac{\left(\frac{\mu}{\sigma^2}+\alpha\right)\sin\frac{k\pi y}{l} - \frac{k\pi}{l}\cos\frac{k\pi y}{l}}{\left(\frac{\mu}{\sigma^2}+\alpha\right)^2 + \left(\frac{k\pi}{l}\right)^2}. \tag{20}$$

Due to the fast convergence of the series Q, in practical application we only need to choose the first five terms. With (19), the boundary condition on $S_2 = 0$ is given by $V_c(S_1,\tau)$; while on $S_1 = 0$, the boundary condition is given by $V_c(S_2,\tau)$.

For this example, we choose the PDE parameters exactly the same as Example 5.5 of Seydel (2012):

$$K = 1,\ T = 1, \sigma_1 = \sigma_2 = 0.25,\ \rho = 0.7,\ r = 0.05, \tag{21}$$

$$\delta_1 = \delta_2 = 0,\ L = 1,\ U = 2. \tag{22}$$

We use 10000, 1000 and 200 training residual points sampled inside the spatio-temporal domain, on the boundary. and for the initial conditions, respectively. The neural network depth is 5 and width is 20. Again, the activation function is the hyperbolic tangent, and we use 'Adam' for 1000 iterations with learning rate 0.001, and then switch to the L-BFGS optimizer.

The obtained numerical solution is plotted in Fig. 6, which looks very similar to Fig. 5.11 of Seydel (2012). To compare more carefully, we extract our numerical solution at point (1.25, 0.25), which is $V(1.25, 0.25, T) = 0.2955$ and is comparable to 0.2949 obtained in Seydel (2012, p. 259). The train loss for 'Adam' is 6.50e-02, and 'train' took 93.78 seconds; while the train loss for 'L-BFGS' is 1.79e-04, and 'train' took 452.19 seconds.

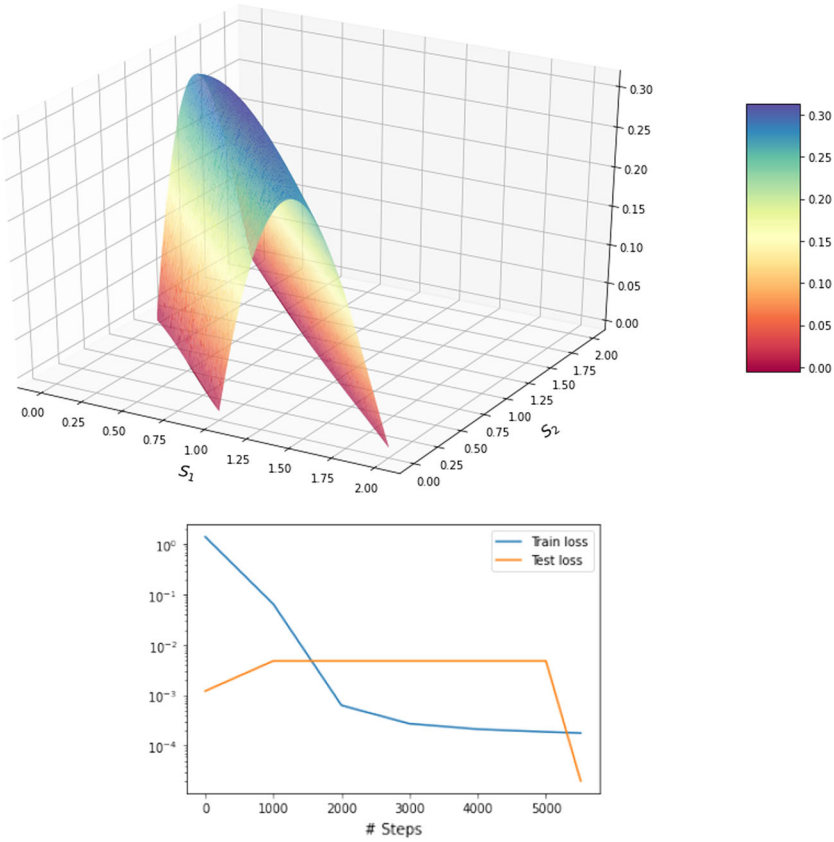

Fig. 6 Example 4. Plots of the numerical solution at the final time T (Top); the training history (Bottom)

5 Conclusion

In this paper, we extended the Physics-Informed Neural Networks (PINNs) method introduced by Raissi et al. (2019) to solve many option pricing PDE models. Our tests demonstrate that the PINNs deep learning method can produce very accurate results comparable to those classical numerical PDE methods. Compared to the classical numerical PDE methods such as finite difference and finite element methods etc (cf., Li and Chen, 2019) and references therein), the PINNs method is very simple in implementation and can achieve very accurate solution within amazing computational time. In the future, we will continue our exploration using PINNs method to to solve high-dimensional and more complicated option pricing PDEs.

Acknowledgements Jichun Li would like to thank UNLV for granting his sabbatical leave during Spring 2021 so that he could enjoy his time working on this paper. We also like to thank four anonymous reviewers for their insightful comments that improved this paper.

Funding Work partially supported by National Natural Science Foundation of China under Grants No. 11961048, NSF of Jiangxi Province with No.20181ACB20001, and National Science Foundation under Grant No. DMS-2011943.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., M. Devin, S. Ghemawat, G. Irving, & M. Isard et al. (2016). TensorFlow: A system for large-scale machine learning. In *12th USENIX symposium on operating systems design and implementation*, pp. 265–283.

Arin, E., & Ozbayoglu, A. M. (2020). Deep learning based hybrid computational intelligence models for options pricing. *Computational Economics*. https://doi.org/10.1007/s10614-020-10063-9

Barucci, E., Cherubini, U., & Landi, L. (1997). Neural networks for contingent claim pricing via the Galerkin method. In H. Amman, B. Rustem, & A. Whinston (Eds.), *Computational approaches to economic problems* (pp. 127–141). Springer.

Black, F., & Scholes, M. (1973). The pricing of options and corporate liabilities. *Journal of Political Economy, 81*, 637–659.

Dilloo, M. J., & Tangman, D. Y. (2017). A high-order finite difference method for option valuation. *Computers & Mathematics with Applications, 74*(4), 652–670.

Eskiizmirliler, S., Günel, K., & Polat, R. (2021). On the solution of the Black–Scholes equation using feed-forward neural networks. *Computational Economics, 58*, 915–941.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press.

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

Higham, D. J. (2004). *An introduction to financial option valuation*. Cambridge University Press.

Hutchinson, J. M., Lo, A. W., & Poggio, T. (1994). A nonparametric approach to pricing and hedging derivative securities via learning networks. *The Journal of Finance, 49*(3), 851–889.

Koffi, R. S., & Tambue, A. (2020). A fitted multi-point flux approximation method for pricing two options. *Computational Economics, 55*, 597–628.

Lagaris, I., Likas, A., & Papageorgiou, D. (2000). Neural-network methods for boundary value problems with irregular boundaries. *IEEE Transactions on Neural Networks, 11*(5), 1041–1049.

Lee, S. T., & Sun, H. (2012). Fourth-order compact scheme with local mesh refinement for option pricing in jump-diffusion model. *Numerical Methods for Partial Differential Equations, 28*, 1079–1098.

Lesmana, D. C., & Wang S, S. (2013). An upwind finite difference method for a nonlinear Black–Scholes equation governing European option valuation under transaction costs. *Applied Mathematics and Computation, 219,* 8811–8828.

Li, J., & Chen, Y.-T. (2019). *Computational partial differential equations using MATLAB* (2nd ed.). CRC Press.

Li, J., & Nan, B. (2019). Simulating backward wave propagation in metamaterial with radial basis functions. *Results in Applied Mathematics*, 2.

Liao, W., & Khaliq, A. Q. M. (2009). High-order compact scheme for solving nonlinear Black–Scholes equation with transaction cost. *International Journal of Computer Mathematics, 86*(6), 1009–1023.

Lin, S., & Zhu, S. P. (2020). Numerically pricing convertible bonds under stochastic volatility or stochastic interest rate with an ADI-based predictor-corrector scheme. *Computers & Mathematics with Applications, 79*(5), 1393–1419.

Lu, L., Meng, X., Mao, Z., & Karniadakis, G. E. (2021). DeepXDE: A deep learning library for solving differential equations. *SIAM Review, 63*(1), 208–228.

Malek, A., & Beidokhti, R. (2006). Numerical solution for high order differential equations using a hybrid neural network-optimization method. *Applied Mathematics and Computation, 183*(1), 260–271.

Mollapourasl, R., Fereshtian, A., & Vanmaele, M. (2019). Radial basis functions with partition of unity method for American options with stochastic volatility. *Computational Economics, 53*(1), 259–287.

Pelsser, A. (1997). *Pricing double barrier options: An analytical approach, discussion paper TI 97–015/2, 1997*. Tinbergen Institute.

Raissi, M., Perdikaris, P., & Karniadakis, G. E. (2019). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics, 378*, 686–707.

Ruf, J., & Wang, W. (2020). Neural networks for option pricing and hedging: A literature review. *Journal of Computational Finance, 24*(1), 1–46.

Seydel, R. U. (2012). *Tools for computational finance* (5th ed.). Springer.

Sirignano, J., & Spiliopoulos, K. (2018). DGM: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics, 375*, 1339–1364.

Soleymani, F., & Zhu, S. (2021). RBF-FD solution for a financial partial-integro differential equation utilizing the generalized multiquadric function. *Computers & Mathematics with Applications, 82*(15), 161–178.

Wang, X., Li, J., & Li, J. (2021). High order approximation of derivatives with applications to pricing of financial derivatives. *Journal of Computational and Applied Mathematics*, 398.

Wang, S., Zhang, S., & Fang, Z. (2015). A superconvergent fitted finite volume method for Black–Scholes equations governing European and American option valuation. *Numerical Methods for Partial Differential Equations, 31*, 1190–1208.

Wilmott, P., Hoggard, T., & Whalley, A. W. (1994). Hedging option portfolios in the presence of transaction costs. *Advances in Futures and Options Research, 7*(4), 21–35.

Zhu, Y.-L., Wu, X., & Chern, I.-L. (2004). *Derivative securities and difference methods*. Springer.

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.