The 9th European Congress on Computational Methods in Applied Sciences and Engineering
ECCOMAS Congress 2024
3–7 June 2024, Lisboa, Portugal

NEW TREATMENT OF THE BOUNDARY CONDITIONS IN DEEP-GALERKIN/PINNS METHODS: APPLICATION TO HIGH DIMENSIONAL NON-LINEAR PDES

José A. García-Rodríguez[1,2], Álvaro Leitao[1,2], Joel P. Villarino[1,2]

[1] M2NICA research group
Department of Mathematics, University of A Coruña, Spain

[2] CITIC Research Centre
University of A Coruña, Spain

Key words: High dimensional, nonlinear parabolic PDEs, deep learning, PINNs

Summary. The goal of this work is to develop robust and stable deep learning numerical methods for solving high-dimensional nonlinear parabolic PDE models using Deep-Galerkin/PINNs techniques. 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 numerical methods and thus also in the ANN setting. The main contribution of this work is the novel treatment of the boundary conditions in PINNs framework. The common approach consists of assigning weights to the different terms involved in the loss function, where the selection of these weights can be done heuristically or using optimization procedures. We introduce a new idea that consists of taking into account the loss terms due to the boundary conditions by means of evaluating the PDE operator restricted to the boundaries. Although this is nonfeasible in the classical PDE solving algorithms, it is very intuitive within the PINNs framework since, using AD, we can evaluate this operator in the boundary even if it contains normal derivatives to such a boundary. In our approach, the boundary operators are reformulated, eliminating the choice of the boundary loss weights. Another advantage is that in this way the value of such addends is of the same magnitude as the interior losses. Although the proposed methodology is general we show the solution of PDE models for challenging problems appearing the the computational finance field. In particular, we consider the derivative valuation problem in the presence of counterparty credit risk, which includes in its formulation the so-called x-value adjustments (XVA).

1 INTRODUCTION

Deep learning techniques are machine learning algorithms based on neural networks, also known as artificial neural networks (ANNs). From a mathematical point of view, ANNs can be interpreted as multiple chained compositions of affine maps with non-linear and element-wise activation functions. Such networks are known for being universal approximators, i.e, in theory, they are capable of representing any continuous function with a prescribed accuracy. This property was exploited in the seminal paper [3] to introduce a technique to solve partial differential equations (PDEs) using ANNs. The main idea behind the proposal of the authors is to write the solution in terms of the network representation, depending on the affine map

coefficients, known in the field as network weights. To achieve the solution approximation in terms of the network, it can be trained to learn data from a physical law that is given by the PDE. This leads to transforming the associate boundary value problem into a high dimensional nonlinear optimization problem where, now, the network weights are adjusted using optimization algorithms such as stochastic gradient descent-based methods and/or quasi-Newton methods.

Nowadays, with the advances in hardware (GPUs) and software (automatic differentiation algorithms), the technique has gained increasing attention in the literature under the name of physics-informed neural networks (PINNs), [4]. This discipline is rapidly advancing in research, largely due to its significant advantages: it can handle nonlinear PDEs with minimal effort, extend to moderately high dimensions, and provide highly accurate approximations of partial derivatives of the solution, thanks to automatic differentiation in modern deep learning frameworks; but also for many of the disadvantages it still entails, such as the limited knowledge of theoretical results related to algorithm convergence, or the difficulty when imposing the boundary conditions.

The present work aims to present a robust and stable PINN method for solving nonlinear parabolic PDE models. We mainly focus on how to impose the boundary conditions within the methodology, a critical and challenging task in both classical and PINN settings. The most widely approach used to impose boundary conditions in the PINNs setting involves assigning weights to the different terms involved in the loss function, where they are selected either heuristically or through optimization. Instead, we introduce the boundary condition loss terms through the evaluation of the PDE operator directly on the boundaries, ensuring that these addends are of similar magnitude to the interior domain losses. Unlike classical PDE solvers, this method is feasible in the PINNs framework due to automatic differentiation, which allows the evaluation of the operator on the boundary, even when normal derivatives are involved.

The outline of this paper is as follows. In Section 2 we summarize the PINNs framework for solving PDEs. Section 3 introduces our proposed methodology for imposing the boundary conditions in the PINNs setting. Lastly, in Section 4, we conduct numerical experiments applied to selected high-dimensional boundary value problems.

2 PINNs

In this section, we present the PINNs methodology for solving PDEs. The illustration is carried out according to the kind of models which we treat in the numerical experiments. Essentially, semilinear parabolic PDEs with source term. 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

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

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 Dirichlet and/or Neumann boundary conditions. The goal is to approximate this unknown function u using a feed-forward neural network, $u_\theta(t, x) := u(t, x; \theta)$, where $\theta \in \mathbb{R}^P$ are the network parameters.

To get an approximation of the function u using a neural network, u_θ, we need to find the parameters of the network, $\theta \in \mathbb{R}^P$, that yields the best approximation of (1), leading to a global optimization problem that can be written in terms of the minimization of a function that measures how good the approximation is. Such a function is commonly known as loss function. Thus, the boundary value problem (1) is transformed into an unconstrained optimization problem where the loss functions involve the L^2 error of the interior, initial and boundary residuals. Mathematically speaking, the loss function, $\mathcal{J}(\theta)$, is defined as

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

or, equivalently,

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

where

$$\mathcal{R}_\theta^{\mathcal{I}}(t,x) := \frac{\partial u_\theta}{\partial t}(t,x) + \mathcal{L}[u_\theta](t,x) - f(u_\theta(t,x)), \qquad (t,x) \in (0,T)\times\mathring{\Omega}, \quad (3)$$

$$\mathcal{R}_\theta^{\mathcal{B}}(t,x) := \mathcal{B}[u_\theta](t,x) - g(t,x), \qquad (t,x) \in (0,T)\times\partial\Omega, \quad (4)$$

$$\mathcal{R}_\theta^{\mathcal{O}}(x) := u_\theta(0,x) - u_0(x), \qquad x \in \Omega, \quad (5)$$

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 or updateable hyperparameters that allow imposing a weight to loss term. Note that, for the computation of the residuals (3), (4), we need to compute the neural network derivatives concerning the input space and time variables, well defined under the premise of using sufficiently smooth activation functions. Numerically, they are calculated with automatic differentiation. Finally, we can write the minimization problem as follows:

Find $\theta^* \in \Theta$ such that

$$\theta^* = \arg\min_{\theta\in\Theta} \mathcal{J}(\theta), \quad (6)$$

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

Except for the simple cases, the integrals appearing in (2) must be computed numerically using quadrature rules, [5]. For this reason, we need to select a set of training points, $\mathcal{P} = \mathcal{P}_{\mathcal{I}} \cup \mathcal{P}_{\mathcal{B}} \cup \mathcal{P}_{\mathcal{O}}$ acting as nodes in the quadrature formulas. Thus, the training points should be selected according to the chosen quadrature technique. Since we work with high dimensional problems, the better choice to circumvent the curse of dimensionality is the use of random sampling, associated with a Monte Carlo integration method. Based on this choice, we can rewrite our loss function as:

$$\hat{\mathcal{J}}(\theta) = \hat{\lambda_{\mathcal{I}}} MSE_{\mathcal{I}} + \hat{\lambda_{\mathcal{B}}} MSE_{\mathcal{B}} + \hat{\lambda_{\mathcal{O}}} MSE_{\mathcal{O}}, \quad (7)$$

where MSE_j, $j \in \{\mathcal{I},\mathcal{B},\mathcal{O}\}$, denotes the mean squared error function for the interior domain, boundaries and initial condition, respectively.

Once all the aforementioned aspects have been addressed, we are in a position to apply an optimization procedure to find the minimum of the problem (6), a process referred to as

"training". The problem, now formulated in the parameter space of the neural network, is high dimensional and highly non-convex, which means that the uniqueness of the solution is not guaranteed. As a result, we can only expect to find a sufficiently low local minimum. Given these characteristics, we employ a stochastic gradient descent-based method known as Adam to conduct our numerical experiments.

3 PROPOSED TREATMENT OF BOUNDARY CONDITIONS

Ideally, the loss function should accurately measure how far away we are from the exact solution and how well the boundary restrictions are satisfied, allowing the optimization algorithm to reach a good local minimum. However, in practice, this is not always the case when applying numerical methods. This issue persists in the case of PINNs algorithm, and previous works, such as [6], suggest that training often prioritizes minimizing the PDE residual on the interior domain, while neglecting boundary conditions, resulting in large errors in boundary fitting.

One of the main solutions to this problem is to introduce the lambda weights seen before, which adjust the contribution of each loss term, see, for example, [1] and the references therein. The optimal selection of these weights is crucial for the success of the algorithm, but it is often problem-dependent and typically determined heuristically, as noted in [6].

In our view, the need for these factors arises because the magnitude of the loss term can vary significantly, with some terms becoming negligible compared to others. This imbalance can lead to poorer local minima or require extended training times. For instance, if boundary losses are much larger than the PDE residual loss, more training time may be needed to reduce these differences. Conversely, if boundary residuals are too small, their contribution may be negligible, providing little value to the training process.

The arbitrary selection of loss function weights often arises due to significant differences in magnitude among residuals, leading to potential imbalances in model training. To overcome this issue, we propose a novel approach that reformulates the residuals associated with Dirichlet or Neumann (including Robin and higher-order derivative) boundary conditions. Instead of using the boundary condition itself as the residual, our technique takes the resulting PDE restricted to the corresponding boundary as the residual. This approach ensures that the magnitude of these residuals is comparable to that of the interior residuals once all quantities are nondimensionalized, thereby producing more balanced losses and mitigating the need for arbitrary weighting. Most of the Neumann, Robin, or higher-order derivative boundary residuals can be expressed using this approach. The process involves substituting the boundary condition directly into the interior PDE operator, and then applying the resulting equation as the boundary residual. However, this method of imposing Dirichlet conditions is only applicable when the Dirichlet condition naturally emerges at the boundary.

As an example, consider a specific parabolic problem with Dirichlet and Neumann boundary conditions, defined within the spatial domain $\Omega = \prod_{i=1}^{d} \left[x_i^{\min}, x_i^{\max}\right]$, focusing on the upper boundaries.

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

and the lower boundaries

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

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

$$\begin{cases} \dfrac{\partial u_\theta}{\partial t} + \displaystyle\sum_{i,j=1}^{d} a_{ij} \frac{\partial^2 u_\theta}{\partial x_i \partial x_j} + \sum_{i=1}^{d} b_i \frac{\partial u_\theta}{\partial x_i} + f(u_\theta) = 0, & \text{in } (0,T) \times \mathring{\Omega}, \\ \dfrac{\partial u_\theta}{\partial x_i} - g_i = 0 & \text{in } \Gamma_i^+ = (0,T) \times \Gamma_{x_i}^+, \ \ i = 1, \dots d, \\ u_\theta - h_i = 0 & \text{in } \Gamma_i^0 = (0,T) \times \Gamma_{x_i}^0, \ \ i = 1, \dots d, \\ u_\theta - u_0 = 0 & \text{in } \Omega, \end{cases} \tag{8}$$

where $\{a_{ij}\}_{i,j=1}^d \subset \mathbb{R}$, $\{b_i\}_{i=1}^d \subset \mathbb{R} \setminus \{0\}$, and $g_i \in \mathcal{C}(\Gamma_i^+, \mathbb{R})$, $h_i \in \mathcal{C}(\Gamma_i^0, \mathbb{R})$, $i = 1, \dots d$. 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.

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

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

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

For Dirichlet conditions, the proposed strategy can only be applied when h_i verifies the PDE and the initial condition of (8) 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.,

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

Because of that, such kind of Dirichlet conditions do 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.

In this work, to avoid adding any optimizable weights to the loss function, we pretrain the network against the initial condition. This not only brings the magnitude of the initial condition loss closer to the other loss terms but also offers additional benefits, such as improving and accelerating training by starting the network from a state closer to the desired solution.

4 NUMERICAL EXPERIMENTS

In this section, we present numerical experiments to validate the proposed methodology. We present the solution of high-dimensional nonlinear scalar PDEs that arise in the context of valuing European options considering Credit Counterparty Risk (CCR), see [2]. These experiments focus on solving the associated boundary value problem, where the goal is to find the option

price, denoted by $\hat{V} : [0,T] \times \Omega \subset \mathbb{R}^{d+1} \longrightarrow \mathbb{R}$. The PDE model is given by:

$$\begin{cases} \dfrac{\partial \hat{V}}{\partial t} - \dfrac{1}{2}\displaystyle\sum_{i=1}^{d}\sum_{j=1}^{d} \rho_{ij}\sigma_i\sigma_j S_i S_j \frac{\partial^2 \hat{V}}{\partial S_i \partial S_j} & \\ \qquad\qquad - \displaystyle\sum_{i=1}^{d} r_{R_i} S_i \frac{\partial \hat{V}}{\partial S_i} + r\hat{V} + f(\hat{V}) = 0, & \text{in } (0,T) \times \mathring{\Omega}, \\ \hat{V} - h_i = 0, & \text{in } \Gamma_i^0,\ i \in \{1,2,\ldots,d\}, \\ \dfrac{\partial^2 \hat{V}}{\partial S_i^2} = 0, & \text{in } \Gamma_i^+,\ i \in \{1,2,\ldots,d\}, \\ \hat{V} - H = 0, & \text{in } \{0\} \times \Omega, \end{cases} \tag{9}$$

with the nonlinear source term

$$f(\hat{V}) = \lambda_B(1-R_B)\min\Big\{\hat{V},0\Big\} + \lambda_C(1-R_C)\max\Big\{\hat{V},0\Big\} + s_F\max\Big\{\hat{V},0\Big\}. \tag{10}$$

The initial condition H is given by the nature of the financial derivative. In this work, we present results depending on the following payoff functions:

- the arithmetic average basket function,

$$H(S_1,S_2,\ldots,S_d) = \max\Big\{K - \frac{1}{d}\sum_{i=1}^{d} S_i, 0\Big\}, \tag{11}$$

- and the worst-of function,

$$H(S_1,S_2,\ldots,S_d) = \max\Big\{K - \min\Big\{S_1,S_2,\ldots,S_d\Big\}, 0\Big\}, \tag{12}$$

with $K > 0$ being the fixed strike price. In addition, we named h_i the Dirichlet condition associated with the chosen payoff.

Such formulation fits into problem (1) so we can apply everything explained in Section 2 to solve it. To do this, we consider the general training set $\mathcal{P} = \mathcal{P}_{\mathcal{I}} \cup \mathcal{P}_{\mathcal{O}} \cup \mathcal{P}_{\mathcal{B}}$, sets of sampled points chosen for the interior domain, initial condition and boundaries, respectively. All of them are sampled using a pseudo-random sampling procedure.

As an approximation of the 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. We assume that the number of neurons per hidden layer is the same, β. Based on the boundary value problem given in (9), we choose the network residuals taking into account the proposal given in Section 3 to solve the aforementioned training issues. Thus,

the residual-based closed-forms are:

$$\mathcal{R}_\theta^{\mathcal{I}} = \frac{\partial \hat{V}_\theta}{\partial t} - \frac{1}{2}\sum_{i=1}^{d}\sum_{j=1}^{d}\rho_{ij}\sigma_i\sigma_j S_i S_j \frac{\partial^2 \hat{V}_\theta}{\partial S_i \partial S_j} - \sum_{i=1}^{d} r_{R_i} S_i \frac{\partial \hat{V}_\theta}{\partial S_i} + r\hat{V}_\theta + f(\hat{V}_\theta), \quad \text{in } (0,T)\times \mathring{\Omega}, \tag{13}$$

$$\mathcal{R}_\theta^{\Gamma_k^0} = \frac{\partial \hat{V}_\theta}{\partial t} - \frac{1}{2}\sum_{\substack{i=1\\ i\neq k}}^{d}\sum_{\substack{j=1\\ j\neq k}}^{d}\rho_{ij}\sigma_i\sigma_j S_i S_j \frac{\partial^2 \hat{V}_\theta}{\partial S_i \partial S_j} - \sum_{\substack{i=1\\ i\neq k}}^{d} r_{R_i} S_i \frac{\partial \hat{V}_\theta}{\partial S_i} + r\hat{V}_\theta + f(\hat{V}_\theta), \quad \text{in } \Gamma_k^0,\ k=1,\dots,d, \tag{14}$$

$$\mathcal{R}_\theta^{\Gamma_k^+} = \frac{\partial \hat{V}_\theta}{\partial t} - \frac{1}{2}\sum_{\substack{i=1\\ i\neq k}}^{d}\sum_{j=1}^{d}\rho_{ij}\sigma_i\sigma_j S_i S_j \frac{\partial^2 \hat{V}_\theta}{\partial S_i \partial S_j} - \sum_{i=1}^{d} r_{R_i} S_i \frac{\partial \hat{V}_\theta}{\partial S_i} + r\hat{V}_\theta + f(\hat{V}_\theta), \quad \text{in } \Gamma_k^+,\ k=1,\dots,d, \tag{15}$$

$$\mathcal{R}_\theta^{\mathcal{O}} = \hat{V}_\theta - H, \quad \text{in } \{0\}\times\Omega. \tag{16}$$

We test the performance of the network trained using the PINNs methodology in the task of approximating the solutions of 5/10-dimensional basket options with all the assets having the same volatility and drift, and they being equally correlated.

In order to validate the accuracy of the solver, we are going to consider the risk-free problem (9) with the initial conditions (11) and (12). In this case we can measure the error against an approximate solution obtained via Monte Carlo simulation with low discrepancy sequences, ensuring at least a maximum error of 10^{-6}. We aim to show that the boundary treatment proposed in Section 3 is readily applicable to high dimensional problems. The model parameters are presented in Table 1.

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

Table 1: Parameters for basket option with 5/10 underlings.

In order to perform the numerical experiment we follow the guidelines presented in [7]. For the network, we consider $L = 4$ layers and $\beta = 128/256$ units per layer. We apply a warm-up to the network weights consisting on a weakly supervised pre-training against the initial condition. The core of the training is based on the minimization of the loss function (7) with the residual-based expressions (13)-(15).

The training set is resampled at specific iterations depending on the optimizer in use. For the Adam optimizer, during each iteration, a sample of 6144/32768 random points is drawn from the domain. This includes:

- 2048/16384 points from the interior domain,
- 2048/8192 from the initial condition,
- 2048/8192 from the boundary set (randomly selected from all boundaries).

For the arithmetic average basket option, we utilized $30,000$ Adam iterations with a linear decay in the learning rate, starting at 5×10^{-3} and decreasing to 5×10^{-5}. In the case of the related 10-dimensional option, we retained the same number of Adam iterations, but applied a linear decay in the learning rate from 1×10^{-3} to 5×10^{-5}.

After the stochastic gradient descent optimization, we apply the L-BFGS optimizer to refine the results achieved. For this, a sample of 24576/49152 random points is selected every 200 iterations, distributed as 8192/16384 from each of the same subsets used in the Adam's case. The maximum number of iterations for L-BFGS is capped at 15,000.

After the whole training process, we achieve an average relative error in the At-The-Money (ATM) neighbourhood between 8.87×10^{-3} and 5.08×10^{-3} for all training converged in the 5 dimensional example. In practice, we found that the application of the the L-BFGS optimizer does not always improve the solution achieved by the Adam optimizer, even loosing the convergence.

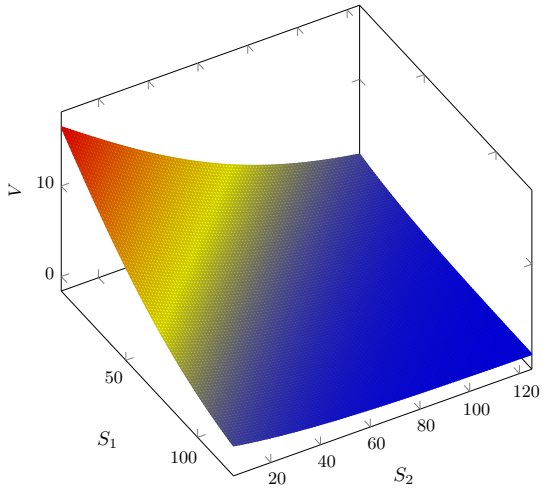
These cases typically exhibit a similar trend: the loss linked to the initial conditions significantly drops beneath the total observed loss throughout the L-BFGS optimization routine. This anomaly is probably caused by integrating L-BFGS with mini-batch sampling, allowing certain training samples to steer the optimizer toward suboptimal local minima.

	Arithmetic average		Worst-of	
(S_1, S_2)	$d = 5$	$d = 10$	$d = 5$	$d = 10$
$(50, 50)$	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}

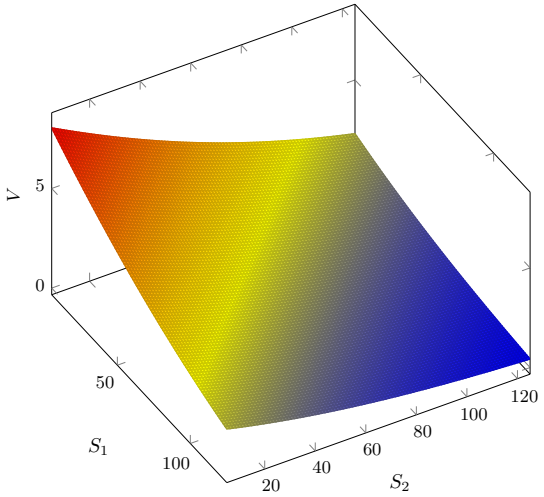
Table 2: Relative errors for the 5/10-dimensional arithmetic average and worst of put options. We fix the values of the first two assets of the basket, while all the remaining ones are set to 50. In addition, the average relative error achieved for 1000/10000 random values of the 5/10 assets between 40 and 60 is added.

For the worst-of options it has been necessary to extend the number of Adam iterations to 10000 while keeping 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 range 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. There is a deterioration of the approximation as we move further into the Out-The-Money (OTM) region, which is expected as the values are closer to zero; and the worst-of option shows more difficulties in the ATM

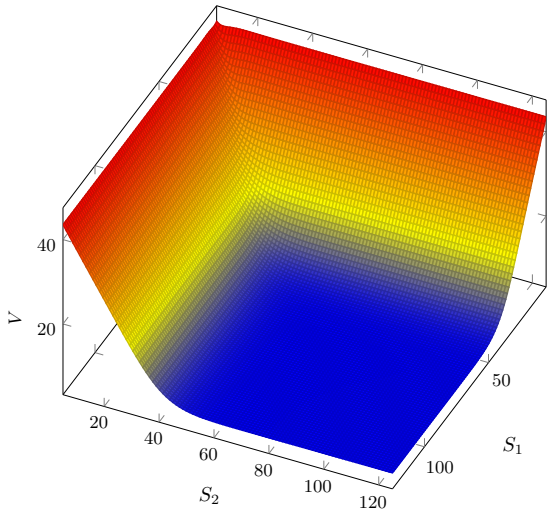
region than the arithmetic option. Table 2 shows the relative errors obtained for specific points, as well as the average errors given above. In addition, surface slices of the 5/10-dimensional option value predictions are plotted in Figure 1. In general, the results are rather satisfactory and sufficient for practical applications within the financial industry.



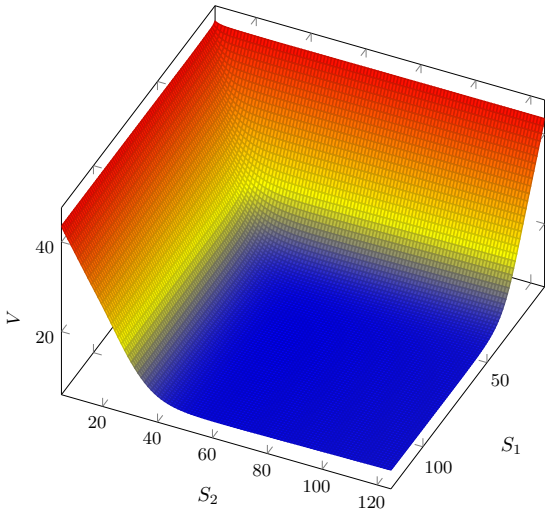
(a) 5-dimensional arithmetic average option value surface computed via trained ANN.



(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.

Figure 1: Arithmetic average and worst-of put options driven by 5/10 assets with parameters given in Table 1. Slice with $S_i = 50,\ i = 3, \ldots, 10$.

5 CONCLUSIONS

With the universal approximation property of artificial neural networks and significant progress in deep learning hardware, PINNs have become an effective method for solving PDEs. A major challenge in applying PINNs is the implementation of boundary conditions, which are typically incorporated into the loss function with weights that must be heuristically determined based on the specific problem and its solution, as these weights are not known beforehand.

This paper introduces a new technique for managing boundary conditions in PINNs, thereby eliminating the need for heuristic weight selection in the loss function. The proposed method directly evaluates the differential operator at the boundaries, thereby integrating the imposed boundary conditions naturally. This results in boundary terms that are automatically scaled to correspond with the interior loss, removing the necessity for arbitrary weight assignments.

Our approach has been tested on several nonlinear PDE problems prevalent in computational finance, especially those involving CCR. Nonetheless, the method is general and not limited to any particular application area, making it suitable for use in various fields. In particular, we applied this technique to solve boundary value problems related to the pricing of five- and ten-dimensional European options. The solutions achieved exhibit high accuracy when compared to established reference solutions, demonstrating the method's efficacy.

REFERENCES

[1] Yu, J. and Lu, L. and Meng, X and Karniadakis, G. M. 2022. “Gradient-enhanced physics-informed neural networks for forward and inverse PDE problems.” Computer Methods in Applied Mechanics and Engineering.

[2] Burgard, C. and Kjaer, M. 2011. Partial differential equation representations of derivatives with bilateral counterparty risk and funding costs.” The Journal of Credit Risk.

[3] Dissanayake, M. and Phan-Thien, N. 1994. ”Neural network-based approximations for solving partial differential equations.” Communications in Numerical Methods in Engineering.

[4] Raissi, M. and Perdikaris, P. and Karniadakis, G. E. 2019. ”Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations.” Journal of Computational Physics.

[5] Mishra, S. and Molinaro, R. 2022. ”Estimates on the generalization error of physics-informed neural networks for approximating PDEs.” IMA Journal of Numerical Analysis.

[6] Karpatne, A. and Kannan, R. and Kumar, V. 2022. ”Knowledge-guided machine learning: Accelerating discovery using scientific knowledge and data.” Chapman and Hall/CRC, first edition.

[7] P. Villarino, J. and Leitao, A. and García Rodríguez, J. A. 2023. ”Boundary-safe PINNs extension: Application to non-linear parabolic PDEs in counterparty credit risk.” Journal of Computational and Applied Mathematics.