

# A DEEP IMPLICIT-EXPLICIT MINIMIZING MOVEMENT METHOD FOR PARTIAL INTEGRO-DIFFERENTIAL EQUATIONS, WITH APPLICATION TO OPTION PRICING IN JUMP-DIFFUSION MODELS

EMMANUIL H. GEORGULIS, ANTONIS PAPAPANTOLEON, AND COSTAS SMARAGDAKIS

**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.

## 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 [30], the Physics-Informed Neural Network methodology [21, 26], and related methods involving the minimization of physical energies of the underlying PDE systems [11, 15, 15, 22, 25], etc. When the models can be posed as backward-forward stochastic control problems, fast methodologies such as see also the deep BSDE method [18] 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

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

---

2020 *Mathematics Subject Classification.* 68T07, 65C20, 91G60, 91G20, 65M12.

*Key words and phrases.* Basket option, jump-diffusion model, PIDE, minimizing movement method, implicit-explicit method, artificial neural network, integral operator, Gauss–Hermite quadrature.

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 [9]. 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).

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

$$\text{Payoff}(x) = u_0(x) := \left( \sum_{i=1}^d \alpha_i x_i - 1 \right)^+. \quad (1.1)$$

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 Cont and Tankov [8], Eberlein and Kallsen [12] or Schoutens [29] for more details and references on models with jumps in finance.

A popular model that incorporates random jumps is the, so-called, Merton [23] 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. Other popular jump-diffusion models in the literature are the Kou [20] model and affine jump-diffusions, see *e.g.* Bates [3] or Duffie, Pan, and Singleton [10].

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.* [7, 13, 14]. 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 [4, 5] 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 (1.1) 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 [17], Hepperger [19] 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 [2], provides discretization of gradient flows in the calculus of variations, by considering a suitable minimization functional for each time step. A canonical example is the minimizing movement method for the Dirichlet energy corresponding to the backward Euler timestepping for the heat equation. In this work, we develop a significant extension of the “deep minimizing movement” method from Georgoulis et al. [15], Park et al. [25] to approximate PIDE problems, with specific focus on those arising from basket option pricing in jump-diffusion models. In [15, 25], parabolic problems are discretized by ANNs (in space) in a time-stepping fashion by minimizing appropriate energy cost functionals; see also E and Yu [11], Liao and Ming [22] for energy minimization for elliptic PDEs. The resulting method in [15, 25] amounts to the backward Euler timestepping (discrete gradient flow) for the Dirichlet energy minimization via a, so-called, deep Nitsche approach. In the current framework, the spatial operator in the PIDE model problem involves skew-symmetric and integral terms, which are (effectively) discretized in an explicit manner. To that end, we consider the family of implicit-explicit versions of Backward Differentiation Formulae (BDF) methods, the first order of which is the implicit-explicit Euler scheme. 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. We note that the minimizing movement approach from [15, 25] has the important advantage of using the

ANN parameters from the previous time-step as starting point in the optimization (learning) of the approximate solution parameters at the current step. This reduces the computational cost of optimization significantly; see [15].

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) [30], the deep BSDE solvers with jumps [18] and Gnoatto, Patacca, and Picarelli [16], 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 [23], whereby the dynamics of the  $i$ -th stock at time  $t$  are provided by

$$S_t^{(i)} = S_0^{(i)} \exp \left( b_i t + \sigma_i W_t^{(i)} + \sum_{k=1}^{N_t} Z_k^{(i)} \right), \quad t \in \mathbb{T} = (0, T], \quad i \in \mathbb{I} = \{1, \dots, d\}, \quad (2.1)$$

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  $\mu_J$  and variance-covariance matrix  $\Sigma_J$ . 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

$$b_i = r - \frac{1}{2} \sigma_i^2 - \lambda \left( \exp \left\{ \mu_{J_i} + \frac{1}{2} \sigma_{J_i}^2 \right\} - 1 \right), \quad (2.2)$$

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 (1.1) satisfies the

following PIDE:

$$\begin{aligned} \frac{\partial u}{\partial t} - \frac{1}{2} \sum_{i,j=1}^d \sigma_i \rho_{ij} \sigma_j x_i x_j \frac{\partial^2 u}{\partial x_i \partial x_j} + \sum_{i=1}^d b_i x_i \frac{\partial u}{\partial x_i} + ru - I_\varphi[u] &= 0 \\ u(0, x) &= u_0(x), \end{aligned} \quad (2.3)$$

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

$$I_\varphi[u] = \lambda \int_{\mathbb{R}^d} (u(t, xe^z) - u(t, x)) \varphi(dz), \quad (2.4)$$

with  $\varphi(\cdot)$  denoting the probability density function of the multivariate normal distribution with mean  $\mu_J$  and variance-covariance matrix  $\Sigma_J$ . Obviously,  $u(t, 0) = 0$  and  $I[u(t, 0)] = 0$ , for each  $t > 0$ . The initial condition in (2.3) is derived from the terminal condition of the classical PIDE for option pricing, *i.e.* from the payoff function in (1.1), by employing the change of variable  $t = T - \cdot$ .

**2.2. General form of the PIDE for option pricing.** Motivated by (2.3), we consider the following general problem: find the arbitrage-free price  $u$  of an option with payoff (1.1), that satisfies the PIDE

$$\begin{aligned} \frac{\partial}{\partial t} u(t, x) + \mathcal{A}u(t, x) &= 0, \quad (t, x) \in \mathbb{T} \times [0, \infty)^d \\ u(0, x) &= u_0(x), \quad x \in [0, \infty)^d, \end{aligned} \quad (2.5)$$

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

$$\mathcal{A}u(t, x) = - \sum_{i,j=1}^d a_{ij}(x) \frac{\partial^2 u}{\partial x_i \partial x_j} + \sum_{i=1}^d b_i(x) \frac{\partial u}{\partial x_i} + ru - I_\nu[u], \quad (2.6)$$

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

$$I_\nu[u] = \lambda \int_{\mathbb{R}^d} (u(t, xe^z) - u(t, x)) \nu(x, dz), \quad (2.7)$$

with the integration performed over a finite jump measure  $\nu$ . The coefficients  $a_{ij}$  and  $\nu$  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 (2.2). This class of models includes Lévy jump-diffusions, such as the Kou [20] model, and affine jump-diffusions, see *e.g.* Bates [3] or Duffie et al. [10]; see also Runggaldier [28] 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 (2.5) 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

$$\mathcal{L}u = - \sum_{j=1}^d \frac{\partial}{\partial x_j} \left( \sum_{i=1}^d a_{ij}(x) \frac{\partial u}{\partial x_i} \right) + ru, \quad (2.8)$$

$$f[u] = \sum_{i=1}^d \left( b_i(x) + \sum_{j=1}^d \frac{\partial}{\partial x_j} a_{ij}(x) \right) \frac{\partial u}{\partial x_i} - I_\nu[u]. \quad (2.9)$$

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,  $\int_{\mathbb{R}^d} u(t, x) \nu(x, dz)$ , 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  $t_k = k\tau$ ,  $k = 0, 1, \dots, n$ , 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, Crouzeix, and Makridakis [1] for their numerical analysis. More specifically, for given parameter sets  $\beta_j, \gamma_j$ , we consider the time-stepping methods

$$\frac{\beta_p u^k - \sum_{j=0}^{p-1} \beta_j u^{k-j-1}}{\tau} + \mathcal{L}u^k + \sum_{j=0}^{p-1} \gamma_j f[u^{k-j-1}] = 0, \quad (2.10)$$

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

$$\frac{u^k - u^{k-1}}{\tau} + \mathcal{L}u^k + f[u^{k-1}] = 0, \quad (2.11)$$

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

$$\frac{\frac{3}{2}u^k - 2u^{k-1} + \frac{1}{2}u^{k-2}}{\tau} + \mathcal{L}u^k + 2f[u^{k-1}] - f[u^{k-2}] = 0. \quad (2.12)$$

A crucial observation, following from the minimizing movement point of view, is that (2.10) (upon multiplication by  $\tau$  and integration by parts) is the Euler–Lagrange equation of the convex minimization problem:

$$\mathcal{C}[u] := \frac{1}{2} \left\| \beta_p u - \sum_{j=0}^{p-1} \beta_j u^{k-j-1} \right\|_{L^2(\Omega)}^2 + \tau \int_{\Omega} \mathcal{E}[u] dx + \tau \sum_{j=0}^{p-1} \gamma_j \int_{\Omega} f[u^{k-j-1}] u dx \rightarrow \min, \quad (2.13)$$

whereby

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

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  $|x| \rightarrow \infty$ . Fortunately, the modelling asserts that these values of the solution  $u$  as  $|x| \rightarrow \infty$  are asymptotically known. In particular, the value of the option at time  $t \geq 0$  can be expressed as

$$u(t, x) = w(t, x) + v(t, x), \quad (2.14)$$

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

$$u(t, x) \geq \left( \sum_{i=1}^d \alpha_i x_i - e^{-rt} \right)^+ = u_0(x) + (1 - e^{-rt}) H \left( \sum_{i=1}^d \alpha_i x_i - e^{-rt} \right) =: v(t, x), \quad (2.15)$$

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

$$\lim_{\sum \alpha_i x_i \rightarrow \infty} u(t, x) = \left( \sum_{i=1}^d \alpha_i x_i - e^{-rt} \right)^+ = \sum_{i=1}^d \alpha_i x_i - e^{-rt}. \quad (2.16)$$

Therefore, the decomposition (2.14) implies that  $\lim_{\sum a_i x_i \rightarrow \infty} w(t, x) = 0$ , 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

$$\tilde{v}(t, x; \eta) = \left( \sum_{i=1}^d \alpha_i x_i - 1 \right)^+ + (1 - e^{-rt}) \text{Sigmoid} \left( \sum_{i=1}^d \alpha_i x_i - e^{-rt}; \eta \right), \quad (2.17)$$

with  $\text{Sigmoid}(x; \eta) := (1 + e^{-\eta x})^{-1}$ ,  $\eta > 0$ . Obviously,  $\lim_{\eta \rightarrow \infty} \tilde{v}(t, x; \eta) = v(t, x)$ . Moreover, the parameter  $\eta$  influences the smoothness of the gradient of  $\tilde{v}(t, x; \eta)$  with respect to  $x$ . In particular, smaller values of  $\eta$  result in smoother approximations. Hence, the parameter  $\eta$  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 (2.16) and observe that

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

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:

$$u(t, x') \approx u(t, x) + \sum_{i=1}^d \alpha_i (x'_i - x_i), \quad (2.18)$$

for  $\sum_{i=1}^d \alpha_i x_i$  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  $x_i \leq x_{\max}$ ,  $i \in \mathbb{I}$ , for a given  $x_r \leq x_{\max}$ . The boundary  $\partial\mathcal{R}$  of  $\mathcal{R}$  consists of the points which satisfy  $\sum_i \alpha_i x_i = x_r$  and  $x_i \leq x_{\max}$  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

$$q(x) = \begin{cases} x_{\max}/\max\{x_i\}, & \text{if } \max\{x_i\} \geq \max\left(\sum_{i=1}^d \alpha_i x_i, x_r\right) x_{\max}/x_r \\ x_r/\max\left(\sum_{i=1}^d \alpha_i x_i, x_r\right), & \text{otherwise.} \end{cases} \quad (2.19)$$

Clearly,  $0 \leq q(x) \leq 1$  for all  $x \in \mathbb{R}_+^d$  and  $q(x)x \in \mathcal{R}$ . Employing the approximation (2.18), 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  $\mathcal{R}$ :

$$u(t, x) \approx u(t, y(x)) + \sum_{i=1}^d \alpha_i (x - y(x)), \quad \text{for each } x \in \mathbb{R}_+^d. \quad (2.20)$$

Figure 2.1 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. [15], see also the deep BSDE method of Han et al. [18], and is in contrast to other popular ANN-type methods for PDEs, such as the deep Galerkin method (DGM) of Sirignano and Spiliopoulos [30] or the physics-informed neural networks (PINNs) by Lagaris et al. [21], Raissi et al. [26],

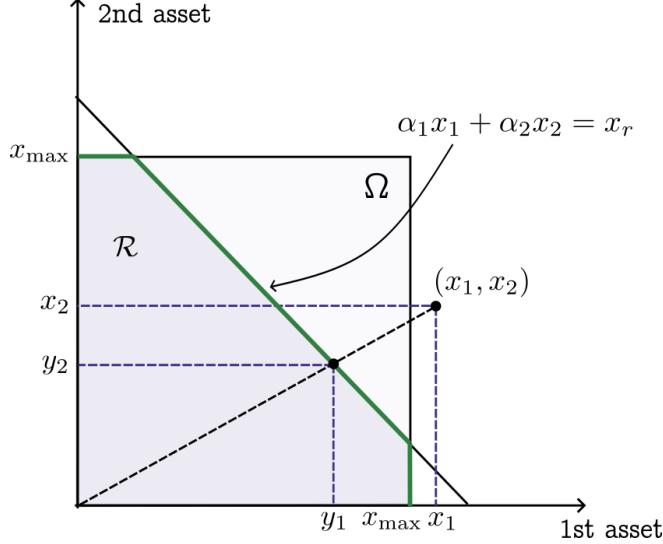


FIGURE 2.1. A two-dimensional example for the domain truncation. The green boundary defines the projection surface for the option price in extreme moneynesses.

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 [30]; the architecture of each layer follows [30] and, for this reason, is henceforth designated as “DGM layer”. The network implements the decomposition and boundary modeling described in the previous subsections. More specifically, for an input  $y$ , we set

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

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

$$\theta = \{W^{\text{in}}, b^{\text{in}}, (V^{*,\ell}, W^{*,\ell}, b^{*,\ell})_{\ell=1,\dots,L}^{*\in\{g,z,r,h\}}, W^{\text{out}}\}, \quad (3.1)$$

and the output of the network is given by

$$U(t_k, x; \theta) = w^k(y; \theta) + \tilde{v}(t_k, y) + \sum_{i=1}^d \alpha_i(x_i - y_i), \quad (3.2)$$

recalling the definition of  $y \equiv y(x)$  (and of the  $y_i$ 's) in (2.19). Figure 3.1 shows a flowchart of the adopted ‘global’ ANN architecture, while Figure 3.2 presents the architecture of a single DGM layer.

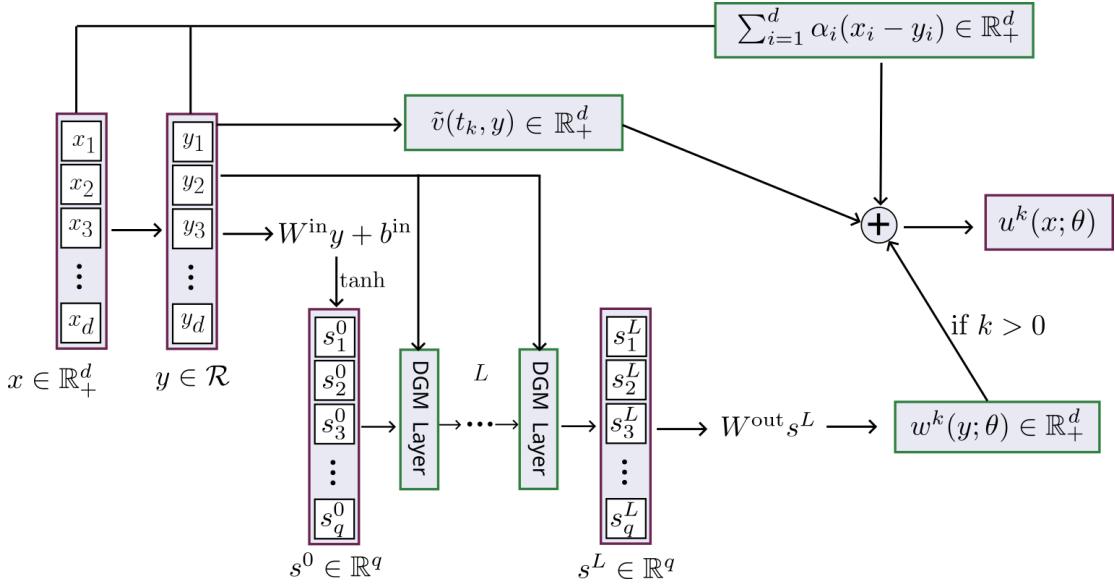


FIGURE 3.1. Flowchart of the deep neural network modeling the solution of the PIDE for a time instance  $t = t_k$ .

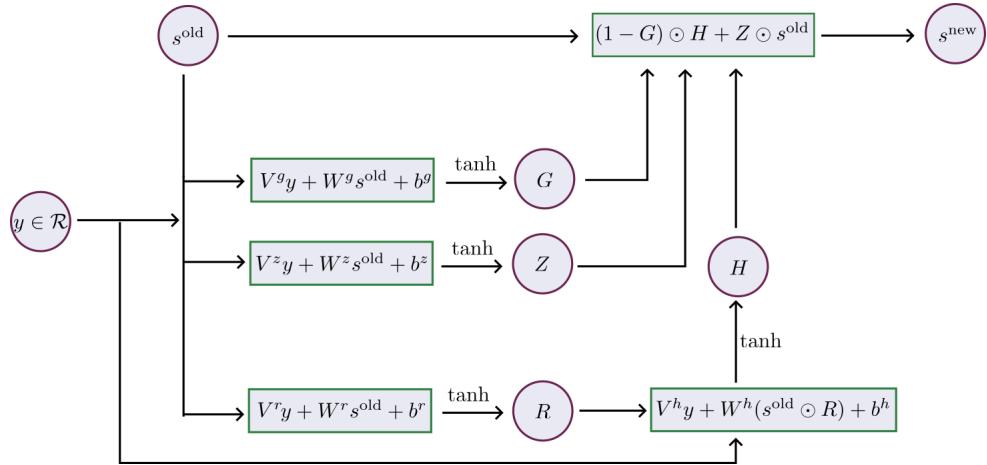


FIGURE 3.2. The architecture of a DGM layer.

The computation of the parameters  $\theta$  which provide the approximate solution at  $t_k$  is performed using (2.13), *i.e.*, we seek  $\theta$  minimizing the cost functional

$$\begin{aligned} \mathcal{C}_k(\theta) := \mathcal{C}[U(t_k, \cdot, \theta)] &= \frac{1}{2} \left\| \beta_p U(t_k, \cdot, \theta) - \sum_{j=0}^{p-1} \beta_j U(t_{j(k)}, \cdot, \theta^{j(k)}) \right\|_{L^2(\Omega)}^2 \\ &\quad + \tau \int_{\Omega} \mathcal{E}[U(t_k, x, \theta)] dx + \tau \sum_{j=0}^{p-1} \gamma_j \int_{\Omega} f[U(t_{j(k)}, x, \theta^{j(k)})] U(t_k, x, \theta) dx, \end{aligned} \tag{3.3}$$

using the notation  $j(k) := k - j - 1$  for brevity. Let us point out that, due to the network architecture, although  $\mathcal{C}$  is convex with respect to its argument,  $\mathcal{C}$  is not, in general. The minimizer is denoted by  $\theta^k$ . Due to the multistep nature of the BDF methods, we observe that  $\theta^k$  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 3.1.** We note the practical significance of mollifying the solution lower bound as per (2.17). Indeed,  $v(t_{j(k)}, \cdot)$  is contained in  $U(t_{j(k)}, \cdot, \theta^{j(k)})$  and it is differentiated in space within  $\mathcal{C}$ .

**3.2. Training.** As is typical in ANN-based methods for PDEs, the cost function (3.3) 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

$$\begin{aligned} \tilde{\mathcal{C}}_k(\theta) := & \frac{(x_{\max})^d}{N} \sum_{i=1}^N \left\{ \frac{1}{2} \left[ \beta_p U(t_k, x^i, \theta) - \sum_{j=0}^{p-1} \beta_j U(t_{j(k)}, x^i, \theta^{j(k)}) \right]^2 \right. \\ & \left. + \tau \mathcal{E}[U(t_k, x^i, \theta)] + \tau \sum_{j=0}^{p-1} \gamma_j f[U(t_{j(k)}, x^i, \theta^{j(k)})] U(t_k, x^i, \theta) \right\}, \quad (3.4) \end{aligned}$$

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

**3.2.1. Initialization.** The initial condition (2.5) 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  $\delta_{ij}$  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:

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

for  $\epsilon > 0$  small, with  $\zeta(x)$  taking two possible positive values  $\zeta_1, \zeta_2$  according to the criterion  $\sum_{i=1}^d \alpha_i x_i > 1$  and we compute  $\theta^0$  such that

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

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  $I_{\nu}$  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  $h(z) = u(xe^z) - u(x)$  multiplied by the Poisson parameter  $\lambda$ ; 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,  $\Sigma_J$  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  $\Sigma_J$ . 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.,

$$\Sigma_J = A\Lambda A^\top = A\Lambda^{1/2}\Lambda^{1/2}A^\top = BB^\top, \quad (3.5)$$

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

$$\begin{aligned} \int_{\mathbb{R}^d} h(z)\nu(dz) &= \lambda\pi^{-d/2} \int_{\mathbb{R}^d} h(\mu + \sqrt{2}By) \exp(-y^\top y) dy \\ &\approx \lambda\pi^{-d/2} \sum_{\mathbf{i} \in \Theta_q} h(\mu + \sqrt{2}By^\mathbf{i}) W^\mathbf{i}, \end{aligned} \quad (3.6)$$

with  $y^\mathbf{i} = (y^{i_1}, \dots, y^{i_d})^\top$  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 [6]. Figure 3.3 presents an example of a sparse grid of points in  $\mathbb{R}^2$  for integrating a function of two correlated normally distributed random variables.

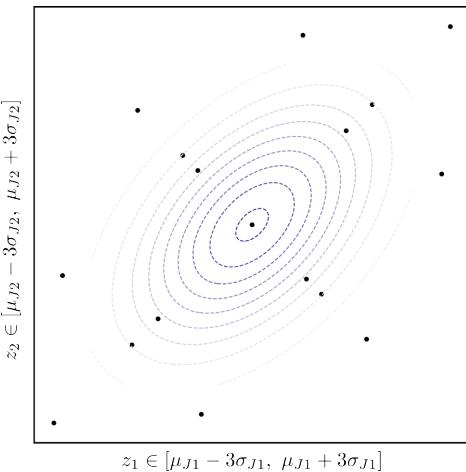


FIGURE 3.3. Sparse sampling in  $\mathbb{R}^2$  using the Gauss–Hermite quadrature for normally distributed random variables with correlation equal to  $\frac{1}{2}$ .

**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. [16], however, we introduce a neural network that directly learns the values of the integral  $\int h(z)\nu(dz)$ , while in [16] 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  $\int h(z)\nu(dz)$  with  $h(z) = u(xe^z) - u(x)$  for each value of  $x$ . The approximate neural network value at time  $t_k$  is denoted by  $\mathcal{I}^k(x; \phi)$ , where  $\phi$  is the set of trainable parameters. In order to determine  $\phi^k$ , we solve the optimization problem

$$\min_{\phi \in \Phi} \mathbb{E} \left[ \mathcal{I}^k(x; \phi) - \sum_{j=1}^{p-1} \gamma_j I_\nu[U(t_{j(k)}, x; \theta^{j(k)})] \right]^2, \quad (3.7)$$

with the expectation taken over  $x$ .

As usual, we discretize the integral in (3.7) 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  $h_{j(k)}(x, z) := U(t_{j(k)}, xe^z, \theta^{j(k)}) - U(t_{j(k)}, x, \theta^{j(k)})$ , 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

$$\min_{\phi \in \Phi} \mathbb{E} \left[ \mathcal{I}^k(x; \phi) - \frac{\lambda}{M} \sum_{r=1}^M \sum_{j=1}^{p-1} \gamma_j h_{j(k)}(x, z^r) \right]^2; \quad (3.8)$$

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 (3.8):

$$\frac{(x_{\max})^d}{N} \sum_{i=1}^N \left[ \mathcal{I}^k(x^i; \phi) - \frac{\lambda}{M} \sum_{r=1}^M \sum_{j=1}^{p-1} \gamma_j h_{j(k)}(x^i, z^r) \right]^2. \quad (3.9)$$

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  $\phi^k$  of the ANN for the integral by minimizing (3.9), followed by the independent optimization for  $\theta^k$  based on (3.4). Fortunately, the training associated with (3.9) 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  $x_1, x_2, \dots, x_d$ . The model parameters are chosen to be the following:

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

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

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

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. Figure 4.1 corresponds to the implicit-explicit Euler scheme, and Figure 4.2 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 minutes, appears slightly more accurate than the BDF-2 scheme, due to the smallest time-stepping. This comes at a computational cost, since it is approximately 1.8 times slower.

**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. Figures 4.3 and 4.4 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 minutes) than the corresponding method with the Gauss–Hermite quadrature.

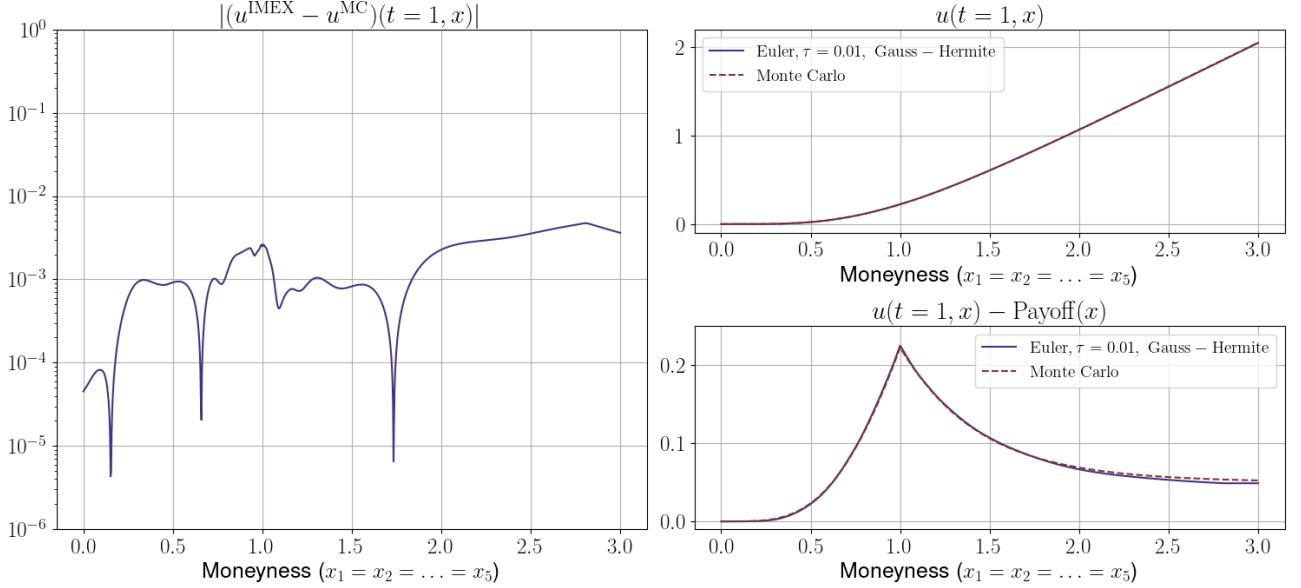


FIGURE 4.1. Basket option prices (top right), differences between price and payoff (bottom right) and differences between the proposed method and quasi Monte Carlo (left), using the implicit-explicit Euler scheme and the Gauss–Hermite quadrature for the integral, for  $d = 5$ .

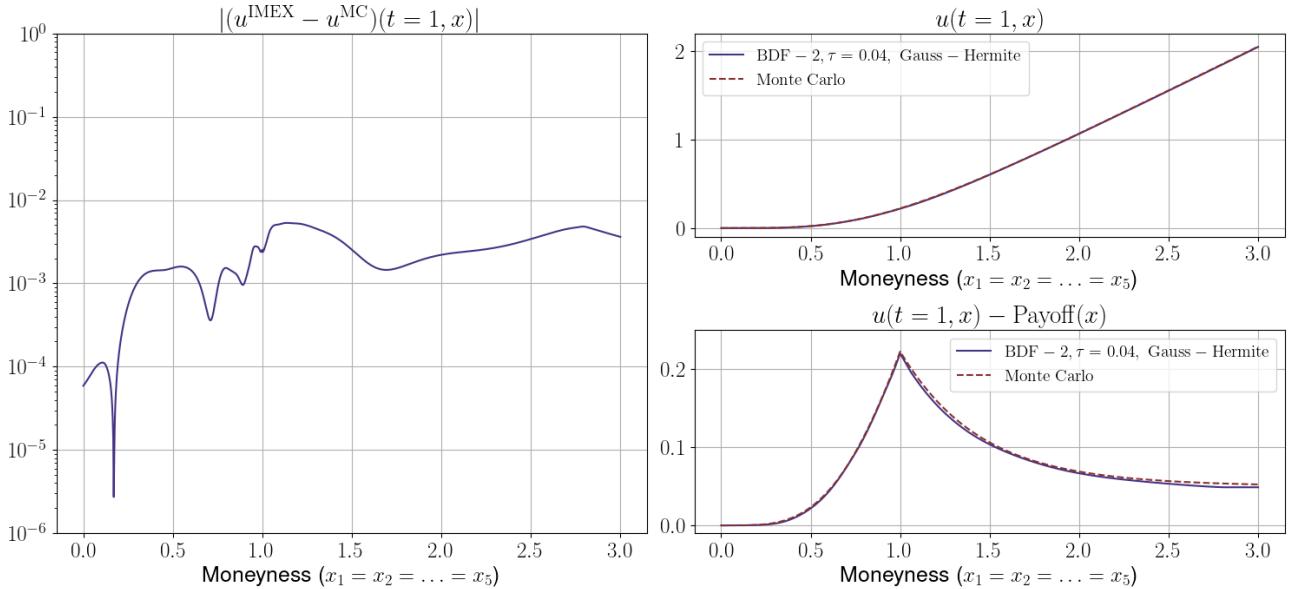


FIGURE 4.2. Basket option prices (top right), differences between price and payoff (bottom right) and differences between the proposed method and quasi Monte Carlo (left), using the BDF-2 scheme and the Gauss–Hermite quadrature for the integral, for  $d = 5$ .

Figure 4.5 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 of the integral for large  $x$ , aligning with the expected jump sizes, *i.e.*:  $\lim_{x \rightarrow \infty} I_\varphi(x) = x\mathbb{E}[e^z - 1]$ . On the other hand, the ANN-based approach tends to yield smoother approximations, particularly for commonly sampled moneyness levels.

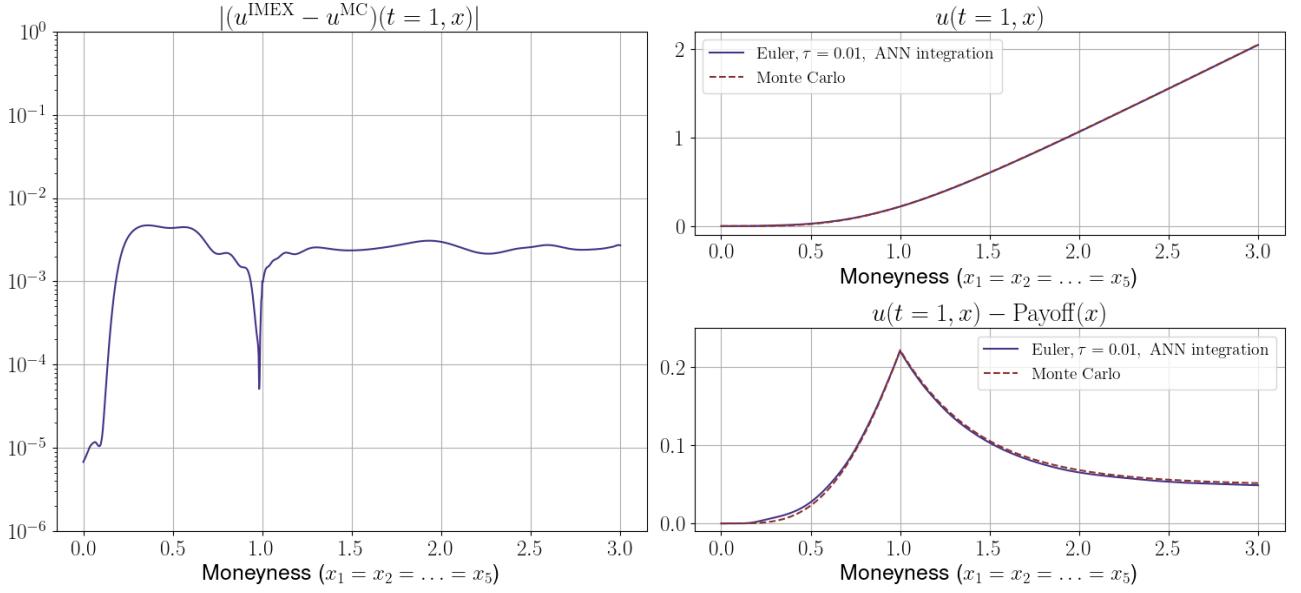


FIGURE 4.3. Basket option prices (top right), differences between price and payoff (bottom right) and differences between the proposed method and quasi Monte Carlo (left), using the implicit-explicit Euler scheme and an ANN for the calculation of the integral, for  $d = 5$ .

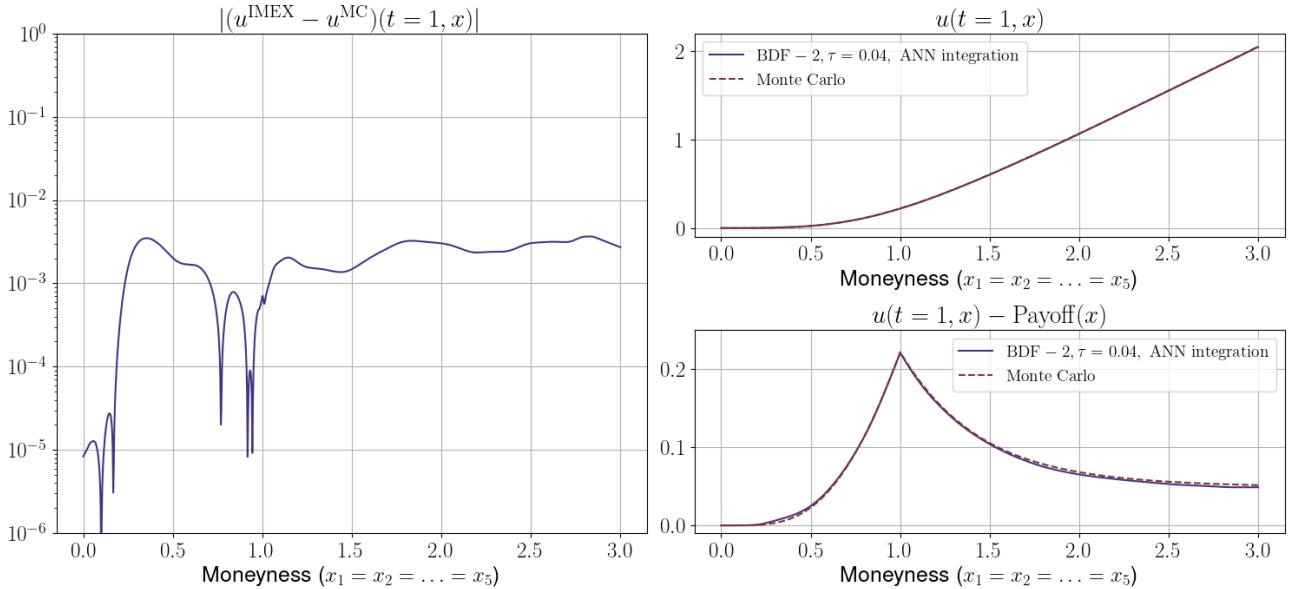
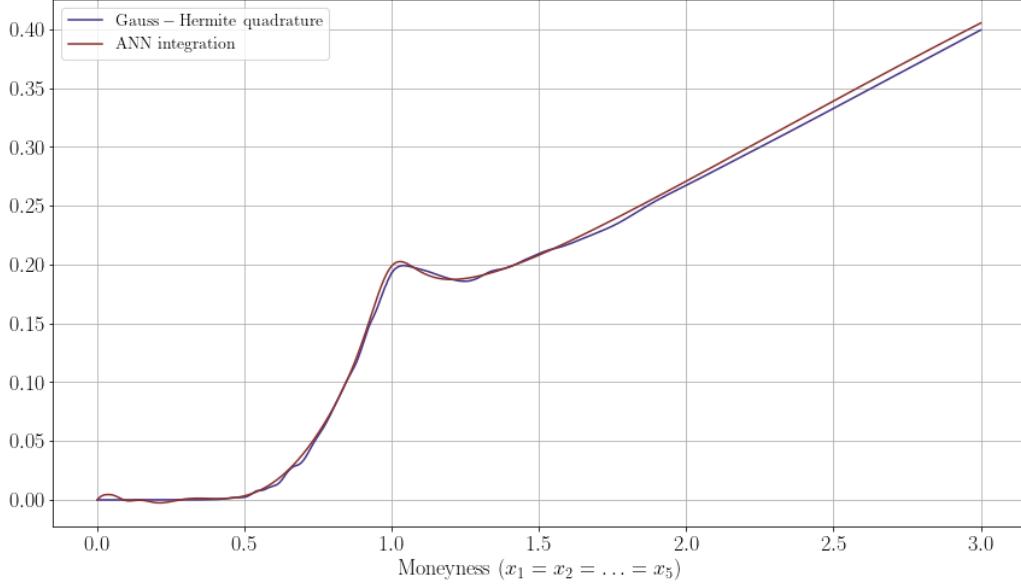


FIGURE 4.4. Basket option prices (top right), differences between price and payoff (bottom right) and differences between the proposed method and quasi Monte Carlo (left), using the BDF-2 scheme and an ANN for the calculation of the integral operator, for  $d = 5$ .

In addition, a crucial factor in choosing the integration method emerges when comparing their computational costs at various dimensions. Here, we compared 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 4.1.

The ANN-based approach performed significantly faster across all the tested dimensions. More importantly, its computational time scales nicely with  $d$ . On the other hand, the runtime of the Gauss–Hermite quadrature grows exponentially, which indicates that this quadrature method suffers heavily from the curse of dimensionality.

FIGURE 4.5. Estimates of the integral operator for  $t = 0.01$ .

$d$	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	$1.45E-1$

TABLE 4.1. Computational cost and relative absolute differences between the two integration methods across the various dimensions.

In conclusion, while Gauss–Hermite quadrature provides a robust approach for low-dimensional problems, its computational burden rapidly becomes prohibitive. The ANN-based quadrature approach, with its better scalability, is the only feasible option for high-dimensional problems, with a practical dimensional limit for the GH method appearing around  $d = 8$  in our test cases.

**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. Figure 4.6 presents the pricing results for the implicit-explicit Euler scheme, whereas Figure 4.7 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 is now 1.7 times faster than the implicit-explicit Euler method (which performed for 1394 minutes), however the time-step is now smaller compared to the case with 5 assets.

**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

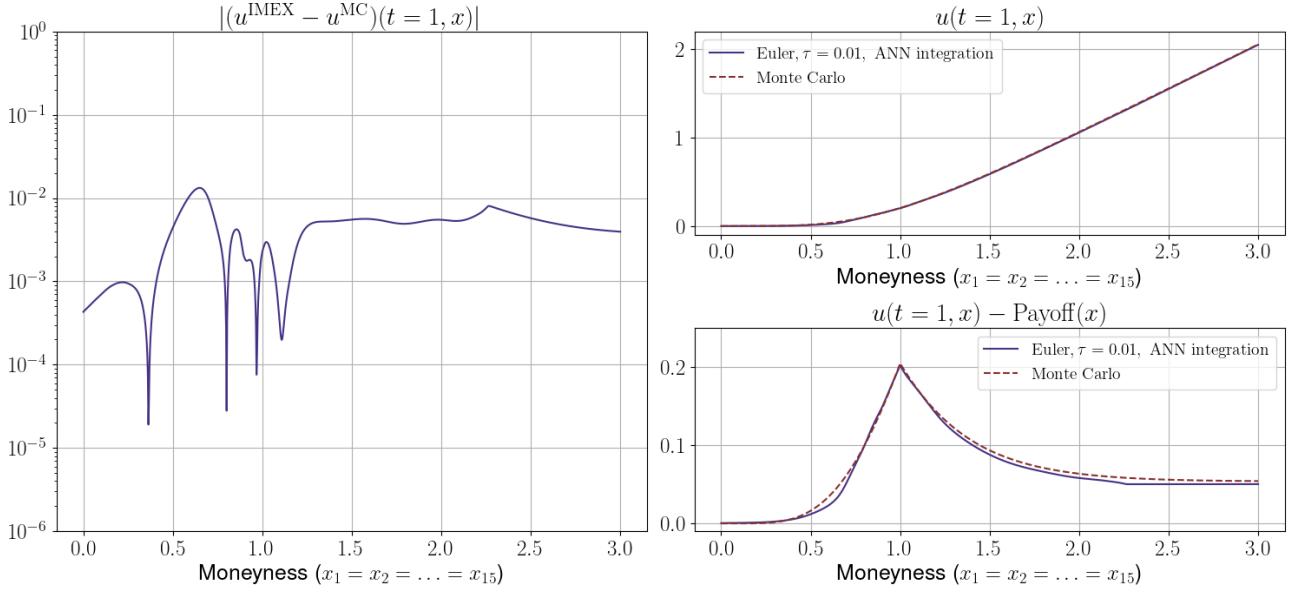


FIGURE 4.6. Basket option prices (top right), differences between price and payoff (bottom right) and differences between the proposed method and quasi Monte Carlo (left), using the implicit-explicit Euler scheme and an ANN for the calculation of the integral, for  $d = 15$ .

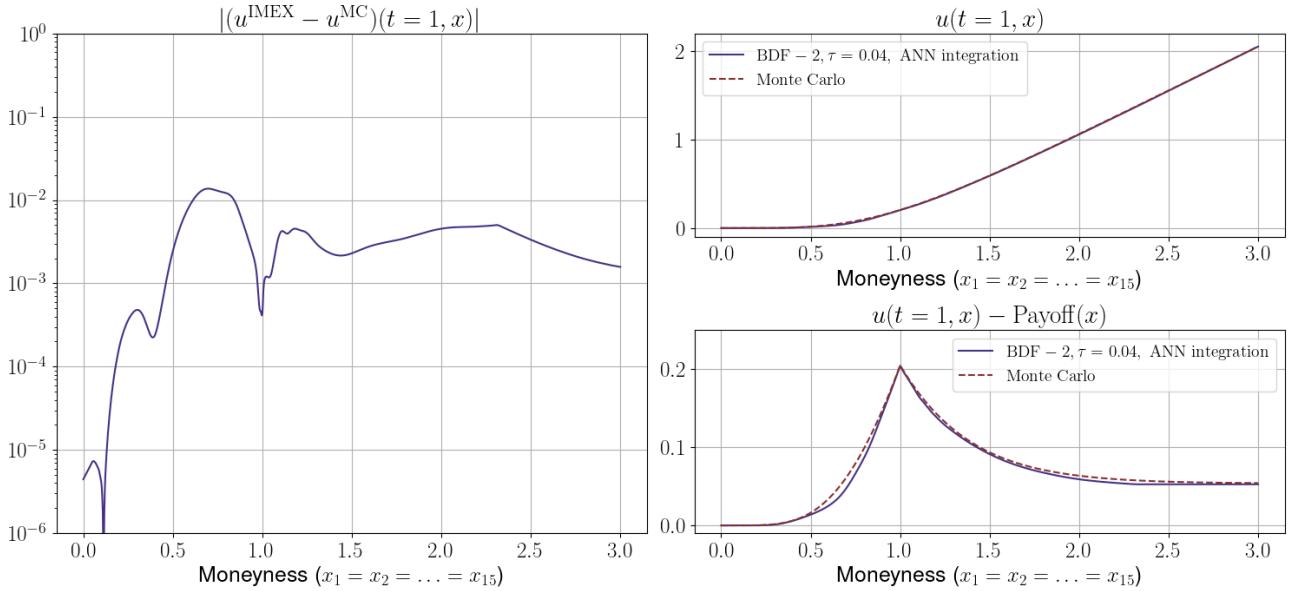


FIGURE 4.7. Basket option prices (top right), differences between price and payoff (bottom right) and differences between the proposed method and quasi Monte Carlo (left), using the BDF-2 scheme and an ANN for the calculation of the integral operator, for  $d = 15$ .

iteration, a total of  $2^{12}d$  points are sampled in  $\mathbb{R}_+^d$ . We have used scrambled Sobol sequences, cf. Owen [24], Sobol' [31], 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.

**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. [16] and the deep Galerkin method by

Sirignano and Spiliopoulos [30]; 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. [16] on GitHub<sup>1</sup>, 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 parameters:  $T = 1$ ,  $K = 1$ ,  $\sigma_i = 0.5$ ,  $r = 0.05$ ,  $\lambda = 1$ ,  $\mu_{J_i} = 0$ ,  $\sigma_{J_i} = 0.5$ . 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  $\text{std} = 1.147 \cdot 10^{-4}$ . 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 4.2. This basic comparison shows that the proposed deep IMEX approach is competitive with both MC and deep BSDE methodologies in terms of error and runtime, to produce 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  $4.5 \times$  (for  $d = 2$ ) to  $2 \times$  (for  $d = 8$ ). Moreover, deep IMEX appears to be more economical than DGM in terms of runtime and parameter cardinality.

$d$		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

TABLE 4.2. Comparison of pricing methods for European basket call options of  $d = 2, 5, 8$  assets under a jump-diffusion model.

<sup>1</sup><https://github.com/AlessandroGnoatto/DeepBsdeSolverWithJumps>

## 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.

## REFERENCES

- [1] G. Akrivis, M. Crouzeix, and C. Makridakis. Implicit-explicit multistep methods for quasilinear parabolic equations. *Numerische Mathematik*, 82:521–541, 1999.
- [2] L. Ambrosio. Movimenti minimizzanti. *Rend. Accad. Naz. Sci. XL Mem. Mat. Appl.* (5), 19:191–246, 1995.
- [3] D. S. Bates. Jumps and stochastic volatility: Exchange rate processes implicit in Deutsche mark options. *The Review of Financial Studies*, 9(1):69–107, 1996.
- [4] C. Bayer, C. Ben Hammouda, A. Papapantoleon, M. Samet, and R. Tempone. Optimal damping with hierarchical adaptive quadrature for efficient Fourier pricing of multi-asset options in Lévy models. *Journal of Computational Finance*, 27(3):43–86, 2023.
- [5] C. Bayer, C. Ben Hammouda, A. Papapantoleon, M. Samet, and R. Tempone. Quasi-Monte Carlo for efficient Fourier pricing of multi-asset options. Preprint, arXiv:2403.02832, 2024.
- [6] H.-J. Bungartz and M. Griebel. Sparse grids. *Acta Numerica*, 13:147–269, 2004.
- [7] P. Carr and D. Madan. Option valuation using the fast Fourier transform. *Journal of Computational Finance*, 2(4):61–73, 1999.
- [8] R. Cont and P. Tankov. *Financial Modelling with Jump Processes*. Chapman & Hall/CRC, 2004.
- [9] Delft High Performance Computing Centre (DHPC). DelftBlue Supercomputer (Phase 1). <https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase1>, 2022.
- [10] D. Duffie, J. Pan, and K. Singleton. Transform analysis and asset pricing for affine jump-diffusions. *Econometrica*, 68:1343–1376, 2000.
- [11] W. E and B. Yu. The deep Ritz method: A deep learning-based numerical algorithm for solving variational problems. *Communications in Mathematics and Statistics*, 6(1):1–12, 2018.
- [12] E. Eberlein and J. Kallsen. *Mathematical Finance*. Springer, 2019.
- [13] E. Eberlein, K. Glau, and A. Papapantoleon. Analysis of Fourier transform valuation formulas and applications. *Applied Mathematical Finance*, 17:211–240, 2010.
- [14] F. Fang and C. W. Oosterlee. A novel pricing method for European options based on Fourier-cosine series expansions. *SIAM Journal on Scientific Computing*, 31:826–848, 2008/09.
- [15] E. H. Georgoulis, M. Loulakis, and A. Tsiourvas. Discrete gradient flow approximations of high dimensional evolution partial differential equations via deep neural networks. *Communications in Nonlinear Science and Numerical Simulation*, 117:106893 (11), 2023.
- [16] A. Gnoatto, M. Patacca, and A. Picarelli. A deep solver for BSDEs with jumps, 2022. Preprint, arXiv:2211.04349.
- [17] M. Griebel and A. Hullmann. An efficient sparse grid Galerkin approach for the numerical valuation of basket options under Kou’s jump-diffusion model. In *Sparse Grids and Applications*, pages 121–150. Springer, 2013.
- [18] J. Han, A. Jentzen, and W. E. Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences*, 115(34):8505–8510, 2018.

- [19] P. Hepperger. Option pricing in Hilbert space-valued jump-diffusion models using partial integro-differential equations. *SIAM Journal on Financial Mathematics*, 1:454–489, 2010.
- [20] S. G. Kou. A jump-diffusion model for option pricing. *Management Science*, 48:1086–1101, 2002.
- [21] I. E. Lagaris, A. Likas, and D. I. Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE Transactions on Neural Networks*, 9(5):987–1000, 1998.
- [22] Y. Liao and P. Ming. Deep Nitsche method: Deep Ritz method with essential boundary conditions. *Communications in Computational Physics*, 29(5):1365–1384, 2019.
- [23] R. Merton. Option pricing when underlying stock returns are discontinuous. *Journal of Financial Economics*, 3:125–144, 1976.
- [24] A. B. Owen. Scrambling Sobol' and Niederreiter–Xing points. *Journal of Complexity*, 14:466–489, 1998.
- [25] M. S. Park, C. Kim, H. Son, and H. J. Hwang. The deep minimizing movement scheme. *Journal of Computational Physics*, 494:112518 (23), 2023.
- [26] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.
- [27] O. Reichmann and C. Schwab. Numerical analysis of additive, Lévy and Feller processes with applications to option pricing. In *Lévy matters I*, pages 137–196. Springer, 2010.
- [28] W. J. Runggaldier. Jump-diffusion models. In S. T. Rachev, editor, *Handbook of Heavy Tailed Distributions in Finance*, pages 169–209. North-Holland, 2003.
- [29] W. Schoutens. *Lévy Processes in Finance*. Wiley, 2003.
- [30] J. Sirignano and K. Spiliopoulos. DGM: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, 375:1339–1364, 2018.
- [31] I. Sobol'. On the distribution of points in a cube and the approximate evaluation of integrals. *USSR Computational Mathematics and Mathematical Physics*, 7(4):86–112, 1967.

THE MAXWELL INSTITUTE FOR MATHEMATICAL SCIENCES AND DEPARTMENT OF MATHEMATICS, HERIOT-WATT UNIVERSITY, EDINBURGH EH14 4AS, UNITED KINGDOM – & – DEPARTMENT OF MATHEMATICS, SCHOOL OF APPLIED MATHEMATICAL AND PHYSICAL SCIENCES, NATIONAL TECHNICAL UNIVERSITY OF ATHENS, 15780 ZOGRAFOU, GREECE – & – INSTITUTE OF APPLIED AND COMPUTATIONAL MATHEMATICS, FORTH, 70013 HERAKLION, GREECE

*Email address:* e.georgoulis@hw.ac.uk

DELMFT INSTITUTE OF APPLIED MATHEMATICS, TU DELFT, 2628 DELFT, THE NETHERLANDS – & – DEPARTMENT OF MATHEMATICS, SCHOOL OF APPLIED MATHEMATICAL AND PHYSICAL SCIENCES, NATIONAL TECHNICAL UNIVERSITY OF ATHENS, 15780 ZOGRAFOU, GREECE – & – INSTITUTE OF APPLIED AND COMPUTATIONAL MATHEMATICS, FORTH, 70013 HERAKLION, GREECE

*Email address:* a.papapantoleon@tudelft.nl

DEPARTMENT OF STATISTICS AND ACTUARIAL-FINANCIAL MATHEMATICS, UNIVERSITY OF THE AEGEAN, 83200 KARLOVASSI, SAMOS, GREECE – & – INSTITUTE OF APPLIED AND COMPUTATIONAL MATHEMATICS, FORTH, 70013 HERAKLION, GREECE

*Email address:* kesmarag@aegean.gr