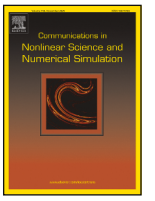
Commun Nonlinear Sci Numer Simulat 156 (2026) 109709

Contents lists available at ScienceDirect

Communications in Nonlinear Science and Numerical Simulation

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



Research Paper

A deep implicit-explicit minimizing movement method for partial integro-differential equations, with application to option pricing in jump-diffusion models ⋆

Emmanuil H. Georgoulis [a,b,c,*], Antonis Papapantoleon [d,b,c], Costas Smaragdakis [e,c]

[a] The Maxwell Institute for Mathematical Sciences and Department of Mathematics, Heriot-Watt University, EH14 4AS, Edinburgh, United Kingdom
[b] Department of Mathematics, School of Applied Mathematical and Physical Sciences, National Technical University of Athens, 15780, Zografou, Greece
[c] Institute of Applied and Computational Mathematics, FORTH, 70013, Heraklion, Greece
[d] Delft Institute of Applied Mathematics, TU Delft, 2628, Delft, The Netherlands
[e] Department of Statistics and Actuarial-Financial Mathematics, University of the Aegean, 83200, Karlovassi Samos, Greece

ARTICLE INFO

2020 MSC:
68T07
65C20
91G60
91G20
65M12

Keywords:
Basket option
Jump-diffusion model
PIDE
Minimizing movement method
Implicit-explicit method
Artificial neural network
Integral operator
Gauss-Hermite quadrature

ABSTRACT

We develop a novel deep learning approach for solving partial integro-differential equations (PIDEs) in high dimensions, involving diffusion and drift terms. To showcase its practicality and versatility, the methodology is presented for the specific challenge of pricing European basket options written on assets that follow jump-diffusion dynamics. The option pricing problem is formulated as a partial integro-differential equation, which is approximated via a new implicit-explicit minimizing movement time-stepping approach, involving approximation by deep, residual-type Artificial Neural Networks (ANNs) for each time step. The integral operator is discretized via two different approaches: (a) a sparse-grid Gauss–Hermite approximation following localised coordinate axes arising from singular value decompositions, and (b) an ANN-based high-dimensional special-purpose quadrature rule. Crucially, the proposed ANN is constructed to ensure the appropriate asymptotic behavior of the solution for large values of the underlyings and also leads to consistent outputs with respect to *a priori* known qualitative properties of the solution. The performance and robustness with respect to the dimension of these methods are assessed in a series of numerical experiments involving the Merton jump-diffusion model, while a comparison with the deep Galerkin method and the deep BSDE solver with jumps further supports the merits of the proposed approach.

⋆ This research work was supported by the Hellenic Foundation for Research and Innovation (H.F.R.I.) under the "First Call for H.F.R.I. Research Projects to support Faculty members and Researchers and the procurement of high-cost research equipment grant" (Project Number: 2152). The authors also acknowledge the use of computational resources of the DelftBlue supercomputer, provided by the Delft High-Performance Computing Centre [1]. EHG also wishes to acknowledge the financial support of The Leverhulme Trust (grant number RPG-2021-238) and of EPSRC (grant number EP/W005840/2).

* Corresponding author.
E-mail addresses: e.georgoulis@hw.ac.uk (E.H. Georgoulis), a.papapantoleon@tudelft.nl (A. Papapantoleon), kesmarag@iacm.forth.gr (C. Smaragdakis).

https://doi.org/10.1016/j.cnsns.2026.109709
Received 16 June 2025; Received in revised form 6 November 2025; Accepted 7 January 2026
Available online 10 January 2026

1. Introduction

The numerical approximation of solutions to initial/boundary value problems involving partial differential equations (PDEs) in high dimensions remains a formidable challenge. A fast growing methodological shift in addressing this challenge is the use of machine learning frameworks such as Deep Galerkin Method [2], the Physics-Informed Neural Network methodology [3,4], and related methods involving the minimization of physical energies of the underlying PDE systems [5–7,7,8], etc. When the models can be posed as backward-forward stochastic control problems, fast methodologies such as see also the deep BSDE method [9] have been successfully applied. The situation is far less clear in the context of high dimensional partial integro-differential equations (PIDEs), whereby the presence of an additional integral term creates a number of new numerical challenges, compared to high-dimensional PDEs.

A central problem in Mathematical Finance is the fast and accurate computation of arbitrage-free prices of financial derivatives, especially for advanced stochastic models and for multi-asset derivatives. A basket option is a contractual agreement between two parties, the buyer and the seller, to buy or sell a derivative whose value fluctuates over time based on the prices of a set of underlying assets (the "basket"). In this work, we consider the problem of pricing European basket call options over d underlying assets using weights $\{\alpha_i\}_{i=1}^d$. The strike price, denoted by K, is the price at which the basket can be bought at the maturity of the option contract, *i.e.* at $t = T$. The payoff of a European basket call option is provided by

$$\left(\sum_{i=1}^{d} \alpha_i S_T^i - K\right)^+,$$

where $\alpha_i > 0$ and $\sum_{i=1}^d \alpha_i = 1$. The value S^i of each asset is associated to the variable $x_i = S_T^i/K$, referred to in the literature as the moneyness of the stock. The payoff function of a basket call option is then provided by

$$\text{Payoff}(x) = u_0(x) := \left(\sum_{i=1}^{d} \alpha_i x_i - 1\right)^+. \tag{1}$$

Models for asset prices that incorporate random jumps are essential in capturing the real-world behavior of financial markets. These models acknowledge that financial asset prices do not always move smoothly; instead, they may present abrupt changes due to unexpected events. Incorporating random jumps into asset price models helps to obtain a better description of the real-world behavior of the markets. Indeed, they allow to capture the fat-tails and skews present in asset log-returns under the "real-world" measure, while they exhibit a volatility smile or skew under the "risk-neutral" measure. We refer the reader to Eberlein and Kallsen [10], Cont and Tankov [11] or Schoutens [12] for more details and references on models with jumps in finance.

A popular model that incorporates random jumps is the, so-called, Merton [13] model, which belongs to the family of Lévy jump-diffusion models, *i.e.* jump-diffusion models with stationary and independent increments. In the Merton model, asset prices follow a superposition of a geometric Brownian motion and a Poisson process with randomly distributed jumps, allowing for both continuous diffusion, as in the Black–Scholes model, and discrete random jump movements, realised by a multivariate normal distribution for the jump size. The intensity and the magnitude of the jumps are determined by parameters that can be estimated from market data, *e.g.* option prices of single- and multi-asset options. The Merton model strikes a nice balance between superior fit of empirical data (compared to classical diffusion models) and relative simplicity (compared to other jump diffusion models), and allows us to showcase the potential of the numerical method developed in the present work. Other popular jump-diffusion models in the literature are the Kou [14] model and affine jump-diffusions, see *e.g.* Bates [15] or Duffie et al. [16]. Empirical evidence from option markets that verify the potential of jump-diffusion models appear, for example, in He et al. [17] and Kindermann and Mayer [18], and more recently in Chen and Huang [19] using data from bitcoin markets.

The industry standard for pricing European single-asset options in Lévy and affine jump-diffusion models are transform methods, such as Fourier or COS; see *e.g.* [20–22]. These methods, however, suffer from the curse of dimensionality and can typically not be applied for the valuation of basket options in dimensions higher than 8 or 10; see [23,24] for the state of the art in this direction.

An alternative and general method for pricing options in Lévy and affine models is to solve the associated second-order partial integro-differential equation (PIDE), where the initial (terminal) condition is determined by the payoff function (1) of the option. The PIDE is derived by the Fundamental Theorem of Asset Pricing and the Feynman–Kac lemma, which relates the discounted expectation of the payoff with an initial (terminal) value problem of the form described in the following section. Once again, finite difference and finite element discretizations for the solution of the option pricing PIDEs suffer from the curse of dimensionality and cannot be used in dimensions higher than 6 or 8; see *e.g.* Griebel and Hullmann [25], Hepperger [26] and Reichmann and Schwab [27].

In this work, we address the challenge of computing the fair prices of financial derivatives by discretizing the PIDE using a novel implicit-explicit minimizing movement approach involving Artificial Neural Networks (ANNs). The use of ANNs aims to address the curse of dimensionality typically encountered in standard grid-based methods, while simultaneously offering improved performance compared to standard or Quasi Monte Carlo approaches.

The classical minimizing movement method of De Giorgi, see Ambrosio [28], provides discretization of gradient flows in the calculus of variations, by considering a suitable minimization functional for each time step. This is an extension of the classical Ritz/Dirichlet minimization principle for elliptic, self-adjoint problems, which is considered in E and Yu [5], Liao and Ming [6] in the context of ANNs. In Georgoulis et al. [7], Park et al. [8] the energy minimization functional, duped "minimizing movement method" of De Giorgi is used to define ANN approaches for the numerical solution of Fokker-Planck type equations which arise as first order optimality conditions of certain energies, which are used as ANN cost functionals. A canonical example is the minimizing movement

method for the Dirichlet energy corresponding to the backward Euler time-stepping for the heat equation. In this work, we develop a significant extension of the "deep minimizing movement" methods from [7,8] to approximate PIDE problems, with specific focus on those arising from basket option pricing in jump-diffusion models. To achieve this, we have to address two new challenges compared to [7,8]: a) the presence of "skew-symmetric" first order differential operators, not naturally arising through energy minimization, and b) the presence of integral operators. We resolve these two challenges simultaneously by the introduction of implicit-explicit time-stepping and the introduction of reduced complexity quadratures. To that end, we develop a minimizing movement approach for the family of implicit-explicit versions of Backward Differentiation Formulae (BDF) methods, which include and generalize the implicit-explicit Euler scheme of [7,8], and further they are capable of including skew-symmetric/transport and integral terms, treated explicitly. This choice is made due to the $A(\alpha)$-stability properties of BDF methods, in conjunction with the favourable computational cost in this context, since the expensive (due to the ANN architecture) spatial operator is evaluated only once for each time-step. This reduces the computational cost of optimization significantly, when compared to "monolithic" space-time ANN discretizations, whereby the cost functionals minimize the complete space-time residuals directly. The introduction of time-stepping has two crucial advantages: a) the ANNs represent time instances of the solution and *not* the total space-time solution, (which can be a far stiffer problem numerically in many cases), and b) the training of the ANN for each time instance can benefit from the trained parameters of the previous time instance as starting points.

A further computational challenge stems from the complexity of the integral operator due to the jumps in the asset price process. To that end, we devise and compare two different approaches to address the curse of dimensionality there: (a) a sparse-grid Gauss–Hermite approximation following localised coordinate axes arising from singular value decompositions, and (b) an ANN-based high-dimensional special-purpose quadrature rule. Moreover, in order to improve the overall performance and accuracy of the method, we introduce a decomposition of the solution into two parts: the option price is expressed as the sum of a non-negative unknown component $w(t,x)$, termed in the literature the time value of the option, and a known lower-bound function $v(t,x)$, termed in the literature the intrinsic value of the option. Finally, a domain truncation method is introduced to enhance the accuracy of the numerical schemes. This involves projecting the option prices onto a bounded subset of $\mathbb{R}^d_+$ and efficiently approximating the solution for extreme moneyness values by exploiting estimates within this bounded domain.

The above numerical methodology is tested through a comprehensive series of numerical experiments, showing the competitiveness of the approach against Quasi Monte Carlo in both low and high dimensional settings. In addition, we provide a basic comparison of the proposed method against the popular deep Galerkin method (DGM) Sirignano and Spiliopoulos [2], the deep BSDE solvers with jumps Han et al. [9] and Gnoatto et al. [29], showcasing the competitiveness of the proposed methodology.

The remainder of this work is structured as follows. In Section 2, we introduce the PIDE for option pricing in jump-diffusion models, present the implicit-explicit minimizing movement method and discuss the decomposition of the solution into a lower bound and an unknown function, as well as the truncation of the domain and the extension of the solution to extreme cases. In Section 3, we describe the architecture of the neural network and the training procedure, and then present two methods for the efficient computation of the integral operator. In Section 4, we test the performance of the methods using the Merton model, in scenarios with 5 and 15 underlying assets. Finally, Section 5 concludes this work.

2. Option pricing in jump-diffusion models

2.1. An illustrative example: The Merton model

Let $(S^1,\dots,S^d)$ denote d financial assets, where each one follows the, so-called, Merton model, see Merton [13], whereby the dynamics of the ith stock at time t are provided by

$$S_t^{(i)} = S_0^{(i)} \exp\left(b_i t + \sigma_i W_t^{(i)} + \sum_{k=1}^{N_t} Z_k^{(i)} \right), \quad t \in \mathbb{T} = (0,T],\ i \in \mathbb{I} = \{1,\dots,d\}, \tag{2}$$

where $W^{(i)}$ denotes a standard Brownian motion, $\sigma_i > 0$ denotes the diffusion volatility, N denotes a Poisson process with parameter $\lambda > 0$, $T > 0$ is a finite time horizon, and $Z_k^{(i)}$ are random variables controlling the jump sizes. The random vector $(Z_k^{(1)},\dots,Z_k^{(d)})$ follows a multivariate normal distribution with mean μ_J and variance-covariance matrix Σ_J. Moreover, assuming that we are already working under an equivalent martingale measure, the drift parameter b_i is determined by the martingale condition, and equals

$$b_i = r - \frac{1}{2}\sigma_i^2 - \lambda\left(\exp\left\{ \mu_{J_i} + \frac{1}{2}\sigma_{J_i}^2 \right\} - 1 \right), \tag{3}$$

where r denotes the risk-free interest rate.

There are three mechanisms that generate dependence between the assets in this example. The assets can have correlated diffusion terms, as well as correlated jump sizes. Moreover, the assets share the timing of the jumps, as the jumps occur according to a *common* Poisson process. These dependencies have a direct impact on the arbitrage-free price of an option. Indeed, according to the Fundamental Theorem of Asset Pricing (FTAP) and using the Feynman–Kac formula, the arbitrage-free price of an option with payoff given by (1) satisfies the following PIDE:

$$\begin{aligned} &\frac{\partial u}{\partial t} - \frac{1}{2}\sum_{i,j=1}^{d} \sigma_i \rho_{ij} \sigma_j x_i x_j \frac{\partial^2 u}{\partial x_i \partial x_j} + \sum_{i=1}^{d} b_i x_i \frac{\partial u}{\partial x_i} + ru - I_\varphi[u] = 0 \\ &u(0,x) = u_0(x), \end{aligned} \tag{4}$$

for $t \in \mathbb{T}$ and $x \in [0,\infty)^d$, where ρ_{ij} denotes the diffusion correlation between assets i and j. The integral operator is provided by

$$I_\varphi[u] = \lambda \int_{\mathbb{R}^d} \big(u(t, xe^z) - u(t,x)\big)\varphi(\mathrm{d}z), \tag{5}$$

with $\varphi(\cdot)$ denoting the probability density function of the multivariate normal distribution with mean μ_J and variance-covariance matrix Σ_J. Obviously, $u(t,0) = 0$ and $I[u(t,0)] = 0$, for each $t > 0$. The initial condition in (4) is derived from the terminal condition of the classical PIDE for option pricing, *i.e.* from the payoff function in (1), by employing the change of variable $t = T - \cdot$.

2.2. General form of the PIDE for option pricing

Motivated by (4), we consider the following general problem: find the arbitrage-free price u of an option with payoff (1), that satisfies the PIDE

$$\begin{aligned} &\frac{\partial}{\partial t}u(t,x) + \mathcal{A}u(t,x) = 0, \quad (t,x) \in \mathbb{T} \times [0,\infty)^d \\ &\qquad u(0,x) = u_0(x), \quad x \in [0,\infty)^d, \end{aligned} \tag{6}$$

where the operator $\mathcal{A}$ belongs to the following family:

$$\mathcal{A}u(t,x) = -\sum_{i,j=1}^{d} a_{ij}(x)\frac{\partial^2 u}{\partial x_i \partial x_j} + \sum_{i=1}^{d} b_i(x)\frac{\partial u}{\partial x_i} + ru - I_\nu[u], \tag{7}$$

where $a_{ij}(x) = a_{ji}(x)$, for $i, j \in \mathbb{I}$, and

$$I_\nu[u] = \lambda \int_{\mathbb{R}^d} \big(u(t, xe^z) - u(t,x)\big)\nu(x, \mathrm{d}z), \tag{8}$$

with the integration performed over a finite jump measure ν. The coefficients a_{ij} and ν are to be derived from the diffusion and the jump terms of the stochastic processes governing the asset prices, respectively, while the drift term will be determined again by the martingale condition; see (3). This class of models includes Lévy jump-diffusions, such as the Kou [14] model, and affine jump-diffusions, see *e.g.* Bates [15] or Duffie et al. [16]; see also Runggaldier [30] for a general overview of jump-diffusion models in finance. In what follows, it is sufficient to assume that $a_{ij}, b_j \in W^{1,\infty}([0,\infty)^d)$, using standard Sobolev space notation. The initial condition in (6) has the meaning that the price of an option with maturity time $T = 0$ (present) is determined by its payoff function. (This, again, is the result of employing the change of variable $t = T - \cdot$.)

In order to design energy minimization-type cost functionals below, we rewrite the differential part of the PIDE operator in divergence form:

$$\mathcal{A}u = \mathcal{L}u + f[u]$$

with $\mathcal{L}, f$ given by

$$\mathcal{L}u = -\sum_{j=1}^{d}\frac{\partial}{\partial x_j}\left(\sum_{i=1}^{d} a_{ij}(x)\frac{\partial u}{\partial x_i}\right) + ru, \tag{9}$$

$$f[u] = \sum_{i=1}^{d}\left(b_i(x) + \sum_{j=1}^{d}\frac{\partial}{\partial x_j}a_{ij}(x)\right)\frac{\partial u}{\partial x_i} - I_\nu[u]. \tag{10}$$

In the decomposition above, $\mathcal{L}$ is a self-adjoint operator, while the remainder term $f[\cdot]$ comprises of both symmetric and non-symmetric components; in particular, the second part of the integral operator, $\int_{\mathbb{R}^d} u(t,x)\nu(x,\mathrm{d}z)$, is symmetric. We chose to retain this in the remainder term for stability reasons of the proposed numerical framework below.

2.3. Implicit-explicit minimizing movement method

We will exploit now the divergence form of the PIDE in order to determine a minimizing movement approach, from which a respective cost functional will arise.

We will first describe the method for the PIDE defined on $\mathbb{T} \times \Omega$ with $\mathbb{T} = (0,T]$ and $\Omega = [0, x_{\max}]^d$, $x_{\max} > 0$. In the next sections, we will also discuss specific modelling issues related to option pricing problems, which are, in turn, incorporated in order to extend the solution domain of the method. Let us, for the moment, prescribe homogeneous Dirichlet boundary conditions on $\partial\Omega$; this will be revisited in the next subsection.

We consider a subdivision of the time interval $(0,T]$ into n equally spaced time intervals $(t_{k-1}, t_k]$ with $t_k = k\tau,\ k = 0, 1, \ldots, n$, for $\tau = T/n$. Let $u^0 := u_0(\cdot)$, then we seek approximations $u^k \approx u(t_k, \cdot)$ by involving values of the p previous time steps.

In this context, we consider implicit-explicit backward differentiation formulae (BDF); we refer to Akrivis et al. [31] for their numerical analysis. More specifically, for given parameter sets β_j, γ_j, we consider the time-stepping methods

$$\frac{\beta_p u^k - \sum_{j=0}^{p-1}\beta_j u^{k-j-1}}{\tau} + \mathcal{L}u^k + \sum_{j=0}^{p-1}\gamma_j f[u^{k-j-1}] = 0, \tag{11}$$

for $k = p, p+1, \dots, n$, along with special treatment of the first p timesteps (*e.g.*, by another, perhaps one-step, method). In particular, for $p = 1$ we get the implicit-explicit Euler scheme

$$\frac{u^k - u^{k-1}}{\tau} + \mathcal{L}u^k + f[u^{k-1}] = 0, \tag{12}$$

and for $p = 2$ the BDF-2 scheme

$$\frac{\frac{3}{2}u^k - 2u^{k-1} + \frac{1}{2}u^{k-2}}{\tau} + \mathcal{L}u^k + 2f[u^{k-1}] - f[u^{k-2}] = 0. \tag{13}$$

A crucial observation, following from the minimizing movement point of view, is that (11) (upon multiplication by τ and integration by parts) is the Euler–Lagrange equation of the convex minimization problem:

$$C[u] := \frac{1}{2}\Big\|\beta_p u - \sum_{j=0}^{p-1}\beta_j u^{k-j-1}\Big\|^2_{L^2(\Omega)} + \tau\int_\Omega \mathcal{E}[u]\mathrm{d}x + \tau\sum_{j=0}^{p-1}\gamma_j\int_\Omega f[u^{k-j-1}]u\,\mathrm{x} \to \min, \tag{14}$$

whereby

$$\mathcal{E}[u] := \frac{1}{2}\sum_{i,j=1}^{d}\Big(\alpha_{ij}(x)\frac{\partial u}{\partial x_i}\frac{\partial u}{\partial x_j} + ru^2\Big),$$

denotes the respective Dirichlet energy.

2.4. Decomposition of the solution

In the above discussion, we assumed, for simplicity, homogeneous Dirichlet boundary conditions on the domain of the truncated boundary $\partial\Omega$. In the option pricing context, however, one expects non-zero values of u as $|x| \to \infty$. Fortunately, the modelling asserts that these values of the solution u as $|x| \to \infty$ are asymptotically known. In particular, the value of the option at time $t \geq 0$ can be expressed as

$$u(t,x) = w(t,x) + v(t,x), \tag{15}$$

whereby $w(t,x) \geq 0$ denotes the time value of the option, while $v(t,x)$ denotes the intrinsic value of the option, given by

$$u(t,x) \geq \Big(\sum_{i=1}^{d}\alpha_i x_i - \mathrm{e}^{-rt}\Big)^+ = u_0(x) + (1-\mathrm{e}^{-rt})H\Big(\sum_{i=1}^{d}\alpha_i x_i - \mathrm{e}^{-rt}\Big) =: v(t,x), \tag{16}$$

where $H(\cdot)$ is the Heaviside function.

Crucially, for large values of the moneyness, *i.e.* of $\sum_{i=1}^d \alpha_i x_i$, the price of the option illustrates an asymptotic behavior

$$\lim_{\sum a_i x_i \to \infty} u(t,x) = \Big(\sum_{i=1}^{d}\alpha_i x_i - \mathrm{e}^{-rt}\Big)^+ = \sum_{i=1}^{d}\alpha_i x_i - \mathrm{e}^{-rt}. \tag{17}$$

Therefore, the decomposition (15) implies that $\lim_{\sum a_i x_i \to \infty} w(t,x) = 0$, which justifies the use of homogeneous Dirichlet boundary conditions on the domain $\Omega = [0, x_{\max}]^d$ for $x_{\max}$ large enough, if we solve for w instead.

On a technical note, in order to avoid any issues due to the lack of smoothness of the Heaviside function, we mollify the lower-bound term by setting

$$\tilde{v}(t,x;\eta) = \Big(\sum_{i=1}^{d}\alpha_i x_i - 1\Big)^+ + (1-\mathrm{e}^{-rt})\mathrm{Sigmoid}\Big(\sum_{i=1}^{d}\alpha_i x_i - \mathrm{e}^{-rt};\eta\Big), \tag{18}$$

with $\mathrm{Sigmoid}(x;\eta) := \big(1+\mathrm{e}^{-\eta x}\big)^{-1}$, $\eta > 0$. Obviously, $\lim_{\eta\to\infty}\tilde{v}(t,x;\eta) = v(t,x)$. Moreover, the parameter η influences the smoothness of the gradient of $\tilde{v}(t,x;\eta)$ with respect to x. In particular, smaller values of η result in smoother approximations. Hence, the parameter η should be chosen appropriately, to strike a balance between the precision of the lower bound approximation and the desired level of smoothness.

2.5. Domain truncation

Motivated by the discussion above, we now provide a framework for the truncation of the spatial domain and the extension to values in $\mathbb{R}^d_+$ that reflects the natural setting of the problem. To that end, recall (17) and observe that

$$\Big(\sum_{i=1}^{d}\alpha_i x_i - \mathrm{e}^{-rt}\Big)^+ \quad \text{is parallel to} \quad \sum_{i=1}^{d}\alpha_i x_i.$$

Consequently, for x' associated with a higher moneyness level than x, we may establish a connection between the option prices inside the truncated domain and those of "far away" values using the following expression:

$$u(t,x') \approx u(t,x) + \sum_{i=1}^{d}\alpha_i(x'_i - x_i), \tag{19}$$

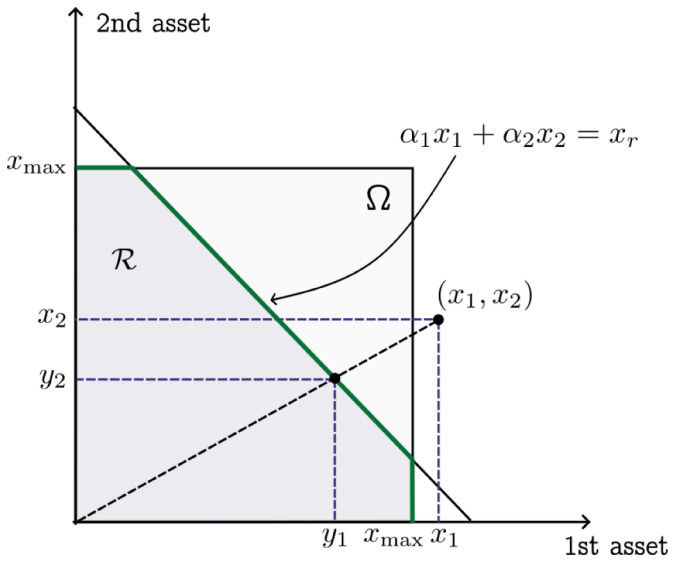


Fig. 1. A two-dimensional example for the domain truncation. The green boundary defines the projection surface for the option price in extreme moneynesses.

for $\sum_{i=1}^{d} a_i x_i$ sufficiently large.

Let $\mathcal{R}$ denote a subset of $\mathbb{R}_+^d$ such that $\sum_{i=1}^{d} \alpha_i x_i < x_r$ and $x_i \le x_{\max}$, $i \in \mathbb{I}$, for a given $x_r \le x_{\max}$. The boundary $\partial\mathcal{R}$ of $\mathcal{R}$ consists of the points which satisfy $\sum_i \alpha_i x_i = x_r$ and $x_i \le x_{\max}$ for $i \in \mathbb{I}$.

We define the projection $y = (y_1, y_2, \dots, y_d)$ of $x \in \mathbb{R}_+^d$ as $y_i = q(x)x_i$, with

$$q(x) = \begin{cases} x_{\max}/\max\{x_i\}, & \text{if } \max\{x_i\} \ge \max\left(\sum_{i=1}^{d} \alpha_i x_i, x_r\right) x_{\max}/x_r \\ x_r/\max\left(\sum_{i=1}^{d} \alpha_i x_i, x_r\right), & \text{otherwise.} \end{cases} \tag{20}$$

Clearly, $0 \le q(x) \le 1$ for all $x \in \mathbb{R}_+^d$ and $q(x)x \in \mathcal{R}$. Employing the approximation (19), we introduce a formula that expresses the option prices for each $x \in \mathbb{R}_+^d$ in relation to the option price within the bounded domain $\mathcal{R}$:

$$u(t, x) \approx u(t, y(x)) + \sum_{i=1}^{d} \alpha_i (x - y(x)), \quad \text{for each } x \in \mathbb{R}_+^d. \tag{21}$$

Fig. 1 illustrates the proposed domain truncation in a two-asset scenario. In this example, x has a higher moneyness than x_r, resulting in $y = (y_1, y_2)$ of moneyness x_r, by following the projection procedure.

The significance of the domain truncation lies in getting estimates of the option price for large values of the moneyness, even within the confines of a solution in a bounded domain. This is crucial considering the random discontinuities in asset prices that we have taken into account.

3. Deep implicit-explicit minimizing movements and network architecture

In this section, we detail the representation of the solution through a deep artificial neural network (ANN) and discuss the network parameter optimization process. A key attribute of the methodology presented below is that the approximate solution U^k is computed at each timestep t_k by a deep ANN with a specific architecture; this has been proposed, for instance, by Georgoulis et al. [7], see also the deep BSDE method of Han et al. [9], and is in contrast to other popular ANN-type methods for PDEs, such as the deep Galerkin method (DGM) of Sirignano and Spiliopoulos [2] or the physics-informed neural networks (PINNs) by Lagaris et al. [3], Raissi et al. [4], whereby a single ANN approximating the solution over a space-time cylinder is employed. A practical advantage of using a single deep ANN architecture for each time instance t_k is that the approximation is computed by retraining the same architecture from one timestep to the next, thereby reducing the training time due to the availability of good "starting" parameter values, *i.e.*, the values from the previous time-step.

3.1. Network architecture

The approximate solution U^k, for each timestep t_k, is represented by a modified version of the deep ANN of residual type by Sirignano and Spiliopoulos [2]; the architecture of each layer follows [2] and, for this reason, is henceforth designated as "DGM

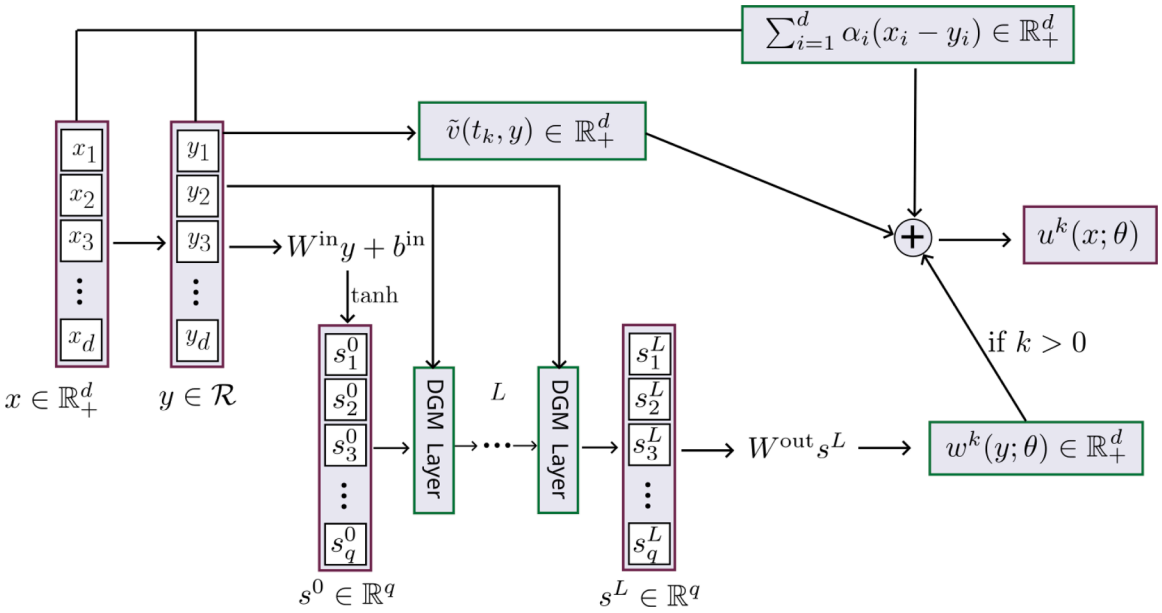


Fig. 2. Flowchart of the deep neural network modeling the solution of the PIDE for a time instance $t = t_k$.

layer". The network implements the decomposition and boundary modeling described in the previous subsections. More specifically, for an input y, we set

$$
\begin{aligned}
S^0 &= \tanh(W^{\text{in}} y + b^{\text{in}}),\\
&\text{DGM layer}\\
&\quad G^\ell = \tanh(V^{g,\ell} y + W^{g,\ell} S^{\ell-1} + b^{g,\ell}),\ \ell = 1,\dots,L\\
&\quad Z^\ell = \tanh(V^{z,\ell} y + W^{z,\ell} S^{\ell-1} + b^{z,\ell}),\ \ell = 1,\dots,L\\
&\quad R^\ell = \tanh(V^{r,\ell} y + W^{r,\ell} S^{\ell-1} + b^{r,\ell}),\ \ell = 1,\dots,L\\
&\quad H^\ell = \tanh(V^{h,\ell} y + W^{h,\ell} (S^{\ell-1} \odot R^\ell) + b^{h,\ell}),\ \ell = 1,\dots,L\\
&\quad S^\ell = (1 - G^\ell) \odot H^\ell + Z^\ell \odot S^{\ell-1},\ \ell = 1,\dots,L\\
\tilde{v}(t_k, y) &= \left(\sum_{i=1}^{d} \alpha_i y_i - 1\right)^+ + (1 - \mathrm{e}^{-rt_k})\ \text{Sigmoid}\Big(\sum_{i=1}^{d} \alpha_i y_i - \mathrm{e}^{-rt_k}; \eta\Big),\\
w^k(y;\theta) &= \text{Softplus}\big(W^{\text{out}} S^L; \delta\big),
\end{aligned}
$$

with L denoting the number of hidden layers and $\odot$ denoting the Hadamard product. The trainable parameters θ of the model are

$$\theta = \{W^{\text{in}}, b^{\text{in}}, (V^{*,\ell}, W^{*,\ell}, b^{*,\ell})_{\ell=1,\dots,L}^{*\in\{g,z,r,h\}}, W^{\text{out}}\}, \tag{22}$$

and the output of the network is given by

$$U(t_k, x; \theta) = w^k(y;\theta) + \tilde{v}(t_k, y) + \sum_{i=1}^{d} \alpha_i (x_i - y_i), \tag{23}$$

recalling the definition of $y \equiv y(x)$ (and of the y_i's) in (20). Fig. 2 shows a flowchart of the adopted 'global' ANN architecture, while Fig. 3 presents the architecture of a single DGM layer.

The choice of the architecture appears to be of significance, as it performs consistently better than a standard fully connected network in our experiments: the residual connections allow the network to learn incremental updates to the identity function, stabilising the training process for the networks required in our application. Our selected configuration, consisting of two residual layers with a node width of 64.

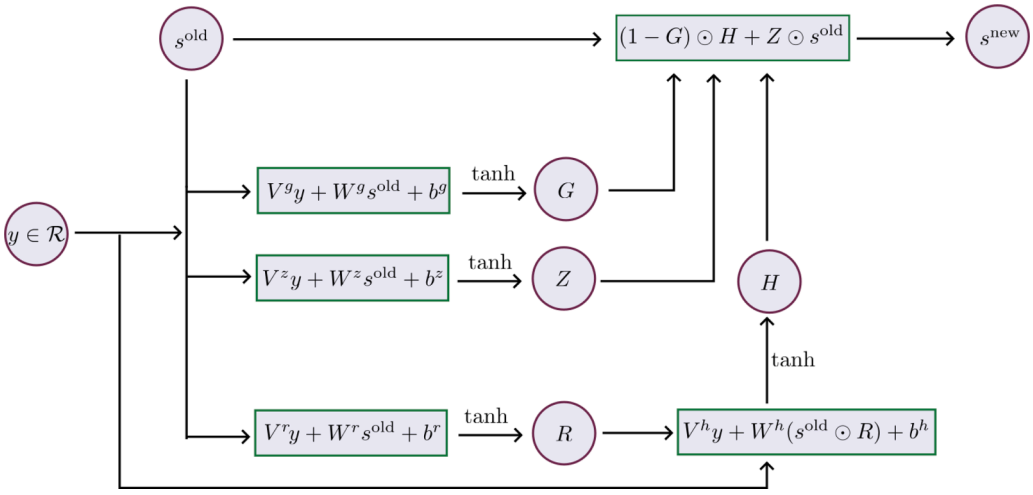


Fig. 3. The architecture of a DGM layer.

The computation of the parameters θ which provide the approximate solution at t_k is performed using (14), *i.e.*, we seek θ minimizing the cost functional

$$\begin{aligned}\mathscr{C}_k(\theta) := C[U(t_k,\cdot,\theta)] = \frac{1}{2}\Big\|\beta_p U(t_k,\cdot,\theta) - \sum_{j=0}^{p-1}\beta_j U(t_{j(k)},\cdot,\theta^{j(k)})\Big\|^2_{L^2(\Omega)} \\ + \tau\int_\Omega \mathcal{E}[U(t_k,x,\theta)]\,\underset{\sim}{\mathrm{x}} + \tau\sum_{j=0}^{p-1}\gamma_j\int_\Omega f[U(t_{j(k)},x,\theta^{j(k)})]U(t_k,x,\theta)\,\underset{\sim}{\mathrm{x}},\end{aligned} \tag{24}$$

using the notation $j(k) := k-j-1$ for brevity. Let us point out that, due to the network architecture, although C is convex with respect to its argument, $\mathscr{C}$ is not, in general. The minimizer is denoted by θ^k. Due to the multistep nature of the BDF methods, we observe that θ^k also depends on $\theta^{k-1},\dots,\theta^{k-p}$. As discussed above, special care has to be taken for the first p time steps (*e.g.*, by employing an one-step method) to initiate the iteration.

Remark 1. We note the practical significance of mollifying the solution lower bound as per (18). Indeed, $v(t_{j(k)},\cdot)$ is contained in $U(t_{j(k)},\cdot,\theta^{j(k)})$ and it is differentiated in space within $\mathscr{C}$.

3.2. Training

As is typical in ANN-based methods for PDEs, the cost function (24) is discretized by a Monte Carlo sampling procedure. In particular, considering N uniformly sampled points $\{x^i\}_{i=1}^N$ in $\Omega=[0,x_{\max}]^d$, the discretized cost functional is given by

$$\widetilde{\mathscr{C}_k}(\theta) := \frac{(x_{\max})^d}{N}\sum_{i=1}^N\left\{\frac{1}{2}\Big[\beta_p U(t_k,x^i,\theta) - \sum_{j=0}^{p-1}\beta_j U(t_{j(k)},x^i,\theta^{j(k)})\Big]^2 + \tau\mathcal{E}[U(t_k,x^i,\theta)] + \tau\sum_{j=0}^{p-1}\gamma_j f[U(t_{j(k)},x^i,\theta^{j(k)})]U(t_k,x^i,\theta)\right\}, \tag{25}$$

using again $j(k) := k-j-1$ for brevity. For notational simplicity, we still denote by θ^k the computed minimizer of $\widetilde{\mathscr{C}_k}(\theta)$. In practice, we update the model parameters using the ADAM optimizer for N_k training epochs and we denote by θ^k the result of the optimization (even if it may not have converged).

3.2.1. Initialization

The initial condition (6) implies that $w^0(x;\theta^0)=0$ for all $x\in[0,x_r]^d$, which may result in practical difficulties in the training process, since it leads to vanishing weights. In order to address this, we use the mathematically equivalent form

$$U(t_k,x;\theta) = (1-\delta_{k0})w^k(y;\theta) + \tilde{v}(t_k,y) + \sum_{i=1}^d \alpha_i(x_i-y_i),$$

with δ_{ij} being the Kronecker delta. In other words, we impose $w^0(x;\theta^0)=0$ strongly at $t=0$, resulting in the network weights becoming independent of the initial condition of the PIDE.

Nevertheless, it is of interest to initialize w^0 in order to get a good starting point for adapting the network for approximating the solution at t_1 and beyond. In that respect, we employ the following smooth function that goes to zero for small and large asset prices:

$$f^0(x) = \epsilon\exp\left[-\frac{1}{2\zeta^2(x)}\left(\sum_{i=1}^d\alpha_i x_i - 1\right)^2\right],$$

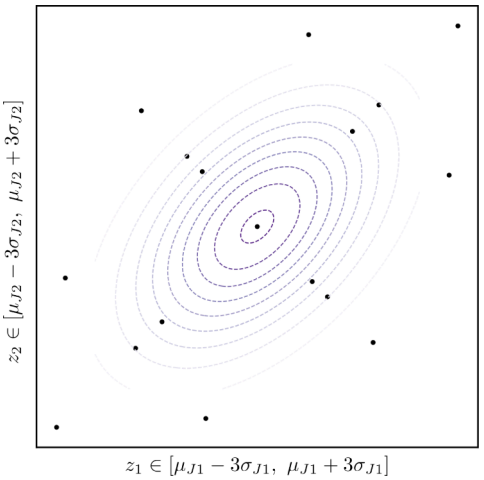


Fig. 4. Sparse sampling in $\mathbb{R}^2$ using the Gauss–Hermite quadrature for normally distributed random variables with correlation equal to $\frac{1}{2}$.

for $\epsilon > 0$ small, with $\zeta(x)$ taking two possible positive values ζ_1, ζ_2 according to the criterion $\sum_{i=1}^{d} \alpha_i x_i > 1$ and we compute θ^0 such that

$$w^0(x;\theta^0) \to \min_{\theta} \left\| w^0(x;\theta) - f^0(x) \right\|^2_{L^2([0,x_r]^d)}.$$

Once computed, we employ $w^0(x;\theta^0)$ in the BDF iteration.

3.3. Computation of the integral operator

Each evaluation of the cost (loss) function requires the numerical calculation of the integral term $I_\nu[u]$ in $f[u]$. Approximating the integral operator I_ν requires special treatment due to the complexity implied by this operator. To that end, we calculate the integral operator using two methods. The first one employs a sparse-grid integration technique based on the Gauss–Hermite quadrature, while the second one involves training a specialized neural network for computing the integral values. The latter method is more time-efficient, whereas the former gives, in general, more accurate results.

The integral term appears in an explicit fashion with respect to the timestepping, therefore we are required to evaluate the values $I_\nu[U(t_{j(k)}, x, \theta^{j(k)})]$ at x for known ANN solution approximations $U(t_{j(k)}, x, \theta^{j(k)})$, $j = 0, \dots, p-1$, again for $j(k) = k - j - 1$.

3.3.1. Sparse Gauss–Hermite quadrature

In the Merton model, the integral can be expressed as the expected value, with respect to a Gaussian measure, of the function $h(z) = u(x\mathrm{e}^z) - u(x)$ multiplied by the Poisson parameter λ; for brevity, for the remainder of this discussion we suppress the dependence on t, and u signifies any of the $U(t_{j(k)}, x, \theta^{j(k)})$ for $j = 0, \dots, p-1$. In addition, the amplitude of the random jumps is determined by a multivariate normal random variable, $Z \sim \mathcal{N}(\mu_J, \Sigma_J)$. Therefore, Σ_J introduces directional features.

The construction of a d-dimensional quadrature rule employing sparse grid interpolation of the integrand, benefits by following the "natural direction" of Σ_J. To that end, we first perform the Singular Value Decomposition (SVD) of the covariance matrix to express the integration in terms of the Gauss–Hermite quadrature by modulating and rotating the original axes, viz.,

$$\Sigma_J = A\Lambda A^{\mathsf{T}} = A\Lambda^{1/2}\Lambda^{1/2}A^{\mathsf{T}} = BB^{\mathsf{T}}, \tag{26}$$

where $B = A\Lambda^{1/2}$. The change of variables $Z - \mu = \sqrt{2}BY$ transforms the integrand such that the multidimensional Gauss–Hermite quadrature can produce a sparse grid of representative points. More specifically, we have

$$\begin{aligned} \int_{\mathbb{R}^d} h(z)\nu(\mathrm{d}z) &= \lambda\pi^{-d/2} \int_{\mathbb{R}^d} h(\mu + \sqrt{2}By)\exp(-y^{\mathsf{T}}y)\mathrm{d}y \\ &\approx \lambda\pi^{-d/2} \sum_{\mathbf{i}\in\Theta_q} h(\mu + \sqrt{2}By^{\mathbf{i}})W^{\mathbf{i}}, \end{aligned} \tag{27}$$

with $y^{\mathbf{i}} = (y^{i_1}, \dots, y^{i_d})^{\mathsf{T}}$ representing a node of the sparse grid, and $W^{\mathbf{i}} = \prod_{k=1}^{d} w^{i_k}$ the associated weight for a multi-index $\mathbf{i} = (i_1, \dots, i_d)$. Here, y^{i_k} are chosen to be the roots of Hermite polynomials, and the index space $\Theta_q \subset \mathbb{N}^d$ is selected in such a way that the resulting quadrature exactly integrates polynomials up to a desired degree q; *cf.* Bungartz and Griebel [32]. Fig. 4 presents an example of a sparse grid of points in $\mathbb{R}^2$ for integrating a function of two correlated normally distributed random variables.

3.3.2. Approximation of the integral using an ANN

We further develop an alternative approach for the evaluation of the integral operator, by introducing a second artificial neural network for this purpose. The introduction of a second ANN is inspired by Gnoatto et al. [29], however, we introduce a neural network that directly learns the values of the integral $\int h(z)\nu(\mathrm{d}z)$, while in [29] they learn the values of a martingale stemming from stochastically integrating h with respect to the compensated Poisson measure of jumps.

The neural network we now consider will serve as an estimator of the values of the integral $\int h(z)\nu(\mathrm{d}z)$ with $h(z) = u(x\mathrm{e}^z) - u(x)$ for each value of x. The approximate neural network value at time t_k is denoted by $\mathcal{I}^k(x;\phi)$, where ϕ is the set of trainable parameters. In order to determine ϕ^k, we solve the optimization problem

$$\min_{\phi\in\Phi} \mathbb{E}\left[\mathcal{I}^k(x;\phi) - \sum_{j=1}^{p-1} \gamma_j I_\nu[U(t_{j(k)}, x; \theta^{j(k)})]\right]^2, \tag{28}$$

with the expectation taken over x.

As usual, we discretize the integral in (28) by Monte Carlo sampling before performing the optimization, *i.e.*, we approximate an unbiased and minimum variance estimator of the integral operator with respect to x. To that end, setting $h_{j(k)}(x,z) := U(t_{j(k)}, x\mathrm{e}^z, \theta^{j(k)}) - U(t_{j(k)}, x, \theta^{j(k)})$, we consider normally distributed points $\{z^r\}_{r=1}^M$ in $\mathbb{R}^d$ according to the distribution that governs the size of the jumps, and seek to find the optimizer $\phi^k \in \Phi$ that minimizes

$$\min_{\phi\in\Phi} \mathbb{E}\left[\mathcal{I}^k(x;\phi) - \frac{\lambda}{M}\sum_{r=1}^{M}\sum_{j=1}^{p-1} \gamma_j h_{j(k)}(x, z^r)\right]^2; \tag{29}$$

again, the expectation is over x. At the time step t^k and given a set of uniformly distributed points $\{x^i\}_{i=1}^N$, we obtain a discrete approximation of (29):

$$\frac{(x_{\max})^d}{N}\sum_{i=1}^{N}\left[\mathcal{I}^k(x^i;\phi) - \frac{\lambda}{M}\sum_{r=1}^{M}\sum_{j=1}^{p-1} \gamma_j h_{j(k)}(x^i, z^r)\right]^2. \tag{30}$$

Concluding, in order to estimate the solution U^k of the PIDE at time t^k, we need to determine the values of the integral operator. To approximate these values, we employ the supplementary ANN.

At each time step, we begin by optimizing the parameters ϕ^k of the ANN for the integral by minimizing (30), followed by the independent optimization for θ^k based on (25). Fortunately, the training associated with (30) is computationally inexpensive, introducing no significant overhead to the overall scheme.

4. Numerical examples

We shall now test the performance of the proposed methodology by pricing a European basket call option in the Merton model. Also, we provide a simple comparison with two popular and successful machine learning-driven solvers. More specifically, we consider a basket of d equally weighted assets with moneyness values $x_1, x_2, \ldots, x_d$. The model parameters are chosen to be the following:

$$\sigma_i = 0.5,\ \rho_{ij} = \delta_{ij} + 0.5(1-\delta_{ij}),\quad i,j \in \mathbb{I},\ t \in \mathbb{T} = (0,T],$$

where δ_{ij} denotes the Kronecker delta again. Moreover, the parameters of the jump distribution are

$$\lambda = 1,\ \mu_{J_i} = 0,\ \sigma_{J_i} = 0.5,\ \rho_{J_{ij}} = \delta_{ij} + 0.2(1-\delta_{ij}),\ i,j \in \mathbb{I}.$$

Note that the elements of the covariance matrices are given in terms of the standard deviations and correlations as follows:

$$\Sigma_{ij} = \sigma_i\sigma_j\rho_{ij} \quad \text{and} \quad \Sigma_{J_{ij}} = \sigma_{J_i}\sigma_{J_j}\rho_{J_{ij}}.$$

Let us point out that we have intentionally selected large values for the volatilities and asset correlations, in order to test the performance of the method in challenging scenarios. Below, we present numerical experiments for dimensions $d = 5$ and $d = 15$.

4.1. 5 correlated assets – Integration using Gauss–Hermite quadrature

The first numerical example concerns the valuation of a European basket call option consisting of 5 correlated assets. The maturity is chosen to be in one year, *i.e.* $T = 1$. Initially, we use the Gauss–Hermite quadrature to compute the integral operator. Results for both the implicit-explicit Euler and the BDF-2 schemes are provided. For the Euler scheme, we consider a time-step of $\tau = 0.01$, while the BDF-2 scheme uses a larger time-step of $\tau = 0.04$. We present the option prices for moneyness values in the interval $[0,3]$, considering the scenario where all assets share the same moneyness. Fig. 5 corresponds to the implicit-explicit Euler scheme, and Fig. 6 to the BDF-2 scheme. For comparison reasons, we also provide an extended Quasi Monte Carlo (QMC) estimation of the solution. We can observe that both methods work very well compared to the QMC simulation, and the difference to the Quasi Monte Carlo is only visible for deep in-the-money options, with moneyness level above 2. Indeed, the absolute error is on the order of 10^{-3}. The implicit-explicit Euler method, which run for 192 min, appears slightly more accurate than the BDF-2 scheme, due to the 4 times smaller time-step used. As expected, the computational cost of the implicit-explicit Euler method is greater than BDF-2, being approximately 1.8 times slower.

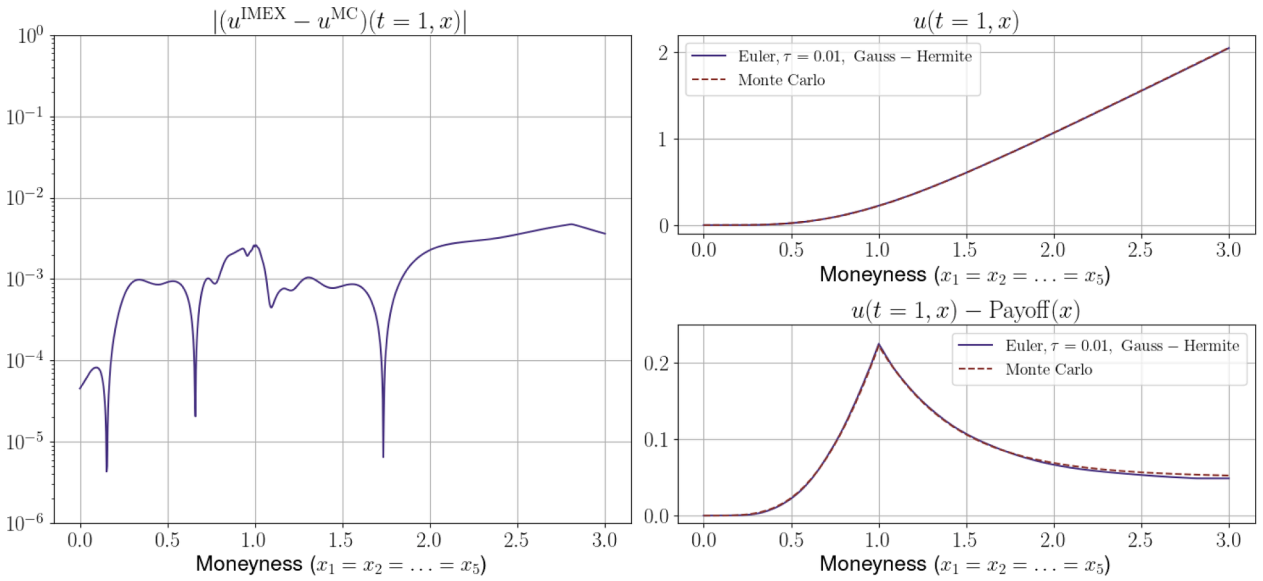


Fig. 5. Basket option prices (top right), differences between price and payoff (bottom right) and differences between the proposed method and quasi Monte Carlo (left), using the implicit-explicit Euler scheme and the Gauss–Hermite quadrature for the integral, for $d = 5$.

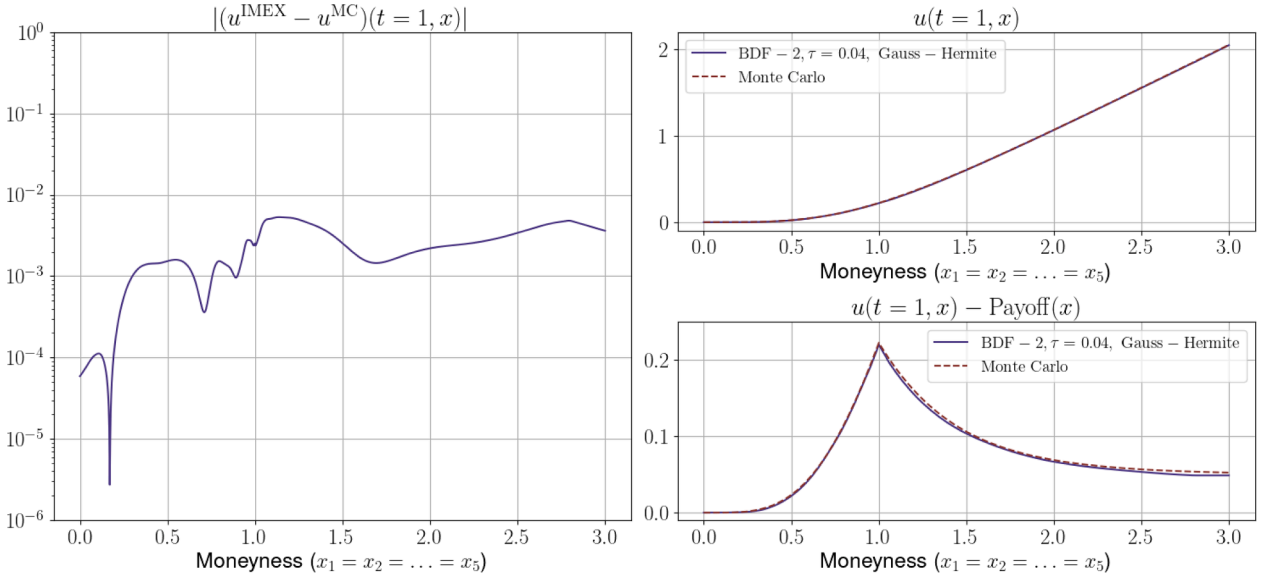


Fig. 6. Basket option prices (top right), differences between price and payoff (bottom right) and differences between the proposed method and quasi Monte Carlo (left), using the BDF-2 scheme and the Gauss–Hermite quadrature for the integral, for $d = 5$.

4.2. 5 correlated assets – Integration using an ANN

Next, we present the corresponding results using the second ANN instead of the Gauss–Hermite quadrature to estimate the integral operator. Figs. 7 and 8 depict the recovered option prices for the implicit-explicit Euler and the BDF-2 schemes, respectively. Again, we compare the results using a greedy Quasi Monte Carlo simulation. We can observe that the ANN computation of the integral operator also works very well, and the absolute error remains in the order of 10^{-3}. The ANN method though results in a computation of basket options prices that is roughly 1.7 times faster (113 min) than the corresponding method with the Gauss–Hermite quadrature.

Fig. 9 illustrates a comparison of the two approaches employed for the computation of the integral operator in the case of the 5-asset European basket option for moneynesses in $[0, 3]$ at $t = 0.01$. We can see that the two methods yield similar approximations of the integral operator. Let us point out that the Gauss–Hermite quadrature tends to outperform the ANN for extreme moneyness values (small or large) due to its sampling-independent nature. More specifically, the quadrature captures the asymptotic behavior

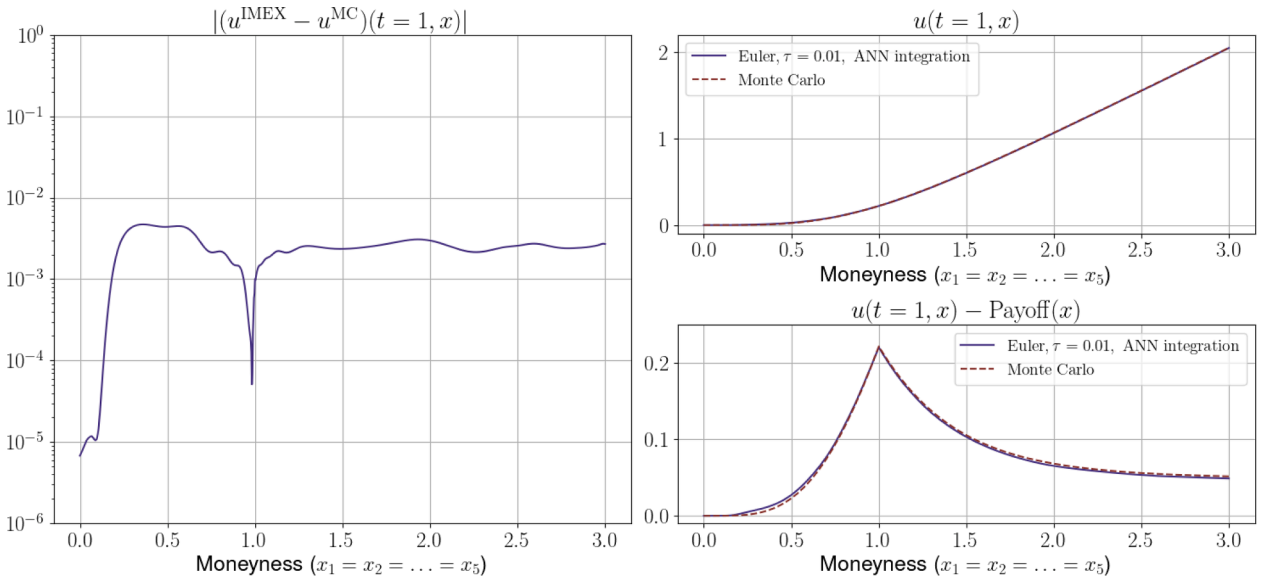


Fig. 7. Basket option prices (top right), differences between price and payoff (bottom right) and differences between the proposed method and quasi Monte Carlo (left), using the implicit-explicit Euler scheme and an ANN for the calculation of the integral, for $d = 5$.

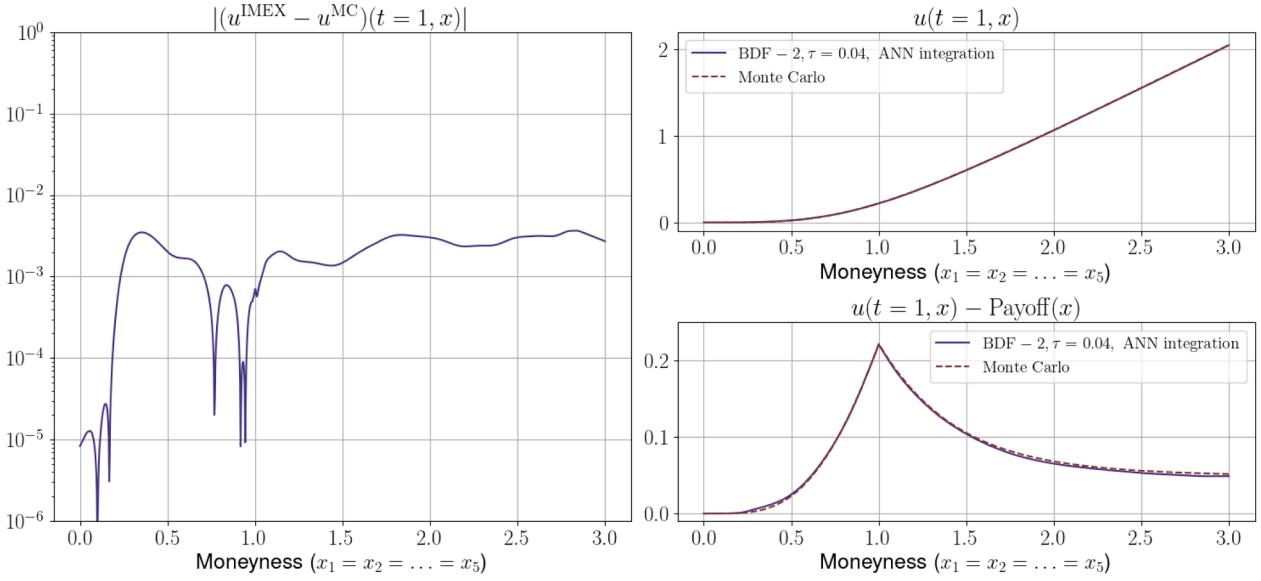


Fig. 8. Basket option prices (top right), differences between price and payoff (bottom right) and differences between the proposed method and quasi Monte Carlo (left), using the BDF-2 scheme and an ANN for the calculation of the integral operator, for $d = 5$.

of the integral for large x, aligning with the expected jump sizes, *i.e.*: $\lim_{x\to\infty} I_\varphi(x) = x\mathbb{E}[\mathrm{e}^z - 1]$. On the other hand, the ANN-based approach tends to yield smoother approximations, particularly for commonly sampled moneyness levels.

In addition, the performance of each of the two integration methods proposed emerges as a key performance factor, since integrations amounts to a significant percentage of the overall cost per time-step. To that end, we compare the average duration per iteration and the relative absolute difference between the methods for $d = 2, 5$, and 8. The results are presented in Table 1.

The ANN-based approach performs significantly faster across all the tested dimensions, without effective detrimental effect in the accuracy. More importantly, its computational time scales nicely with d. On the other hand, the runtime of the Gauss–Hermite quadrature grows significantly with dimension, which indicates that it is not immune to the curse of dimensionality. In conclusion, while Gauss–Hermite quadrature provides a robust and reliable approach for low-dimensional problems, its computational burden rapidly becomes prohibitive when increasing spatial dimension above a certain threshold. The ANN-based quadrature approach, appears to admit better scalability with respect to spatial dimension. Thus, the latter is the only feasible option for high-dimensional problems, with a practical dimensional limit for the Gauss–Hermite quadrature appearing around $d = 8$ in our test cases.

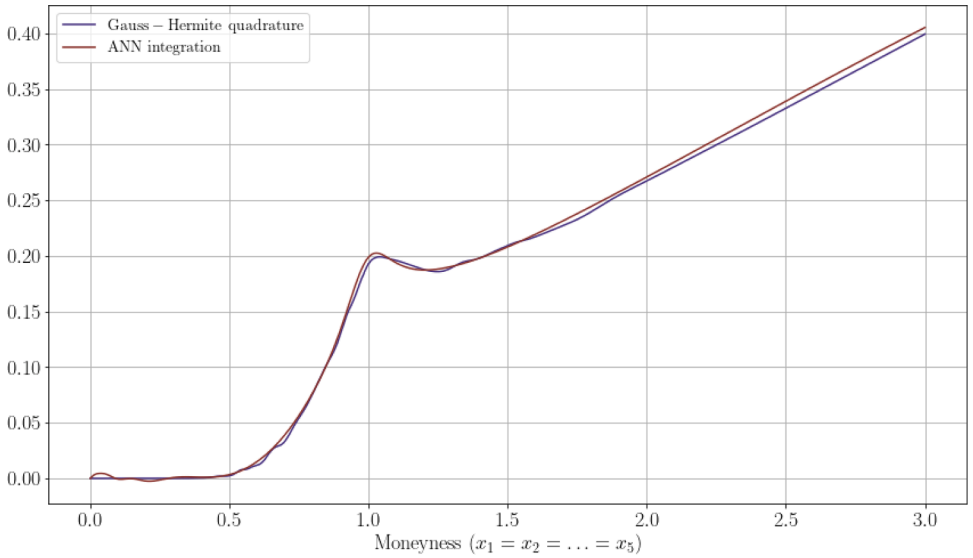


Fig. 9. Estimates of the integral operator for $t = 0.01$.

Table 1
Computational cost and relative absolute differences between the two integration methods across the various dimensions.

d	Runtime per Iteration (sec)		Rel. Abs. Diff.
	Gauss–Hermite	ANN-based	
2	0.0746	0.0100	$2.63E{-}2$
5	0.1223	0.0238	$9.62E{-}2$
8	1.0192	0.0388	$1.45E{-}1$

4.3. 15 correlated assets – Integration using an ANN

We further test our numerical methodology in a rather challenging high-dimensional option pricing scenario. In particular, we consider the case of a 15-asset European basket call option, as described earlier, with the maturity time being one year, *i.e.* $T = 1$. We employ time steps of $\tau = 0.01$ for the implicit-explicit Euler method and $\tau = 0.03$ for the BDF-2 scheme, respectively. Fig. 10 presents the pricing results for the implicit-explicit Euler scheme, whereas Fig. 11 provides the corresponding results for the BDF-2 scheme. Both methods are performing very well once again, and the important observation is that the absolute error remains relative small, although the dimension of the problem is now three times higher. The BDF-2 scheme with triple size timestep is 1.7 times faster than the implicit-explicit Euler method.

4.4. Hyperparameters and performance

We report the hyperparameters, the computational setup, and the computational run times of the implementation. The neural networks used consist of 2 DGM layers with a width of 2^6 neurons each. The weights are initialized via the Xavier initialization. The optimization is driven by the Adam optimizer using a fixed learning rate $\alpha = 3 \cdot 10^{-4}$. During the initialization phase ($t = 0$), the neural networks (for the solution and for integration where applicable) are trained for a total of 2^{15} epochs, using 2^{15} sampled points at each epoch. A combination of uniform sampling for the moneyness and then sampling from the Dirichlet distribution is used during this step, providing a better initial start for the subsequent training steps. For the first time step ($t = \tau$), we perform 2^{14} and 2^{15} epochs for the 5-asset and 15-asset cases, respectively. Subsequently, 2^{12} (5-assets) and 2^{13} (15-assets) epochs are performed for the following time steps. At each iteration, a total of $2^{12}d$ points are sampled in $\mathbb{R}^d_+$. We have used scrambled Sobol sequences, *cf.* Sobol' [33], Owen [34], for the sampling, which provide low-discrepancy quasi-random samples. The algorithms have been executed on a computational node of the DelftBlue supercomputer. In particular, we used 12 CPU cores (Intel XEON E5), 64GB RAM, and an NVIDIA Tesla V100S with 32GB RAM.

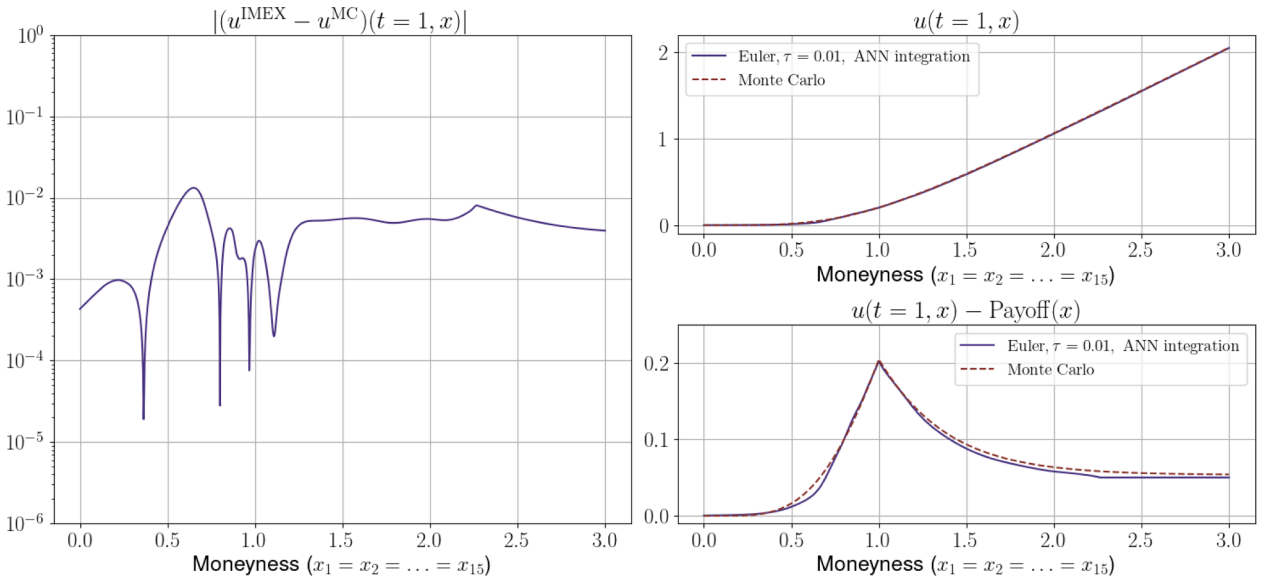


Fig. 10. Basket option prices (top right), differences between price and payoff (bottom right) and differences between the proposed method and quasi Monte Carlo (left), using the implicit-explicit Euler scheme and an ANN for the calculation of the integral, for $d = 15$.

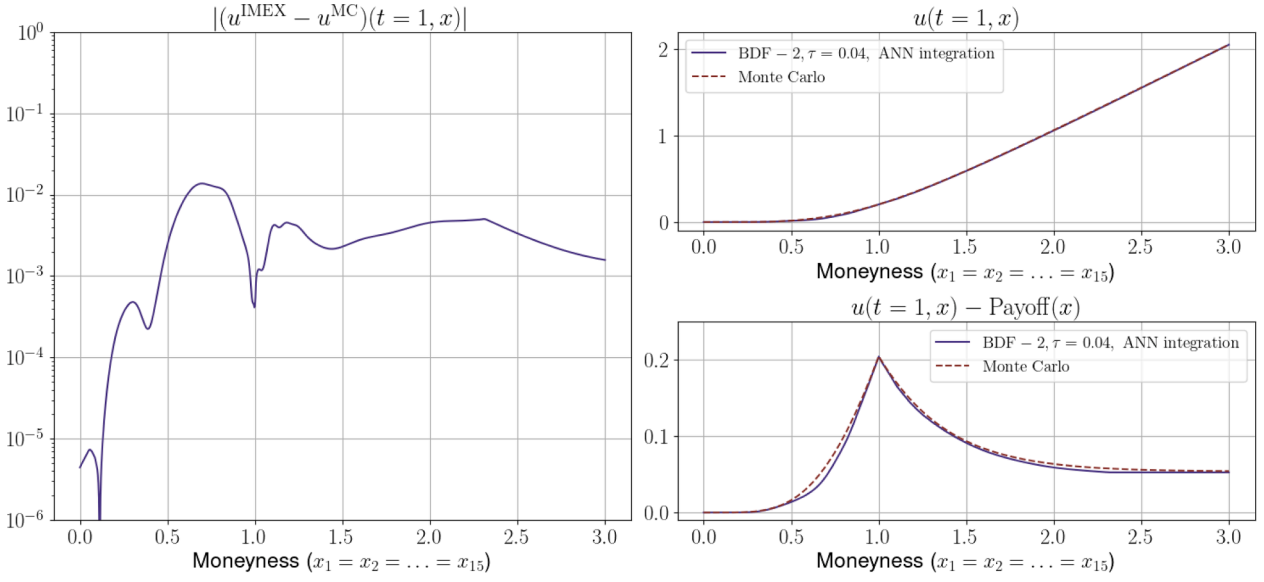


Fig. 11. Basket option prices (top right), differences between price and payoff (bottom right) and differences between the proposed method and quasi Monte Carlo (left), using the BDF-2 scheme and an ANN for the calculation of the integral operator, for $d = 15$.

4.5. Basic comparison with other ML solvers

In an effort to position the proposed method above in terms of computational cost, as well as comment on its benefits, we now provide a basic comparison against two popular alternative methods: the deep BSDE solver with jumps by Gnoatto et al. [29] and the deep Galerkin method by Sirignano and Spiliopoulos [2]; in the latter method, we incorporate the ANN approximation of the integral operator implicitly.

For the Deep BSDE solver, we adapt the code provided by Gnoatto et al. [29] on GitHub,[1] which prices European basket call options under a jump-diffusion model with uncorrelated assets. To ensure fair comparison, we also consider uncorrelated assets for the other two methods. We run the pricing schemes for European basket call options with 2, 5, and 8 underlying assets, using the

[1] https://github.com/AlessandroGnoatto/DeepBsdeSolverWithJumps

Table 2
Comparison of pricing methods for European basket call options of $d = 2, 5, 8$ assets under a jump–diffusion model.

d		MC	Deep IMEX	Deep BSDE	DGM (+1 layer)	DGM (+2 layers)
2	Value	0.2286	0.2269	0.2275	0.2168	0.2267
	Abs. Error	–	$1.74E{-}3$	$1.16E{-}3$	$1.19E{-}2$	$1.95E{-}3$
	Runtime (min)	179	373	83	498	658
5	Value	0.1647	0.1633	0.1661	0.1509	0.1669
	Abs. Error	–	$1.41E{-}3$	$1.36E{-}3$	$1.38E{-}2$	$2.14E{-}3$
	Runtime (min)	533	885	376	1330	1634
8	Value	0.1409	0.1393	0.1399	0.1498	0.1447
	Abs. Error	–	$1.61E{-}3$	$1.01E{-}3$	$8.89E{-}3$	$3.80E{-}3$
	Runtime (min)	822	1593	789	3521	4822

parameters: $T = 1,\ K = 1,\ \sigma_i = 0.5,\ r = 0.05,\ \lambda = 1,\ \mu_{J_i} = 0,\ \sigma_{Ji} = 0.5$. For the deep BSDE solver, we solve for the initial moneyness value of $x_i = 1$, while the other two methods it is possible to price the option for an array of initial values at once.

Due to library incompatibilities with NVIDIA drivers, we have been unable to execute a CUDA-accelerated version of the deep BSDE solver. To facilitate a fair computational time assessment we, thus, compare CPU runtimes for all methodologies. We expect that CUDA execution would likely result in speedups of 10 times or more for all methods in a similar fashion. Moreover, to ensure a robust Monte Carlo (MC) estimation, we run 10 million iterations, leading to a standard deviation of $\mathrm{std} = 1.147 \cdot 10^{-4}$. Thus, we feel it is appropriate to consider the Monte Carlo solution as the ground truth.

Before providing the results of the basic comparison, we highlight the key differences on how these methods approach the solution space. The MC method solves for a single time-space point, providing an estimate for a specific set of initial conditions at maturity. The deep BSDE solver provides solutions across multiple time steps but for fixed spatial values, *i.e.*, fixed initial asset or moneyness values. In contrast, both DGM and the deep IMEX scheme, proposed in this work, provide solution that can be evaluated across the entirety of a space-time domain. This broader coverage makes DGM and deep IMEX more versatile but, as expected, they are also more computationally intensive. For instance, a key advantage of being provided with a complete solutions across the space-time domain, allows for straightforward calculation of Greeks. The latter is more challenging in local solvers, such as MC or deep BSDE. We note that both deep IMEX and DGM are implemented with the same network architecture given in Section 3.1. To ensure, however, comparable accuracy, we test two different configurations for the DGM method: one with an additional layer and another with two additional layers, compared to the deep IMEX architecture. The results, including option values, absolute errors against the "ground truth" provided by MC, and runtimes, are summarized in Table 2. This basic comparison shows that the proposed deep IMEX approach is competitive with both MC and deep BSDE methodologies in terms of error and in terms of runtime. In contrast, however, to MC and BSDE, the proposed method produces a *complete* space-time solution.

The accuracy of the proposed scheme is stable across all tested dimensions ($d = 2, 5, 8$) and comparable to the deep BSDE solver. While it is computationally more expensive than the point-wise MC and deep BSDE methods, its relative cost scaling is favourable, particularly against the deep BSDE solver, where the runtime multiple drops from 4.5× (for $d = 2$) to 2× (for $d = 8$). Moreover, deep IMEX appears to be more economical than DGM in terms of runtime and parameter cardinality.

5. Conclusions

This work develops a novel deep learning method for PIDEs, focused on the particular challenge of pricing of European basket options in models that follow jump-diffusion processes. To address the intricacies of the problem, we introduce a decomposition technique, expressing the option price as the sum of an unknown component (time value) and a known lower-bound function (intrinsic value). The incorporation of a domain truncation method further enhances the accuracy of our numerical schemes. By projecting option prices onto a bounded subset and efficiently approximating solutions for extreme moneyness values within this truncated domain, we arrive at an accurate and reliable approximation of the underlying solution to the PIDE problem. The combination of the deep implicit-explicit minimizing movement methodology, the decomposition of the solution, and the domain truncation method form a robust toolkit for enhancing the accuracy and efficiency of option pricing methods in advanced financial models, by providing a complete solution at every point of the space-time domain. As such, it is trivially possible to infer further quantities, such as Greeks, from the computed space-time solution. A basic comparison with popular and successful approaches further showcases the practical relevance of the proposed method. We conclude by noting that the methodology presented above is also applicable to other PIDE problems.

CRediT authorship contribution statement

Emmanuil H. Georgoulis: Writing – original draft, Methodology, Funding acquisition; **Antonis Papapantoleon:** Writing – original draft, Supervision, Project administration, Methodology, Funding acquisition, Conceptualization; **Costas Smaragdakis:** Writing – original draft, Software, Methodology, Conceptualization.

Data availability

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

Credit author statement

We confirm that all authors having contributed to the manuscript in an equally substantial fashion. This includes the present revision of the manuscript.

Declaration of competing interests

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests:

Costas Smaragdakis reports financial support was provided by Hellenic Foundation for Research and Innovation. Antonis Papapantoleon reports financial support was provided by Hellenic Foundation for Research and Innovation. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] (DHPC) D H P CC. DelftBlue supercomputer (Phase 1). https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase1; 2022.
[2] Sirignano J, Spiliopoulos K. DGM: a deep learning algorithm for solving partial differential equations. J Comput Phys 2018;375:1339–64.
[3] Lagaris IE, Likas A, Fotiadis DI. Artificial neural networks for solving ordinary and partial differential equations. IEEE Trans Neural Netw 1998;9(5):987–1000.
[4] Raissi M, Perdikaris P, Karniadakis GE. Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. J Comput Phys 2019;378:686–707.
[5] E W, Yu B. The deep Ritz method: a deep learning-based numerical algorithm for solving variational problems. Commun Math Stat 2018;6(1):1–12.
[6] Liao Y, Ming P. Deep Nitsche method: deep Ritz method with essential boundary conditions. Commun Comput Phys 2019;29(5):1365–84.
[7] Georgoulis EH, Loulakis M, Tsiourvas A. Discrete gradient flow approximations of high dimensional evolution partial differential equations via deep neural networks. Commun Nonlinear Sci Numer Simul 2023;117:106893(11).
[8] Park MS, Kim C, Son H, Hwang HJ. The deep minimizing movement scheme. J Comput Phys 2023;494:112518(23).
[9] Han J, Jentzen A, E W. Solving high-dimensional partial differential equations using deep learning. Proc Nat Acad Sci 2018;115(34):8505–10.
[10] Eberlein E, Kallsen J. Mathematical finance. Springer; 2019.
[11] Cont R, Tankov P. Financial modelling with jump processes. Chapman & Hall/CRC; 2004.
[12] Schoutens W. Lévy processes in finance. Wiley; 2003.
[13] Merton R. Option pricing when underlying stock returns are discontinuous. J financ Econ 1976;3:125–44.
[14] Kou SG. A jump-diffusion model for option pricing. Manage Sci 2002;48:1086–101.
[15] Bates DS. Jumps and stochastic volatility: exchange rate processes implicit in Deutsche mark options. Rev Financ Stud 1996;9(1):69–107.
[16] Duffie D, Pan J, Singleton K. Transform analysis and asset pricing for affine jump-diffusions. Econometrica 2000;68:1343–76.
[17] He C, Kennedy JS, Coleman TF, Forsyth PA, Li Y, Vetzal KR. Calibration and hedging under jump diffusion. Rev Derivatives Res 2006;9(1):1–35.
[18] Kindermann S, Mayer PA. On the calibration of local jump-diffusion asset price models. Finance Stochastics 2011;15:685–724.
[19] Chen K-S, Huang Y-C. Detecting jump risk and jump-diffusion model for bitcoin options pricing and hedging. Mathematics 2021;9(20):2567.
[20] Carr P, Madan D. Option valuation using the fast Fourier transform. J Comput Finance 1999;2(4):61–73.
[21] Fang F, Oosterlee CW. A novel pricing method for European options based on Fourier-cosine series expansions. SIAM J Sci Comput 2008/09;31:826–48.
[22] Eberlein E, Glau K, Papapantoleon A. Analysis of Fourier transform valuation formulas and applications. Appl Math Finance 2010;17:211–40.
[23] Bayer C, Ben Hammouda C, Papapantoleon A, Samet M, Tempone R. Optimal damping with hierarchical adaptive quadrature for efficient Fourier pricing of multi-asset options in Lévy models. J Comput Finance 2023;27(3):43–86.
[24] Bayer C, Ben Hammouda C, Papapantoleon A, Samet M, Tempone R. Quasi-Monte Carlo for efficient Fourier pricing of multi-asset options; 2024. arXiv:2403.02832.
[25] Griebel M, Hullmann A. An efficient sparse grid Galerkin approach for the numerical valuation of basket options under Kou's jump-diffusion model. In: Sparse grids and applications. Springer; 2013, p. 121–50.
[26] Hepperger P. Option pricing in Hilbert space-valued jump-diffusion models using partial integro-differential equations. SIAM J Financ Math 2010;1:454–89.
[27] Reichmann O, Schwab C. Numerical analysis of additive, Lévy and Feller processes with applications to option pricing. In: Lévy matters I. Springer; 2010, p. 137–96.
[28] Ambrosio L. Movimenti minimizzanti. Rend Accad Naz Sci XL Mem Mat Appl (5) 1995;19:191–246.
[29] Gnoatto A, Patacca M, Picarelli A. A deep solver for BSDEs with jumps. 2022. arXiv:2211.04349.
[30] Runggaldier WJ. Jump-diffusion models. In: Rachev ST, editor. Handbook of heavy tailed distributions in finance. North-Holland; 2003, p. 169–209.
[31] Akrivis G, Crouzeix M, Makridakis C. Implicit-explicit multistep methods for quasilinear parabolic equations. Numer Math 1999;82:521–41.
[32] Bungartz H-J, Griebel M. Sparse grids. Acta Numerica 2004;13:147–269.
[33] Sobol' IM. On the distribution of points in a cube and the approximate evaluation of integrals. USSR Comput Math Math Phys 1967;7(4):86–112.
[34] Owen AB. Scrambling Sobol' and Niederreiter-Xing points. J Complex 1998;14:466–89.