Journal of Computational and Applied Mathematics 425 (2023) 115041

Contents lists available at ScienceDirect

Journal of Computational and Applied Mathematics

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

Boundary-safe PINNs extension: Application to non-linear parabolic PDEs in counterparty credit risk

Joel P. Villarino, Álvaro Leitao *, J.A. García Rodríguez

M2NICA Research Group, University of Coruña, Spain
CITIC Research Center, Spain

ARTICLE INFO

Article history:
Received 30 September 2022
Received in revised form 21 December 2022

MSC:
68T07
35Q91
65M99
91G20

Keywords:
Deep learning
PDEs
PINNs
Boundary conditions
Nonlinear
Couterparty credit risk

ABSTRACT

The goal of this work is to develop a novel strategy for the treatment of the boundary conditions for multi-dimension nonlinear parabolic PDEs. The proposed methodology allows to get rid of the heuristic choice of the weights for the different addends that appear in the loss function related to the training process. It is based on defining the losses associated to the boundaries by means of the PDEs that arise from substituting the related conditions into the model equation itself. The approach is applied to challenging problems appearing in quantitative finance, namely, in counterparty credit risk management. Further, automatic differentiation is employed to obtain accurate approximation of the partial derivatives, the so called Greeks, that are very relevant quantities in the field.

1. Introduction

Deep learning techniques are machine learning algorithms based on neural networks, also known as artificial neural networks (ANNs), and representation learning, see [1] and the references therein. From a mathematical point of view, ANNs can be interpreted as multiple chained compositions of multivariate functions, and deep neural networks is the term used for ANNs with several interconnected layers. Such networks are known for being universal approximators, property given by the Universal Approximation Theorem, which essentially states that any continuous function in any dimension can be represented to arbitrary accuracy by means of an ANN (although the result does not provide statement regarding the rate of convergence in accuracy). For this reason, ANNs have a wide range of application, and their use has become ubiquitous in many fields: computer vision, natural language processing, autonomous vehicles, etc. Deep learning algorithms are usually classified according to the amount and type of supervision they get during training and, among all the categories that can be identified, we highlight the supervised and the unsupervised algorithms. They differ in whether they receive the desired solutions in the training set or not.

The aforementioned universal approximation property was exploited in the seminal papers [2–4] to introduce a technique to solve partial differential equations (PDEs) by means of ANNs. In recent years there has been a growing interest in approximating the solution of PDEs by means of deep neural networks. They promise to be an alternative to classical methods such as Finite Differences (FD), Finite Volumes (FV) or Finite Elements (FE). For example, the FE technique

* Corresponding author.
E-mail addresses: joel.perez.villarino@udc.es (J. P. Villarino), alvaro.leitao@udc.es (Á. Leitao), jose.garcia.rodriguez@udc.es (J.A. García Rodríguez).

https://doi.org/10.1016/j.cam.2022.115041

consists in projecting the solution in some functional space, the Galerkin spaces. Then, by passing to the weak variational formulation and taking the discrete basis, we can find a linear system of equations whose unknowns are the approximated values of the solution as each point of the mesh. In a similar manner, the ANN can be trained to learn data from a physical law that is given by a PDE or a system of PDEs. The idea is quite similar to the classical Galerkin methods, but instead of representing the solution as a projection in some flavour of Galerkin space, the solution is written in terms of ANNs as the composition of nonlinear functions depending on some network weights. As a result, instead of a high dimensional linear system, a high dimensional nonlinear optimization problem is obtained for the ANN weights. This problem must be solved using nonlinear optimization algorithms such as stochastic gradient descent-based methods, e.g., [5], and/or quasi-Newton methods, e.g., L-BFGS, [6]. More recently, with the advances in automatic differentiation algorithms (AD) and hardware (GPUs), this kind of techniques have gained more momentum in the literature and, currently, the most promising approach is known as physics-informed neural networks (PINNs), see [7–11].

In the last few years, PINNs have shown a remarkable performance. However, there is still some room for improvements within the methodology. One of the disadvantages of PINNs is the lack of theoretical results to control the approximation error. Obtaining error estimates or results for the order of approximation in PINNs is a non-trivial task, much more challenging than in classical methods. Even so, the authors in [10,12–17] (among others) have derived estimates and bounds for the so-called generalization error considering particular models. Another drawback is the difficulty when imposing the boundary conditions (a fact discussed further later in this section). Nevertheless the use of ANNs has several advantages for solving PDEs: they can be used for nonlinear PDEs without any extra effort; they can be extended to (moderate) high dimensions; and they yield accurate approximations of the partial derivatives of the solution thanks to the AD modules provided by modern deep learning frameworks.

PINNs is not the only approach relying on ANNs to address PDE-based problems. They can be used as a complement for classical numerical methods, for example training the neural network to obtain smoothness indicators, or WENO reconstructions in order for them to be used inside a classical FV method, see [18]. Also ANNs are being used to solve PDE models by means of their backward stochastic differential equation (BSDE) representation as long as the Feynman-Kàc theorem can be applied, which is the usual situation in computational finance, for example. In [19], the authors present the so called DeepBSDE numerical methods and their application to the solution of the nonlinear Black–Scholes equation, the Hamilton–Jacobi–Bellman equation, and the Allen–Cahn equation in very high (hundreds of) dimensions. The connection of such method with the recursive multilevel Picard approximations allows the authors to prove that DeepBSDEs are capable of overcoming the so called "curse of dimensionality" for a certain kind of PDEs, see [20,21].

The main goal of the present work is to develop robust and stable deep learning numerical methods for solving nonlinear parabolic PDE models by means of PINNs. The motivation arises from the difficulty of finding and numerically imposing the boundary conditions, which are always delicate and critical tasks both in the classical FD/FV/FE setting and thus also in the ANN setting. The common approach consists in assigning weights to the different terms involved in the loss function, where the selection of this weights can be done heuristically or by means of optimization procedures, see, for example, [22] and the references therein. We introduce a new idea that consists in introducing the loss terms due to the boundary conditions by means of evaluating the PDE operator restricted to the boundaries. In this way the value of such addends is of the same magnitude of the interior losses. Although this is non feasible in the classical PDE solving algorithms, it is very intuitive within the PINNs framework since, by means of AD, we can evaluate this operator in the boundary even in the case it contains normal derivatives to such a boundary. Thus, this novel treatment of the boundary conditions in PINNs is the main contribution of this work, differing substantially from other boundary approaches such as the presented in [23]. Both try to avoid, as far as possible, adjusting the hyperparameters that control the contribution of each addend to the loss function, but by different mechanisms. In [23], the contribution of the Dirichlet boundaries and initial condition are avoided in the loss function by exploiting the fact that they can be imposed as "hard constrains". In our approach the boundary operators are reformulated, based on the given PDE, with the aim to get rid the choice of such weights. In this way, our framework is more generalist since more boundary conditions fit into it and imposing hard constrains become unfeasible as the dimension of the problem grows. Further, AD can be naturally exploited to obtain accurate approximations of the partial derivatives of the solution with respect to the input parameters (quantities of much interest in several fields).

Although the proposed methodology could be presented for a wide range of applications, here we will focus on the solution of PDE models for challenging problems appearing the computational finance field. In particular, we consider the derivative valuation problem in the presence of counterparty credit risk (CCR), which includes in its formulation the so-called x-value adjustments (XVA). These terms refer to the different valuation adjustments that arise in the models when the CCR is considered, i.e., when the possibility of default of the parties involved in the transaction is taking into account. After the 2007–2009 financial crisis, CCR management became of key importance in the financial industry. A vast literature can be found on this topic, among which we highlight [24–42].

Of course, the world of quantitative finance in general, and CCR management in particular, has not been exempt from the advances in deep learning and, nowadays, ANNs are employed for a wide variety of tasks in the industry. Unsupervised ANNs, in both flavours, PINNs and DeepBSDEs, have been recently used for solving financial problems. For example, in [43] the authors apply PINNs for solving the linear one and two dimensional Black–Scholes equation, and [44] introduces the solution of high dimensional Black–Scholes problems using BSDEs. In [45] the authors present a novel computational framework for portfolio-wide risk management problems with a potentially large number of risk

factors that makes traditional numerical techniques ineffective. They use a coupled system of BSDEs for XVA which is addressed by a recursive application of an ANN-based BSDE solver. Other relevant works that make use of ANNs for computational finance problems, although not formulated as PDEs, include [46,47], or [48], for example.

The outline of this paper is as follows. In Section 2 we start by revisiting the PINNs framework for solving PDEs. Section 3 introduces the new methodology for the treatment of the boundary conditions in the PINNs setting. In Section 4, the XVA PDE models that we solve in this paper and the adaptation to our PINNs extension are described; more precisely, XVA problems under on Black–Scholes and Heston models. Finally, in Section 5, the numerical experiments that assess the accuracy of the approximation for option prices and their partial derivatives (the so-called Greeks) are presented.

2. PINNs

In this section we introduce the so-called PINNs methodology for solving PDEs. The illustration is carried out according to the kind of PDEs that arise in the selected financial problems, i.e., semilinear parabolic PDEs with source terms. Thus, let $\Omega \subset \mathbb{R}^d$, $d \in \mathbb{N}$, be a bounded, closed and connected domain and $T > 0$. Consider the following boundary value problem. Given a function $f \in \mathcal{C}(\mathbb{R})$ and setting $\hat{d} = d + 1$, find $u : (t, x) \in [0, T] \times \Omega \subset \mathbb{R}^{\hat{d}} \longrightarrow \mathbb{R}$ such that

$$\begin{cases} \dfrac{\partial u}{\partial t}(t,x) + \mathcal{L}[u](t,x) - f(u(t,x)) = 0, & \forall (t,x) \in (0,T) \times \overset{\circ}{\Omega}, \\ \mathcal{B}[u](t,x) - g(t,x) = 0, & \forall (t,x) \in (0,T) \times \partial\Omega, \\ u(0,x) - u_0(x) = 0, & \forall x \in \Omega, \end{cases} \tag{1}$$

where $\mathcal{L}[\cdot]$ is a strongly elliptic differential operator of second order in the space variables x, and $\mathcal{B}[\cdot]$ is a boundary operator defined, for example, by a Dirichlet and/or Neumann boundary conditions. The goal is to approximate this unknown function u by means of a feed-forward neural network, $u_\theta(t, x) \coloneqq u(t, x; \theta)$, where $\theta \in \mathbb{R}^P$ are the network parameters.

2.1. Feed-forward neural networks

A feed-forward network is a map that transforms an input $y \in \mathbb{R}^{\hat{d}}$ into an output $z \in \mathbb{R}^m$ by means of the composition of a variable number, L, of vector-valued functions called layers. These consist of units (neurons), which are the composition of affine-linear maps with scalar non-linear activation functions, [1]. Thus, assuming a L-layer network with β_l neurons per layer, it admits the representation

$$h(y; \theta) \coloneqq h_L(\,\cdot\,, \theta^L) \circ h_{L-1}(\,\cdot\,, \theta^{L-1}) \circ \cdots \circ h_1(\,\cdot\,, \theta^1)(y), \tag{2}$$

where, for any $1 \leq l \leq L$,

$$h_l(z_l; \theta^l) = \sigma_l(W_l z_l + b_l), \quad W_l \in \mathbb{R}^{\beta_{l+1} \times \beta_l}, z_l \in \mathbb{R}^{\beta_l}, b_l \in \mathbb{R}^{\beta_{l+1}}, \tag{3}$$

with $z_1 = y$, $\beta_1 = \hat{d}$ and $\beta_L = m$.

Usually (and this is taken as a guideline in this paper) the activation functions are assumed to be the same in all layers except in the last one, where we consider the identity map, $\sigma_L(\,\cdot\,) = Id(\,\cdot\,)$. In addition, taking into account the nature of the problem, it is required that the neural network fulfils the differentiability conditions imposed by (1), requiring sufficiently smooth activation functions such as the sigmoid or the hyperbolic tangent, [49].

Lastly, it should be noted that a network as the one described above has $\hat{d} + m + \sum_{l=2}^{L-1} \beta_l$ neurons, with parameters $\theta_l = \{W_l, b_l\}$ per layer, yielding a total of

$$P = \sum_{l=1}^{L-1} (\beta_l + 1)\beta_{l+1} \tag{4}$$

parameters, which determine the network's capacity.

2.2. Loss function and training algorithm

In order to obtain an approximation of the function u by means of a neural network, u_θ, we need to find the network's parameters, $\theta \in \mathbb{R}^P$, that yields the best approximation of (1). This leads to a global optimization problem that can be written in terms of the minimization of a loss function, that measures how good the approximation is. The most popular choice for PINNs is to reduce the problem (1) to an unconstrained optimization problem, [3], leading to the family of loss functions involving the L^p, $p \in \{1, 2, \ldots, \infty\}$, error minimization of the interior, initial and boundary residuals. Considering the case $p = 2$, the loss function, $\mathcal{J}(\theta)$, is defined as

$$\mathcal{J}(\theta) \coloneqq \lambda_{\mathcal{I}} \left\| \mathcal{R}_\theta^{\mathcal{I}} \right\|^2_{L^2((0,T)\times\Omega)} + \lambda_{\mathcal{B}} \left\| \mathcal{R}_\theta^{\mathcal{B}} \right\|^2_{L^2((0,T)\times\partial\Omega)} + \lambda_{\mathcal{O}} \left\| \mathcal{R}_\theta^{\mathcal{O}} \right\|^2_{L^2(\Omega)},$$

or, equivalently,

$$\mathcal{J}(\theta) = \lambda_{\mathcal{I}} \int_0^T \int_{\Omega} \left|\mathcal{R}_\theta^{\mathcal{I}}(t,x)\right|^2 \, \mathrm{d}x\mathrm{d}t + \lambda_{\mathcal{B}} \int_0^T \int_{\partial\Omega} \left|\mathcal{R}_\theta^{\mathcal{B}}(t,x)\right|^2 \, \mathrm{d}\sigma_x\mathrm{d}t + \lambda_{\mathcal{O}} \int_{\Omega} \left|\mathcal{R}_\theta^{\mathcal{O}}(x)\right|^2 \, \mathrm{d}x, \tag{5}$$

where

$$\mathcal{R}_\theta^{\mathcal{I}}(t,x) := \frac{\partial u_\theta}{\partial t}(t,x) + \mathcal{L}[u_\theta](t,x) - f(u_\theta(t,x)), \qquad (t,x) \in (0,T) \times \mathring{\Omega}, \tag{6}$$

$$\mathcal{R}_\theta^{\mathcal{B}}(t,x) := \mathcal{B}[u_\theta](t,x) - g(t,x), \qquad (t,x) \in (0,T) \times \partial\Omega, \tag{7}$$

$$\mathcal{R}_\theta^{\mathcal{O}}(x) := u_\theta(0,x) - u_0(x), \qquad x \in \Omega, \tag{8}$$

account for the residuals of the equation, the boundary condition and the initial condition, respectively. The $\lambda_j \in \mathbb{R}^+$, $j \in \{\mathcal{I}, \mathcal{B}, \mathcal{O}\}$, are preset hyperparameters (or updateables during optimization) that allow to impose a weight to each addend of the loss, as can be seen in, e.g., [50,51]. Note that, for the computation of the residuals (6), (7), it is necessary to obtain the derivatives of the neural network with respect to the input space and time variables, well defined under the premise of using sufficiently smooth activation functions. Numerically, they are calculated with the help of AD modules, such those included in Tensorflow, [52], and Pytorch, [53]. Finally, the strategy followed in PINNs consists of minimizing the loss function (5), i.e, finding $\theta^* \in \Theta$ such that

$$\theta^* = \arg\min_{\theta \in \Theta} \mathcal{J}(\theta), \tag{9}$$

where $\Theta \subset \mathbb{R}^p$ is the set of admissible parameters.

Except for the simple cases, the integrals appearing in (5) must be computed numerically by means of quadrature rules, [10]. For this reason, we need to select a set of training points, $\mathcal{P} = \mathcal{P}_{\mathcal{I}} \cup \mathcal{P}_{\mathcal{B}} \cup \mathcal{P}_{\mathcal{O}}$, where

$$\begin{aligned}
\mathcal{P}_{\mathcal{I}} &= \{(t_i^{\mathcal{I}}, x_i^{\mathcal{I}})\}_{i=1}^{N_{\mathcal{I}}}, \quad (t_i^{\mathcal{I}}, x_i^{\mathcal{I}}) \in (0,T) \times \mathring{\Omega} \quad \forall i \in \{1,2,\dots,N_{\mathcal{I}}\},\\
\mathcal{P}_{\mathcal{B}} &= \{(t_i^{\mathcal{B}}, x_i^{\mathcal{B}})\}_{i=1}^{N_{\mathcal{B}}}, \quad (t_i^{\mathcal{B}}, x_i^{\mathcal{B}}) \in (0,T) \times \partial\Omega \quad \forall i \in \{1,2,\dots,N_{\mathcal{B}}\},\\
\mathcal{P}_{\mathcal{O}} &= \{(0, x_i^{\mathcal{O}})\}_{i=1}^{N_{\mathcal{O}}}, \quad x_i^{\mathcal{O}} \in \Omega \quad \forall i \in \{1,2,\dots,N_{\mathcal{O}}\},
\end{aligned}$$

acting as nodes in the quadrature formulas.

Clearly, the choice of the quadrature technique has a direct influence on how these points are selected, and may correspond to, for example, a suitable mesh for a trapezoidal quadrature rule, SOBOL low-discrepancy sequences, a latin hypercube sampling, etc. Moreover, such a choice is highly influenced by the problem's time–space dimension, being necessary to use random sampling in high-dimensional domains.

In general terms, we can define the quadrature rule to calculate the integral of a function $\phi : A \subset \mathbb{R}^{\hat{d}} \longrightarrow \mathbb{R}$, as

$$\Phi_M := \sum_{i=1}^{M} w_i \phi(y_i) \tag{10}$$

with $\{w_i\}_{i=1}^M \subset \mathbb{R}_+$ the weights and $\{y_i\}_{i=1}^M \subset A$ the nodes of the quadrature rule. This allows us to rewrite the loss function (5) taking into account the chosen discretization and quadrature as follows,

$$\hat{\mathcal{J}}(\theta) = \lambda_{\mathcal{I}} \sum_{i=1}^{N_{\mathcal{I}}} w_i^{\mathcal{I}} |\mathcal{R}_\theta^{\mathcal{I}}(t_i^I, x_i^I)|^2 + \lambda_{\mathcal{B}} \sum_{i=1}^{N_{\mathcal{B}}} w_i^{\mathcal{B}} |\mathcal{R}_\theta^{\mathcal{B}}(t_i^{\mathcal{B}}, x_i^{\mathcal{B}})|^2 + \lambda_{\mathcal{O}} \sum_{i=1}^{N_{\mathcal{O}}} w_i^{\mathcal{O}} |\mathcal{R}_\theta^{\mathcal{O}}(x_i^{\mathcal{O}})|^2. \tag{11}$$

From now on, we will call "training" the process of finding the minimum of the problem (9) with the loss function defined in (11). Even in the case of working with linear PDEs, where the defined functional would be convex, transferring the problem to the parameter space of the neural network yields a high dimensional and highly non-convex problem, [50]. As a consequence, the uniqueness of the solution is not guaranteed, and we can only expect to reach a sufficiently low local minima. For this reason, it is common to employ stochastic gradient descent-based methods, such as Adam, [5], or higher-order quasi-Newton optimizers, such as L-BFGS, [54]. In practice, it also implies that a proper choice of model hyperparameters, such as the network size or the learning rate, is essential to achieve a high degree of accuracy. Taking into account what has been explained throughout this section, we detail the steps to find a neural network that approximates the solution of the problem (1) in Algorithm 1.

In high dimensional problems, where the volume of the associated domain is extremely large, several modifications to the Algorithm 1 have been proposed, many of them are related to the selection and updating of the quadrature nodes during training. Among the most popular choices are the mini-batch sampling strategy where, at each training iteration, the set of quadrature nodes is resampled, see e.g., [55]; or the ones known as *residual-based sampling* [56], where the training set is refined based on the training losses.

Essentially, and except for some particularities, the optimization process in the case of PINNs is similar to that presented in any other supervised or unsupervised tasks in the field of deep learning. Thus, many of the techniques developed to

Algorithm 1 PINNs' training algorithm

Require: Select a set of training points $\mathcal{P}$, a quadrature rule and an optimization procedure. Define a number of training steps, N. Initialize a neural network , u_{θ^0}, with initial parameters θ^0.
Ensure: Find an approximate local minimum θ^* of (9)
1: **for** $k = 0, k{+}{+}, k < N$ **do**
2: $u_{\theta^k} \leftarrow u_{\theta^k}(t, x)$ ▷ Evaluate the neural network
3: $\mathcal{R}^{\mathcal{I}}_{\theta^k}, \mathcal{R}^{\mathcal{B}}_{\theta^k}, \mathcal{R}^{\mathcal{O}}_{\theta^k} \leftarrow u_{\theta^k}$ ▷ Compute the residuals
4: $\hat{\mathcal{J}}(\theta^k) \leftarrow \mathcal{R}^{\mathcal{I}}_{\theta^k}, \mathcal{R}^{\mathcal{B}}_{\theta^k}, \mathcal{R}^{\mathcal{O}}_{\theta^k}$ ▷ Compute the loss function
5: $\theta^{k+1} \leftarrow \theta^k$ ▷ Apply the optimizer step
6: **end for**

improve training in such areas can be trivially applied to our case, such as regularization techniques, [1], Dropout, [57], transfer learning, [58], or other strategies designed to improve the performance of the global optimizer. In particular, in the experiments performed in Section 5, we employ an adaptive learning rate schedule known as inverse time decay, [58], which follows the construction,

$$\epsilon_k = \frac{\epsilon_0}{1 + \delta k/a}, \tag{12}$$

where ϵ_0 is the initial learning rate, ϵ_k the learning rate at step k, δ the decay rate and a the decay step.

Once trained, the network serves as an approximate solution to problem (1). It can be evaluated at any point in the domain, and its derivatives can be calculated by AD in a few seconds.

Remark 1. One of the most popular quadrature techniques is Monte Carlo integration. On the one hand, it is a mesh-free method since the points are sampled randomly, making it suitable for high dimensional problems as it does not suffer from the curse of dimensionality. On the other hand, applied to the L^2 error expression (11), it gives rise to the mean squared error function, widely used in the deep learning's world.

If we consider a random set of collocation points and define the quadrature weights as

$$w_i^{\mathcal{I}} = \frac{|(0,T) \times \overset{\circ}{\Omega}|}{N_{\mathcal{I}}}, \qquad w_i^{\mathcal{B}} = \frac{|(0,T) \times \partial\Omega|}{N_{\mathcal{B}}}, \qquad w_i^{\mathcal{O}} = \frac{|\Omega|}{N_{\mathcal{O}}}, \tag{13}$$

and taking

$$\lambda_{\mathcal{I}} = \frac{\hat{\lambda_{\mathcal{I}}}}{|(0,T) \times \overset{\circ}{\Omega}|}, \qquad \lambda_{\mathcal{B}} = \frac{\hat{\lambda_{\mathcal{B}}}}{|(0,T) \times \partial\Omega|}, \qquad \lambda_{\mathcal{O}} = \frac{\hat{\lambda_{\mathcal{O}}}}{|\Omega|}, \tag{14}$$

with $\hat{\lambda_j} \in \mathbb{R}_+,\ j \in \{\mathcal{I}, \mathcal{B}, \mathcal{O}\}$, then we obtain

$$\hat{\mathcal{J}(\theta)} = \hat{\lambda_{\mathcal{I}}} MSE_{\mathcal{I}} + \hat{\lambda_{\mathcal{B}}} MSE_{\mathcal{B}} + \hat{\lambda_{\mathcal{O}}} MSE_{\mathcal{O}},$$

where

$$MSE_{\mathcal{I}} = \frac{1}{N_{\mathcal{I}}} \sum_{i=1}^{N_{\mathcal{I}}} |\mathcal{R}^{\mathcal{I}}_{\theta}(t_i^{\mathcal{I}}, x_i^{\mathcal{I}})|^2, \qquad MSE_{\mathcal{B}} = \frac{1}{N_{\mathcal{B}}} \sum_{i=1}^{N_{\mathcal{B}}} |\mathcal{R}^{\mathcal{B}}_{\theta}(t_i^{\mathcal{B}}, x_i^{\mathcal{B}})|^2, \qquad MSE_{\mathcal{O}} = \frac{1}{N_{\mathcal{O}}} \sum_{i=1}^{N_{\mathcal{O}}} |\mathcal{R}^{\mathcal{O}}_{\theta}(x_i^{\mathcal{O}})|^2,$$

which resembles the loss function employed in most of the works in this topic.

2.3. Convergence and generalization error bounds

With the growth of these methodologies, it is of increasing interest to derive convergence results, as they exist in the finite differences and finite elements world. There are works, such as [49], in which classical notions of consistency and stability are exploited to prove the strongly convergence of the minimizer to the solution of the linear second-order elliptic or parabolic problem, as the number of collocation points grows. This assumes a random discretization of the domain together with Monte Carlo integration.

However, most of the theoretical work on PINNs is dominated by the search for generalization error bounds, where the generalization error, $\mathcal{E}_G(\theta)$, is understood as the total error of the approximated solution, which in our case is given by the square root of the loss function (5), i.e.,

$$\mathcal{E}(\theta)_G^2 = \mathcal{J}(\theta),$$

and depends on the network parameters $\theta \in \Theta$. As discussed in the previous section, evaluating this expression requires the use of numerical integration methods with their respective quadrature points, $\mathcal{P}$. In this sense, the square root of

the discretized version of the loss function, given in (11), serves to approximate the generalization error and is known as training error, $\mathcal{E}_T(\theta, \mathcal{P})$.

Under this setting, we find several papers that attempt to bound the generalization error, for specific problems, in terms of the training error, the chosen quadrature rule, the number of collocation points and the stability of the underlying PDE. For example, such bounds are obtained for the linear Kolmogorov equation, [12], the equation related to the viscous scalar conservation laws and the semi-linear parabolic equation, [10], among others. Thus, under existence, uniqueness and regularity assumptions for the semi-linear parabolic case with Lipschitz nonlinearities, the Theorem 3.1 from [10] states that the generalization error can be estimated as

$$\mathcal{E}_G \leq C_1 \left(\mathcal{E}_T^{\mathcal{O}} + \mathcal{E}_T^{\mathcal{I}} + C_2 (\mathcal{E}_T^{\mathcal{B}})^{\frac{1}{2}} + (C_q^{\mathcal{O}})^{\frac{1}{2}} N_{\mathcal{O}}^{-\frac{\alpha_{\mathcal{O}}}{2}} + (C_q^{\mathcal{I}})^{\frac{1}{2}} N_{\mathcal{I}}^{-\frac{\alpha_{\mathcal{I}}}{2}} + C_2 (C_q^{\mathcal{B}})^{\frac{1}{4}} N_{\mathcal{B}}^{-\frac{\alpha_{\mathcal{B}}}{4}} \right),$$

where $\mathcal{E}_T^{\mathcal{X}}$ are the training errors which verify the relationship $(\mathcal{E}_T^{\mathcal{X}})^2 = \mathcal{R}_\theta^{\mathcal{X}}$, $\mathcal{X} \in \{\mathcal{O}, \mathcal{I}, \mathcal{B}\}$. In addition, $C_q^{\mathcal{X}} N_{\mathcal{X}}^{-\alpha_{\mathcal{X}}}$ are the bounds of the quadrature error related to the initial condition, interior domain and boundary, respectively; and C_1, C_2 are constants that depend on the regularity of the true solution and neural network approximation on the boundary, together with the temporal domain. This result is of special interest because its hypotheses fit within our general problem (1) and, furthermore, since we will work with a non-linear contractive source term, the result can be easily applied to the particular problems presented in Section 4.

In a recently published paper, [17] presents several error bounds in a more abstract framework. Taking sufficiently smooth domains and under the assumptions: (1) there exist a neural network that can approximate the solution of the time-dependent PDE at time T with a prescribed tolerance ϵ; and (2) the error of the PINNs algorithm can be bounded by means of the error related to its partial derivatives; the following theorem holds.

Theorem 1 (*[17]*). *Let $r, s \in \mathbb{N}$, let $u \in \mathcal{C}^{(s,r)}([0,T] \times \Omega)$ be the solution of the abstract time-dependent PDE with initial condition $u_0 \in L^2(\Omega)$ and let the above assumptions be satisfied. There exists a constant $C(s,r) > 0$ such that for every $M \in \mathbb{N}$ and $\epsilon > 0$ there exist a neural network $u_\theta : [0,T] \times \Omega \longrightarrow \mathbb{R}$, with the hyperbolic tangent as activation function, for which it holds that,*

$$\|u_\theta - u\|_{L^q([0,T]\times\Omega)} \leq C(\|u\|_{C^0} M^{-s} + \epsilon),$$

where M is the number of spatial intervals chosen in the discretization.

Moreover, this theorem includes an additional result in which the L^2-norm of the operator applied to the neural network is bounded. Both statements together imply that there exists a neural network for which the generalization error and the PINN's loss function can be made as small as possible. Since our framework is embedded within this abstract formulation, such a result ensures a solid theoretical foundation for our work.

3. Novel treatment of boundary conditions

Ideally, the loss function correctly captures how far away we are from the exact solution of the problem and how well the boundary restrictions are fulfilled, so that the optimization algorithm can get us close to a good local minima, at least. However, in practice, this situation is not always reproduced when applying numerical methods. In the case of PINNs we also have this problem and, although the reasons why this happens are poorly understood, previous works, such as [51], point to the fact that training is focused on getting a small PDE residual in the interior domain, while leading to large errors in the fitting of the boundary conditions. This suggest that the contribution of the some boundary errors vanishes.

In most of the works on this topic this problem is usually solved by introducing the lambda weights seen before, which preponderate the contribution of each of the terms involved in the elaboration of the loss function (11). The optimal choice of this weights is of paramount importance for the algorithm. The main drawback of this methodology is that the choice of these values is problem-dependent and in most situations is carried out heuristically, [51].

We identify that the introduction of the overriding factors is mainly driven by two features. On the one hand, we encounter the problem that the integrals involved in the loss function present different domain dimensionality, i.e., introduce different magnitudes of volume. The integral referring to the residual in the interior of the domain involves a $\hat{d}$-volume, while the integrals associated with the initial and boundary residuals involve a $(\hat{d}-1)$-volume.

An easy solution to solve this situation is to force these lambdas to be inversely proportional to the volume of the each integral's domain considered (as we have shown for the Monte Carlo case). Then, taking into account (14), we rewrite the discrete loss function (11) as

$$\mathcal{J}(\hat{\theta}) = \frac{\hat{\lambda_{\mathcal{I}}}}{|(0,T) \times \overset{\circ}{\Omega}|} \sum_{i=1}^{N_{\mathcal{I}}} w_i^{\mathcal{I}} |\mathcal{R}_\theta^{\mathcal{I}}(t_i^I, x_i^I)|^2 + \frac{\hat{\lambda_{\mathcal{B}}}}{|(0,T) \times \partial\Omega|} \sum_{i=1}^{N_{\mathcal{B}}} w_i^{\mathcal{B}} |\mathcal{R}_\theta^{\mathcal{B}}(t_i^{\mathcal{B}}, x_i^{\mathcal{B}})|^2 + \frac{\hat{\lambda_{\mathcal{O}}}}{|\Omega|} \sum_{i=1}^{N_{\mathcal{O}}} w_i^{\mathcal{O}} |\mathcal{R}_\theta^{\mathcal{O}}(x_i^{\mathcal{O}})|^2. \tag{15}$$

On the other hand, the magnitude of the contributions to the loss function can differ in several orders, i.e., there are addends which are negligible with respect to others, leading to a worse local minima in the training or the need to extend training time. In general, there are two possible situations that can occur simultaneously in a boundary value problem.

One of them is that we can find residuals with large relative losses as the beginning of training. As a consequence, they can cause longer training times because, in the early stages of the optimization procedure, the loss function only provides information regarding such losses. The other possibility is that we can find boundaries in which the residuals exhibit relatively much smaller values, so their contribution to the loss function is, in many cases, negligible. As a result, such constrains could not provide information to the training.

In order to avoid the arbitrary selection of the loss function weights it is essential to reduce the differences in magnitude among residuals. For this reason we propose, for the first time to the best of our knowledge, a novel approach which overcomes this issue. It is based on reformulating, whenever possible, the residuals related to Dirichlet or Neumann (Robin, higher order derivatives) conditions. This reformulation relies on taking as a residual not the boundary condition itself but the resulting PDE restricted to the corresponding boundary. This will produce losses of an order of magnitude similar to that produced by the interior residual, once these quantities are dimensionless.

Thus, most of the Neumann, Robin or higher order derivative boundary residuals we will work with can be written in this form. It suffices to substitute the condition into the PDE of the interior domain and impose the resulting equation on the related boundary residual. However, it will only be possible to impose Dirichlet conditions in this way when they naturally occur at the boundary, i.e., when the Dirichlet condition arises from solving the differential equation that results at the boundary.

As an illustrative example, let us consider a particular case of the parabolic problem defined in (1), where Dirichlet and Neumann boundary conditions are presented. Under the spatial domain $\Omega = \prod_{i=1}^{d} \left[x_i^{min}, x_i^{max}\right]$, the upper boundaries

$$\Gamma_{x_i}^{+} = (x_1^{min}, x_1^{max}) \times \cdots \times \{x_i^{max}\} \times \cdots \times (x_d^{min}, x_d^{max}), \quad i = 1, \ldots, d,$$

and the lower boundaries

$$\Gamma_{x_i}^{0} = (x_1^{min}, x_1^{max}) \times \cdots \times \{x_i^{min}\} \times \cdots \times (x_d^{min}, x_d^{max}), \quad i = 1, \ldots, d,$$

we want to find the parameters of an ANN u_θ in order to make it verify

$$\begin{cases} \dfrac{\partial u_\theta}{\partial t} + \displaystyle\sum_{i,j=1}^{d} a_{ij} \frac{\partial^2 u_\theta}{\partial x_i \partial x_j} + \sum_{i=1}^{d} b_i \frac{\partial u_\theta}{\partial x_i} + f(u_\theta) = 0, & \text{in } (0,T) \times \overset{\circ}{\Omega}, \\ \dfrac{\partial u_\theta}{\partial x_i} - g_i = 0 & \text{in } \Gamma_i^{+} = (0,T) \times \Gamma_{x_i}^{+}, \ \ i = 1, \ldots d, \\ u_\theta - h_i = 0 & \text{in } \Gamma_i^{0} = (0,T) \times \Gamma_{x_i}^{0}, \ \ i = 1, \ldots d, \\ u_\theta - u_0 = 0 & \text{in } \Omega, \end{cases} \tag{16}$$

where $\{a_{ij}\}_{i,j=1}^{d} \subset \mathbb{R}$, $\{b_i\}_{i=1}^{d} \subset \mathbb{R} \setminus \{0\}$, and $g_i \in \mathcal{C}(\Gamma_i^{+}, \mathbb{R})$, $h_i \in \mathcal{C}(\Gamma_i^{0}, \mathbb{R})$, $i = 1, \ldots d$. For example, when defining the residuals associated with the Neumann conditions, the usual approach is to take the condition itself as the residual, i.e.

$$\mathcal{R}_\theta^{\Gamma_i^{+}} = \frac{\partial u_\theta}{\partial x_i} - g_i, \quad i = 1, \ldots d.$$

Alternatively, in our proposal, we plug the Neumann condition into the PDE and impose the resulting equation as a residual, obtaining

$$\mathcal{R}_\theta^{\Gamma_i^{+}} = \frac{\partial u_\theta}{\partial t} + \sum_{i,j=1}^{d} a_{ij} \frac{\partial^2 u_\theta}{\partial x_i \partial x_j} + \sum_{\substack{j=1 \\ j \neq i}}^{d} b_j \frac{\partial u_\theta}{\partial x_j} + b_i g_i + f(u_\theta), \quad i = 1, \ldots, d.$$

For Dirichlet conditions, the proposed strategy can only be applied when h_i verifies the PDE and the initial condition of (16) at the boundary Γ_i^{0}. In such cases we can define the residual in the same way as the residual of the PDE, i.e.,

$$\mathcal{R}_\theta^{\Gamma_i^{0}} = \frac{\partial u_\theta}{\partial t} + \sum_{i,j=1}^{d} a_{ij} \frac{\partial^2 u_\theta}{\partial x_i \partial x_j} + \sum_{i=1}^{d} b_i \frac{\partial u_\theta}{\partial x_i} + f(u_\theta), \quad i = 1, \ldots d.$$

Because of that, such kind of Dirichlet conditions does not even need to be included as boundary residuals. Depending on the quadrature scheme employed, it would be enough to force the existence of interior domain collocation points on such a boundary.

Unfortunately, our proposal cannot be applied to the residual related to the initial condition, so another treatment will be necessary. Some possible alternatives could come from combining our technique with other boundary treatments existing in the literature. For example, the addition of only two hyperparameters that weight the contributions of the initial condition and the remaining residual-related integrals, yielding a discrete loss function of the form

$$\hat{\mathcal{J}}(\theta) = \hat{\lambda_{\mathcal{I}}} \left(\frac{1}{|(0,T) \times \overset{\circ}{\Omega}|} \sum_{i=1}^{N_{\mathcal{I}}} w_i^{\mathcal{I}} |\mathcal{R}_\theta^{\mathcal{I}}(t_i^{I}, x_i^{I})|^2 + \frac{1}{|(0,T) \times \partial\Omega|} \sum_{i=1}^{N_{\mathcal{B}}} w_i^{\mathcal{B}} |\mathcal{R}_\theta^{\mathcal{B}}(t_i^{\mathcal{B}}, x_i^{\mathcal{B}})|^2 \right) + \frac{\hat{\lambda_{\mathcal{O}}}}{|\Omega|} \sum_{i=1}^{N_{\mathcal{O}}} w_i^{\mathcal{O}} |\mathcal{R}_\theta^{\mathcal{O}}(x_i^{\mathcal{O}})|^2,$$

where it is possible to find such weights by means of, for example, biobjective optimization [22]; or avoiding the initial state-loss term by imposing such a condition as a hard constrain [23].

In our case we do not use either of these two proposals in the experiments performed in Section 5 since, in relation to the first approach, the aim of this work is to avoid the inclusion of any optimizable weight in the loss function and, in relation to the second approach, we work with non-differentiable initial conditions. In the one and two-dimensional problems treated later no additional treatment for the initial condition has been necessary but this is insufficient for higher dimension. In such cases, we propose to perform a weakly converged pre-training of the ANN against the initial condition. In addition to achieving the goal of bringing the order of magnitude of the initial condition loss closer to the remaining loss terms, we have found that it has other benefits such as improving and accelerating training by starting from a state closer to the desired one.

Remark 2. As a summary, we have first briefly described the main problems that lead to the introduction of additional weights in the loss function. Then we have proposed a new treatment of the boundary residuals that allows to avoid such weights. When we deal with derivative-based boundary conditions, the related residuals are defined by taking the equation resulting from substituting the boundary conditions in the PDE. For residuals associated with Dirichlet boundary conditions, we can impose the PDE itself as a boundary residual as long as it arises naturally on such boundary.

4. Application to problems in computational finance

In this section we present the PDE formulation of the particular problems we will address in this work. We focus on some relevant (and challenging) state-of-the-art problems appearing in computational finance, specifically, in the area of the CCR assessment. Thus, we consider the valuation of some financial derivatives when accounting for such a risk, namely the pricing of different *risky* European option under the Black–Scholes and Heston models. All of them are extensions of the risk-free derivative pricing models to a formulation that takes into account the effects of bilateral default risk and the funding costs, i.e., which includes CVA, DVA and FVA adjustments, following the approach of [38]. We chose this methodology for its simplicity, but any more complex extension, such as [59], can fit into our framework.

4.1. General pricing problem formulation

Let us consider a derivative contract $\hat{V}$ on $d \geq 1$ spot assets, $S \in \mathbb{R}^d_+$, between two parties, the seller B and its counterparty C, where both may default. We assume that the default of either B or C does not affect S. Such derivative pays the seller B the amount $H(S) \in \mathbb{R}$ at maturity T. In addition, let V the same derivative between two parties that cannot default, i.e., the non-risky derivative value.

Under the described setting, if either the seller or the counterparty defaults, the International Swaps and Derivative Association (ISDA) Master Agreement determines that the value of the derivative is fixed by a Mark-to-Market rule M, which is chosen to be either $\hat{V}$ or V, adjusted by means of $R_B, R_C \in [0, 1]$, the recovery rates on M if seller or counterparty defaults, respectively. Following [38,60] we can define the B and C's default intensities, λ_B and λ_C, by means of the spread between their bond yields, r_B and r_C, and the risk-free interest rate, r. In addition, let s_F be the funding spread, which is $(1 - R_B)\lambda_B$ if the derivative cannot be used as collateral, or 0 otherwise.

From now on, we establish the Mark-to-Market rule $M = \hat{V}$ and that the derivative cannot be used as collateral, so a non-linear PDE model for $\hat{V}$ is obtained. It follows the general definition

$$\begin{cases} \dfrac{\partial \hat{V}}{\partial t} + \mathcal{L}[\hat{V}] + f(\hat{V}) = 0, \\ \hat{V}(0, S) - H(S) = 0, \end{cases} \tag{17}$$

where t is the time to maturity variable, $\mathcal{L}$ the differential elliptic operator defined by the chosen problem and f the non-linear source term given by

$$f(\hat{V}) = \lambda_B(1 - R_B)\min\left\{\hat{V}, 0\right\} + \lambda_C(1 - R_C)\max\left\{\hat{V}, 0\right\} + s_F \max\left\{\hat{V}, 0\right\}. \tag{18}$$

In addition, the derivative value without considering counterparty risk, V, obeys the PDE

$$\begin{cases} \dfrac{\partial V}{\partial t} + \mathcal{L}[V] = 0, \\ V(0, S) - H(S) = 0. \end{cases} \tag{19}$$

4.2. Specific pricing problem formulation

Having defined the general context of the financial problems to be addressed, we are in a position to present the boundary value problems obtained in each specific case, as well as their adaptation to the methodology presented in Section 2.

4.2.1. European options under the Black–Scholes model

We consider an European option driven by $d \in \mathbb{N}$ stock values $S_1, S_2, \ldots, S_d$, with strike $K \in \mathbb{R}$ and maturity $T > 0$. We assume that each asset S_i follows a geometric Brownian motion with drift the repo rate r_{R_i}, and diffusion its volatility σ_i. Further, the correlation between two assets S_i and S_j is given by ρ_{ij} with the constraints $|\rho_{ij}| \leq 1$ and $\rho_{ii} = 1$.

Under such a setting, the option price, $\hat{V}$, is given by the d-dimensional Black–Scholes equation, defined by (17) taking the elliptic operator as

$$\mathcal{L} = -\frac{1}{2}\sum_{i=1}^{d}\sum_{j=1}^{d}\rho_{ij}\sigma_i\sigma_j S_i S_j \frac{\partial^2}{\partial S_i \partial S_j} - \sum_{i=1}^{d} r_{R_i} S_i \frac{\partial}{\partial S_i} + r\mathcal{I}, \tag{20}$$

and the initial condition the payoff agreed between the two parties. In particular, we work with the following contract functions for the multidimensional case:

- the arithmetic average basket function,

$$H(S_1, S_2, \ldots, S_d) = \max\left\{\alpha\left(\frac{1}{d}\sum_{i=1}^{d} S_i - K\right), 0\right\}, \tag{21}$$

- and the worst-of function,

$$H(S_1, S_2, \ldots, S_d) = \max\left\{\alpha\left(\min\left\{S_1, S_2, \ldots, S_d\right\} - K\right), 0\right\}, \tag{22}$$

with $\alpha \in \{-1, 1\}$ for put and call options, respectively. The choice of this contract functions is not accidental as both are common in the industry. Arithmetic average basket options are often traded because they are cheaper than the total of single-asset options on each particular asset, [61], while worst-of options have application in a wide variety of contingent claims, such as option bonds, [62].

The spatial domain of our PDE is a Cartesian product of semi-infinite intervals, $[0, +\infty)^d$. In order to apply the methodology introduced in Section 2 for the numerical resolution of the equation, each interval is truncated, obtaining $\Omega = \prod_{i=1}^{d}[0, S_{i,max}]$. Moreover, additional conditions must be imposed on the domain boundaries $\Gamma_i^0 = (0, T) \times \prod_{j=1}^{i-1}[0, S_{j,max}] \times \{0\} \times \prod_{j=i+1}^{d}[0, S_{j,max}]$ and $\Gamma_i^+ = (0, T) \times \prod_{j=1}^{i-1}[0, S_{j,max}] \times \{S_{i,max}\} \times \prod_{j=i+1}^{d}[0, S_{j,max}]$, $i \in \{1, 2, \ldots, d\}$.

On the one hand, it is possible to impose Dirichlet conditions for the lower boundaries since they arise naturally on them. At each boundary Γ_k^0 we substitute $S_k = 0$ obtaining the boundary operator

$$\frac{\partial \hat{V}}{\partial t} - \frac{1}{2}\sum_{\substack{i=1\\ i\neq k}}^{d}\sum_{\substack{j=1\\ j\neq k}}^{d}\rho_{ij}\sigma_i\sigma_j S_i S_j \frac{\partial^2 \hat{V}}{\partial S_i \partial S_j} - \sum_{\substack{i=1\\ i\neq k}}^{d} r_{R_i} S_i \frac{\partial \hat{V}}{\partial S_i} + r\hat{V} + f(\hat{V}) = 0, \tag{23}$$

which corresponds to the $(d-1)$-dimensional Black–Scholes equation depending on the remaining underlying assets. For the one-dimensional and multidimensional cases where the initial condition does not depend on the other underlyings, (22), we can impose the following Dirichlet condition,

$$\varphi_k(t, S_{k_1}, \ldots, S_{k_{d-1}}) = \frac{|\alpha - 1|}{2} K \exp\left\{-(r + \lambda_B(1 - r_B) + \lambda_C(1 - r_C))\, t\right\}, \tag{24}$$

whereas, with the initial condition (21), we could impose the solution of the $(d-1)$-dimensional arithmetic average basket option, denoted by $\widehat{\mathcal{BS}}_k^{d-1}(t, S_{k_1}, \ldots, S_{k_{d-1}})$, which would need to be simulated numerically when $d > 2$.

On the other hand, the following linearity condition is verified for the considered options,

$$\lim_{S_k \to \infty} \frac{\partial^2 \hat{V}}{\partial S_k^2} = 0, \tag{25}$$

so that such a condition can be imposed on the upper boundaries Γ_k^+, $k \in \{1, 2, \ldots, d\}$, when the truncation value $S_{k,max}$ is large enough. Examples of application can be found in [63] for the risky one-dimensional case, or in [64] for the multidimensional risk-free case. Working with such a condition in the multidimensional risky case is not a problem since the qualitative behaviour of the solution does not change in the limit.

Thus, the set of European options considering CCR described above verifies the following boundary value problem. Find $\hat{V} : [0, T] \times \Omega \subset \mathbb{R}^{d+1} \longrightarrow \mathbb{R}$ such that

$$\begin{cases} \dfrac{\partial \hat{V}}{\partial t} - \dfrac{1}{2}\displaystyle\sum_{i=1}^{d}\sum_{j=1}^{d}\rho_{ij}\sigma_i\sigma_j S_i S_j \frac{\partial^2 \hat{V}}{\partial S_i \partial S_j} - \sum_{i=1}^{d} r_{R_i} S_i \frac{\partial \hat{V}}{\partial S_i} + r\hat{V} + f(\hat{V}) = 0, & \text{in } (0, T) \times \mathring{\Omega}, \\ \hat{V} - h_i = 0, & \text{in } \Gamma_i^0,\ i \in \{1, 2, \ldots, d\}, \\ \dfrac{\partial^2 \hat{V}}{\partial S_i^2} = 0, & \text{in } \Gamma_i^+,\ i \in \{1, 2, \ldots, d\}, \\ \hat{V} - H = 0, & \text{in } \{0\} \times \Omega, \end{cases} \tag{26}$$

where the initial condition H can be (21) or (22), and h_i refers to the Dirichlet condition associated with the chosen payoff, φ_i or $\widehat{\mathcal{BS}}_i^{d-1}$. Moreover, it is straightforward to prove that the European option without considering CCR verifies Eq. (26) by taking $\lambda_{\mathcal{B}} = \lambda_{\mathcal{C}} = 0$ or, equivalently, $f = 0$.

Such formulation fits into problem (1) so we can apply everything explained in Section 2 to solve it. In order to do this, we consider the general training set $\mathcal{P} = \mathcal{P}_{\mathcal{I}} \cup \mathcal{P}_{\mathcal{B}} \cup \mathcal{P}_{\mathcal{O}}$, where $\mathcal{P}_{\mathcal{I}}$ and $\mathcal{P}_{\mathcal{O}}$ admit the definition given above, while the abstract set $\mathcal{P}_{\mathcal{B}}$ must be divided into as many subsets as boundary conditions exist, i.e.,

$$\mathcal{P}_{\mathcal{B}} = \left(\bigcup_{i=1}^{d} \mathcal{P}_{\Gamma_i^0}\right) \bigcup \left(\bigcup_{i=1}^{d} \mathcal{P}_{\Gamma_i^+}\right),$$

where $\mathcal{P}_{\Gamma_i^0}$ and $\mathcal{P}_{\Gamma_i^+}$ represent the set of training points taken on the boundary Γ_i^0 and Γ_i^+, respectively. The mechanism for sampling points will be given by the problem dimensionality. For one and two-dimensional problems, each subset of training points will be a uniform mesh of the subset to which it refers while, in higher dimension, we will use a pseudo-random sampling, again for each subset of training points defined above.

To approximate the desired solution we consider a neural network, $\hat{V}_\theta : [0,T] \times \Omega \subset \mathbb{R}^{d+1} \longrightarrow \mathbb{R}$, with L hidden layers. Without loss of generality, we assume that the number of neurons per hidden layer is the same, β. Based on the boundary value problem given in (26), we choose the network residuals taking into account the proposal given in Section 3 to solve the aforementioned training issues. Since on the boundaries $\Gamma_i^0,\ i = \{1, 2, \ldots, d\}$, the Dirichlet conditions arise naturally, we can use the boundary operator (23) as a residual. This avoid the required numerical simulation of the Black–Scholes prices in $(d-1)$-dimensions. Moreover, on the boundaries $\Gamma_i^+\ i = \{1, 2, \ldots, d\}$, we have a higher-order derivative condition, so we can substitute this condition, (25), into Eq. (17) in order to impose such residual in the same way as we explain in Section 3. Applying these considerations, we obtain the following residuals,

$$\mathcal{R}_\theta^{\mathcal{I}} = \frac{\partial \hat{V}_\theta}{\partial t} - \frac{1}{2}\sum_{i=1}^{d}\sum_{j=1}^{d} \rho_{ij}\sigma_i\sigma_j S_i S_j \frac{\partial^2 \hat{V}_\theta}{\partial S_i \partial S_j} - \sum_{i=1}^{d} r_{R_i} S_i \frac{\partial \hat{V}_\theta}{\partial S_i} + r\hat{V}_\theta + f(\hat{V}_\theta), \qquad \text{in } (0,T)\times \mathring{\Omega}, \tag{27}$$

$$\mathcal{R}_\theta^{\Gamma_k^0} = \frac{\partial \hat{V}_\theta}{\partial t} - \frac{1}{2}\sum_{\substack{i=1\\ i\neq k}}^{d}\sum_{\substack{j=1\\ j\neq k}}^{d} \rho_{ij}\sigma_i\sigma_j S_i S_j \frac{\partial^2 \hat{V}_\theta}{\partial S_i \partial S_j} - \sum_{\substack{i=1\\ i\neq k}}^{d} r_{R_i} S_i \frac{\partial \hat{V}_\theta}{\partial S_i} + r\hat{V}_\theta + f(\hat{V}_\theta), \qquad \text{in } \Gamma_k^0,\ k = 1,\ldots,d, \tag{28}$$

$$\mathcal{R}_\theta^{\Gamma_k^+} = \frac{\partial \hat{V}_\theta}{\partial t} - \frac{1}{2}\sum_{\substack{i=1\\ i\neq k}}^{d}\sum_{j=1}^{d} \rho_{ij}\sigma_i\sigma_j S_i S_j \frac{\partial^2 \hat{V}_\theta}{\partial S_i \partial S_j} - \sum_{i=1}^{d} r_{R_i} S_i \frac{\partial \hat{V}_\theta}{\partial S_i} + r\hat{V}_\theta + f(\hat{V}_\theta), \qquad \text{in } \Gamma_k^+,\ k = 1,\ldots,d, \tag{29}$$

$$\mathcal{R}_\theta^{\mathcal{O}} = \hat{V}_\theta - H, \qquad \text{in } \{0\}\times\Omega. \tag{30}$$

Using the above residuals, the loss function is defined in the same way as in (15) by taking the lambda weights equal to one, i.e.,

$$\begin{aligned}\mathcal{J}(\hat{\theta}) = &\frac{1}{|(0,T)\times\mathring{\Omega}|}\sum_{i=1}^{N_{\mathcal{I}}} w_i^{\mathcal{I}} |\mathcal{R}_\theta^{\mathcal{I}}(y_i^{\mathcal{I}})|^2 + \sum_{k=1}^{d}\frac{1}{|\Gamma_k^0|}\sum_{i=1}^{N_{\Gamma_k^0}} w_i^{\Gamma_k^0} |\mathcal{R}_\theta^{\Gamma_k^0}(y_i^{\Gamma_k^0})|^2 \\ &+ \sum_{k=1}^{d}\frac{1}{|\Gamma_k^+|}\sum_{i=1}^{N_{\Gamma_k^+}} w_i^{\Gamma_k^+} |\mathcal{R}_\theta^{\Gamma_k^+}(y_i^{\Gamma_k^+})|^2 + \frac{1}{|\Omega|}\sum_{i=1}^{N_{\mathcal{O}}} w_i^{\mathcal{O}} |\mathcal{R}_\theta^{\mathcal{O}}(y_i^{\mathcal{O}})|^2,\end{aligned} \tag{31}$$

where $y_i^{\mathcal{X}} \in \mathcal{P}_{\mathcal{X}}$ and $N_{\mathcal{X}} = |\mathcal{P}_{\mathcal{X}}|,\ \mathcal{X} \in \{\mathcal{I}, \mathcal{O}\} \cup \{\Gamma_k^0\}_{k=1}^d \cup \{\Gamma_k^+\}_{k=1}^d$.

In the case of working with a uniformly generated training set, we will use the quadrature weights corresponding to the trapezoidal rule while, in training sets generated by random sampling, we will apply Monte-Carlo integration with the quadrature weights defined in (13).

4.2.2. European options under the Heston model

In order to emphasize the general applicability of our approach, we address the problem of pricing a European option accounting for CCR, with strike $K \in \mathbb{R}$ and maturity $T > 0$, under the assumption that the variance of the underlying follows a stochastic process. Thus, let S be the underlying stock value following a geometric Brownian motion with drift r_R. We define the volatility of S from its variance, ν, which follows a CIR process, [65], with $\eta > 0$ the mean variance, $\kappa > 0$ the mean reversion rate, $\sigma > 0$ the volatility of the variance and $\rho \in [-1, 1]$ the correlation between the asset and variance processes. Under this setting we obtain the Heston model, [66], which is broadly used in the industry.

The PDE problem for pricing the risky European option under the Heston model is derived in [60]. The option price $\hat{V}$ is the solution of Eq. (17) taking the elliptic operator

$$\mathcal{L} = -\frac{S^2\nu}{2}\frac{\partial^2}{\partial S^2} - \rho\sigma S\nu\frac{\partial^2}{\partial S\partial\nu} - \frac{\sigma^2\nu}{2}\frac{\partial^2}{\partial\nu^2} - r_R S\frac{\partial}{\partial S} - \kappa(\eta-\nu)\frac{\partial}{\partial\nu} + r\mathcal{I}, \tag{32}$$

and as an initial condition the payoff (21) with $d = 1$.

As in the previous case, it is necessary to establish an effective domain in order to apply numerical methods. Thus, we define our computational domain as $\Omega = [0, S_{max}] \times [0, \nu_{max}]$ and, again, additional conditions must be imposed over the boundaries $\Gamma_S^0 = (0,T) \times \{0\} \times [0, \nu_{max}]$, $\Gamma_\nu^0 = (0,T) \times (0, S_{max}] \times \{0\}$, $\Gamma_S^+ = (0,T) \times \{S_{max}\} \times (0, \nu_{max})$ and $\Gamma_\nu^+ = (0,T) \times (0, S_{max}] \times \{\nu_{max}\}$.

Following the boundary condition analysis carried out in [60,67], it is not necessary to impose an additional condition on the boundary Γ_S^0. In addition, it will be only necessary to impose a condition on the boundary Γ_ν^0 if the Feller condition, $2\kappa\eta > \sigma^2$, is violated. In such cases, a common choice is to impose a Dirichlet condition obtained from the numerical resolution of the equation

$$\frac{\partial \hat{V}}{\partial t} - r_R S \frac{\partial \hat{V}}{\partial S} - \kappa\eta \frac{\partial \hat{V}}{\partial \nu} + r\hat{V} + f(\hat{V}) = 0, \qquad \text{in } \Gamma_\nu^0. \tag{33}$$

On the boundary Γ_S^+ we keep the linearity condition (25), while on the boundary Γ_ν^+ we choose to employ the Neumann condition derived from the fact that

$$\lim_{\nu \to \infty} \frac{\partial \hat{V}}{\partial \nu}(t, S, \nu) = 0. \tag{34}$$

We are in position to present the boundary value problem for pricing the risky European option under the Heston model. Therefore, we must find $\hat{V} : [0,T] \times \Omega \subset \mathbb{R}^3 \longrightarrow \mathbb{R}$ such that

$$\begin{cases} \dfrac{\partial \hat{V}}{\partial t} - \dfrac{S^2\nu}{2}\dfrac{\partial^2 \hat{V}}{\partial S^2} - \rho\sigma S\nu \dfrac{\partial^2 \hat{V}}{\partial S \partial \nu} - \dfrac{\sigma^2 \nu}{2}\dfrac{\partial^2 \hat{V}}{\partial \nu^2} - r_R S \dfrac{\partial \hat{V}}{\partial S} - \kappa(\eta - \nu)\dfrac{\partial \hat{V}}{\partial \nu} + r\hat{V} + f(\hat{V}) = 0, & \text{in } (0,T) \times \overset{\circ}{\Omega}, \\ \dfrac{\partial^2 \hat{V}}{\partial S^2} = 0, & \text{in } \Gamma_S^+, \\ \dfrac{\partial \hat{V}}{\partial \nu} = 0, & \text{in } \Gamma_\nu^+, \\ \hat{V} - \max\left\{\alpha\,(S - K)\,, 0\right\} = 0, & \text{in } \{0\} \times \Omega, \end{cases} \tag{35}$$

when the Feller condition is satisfied. Again, the risk-free Heston boundary problem is recovered by taking the risk parameters $\lambda_B = \lambda_C = 0$; and such formulation fits, again, into the problem (1), so the techniques in Section 2 can be readily applied.

At the methodological level, the development of this problem is similar to the d-dimensional Black–Scholes boundary value problem when $d = 2$. Thus, the set of training points $\mathcal{P}$ will be generated from a uniform sampling of each of its subsets. We define the residuals used in the training of a neural network $\hat{V}_\theta : [0,T] \times \Omega \subset \mathbb{R}^3 \longrightarrow \mathbb{R}$ in the task of approximating the solution of (35) as

$$\begin{aligned} \mathcal{R}_\theta^{\mathcal{I}} &= \frac{\partial \hat{V}_\theta}{\partial t} - \frac{S^2\nu}{2}\frac{\partial^2 \hat{V}_\theta}{\partial S^2} - \rho\sigma S\nu \frac{\partial^2 \hat{V}_\theta}{\partial S \partial \nu} \\ &\quad - \frac{\sigma^2 \nu}{2}\frac{\partial^2 \hat{V}_\theta}{\partial \nu^2} - r_R S \frac{\partial \hat{V}_\theta}{\partial S} - \kappa(\eta - \nu)\frac{\partial \hat{V}_\theta}{\partial \nu} + r\hat{V}_\theta + f(\hat{V}_\theta), && \text{in } (0,T) \times \overset{\circ}{\Omega}, \quad (36) \\ \mathcal{R}_\theta^{\Gamma_S^+} &= \frac{\partial \hat{V}_\theta}{\partial t} - \rho\sigma S\nu \frac{\partial^2 \hat{V}_\theta}{\partial S \partial \nu} - \frac{\sigma^2 \nu}{2}\frac{\partial^2 \hat{V}_\theta}{\partial \nu^2} - r_R S \frac{\partial \hat{V}_\theta}{\partial S} - \kappa(\eta - \nu)\frac{\partial \hat{V}_\theta}{\partial \nu} + r\hat{V}_\theta + f(\hat{V}_\theta), && \text{in } \Gamma_\nu^+, \quad (37) \\ \mathcal{R}_\theta^{\Gamma_S^0} &= \frac{\partial \hat{V}_\theta}{\partial t} - \frac{\sigma^2 \nu}{2}\frac{\partial^2 \hat{V}_\theta}{\partial \nu^2} - \kappa(\eta - \nu)\frac{\partial \hat{V}_\theta}{\partial \nu} + r\hat{V}_\theta + f(\hat{V}_\theta), && \text{in } \Gamma_S^0, \quad (38) \\ \mathcal{R}_\theta^{\Gamma_\nu^0} &= \frac{\partial \hat{V}_\theta}{\partial t} - r_R S \frac{\partial \hat{V}_\theta}{\partial S} - \kappa\eta \frac{\partial \hat{V}_\theta}{\partial \nu} + r\hat{V}_\theta + f(\hat{V}_\theta), && \text{in } \Gamma_\nu^0, \quad (39) \\ \mathcal{R}_\theta^{\mathcal{O}} &= \hat{V}_\theta - \max\left\{\alpha\,(S - K)\,, 0\right\}, && \text{in } \{0\} \times \Omega. \quad (40) \end{aligned}$$

In this case, we decide to include the boundary-related residuals (38) and (39) as if they were boundary conditions, but they could be also considered as part of the interior of the domain straightforwardly. Then, $\hat{V}_\theta$ is trained by means of a loss function like the one presented in (15), taking the lambda weights equal to one and the quadrature weights corresponding to the trapezoidal rule.

5. Numerical experiments

After presenting the mathematical models and discussing how they fit under our reformulation via PINNs, in this section we show the results of the tests performed to assess their effectiveness. One of the main advantages of this methodology over traditional numerical methods is that the container of the approximate solution is an ANN, i.e., a differentiable function. Thus, it is possible to compute its derivatives via AD. In this regard, for the low dimensional cases

Table 1
Parameters for the 2-dimensional Black–Scholes model considering counterparty risk.

Black–Scholes parameters	
Domain, Ω	$[0, 200] \times [0, 200]$
Strike, K	50
Time to maturity, T	1
Interest rate, r	0.03
Volatilities, (σ_1, σ_2)	$(0.25, 0.15)$
Repo rates, (r_{R_1}, r_{R_2})	$(0.015, 0.022)$
Correlation, ρ	-0.65
xVA parameters	
Seller hazard rate, λ_B	$[0.0, 0.1]$
Counterparty hazard rate, λ_C	0.07
Seller recovery rate, R_B	0.5
Counterparty recovery rate, R_C	0.3
Funding spread, s_F	$(1 - R_B)\lambda_B$

we will focus not only on how well it approximates the desired solution, but also on how accurately it approximates its derivatives.

The section is divided into two parts. In the first one, we solve two-dimensional parabolic problem under the Black–Scholes and Heston framework. The second one is devoted to solving high-dimensional parabolic PDEs under the Black–Scholes model. The same pattern is followed in both. We discuss the network configuration chosen and we analyse the predictions provided by the trained networks in the proposed problems. For this purpose, we compare them with extremely reliable approximations of the desired solutions computed by either FD ($d = 2$ cases) or Monte Carlo ($d > 2$ cases). In addition, the same experiments for the one-dimensional case are presented in Appendix, where exact analytical solutions to validate the numerical methods are available. Our codes are implemented based on Tensorflow and all the models are trained on one NVIDIA Ampere A100 GPU with 82 GB memory.

5.1. Two-asset basket options under the Black–Scholes model

In this section we present the results obtained when approximating solutions of the two-dimensional parabolic problem (26) with initial conditions (21) and (22), i.e., when approximating the price of arithmetic average and worst-of two-asset basket options under the Black–Scholes framework. The model parameters are presented in Table 1.

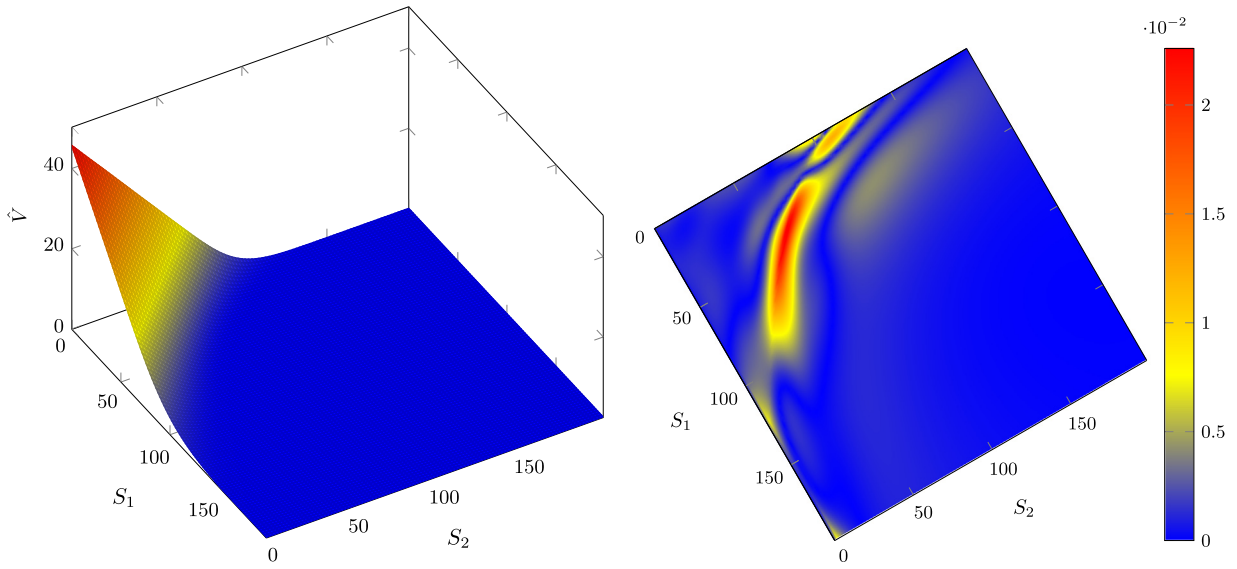
5.1.1. Test settings

The neural network to be trained follows the description given in Section 2.1. The number of layers, L, and the units per layer, β, is discussed below. The training loss function is given by (31) with $d = 2$. The integrals appearing in this expression are computed with the trapezoidal rule, thus the set of training points, $\mathcal{P} = \mathcal{P}_{\mathcal{I}} \cup \mathcal{P}_{\mathcal{O}} \cup (\cup_{k=1}^{2} \mathcal{P}_{\Gamma_k^0}) \cup (\cup_{k=1}^{2} \mathcal{P}_{\Gamma_k^+})$, comes from the uniform discretization of the interior domain and its boundaries. The experiments presented below are run with $|\mathcal{P}_{\mathcal{I}}| = 128\,000$, $|\mathcal{P}_{\mathcal{O}}| = 6724$ and $|\mathcal{P}_{\Gamma_k^0}| = |\mathcal{P}_{\Gamma_k^+}| = 1620$, $k \in \{1, 2\}$, yielding a total of 141 204 training points. The optimization involves 22 500 iterations. The first 20 000 are performed with Adam taking the learning rate as 10^{-3}, adding the inverse time decay schedule (12) with decay rate $\delta = 0.75$ and decay step $a = 5000$. The remaining ones are performed with L-BFGS. The accuracy of the network prediction is measured by comparing its relative error with an approximate solution obtained via FD (Crank–Nicolson timestepping and centre differences) with a fixed point scheme to deal with the non-linearity [63,68], ensuring a maximum error of 10^{-6}.

5.1.2. Numerical results

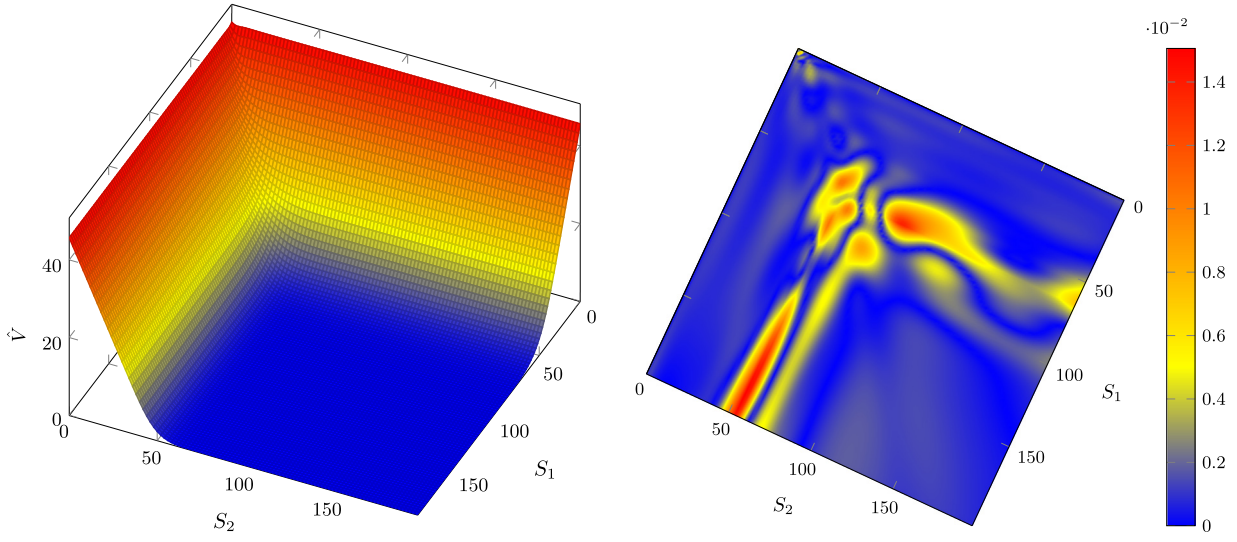
First of all, we are interested in finding an optimal combination of layers and neurons per layer in terms of accuracy and training time required. Thus, we consider all 16 possible combinations between $L \in \{2, 4, 8, 12\}$ layers and $\beta \in \{10, 20, 40, 60\}$ units per layer. A sample of 10 training trials is considered per combination and the worst of them is chosen. The pricing of a non-risky arithmetic average put option, V, is the target. We have observed that the worst performing combinations are those with larger number of layers, a situation probably related to problems in updating the network's weights due to the combination of very deep networks and bounded activation functions, [58]. For the remaining combinations, the L^2-norm relative errors are of the order of 10^{-3}, similar to those obtained for the same task in [43], where the tuning of the lambda weights is performed and the Monte Carlo integration is employed as a quadrature rule. With respect to the training time, it increases polynomially in relation to the number of units per layer and exponentially in relation to the number of layers. Taking into account the training time required for each case, we choose the combination of $L = 4$ layers and $\beta = 60$ units per layer. Under this setting, relative errors of 3.71×10^{-4}, 4.72×10^{-4} and 1.12×10^{-3} for the $L1$, L^2 and L^∞-norm are achieved.

In order to evaluate the performance of our training algorithm for the risky non-linear case, we consider different default scenarios (varying the seller hazard rate, λ_B) for the pricing of the two proposed products.



(a) Put option value surface computed via trained ANN. (b) Relative error colour map.

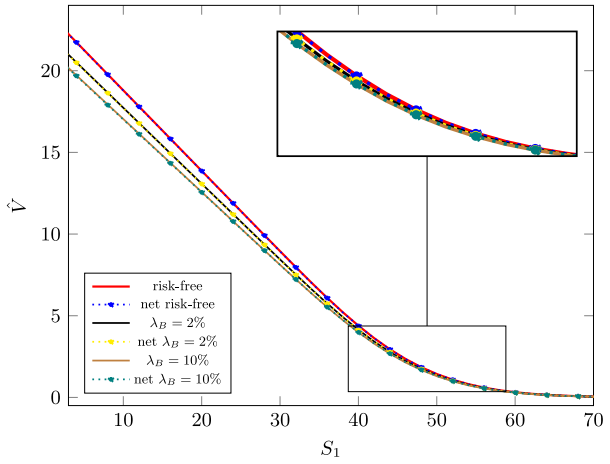
Fig. 1. Risky arithmetic average put option with parameters given in Table 1, $\lambda_B = 2\%$.



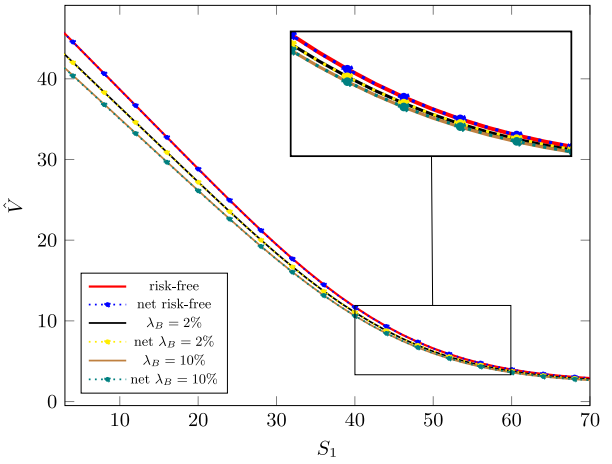
(a) Put option value surface computed via trained ANN. (b) Relative error colour map.

Fig. 2. Risky worst-of put option with parameters given in Table 1, $\lambda_B = 2\%$.

In Figs. 1(a) and 2(a) the PINN approximations for the risky arithmetic average and worst-of put options, with $\lambda_B = 2\%$, are plotted; while Figs. 1(b) and 2(b) show the error compared to the reference solutions. As expected, the worst relative errors are obtained when the value of the option tends to zero since they are below the number of significant digits we expect to achieve. In order to avoid such a behaviour and thus obtain an adequate visualization of the error in the area of interest, we only consider the relative error for option values greater than or equal to 0.01. Regions with smaller option values are treated in terms of the absolute error, scaled by the imposed threshold. This is sufficient for quantitative finance purposes and it is also followed in the error plots for the following cases.

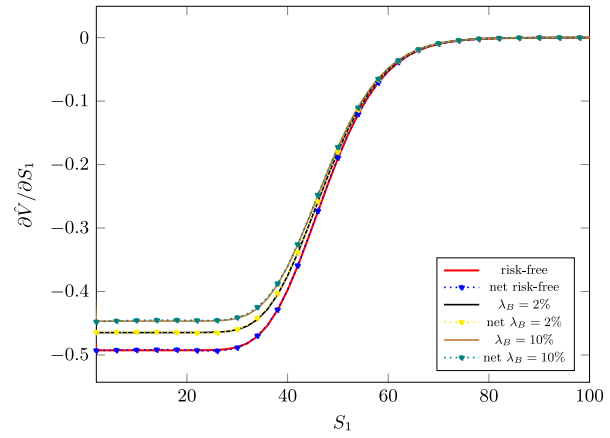


(a) Arithmetic average put option price.

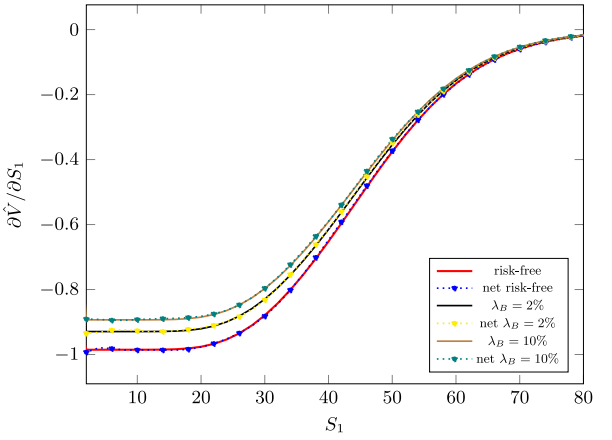


(b) Worst-of put option price.

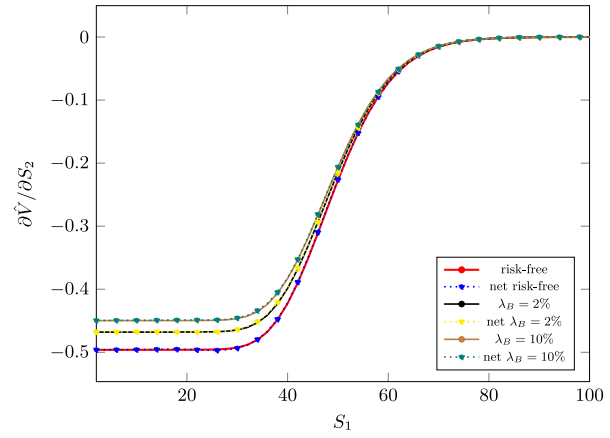
Fig. 3. Comparison between finite differences and trained networks option values with $S_2 = 50$ fixed. Risk-free case ($\lambda_B = \lambda_C = 0$), case $\lambda_B = 2\%$ and case $\lambda_B = 10\%$ are plotted.



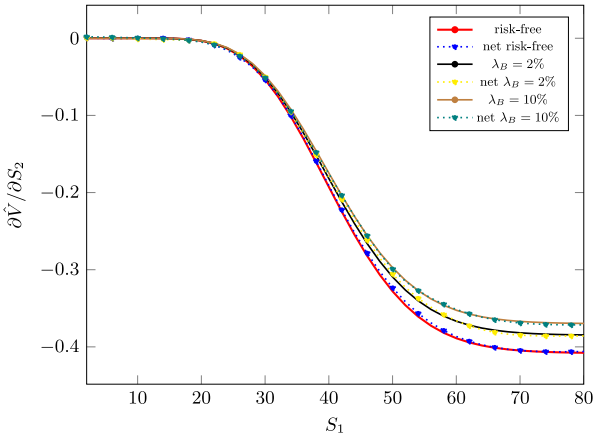
(a) Arithmetic average put option delta with respect to the $S1$.



(b) Worst-of put option delta with respect to the $S1$.



(c) Arithmetic average put option delta with respect to the $S2$.



(d) Worst-of put option delta with respect to the $S2$.

Fig. 4. Comparison between finite differences and trained networks option deltas with $S_2 = 50$ fixed. Risk-free case ($\lambda_B = \lambda_C = 0$), case $\lambda_B = 2\%$ and case $\lambda_B = 10\%$ are plotted.

Table 2
Relative errors for the arithmetic average and worst-of option prices and deltas. The set of parameters given in Table 1 is considered, taking $\lambda_B = \lambda_C = 0$ in the risk-free case. For each scenario, an ATM, a OTM and a ITM values are taken.

Arithmetic average				
Case	(S_1, S_2)	$\hat{V}$	$\partial\hat{V}/\partial S_1$	$\partial\hat{V}/\partial S_2$
Risk-free	(50.0, 50.0)	1.19×10^{-3}	4.17×10^{-3}	4.50×10^{-3}
	(55.0, 50.0)	4.33×10^{-5}	1.71×10^{-4}	1.09×10^{-4}
	(50.0, 45.0)	3.39×10^{-3}	3.49×10^{-3}	3.99×10^{-3}
$\lambda_B = 2\%$	(50.0, 50.0)	4.98×10^{-4}	3.04×10^{-3}	4.31×10^{-3}
	(55.0, 50.0)	3.08×10^{-3}	9.87×10^{-4}	4.46×10^{-3}
	(50.0, 45.0)	7.52×10^{-4}	3.05×10^{-3}	3.45×10^{-3}
Worst-of				
Case	(S_1, S_2)	$\hat{V}$	$\partial\hat{V}/\partial S_1$	$\partial\hat{V}/\partial S_2$
Risk-free	(50.0, 50.0)	1.34×10^{-3}	5.87×10^{-3}	1.41×10^{-2}
	(60.0, 53.0)	4.66×10^{-3}	1.12×10^{-2}	1.06×10^{-3}
	(57.1, 42.9)	7.42×10^{-4}	3.59×10^{-3}	1.45×10^{-3}
$\lambda_B = 2\%$	(50.0, 50.0)	3.52×10^{-3}	1.37×10^{-2}	1.24×10^{-2}
	(60.0, 53.0)	9.18×10^{-3}	2.30×10^{-2}	5.98×10^{-3}
	(57.1, 42.9)	2.78×10^{-4}	2.09×10^{-3}	9.25×10^{-3}

As expected, the largest errors are observed at-the-money[1] (ATM) levels and its neighbourhood, as is the case in classical schemes. This is because the initial conditions we are working with, (21) and (22), are non-differentiable in such subdomains. Even so, reasonable errors are obtained, being at most of the order of 10^{-2}.

Fig. 3 shows a comparison between reference and network approximated prices in slice $S_2 = K$ for the two proposed derivatives. The solutions approximated by the trained ANN have an identical qualitative behaviour in the plotted cases and there are no significant differences when working with different λ_B values.

Furthermore, the first order derivatives with respect to the underlying assets, known as deltas in the financial industry, are computed by means of AD of the trained ANNs. Fig. 4 shows a comparison between reference and network deltas in slice $S_2 = K$ for the two proposed derivatives. Both derivative approximations suffer from some slight oscillations near the lower boundaries (behaviour also observed in the one-dimensional case, see Appendix), but such oscillations are not observed in the rest of the domain. In addition, the predictions related to the arithmetic average case keep errors of a similar order of magnitude to those incurred in estimating prices. In the worst-of case, the quality of the delta approximation is influenced by the direction it follows in relation to the ATM region. Thus, better approximations are obtained when they follow the downward direction, while the predictions deteriorate in transverse direction. The relative errors obtained for the approximation of both prices and deltas in the neighbourhood of the ATM region are shown in Table 2. Only the risk-free and the $\lambda_B = 2\%$ risky cases are shown since the remaining scenarios present a similar behaviour.

5.2. Vanilla option pricing under the Heston model

We present the results related to the valuation of options using the Heston model, which is based on the description given in Section 4.2.2. For this purpose, we work with the model data given in Table 3.

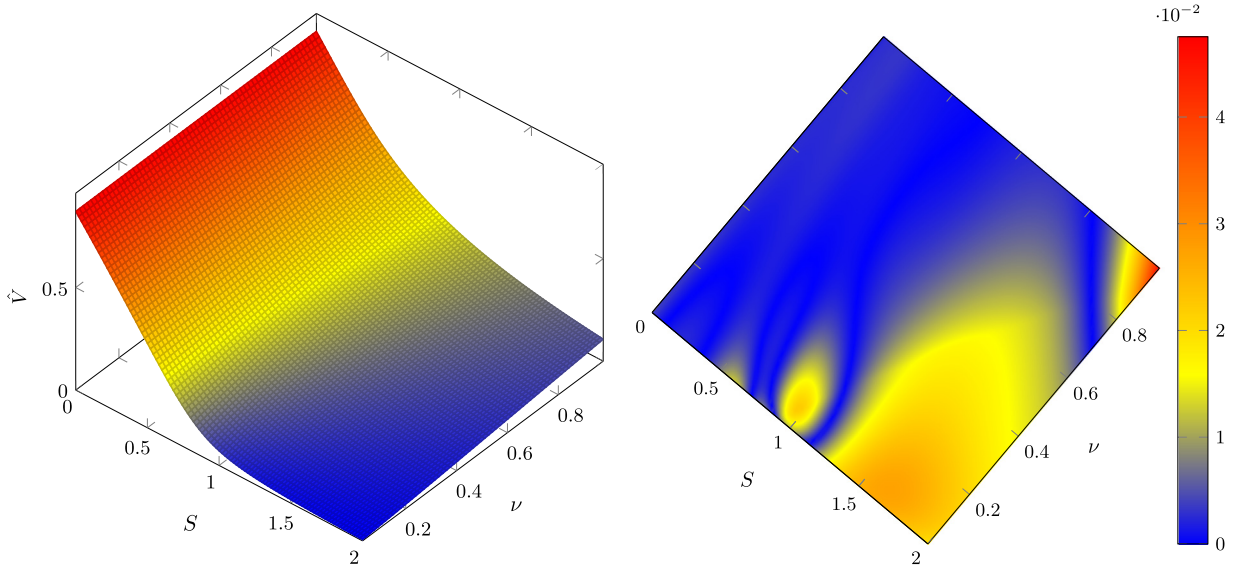
5.2.1. Test settings

The same architecture used to produce the outcomes in the previous section is kept, i.e., an ANN with $L = 4$ layers and $\beta = 60$ units per layer. The loss function is similar to that provided for the two-dimensional Black–Scholes case, but using the residuals (36)–(40). Again, the integrals that compose this loss function are computed with the trapezoidal rule. The set of training points, $\mathcal{P} = \mathcal{P}_{\mathcal{I}} \cup \mathcal{P}_{\mathcal{O}} \cup \mathcal{P}_{\Gamma_S^0} \cup \mathcal{P}_{\Gamma_v^0} \cup \mathcal{P}_{\Gamma_S^+} \cup \mathcal{P}_{\Gamma_v^+}$, comes from the uniform discretization of the interior domain and its boundaries. The experiments presented below are run with $|\mathcal{P}_{\mathcal{I}}| = 304\,200$, $|\mathcal{P}_{\mathcal{O}}| = 6400$, $|\mathcal{P}_{\Gamma_S^0}| = |\mathcal{P}_{\Gamma_v^0}| = |\mathcal{P}_{\Gamma_S^+}| = |\mathcal{P}_{\Gamma_v^+}| = 4000$, yielding a total of 334 600 training points. The optimization procedure is kept with the exception of the Adam parameters. We establish 25 000 Adam iterations with learning rate 10^{-3}, decay rate $\delta = 0.5$ and decay steps $a = 10\,000$. The accuracy of the network prediction is measured by comparing its relative error with an approximate solution obtained with the aforementioned fixed point scheme, ensuring a minimum precision of 10^{-4}.

[1] The at-the-money region is the subset of the domain where the option's strike price is identical to the price given by the combination of the underlyings which defines the derivative contract. For example, the at-the-money region for the arithmetic average basket option is $\{(S_1, S_2) \in \Omega : S_1 + S_2 - 2K = 0\}$. In this way, the out-the-money region is the domain's subset where the call (put) option's strike price is larger (smaller) than the price which defines the derivative contract, and the in-the-money region is its opposite.

Table 3
Parameters for the Heston model, adapted from [69], and risky parameters.

Heston parameters	
Domain, Ω	$[0, 4] \times [0, 3]$
Strike, K	1
Time to maturity, T	2
Repo rate, r_R	0.025
Interest rate, r	0.025
Mean reversion rate, κ	1.5
Mean variance, η	0.04
Volatility of variance, σ	0.3
Correlation, ρ	-0.9
xVA parameters	
Seller hazard rate, λ_B	$[0.0, 0.1]$
Counterparty hazard rate, λ_C	0.04
Seller recovery rate, R_B	0.3
Counterparty recovery rate, R_C	0.3
Funding spread, s_F	$(1 - R_B)\lambda_B$



(a) Put option value surface computed via ANN.

(b) Relative error colour map.

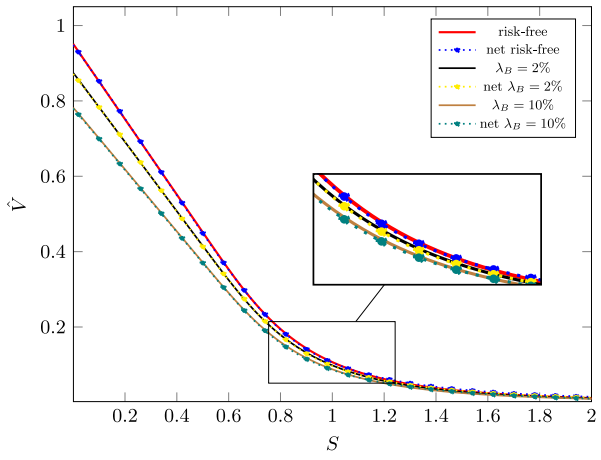
Fig. 5. Risky put option under the Heston model with parameters given in Table 3, $\lambda_B = 2\%$.

5.2.2. Numerical results

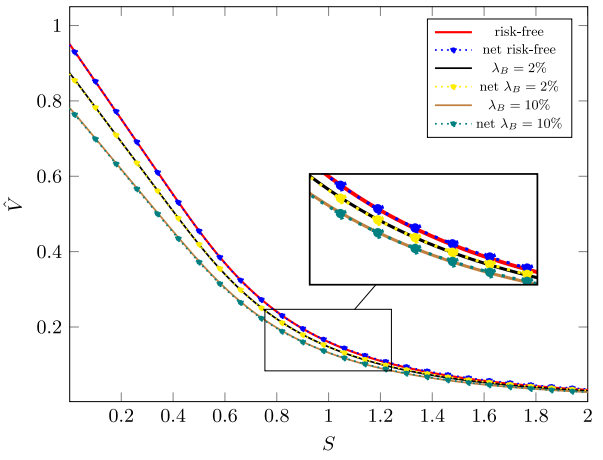
Fig. 5 shows, for the risky case $\lambda_B = 2\%$, the pricing surface computed by the trained ANN (Fig. 5(a)), as well as the errors obtained in relation with respect to the reference solution (Fig. 5(b)). Both the qualitative and quantitative behaviour of the solution achieve the precision standards of the other two-dimensional cases studied above. However, a different distribution of the committed error is observed. In previous cases, the error is concentrated in the ATM region, mostly due to the non-differentiability of the payoff. Now, although we see the expected larger error in the ATM region when the values of ν are close to zero, it becomes dominant in the OTM region. Such error pattern has also been found in the FD algorithms employed to compute the reference solution. This fact suggests that the chosen boundary conditions due to the truncation domain could be hampering the accuracy of the approximation.

In Fig. 6, ν-slices of the solution and its first order derivatives are shown. Such slices correspond to sections with $\nu = 0.1$ and $\nu = 0.3$ (values of interest in the industry). The risk-free case and the cases with $\lambda_B = 2\%$, 10% are considered. Regarding the pricing approximations, we observe a similar qualitative performance to that seen in the previous examples, particularly as we move away from ν values close to 0. For the first-order derivatives a slightly decrease in the performance is found due to the more complex physics described by the PDE, although the obtained accuracy is sufficient for financial purposes. Table 4 provides the errors obtained for different default scenarios considering a standard value for ν.

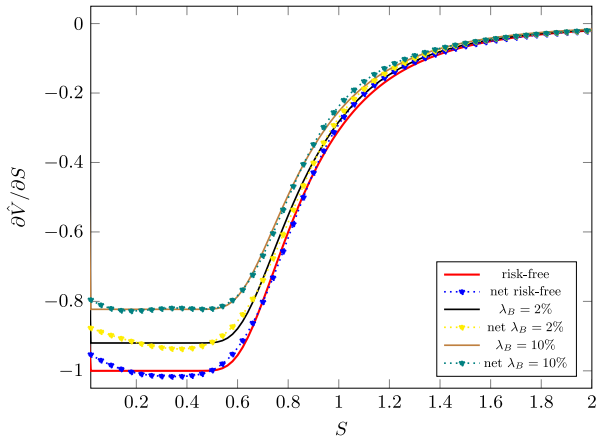
In the case of deltas, Figs. 6(c) and 6(d), the oscillatory behaviour near the $S = 0$ boundary seen before is slightly magnified, specially for the risk-free and lower λ_B scenarios. However, it is able to perfectly capture its asymptotic behaviour as S grows. The approximations around the strike are remarkably good, with relative errors and the order of



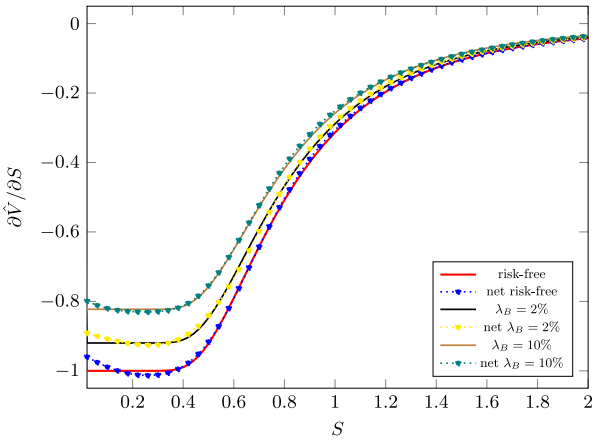
(a) Option prices with $\nu = 0.1$.



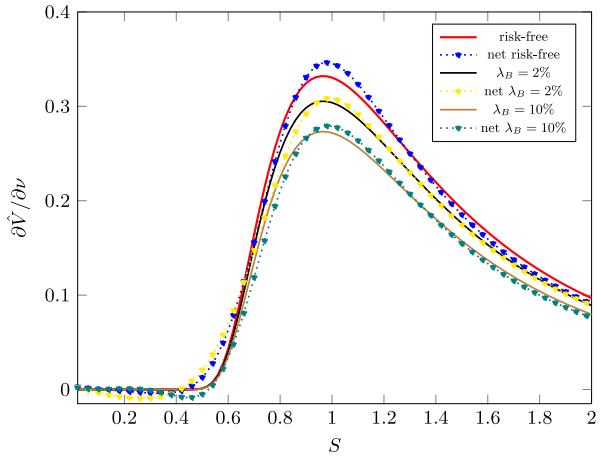
(b) Option prices with $\nu = 0.3$.



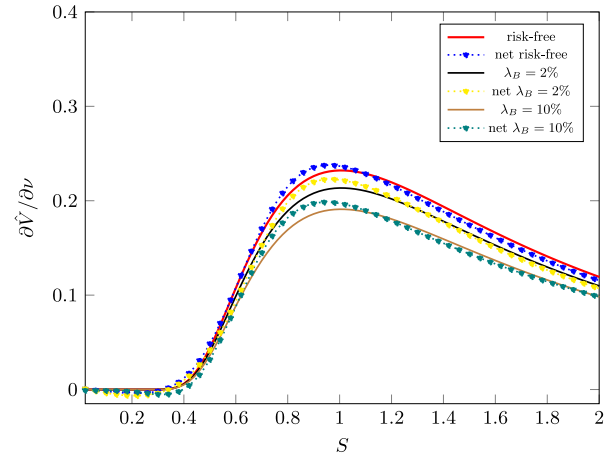
(c) Option deltas with $\nu = 0.1$.



(d) Option deltas with $\nu = 0.3$.



(e) Option vegas with $\nu = 0.1$.



(f) Option vegas with $\nu = 0.3$.

Fig. 6. Comparison between finite differences and network put option prices, deltas and vegas under the Heston model. Risk-free case ($\lambda_B = \lambda_C = 0$), case $\lambda_B = 2\%$ and case $\lambda_B = 10\%$ are plotted.

Table 4
Relative errors for the put option price and first order derivatives, under the Heston model and taking $\nu = 0.1$. The set of parameters given in Table 3 is considered, taking $\lambda_B = \lambda_C = 0$ in the risk-free case, and varying λ_B in the others. One ITM, one ATM and one OTM case are presented for each default scenario.

Case	S	$\hat{V}$	$\partial\hat{V}/\partial S$	$\partial\hat{V}/\partial\nu$
Risk-free	0.8	7.82×10^{-3}	2.76×10^{-2}	4.46×10^{-2}
	1.0	1.18×10^{-2}	5.62×10^{-2}	4.58×10^{-2}
	1.2	3.33×10^{-2}	7.48×10^{-2}	1.80×10^{-2}
$\lambda_B = 2\%$	0.8	5.95×10^{-3}	3.65×10^{-2}	7.60×10^{-2}
	1.0	2.02×10^{-2}	4.03×10^{-2}	1.43×10^{-2}
	1.2	1.11×10^{-2}	7.40×10^{-2}	5.87×10^{-3}
$\lambda_B = 10\%$	0.8	1.81×10^{-2}	1.72×10^{-2}	6.38×10^{-2}
	1.0	2.05×10^{-2}	5.99×10^{-2}	2.59×10^{-2}
	1.2	1.87×10^{-2}	7.35×10^{-2}	1.85×10^{-2}

Table 5
Parameters for the d-dimensional Black–Scholes model. It is assumed that all the underlying assets have the same volatility and drift. In addition, the correlation between any two assets is the same.

Black Scholes parameters	
Domain, Ω	$[0, 200]^d$
Strike, K	50
Time to maturity, T	1
Volatility, σ_i	0.25
Repo rate, r_{R_i}	0.03
Interest rate, r	0.03
Correlation, ρ_{ij}	0.65

$10^{-2}/10^{-3}$. Figs. 6(e) and 6(f) show the vega slices, understanding vega as the derivative of the price with respect to the underlying's variance.[2] Regardless of the chosen default scenario, the approximations close to $S = 0$ are worse. Moreover, the estimations are affected by the closeness to the boundary $\nu = 0$, so that the closer you are to such boundary, the lower the accuracy is. However, this effect looses intensity or directly disappears for larger values of S. Thus, the same order as that obtained for the deltas is observed in the neighbourhood of the strike (the area of interest) and the asymptotic behaviour is consistent with the Neumann condition imposed on the ν_{max} boundary.

5.3. Multidimensional basket options under the Black–Scholes model

In this last section we test the performance of the ANNs trained by means of the PINNs methodology in the task of approximating the solutions of 5/10-dimensional[3] parabolic PDEs. We focus on providing results for the risk-free problem (26) with the initial conditions (21) and (22) in order to show that the boundary treatment proposed in Section 3 is readily applicable to high dimensional problems. The model parameters are presented in Table 5.

5.3.1. Test settings

As in the previous cases, the network is configured as set out in Section 2.1. In view of the effects observed in the previous experiments, we are now only concern with finding a suitable number of units per layer, β, keeping the number of layers employed before, $L = 4$. We perform a weakly converged pre-training of the network against the initial condition in order to exploit the benefits discussed in Section 3. The training loss function is given by (31) with $d = 5/10$. Now, the integrals that we need to calculate in each training iteration are computed by means of the Monte Carlo integration, so that each residual-related integral approximation is given by the mean squared error function, see Remark 1. The training set is now re-sampled every given number of iterations (mini-batch sampling), which will depend on the optimizer employed. Thus, in each iteration with Adam a sample of 6144/32 768 random points will be taken from the domain, where 2048/16 384 belong to the interior domain, 2048/8192 to the initial condition and 2048/8192 to the set of all boundaries (chosen randomly between them). Every 200 L-BFGS iterations a sample of 24 576/49 152 random points will be selected to constitute the training set, i.e., 8192/16 384 from each subset described in Adam's case. The number of iterations with Adam is specified later for each specific case, as well as the initial learning rate and the decay rate. The maximum number of iterations with L-BFGS is set at 15 000. Other training considerations are discussed below. The accuracy of the network prediction is measured by comparing its relative error with an approximate solution obtained via Monte Carlo simulation with low discrepancy sequences, ensuring at least a maximum error of 10^{-6}.

[2] We assume an abuse of language. In reality, vega is understood as the partial derivative with respect to the square root of the variance but, considering fixed-ν slices, both expressions only differ in being multiplied by a constant.

[3] For the sake of brevity, whenever the slash notation is used, the value preceding it refers to the 5-dimensional case, while the remaining value refers to the 10-dimensional case.

Table 6
Relative errors for the 5/10-dimensional arithmetic average and worst of put options. One ATM, two ITM and two OTM cases are presented taking $S_i = 50,\ i = 3, \ldots, 5/10$. In addition, the average relative error achieved for 1000/10000 random values of the 5/10 assets between 40 and 60 is added.

(S_1, S_2)	Arithmetic average		Worst-of	
	$d = 5$	$d = 10$	$d = 5$	$d = 10$
(50, 50)	5.02×10^{-3}	1.07×10^{-2}	1.47×10^{-2}	2.96×10^{-2}
(40, 50)	3.27×10^{-3}	4.93×10^{-4}	8.25×10^{-3}	4.08×10^{-3}
(50, 40)	3.15×10^{-3}	1.14×10^{-2}	6.19×10^{-3}	3.58×10^{-3}
(60, 50)	6.96×10^{-3}	2.10×10^{-2}	1.17×10^{-2}	1.66×10^{-2}
(50, 60)	6.97×10^{-3}	9.29×10^{-3}	7.51×10^{-3}	2.26×10^{-2}
Mean from $[40, 60]^d$	5.08×10^{-3}	1.15×10^{-2}	7.26×10^{-3}	1.92×10^{-2}

5.3.2. Numerical results

In these experiments, a different methodology has been followed to establish the number of units per layer since the choice of optimization parameters becomes more important than in previous cases. On the one hand, pre-training the ANN against the initial condition allows us to start from a fixed configuration of network weights, removing the random initialization of such parameters. To perform this step it is only necessary that the network has sufficient capacity to adequately approximate the initial condition (in the whole domain). On the other hand, such randomness is now incorporated by the use of mini-batch sampling since each batch of training points is chosen in a pseudo-random way. It is important that the ANN has sufficient degrees of freedom to ensure that the losses of each batch provide information for updating the network weights. Under these factors we work with $L = 128/256$ units per layer, which provides a suitable pre-training, a sufficient degree of freedom for updating the network parameters and a relatively moderate training time. Other choices may work but it is probably that the given optimization parameters have to be modified.

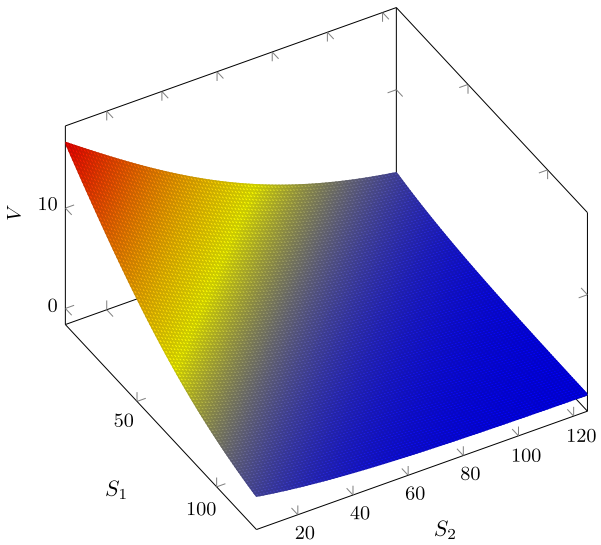
In the arithmetic average 5-asset basket option case we have employed 30 000 Adam iterations with a linear decay of the learning rate from 5×10^{-3} to 5×10^{-5}. Together with the corresponding L-BFGS iterations we have achieved an average relative error in the ATM neighbourhood between 8.87×10^{-3} and 5.08×10^{-3} for all training converged. The training measures have not been shown to be as robust as for the two-dimensional case when applying L-BFGS. There have been training samples that, once converged with Adam (we are talking about relative errors between 3.74×10^{-2} and 1.01×10^{-2}), L-BFGS has ruined them. Most of these cases share a common pattern, the loss related to the initial condition fall significantly below the total observed loss during the L-BFGS procedure. It is likely that the cause of this pathology is the mixing of L-BFGS with mini-batch sampling, in the sense that some training sample may drive the optimizer to a bad local minima. For the 10-dimensional related option we have kept the number of Adam iterations used before, imposing a linear decay of the learning rate from 1×10^{-3} to 5×10^{-5} and dispensing with the use of L-BFGS as it has not improved Adam's metrics in any sample. Under this setting, a stable training is achieved, with average errors between 2.52×10^{-2} and 1.15×10^{-2}.

For the worst-of options it has been necessary to extend the number of Adam iteration to 10 000 while maintaining a linear decay of the learning rate with the initial and final values given for the arithmetic case with the same dimension. The average relative errors measured in the neighbourhood of the ATM region ranges between $2.19 \times 10^{-2}/2.83 \times 10^{-2}$ and $7.26 \times 10^{-3}/1.92 \times 10^{-2}$ for the 5/10-dimensional case. In contrast to the previous product, the L-BFGS application is now stable and improves predictions by an order of magnitude. The error patterns for the four products presented in this section are similar to that observed in the 2-dimensional examples. There is a deterioration of the approximation as we move further into the OTM region, which is expected as the values are closer to zero; and the worst-of option shows more difficulties in the ATM region than the arithmetic option. Table 6 shows the relative errors obtained for specific points, as well as the average errors given above. In addition, surface slices of the 5/10-dimensional option value predictions are plotted in Fig. 7. In general, the results are rather satisfactory and sufficient for the practical applications within the financial industry.

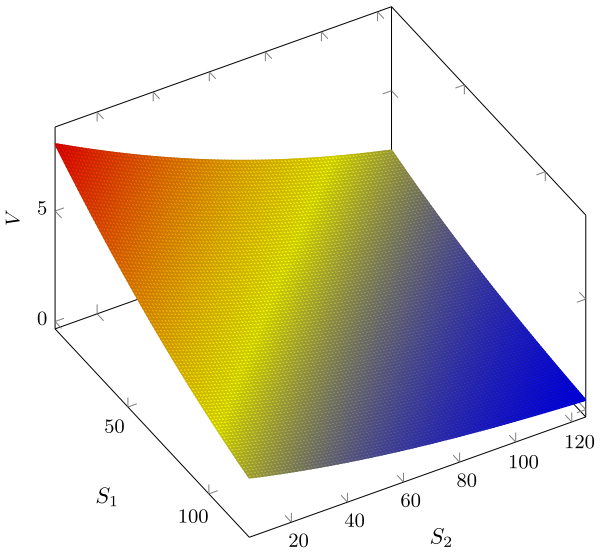
6. Conclusions

Thanks to the universal approximation property of ANNs and the dramatic increase of computing power of deep learning hardware, PINNs methods have become a serious alternative for solving complex PDE problems. Maybe, the biggest weakness of PINNs is the imposition of the boundary conditions, as they enter as addends into the loss function for the network calibration, and the user must choose heuristically the magnitude of the addends that depend, of course, on the type of problem and the type of boundary conditions. These weights are not known a priori, as they depend of the solution itself, and must be estimated in some way, which is also problem dependent.

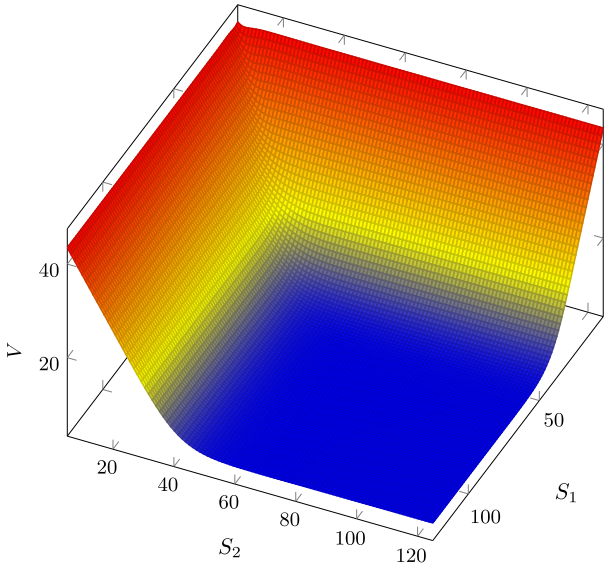
In this work a novel technique for the treatment of the boundary conditions in the PINNs framework has been introduced. It allows to get ride the heuristic selection of the weights of the boundary addends that appear in the loss function of the ANN that approximates the solution. The strategy is based on the direct evaluation of the differential operator at the boundaries taking into account the imposed boundary conditions. This yields an addend for the boundary



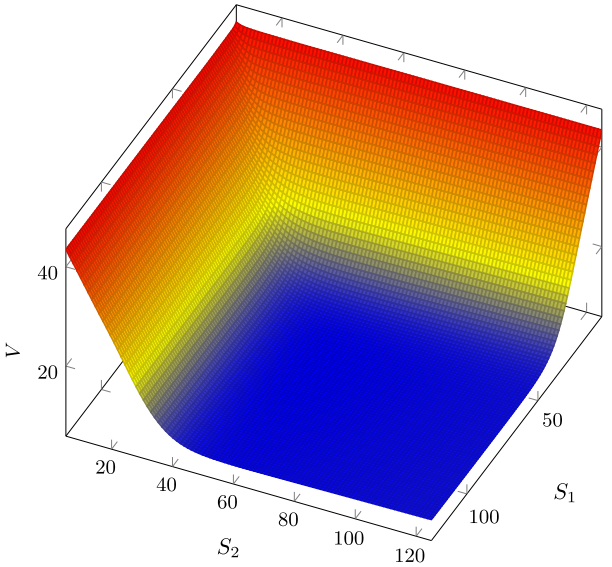
(a) 5-dimensional arithmetic average option value surface computed via trained ANN.



(b) 10-dimensional arithmetic average option value surface computed via trained ANN.



(c) 5-dimensional worst-of option value surface computed via trained ANN.



(d) 10-dimensional worst-of option value surface computed via trained ANN.

Fig. 7. Arithmetic average and worst-of put options driven by 5/10 assets with parameters given in Table 5. Slice with $S_i = 50,\ i = 3, \ldots, 10$.

that is in the same magnitude of the loss in the interior of the domain, avoiding to deal with the arbitrary choice of the weights. To the best of our knowledge, this procedure is introduced in this paper for the first time, and we feel that is a very interesting contribution that makes PINNs much more powerful and easier to use.

The new approach has been applied to several non-linear PDE problems that arise in computational finance when CCR is taking into account, although it is general enough and non problem-dependent to be applied in other fields, like for example fluid dynamics or solid mechanics. In particular, it has been employed to solve the boundary value problems related to the pricing of risky European options under the one and two-dimensional Black–Scholes model, as well as under the Heston model. In addition, the valuation of risk-free European 5 and 10-assets options under the Black–Scholes model is included in order to show the robustness of the proposed method for higher dimensional tasks. The obtained solutions yield a good accuracy when compared with reference solutions. Furthermore, embedding the obtained solution into an ANN has allowed us to compute their relevant partial derivatives by means of AD.

All in all, the partial derivatives computation in the PINNs framework is so far a generally unexplored avenue and we believe it may have a lot of potential, being one of the main advantages of PINNs over other deep-learning based

Table A.7
Parameters for Black–Scholes model considering counterparty risk.
Source: Obtained from [63].

Black Scholes parameters	
Domain, Ω	$[0, 60]$
Strike, K	15
Time to maturity, T	5
Volatility, σ	0.25
Repo rate, r_R	0.015
Interest rate, r	0.03
xVA parameters	
Seller hazard rate, λ_B	$[0.0, 0.1]$
Counterparty hazard rate, λ_C	0.05
Seller recovery rate, R_B	0.4
Counterparty recovery rate, R_C	0.4
Funding spread, s_F	$(1 - R_B)\lambda_B$

methodologies. The optimization procedure takes these quantities into account since they are implicitly part of the loss function, so that under the assumption of having an ideal optimizer, there would be a perfect fit of both the solution and the derivatives that conform the PDE. Another of the most notables advantages is in terms of interpretability. Compared to other techniques, this methodology is closer to the classical PDE schemes, in the sense that the PDE solution is projected onto a space formed by the ANN weights.

Data availability

No data was used for the research described in the article.

Acknowledgements

All the authors thank to the support received from the CITIC research center, funded by Xunta de Galicia and the European Union (European Regional Development Fund - Galicia Program, Spain), by grant ED431G 2019/01.

A.L and J.A.G.R. acknowledge the support received by the Spanish MINECO under research project number PDI2019-108584RB-I00, and by the Xunta de Galicia, Spain under grant ED431C 2018/33.

Appendix. Numerical experiments for the one-dimensional Black–Scholes equation

Numerical experiments for the one-dimensional Black–Scholes boundary value problem (26) with the initial condition (21) and the model parameters given in Table A.7 are presented.

A.1. Test settings

The neural network to be trained follows the description given in Section 2.1. The number of layers, l, and the units per layer, β, is discussed below. The training loss function is given by (31) for $d = 1$. The integrals appearing in this expression are computed with the trapezoidal rule, thus the set of training points, $\mathcal{P} = \mathcal{P}_{\mathcal{I}} \cup \mathcal{P}_{\mathcal{O}} \cup \mathcal{P}_{\Gamma^0} \cup \mathcal{P}_{\Gamma^+}$, comes from the uniform discretization of the interior domain and its boundaries. The experiments presented below are run with $|\mathcal{P}_{\mathcal{I}}| = 10\,692$, $|\mathcal{P}_{\mathcal{O}}| = 110$ and $|\mathcal{P}_{\Gamma^0}| = |\mathcal{P}_{\Gamma^+}| = 99$, yielding a total of 11,000 training points, which falls within the reference values that can be found in other works, such as [8]. The optimization involves 12 500 iterations. The first 10 000 are performed with Adam taking the learning rate as 10^{-3}, while the remaining ones are performed with L-BFGS. The accuracy of the approximation is measured by comparing its relative error with the analytic solution given in [63].

A.2. Numerical results

First, we check how the network's training behaves when varying its number of layers and neurons per layer. For this purpose, all 16 possible combinations between $l \in \{2, 4, 8, 16\}$ layers and $\beta \in \{10, 20, 40, 80\}$ units per layer are considered. For each combination, a sample of 10 training is made and the worst of them is chosen. The pricing of an European put option, V, is the target.

Most combinations give L^2 relative errors of the order of $10^{-3}/10^{-4}$, the worst ones being those where the number of layers is high, as the convergence of the method fails for some trials. Finding a balance between accuracy, robustness and execution time, we decide to work with $l = 4$ layers and $\beta = 40$ units per layer, where we have achieved relative errors of 2.33×10^{-4}, 2.90×10^{-4} and 5.13×10^{-4} for the L^1, L^2 and L^∞-norms, respectively.

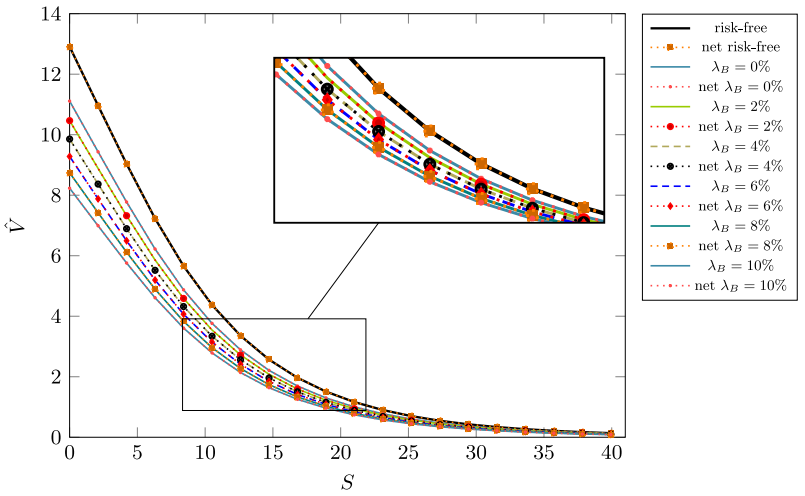
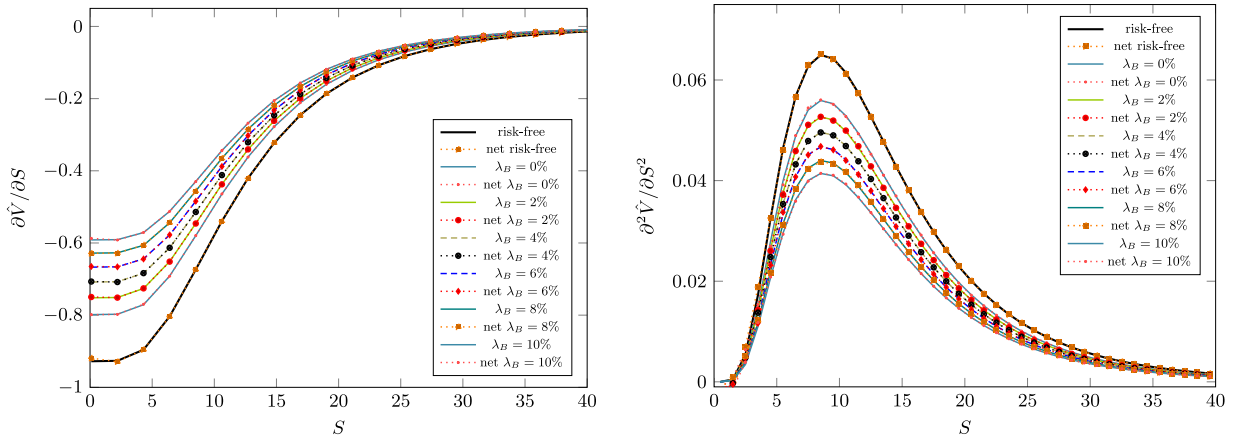


Fig. A.8. Comparison between analytical and approximated put option values for each default scenario.



(a) Put option deltas.

(b) Put option gammas.

Fig. A.9. Comparison between analytical and approximated put option deltas and gammas for each default scenario.

Once the size of our network has been selected, we show some results on its performance considering different default scenarios based on varying the seller hazard rate, λ_B.

Fig. A.8 shows the comparison between the analytical and PINNs approximated solution for each λ_B considered. The risk-free option is added for completeness. Regardless of the default scenario chosen, the quantitative behaviour of our approximation is identical to that given by the analytical solution. The accuracy of the approximation is particularly good in the neighbourhood of the strike (of the order of 10^{-4}), an area of interest in our pricing task.

By means of the AD, we can compute the derivative of the option price with respect to its related quantities. In Figs. A.9(a) and A.9(b) we observe the same comparison made for the price, now for delta and gamma Greeks, respectively. In the delta case, a slight decrease in accuracy is observed near the boundary $S = 0$, which also transfers to the gamma case, as expected. In the rest of the domain there is not a loss of accuracy with respect to the pricing case. The errors achieved for the second derivative are of the order of 10^{-3}. It is a solid performance since such a derivative presents numerical instabilities that makes it more difficult to compute. The Table A.8 presents the relative errors achieved in the neighbourhood of the strike for the risk-free case and the cases $\lambda_B = 2\%$, $\lambda_B = 10\%$. The remaining cases present a similar performance.

Table A.8
Relative errors for the put option price, delta and gamma, with S near the strike, for some default scenarios. Risk-free case ($\lambda_B = \lambda_C = 0$) is added for completeness.

Case	S	$\hat{V}$	$\partial\hat{V}/\partial S$	$\partial^2\hat{V}/\partial S^2$
Risk-free	12.5	5.55×10^{-4}	8.87×10^{-5}	1.39×10^{-4}
	15.0	6.99×10^{-4}	4.51×10^{-4}	2.22×10^{-3}
	17.5	6.71×10^{-4}	8.82×10^{-4}	2.36×10^{-3}
$\lambda_B = 2\%$	12.5	2.28×10^{-4}	5.92×10^{-4}	2.42×10^{-3}
	15.0	1.79×10^{-4}	2.60×10^{-4}	2.57×10^{-3}
	17.5	3.48×10^{-4}	1.43×10^{-5}	2.07×10^{-3}
$\lambda_B = 10\%$	12.5	1.01×10^{-4}	4.82×10^{-4}	8.89×10^{-4}
	15.0	1.67×10^{-5}	1.17×10^{-4}	2.40×10^{-3}
	17.5	4.51×10^{-6}	6.86×10^{-5}	1.21×10^{-3}

References

[1] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016.
[2] A.J. Meade, A.A. Fernandez, The numerical solution of linear ordinary differential equations by feedforward neural networks, Math. Comput. Modelling 19 (12) (1994) 1–25.
[3] M. Dissanayake, N. Phan-Thien, Neural network-based approximations for solving partial differential equations, Commun. Numer. Methods. Eng. 10 (3) (1994) 195–201.
[4] I.E. Lagaris, A. Likas, D.I. Fotiadis, Artificial neural networks for solving ordinary and partial differential equations, IEEE Trans. Neural Netw. 9 (5) (1998) 987–1000.
[5] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, 2014, arXiv:1412.6980.
[6] L. Bottou, F.E. Curtis, J. Nocedal, Optimization methods for large-scale machine learning, SIAM Rev. 60 (2) (2018) 223–311.
[7] X. Meng, Z. Li, D. Zhang, G.E. Karniadakis, PPINN: Parallel physics-informed neural network for time-dependent PDEs, Comput. Methods Appl. Mech. Engrg. 370 (2020) 113250.
[8] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, J. Comput. Phys. 378 (2019) 686–707.
[9] L. Lu, X. Meng, Z. Mao, G.E. Karniadakis, DeepXDE: A deep learning library for solving differential equations, SIAM Rev. 63 (1) (2021) 208–228.
[10] S. Mishra, R. Molinaro, Estimates on the generalization error of physics-informed neural networks for approximating PDEs, IMA J. Numer. Anal. (2022).
[11] M. De Florio, E. Schiassi, R. Furfaro, Physics-informed neural networks and functional interpolation for stiff chemical kinetics, Chaos 32 (6) (2022) 063107.
[12] T. De Ryck, S. Mishra, Error analysis for physics informed neural networks (PINNs) approximating Kolmogorov PDEs, 2021, arXiv:2106.14473.
[13] G. Bai, U. Koley, S. Mishra, R. Molinaro, Physics informed neural networks (PINNs) for approximating nonlinear dispersive PDEs, J. Comput. Math. 39 (6) (2021) 816–847.
[14] T. De Ryck, S. Mishra, R. Molinaro, wPINNs: Weak physics informed neural networks for approximating entropy solutions of hyperbolic conservation laws, 2022, arXiv:2207.08483.
[15] T. De Ryck, S. Mishra, Error analysis for deep neural network approximations of parametric hyperbolic conservation laws, 2022, arXiv:2207.07362.
[16] T. De Ryck, A.D. Jagtap, S. Mishra, Error estimates for physics informed neural networks approximating the Navier-Stokes equations, 2022, arXiv:2203.09346.
[17] T. De Ryck, S. Mishra, Generic bounds on the approximation error for physics-informed (and) operator learning, 2022, arXiv:2205.11393.
[18] T. Kossaczká, M. Ehrhardt, M. Günther, A neural network enhanced WENO method for nonlinear degenerate parabolic equations, Phys. Fluids 34 (2022).
[19] J. Han, A. Jentzen, E. Weinan, Solving high-dimensional partial differential equations using deep learning, Proc. Natl. Acad. Sci. 115 (34) (2018) 8505–8510.
[20] E. Weinan, M. Hutzenthaler, A. Jentzen, T. Kruse, Multilevel Picard iterations for solving smooth semilinear parabolic heat equations, Partial Differ. Equ. Appl. 2 (6) (2021).
[21] M. Hutzenthaler, A. Jentzen, T. Kruse, T. Anh Nguyen, P. von Wurstemberger, Overcoming the curse of dimensionality in the numerical approximation of semilinear parabolic partial differential equations, Proc. R. Soc. Lond. Ser. A Math. Phys. Eng. Sci. 476 (2020).
[22] F. Heldmann, S. Treibert, M. Ehrhardt, K. Klamroth, PINN training using biobjective optimization: The trade-off between data loss and residual loss, SSRN Electron. J. (2022).
[23] J. Yu, L. Lu, X. Meng, G.M. Karniadakis, Gradient-enhanced physics-informed neural networks for forward and inverse PDE problems, Comput. Methods Appl. Mech. Engrg. 393 (2022) 114823.
[24] D. Duffie, M. Huang, Swap rates and credit quality, J. Finance 51 (3) (1996) 921–949.
[25] D. Brigo, M. Masetti, in: Pykhtin (Ed.), Risk-neutral pricing of counterparty risk, Risk Books, London, 2005.
[26] U. Cherubini, Counterparty risk in derivatives and collateral policies: The replicating portfolio approach, in: L. Tilman (Ed.), ALM of Financial Institutions, institutional Investor Books, 2005.
[27] D. Brigo, A. Pallavicini, V. Papatheodorou, Arbitrage-free valuation of bilateral counterparty risk for interest-rate products: impact of volatilities and correlations, Int. J. Theor. Appl. Finance 14 (06) (2011) 773–802.
[28] D. Brigo, A. Capponi, A. Pallavicini, Arbitrage-free bilateral counterparty risk valuation under collateralization and application to credit default swaps, Math. Finance 24 (1) (2014) 125–146.
[29] V. Piterbarg, Funding beyond discounting: collateral agreements and derivatives pricing, Risk Mag. 23 (02) (2010) 97–102.
[30] V. Piterbarg, Cooking with collateral, Risk Mag. 23 (02) (2012) 58–63.
[31] M. Fujii, Y. Shimada, A. Takahashi, Note on construction of multiple swap curves with and without collateral, SSRN Electron. J. 23 (02) (2010).
[32] M. Fujii, Y. Shimada, A. Takahashi, A market model of interest rates with dynamic basis spreads in the presence of collateral and multiple currencies, Wilmott J. 54 (2011) 61–73.
[33] A. Gnoatto, N. Seiffert, Cross currency valuation and hedging in the multiple curve framework, SIAM J. Financial Math. 12 (3) (2021) 967–1012.

[34] C. Cuchiero, C. Fontana, A. Gnoatto, Affine multiple yield curve models, Math. Finance 29 (2) (2019) 568–611.
[35] A. Pallavicini, D. Perini, D. Brigo, Funding valuation adjustment: a consistent framework including CVA, DVA, collateral, netting rules and re-hypothecation, SSRN Electron. J. (2011).
[36] D. Brigo, A. Pallavicini, Nonlinear consistent valuation of CCP cleared or CSA bilateral trades with initial margins under credit, funding and wrong-way risks, J. Financ. Eng. 01 (01) (2014) 1450001.
[37] D. Brigo, C. Buescu, M. Francischello, A. Pallavicini, M. Rutkowski, Risk-neutral valuation under differential funding costs, defaults and collateralization, Risk Manag. Anal. Financ. Inst. eJ. (2018).
[38] C. Burgard, M. Kjaer, Partial differential equation representations of derivatives with bilateral counterparty risk and funding costs, J. Credit Risk 7 (3) (2011) 1–19.
[39] C. Burgard, M. Kjaer, In the balance, Risk J. (2011) 72–75.
[40] S. Crépey, Bilateral counterparty risk under funding constraints-part II: CVA, Math. Finance 25 (1) (2015) 23–50.
[41] S. Crépey, Gaussian process regression for derivative portfolio modelling and application to credit valuation adjustment computations, Risk J. 24 (1) (2020) 47–81.
[42] M. Bichuch, A. Capponi, S. Sturm, Arbitrage-free pricing of XVA - Part II: PDE representation and numerical analysis, SSRN Electron. J. (2015).
[43] B. Salvador, C.W. Oosterlee, R. van der Meer, Financial option valuation by unsupervised learning with artificial neural networks, Mathematics 9 (1) (2021).
[44] E. Weinan, J. Han, A. Jentzen, Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations, Commun. Math. Stat. 5 (2017) 349–380.
[45] A. Gnoatto, A. Picarelli, C. Reisinger, Deep xVA solver – A neural network based counterparty credit risk management framework, SSRN Electron. J. (2020).
[46] B. Horvath, A. Muguruza, M. Tomas, Deep learning volatility: a deep neural network perspective on pricing and calibration in (rough) volatility models, Quant. Finance 21 (1) (2021) 11–27.
[47] B. Huge, A. Savine, Differential machine learning, 2020, arXiv:2005.02347.
[48] S. Liu, A. Leitao, A. Borovykh, C.W. Oosterlee, On a neural network to extract implied information from American options, Appl. Math. Finance 28 (5) (2021) 449–475.
[49] Y. Shin, J. Darbon, G.E. Karniadakis, On the convergence of physics informed neural networks for linear second-order elliptic and parabolic type PDEs, Commun. Comput. Phys. 28 (5) (2020) 2042–2074.
[50] R. van der Meer, C.W. Oosterlee, A. Borovykh, Optimally weighted loss functions for solving PDEs with neural networks, J. Comput. Appl. Math. 405 (2022).
[51] A. Karpatne, R. Kannan, V. Kumar, Knowledge-guided machine learning: Accelerating discovery using scientific knowledge and data, first ed., Chapman and Hall/CRC, 2022.
[52] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al., TensorFlow: A system for large-scale Machine Learning, in: 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), 2016, pp. 265–283.
[53] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Automatic differentiation in PyTorch, in: Neural Information Processing Systems, Tech. Rep, 2017.
[54] D.C. Liu, J. Nocedal, On the limited memory BFGS method for large scale optimization, Math. Program. 45 (1) (1989) 503–528.
[55] C. Wang, S. Li, D. He, L. Wang, Is L^2 physics-informed loss always suitable for training physics-informed neural network? 2022.
[56] C. Wu, M. Zhu, Q. Tan, Y. Kartha, L. Lu, A comprehensive study of non-adaptive and residual-based adaptive sampling for physics-informed neural networks, Comput. Methods Appl. Mech. Engrg. 403 (2023) 115671.
[57] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: A simple way to prevent neural networks from overfitting, J. Mach. Learn. Res. 15 (1) (2014) 1929–1958.
[58] A. Geron, Hands-on machine learning with Scikit-Learn and TensorFlow : Concepts, tools, and techniques to build intelligent systems, O'Reilly Media, Sebastopol, CA, 2017.
[59] D. Brigo, M. Francischello, A. Pallavicini, Nonlinear valuation under credit, funding, and margins: Existence, uniqueness, invariance, and disentanglement, European J. Oper. Res. 274 (2) (2019) 788–805.
[60] B. Salvador, C.W. Oosterlee, Total value adjustment for a stochastic volatility model. A comparison with the Black-Scholes model, Appl. Math. Comput. 391 (2021) 125489.
[61] C.C.W. Leentvaar, Pricing multi-asset options with sparse grids (Ph.D. thesis), Electrical Engineering, Mathematics and Computer Science, University of Delft, 2008.
[62] R.M. Stulz, Options on the minimum or the maximum of two risky assets: Analysis and applications, J. Financ. Econ. 10 (2) (1982) 161–185.
[63] Y. Chen, C. Christara, Penalty methods for bilateral XVA pricing in European and American contingent claims by a partial differential equation model, J. Comput. Finance 24 (4) (2021) 41–70.
[64] C. Randall, D.A. Tavella, Pricing financial instruments: The finite difference method, Wiley, 2000.
[65] J.C. Cox, J.E. Ingersoll, S.A. Ross, A theory of the term structure of interest rates, Econometrica 53 (2) (1985) 385–407.
[66] S.L. Heston, A closed-form solution for options with stochastic volatility with applications to bond and currency options, Rev. Financ. Stud. 6 (2) (1993) 327–343.
[67] D. Castillo, A.M. Ferreiro, J.A. García-Rodríguez, C. Vázquez, Numerical methods to solve PDE models for pricing business companies in different regimes and implementation in GPUs, Appl. Math. Comput. 219 (24) (2013) 11233–11257.
[68] I. Arregui, B. Salvador, C. Vázquez, PDE models and numerical methods for total value adjustment in European and American options with counterparty risk, Appl. Math. Comput. 308 (2017) 31–53.
[69] K.J. In 't Hout, S. Foulon, ADI finite difference schemes for option pricing in the Heston model with correlation, Int. J. Numer. Anal. Model. 7 (2) (2010) 303–320.