Engineering Analysis with Boundary Elements 151 (2023) 68–82

Contents lists available at ScienceDirect

Engineering Analysis with Boundary Elements

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

Meshless methods for American option pricing through Physics-Informed Neural Networks

Federico Gatta [a,*], Vincenzo Schiano Di Cola [b], Fabio Giampaolo [b], Francesco Piccialli [b], Salvatore Cuomo [b]

[a] Scuola Normale Superiore, Pisa, Italy
[b] Department of Mathematics and Applications, University of Naples Federico II, Naples, Italy

ARTICLE INFO

Keywords:
Finance
Black–Scholes
PINN
Free boundary problem
Deep learning
Parametric model

ABSTRACT

Nowadays, Deep Learning is drastically revolutionizing financial research as well as industry. Many methods have been discussed in the last few years, mainly related to option pricing. In fact, traditional approaches such as Monte Carlo simulation or finite difference methods are seriously harmed by multi-dimensional underlying and path dependency. Thus, dealing with particular contracts such as American multi-asset options is still rough. This paper addresses such a problem by pricing said put options with a novel meshless methodology, named Physics-Informed Neural Networks (PINNs), based on Artificial Intelligence. PINN paradigm has been recently introduced in Deep Learning literature. It exploits the theoretical background of the universal approximation theorem for neural networks to solve Partial Differential Equations numerically. This Deep Learning meshless method incorporates the equation and its initial and boundary conditions thanks to a specially designed loss function. We develop a suitable PINN for the proposed problem by introducing an algorithmic trick for improving the convergence of the free boundary problem. Furthermore, the worthiness of the proposal is assessed by several experiments concerned with single and multi-asset options. Finally, a parametric model is built to benefit further studies of option value behaviour related to particular market conditions.

1. Introduction

Nowadays, Finance dramatically benefits from the introduction of Artificial Intelligence and Deep Learning methodologies. In fact, the growth of financial data gathered through the years and the subsequent development of considerable literature concerned with their analysis has supported the growth of extensive literature in this field. Furthermore, this subject has attracted researchers and practitioners with different backgrounds, such as computer scientists, mathematicians or statisticians, thanks to its strong interconnections with the real-world financial industry. In particular, among the several topics discussed in the last years, we deal with numerical methods for option pricing.

Option pricing is concerned about the study and development of different type methods to estimate the fair value (under some particular assumptions) of financial *option contracts*, or simpler *options*, based on current market conditions (such as asset price and interest rate). Although there are different types of financial options, the most important styles are European and American. Both of them give the holder the right to do a specific action in the future. In particular, a *call* option gives the holder the right to buy a specific (underlying) asset at a fixed price, while a *put* option gives the right to sell. The option style is considered European if the holder can exercise such right only at option maturity. Otherwise, if such right is exercisable whenever until maturity (early exercise), the option is said to be American. In other words, European options look only to maturity price, when American ones also consider price evolution from the contract start to maturity. This characteristic is also known as *path dependency*. Despite being quite similar from a theoretical point of view, the two types are very different from a mathematical point of view. In particular, under the so-called *Black–Scholes model*, for European options, it is possible to obtain an explicit pricing formula [1]. Instead, no explicit solution exists for American options, and their pricing strategies principally rely on numerical methods.

In this paper, we deal with multi-asset American put options. We select this particular kind of contract as it has been shown that, under some general hypothesis, an American call is equivalent to the European one. Nonetheless, put options have great practical relevance in that they are also used for insurance purposes [2]. Moreover, multi-asset options are not just a mere theoretical exercise, but they are

* Corresponding author.
E-mail address: federico.gatta@sns.it (F. Gatta).

https://doi.org/10.1016/j.enganabound.2023.02.040
Received 11 September 2022; Received in revised form 11 February 2023; Accepted 21 February 2023
Available online 7 March 2023

worth studying in that they also introduce diversification for derivative contracts [3]. We apply a *Partial Differential Equations* (PDE) method to reach our goal. That is, we compute the price function as the solution of a PDE. We work on the framework of the Black–Scholes model, where a parabolic nonlinear PDE is used to describe option price dynamics. Furthermore, specific initial and boundary conditions are exploited to represent some contract features mathematically, e.g. call or put, European or American. More specifically, the American put option pricing problem is a free boundary problem with Dirichlet and Neumann conditions. The approaches developed to solve this kind of PDEs can be clustered into several groups. Among them, Finite Difference Methods (FDMs), Finite Element Methods (FEMs), and Finite Volume Methods (FVMs) are numerical schemes generally adopted [4].

In this paper, the option pricing problem is addressed by applying a recent Deep Learning strategy for the numerical resolution of PDE, namely the Physics-Informed Neural Networks (PINNs) [5]. PINNs exploit neural networks to approximate the solution and incorporate the physics of the problem by carefully designing the loss function, which is made up in such a way to represent the PDE, the initial, and the boundary conditions as well. Thus, we aim to overcome the standard limitations of the well-established methods in the literature, particularly the difficulties in handling path dependency and multiple dimensions, by relying on a powerful universal approximator, the neural networks. In fact, as stated by the universal approximation theorem [6,7], conveniently designed networks can approximate any continuous function on a compact domain. Further details about PINNs are provided in the following sections, which examine this topic in depth. As for the experimental stage, we carry out different experiments on one-dimensional and multi-dimensional equations. That is, we consider options with one and multiple underlying assets. Furthermore, we also develop a parametric model that could be useful for simulations with different market conditions or to extract implicit parameters from real, observed market prices.

Numerical Methods for Option Pricing Historically, the most widely used methods for American option pricing are *binomial models*, *Monte Carlo simulations*, and *FDMs*. Binomial models exploit binomial trees to approximate asset price dynamics over time and evaluate the option price. These models create a partition on the time interval and represent asset price movements as jumps from one time in the partition to another. Each jump can assume only two values, either up or down, of fixed magnitude. Although these models have been historically used for pricing single asset options, there are also some attempts to generalize pricing formulas for multi-dimensional options, such as [8]. On the other hand, Monte Carlo methods evaluate option price by simulating several possible paths and then averaging the option value in each simulation [9]. Moreover, this approach makes it possible also to compute other quantities related to the price estimator, such as its variance. Furthermore, it can easily work also for multi-dimensional options. In contrast, it exhibits a big drawback in handling path dependency, so they are not naturally suited for American options [10]. As for the FDMs, they have been introduced to overcome some limitations of binomial models and Monte Carlo simulations, especially related to inaccuracies when working with high volatility assets [11]. The FDMs are numerical methods designed to solve PDE. By solving appropriate PDEs whose unknown function is the option price, these methods can evaluate options as a function of variables such as time and underlying asset value. Usually, the Black–Scholes equation is considered. As binomial models, also FDMs rely on time interval partitioning. Then, the derivatives are approximated with finite differences, and the starting PDE is turned into a linear system and finally solved. Methods of this type are widely used in financial engineering at whole [12]. Specifically concerned with option pricing, there is [13]. In [13], the author studies American put options with one and two underlying assets by using *explicit* and *implicit* finite difference schemes. Furthermore, the author test the Brennan–Schwartz model [14], which is a log transformation of the standard Black–Scholes model in such a way to obtain constant coefficients.

Despite their wide diffusion in the early stage, it has been reported that FDMs have drawbacks when working with multiple dimensions [10]. So, further research has been carried out on other model classes. One example are the *FEMs* [15]. In contrast to finite difference, FEMs exploit a variational formulation of the problem to approximate the solution. Specifically, the whole domain is partitioned into subregions, where the equation can be locally approximated. Then, such local approximations are gathered together to find the overall numerical solution. An application of this method can be found, for example, in [2]. In this work, the author exploits a particular class of finite difference method, namely the method of lines, to price American put option defined on multiple underlying assets. In this case, the geometric mean of asset values is considered for the computation of the payoff function. Furthermore, a comparison with a finite difference method is carried out.

Meshless and Deep Learning Methods in Finance All the approaches shown till now are also referred to as *Mesh-based* methods, in that they need the construction of a specific *mesh*. That is, two steps are required: (i) the generation of grid points or elements, usually obtained as the discretization of differential formulas (ii) the solution of discrete equations, often by using iterative algorithms. Unfortunately, these numerical methods have several issues to be addressed. For example, we can mention the shape complexity of the computational domain where the grid generation itself could become very difficult or even infeasible; moreover, the problem discretization could introduce a bias between the mathematical nature of the PDE and its approximating model. To overcome the difficulties and limitations related to the adoption of standard mesh-based approaches, recently novel meshless strategies, mainly based on Artificial Intelligence and Deep Learning methodologies, have been proposed [16].

For example, [17] propose the Deep Galerkin Method for high-dimensional PDEs. The proposal exploits a neural network to approximate the solution instead of basis functions as in the standard Galerkin method. Furthermore, the training points are re-sampled at each iteration so that there is no need for a fixed mesh. Instead, [18] introduces a novel iterative Deep Learning approach for semi-linear PDE based on Backward Stochastic Differential Equation (BSDE) representation and carries out an intensive experimental stage by comparing it with other methods in literature on different benchmarks equations, such as Black–Scholes one. A similar approach has been proposed in [19,20], where the case study is on a slightly modified version of the Black–Scholes equation to take into account also default possibility. Another approach is [21], which deals with the European and American option pricing problem by using radial basis point interpolation and different numerical tricks to improve accuracy. Similarly, also [22] exploits a meshless radial basis method for multi-asset European and American option pricing. Finally, [23] proposes another Deep Learning approach, which the authors claim can work up to 10000 dimensions. The method works by splitting PDE into linear and nonlinear parts, which are locally approximated for small time intervals. Then, a neural network is trained to return conditional expectations within each interval. Again, the Black–Scholes equation is one of the study cases for the proposal.

PINN Approaches in Finance The applications of PINNs for Finance are, to the best of our knowledge, still little studied and related to standard European option pricing. There are just a few works which address this problem. In particular, [24] deals with applying PINNs for European option pricing under the Black–Scholes and Heston models. This work faces both forward and inverse problems and shows the contribution of automatic differentiation in computing the Greeks. Furthermore, multi-asset options are studied, and the capability of PINNs to alleviate the course of dimensionality is emphasized. Another interesting contribution is [25]. In this case, the original problem addressed is the American call option pricing under the Black–Scholes model. Actually, it is assessed that the American call option pricing

problem can be turned into a European one if no dividends are paid by the underlying. So, the authors assume no dividends are paid and what they really solve is a European call option pricing problem. This study highlights the error growth in space regions close to the strike price or in time regions far away from the expiration date. The authors also solve these issues by introducing a supervised loss obtained by knowing the true solution of the problem. However, for the cases when no real solution is known, some ideas to overcome these limitations are found in a more conveniently weighted loss function. *Improved PINNs* (IPINNs) are proposed by [26] to improve standard PINNs in different aspects, mainly stability, accuracy and convergence speed. Both two methods are compared in the European option pricing problem under the Black–Scholes and Ivancevic models. The experimental results seem to confirm the worthiness of the IPINN method, even if several limitations have to be solved, as stated by the authors in their Conclusion. Finally, the authors in [27] combine a PINN-based approach with a Convolutional Transformer to predict the dynamics of the volatility surface. Transformers are a cutting-edge neural architecture based on Autoencoder that has shown promising performances in different fields, such as Natural Language Processing. The proposed methodology is compared against different benchmarks using European call options data.

Main Contribution The contributions of this paper can be summarized as follows:

- We apply a cutting-edge framework, namely the PINNs, to solve the American put option pricing problem from a numerical perspective. This innovative meshless method exploits neural networks flexibility to approximate PDEs solution. Furthermore, its applications in financial problems are still little studied.
- We introduce a novel algorithmic trick for free boundary PINN training. To the best of our knowledge, this strategy has not been used in the PINN context. It consists in performing multiple steps of the solution network against a single step of the free boundary. In this way, the convergence and the network's overall performance are enhanced.
- We assess the worthiness of our proposal in one and multi-dimensional cases in the experimental stage. Moreover, we also provide insight into a parametric model, which could greatly benefit the parametric study of the solution.
- We publicly share the code developed in the experimental stage to help researchers analyze the American option pricing problem deeper.

Roadmap The following of this paper is organized as follows. Section 2 provides an overview of PINNs, the main results in the literature, and the works that apply them in financial contexts. Section 3 describes the American put option problem we face up from a mathematical point of view by describing the PDE, the initial and boundary conditions, and the reasoning behind them. Section 4 formally describes the specific models we develop to price American options in uni and multi-dimensional frameworks. Furthermore, also an insight into parametric models is provided. Section 5 shows the experiments and results obtained. Finally, Section 6 concludes this work by highlighting the strengths and weaknesses of our proposal and by suggesting possible directions for further studies.

2. Background on PINNs

Big data availability supports Machine Learning (ML) solve challenging science and engineering problems. Despite numerous papers in the literature, there is still room for improvement for product-level ML applications. One of the most promising principles is to combine physical models and Machine Learning techniques. In this direction, [28] reviewed informed ML and explained how to assist the ML algorithm with physical knowledge. Their paper shows that physical knowledge can be mathematically represented in many ways, such as algebraic equations, differential equations, simulation results, logic rules, and probabilistic relations. Combining physics and Machine Learning can help accuracy and data efficiency (e.g., reducing data amount). Physics-informed Machine learning (PIML) based neural network was introduced in [5] and later reviewed in [29], where the authors presented the term PINN to indicate a Deep Learning framework for addressing forward and inverse problems involving nonlinear PDEs.

Deep Learning techniques are increasingly able to handle complex physics-driven problems; in fact, combining physical rules with Machine Learning techniques gives rise to PINN approaches that integrate PDEs into the neural network's loss function, constraining the training based on observable data or mathematical models. Still, the development and implementation of PINN require great processing capacity, as with the development of other AI systems. Different terminology such as science-informed neural networks, physics-inspired neural networks, and physics-constrained neural networks have been used to express similar ideas [30]. For consistency, we use Physics Informed Neural Network (PINN), although, in the context of Finance, we could call it Finance Informed Neural Networks (FINNs), which would reflect the fact that we could use fewer physics-inspired models and other information from human behaviours.

PINN generally requires choosing a network class $\mathcal{H}_n$ and loss function, given N-training data (called collocations points). Quantity and quality of training data affect $\mathcal{H}_n$; therefore, the goal is to minimize the loss by finding a $u_{\theta^*} \in \mathcal{H}_n$. However, even if $\mathcal{H}_n$ contains the exact PDE solution u, there is no guarantee that the minimization output u_{θ^*} and the solution u will coincide. The entire learning process of a PINN can be considered a statistical learning problem, and it involves mathematical foundations aspects for the error analysis [31].

Finally, as mentioned in [32,33], the global error between the trained deep neural network $\hat{u}^*_\theta$ and the right solution function u of a differential problem can be bounded by the sum of three errors an optimization error, a generalization error, and an approximation error:

$$\mathcal{R}[\hat{u}^*_\theta] \leq \mathcal{E}_O + 2\mathcal{E}_G + \mathcal{E}_A \tag{1}$$

where $\hat{(\cdot)}_\theta$ denotes a NN approximation achieved with parameters θ, selected among all the possible NN encoding Θ, and

$$\mathcal{R}[\hat{u}^*_\theta] \leq \underbrace{\widehat{\mathcal{R}}[\hat{u}^*_\theta] - \inf_{\theta\in\Theta} \widehat{\mathcal{R}}[u_\theta]}_{\text{Optimization error}} + \underbrace{2 \sup_{\theta\in\Theta} |\mathcal{R}[u_\theta] - \widehat{\mathcal{R}}[u_\theta]|}_{\text{Generalization error}} + \underbrace{\inf_{\theta\in\Theta} \mathcal{R}[u_\theta]}_{\text{Approximation error}}. \tag{2}$$

considering that $\widehat{\mathcal{R}}[u_\theta]$ is the empirical risk, which represents how effectively the NN predicts the exact value of the problem, and its continuum counterpart is the *risk* $\mathcal{R}[\hat{u}_\theta]$ of using an approximation $\hat{u}_\theta$, where the $L2$-norm is used to calculate the distance between the approximation and the solution u. Among all the possible networks used in literature for generating $\mathcal{H}_n$, here we recall some of the most used neural network frameworks, in particular Feedforward Neural Networks (FNN), Convolutional Neural Networks (CNN) or Recurrent Neural Networks (RNN). As for the optimizer used to minimize PINN's nonconvex and nonlinear loss functions, the most popular choices are Adam, RMSprop, and BFGS. An extensive review on such topics regarding the application, mathematical formalization, and network architectures can be found in [30].

3. Black–Scholes equation for American put option

This section describes the Black–Scholes equation for American put option pricing in univariate and multivariate settings. Furthermore, we discuss the initial and boundary conditions and also the assumptions we make.

3.1. Univariate setting

The first case we discuss is the univariate setting. In this situation, which is the most common one, we consider an option with only one underlying asset. Thus, there is only one spatial dimension.

3.1.1. The American put option pricing problem

In the univariate setting, an American put option is a contract which gives the possibility to sell a certain underlying asset at a fixed price K, also known as *strike*, in every moment in the time domain $\Omega_t = [0, T]$, where 0 is the present time and T is also referred to as the *maturity* of the option. Usually, the option price P is represented as a function of the underlying asset price S and the time t, while the other quantities, such as the strike price K or the maturity T, are considered as parameters. So, $P = P(S, t)$. Actually, the asset price S is itself a function of the time, that is, $S = S(t)$. However, we often omit time dependence in S in the following. As for the domain Ω of function P, observe that the price domain Ω_S is straightforwardly defined as $[0, +\infty]$. So, Ω must be a subset of $\Omega^* = \Omega_S \times \Omega_t = [0, +\infty] \times [0, T]$.

Evaluate P at maturity is a straightforward task, as $P(S, T)$ has to be equal to $\max(K - S, 0)$. In fact, if the price goes below K, then the option holder exercises the option and gains the difference between the current market value $S(T)$ and K. In contrast, if $S(T) > K$, the holder does not exercise the option, so the total cash flow is 0. However, we stress the fact that American options can be exercised every time before maturity T. Thus, from an analytical point of view, this implies a free boundary domain Ω. In fact, it is reasonable to assume that if the asset price S is sufficiently low, i.e. $S \ll K$, the holder exercises the option to immediately sell the underlying at a favourable price, thus gaining the difference $K - S$. The concept mentioned above of "sufficiently low price" can be mathematically represented by a function $B(t)$, also known as *free boundary*. Thus, $B(t)$ splits Ω^* into two subsets, namely Ω_1^* and Ω_2^*, respectively known as *continuation region* and stopping *region*. In formula:

$$\Omega_1^* = \left\{(S, t) \in \Omega^* \ s.t. \ S > B(t)\right\} \qquad \Omega_2^* = \left\{(S, t) \in \Omega^* \ s.t. \ S \leq B(t)\right\} \tag{3}$$

In other words, $B(t)$ can be considered the max price that makes it convenient to early exercise the option. So, in literature, it is also referred to as *optimal exercise boundary*. Note that $B(t)$ should always be less or equal to K, as it is not convenient for the holder to exercise the option when the current market price is above the strike K. Furthermore, observe that in the domain Ω_2^*, the option pricing problem is trivial, as in the stopping region, the option is exercised and its value overlaps with its payoff, that is, $K - S$. So, the real problem is to estimate the American put option price in the domain:

$$\Omega = \overline{\Omega_1^*} = [B(t), +\infty] \times [0, T] \tag{4}$$

3.1.2. The mathematical model

The Black–Scholes equation for American put options is a free boundary parabolic Partial Differential Equation (PDE). In more detail, the Black–Scholes model describes the option price dynamic as:

$$\frac{1}{2}\sigma^2 S^2 \frac{\partial^2 P}{\partial S^2} + (r - q) S \frac{\partial P}{\partial S} - rP + \frac{\partial P}{\partial t} = f(S, t) = 0 \ S > B(t); \ 0 \leq t \leq T \tag{5}$$

where $P = P(S, t)$ is the option price at time t and with underlying asset price S; r is the risk-free rate; q is the asset dividend yield; σ^2 is the asset variance; $B(t)$ is an unknown auxiliary function that represents the free boundary. The domain is as in Eq. (4). Furthermore, we can define the differential operator Λ as in Eq. (6).

$$\Lambda P = \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 P}{\partial S^2} + (r - q) S \frac{\partial P}{\partial S} - rP + \frac{\partial P}{\partial t} \tag{6}$$

Following the above considerations, the initial condition is formalized by Eq. (7).

$$P(S, T) = (K - S)_+ \qquad S \geq 0 \tag{7}$$

The Dirichlet boundary condition is:

$$\lim_{S \to +\infty} P(S, t) = 0 \qquad 0 \leq t \leq T \tag{8}$$

As for the free boundary $B(t)$, the initial condition is:

$$B(T) = K \tag{9}$$

The free boundary is also related to Dirichlet and Neumann boundary conditions, as reported in Eqs. (10) and (11).

$$P\big(B(t), t\big) = K - B(t) \qquad 0 \leq t \leq T \tag{10}$$

$$\frac{\partial P}{\partial S}\big(B(t), t\big) = -1 \qquad 0 \leq t \leq T \tag{11}$$

Observe that, in the previous equations, there are four parameters: risk-free rate r, maturity T, strike price K and asset volatility σ^2. Furthermore, there are two unknown functions, as both the option price $P(S, t)$ and the free boundary $B(t)$ have to be approximated.

Finally, we underline that, from a computational point of view, the non-limited domain Ω causes some issues. To overcome these difficulties, we exploit a bounded version of Ω, namely $\Omega_{bounded} = [B(t), S_{max}] \times [0, T]$. With a slight abuse of notation, in the following, we use Ω also to indicate $\Omega_{bounded}$. As for the constant S_{max}, we use the *three times rule*, that is we consider $S_{max} = 3K$. This is a well-established approach in the literature (see, for example, [34]).

3.2. Multi-dimensional setting

The discussion in the previous subsection is now extended to the multi-dimensional setting. We consider options with multiple underlying assets in this case, so the spatial dimension is $d \in \mathbb{N}$. This implies a more complex model. Furthermore, several contracts are possible, so we restrict our study to a particular case.

3.2.1. The pricing problem

In literature, there are several contracts with multiple underlying assets, each one with its own peculiarities. Firstly, we indicate the dimension with d and the assets value with the vector $\mathbf{S} = (S_1, \dots, S_d) \in \mathbb{R}^d$. Furthermore, we do not make any assumptions about asset correlation. So, all assets are related to each other, and we indicate the covariance matrix with $\Sigma \in \mathbb{R}^{d \times d}$. In particular, we write $\Sigma = \{\sigma_{i,j}\}_{i,j \in \{1,\dots,d\}}$. Finally, each asset i has its own dividend yield q_i.

In the following, we consider the contract whose payoff $\psi = \psi(\mathbf{S})$ is described in Eq. (12).

$$\psi(\mathbf{S}) = K - \min(\mathbf{S}) = K - \min(S_1, \dots, S_d) \tag{12}$$

We have chosen this type of contract as it is the natural generalization to the multi-dimensional setting of the one-dimensional put option discussed in the previous subsection. Furthermore, we are still considering an American option, so there are two unknown functions, namely the option price $P(\mathbf{S}, t)$ and the early exercise boundary $\mathbf{B}(t)$. In this case, $\mathbf{B}(t) = \big(B_1(t), \dots, B_d(t)\big)$ is a function from $[0, T]$ to $[0, +\infty]^d$. The domain $\Omega^* = [0, +\infty]^d \times [0, T]$ is split into the continuation and stopping regions Ω_1^*, Ω_2^* in this way:

$$\begin{aligned} \Omega_2^* &= \left\{(S_1, \dots, S_d, t) \in \Omega^* \mid \forall j, \ S_j > B_j(t)\right\} \\ \Omega_1^* &= \left\{(S_1, \dots, S_d, t) \in \Omega^* \mid \exists j \ s.t. \ S_j \leq B_j(t)\right\} \end{aligned} \tag{13}$$

3.2.2. The mathematical model

In the following, we extend the Black–Scholes model, as described in Eqs. (5)–(11), to the multi-dimensional case. We assume Eq. (14) describes the dynamic of each asset:

$$dS_i = S_i \mu_i dt + S_i \sum_{j=1}^{d} \sigma_{ij} dW_j \tag{14}$$

where W_j are independent one-dimensional Brownian motions whose increments have 0 mean and variance equal to dt. In the multi-dimensional framework, the Black–Scholes equation is represented in (15):

$$\frac{1}{2}\sum_{i,j=1}^{d}\left(\sum_{k=1}^{d}\sigma_{ik}\sigma_{jk}\right)S_i S_j \frac{\partial^2 P}{\partial S_i\, \partial S_j} + \sum_{i=1}^{d}(r-q_i)S_i\frac{\partial P}{\partial S_i} - rP + \frac{\partial P}{\partial t} = f(\mathbf{S},t) = 0 \tag{15}$$

The differential operator Λ becomes:

$$\Lambda P = \frac{1}{2}\sum_{i,j=1}^{d}\left(\sum_{k=1}^{d}\sigma_{ik}\sigma_{jk}\right)S_i S_j \frac{\partial^2 P}{\partial S_i\, \partial S_j} + \sum_{i=1}^{d}(r-q_i)S_i\frac{\partial P}{\partial S_i} - rP + \frac{\partial P}{\partial t} \tag{16}$$

According to Eq. (13), the domain is defined as:

$$\Omega = \prod_{i=1}^{d}\left[B_i(t), +\infty\right] \times \left[0, T\right] \tag{17}$$

Regarding the initial condition, analogously to the 1D case, we write:

$$P(\mathbf{S}, T) = \max\left(K - \min\left(S_i\right), 0\right) \tag{18}$$

The Dirichlet boundary condition now is far more complicated. In fact, even if one asset approaches infinity, the others may keep near to the exercise price. So, the option is not worthless, as it happens in the 1D case. To solve this issue, we exploit the so-called *time-discounted payoff boundary condition*, as in other works in literature, for example, [35]. Thus, we approximate the boundary condition with the payoff function whose strike is discounted to the current time t, as shown in Eq. (19). Observe that Eq. (19) can be viewed as a multi-dimensional generalization of Eq. (8), as when S approach to infinity, then the time-discounted payoff approaches to 0.

$$\lim_{S_i \to +\infty} P(\mathbf{S}, t) = \max\left(Ke^{-r(T-t)} - \min(S_1, \dots, S_d),\ 0\right) \qquad \forall i \in \{1, \dots, d\} \tag{19}$$

The free boundary initial condition does not show relevant changes, so:

$$\mathbf{B}(T) = (K, \dots, K) \tag{20}$$

Also, for the Dirichlet boundary condition related to the free boundary, the situation is almost the same:

$$P\left(\mathbf{B}(t), t\right) = K - \min\left(\mathbf{B}(t)\right) \qquad 0 \le t \le T \tag{21}$$

As for the Neumann condition, we have to consider that only the lowest component of the price vector affects the exercise value. So, Eq. (11) is generalized to the multi-dimensional case as in Eq. (22).

$$\frac{\partial P}{\partial S_i}\left(\mathbf{B}(t), t\right) = \begin{cases} -1 & if\ i = \operatorname{argmin}\left(\mathbf{B}(t)\right) \\ 0 & otherwise \end{cases} \qquad 0 \le t \le T \tag{22}$$

Finally, observe that, as already discussed above, also in this multi-dimensional case, we consider a bounded domain $\Omega_{bounded}$ instead of Ω, with:

$$\Omega_{bounded} = \prod_{i=1}^{d}\left[B_i(t), S_{max}\right] \times \left[0, T\right] \tag{23}$$

Observe that S_{max} is not dependent on i. In fact, as the strike price K is assumed to be the same for each asset, also the upper bound S_{max} is the same along all spatial axes. Furthermore, the same abuse of notation of the 1D case is done, so in the following, we refer to $\Omega_{bounded}$ as Ω.

4. PINN model

This section describes the general model developed for solving the American put option pricing problem. In particular, the general details about the models exploited are given. We refer to the next section for further detail about the specific network architecture used within each experiment.

4.1. Feedforward neural networks for PINN

As already discussed above, the problem we face up is a free boundary problem. This means there are two unknown functions, namely the solution $P(\mathbf{S}, t) : \mathbb{R}^{d+1} \to \mathbb{R}$ and the free boundary $\mathbf{B}(t) : \mathbb{R} \to \mathbb{R}^d$. Following a promising approach in the literature, see, for example, [36], two FNNs are exploited, one to approximate each unknown.

A FNN can be thought of as a collection of vectors, named *layers* l_j, *weights matrices* and *bias vectors* W_j and b_j, and usually non-linear *activation functions* ϕ_j. More formally:

$$\begin{aligned} &n_j \in \mathbb{N},\ j \in \{0, 1, \dots, L\} \qquad l_j \in \mathbb{R}^{n_j},\ j \in \{0, 1, \dots, L\} \\ &W_j \in \mathbb{R}^{n_j \times n_{j-1}},\ j \in \{1, \dots, L\} \qquad b_j \in \mathbb{R}^{n_j},\ j \in \{1, \dots, L\} \end{aligned} \tag{24}$$

The layers are linked to each other as described in Eq. (25).

$$l_j = \phi_j(W_j l_{j-1} + b_j) \qquad j \in \{1, \dots, L\} \tag{25}$$

The first layer, also known as *input layer*, contains problem input. The last layer, or *output layer* returns model outputs. For the solution network FNN^{sol} and for the free boundary network FNN^{fb}, the input and output layers are as follows:

$$\begin{aligned} &Solution \quad l_0^{sol} = (\mathbf{S}, t) \in \mathbb{R}^{d+1} \quad l_{L^{sol}}^{sol} \approx P(\mathbf{S}, t) \in \mathbb{R} \\ &Free\ Boundary \quad l_0^{fb} = (t) \in \mathbb{R} \quad l_{L^{fb}}^{fb} \approx \mathbf{B}(t) \in \mathbb{R}^d \end{aligned} \tag{26}$$

In the previous equation, the approximation $\approx$ is intended to be after the convergence of the learning process by minimizing a convenient loss function. The number of layers L^{sol} and L^{fb}, their dimension n_j^{sol}, n_j^{fb}, and also the activation functions ϕ_j^{sol} and ϕ_j^{fb} make up the so-called *network architecture* and they are treated as hyperparameters. So, they are considered fixed within each experiment, and their value is often heuristically determined. Instead, the weights matrices and bias vectors W_j^{sol}, b_j^{sol}, W_j^{fb} and b_j^{fb} are obtained through an optimization process, also known as *backpropagation*, by minimizing an appropriately defined loss function, namely $\mathcal{L}^{sol}$ for FNN^{sol} and $\mathcal{L}^{fb}$ for FNN^{fb}. Such a type of minimization process iteratively updates network weights. Each complete updating cycle is also referred to as *epoch*. Usually, these loss functions act on the output layers by evaluating some quantities related to them. In fact, observe that the output layers $l_{L^{sol}}^{sol}$ and $l_{L^{fb}}^{fb}$ can be written as function of the input layers l_0^{sol} and l_0^{fb}. That is:

$$\begin{aligned} l_{L^{sol}}^{sol} &= \phi_{L^{sol}}^{sol}\left(W_{L^{sol}}^{sol}\phi_{L^{sol}-1}^{sol}\left(W_{L^{sol}-1}^{sol}\ \cdots\ \phi_1^{sol}(W_1^{sol}l_0^{sol} + b_1^{sol})\ \cdots\ + b_{L^{sol}-1}^{sol}\right) + b_{L^{sol}}^{sol}\right) \\ &= FNN^{sol}(l_0^{sol}; W_1^{sol}, \dots, W_{L^{sol}}^{sol}, b_1^{sol}, \dots, b_{L^{sol}}^{sol}) \end{aligned} \tag{27}$$

$$\begin{aligned} l_{L^{fb}}^{fb} &= \phi_{L^{fb}}^{fb}\left(W_{L^{fb}}^{fb}\phi_{L^{fb}-1}^{fb}\left(W_{L^{fb}-1}^{fb}\ \cdots\ \phi_1^{fb}(W_1^{fb}l_0^{fb} + b_1^{fb})\ \cdots\ + b_{L^{fb}-1}^{fb}\right) + b_{L^{fb}}^{fb}\right) \\ &= FNN^{fb}(l_0^{fb}; W_1^{fb}, \dots, W_{L^{fb}}^{fb}, b_1^{fb}, \dots, b_{L^{fb}}^{fb}) \end{aligned} \tag{28}$$

In particular, observe that the two resulting functions FNN^{sol} and FNN^{fb} in Eqs. (27) and (28) can be also viewed as functions of the weights and biases sets. So, the backpropagation aims to find the sets that minimize the loss function. For example, the parameters of FNN^{sol} are obtained as in Eq. (29), and the situation for FNN^{fb} is almost the same.

$$(W^{sol}, b^{sol}) = (W_1^{sol}, \dots, W_{L^{sol}}^{sol}, b_1^{sol}, \dots, b_{L^{sol}}^{sol})$$

$$= \text{argmin}\left\{ \mathcal{L}^{sol}(W^{sol}, b^{sol}) \; s.t. \; W^{sol} \in \prod_{j=1}^{L^{sol}} \mathbb{R}^{n_j \times n_{j-1}}, \; b^{sol} \in \prod_{j=1}^{L^{sol}} \mathbb{R}^{n_j} \right\} \tag{29}$$

The peculiarity of PINN models is that the problem physics gives the loss function. In other words, the loss function is defined in such a way as to combine the PDE formulation and the initial and boundary conditions. The following subsection clarifies this point.

4.2. Collocation points & loss functions

The definition of the loss function plays a crucial role in adequately training an FNN. PINN models exploit it to incorporate PDE and initial and boundary conditions in the neural network. This is done by creating an appropriate training set $\mathcal{X} = \{x_i\}_{i \in N}$ made up of several *collocation points* belonging to different types. As pointed out by Equations from (15) to (22), in our experiments, we face up with: (i) PDE (ii) initial condition for the solution (iii) Dirichlet condition for the solution (iv) initial condition for the free boundary (v) Dirichlet condition at the free boundary (vi) Neumann condition at the free boundary. Accordingly, the train set is divided into several subsets.

Firstly, we analyze the free boundary training set $\mathcal{X}^{fb} = \{x_i^{fb}\}_{i \in \{1,\dots,N^{fb}\}}$, $x_i^{fb} \in [0,T] \; \forall i \in \{1,\dots,N^{fb}\}$. We can write:

$$\mathcal{X}^{fb} = \mathcal{X}^{fb}_{init} \cup \mathcal{X}^{fb}_{Dir} \cup \mathcal{X}^{fb}_{Neu} \tag{30}$$

In particular, the initial condition given by Eq. (20) requires one collocation point and it is represented by $\mathcal{X}^{fb}_{init} = \{x_1^{fb}\} = \{T\}$. Instead, Dirichlet and Neumann conditions at the free boundary (Eqs. (21) and (22)) require several collocation points, and they are respectively represented by $\mathcal{X}^{fb}_{Dir} \subset [0,T]$ and $\mathcal{X}^{fb}_{Neu} \subset [0,T]$. Clearly, we have: $N^{fb} = 1 + |\mathcal{X}^{fb}_{Dir}| + |\mathcal{X}^{fb}_{Neu}|$. As for the solution training set $\mathcal{X}^{sol} = \{x_i^{sol}\}_{i \in \{1,\dots,N^{sol}\}}$, each collocation point x_i^{sol} is in the domain Ω. Furthermore, we can split $\mathcal{X}^{sol}$ as follows:

$$\mathcal{X}^{sol} = \mathcal{X}^{sol}_{PDE} \cup \mathcal{X}^{sol}_{init} \cup \mathcal{X}^{sol}_{Dir} \cup \mathcal{X}^{sol}_{Fb\_Dir} \cup \mathcal{X}^{sol}_{Fb\_Neu} \tag{31}$$

In more detail, the PDE is formed by the collocation points $\mathcal{X}^{sol}_{PDE} \subset \Omega$. Regarding the initial condition in Eq. (18) and the Dirichlet condition given by Eq. (19), we have:

$$\begin{aligned} &\mathcal{X}^{sol}_{init} \subset \prod_{i=1}^{d} [B_i(t), S_{max}] \times \{T\} \\ &\mathcal{X}^{sol}_{Dir} = \{(S_1,\dots,S_d,t) \in \Omega | \exists i \; s.t. \; S_i = S_{max}\} \end{aligned} \tag{32}$$

Finally, the boundary conditions at the free boundary are strongly related to the estimate of the free boundary provided by FNN^{fb}, so the solution collocation points are obtained from the free boundary ones as described in Eq. (33):

$$\begin{aligned} &\mathcal{X}^{sol}_{Fb\_Dir} = \left\{ \left(FNN^{fb}(t), t\right) \in \Omega | t \in \mathcal{X}^{fb}_{Dir} \right\} \quad and \\ &\mathcal{X}^{sol}_{Fb\_Neu} = \left\{ \left(FNN^{fb}(t), t\right) \in \Omega | t \in \mathcal{X}^{fb}_{Neu} \right\} \end{aligned} \tag{33}$$

After defining the collocation points, we discuss how to exploit them to define the correct loss function. Regarding the solution network FNN^{sol}, its loss function can be viewed as the sum of the contribution of PDE, initial and boundary conditions in this way:

$$\mathcal{L}^{sol} = \mathcal{L}^{sol}_{PDE} + \mathcal{L}^{sol}_{init} + \mathcal{L}^{sol}_{Dir} + \mathcal{L}^{fb}_{Dir} + \mathcal{L}^{fb}_{Neu} \tag{34}$$

Let analyze individually the components in Eq. (34). $\mathcal{L}^{sol}_{PDE}$ takes into account the PDE. In particular, as from Eqs. (15) and (16) we have $\Lambda P(\mathbf{S},t) = f(\mathbf{S},t) = 0$, then $\mathcal{L}^{sol}_{PDE}$ is defined as the norm of the vector obtained by evaluating Λ in the collocation points $\mathcal{X}^{sol}_{PDE}$, under a convenient metric $\|\cdot\|$. So:

$$\mathcal{L}^{sol}_{PDE} = \left\| \left(\Lambda FNN^{sol}(\mathbf{S},t) \right)_{(\mathbf{S},t) \in \mathcal{X}^{sol}_{PDE}} \right\| \tag{35}$$

As for the initial and boundary conditions, we have an analytical expression of the desired output, so the respective losses are computed as the distance between the FNN^{sol} output and the expected target, under the same metric $\|\cdot\|$. In more detail:

$$\mathcal{L}^{sol}_{init} = \left\| \left(FNN^{sol}(\mathbf{S},T) - P(\mathbf{S},T) \right)_{(\mathbf{S},T) \in \mathcal{X}^{sol}_{init}} \right\| \tag{36}$$

$$\mathcal{L}^{sol}_{Dir} = \left\| \left(FNN^{sol}(\mathbf{S},t) - P(\mathbf{S},t) \right)_{(\mathbf{S},t) \in \mathcal{X}^{sol}_{Dir}} \right\| \tag{37}$$

$$\begin{aligned} \mathcal{L}^{fb}_{Dir} &= \left\| \left(FNN^{sol}(\mathbf{S},t) - P(\mathbf{S},t) \right)_{(\mathbf{S},t) \in \mathcal{X}^{sol}_{fb\_Dir}} \right\| \\ &= \left\| \left(FNN^{sol}(FNN^{fb}(t),t) - P(FNN^{fb}(t),t) \right)_{t \in \mathcal{X}^{fb}_{Dir}} \right\| \end{aligned} \tag{38}$$

$$\begin{aligned} \mathcal{L}^{fb}_{Neu} &= \sum_{i=1}^{d} \left\| \left(\frac{\partial FNN^{sol}}{\partial S_i}(\mathbf{S},t) - \frac{\partial P}{\partial S_i}(\mathbf{S},t) \right)_{(\mathbf{S},t) \in \mathcal{X}^{sol}_{fb\_Neu}} \right\| \\ &= \sum_{i=1}^{d} \left\| \left(\frac{\partial FNN^{sol}}{\partial S_i}(FNN^{fb}(t),t) - \frac{\partial P}{\partial S_i}(FNN^{fb}(t),t) \right)_{t \in \mathcal{X}^{fb}_{Neu}} \right\| \end{aligned} \tag{39}$$

where the equivalences contained in Eqs. (18), (19), (21) and (22) have been used. Finally, we remark that PDE loss is also referred to as *Unsupervised loss*. In fact, for PDE collocation points, there is not a real target output, as it happens for initial and boundary conditions. So, it can be viewed as an example of Unsupervised Learning. In contrast, all the other losses are defined jointly with a target result. That is, each observation is individually labelled. This is an example of Supervised Learning. In this view, PINNs are an example of a mixed Unsupervised and Supervise Learning problem.

As for the free boundary, its loss can be defined as the sum of the losses related to the initial, Dirichlet and Neumann condition, that is:

$$\mathcal{L}^{fb} = \mathcal{L}^{fb}_{init} + \mathcal{L}^{fb}_{Dir} + \mathcal{L}^{fb}_{Neu} \tag{40}$$

$\mathcal{L}^{fb}_{Dir}$ and $\mathcal{L}^{fb}_{Neu}$ have already been discussed above. Instead, as for $\mathcal{L}^{fb}_{init}$, the reasoning is the same as in Eq. (36):

$$\mathcal{L}^{fb}_{init} = \left\| FNN^{fb}(T) - (K,\dots,K) \right\| \tag{41}$$

So, the loss on the free boundary is strictly related to the estimate of the solution provided by FNN^{sol}, and vice-versa the loss of the solution is strongly related to the estimate FNN^{fb} of the free boundary. In other words, the two neural networks are trained concurrently to reach the same goal, i.e. the minimization of the losses $\mathcal{L}^{fb}_{Dir}$ and $\mathcal{L}^{fb}_{Neu}$. Till now, to the best of our knowledge, no attempts have been made to prefer one of the two networks over the other, and in all the works concerning free boundary problems (see, for example, [37] applied to the Heat equation, which can be transformed into the Black–Scholes PDE) at each epoch FNN^{sol} and FNN^{fb} are updated once each. So, we have studied what happens if we give more relevance to one network. In other words, we have modified PINN training to perform multiple steps of FNN^{sol} against a single step of FNN^{fb} at each epoch and vice-versa. The experimental results (the next section provides the full description of the experimental setting) show that performing more steps of FNN^{sol} at each epoch could benefit the model convergence and final loss. Instead, performing multiple steps of FNN^{fb} seems to degrade overall performance. So, the algorithmic trick we propose is to perform more training steps of the solution network against a single step of the free boundary at each epoch. It is not a novel algorithmic trick in that it is already used in other examples of concurrent or adversarial training (for example, it is common to exploit more Discriminator steps for each Generator step in the Generative Adversarial Training [38]). However, to the best of our knowledge, it is the first time this trick has been applied to PINNs. Finally, the training procedure is summarized in Algorithm 1 and in Fig. 1.

Algorithm 1 PINN Training Process

Require: Networks FNN^{sol}, FNN^{fb}; Epochs number *epochs*; Optimizers opt^{sol}, opt^{fb}; Number of sol steps *N_steps*.

Initialize networks weights and biases W_j^{sol}, b_j^{sol}, W_j^{fb} and b_j^{fb}.
for $epoch = 0, \cdots, epochs$ **do**
  Compute free boundary values $FNN^{fb}(t)$ on the sets $\mathcal{X}_{Dir}^{fb}, \mathcal{X}_{Neu}^{fb}$. ▷ *More sol steps for each fb*
  for $iter = 1, \cdots, N\_steps - 1$ **do**
    Exploit the free boundary estimate on $\mathcal{X}_{Dir}^{fb}$ and $\mathcal{X}_{Neu}^{fb}$ to obtain the sets $\mathcal{X}_{Fb\_Dir}^{sol}$ and $\mathcal{X}_{Fb\_Neu}^{sol}$.
    Estimate FNN^{sol} and its derivatives on the training sets $\mathcal{X}_{PDE}^{sol}, \mathcal{X}_{init}^{sol}, \mathcal{X}_{Dir}^{sol}, \mathcal{X}_{Fb\_Dir}^{sol}, \mathcal{X}_{Fb\_Neu}^{sol}$.
    Compute solution loss $\mathcal{L}^{sol} = \mathcal{L}_{PDE}^{sol} + \mathcal{L}_{init}^{sol} + \mathcal{L}_{Dir}^{sol} + \mathcal{L}_{Dir}^{fb} + \mathcal{L}_{Neu}^{fb}$.
    Update solution weights $W_j^{sol}, b_j^{sol} = opt^{sol}(\mathcal{L}^{sol}, W_j^{sol}, b_j^{sol})$.
  end for
  Estimate free boundary values $FNN^{fb}(t)$ on the sets $\mathcal{X}_{init}^{fb}, \mathcal{X}_{Dir}^{fb}, \mathcal{X}_{Neu}^{fb}$. ▷ *Last step fb and sol jointly*
  Exploit the free boundary estimate on $\mathcal{X}_{Dir}^{fb}$ and $\mathcal{X}_{Neu}^{fb}$ to obtain the sets $\mathcal{X}_{Fb\_Dir}^{sol}$ and $\mathcal{X}_{Fb\_Neu}^{sol}$.
  Evaluate FNN^{sol} and its derivatives on the training sets $\mathcal{X}_{Fb\_Dir}^{sol}$ and $\mathcal{X}_{Fb\_Neu}^{sol}$ to obtain the losses $\mathcal{L}_{Dir}^{fb}$ and $\mathcal{L}_{Neu}^{fb}$.
  Compute free boundary loss $\mathcal{L}^{fb} = \mathcal{L}_{init}^{fb} + \mathcal{L}_{Dir}^{fb} + \mathcal{L}_{Neu}^{fb}$.
  Update free boundary weights $W_j^{fb}, b_j^{fb} = opt^{fb}(\mathcal{L}^{fb}, W_j^{fb}, b_j^{fb})$.
  Estimate FNN^{sol} and its derivatives on the training sets $\mathcal{X}_{PDE}^{sol}, \mathcal{X}_{init}^{sol}, \mathcal{X}_{Dir}^{sol}$ to obtain $\mathcal{L}_{PDE}^{sol} + \mathcal{L}_{init}^{sol} + \mathcal{L}_{Dir}^{sol}$.
  Compute solution loss $\mathcal{L}^{sol} = \mathcal{L}_{PDE}^{sol} + \mathcal{L}_{init}^{sol} + \mathcal{L}_{Dir}^{sol} + \mathcal{L}_{Dir}^{fb} + \mathcal{L}_{Neu}^{fb}$.
  Update solution weights $W_j^{sol}, b_j^{sol} = opt^{sol}(\mathcal{L}^{sol}, W_j^{sol}, b_j^{sol})$.
end for

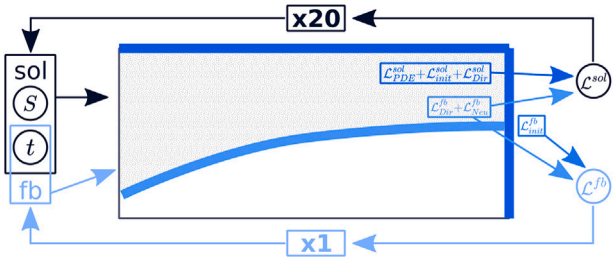

Fig. 1. Graphical Abstract in one dimension - The values generated by FNN^{sol} and FNN^{fb} are used to compute the losses, which are fed back to train the networks. Furthermore, more iterations of the solution network are performed to enhance result quality (in the figure, 4 iterations).

Finally, observe that, sometimes, Eqs. (34) and (40) are rewritten to allow for weighted losses, that is:

$$\mathcal{L}^{sol} = \lambda_{PDE}^{sol}\mathcal{L}_{PDE}^{sol} + \lambda_{init}^{sol}\mathcal{L}_{init}^{sol} + \lambda_{Dir}^{sol}\mathcal{L}_{Dir}^{sol} + \lambda_{Dir}^{fb}\mathcal{L}_{Dir}^{fb} + \lambda_{Neu}^{fb}\mathcal{L}_{Neu}^{fb} \tag{42}$$

$$\mathcal{L}^{fb} = \lambda_{init}^{fb}\mathcal{L}_{init}^{fb} + \lambda_{Dir}^{fb}\mathcal{L}_{Dir}^{fb} + \lambda_{Neu}^{fb}\mathcal{L}_{Neu}^{fb} \tag{43}$$

Where the λs are positive scalars. However, from the early experimental stage, it results that no significant changes occur in the final result if we change loss weights. So, according to the parsimonious principle for neural networks, in the following we consider equally-weighted losses, as those in Eqs. (34) and (40).

4.3. Equation parameters and parametric model

As already mentioned in Section 3, there are four parameters in the Equations considered till now, namely assets volatility σ, strike K, maturity T and risk-free rate r. These parameters are fixed in the experiments and do not interact in the PINN model. In other words, we can say that our model is tailored to these parameters. So, to price options under different market conditions, that is, with a different combination of parameters, we have to train another PINN. As we discuss in the experimental part, see Section 5.8, although the computational time to price an option with an already-trained PINN (i.e. by varying asset price $\mathbf{S}$ and time t) is minimal, training is usually an expensive operation. This can represent a strong limitation for PINN applications in a real-world context.

To overcome this problem, we also propose a *parametric model*, i.e. a model that contains the market conditions and work on them as input for option pricing. As we aim to provide the first step in this direction, we develop and test such a solution only for the 1D case. So, the input layers of FNN^{sol} and FNN^{fb} defined in Eq. (26) are modified as in Eq. (44). No changes occur in the outputs.

$$\begin{aligned} &Solution \quad I_0^{sol} = (S, t, r, \sigma, T, K) \in \mathbb{R}^6 \\ &Free\ Boundary \quad I_0^{fb} = (t, r, \sigma, T, K) \in \mathbb{R}^5 \end{aligned} \tag{44}$$

So, by modifying the inputs, we are also allowing the network to handle the parameters, which now become model variables. This approach dramatically benefits by allowing a parametric study and several simulations with different market conditions with just one PINN training. Furthermore, as stated in Section 5.8, the computational time for pricing an option with an already-trained PINN is negligible. This could make such a study and simulation possible within a relatively short time. For example, the development of reliable parametric models could help in the computation of the so-called *implied volatility*, i.e. the volatility obtained by looking at options prices in the real market. In fact, with a parametric model as fast in the prediction as PINN, it is sufficient to use any root-finding algorithm, such as Newton–Raphson, to compute implied volatility straightforwardly. It could be a significant advantage in the financial industry. So, its study for further applications is undoubtedly noteworthy. In fact, calibration of pricing models is critical in modern financial engineering [39]. However, the problem becomes far more complex as input dimensions in both networks are bigger. So, including also the parameters as model input could lead to an increase in computational time and degradation in performance.

One possible way to limit this drawback of a parametric model is to restrict the analysis only to some interesting parameters. In other words, while in the non-parametric model we consider all four parameters as fixed, and in the parametric model we consider the four parameters as input variables for the network, we can build a model which treats some parameters as fixed and the other ones as network inputs. We refer to this kind of model as *semi-parametric model*. A concrete example can be the following: we consider T and r as fixed and σ and K as variables. Undoubtedly, σ is the most critical parameter, at least accordingly to most of the existing financial literature, so it is helpful to study it. Moreover, also studying the price behaviour as K changes can be interesting in the plot of the so-called *volatility smile*. The volatility smile represents the dependence of implied volatility with respect to the strike price (or the ratio $\frac{K}{S}$). It is empirically shown that this relationship is not constant [40]. This is due to several reasons, mainly related to the asset type. However, some general reasons are: (i) price variations are often discontinued and exhibit extreme variation due to certain news (this is especially true for stocks, where it is referred to with the term *crash-fobia*); (ii) volatility often is not constant, either with respect to the time and the underlying asset value (in contrast to the assumption in Eq. (14)). Understanding this phenomenon is a crucial task in different aspects of financial engineering, such as the computation of the *implied distribution* of the underlying asset [41]. So, such a semi-parametric model could be a valuable ally in studying this particular pattern.

Table 1
The number of collocation points used in each experiment. Note that, for construction, $\mathcal{X}^{sol}_{fb\_Dir}$ and $\mathcal{X}^{sol}_{fb\_Neu}$ have the same number of collocation points of $\mathcal{X}^{fb}_{Dir}$ and $\mathcal{X}^{fb}_{Neu}$, respectively. So, we have neglected these sets in the Table. Furthermore, observe how the training set dimension increases as the input dimension becomes bigger. Finally, note that the initial condition for the free boundary requires just one point if the maturity T and strike K are fixed. This is what happens in the non-parametric models. Instead, in the (semi-)parametric model, as at least one between T and K is considered as a variable of the network, we use more than one collocation point in $\mathcal{X}^{fb}_{init}$.

Model	$\mathcal{X}^{sol}_{PDE}$	$\mathcal{X}^{sol}_{init}$	$\mathcal{X}^{sol}_{Dir}$	$\mathcal{X}^{fb}_{init}$	$\mathcal{X}^{fb}_{Dir}$	$\mathcal{X}^{fb}_{Neu}$
1D	30,000	300	300	1	300	300
2D	60,000	1500	1500	1	1500	1500
3D	120,000	4000	9000	1	9000	9000
4D	120,000	12,000	12,000	1	12,000	12,000
1D - SemiPar.	80,000	9000	9000	4500	9000	9000
1D - Parametric	100,000	10,000	10,000	5000	10,000	10,000

5. Experimental results

In this section, we describe the experimental setting and the results obtained. Furthermore, we discuss some properties of the approximated solution and show the computational time of our proposal.

5.1. Experimental setting

In describing the experimental setting, we start with the non-parametric models and then discuss the parametric models. To give an overview of PINN performance in the univariate and multivariate framework, we conduct experiments in 1 dimension (1D), that is, one spatial dimension and the time, and 2, 3, and 4 dimensions (2D, 3D, and 4D). In both cases, the maturity is $T = 3$, the strike is $K = 10$, the dividend yield q is 0 for every asset, and the risk-free rate is $r = 0.01$. As for the variance matrix σ, in the 4D case, it is:

$$\begin{pmatrix} 0.050.01 & 0.1 & 0 & \\ 0.01 & 0.06 & -0.03 & 0 \\ 0.1 & -0.03 & 0.4 & 0.2 \\ 0 & 0 & 0.2 & 0.3 \end{pmatrix}$$

For lower dimensions, the first rows and columns are considered. Instead, for the parametric models, the domain of the parameters is the following one:

$$\Omega_{par} = \{(r, \sigma^2, T, K)\ s.t.\ r \in [0.001, 0.03], \\ \sigma^2 \in [0.001, 0.005],\ T \in [2, 5],\ K \in [8, 12]\}$$

In the semi-parametric model, T and r are fixed as in the non-parametric models, and σ^2 and K vary in the above interval.

As for the number and type of collocation points, some considerations have to be made. Firstly, as Ω as described in Eq. (23) is strictly related to the free boundary, which is unknown, we initially sample points for $\mathcal{X}^{sol}_{PDE}$ in the hypercube $[0, S_{max}]^d \times [0, T]$. Then, following the procedure described by [36], at each iteration of the free-boundary networks, only the points in the continuation region are used to compute the loss. Secondly, as for the sampling criterion we use, we exploit the so-called *Sobol sequences* [42]. Thirdly, the dimension of the training sets $\mathcal{X}^{sol}$ and $\mathcal{X}^{fb}$ should be directly related to the dimension of the problem we face up. In other words, the more dimensions we consider, the more collocation points we should use to guarantee the network's convergence and a sufficient generalization capability of the model. Finally, the number of collocation points, i.e., the dimension of the training sets, is summarized in Table 1.

The network architecture and the hyperparameters of the model are empirically determined. As for the network architecture, the solution has 8 hidden layers of dimension 20, and the free boundary has 3 hidden layers of dimension 100. The activation functions are hyperbolic tangents for every layer except output layers, where the identity function is used. As for the weights initialization, the well-known *Glorot normal* initializer is used [43]. The architecture is summarized in Eq. (45).

Table 2
Summary of the optimizer features. All the experiments use RMSprop as the optimizer and the same initial learning rate, which is 1e−2. The number of steps until decay and its magnitude are reported in the table. For multidimensional experiments, an optimizer is used for the solution and one for the free boundary network. In the 1D experiment, only one optimizer is used. Finally, the number of epochs for each model and the number of steps in the algorithmic trick are shown.

	1D	2D	3D	4D	Param	Semi
lr Step	500	400	320	640	320	320
lr Dec	0.9	0.9	0.975	0.975	0.975	0.975
lr Step	–	100	80	160	80	80
lr Dec	–	0.9	0.975	0.975	0.975	0.975
Epochs	4000	10,000	15,000	25,000	15,000	15,000
n_steps	20	4	4	4	4	4

$$L^{sol} = 9 \quad n_j^{sol} = 20\ \forall j \in \{1, \dots, 8\}$$
$$L^{fb} = 4 \quad n_j^{fb} = 100\ \forall j \in \{1, \dots, 3\} \qquad \phi_j^{sol} = \phi_j^{fb} = tanh\ \forall j \tag{45}$$

Regarding the loss function, the square of the L2 norm is used, i.e. the well-known *Mean Square Error* (MSE). Although in our code we have implemented the possibility for different-weighted loss functions, from empirical tests, it seems that, in our context and for the problem we deal with, there are no significant benefits from using a particular weighting system different. So, in the following, it is implied that all the loss weights are equal to 1. As for the optimizer used for the backpropagation, we carried out experiments by using two popular algorithms, namely *Adam* and *RMSprop* [44]. From some early results, mainly related to the 1D case, it seems that RMSprop performs much better than Adam, so we use the former in all our experiments. As for the network learning rate, the initial one is indicated with *lr*, and it is set to 1e−2. Then, it decreases over iterations with an exponential behaviour. This means that the learning rate function against the number of iterations is a staircase function. After each iteration, there is a little decrement in the learning rate, computed in such a way that after *lr Step* iterations, the total decrement is equal to *lr Dec*. This kind of strategy, which is common in deep learning applications, allows us to use a bigger learning rate for a faster convergence at the first epochs and a small learning rate, for fine-tuning, at the last epochs. Moreover, we report that for multi-dimensional and parametric experiments, two different optimizers are used: one for the solution network and the other for the free boundary. The notation for the hyperparameters is adjusted accordingly. Finally, we indicate with *n_steps* the number of FNN^{sol} iterations against each FNN^{fb} step. Table 2 shows a summary about optimizer features.

The training of the network is performed on Google Colab. GPU hardware accelerator is used.

As mentioned above, there is no closed-form solution for American put options under the Black–Scholes model. Thus, we cannot compare the PINN solution with the real one, and we must find another way to assess model performance. This task is accomplished by generating another set of collocation points (by using a random sampler in the domain $[0, S_{max}]^d \times [0, T]$ and then filtering the points in Ω, as already explained before). This new set of points is also referred to as *test set*. In more detail, we generate 1000000 points for the PDE (10000 for the parametric models) and 1000 for each condition (except for the free boundary initial one in the non-parametric model, which requires only 1 point). Finally, we compute model performance in the test set, and we look at the obtained loss as an estimate of the generalization capability of PINN.

It is well-known that neural networks strongly depend on random weight initialization. So, it can also be interesting to study the network's behaviour related to the random seed. Thus, we perform 10 iterations for each model by using the first 10 prime numbers as the

Table 3
Losses values for the launches of the 1D model, approximated to the first 6 decimal places.

Seed	Unsupervised	Initial	Dirichlet	FB_Init	FB_Dir	FB_Neu	Free boundary	Solution
2	0.001272	0.000295	0.000000	0.000031	0.000031	0.000004	0.000067	0.001603
3	0.000659	0.000247	0.000000	0.000011	0.000020	0.000003	0.000036	0.000931
5	0.000556	0.000173	0.000000	0.000008	0.000030	0.000006	0.000045	0.000766
7	0.001105	0.000201	0.000000	0.000024	0.000027	0.000003	0.000056	0.001338
11	0.001444	0.000430	0.000000	0.000034	0.000035	0.000006	0.000076	0.001916
13	0.000821	0.000139	0.000000	0.000020	0.000027	0.000005	0.000053	0.000995
17	0.000536	0.000183	0.000000	0.000011	0.000033	0.000005	0.000051	0.000759
19	0.000655	0.000113	0.000000	0.000012	0.000018	0.000018	0.000049	0.000805
23	0.000867	0.000158	0.000000	0.000021	0.000025	0.000002	0.000049	0.001054
29	0.001445	0.000231	0.000000	0.000004	0.000038	0.000008	0.000051	0.001724
Mean	0.000936	0.000217	0.000000	0.000018	0.000028	0.000006	0.000053	0.001189
Median	0.000844	0.000192	0.000000	0.000016	0.000029	0.000005	0.000051	0.001024

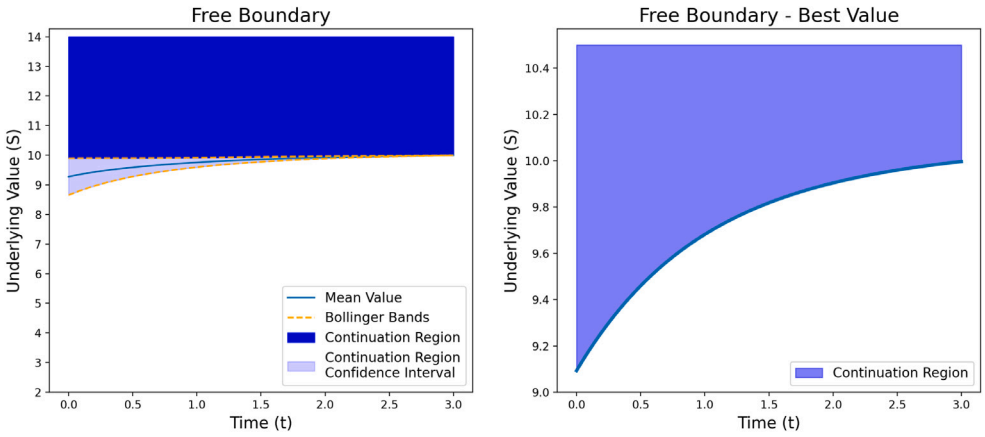

Fig. 2. Free Boundary - Early Exercise in the 1D experiment. The left plot shows the mean curve and the confidence interval, with the latter being obtained as the Bollinger bands with 1 standard deviation up and down. Furthermore, the continuation region is highlighted. The right plot shows the free boundary of the best model (i.e. with the lowest error in the test set, that is, random seed equal to 5).

random seed. As the following subsections show, noticeable differences among different random seeds exist only in the multi-dimensional experiments.

5.2. 1D model - result and discussion

The 1D model works in the standard uni-dimensional case. That is, the American option has just one underlying asset, so that it can be viewed as a function of only two variables: the asset value S and time t. As already explained in the previous subsection, the parameters are fixed to the following values: maturity $T = 3$; risk-free rate $r = 0.01$; strike $K = 10$; asset standard deviation $\sigma = 0.05$. Furthermore, 10 trials with equally random seeds are carried out. The results of launches for the 1D experiment are reported in Table 3.

As shown in the table, the error seems to be relatively stable across the different iterations. Furthermore, most of the error is related to the solution losses, especially PDE (Unsupervised) and initial conditions. In fact, the Dirichlet condition is almost perfectly learned by the network, with a negligible error in the first 6 digital places. Also, the sum of free boundary losses is relatively restrained, about two orders of magnitude lower than solution one. For the free boundary, the primary error drivers are initial and Dirichlet conditions, while Neumann one is one order of magnitude smaller than the others.

In this particular case, the exact shape of the free boundary is shown in Fig. 2. As reported in the figure, the shapes of the two curves (the mean and the optimal one) are pretty similar. Both of them are monotonically increasing until they approach the initial condition $B(T) = K$. Instead, the optimal is slightly up-shifted.

Furthermore, the solution obtained by the PINN is visualized as a heatmap in Fig. 3.

5.3. Comparison with finite difference methods

This subsection has two main purposes: (i) we ensure that the option values predicted by the PINN are consistent, i.e., the computed values are close to ones obtained by finite-difference methods, and (ii) we show how the NN architecture design affects the learning process in terms of loss.

It has been shown that not always low losses values $\mathcal{L}^{sol}$ and $\mathcal{L}^{fb}$ mean the closeness of the approximated solution to the correct one (see, for example, [45,46]). So, the PINN numerical solution is compared to a MATLAB code [47] by using an Euler-based explicit/implicit method, used to value dividend-paying American options. The algorithm uses explicit and implicit finite-difference methods to solve the Black–Scholes partial differential problem, accounting for early exercise and dividend payments in dividend-paying stocks. The test was carried out by gradually increasing the mesh points and selecting the same subset of 120 × 120 each time. The mesh for the explicit and implicit methods begins with a grid of 2000 × 500 (2k) and is doubled three times before reaching 8000 × 2000 (8k). The results can be seen in Table 4. The L2 norm is used to report all errors.

According to the theoretical considerations reported in Eq. (1) and by looking at the first column in the 2k setting against the columns of the losses, we observe that despite the growth of the loss values, a deeper and wider NN can better mimic the option values computed with implicit and explicit numerical methods. In other words, the optimization error ($\mathcal{E}_O$) is growing but overall does not impact the global error. Indeed, by using a larger network, the approximation error $\mathcal{E}_A$ is decreasing, while the generalization error $\mathcal{E}_G$ should increase. However, in this case, as the total error is decreasing, we presume that the sum of generalization and optimization errors $\mathcal{E}_G + \mathcal{E}_O$ increases slower than the approximation one $\mathcal{E}_A$.

Table 4
The table shows how the error varies according to the network dimension. In particular, four architectures are compared. Starting from that described in Eq. (45), we modify n_j^{sol} and n_j^{fb}. Accordingly to the difference in the architecture, also the PINN losses $\mathcal{L}^{sol}$ and $\mathcal{L}^{fb}$ (in the MSE sense) vary, as described in the column **Losses** (observe that the error is reported in the scale 10^{-3}). Finally, the comparison with the numerical methods, that is the L2 distance between PINN and numerical solutions, is shown. Three different resolutions are considered: 2000 (**2k**), 4000 (**4k**), and 8000 (**8k**). Furthermore, explicit and implicit methods have been used for each resolution.

Neural architecture	Losses (MSE) 10^{-3}		2k - L2 distance		4k- L2 distance		8k- L2 distance	
	Sol	Fb	Expl	Impl	Expl	Impl	Expl	Impl
5 × 25	0.9406	0.0101	0.0109	0.0109	0.0139	0.0138	0.0159	0.0157
10 × 50	0.4564	0.0485	0.0184	0.0184	0.0153	0.0153	0.0142	0.0141
20 × 100	1.024	0.051	0.0069	0.0069	0.0178	0.0178	0.0214	0.0212
40 × 200	7.5492	6.0605	0.0058	0.0058	0.0205	0.0205	0.0248	0.0245

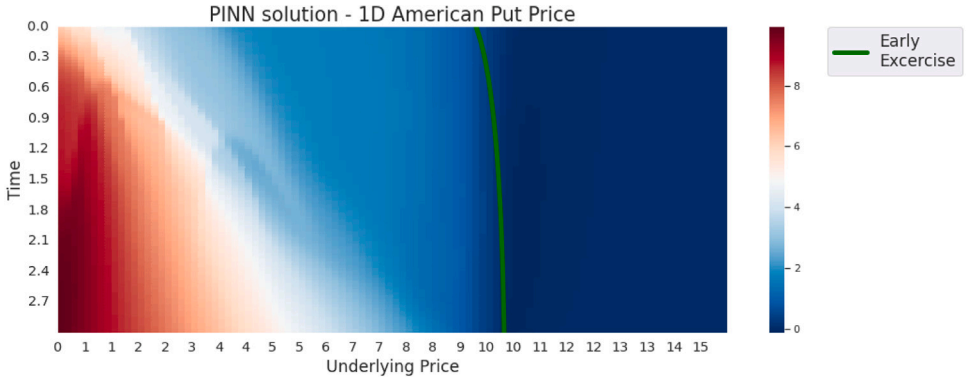

Fig. 3. PINN Solution - The solution found by the PINN zoomed for S values between 0 and 15. The green line represents the free boundary value.

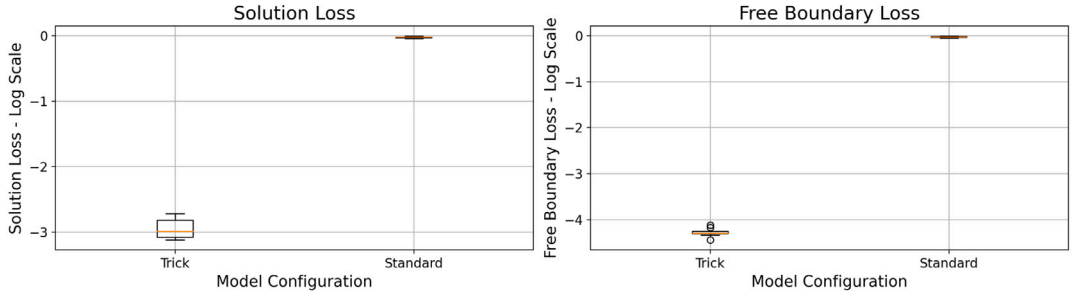

Fig. 4. Comparison of the results obtained in the test set with and without the algorithmic trick. On the left side, there is the model configuration that exploits several solution iterations for each free boundary step. On the right side, there is model 1 vs 1. The results are on a logarithmic scale.

Moreover, we found that the difference between the PINN and the finite difference technique grew when the mesh was refined (columns 4k and 8k). This is more due to the generalization error ($\mathcal{E}_G$) and could be balanced by optimizing the collocation points distribution. Furthermore, to make comparable observations, we consistently noted that the difference between the implicit method and the PINN was slightly lower than the difference between explicit methods and the PINN.

5.4. Algorithmic trick explanation

As stated in the previous section, increasing the number of FNN^{sol} iterations at each epoch helps improve PINN performance. In this subsection, we provide some evidence for this. In fact, early experiments highlight that it is possible to obtain better results both in training and in the test simply by performing multiple solution steps at each epoch. For example, Fig. 4 shows the box plots of the error in the test set of the 1D model obtained by changing the random seed. In particular, two different configurations are compared: on the left side, there is that with 20 FNN^{sol} iterations at each epoch; on the right side, there is the standard one with just 1 iteration.

The plot clearly shows better results for the model exploiting the algorithmic trick. This finding can be further analyzed by comparing Table 3 with Table 5, which shows the results obtained by the 1 iteration model. As shown, most of the error is related to the Neumann free boundary condition, which is not well approximated by the model. However, the other losses are also far from the results obtained with the algorithmic trick, showing, in the end, the advantage of using the algorithmic trick. This means that FNN^{fb} requires an accurate estimate of FNN^{sol} to work correctly. This could be because, while $\mathcal{L}^{sol}$ is made up of different components, some of them completely independent from FNN^{fb}, $\mathcal{L}^{fb}$ is almost entirely linked to the outputs of FNN^{sol}. So, to enhance the final performance, it can be helpful to exploit the "independence property" of the solution network to improve model quality in an early stage and then use this solution to train the free boundary.

As for the number of steps to exploit, Fig. 5 describes the loss behaviour as the number of steps changes. The figure shows the variation in the error with respect to the number of FNN^{sol} steps. Observe that, as the independent variable is defined as the number of FNN^{sol} iterations for each FNN^{fb}, we indicate with a negative value the case where multiple FNN^{fb} iterations are performed against a single FNN^{sol} step.

The plot shows a minimum point near the value of 10 iterations. However, in the experimental stage, we have exploited 20 iterations.

Table 5
Results obtained in the 1D experiment without using the algorithmic trick. In addition to the free boundary Neumann loss, which absorbs most of the error, the other losses are also considerable.

Seed	Unsupervised	Initial	Dirichlet	FB_Init	FB_Dir	FB_Neu	Free boundary	Solution
2	0.001185	0.002067	0.000765	0.001167	0.008013	0.927812	0.936993	0.939844
3	0.007435	0.002488	0.000324	0.005496	0.021174	0.882764	0.909435	0.914187
5	0.001562	0.005792	0.001443	0.000798	0.003142	0.947291	0.951232	0.959232
7	0.001093	0.001922	0.000555	0.009569	0.002094	0.953169	0.964833	0.958835
11	0.016059	0.001687	0.000488	0.006557	0.036044	0.856507	0.899110	0.910788
13	0.003254	0.004141	0.001464	0.000000	0.002567	0.962209	0.964777	0.973637
17	0.001395	0.015225	0.003493	0.001597	0.007564	0.927944	0.937105	0.955622
19	0.000241	0.002165	0.000320	0.000717	0.000380	0.983726	0.984824	0.986835
23	0.002310	0.004868	0.001023	0.000835	0.002508	0.941294	0.944638	0.952006
29	0.013918	0.001320	0.000157	0.015796	0.024725	0.885708	0.926230	0.925830
Mean	0.004845	0.004167	0.001003	0.004253	0.010821	0.926842	0.941918	0.947682
Median	0.001936	0.002327	0.000660	0.001382	0.005353	0.934619	0.940872	0.953814

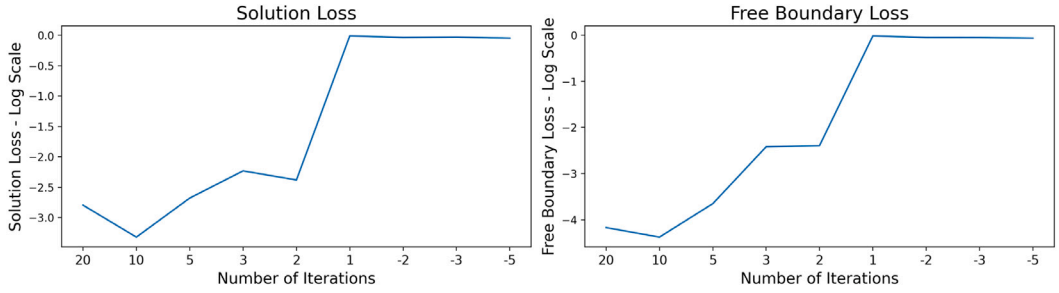

Fig. 5. Loss behaviour as the number of FNN^{sol} and FNN^{fb} iterations per epoch changes. A positive number means more FNN^{sol} iterations are executed against a single FNN^{fb} iteration. A negative number indicates more FNN^{fb} steps and just one FNN^{sol} iteration at each epoch. For example, 20 iterations mean 20 FNN^{sol} and 1 FNN^{fb} iterations at each epoch. Instead, −5 iterations mean 1 FNN^{sol} and 5 FNN^{fb} steps at each epoch.

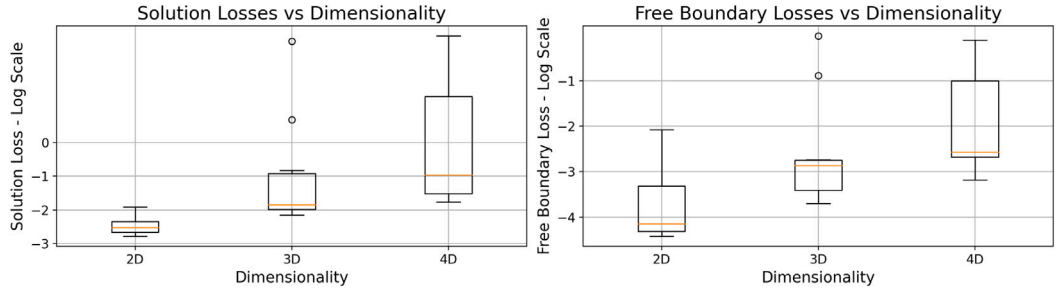

Fig. 6. Box plot of the logarithmic test losses. In the 1D case, the median value for the solution and free boundary networks are 0.000051 and 0.001024, respectively. As we can see, the loss increases as the dimension increase. Furthermore, it seems that also the uncertainty has become bigger.

This is because we have found this value more stable to variations in the other hyperparameters. For example. by changing the weights initialization to the so-called *Glorot Uniform* initialization, the performance for the model with 20 steps is nearly the same, with free boundary and solution losses equal to $7.8e-5$ and $1.8e-3$, respectively. Instead, there is a bigger degradation in the 10 steps model, whose errors are $\mathcal{L}^{fb} = 5e{-}4$ and $\mathcal{L}^{sol} = 2.3e-3$.

5.5. Multidimensional models - result and discussion

Fig. 6 shows the box plot of the test losses related to the solution and the free boundary networks, respectively $\mathcal{L}^{sol}$ and $\mathcal{L}^{fb}$.

As we can see, both the error and the uncertainty, at least for the solution network, grow as the dimension increases. We report each type of loss for every launch for 2D, 3D, and 4D models in Tables 6, 7, and 8, respectively. Furthermore, also the mean and median values for each experiment are shown.

In the 2D case, regarding the total solution error, we can observe how all launches give a loss in the order of 10^{-3}. The only exception is that random seed 11 gives an error one order bigger. This is mainly due to the Neumann condition at the free boundary, which exhibits a huge error. Instead, regarding the free boundary losses, the situation is more heterogeneous, with errors order between 10^{-5} and 10^{-3}. In general, for the 2D model, the error seems to be quite stable across epochs.

As for the 3D model, an increase in uncertainty is noticed. In fact, the error is considerably heterogeneous among different seeds. This is clear by looking at the solution loss, which we use as a benchmark as it is the sum of almost all the losses except $\mathcal{L}^{fb}_{init}$, which is usually negligible. $\mathcal{L}^{sol}$ can vary by several orders of magnitude between one seed and another. In this direction, it is particularly symbolic that the mean value of the loss is equal to 100, while the median value is just 0.01. Indeed, it is possible to identify just three anomalous launches. Two of them are characterized by an abnormal unsupervised loss, while for the third one, most of the error is related to the Neumann condition at the free boundary, which sometimes could be particularly hard to approximate, as also seen for the 2D case.

Finally, the 4D case confirms the trend previously observed, with a clear rise in the error uncertainty. In fact, the solution loss in some launches is very low, in the order of 0.01, while in others, it explodes up to 1000. By deeply looking into the errors, it can be noticed that in this case, too, the main error driver for the disastrous launches is the unsupervised loss.

Table 6
Summary of the losses regarding the 2D model.

Seed	Unsupervised	Initial	Dirichlet	FB_Init	FB_Dir	FB_Neu	Free boundary	Solution
2	0.001313	0.000570	0.000199	0.000000	0.000029	0.000008	0.000037	0.002120
3	0.001349	0.001619	0.000377	0.000000	0.000030	0.000016	0.000046	0.003393
5	0.000775	0.001436	0.000325	0.000001	0.000021	0.000052	0.000076	0.002611
7	0.000731	0.000580	0.000265	0.000000	0.000058	0.000006	0.000065	0.001643
11	0.001206	0.002278	0.000274	0.000117	0.001298	0.007064	0.008480	0.012123
13	0.001925	0.002310	0.000269	0.000002	0.000250	0.000486	0.000739	0.005242
17	0.001043	0.001180	0.000254	0.000000	0.000057	0.001971	0.002028	0.004507
19	0.001189	0.000788	0.000224	0.000002	0.000122	0.000015	0.000139	0.002339
23	0.000796	0.001120	0.000188	0.000000	0.000033	0.000007	0.000041	0.002146
29	0.003092	0.000929	0.000407	0.000004	0.000030	0.000018	0.000053	0.004478
Mean	0.001342	0.001281	0.000278	0.000013	0.000193	0.000964	0.001170	0.004060
Median	0.001198	0.001150	0.000267	0.000001	0.000045	0.000017	0.000070	0.003002

Table 7
Errors in the 3D experiments.

Seed	Unsupervised	Initial	Dirichlet	FB_Init	FB_Dir	FB_Neu	Free boundary	Solution
2	0.000228	0.005354	0.001135	0.000142	0.000140	0.000146	0.000428	0.007005
3	0.001368	0.007969	0.000558	0.000003	0.000125	0.000071	0.000200	0.010093
5	4.755147	0.009160	0.001154	0.000040	0.000124	0.001469	0.001633	4.767055
7	0.003328	0.008617	0.001089	0.000057	0.000113	0.000204	0.000375	0.013354
11	0.000975	0.011386	0.001243	0.000073	0.000465	0.001207	0.001746	0.015278
13	0.003540	0.004172	0.001396	0.000146	0.000756	0.000273	0.001176	0.010139
17	1001.154	0.008811	0.001946	0.003124	0.054935	0.931168	0.989228	1002.150
19	0.000205	0.009785	0.000803	0.000023	0.000166	0.000052	0.000243	0.011014
23	0.039958	0.016549	0.001872	0.000032	0.000557	0.001233	0.001823	0.060171
29	0.000138	0.017036	0.001465	0.000035	0.001274	0.130187	0.131497	0.150102
Mean	100.5959	0.009884	0.001266	0.000367	0.005866	0.106601	0.112835	100.7195
Median	0.002348	0.008986	0.001198	0.000048	0.000316	0.000740	0.001404	0.014316

Table 8
Test set values for the 4D models.

Seed	Unsupervised	Initial	Dirichlet	FB_Init	FB_Dir	FB_Neu	Free boundary	Solution
2	0.000215	0.012743	0.003536	0.000045	0.000444	0.000173	0.000662	0.017113
3	0.006100	0.007089	0.003504	0.000382	0.000967	0.000678	0.002028	0.018339
5	0.074205	0.018106	0.003108	0.000059	0.001411	0.001128	0.002598	0.097959
7	0.119458	0.015205	0.008666	0.002461	0.000103	0.000205	0.002771	0.143640
11	0.001475	0.019099	0.003976	0.000134	0.000900	0.000851	0.001886	0.026303
13	133.0739	0.030052	0.011353	0.000126	0.081407	0.118090	0.199625	133.3148
17	1448.024	0.013883	0.005416	0.002588	0.119397	0.678356	0.800342	1448.841
19	0.000872	0.027174	0.009227	0.001819	0.001265	0.009992	0.013077	0.048532
23	0.093390	0.017579	0.007845	0.000000	0.001218	0.001088	0.002308	0.121124
29	127.1520	0.019168	0.012434	0.000627	0.074705	0.190620	0.265953	127.4489
Mean	170.8545	0.018010	0.006906	0.000824	0.028182	0.100118	0.129125	171.0078
Median	0.083797	0.017843	0.006630	0.000258	0.001242	0.001108	0.002684	0.109541

5.6. Multidimensional models - theoretical notes

Understanding how a network correctly approximates a function as the dimension rises is connected to how well a network can approximate a function. Here we want to show that our PINN architecture can still provide a good approximation as the dimension of the problem grows. Regarding the theory of neural network approximation, the main issue is understanding what size a neural network $\hat{u}$ should have to obtain a target error ϵ. Starting from the papers [48,49], we know that shallow (depth-3) networks with smooth activation (such as $tanh$) complexity scales as $\sim \epsilon^{-d/n}$. These results were extended by [50], and a first rough estimate is that for every $\epsilon > 0$ and given a function $f \in \mathcal{W}^{n,\infty}([0,1]^d) \subset C^2([0,1]^d)$, for $n > 2$, there is a neural network $\hat{f}$ with $O(\epsilon^{-d/n})$ neurons such that $\hat{f}$ approximates f.

However, the provided estimates in the literature are dependent on the dimension d. In fact, high-dimensional PDEs solutions cannot be approximated due to the curse of dimensionality, which means the neural network size scales exponentially in the input dimension. Nevertheless, for some PDEs, the solution at a fixed time can be approximated with a network of polynomial size. More recently [51] have demonstrated that a PINN can overcome the curse of dimensionality for linear Kolmogorov PDEs, which include the heat equation and the Black–Scholes equation. Even though the paper's hypotheses are not valid in our architecture, the results show that the error does not grow exponentially for each best-case scenario, implying that similar properties may hold true for other hypotheses and similar Black–Scholes equations.

We eventually evaluated and updated the optimizer in our test, specifically for the multidimensional scenario, to better understand the optimization error (1), i.e., the greater loss values that may be detected. We noticed in our studies that even if the loss is more significant as an absolute value greater than 1D, this does not necessarily imply that the global error is increasing. Indeed, by using Adam as an optimizer, the magnitude of the loss is $7.7 * 10^{-3}$, against $3 * 10^{-3}$ obtained with RMSprop (see Table 3) and so more than doubled. The numerical value of the loss function impacts on the global error. So, in this framework, it is strongly suggested a proper optimizer choice.

Moreover, the authors in [52] prove that the global error exponentially grows as the problem dimension increases. In our tests, we let the NN have the same size in all the dimensions, so it is reasonable that the loss increases exponentially with the dimension, as in Fig. 6. However, more investigations are needed in the case of the American option pricing problem.

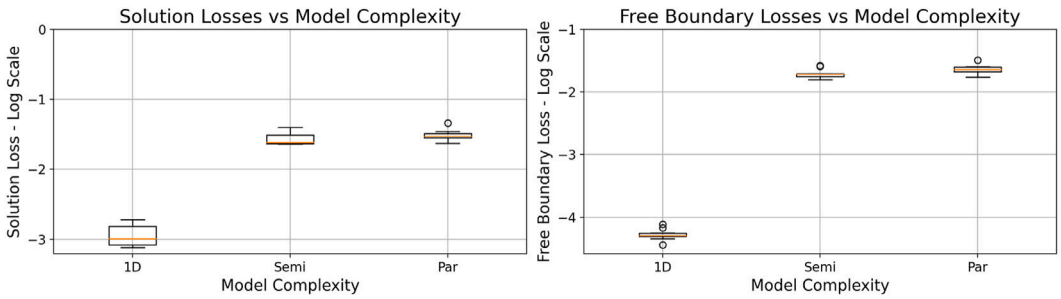

Fig. 7. Box plot of the logarithmic test losses. As we can see, the loss increases as the complexity increase. Furthermore, it seems that also the uncertainty has become more significant.

Table 9
Test result regarding the semi-parametric model.

Seed	Unsupervised	Initial	Dirichlet	FB_Init	FB_Dir	FB_Neu	Free boundary	Solution
2	0.004564	0.001568	0.000001	0.001256	0.012349	0.005589	0.019195	0.024073
3	0.003025	0.002985	0.000001	0.001131	0.019425	0.005643	0.026200	0.031081
5	0.007274	0.001623	0.000000	0.000351	0.010848	0.004547	0.015747	0.024294
7	0.003391	0.001998	0.000000	0.001697	0.012157	0.005460	0.019315	0.023009
11	0.004189	0.001868	0.000002	0.000580	0.012227	0.004893	0.017701	0.023180
13	0.006404	0.002659	0.000002	0.000641	0.009433	0.006350	0.016425	0.024850
19	0.003016	0.001333	0.000000	0.000582	0.011739	0.007168	0.019489	0.023257
23	0.004506	0.002790	0.000000	0.000229	0.017123	0.008327	0.025679	0.032747
29	0.019027	0.001769	0.000001	0.000108	0.014384	0.004530	0.019023	0.039714
Mean	0.006155	0.002066	0.000001	0.000730	0.013298	0.005834	0.019864	0.027356
Median	0.004506	0.001868	0.000001	0.000582	0.012227	0.005589	0.019195	0.024294

Table 10
Result obtained in the test set by the parametric model.

Seed	Unsupervised	Initial	Dirichlet	FB_Init	FB_Dir	FB_Neu	Free boundary	Solution
2	0.006199	0.001715	0.000004	0.000599	0.014236	0.005781	0.020618	0.027939
3	0.004264	0.002243	0.000000	0.001448	0.018906	0.004794	0.025149	0.030210
5	0.010786	0.001908	0.000000	0.000497	0.011879	0.009859	0.022236	0.034434
7	0.004182	0.001890	0.000001	0.001611	0.013991	0.009255	0.024858	0.029321
11	0.005101	0.001703	0.000001	0.000726	0.016519	0.005201	0.022447	0.028527
13	0.004694	0.002054	0.000003	0.000348	0.010483	0.006320	0.017152	0.023556
17	0.005508	0.001964	0.000003	0.000463	0.015245	0.009160	0.024870	0.031882
19	0.003898	0.001917	0.000000	0.000647	0.013734	0.005116	0.019498	0.024665
23	0.012714	0.002355	0.000000	0.001236	0.027062	0.003963	0.032263	0.046096
29	0.007378	0.002292	0.000001	0.000421	0.017068	0.005851	0.023341	0.032593
Mean	0.006472	0.002004	0.000001	0.000800	0.015912	0.006530	0.023243	0.030922
Median	0.005304	0.001940	0.000001	0.000623	0.014741	0.005816	0.022894	0.029765

5.7. Parametric models - result and discussion

Now, we report the result obtained when working with the parametric (Par) and semi-parametric (Semi) models, as described in the previous section. The box-plot in Fig. 7 describes error evolution when moving from the standard 1D model to semi-parametric and parametric models.

The figure, as expected, clearly shows a considerable decline in performance when moving from the non-parametric to the semi-parametric model. Instead, the difference between semi-parametric and parametric models is far more contained. This can be explained by considering that the two most sensible parameters are asset volatility and strike, particularly the first. In other words, the network's main difficulties are in understanding how volatility affects option price. So, most of the error is related to σ. As both the models investigate σ, there is not a big difference between them.

Tables 9 and 10 show the test result across the seed and the mean and median values.

Regarding the single losses, most of the error is related to the Dirichlet condition for the free boundary. In fact, the free boundary initial condition is two orders of magnitude lower, while the Neumann condition is less than one-half the Dirichlet error. Furthermore, the free boundary loss is more than one-half the solution error. In other words, the contribution given by $\mathcal{L}^{fb}_{Dir}+\mathcal{L}^{fb}_{Neu}$ is greater than $\mathcal{L}^{sol}_{PDE}+\mathcal{L}^{sol}_{init}+\mathcal{L}^{sol}_{Dir}$. This is a turnaround with respect to the non-parametric model, where the free boundary error is about two orders of magnitude lower than the total solution loss. This means that the mobility of the parameters, especially K, causes more difficulties for the network to understand such conditions.

5.8. Computational time

We record the computational time in every experiment. Fig. 8 shows median time performance.

In particular, three plots are shown, concerning train time, train time per epoch and test time. The first observation is related to the difference between the computational time per epoch required in the 1D and 2D experiments. The first case is far more expensive. This is due to the number of steps per epoch. In fact, while in the 1D case, there are 20 steps of solution for each free boundary, in the multi-dimensional cases, there are just 4. Then, the computational time required per epoch between 2D, 3D, and 4D experiments seems nearly linear. It increases not so much for the number of parameters (which are very close in that only solution input and free boundary output layers change) but mainly for the computation complexity at each stage and for the number of collocation points. In fact, the differential operator involves

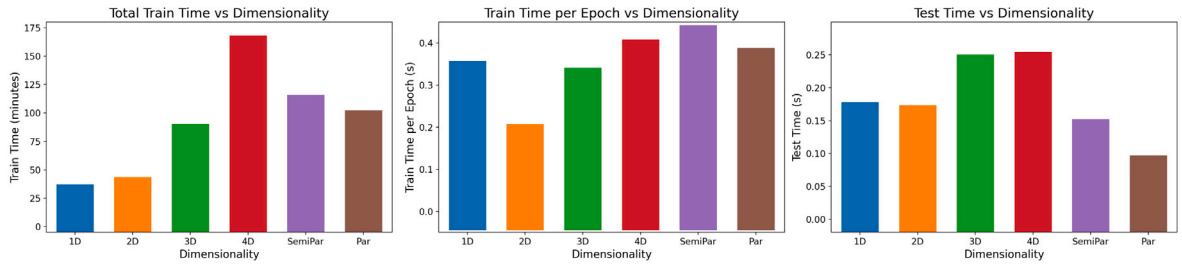

Fig. 8. Computational time. The median time (across the seed) is shown for each experiment. The figure gives information about training time, training per epoch, and test time. As reported on the *y*-axis, test times are drastically lower than training times. Thus, most of the computational effort has to be put only in the training stage, and a properly trained model can be easily used to perform multiple simulations by varying input vectors.

more operations as the number of underlying assets increases, and the training sets become even bigger, as shown in Table 2.

The exponential growth in the total training time is given by the above reasons and the increase in the number of epochs. Finally, as for the test time, it should be noted that they are dramatically lower than the training time. This is a general peculiarity of neural networks. In particular, by looking at the *y*-axis and keeping in mind the dimension of the test set, it can be noticed that in the worst case, about 1 millisecond for 100 points evaluation is needed. This makes PINNs extremely useful if the aim is to perform multiple evaluations once the model is trained. Instead, the computational time is more severe and prohibitive if just a few evaluations have to be performed.

6. Conclusion

In this work, we apply a recent Deep Learning paradigm, namely the Physical-Informed Neural Network, to study the American option pricing problem. Currently, no analytical solution exists for this problem, so its study has attracted both academics and practitioners, giving rise to increasing literature. Our strategy relies on the neural network universal approximation property, which theoretically justifies exploiting Deep Learning to approximate the price function. In particular, the American option pricing problem is mathematically described as a parabolic partial differential equation with free boundary conditions, whose solution is the unknown price function. Two concurrent neural networks are used to approximate the solution and the boundary. This is because American options are path-dependent, and in particular, not only the solution but also the domain boundary is unknown. The training process is enhanced thanks to an algorithmic trick, for the first time, exploited in the PINN context. The trick consists in applying more solution training steps for each free boundary iteration.

The experimental stage aims to assess the reliability of the proposed models. Firstly, the one-dimensional case is studied, also by providing evidence for the algorithmic trick's usefulness. Then, the analysis is generalized to multi-asset options. Finally, parametric models are discussed and developed. The idea behind the latter class of models is to give up accuracy and computational time in order to allow the networks to understand the parameters' role. In this way, the PINN should be able to handle risk-free rate, asset volatility, maturity, and strike, as well as the underlying asset price or time. So, we expect to find PINNs that are easily exploitable to perform model calibration.

The last consideration is about possible further directions of analysis. Firstly, the study of the parametric model should be examined in depth, also by comparison with real-market data. Then, it would be interesting to analyze this framework also with different contracts or pricing models. Indeed, both affect the underlying problem, adding new features, such as the curse of dimensionality, strongly non-linear PDE, more complex boundary conditions, or different types of path dependency. Dealing with all these issues is helpful to have a more exhaustive overview of the power of PINNs in Finance, as there is still little literature on this way. Finally, from a theoretical point of view, our proposal can benefit, without doubt, from a deeper analysis of the approximation error and its relationship with the computational time, maybe also increasing the number of considered dimensions.

Code availability statement

The Python code that supports this study is publicly available at the following GitHub repository: https://github.com/fgt996/Meshless-Methods-for-American-Option-Pricing-by-using-Physics-Informed-Neural-Networks.

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.

Data availability

The Python code that supports this study is publicly available in a GitHub repository, as stated in the main document.

References

[1] Black F, Scholes M. The pricing of options and corporate liabilities. J Polit Econ 1973;81(3):637–54.
[2] Kaya D. Pricing a multi-asset American option in a parallel environment by a finite element method approach. 2011.
[3] Ikamari C, Ngare P, Weke P. Multi-asset option pricing using an information-based model. Sci Afr 2020;10:e00564.
[4] Quarteroni A. Numerical models for differential problems, vol. 2. Springer; 2009.
[5] Raissi M, Perdikaris P, Karniadakis GE. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. J Comput Phys 2019;378:686–707.
[6] Hornik K, Stinchcombe M, White H. Multilayer feedforward networks are universal approximators. Neural Netw 1989;2(5):359–66.
[7] Hornik K. Approximation capabilities of multilayer feedforward networks. Neural Netw 1991;4(2):251–7.
[8] Moon K-S, Kim W-J, Kim H. Adaptive lattice methods for multi-asset models. Comput Math Appl 2008;56(2):352–66.
[9] Podlozhnyuk V, Harris M. Monte carlo option pricing. In: CUDA SDK. 2008.
[10] Wilmott P. Paul Wilmott on quantitative finance. John Wiley & Sons; 2013.
[11] Niklasson V, Tivedal F. Multi-asset options: a numerical study [Master's thesis], 2018.
[12] Duffy DJ. Finite difference methods in financial engineering: A partial differential equation approach. John Wiley & Sons; 2013.
[13] Han J. Pricing some American multi-asset options. 2009.
[14] Brennan MJ, Schwartz ES. An equilibrium model of bond pricing and a test of market efficiency. J Financ Quant Anal 1982;17(3):301–29.
[15] Hilber N, Reichmann O, Schwab C, Winter C. Computational methods for quantitative finance: Finite element methods for derivative pricing. Springer Science & Business Media; 2013.
[16] Ruf J, Wang W. Neural networks for option pricing and hedging: a literature review. 2019, arXiv preprint arXiv:1911.05620.
[17] Sirignano J, Spiliopoulos K. DGM: A deep learning algorithm for solving partial differential equations. J Comput Phys 2018;375:1339–64.
[18] Chan-Wai-Nam Q, Mikael J, Warin X. Machine learning for semi linear PDEs. J Sci Comput 2019;79(3):1667–712.

[19] Han J, Jentzen A, et al. Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. Commun Math Stat 2017;5(4):349–80.
[20] Han J, Jentzen A, Weinan E. Solving high-dimensional partial differential equations using deep learning. Proc Natl Acad Sci 2018;115(34):8505–10.
[21] Rad JA, Parand K, Ballestra LV. Pricing European and American options by radial basis point interpolation. Appl Math Comput 2015;251:363–77.
[22] Ahmad I, et al. A comparative analysis of local meshless formulation for multi-asset option models. Eng Anal Bound Elem 2016;65:159–76.
[23] Beck C, Becker S, Cheridito P, Jentzen A, Neufeld A. Deep splitting method for parabolic PDEs. SIAM J Sci Comput 2021;43(5):A3135–54.
[24] Tanios R. Physics informed neural networks in computational finance: High dimensional forward & inverse option pricing [Master's thesis], 2021.
[25] Louskos A. Physics-informed neural networks and option pricing [Master's thesis].
[26] Bai Y, Chaolu T, Bilige S. The application of improved physics-informed neural network (IPINN) method in finance. Nonlinear Dynam 2022;107(4):3655–67.
[27] Kim S, Yun S-B, Bae H-O, Lee M, Hong Y. Physics-informed convolutional transformer for predicting volatility surface. 2022, arXiv preprint arXiv:2209.10771.
[28] von Rueden L, Mayer S, Beckh K, Georgiev B, Giesselbach S, Heese R, et al. Informed machine learning - A taxonomy and survey of integrating prior knowledge into learning systems. IEEE Trans Knowl Data Eng 2021;1. http://dx.doi.org/10.1109/TKDE.2021.3079836.
[29] Karniadakis GE, Kevrekidis IG, Lu L, Perdikaris P, Wang S, Yang L. Physics-informed machine learning. Nat Rev Phys 2021;3(6):422–40. http://dx.doi.org/10.1038/s42254-021-00314-5.
[30] Cuomo S, Di Cola VS, Giampaolo F, Rozza G, Raissi M, Piccialli F. Scientific machine learning through physics–informed neural networks: Where we are and what's next. J Sci Comput 2022;92(3):88. http://dx.doi.org/10.1007/s10915-022-01939-z.
[31] Mathematical aspects of deep learning. Cambridge University Press; 2022, http://dx.doi.org/10.1017/9781009025096.
[32] Berner J, Grohs P, Kutyniok G, Petersen P. The modern mathematics of deep learning. 2022, p. 1–111. http://dx.doi.org/10.1017/9781009025096.002, arXiv:2105.04026 [cs, stat].
[33] Kutyniok G. The mathematics of artificial intelligence. 2022, arXiv:2203.08890 [Cs, Math, Stat].
[34] Shuka S. Finite-difference methods for pricing the American put option. 2001, Advanced Mathemathics if Finance Honours Project.
[35] Chen Y. Numerical methods for pricing multi-asset options. Toronto: University of Toronto; 2017, p. 1–75.
[36] Wang S, Perdikaris P. Deep learning of free boundary and Stefan problems. J Comput Phys 2021;428:109914.
[37] Cai S, Wang Z, Wang S, Perdikaris P, Karniadakis GE. Physics-informed neural networks for heat transfer problems. J Heat Transfer 2021;143(6). http://dx.doi.org/10.1115/1.4050542.
[38] Goodfellow I, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, Ozair S, et al. Generative adversarial nets. Adv Neural Inf Process Syst 2014;27.
[39] Liu S, Borovykh A, Grzelak LA, Oosterlee CW. A neural network-based framework for financial model calibration. J Math Ind 2019;9(1):1–28.
[40] Hull JC. Options futures and other derivatives. Pearson Education India; 2003.
[41] Breeden DT, Litzenberger RH. Prices of state-contingent claims implicit in option prices. J Bus 1978;621–51.
[42] Sobol I. The distribution of points in a cube and the accurate evaluation of integrals. Zh Vychisl Mat I Mater Phys 1967;7:784–802 [in Russian].
[43] Glorot X, Bengio Y. Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the thirteenth international conference on artificial intelligence and statistics. 2010, p. 249–56, JMLR Workshop and Conference Proceedings.
[44] Hinton G, Srivastava N, Swersky K. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. 2012, p. 2, Cited on 14 (8).
[45] Krishnapriyan A, Gholami A, Zhe S, Kirby R, Mahoney MW. Characterizing possible failure modes in physics-informed neural networks. Adv Neural Inf Process Syst 2021;34:26548–60.
[46] Wang C, Li S, He D, Wang L. Is L^2 physics-informed loss always suitable for training physics-informed neural network? 2022, arXiv preprint arXiv:2206.02016.
[47] Everett J. Black-Scholes PDE solver. 2022, URL https://github.com/jeffeverett/black-scholes-pde.
[48] Mhaskar HN. Neural networks for optimal approximation of smooth and analytic functions. Neural Comput 1996;8:164–77.
[49] Maiorov V, Meir R. On the near optimality of the stochastic approximation of smooth functions by neural networks. Adv Comput Math 2000;13(1):79–103. http://dx.doi.org/10.1023/A:1018993908478.
[50] Yarotsky D. Error bounds for approximations with deep ReLU networks. Tech rep, 2017, http://dx.doi.org/10.48550/arXiv.1610.01145, arXiv:1610.01145 [cs] type: article.
[51] De Ryck T, Mishra S. Generic bounds on the approximation error for physics-informed (and) operator learning. Tech rep, 2022, http://dx.doi.org/10.48550/arXiv.2205.11393, arXiv:2205.11393 [cs, math] type: article.
[52] De Ryck T, Mishra S. Error analysis for physics informed neural networks (PINNs) approximating Kolmogorov PDEs. 2021, arXiv:2106.14473 [cs, Math].