A time-stepping deep gradient flow method for option pricing in (rough) diffusion models

Papapantoleon, Antonis; Rou, Jasper

DOI
10.1080/14697688.2025.2572318

Publication date
2025

Document Version
Final published version

Published in
Quantitative Finance

Quantitative Finance

ISSN: 1469-7688 (Print) 1469-7696 (Online) Journal homepage: www.tandfonline.com/journals/rquf20

A time-stepping deep gradient flow method for option pricing in (rough) diffusion models

Antonis Papapantoleon & Jasper Rou

To cite this article: Antonis Papapantoleon & Jasper Rou (17 Nov 2025): A time-stepping deep gradient flow method for option pricing in (rough) diffusion models, Quantitative Finance, DOI: 10.1080/14697688.2025.2572318

To link to this article: https://doi.org/10.1080/14697688.2025.2572318

Quantitative Finance, 2025
https://doi.org/10.1080/14697688.2025.2572318

A time-stepping deep gradient flow method for option pricing in (rough) diffusion models

ANTONIS PAPAPANTOLEON *†‡§ and JASPER ROU †

†Delft Institute of Applied Mathematics, EEMCS, TU Delft, 2628CD 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

(*Received 6 March 2024; accepted 19 September 2025*)

We develop a novel deep learning approach for pricing European options in diffusion models, that can efficiently handle high-dimensional problems resulting from Markovian approximations of rough volatility models. The option pricing partial differential equation is reformulated as an energy minimization problem, which is approximated in a time-stepping fashion by deep artificial neural networks. The proposed scheme respects the asymptotic behavior of option prices for large levels of moneyness, and adheres to *a priori* known bounds for option prices. The accuracy and efficiency of the proposed method is assessed in a series of numerical examples, with particular focus in the lifted Heston model.

Keywords: Option pricing; Diffusion models; Rough volatility models; Lifted volatility models; PDE; Time-stepping; Gradient flow; Artificial neural network

MSC Classifications: 91G20, 91G60, 68T07

1. Introduction

Stochastic volatility models have been popular in the mathematical finance literature because they allow to accurately model and reproduce the shape of implied volatility smiles for a single maturity. They require though certain modifications, such as making the parameters time- or maturity-dependent, in order to reproduce a whole volatility surface; see e.g. the comprehensive books by Gatheral (2012) or Bergomi (2016). The class of rough volatility models, in which the volatility process is driven by a *fractional* Brownian motion, offers an attractive alternative to classical volatility models, since they allow to reproduce many stylized facts of asset and option prices with only a few (constant) parameters; see e.g. the seminal articles by Gatheral *et al.* (2018) and Bayer *et al.* (2016), and the recent volume by Bayer *et al.* (2023). The presence of fractional Brownian motion in the dynamics of rough volatility models yields that the volatility process is not a semimartingale, and the model is not Markovian, which means that the simulation of the dynamics, or the pricing and hedging of derivatives, become challenging tasks. This has attracted increasing attention from the mathematical finance community and has led to the creation of several innovative methods to tackle these challenges.

Markovian approximations of fractional processes is a popular method for tackling the challenges arising in these models, since they allow to relate rough volatility models to their classical counterparts. Abi Jaber and El Euch (2019a, 2019b) have designed multifactor stochastic volatility models approximating rough volatility models and enjoying a Markovian structure. Zhu *et al.* (2021) have showed that the rough Bergomi model can be well-approximated by the forward-variance Bergomi model with appropriately chosen weights and mean-reversion speed parameters, which has the Markovian property. Bayer and Breneis (2023a, 2023b) have studied Markovian approximations of stochastic Volterra equations using an N-dimensional diffusion process defined as the solution to a system of (ordinary) stochastic differential equations. Cuchiero and Teichmann (2020) have provided existence, uniqueness and approximation results for Markovian lifts of affine rough volatility models of general (jump) diffusion type. Moreover, Bonesini *et al.* (2023) have proposed a theoretical framework that exploits convolution kernels to transform a Volterra path-dependent (non-Markovian) stochastic process into a standard (Markovian) diffusion process.

*Corresponding author. Email: a.papapantoleon@tudelft.nl

Novel simulation schemes have also been developed in order to speed up Monte Carlo methods for rough volatility models. Bennedsen *et al.* (2017) have developed a hybrid simulation scheme for Brownian semistationary processes, and applied this to the rough Bergomi model. This method was further refined by McCrickerd and Pakkanen (2018) who developed variance reduction techniques for conditionally log-normal models, and by Fukasawa and Hirano (2021) who optimized the use of random numbers through orthogonal projections. Moreover, Gatheral (2022) has developed hybrid quadratic-exponential schemes for the simulation of rough affine forward variance models, while Bayer and Breneis (2024) have developed efficient and accurate simulation schemes for the rough Heston model based on low-dimensional Markovian representations.

Deep learning and neural network methods have also been applied in the field of rough stochastic volatility modeling. Horvath *et al.* (2021) presented a neural network-based method that performs the calibration task for the full implied volatility surface in (rough) volatility models. Jacquier and Oumgari (2023) introduced a deep learning–based algorithm to evaluate options in affine rough stochastic volatility models. Jacquier and Zuric (2023) constructed a deep learning-based numerical algorithm to solve path-dependent partial differential equations arising in the context of rough volatility.

Many other methods have been utilized and applied to rough stochastic volatility models. Let us mention here, among others, that El Euch and Rosenbaum (2019) have computed the characteristic function of the log-price in the rough Heston model, thus opening the door for the application of Fourier transform methods for option pricing. Bayer *et al.* (2020a) proposed the application of adaptive sparse grids and quasi Monte Carlo methods for option pricing in the rough Bergomi model, while Bayer *et al.* (2020b) have applied Hairer's theory of regularity structures in order to analyze rough volatility models. In addition, Bayer *et al.* (2019), Horvath *et al.* (2019) and Jacquier and Pannier (2022), among others, have studied asymptotic properties of rough volatility models in various regimes.

The connection between partial differential equations (PDEs) and option pricing is deep and well-studied in the literature. Several methods have been developed in order to price options in diffusion or stochastic volatility models, especially in high-dimensional settings, using sparse grids, asymptotic expansions, finite difference and finite elements methods; see e.g. Reisinger and Wittum (2007), Düring and Fournié (2012), Hilber *et al.* (2005) and the comprehensive book by Hilber *et al.* (2013).

The recent advances in deep learning methods have also created an intense interest in the solution of PDEs by deep learning and neural network methods. Sirignano and Spiliopoulos (2018) proposed to solve high-dimensional PDEs by approximating the solution with a deep neural network which is trained to satisfy the differential operator, initial condition, and boundary conditions. Raissi *et al.* (2019) introduced physics informed neural networks, i.e. neural networks that are trained to solve supervised learning tasks while respecting any given law of physics described by general nonlinear PDEs. Van der Meer *et al.* (2022) discussed the choice of a norm in the loss function for the training of neural networks. Han *et al.* (2018) introduced a deep learning-based approach that can handle general high-dimensional parabolic PDEs by reformulating the PDE using backward stochastic differential equations and approximating the gradient of the unknown solution by neural networks. E and Yu (2018) proposed a deep learning-based method for numerically solving variational problems, particularly the ones that arise from PDEs. Liao and Ming (2019) proposed a method to deal with the essential boundary conditions encountered in the deep learning-based numerical solvers for PDEs. Georgoulis *et al.* (2023) considered the approximation of initial and boundary value problems involving high-dimensional, dissipative evolution PDEs using a deep gradient flow neural network framework. In similar spirit, Park *et al.* (2023) proposed a deep learning-based minimizing movement scheme for approximating the solutions of PDEs. The work of Georgoulis *et al.* (2023) was followed up by Georgoulis *et al.* (2024), who developed an implicit-explicit deep minimizing movement method for pricing European basket options written on assets that follow jump-diffusion dynamics. Let us refer the reader to the forthcoming book by Jentzen *et al.* (2023), who provide an introduction to the topic of deep learning algorithms and a survey of the (vast) literature.

The starting point for our research is the existence of a Markovian approximation for a rough volatility model or, more generally, another Markovian model that describes empirical data well; see e.g. Rømer (2022) and Abi Jaber and Li (2024). Inspired by Georgoulis *et al.* (2023), we consider the option pricing PDE, introduce a time-stepping discretization of the time derivative, and reformulate the spatial derivatives as an energy minimization problem; the main difference with Georgoulis *et al.* (2023) is the presence of the drift (first order) term, which we treat in an explicit fashion in the discretization. This gradient flow formulation of the PDE gives rise to a suitable cost function for the training, via stochastic gradient descent, of deep neural networks that approximate the solution of the PDE. The main advantages of this method are twofold: on the one hand, the neural network from the previous time step is a good initial guess for the neural network optimization in the current time step, thus reducing the number of training stages necessary. On the other hand, the energy formulation allows to consider only first-order derivatives, thus reducing the training time in each time step / training stage significantly. In addition, we utilize available information about the qualitative behavior of option prices in order to assist the training of the neural network. More specifically, the architecture of the neural network respects the asymptotic behavior of option prices for large levels of moneyness, and adheres to *a priori* known bounds for option prices. The empirical results show that the proposed method is accurate, with a small overall error in the order of 10^{-3} and comparable with other deep learning methods, while the training times are significantly smaller compared to similar methods and do not grow significantly with the dimension of the underlying space.

This article is organized as follows: in section 2, we revisit the option pricing PDE for Markovian diffusion models. In section 3, we explain how we develop our method for solving option pricing PDEs, with focus on the time discretization

and energy formulation. In section 4, we revisit classical models for asset prices, with particular focus in the lifted Heston model. In section 5, we explain certain details for the design and implementation of the method. In section 6, we present and discuss the outcome of our numerical experiments, while section 7 concludes this article.

2. Problem formulation

Let S denote the price process of a financial asset that evolves according to a (Markovian) diffusion model, and consider a European vanilla derivative on S with payoff $\Psi(S_T)$ at maturity time $T > 0$. Using the fundamental theorem of asset pricing and the Feynman–Kac formula, the price of this derivative can be written as the solution to a PDE in these models. Indeed, let $u : [0,T] \times \Omega \to \mathbb{R}$ denote the price of this derivative, where $\Omega \subseteq \mathbb{R}^{n+1}$ and t denotes the time to maturity. Then, u solves the general PDE

$$\begin{cases} u_t(t,x) + \mathcal{A}u(t,x) + ru(t,x) = 0, & (t,x) \in (0,T] \times \Omega, \\ u(0,x) = \Psi(x), & x \in \Omega, \end{cases} \tag{1}$$

where $\mathcal{A}$ is a second order differential operator of the form

$$\mathcal{A}u = -\sum_{i,j=0}^{n} a^{ij} \frac{\partial^2 u}{\partial x_i \partial x_j} + \sum_{i=0}^{n} \beta^i \frac{\partial u}{\partial x_i}. \tag{2}$$

The coefficients a^{ij}, β^i of the generator $\mathcal{A}$ are directly related to the dynamics of the stochastic process S and can, in general, depend on the time and the spatial variables. Their particular form will be clarified in the applications later; see section 4. Moreover, for the time being, let us also prescribe homogeneous Dirichlet boundary conditions on $\partial\Omega$.

We will later rewrite the PDE (1) as an energy minimization problem, therefore we need to split the operator in a symmetric and an asymmetric part. The symmetric part will then be associated to a Dirichlet energy. Using the product rule, we can rewrite $\mathcal{A}$ as follows

$$\begin{aligned} \mathcal{A}u &= \sum_{i=0}^{n} \beta^i \frac{\partial u}{\partial x_i} - \sum_{i,j=0}^{n} \frac{\partial}{\partial x_j}\left(a^{ij} \frac{\partial u}{\partial x_i}\right) + \sum_{i,j=0}^{n} \frac{\partial a^{ij}}{\partial x_j} \frac{\partial u}{\partial x_i} \\ &= \sum_{i=0}^{n} \left(\beta^i + \sum_{j=0}^{n} \frac{\partial a^{ij}}{\partial x_j}\right) \frac{\partial u}{\partial x_i} - \sum_{i,j=0}^{n} \frac{\partial}{\partial x_j}\left(a^{ij} \frac{\partial u}{\partial x_i}\right). \end{aligned}$$

Define $b^i = \beta^i + \sum_{j=0}^{n} \frac{\partial a^{ij}}{\partial x_j}$. Then, $\mathcal{A}$ takes the form

$$\mathcal{A}u = -\nabla \cdot (A \nabla u) + \mathbf{b} \cdot \nabla u, \tag{3}$$

where

$$A = \begin{bmatrix} a^{00} & a^{10} & \dots & a^{n0} \\ a^{01} & a^{11} & \dots & a^{n1} \\ \vdots & \vdots & \ddots & \vdots \\ a^{0n} & a^{1n} & \dots & a^{nn} \end{bmatrix} \quad \text{and} \quad \mathbf{b} = \begin{bmatrix} b^0 \\ b^1 \\ \vdots \\ b^n \end{bmatrix}. \tag{4}$$

Let us point out that, the order of the derivatives with respect to x_i and x_j in the second derivative term can be interchanged in (2), hence we can always choose $a^{ij} = a^{ji}$. Therefore, A is a symmetric matrix.

3. Energy minimization and a deep gradient flow method

Let us now explain how we will develop our deep neural network method for solving the PDE (1). We proceed in three steps. First, we discretize the time derivative in section 3.1 and introduce a time-stepping scheme. Second, we rewrite the solution of the PDE as an energy minimization problem in section 3.2, which gives rise to a suitable cost function. Third, we approximate the minimizer by a neural network, which is trained using stochastic gradient descent, in section 3.3

3.1. Time discretization

A key ingredient of our method is that we do not use time as an input variable for the neural network, thus training a global space-time network for solving the PDE, as in Sirignano and Spiliopoulos (2018) and Raissi *et al.* (2019). Instead, following Georgoulis *et al.* (2023), we wish to train a neural network for each time step. The main insight is that the neural network from the previous time step is a good initial guess for the neural network optimization in the current time step, thus reducing the number of training stages necessary.

Let us divide the time interval $(0,T]$ into N equally spaced intervals $(t_{k-1}, t_k]$, with $h = t_k - t_{k-1} = \frac{1}{N}$ for $k = 0, 1, \dots, N$. Let U^k denote the approximation to the solution of the PDE $u(t_k)$ at time step t_k, using the following time discretization scheme

$$\begin{aligned} &\frac{U^k - U^{k-1}}{h} - \nabla \cdot \left(A \nabla U^k\right) + \mathbf{b} \cdot \nabla U^k + rU^k = 0, \\ &U^0 = \Psi. \end{aligned}$$

In order to be able to rewrite the PDE as an energy minimization problem, we wish to treat the asymmetric part as a constant function. Therefore, we substitute the function U^k in the asymmetric part with its value from the previous time point, and thus consider the backward differentiation scheme

$$\begin{aligned} &\frac{U^k - U^{k-1}}{h} - \nabla \cdot \left(A \nabla U^k\right) + F\left(U^{k-1}\right) + rU^k = 0, \\ &U^0 = \Psi, \end{aligned} \tag{5}$$

where $F(u) = \mathbf{b} \cdot \nabla u$. Convergence analysis of this scheme appears in Akrivis *et al.* (1999).

REMARK 3.1 Let us point out that non-uniform grids can also be used for the discretization of time, in order to employ a finer grid initially when the solution is less smooth. The numerical experiments in section 4 show that this is not necessary in the examples we consider.

3.2. Variational formulation

We would like now to associate the discretized PDE in (5) with an energy functional, in order to define later a suitable cost function for the training of the neural network. In other words, we would like to find an energy functional $I(u)$ such that U^k is a critical point of I. Let us rewrite (5) as follows:

$$\begin{aligned} &\left(U^k - U^{k-1}\right) + h\left(-\nabla\cdot\left(A\nabla U^k\right) + rU^k + F\left(U^{k-1}\right)\right) = 0,\\ &U^0 = \Psi. \end{aligned} \tag{6}$$

Then, we have the following result.

LEMMA 3.2 *Assume that A is symmetric and positive semi-definite. Then, U^k solves* (6) *if and only if U^k minimizes*

$$\begin{aligned} I^k(u) = &\frac{1}{2}\left\|u - U^{k-1}\right\|^2_{L^2(\Omega)}\\ &+ h\left(\int_\Omega \frac{1}{2}\left((\nabla u)^\mathsf{T} A\nabla u + ru^2\right) + F\left(U^{k-1}\right)u\mathrm{d}x\right). \end{aligned}$$

Proof Let v be a smooth function that is zero on the boundary. Then, for all such v, consider the function

$$\begin{aligned} i^k(\tau) = I^k\left(U^k + \tau v\right) = &\int_\Omega \frac{1}{2}\left(U^k + \tau v - U^{k-1}\right)^2 \mathrm{d}x\\ &+ h\int_\Omega \frac{1}{2}\left(\left(\nabla\left(U^k + \tau v\right)\right)^\mathsf{T} A\nabla\left(U^k + \tau v\right)\right.\\ &\left. + r\left(U^k + \tau v\right)^2\right) + F\left(U^{k-1}\right)\left(U^k + \tau v\right)\mathrm{d}x, \end{aligned}$$

for $\tau\in\mathbb{R}$. U^k minimizes I^k if and only if $\tau = 0$ minimizes i^k. Therefore, we get

$$\begin{aligned} 0 = (i^k)'(0) = &\left[\int_\Omega \left(U^k + \tau v - U^{k-1}\right)v\,\mathrm{d}x\right.\\ &+ h\int_\Omega \frac{1}{2}\left((\nabla v)^\mathsf{T} A\nabla\left(U^k + \tau v\right)\right.\\ &\left.\left. + \left(\nabla\left(U^k + \tau v\right)\right)^\mathsf{T} A\nabla v\right)\right.\\ &\left. + r\left(U^k + \tau v\right)v + F\left(U^{k-1}\right)v\,\mathrm{d}x\right]_{\tau=0}\\ = &\int_\Omega \left(U^k - U^{k-1}\right)v + \frac{h}{2}\left((\nabla v)^\mathsf{T} A\nabla U^k + \left(\nabla U^k\right)^\mathsf{T} A\nabla v\right)\\ &+ hrU^k v + hF\left(U^{k-1}\right)v\,\mathrm{d}x\\ = &\int_\Omega \left(U^k - U^{k-1}\right)v + hA\nabla U^k\cdot\nabla v\\ &+ h\left(rU^k + F\left(U^{k-1}\right)\right)v\,\mathrm{d}x\\ = &\int_\Omega \left(\left(U^k - U^{k-1}\right) + h\left(-\nabla\cdot\left(A\nabla U^k\right)\right.\right.\\ &\left.\left. + rU^k + F\left(U^{k-1}\right)\right)\right)v\,\mathrm{d}x, \end{aligned}$$

where in the second to last equality we used that A is symmetric, and in the last equality we used the divergence theorem. This equality holds for all v if and only if (6) holds. The second derivative is positive; indeed,

$$\left(i^k\right)''(\tau) = (1 + rh)\int_\Omega v^2\,\mathrm{d}x + h\int_\Omega (\nabla v)^\mathsf{T} A\nabla v\,\mathrm{d}x > 0,$$

since A is positive semi-definite. Therefore, $\tau = 0$ is indeed the minimizer. ■

3.3. Neural network approximation

Using the results of the previous subsection, we define the following cost (loss) function for training the neural network:

$$\begin{aligned} L^k(u) = &\frac{1}{2}\left\|u - U^{k-1}\right\|^2_{L^2(\Omega)}\\ &+ h\int_\Omega \mathcal{E}[u]\,\mathrm{d}x + h\int_\Omega F\left(U^{k-1}\right)u\,\mathrm{d}x, \end{aligned}$$

where

$$\mathcal{E}[u] = \tfrac{1}{2}\left((\nabla u)^\mathsf{T} A\nabla u + ru^2\right),$$

denotes the respective Dirichlet energy.

Let $f^k(\mathbf{x};\theta)$ denote a neural network approximation of U^k with trainable parameters θ. Applying a Monte Carlo approximation to the integrals, the discretized cost functional takes the form

$$L^k(\theta;\mathbf{x}) = \frac{|\Omega|}{2M}\sum_{i=1}^M \left(f^k(\mathbf{x}_i;\theta) - f^{k-1}(\mathbf{x}_i)\right)^2 + hN^k(\theta;\mathbf{x}),$$

where

$$\begin{aligned} N^k(\theta;\mathbf{x}) = \frac{|\Omega|}{M}\sum_{i=1}^M \left[\frac{1}{2}\left(\left(\nabla f^k(\mathbf{x}_i;\theta)\right)^\mathsf{T} A\nabla f^k(\mathbf{x}_i;\theta)\right.\right.\\ \left.\left. + r\left(f^k(\mathbf{x}_i;\theta)\right)^2\right) + \left(\mathbf{b}\cdot\nabla f^{k-1}(\mathbf{x}_i)\right)f^k(\mathbf{x}_i;\theta)\right]. \end{aligned}$$

Here, M denotes the number of samples $\mathbf{x}_i$.

In order to minimize this cost function, we use a stochastic gradient descent-type algorithm, i.e. an iterative scheme of the form:

$$\theta_{n+1} = \theta_n - \alpha_n\nabla_\theta L^k(\theta_n;\mathbf{x});$$

more specifically, we will use the Adam algorithm (Kingma and Ba 2014). The hyper-parameter α_n is the step-size of our update, called the learning rate. An overview of the Time Deep Gradient Flow (TDGF) method appears in Algorithm 1.

Algorithm 1 Time Deep Gradient Flow method

1: Initialize θ_0^0.
2: Set $f^0(\mathbf{x};\theta) = \Psi(\mathbf{x})$.
3: **for** each time step $k = 1,\dots,N$ **do**
4: Initialize $\theta_0^k = \theta^{k-1}$.
5: **for** each sampling stage n **do**
6: Generate random points $\mathbf{x}_i$ for training.
7: Calculate the cost functional $L^k(\theta_n^k;\mathbf{x}_i)$ for the sampled points.
8: Take a descent step $\theta_{n+1}^k = \theta_n^k - \alpha_n\nabla_\theta L^k(\theta_n^k;\mathbf{x}_i)$.
9: **end for**
10: **end for**

4. Examples

Let us now outline the different models to which we will apply our method, and derive the appropriate form of the generator in the option pricing PDE for each of these models. In particular, we will treat the Black–Scholes model in section 4.1, the Heston model in section 4.2, and the lifted Heston model in section 4.3. Let us point out that, straightforward but tedious computations, yield that the operator A satisfies the assumptions of Lemma 3.2 for all three examples.

4.1. Black–Scholes model

In the Black and Scholes (1973) model, the dynamics of the stock price S follow a geometric Brownian motion:

$$\mathrm{d}S_t = rS_t\,\mathrm{d}t + \sigma S_t \mathrm{d}W_t, \quad S_0 > 0,$$

with $r, \sigma \in \mathbb{R}_+$ the risk free rate and the volatility respectively. The generator corresponding to these dynamics, in the form (2), equals

$$\mathcal{A}u = -\frac{1}{2}\sigma^2 x^2 \frac{\partial^2 u}{\partial x^2} - rx\frac{\partial u}{\partial x},$$

and, applying the chain rule, we have

$$\mathcal{A}u = -\frac{\partial}{\partial x}\left(\frac{1}{2}\sigma^2 x^2 \frac{\partial u}{\partial x}\right) + \sigma^2 x\frac{\partial u}{\partial x} - rx\frac{\partial u}{\partial x}.$$

Therefore, the operator $\mathcal{A}$ takes the form (3) with the coefficients in (4) provided by

$$\begin{aligned} a &= \tfrac{1}{2}\sigma^2 x^2, \\ b &= (\sigma^2 - r)x. \end{aligned}$$

4.2. Heston model

The Heston (1993) model is a popular stochastic volatility model with dynamics

$$\begin{aligned} \mathrm{d}S_t &= rS_t\,\mathrm{d}t + \sqrt{V_t}S_t\,\mathrm{d}W_t, \quad S_0 > 0, \\ \mathrm{d}V_t &= \lambda(\kappa - V_t)\,\mathrm{d}t + \eta\sqrt{V_t}\mathrm{d}B_t, \quad V_0 > 0. \end{aligned}$$

Here V is the variance process, W, B are correlated (standard) Brownian motions, with correlation coefficient ρ, and $\lambda, \kappa, \eta \in \mathbb{R}_+$. The option pricing PDE (1) in this model has two spatial variables (x, v), corresponding to the asset price and the variance, and the generator corresponding to these dynamics, in the form (2), equals

$$\begin{aligned} \mathcal{A}u = &-rx\frac{\partial u}{\partial x} - \lambda(\kappa - v)\frac{\partial u}{\partial v} - \frac{1}{2}x^2 v\frac{\partial^2 u}{\partial x^2} - \frac{1}{2}\eta^2 v\frac{\partial^2 u}{\partial v^2} \\ &- \rho\eta x v\frac{\partial^2 u}{\partial x \partial v}. \end{aligned}$$

Applying the chain rule again, we get

$$\begin{aligned} \mathcal{A}u = &-rx\frac{\partial u}{\partial x} - \lambda(\kappa - v)\frac{\partial u}{\partial v} - \frac{\partial}{\partial x}\left(\frac{1}{2}x^2 v\frac{\partial u}{\partial x}\right) \\ &+ xv\frac{\partial u}{\partial x} - \frac{\partial}{\partial v}\left(\frac{1}{2}\eta^2 v\frac{\partial u}{\partial v}\right) + \frac{1}{2}\eta^2\frac{\partial u}{\partial v} \\ &- \frac{\partial}{\partial x}\left(\frac{1}{2}\rho\eta x v\frac{\partial u}{\partial v}\right) + \frac{1}{2}\rho\eta v\frac{\partial u}{\partial v} - \frac{\partial}{\partial v}\left(\frac{1}{2}\rho\eta x v\frac{\partial u}{\partial x}\right) \\ &+ \frac{1}{2}\rho\eta x\frac{\partial u}{\partial x}. \end{aligned}$$

Therefore, the operator $\mathcal{A}$ takes the form (3) with the coefficients in (4) provided by

$$\begin{aligned} a^{00} &= \tfrac{1}{2}x^2 v, \\ a^{10} = a^{01} &= \tfrac{1}{2}\rho\eta x v, \\ a^{11} &= \tfrac{1}{2}\eta^2 v, \\ b^0 &= \left(-r + v + \tfrac{1}{2}\rho\eta\right)x, \\ b^1 &= \lambda(v - \kappa) + \tfrac{1}{2}\eta^2 + \tfrac{1}{2}\rho\eta v. \end{aligned}$$

4.3. Lifted Heston model

The Heston model requires difficult modifications, such as making the parameters time dependent, in order to adequately calibrate the model to financial market data. The rough Heston model, in which the volatility process is driven by a *fractional* Brownian motion, is an alternative where only a few (constant) parameters are required in order to adequately calibrate the model; see Gatheral *et al.* (2018) and Bayer *et al.* (2016). The dynamics of the variance process in the rough Heston model are described by

$$V_t = V_0 + \int_0^t K(t-s)\left(\lambda\left(\kappa - V_s\right)\mathrm{d}s + \eta\sqrt{V_s}\mathrm{d}B_s\right),$$

where $K(t) = \frac{t^{H-\frac{1}{2}}}{\Gamma\left(H+\frac{1}{2}\right)}$ is the fractional kernel and $H \in \left(0, \frac{1}{2}\right)$ is the Hurst index. This model is not Markovian, due to the presence of the fractional Brownian motion in the dynamics, hence the pricing and hedging of derivatives becomes a challenging task. In particular, we do not have access to a PDE of the form (1) for the rough Heston model.

The *lifted* Heston model is a Markovian approximation of the rough Heston model, see Abi Jaber and El Euch (2019b), that is interesting as a model in its own right; see also Abi Jaber (2019). The dynamics of the asset price in the lifted Heston model are the following:

$$\begin{aligned} \mathrm{d}S_t &= rS_t\,\mathrm{d}t + \sqrt{V_t^n}S_t\,\mathrm{d}W_t, \quad S_0 > 0, \\ \mathrm{d}V_t^{n,i} &= -\left(\gamma_i^n V_t^{n,i} + \lambda V_t^n\right)\mathrm{d}t + \eta\sqrt{V_t^n}\mathrm{d}B_t, \quad V_0^{n,i} = 0, \quad (7) \\ V_t^n &= g^n(t) + \sum_{i=1}^n c_i^n V_t^{n,i}, \quad (8) \\ g^n(t) &= V_0 + \lambda\kappa\sum_{i=1}^n c_i^n \int_0^t \mathrm{e}^{-\gamma_i^n(t-s)}\mathrm{d}s, \end{aligned}$$

where $\lambda, \eta, c_i^n, \gamma_i^n, V_0, \kappa \in \mathbb{R}_+$, and W, B are two correlated (standard) Brownian motions with correlation coefficient ρ.

The variance factors $V^{n,i}$ start from zero and share the same one-dimensional Brownian motion B.

The option pricing PDE (1) has $n+1$ spatial variables $(x, v_1, \dots, v_n)$ in the lifted Heston model, corresponding to the asset price S and the n variance processes $V^{n,i}$ in (7). The next section explains the sampling of the process V^n in (8).

Let us now derive the generator for this system of SDEs. We will denote by $V_i = V^{n,i}$, $f_S = \frac{\partial f}{\partial S}$ and $f_{V_i} = \frac{\partial f}{\partial V^{n,i}}$, and suppress the dependence on time for simplicity. Let $f(S, V_1, \dots, V_n) : \mathbb{R}^{n+1} \to \mathbb{R}$ be a $\mathcal{C}^2$-function. Using Itô's formula, we have that

$$\begin{aligned}
\mathrm{d}f\,(S, V_1, \dots, V_n) &= f_S\,\mathrm{d}S + \sum_{i=1}^n f_{V_i}\,\mathrm{d}V_i + \frac{1}{2} f_{SS}\,\mathrm{d}\langle S, S\rangle \\
&\quad + \sum_{i=1}^n f_{SV_i}\,\mathrm{d}\langle S, V_i\rangle + \frac{1}{2}\sum_{i,j=1}^n f_{V_iV_j}\,\mathrm{d}\langle V_i, V_j\rangle \\
&= f_S rS\,\mathrm{d}t - \sum_{i=1}^n f_{V_i}\left(\gamma_i^n V_i + \lambda V_t^n\right)\mathrm{d}t \\
&\quad + \frac{1}{2} f_{SS} V_t^n S^2\,\mathrm{d}t + \sum_{i=1}^n f_{SV_i}\eta\rho V_t^n S\,\mathrm{d}t \\
&\quad + \frac{1}{2}\sum_{i,j=1}^n f_{V_iV_j}\eta^2 V_t^n\,\mathrm{d}t \\
&\quad + \text{(local) martingale.}
\end{aligned}$$

Hence, the generator $\tilde{\mathcal{A}}$ of this system takes the form:

$$\begin{aligned}
\tilde{\mathcal{A}}f &= -rxf_x + \sum_{i=1}^n \left(\gamma_i^n v_i + \lambda V_t^n\right) f_{v_i} - \frac{1}{2} V_t^n x^2 f_{xx} \\
&\quad - \eta\rho V_t^n x \sum_{i=1}^n f_{xv_i} - \frac{\eta^2}{2} V_t^n \sum_{i,j=1}^n f_{v_iv_j}.
\end{aligned}$$

In order to bring this generator in the form (2), we need to switch to the time to maturity, denoted now $\tau = T - t$. We have to be particularly careful in this case, because the function g^n in V^n depends on the time t. Let us denote

$$\tilde{V}_\tau^n := g^n(T - \tau) + \sum_{i=1}^n c_i^n V_\tau^{n,i}.$$

Then, applying the chain rule again, we have that

$$\begin{aligned}
\mathcal{A}u &= -rxu_x + \sum_{i=1}^n \left(\gamma_i^n v_i + \lambda \tilde{V}_\tau^n\right) u_{v_i} - \left(\frac{1}{2}\tilde{V}_\tau^n x^2 u_x\right)_x \\
&\quad + \tilde{V}_\tau^n x u_x - \frac{1}{2}\sum_{i=1}^n \left(\eta\rho \tilde{V}_\tau^n x u_{v_i}\right)_x \\
&\quad + \frac{1}{2}\eta\rho\tilde{V}_\tau^n \sum_{i=1}^n u_{v_i} - \frac{1}{2}\sum_{i=1}^n \left(\eta\rho\tilde{V}_\tau^n x u_x\right)_{v_i} \\
&\quad + \frac{1}{2}\eta\rho x \sum_{i=1}^n c_i^n u_x - \sum_{i,j=1}^n \left(\frac{\eta^2}{2}\tilde{V}_\tau^n u_{v_i}\right)_{v_j} \\
&\quad + \frac{\eta^2}{2}\sum_{i,j=1}^n c_j^n u_{v_i} \\
&= \left(\tilde{V}_\tau^n x - rx + \frac{1}{2}\eta\rho x \sum_{i=1}^n c_i^n\right) u_x \\
&\quad + \sum_{i=1}^n \left(\gamma_i^n v_i + \lambda\tilde{V}_\tau^n + \frac{1}{2}\eta\rho\tilde{V}_\tau^n + \frac{\eta^2}{2}\sum_{j=1}^n c_j^n\right) u_{v_i} \\
&\quad - \left(\frac{1}{2}\tilde{V}_\tau^n x^2 u_x\right)_x - \frac{1}{2}\sum_{i=1}^n \left(\eta\rho\tilde{V}_\tau^n x u_{v_i}\right)_x \\
&\quad - \frac{1}{2}\sum_{i=1}^n \left(\eta\rho\tilde{V}_\tau^n x u_x\right)_{v_i} - \sum_{i,j=1}^n \left(\frac{\eta^2}{2}\tilde{V}_\tau^n u_{v_i}\right)_{v_j}.
\end{aligned}$$

Therefore, the operator $\mathcal{A}$ takes the form (3) with the coefficients in (4) provided by

$$\begin{aligned}
a^{00} &= \frac{1}{2}\tilde{V}_\tau^n x^2, \\
a^{i0} = a^{0i} &= \frac{1}{2}\eta\rho\tilde{V}_\tau^n x, \quad i = 1, \dots, n, \\
a^{ij} &= \frac{\eta^2}{2}\tilde{V}_\tau^n, \quad i, j = 1, \dots, n, \\
b^0 &= \left(\tilde{V}_\tau^n - r + \frac{1}{2}\eta\rho\sum_{i=1}^n c_i^n\right) x, \\
b^i &= \left(\gamma_i^n v_i + \lambda\tilde{V}_\tau^n + \frac{1}{2}\eta\rho\tilde{V}_\tau^n + \frac{\eta^2}{2}\sum_{j=1}^n c_j^n\right), \\
&\quad i = 1, \dots, n.
\end{aligned}$$

5. Implementation details

Let us now describe some details about the design of the neural network architecture and the implementation of the numerical method. Motivated by Georgoulis *et al.* (2024), we would like to use information about the option price in order to facilitate the training of the neural network. On the one hand, using the lower no-arbitrage bound of a call option, i.e. that $u(t,x) \geq (x - K\,\mathrm{e}^{-rt})^+$, the neural network will learn the difference between the option price and this bound, instead of the option price itself. This difference turns out to be an easier function to approximate than the option price.

On the other hand, after some large level of asset price/moneyness, the option price will grow linearly with the asset price/moneyness. Let us denote this level by x_p. Therefore, instead of letting the neural network learn the option price at points larger than x_p, we will approximate this price by adding the difference between x_p and this point to the option value at x_p. In other words,

$$u(x_p + y; \theta) = u(x_p; \theta) + y, \quad y > 0.$$

This is consistent with the imposition of Dirichlet boundary conditions on Ω.

Moreover, we will consider the moneyness $\frac{S}{K}$ as an input variable instead of treating the asset price S and the strike price K separately, in order to reduce the size of the parameter space.

The architecture of the neural network for the TDGF method is inspired by the Deep Galerkin method (DGM) of Sirignano and Spiliopoulos (2018), hence we call the hidden layers 'DGM layers'. Overall, we set the following:

$$
\begin{aligned}
&\text{input} \qquad X^0 = \sigma_1\left(W^1\mathbf{x} + b^1\right),\\
&\quad \text{DGM layer}\\
&\qquad | \quad Z^l = \sigma_1\left(U^{z,l}\mathbf{x} + W^{z,l}X^{l-1} + b^{z,l}\right),\\
&\qquad\qquad l = 1,\dots,L,\\
&\qquad | \quad G^l = \sigma_1\left(U^{g,l}\mathbf{x} + W^{g,l}X^{l-1} + b^{g,l}\right),\\
&\qquad\qquad l = 1,\dots,L,\\
&\qquad | \quad R^l = \sigma_1\left(U^{r,l}\mathbf{x} + W^{r,l}X^{l-1} + b^{r,l}\right),\\
&\qquad\qquad l = 1,\dots,L,\\
&\qquad | \quad H^l = \sigma_1\left(U^{h,l}\mathbf{x} + W^{h,l}\left(X^{l-1}\odot R^l\right) + b^{h,l}\right),\\
&\qquad\qquad l = 1,\dots,L,\\
&\qquad \lfloor \quad X^l = \left(1 - G^l\right)\odot H^l + Z^l \odot X^{l-1},\\
&\qquad\qquad l = 1,\dots,L,\\
&\text{output} \quad f(x;\theta) = \left(x - K\mathrm{e}^{-r\tau}\right)^+ + \sigma_2\left(WX^L + b\right).
\end{aligned}
$$

Here, L is the number of hidden layers, σ_i is the activation function for $i = 1, 2$, and $\odot$ denotes the element-wise multiplication. Moreover, let us point here that $\mathbf{x}$ denotes the vector of inputs, which are the asset price and the variances (in the stochastic volatility models), and then $\mathbf{x}_1 = x$. In the numerical experiments, we have used 3 layers and 50 neurons per layer. The activation functions we have selected are the hyperbolic tangent function, $\sigma_1(x) = \tanh(x)$, and the softplus function, $\sigma_2(x) = \log(\mathrm{e}^x + 1)$, which guarantees that the option price remains above the no-arbitrage bound.

We consider a maturity of $T = 1.0$ year, and use the parameters sets for the Heston and the lifted Heston model from Heston (1993, table 1) and Abi Jaber and El Euch (2019b, page 321). We set the number of time steps equal to $N = 100$, use 2000 sampling stages in each time step, while in each sampling stage we take 600 samples per dimension, i.e. $600(n + 1)$ samples, following the recommendations in Georgoulis *et al.* (2023). Here, n denotes the number of variance processes, in the case of stochastic volatility models. As for the sampling domain, we consider the moneyness $x \in [0.01, 3.0]$, the Heston volatility $V \in [0.001, 0.1]$ and the lifted Heston volatilities $V^{n,i} \in [-V_i^{\text{high}}, V_i^{\text{high}}]$, with

$$
V_i^{\text{high}} = 3\sqrt{\frac{\eta^2 V_0}{2\gamma_i^n}\left(1 - \mathrm{e}^{-2\gamma_i^n T}\right)}.
$$

After sampling $V^{n,i}$ uniformly, we calculate V^n and discard all combinations that result in a negative V^n. Moreover, we set $x_p = 2.0$ In the optimization stage, we use the Adam algorithm (Kingma and Ba 2014) with a learning rate $\alpha = 3 \times 10^{-4}$, $(\beta_1, \beta_2) = (0.9, 0.999)$ and zero weight decay. Finally, the training was performed on the DelftBlue supercomputer (Delft High Performance Computing Centre (DHPC) 2022), using a single NVidia Tesla V100S GPU.

6. Numerical results

Let us now present and discuss the outcome of certain numerical experiments involving all models outlined in section 4. We are interested in both the accuracy, in section 6.1, and the speed, in section 6.2, of the numerical method developed in this article.

We will compare the TDGF method with another popular deep learning method for solving PDEs, the DGM of Sirignano and Spiliopoulos (2018). In the DGM approach, the solution of the PDE is translated into a quadratic minimization problem, which is minimized using stochastic gradient descent. In our setting, this minimization takes the form:

$$
\begin{aligned}
&\|u_t - \nabla\cdot(A\nabla u) + \mathbf{b}\cdot\nabla u + ru\|^2_{L^2([0,T]\times\Omega)}\\
&\quad + \|u(0,\mathbf{x}) - \Psi(x)\|^2_{L^2(\Omega)} \to \min.
\end{aligned}
$$

Let us recall that, for large levels of asset price/moneyness, the option price grows linearly with the asset price/moneyness. Therefore, we add a loss term punishing the derivative of the option price with respect to the stock deviating from 1 at $x = 3.0$.

In order to ensure a fair comparison between the two methods, we use 200,000 sampling stages for the DGM. In each stage, we use $600(n + 2)$ samples for the PDE in the interior domain and $600(n + 1)$ samples for the initial and boundary conditions. Moreover, we use the same network architecture as for the TDGF with the same number of layers and neurons per layer, and also use the Adam optimizer.

In order to compare the accuracy of the two methods, we need a reference value. In the Black–Scholes model, in the case of a European call option, the option pricing PDE has an exact solution:

$$
u(t,x) = x\Phi(d_1) - K\,\mathrm{e}^{-r\tau}\Phi(d_2),
$$

with Φ the cumulative standard normal distribution function, $\tau = T - t$ the time to maturity,

$$
d_1 = \frac{\log\left(\frac{x}{K}\right) + \left(r + \frac{\sigma^2}{2}\right)\tau}{\sigma\sqrt{\tau}} \quad \text{and} \quad d_2 = d_1 - \sigma\sqrt{\tau}.
$$

In the Heston model, the option pricing PDE does not admit an analytical solution. However, the model is affine and the characteristic function does have an analytical expression; see Heston (1993). Using the characteristic function, we compute the reference price with the COS method of Fang and Oosterlee (2009), with $N = 512$ terms and truncation domain $[-10\tau^{\frac{1}{4}}, 10\tau^{\frac{1}{4}}]$.

Finally, in the lifted Heston model, the characteristic function does not have an analytical expression but, as this model is again affine, the characteristic function is known

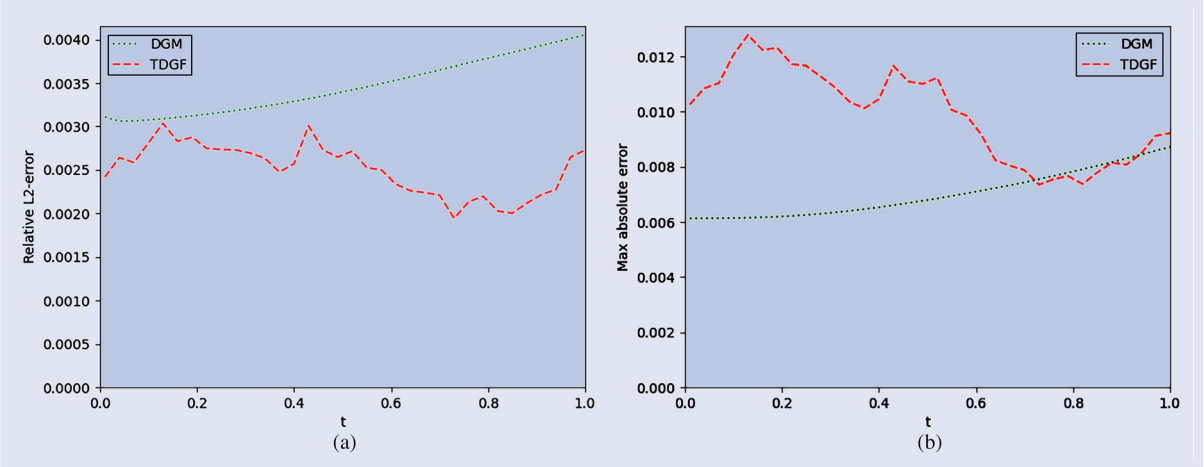

Figure 1. Errors of the two methods in the Black–Scholes model against time, with interest rate $r = 0.05$ and volatility $\sigma = 0.25$. (a) Relative L^2-error and (b) Maximal absolute error.

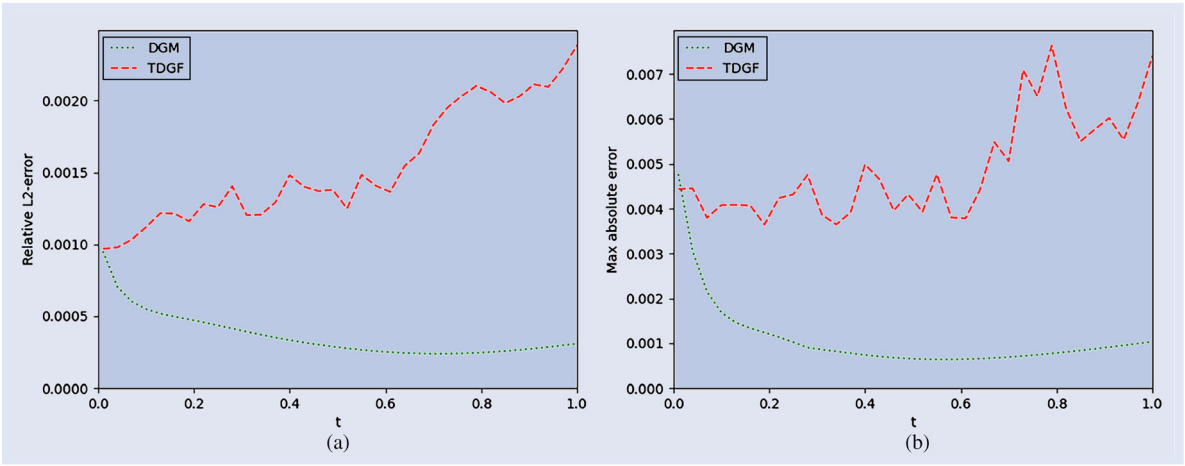

Figure 2. Error of the two methods in the Heston model against time, with $r = 0.0$ and $\eta = 0.1$, $\rho = 0.0$, $\kappa = 0.01$, $V_0 = 0.03$ and $\lambda = 2.0$. (a) Relative L^2-error and (b) Maximal absolute error.

up to the solution of a system of Riccati equations; see Abi Jaber (2019). We solve these equations using an implicit-explicit discretization scheme with 500 time steps, after which we apply the COS method again to compute the reference price.

6.1. Accuracy

In the Black–Scholes model, we have only one input variable, the asset price S. Figure 1 presents the relative L^2- and the absolute maximum error for the TDGF and the DGM methods in the Black–Scholes model. These errors are calculated over 47 evenly spaced grid points in the interval $[0.01, 3]$. Both methods have a comparable (small) error.

In the Heston model, we have two input variables, the asset price S and the variance V. Figure 2 presents the relative L^2- and the absolute maximum error for the TDGF and the DGM methods in the Heston model. These figures are presented for fixed $V_0 = 0.03$, however other values of V_0 between 0.01 and 0.09 provide similar results. The DGM is more accurate than the TDGF method, although the TDGF is just as accurate as in the Black–Scholes model, and the error is any case small, in the order of 10^{-3}.

In the lifted Heston model, we have $n + 1$ input variables, the asset price S and the variances $V^{n,i}$, with $i = 1, \ldots, n$. Figure 3 presents the relative L^2- and the maximum absolute error for the TDGF and the DGM methods in this model, in the case $n = 1$, using the same parameters as in the Heston model before. The numerical results are comparable for the two methods, and both methods even seem to perform slightly better than in the previous example.

Figure 4 presents the relative L^2- and the maximum absolute error for the TDGF and the DGM methods again in the lifted Heston model, for the same parameter set as previously, but now with $n = 20$ variance factors. The errors of both methods are comparable and, although they have increased compared to the $n = 1$ case, they are still low, in the order of 10^{-3}.

Figure 5 presents the relative L^2- and the absolute maximum error for the two methods in the lifted Heston model, now for the parameter set suggested by Abi Jaber (2019) with $n = 1$. The main difference here is that the correlation between the asset price and the variance processes is no longer zero. The error in both methods has slightly increased with respect to the previous example, with the DGM performing

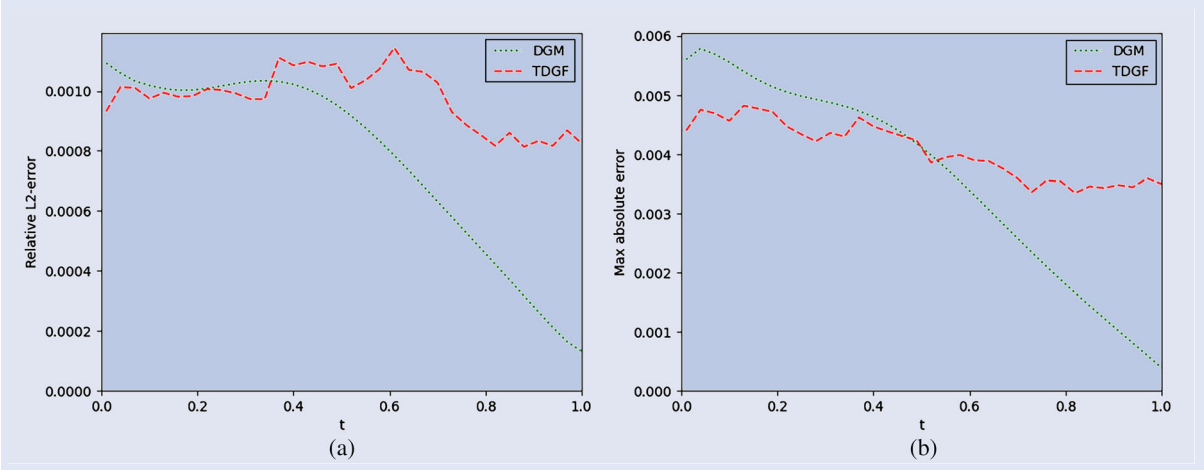

Figure 3. Errors of the two methods in the lifted Heston model with $n = 1$ variance process against time, with $r = 0.0$, $\eta = 0.1$, $\rho = 0.0$, $\kappa = 0.01$, $V_0 = 0.01$ and $\lambda = 2.0$. (a) Relative L^2-error and (b) Maximal absolute error.

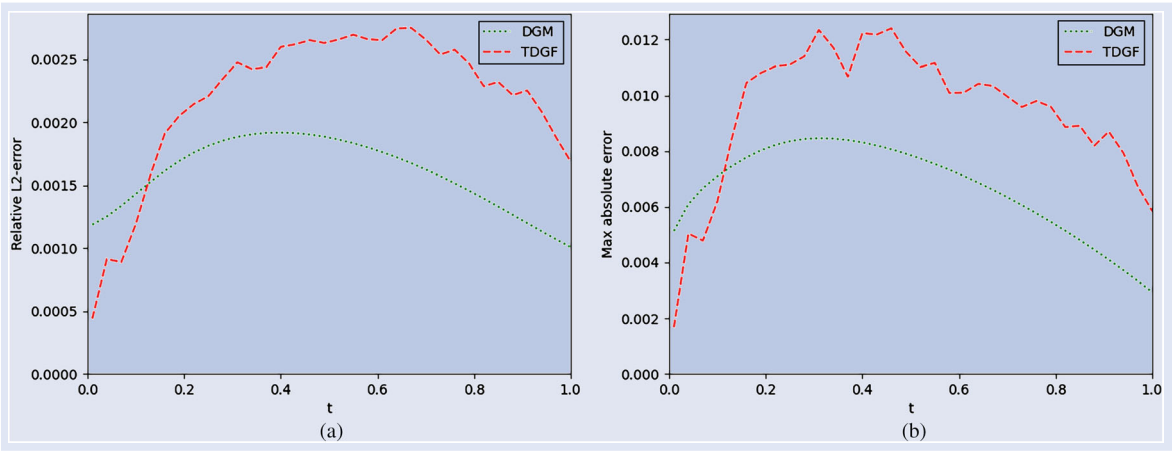

Figure 4. Errors of the two methods in the lifted Heston model with $n = 20$ variance process against time, with $r = 0.0$, $\eta = 0.1$, $\rho = 0.0$, $\kappa = 0.01$, $V_0 = 0.01$ and $\lambda = 2.0$. (a) Relative L^2-error and (b) Maximal absolute error.

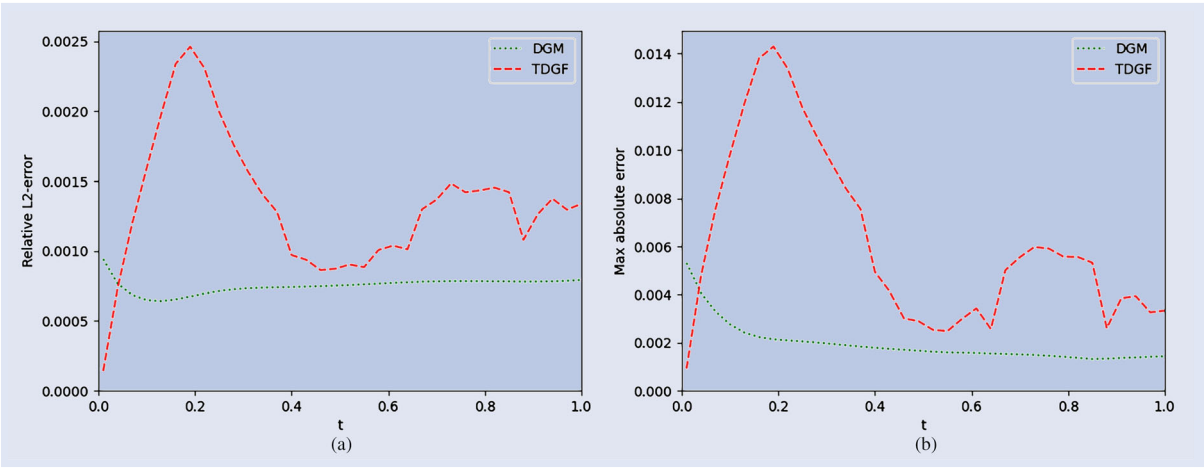

Figure 5. Errors of the two methods in the lifted Heston model with $n = 1$ variance process against time, with $r = 0.0$, $\eta = 0.3$, $\rho = -0.7$, $\kappa = 0.02$, $V_0 = 0.02$ and $\lambda = 0.3$. (a) Relative L^2-error and (b) Maximal absolute error.

slightly better than the TDGF. However, the overall error is still small, in the order of 10^{-3}.

Finally, figure 6 presents the relative L^2- and the absolute maximum error for the two methods in the lifted Heston model, for the same parameter set as in the previous two figures, but now with $n = 5$ and $\rho = -0.2$. The combination of increased dimension (i.e. 5 variance processes compared to one before) and decreased correlation results in the errors of both methods being similar to the previous case, and still small, in the order of 10^{-3}. However, as the number of

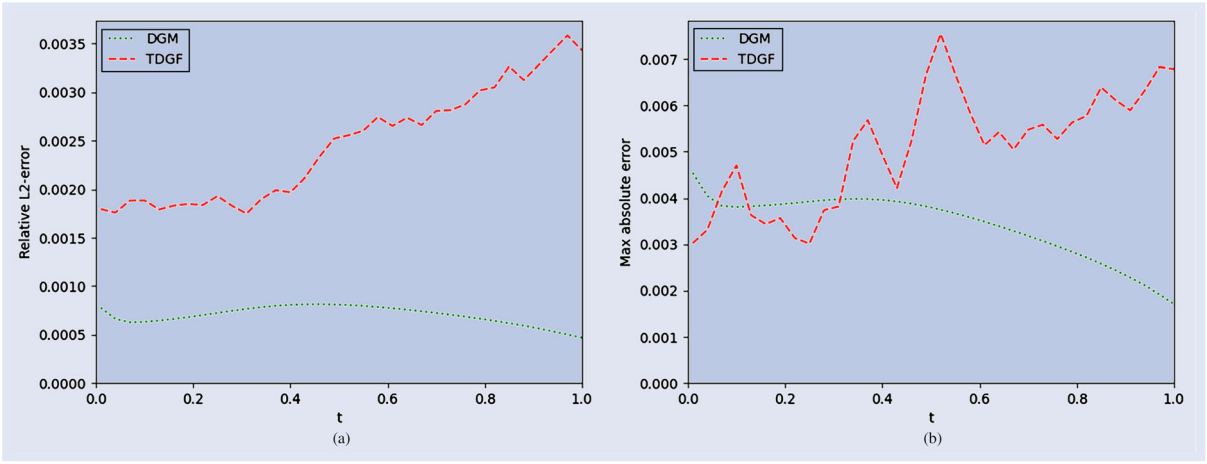

Figure 6. Errors of the two methods in the lifted Heston model with $n = 5$ variance processes against time, with $r = 0.0$, $\eta = 0.3$, $\rho = -0.2$, $\kappa = 0.02$, $V_0 = 0.02$ and $\lambda = 0.3$. (a) Relative L^2-error and (b) Maximal absolute error.

Table 1. Training time in seconds of the different methods for a European call option in the different models.

Model	Black–Scholes	Heston	ℓ-Heston, $n = 1$	ℓ-Heston, $n = 5$	ℓ-Heston, $n = 20$
DGM	7.0×10^3	12.2×10^3	12.5×10^3	21.0×10^3	54.3×10^3
TDGF	3.8×10^3	5.8×10^3	5.9×10^3	6.3×10^3	7.4×10^3

Table 2. Computational time in seconds of the different methods for a European call option in the different models.

Model	Black–Scholes	Heston	ℓ-Heston, $n = 1$	ℓ-Heston, $n = 5$	ℓ-Heston, $n = 20$
Exact/COS	0.0015	0.013	9.1	9.5	10.0
DGM	0.0086	0.0015	0.0043	0.0052	0.0015
TDGF	0.0018	0.0018	0.0019	0.0019	0.0018

variance processes increase while the correlation is non-zero, then the errors in both methods are increasing rapidly, and we can no longer accommodate e.g. 20 variance factors as we did in the uncorrelated case.

6.2. *Training and computational times*

Table 1 summarizes the training times for the TDGF and the DGM methods in the different models. As we have expected, due to the time stepping and the absence of a second derivative in the cost function, the training of the TDGF method is faster than for the DGM method. Moreover, the training time for the TDGF method does not increase as the dimension of the problem (i.e. the number of variance processes) increases. On the contrary, the training time for the DGM method increases significantly with the number of dimensions, and becomes more than 4 times slower in the lifted Heston model when the dimension of the variance processes increases from 1 to 20.

The computational times for the TDGF and the DGM methods in all models are presented in table 2. The computational times are taken as the average over 34 time points of computing the option prices on a grid of 47 moneyness points. Even though we take an average, there is still a high variance in the computational times. Both methods are significantly faster than the COS method in the lifted Heston model, in which we do not know the characteristic function explicitly and a system of Riccati equations needs to be solved for every evaluation of the characteristic function.

The possibility of splitting the computation of an option price in a slow, offline phase (training) and a very fast, online one (computation), is one of the advantages of deep learning methods such as the TDGF proposed here and the DGM. However, we should point out that when changing the model parameters, then the TDGF and the DGM need to be retrained, thus the overall time (training and computation) is slower than the evaluation of the COS method. The development of 'deep parametric PDE' methods as in Glau and Wunderlich (2022) would be an interesting avenue to explore in that respect. Finally, let us also mention that the TDGF requires more memory than the DGM, around 10–100 MB, as it has to store a separate neural network for each time step, whereas the DGM needs only one neural network for all times; the increase in speed though justifies this cost.

7. Conclusion

We have developed in this article a novel deep learning method for option pricing in diffusion models. Starting from the option pricing PDE, using a backward differentiation time-stepping scheme and results from the calculus of variations, we can rewrite the PDE as an energy minimization problem for a suitable energy functional. The time-stepping provides a good initial guess for the next step in the optimization procedure, while the energy formulation yields an equation that only has first derivatives, instead of second ones.

The energy minimization problem offers a suitable candidate for a cost function, and the PDE is solved by training a neural network using stochastic gradient descent-type optimizers (e.g. Adam). We have considered as examples for our method the Black–Scholes model, the Heston model and the lifted Heston model, and derived the appropriate representations for the PDE in all these cases.

Overall, the TDGF method developed here performs well in the numerical experiments, with an error in the order of 10^{-3}. Moreover, the training and computation (evaluation) times for the TDGF method are faster and more consistent compared to the popular DGM method. However, when the dimension of the variance processes in the lifted Heston model increases and the correlation increases simultaneously, then the error also increases. This increase in the error is due to the diffusion term becoming smaller, which means that the explicit term in the decomposition (5) becomes the dominant term. The development of suitable numerical methods for this regime will be the subject of future research.

Acknowledgments

The authors are grateful to Emmanuil Georgoulis and Costas Smaragdakis for several helpful discussions and advice during the work on this project. The authors also acknowledge the use of computational resources of the DelftBlue supercomputer, provided by the Delft High Performance Computing Centre (DHPC) (2022).

Disclosure statement

No potential conflict of interest was reported by the author(s).

Funding

AP gratefully acknowledges the financial support from 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).

ORCID

Antonis Papapantoleon http://orcid.org/0000-0002-9504-2822
Jasper Rou https://orcid.org/0009-0008-3712-9461

References

Abi Jaber, E., Lifting the Heston model. *Quant. Finance*, 2019, **19**, 1995–2013.

Abi Jaber, E. and El Euch, O., Markovian structure of the Volterra Heston model. *Stat. Probab. Lett.*, 2019a, **149**, 63–72.

Abi Jaber, E. and El Euch, O., Multifactor approximation of rough volatility models. *SIAM J. Financ. Math.*, 2019b, **10**, 309–349.

Abi Jaber, E. and Li, S.X., Volatility models in practice: Rough, path-dependent or markovian? Preprint, arXiv:2401.03345, 2024.

Akrivis, G., Crouzeix, M. and Makridakis, C., Implicit-explicit multistep methods for quasilinear parabolic equations. *Numer. Math.*, 1999, **82**, 521–541.

Bayer, C. and Breneis, S., Markovian approximations of stochastic Volterra equations with the fractional kernel. *Quant. Finance*, 2023a, **23**(1), 53–70.

Bayer, C. and Breneis, S., Weak markovian approximations of rough Heston. Preprint arXiv:2309.07023, 2023b.

Bayer, C. and Breneis, S., Efficient option pricing in the rough Heston model using weak simulation schemes. *Quant. Finance*, 2024, **24**(9), 1247–1261.

Bayer, C., Friz, P.K. and Gatheral, J., Pricing under rough volatility. *Quant. Finance*, 2016, **16**, 887–904.

Bayer, C., Friz, P.K., Gulisashvili, A., Horvath, B. and Stemper, B., Short-time near-the-money skew in rough fractional volatility models. *Quant. Finance*, 2019, **19**, 779–798.

Bayer, C., Ben Hammouda, C. and Tempone, R., Hierarchical adaptive sparse grids and quasi-Monte Carlo for option pricing under the rough Bergomi model. *Quant. Finance*, 2020a, **20**, 1457–1473.

Bayer, C., Friz, P.K., Gassiat, P., Martin, J. and Stemper, B., A regularity structure for rough volatility. *Math. Finance*, 2020b, **30**, 782–832.

Bayer, C., Friz, P.K., Fukasawa, M., Gatheral, J., Jacquier, A. and Rosenbaum, M., *Rough Volatility*, 2023 (Society for Industrial and Applied Mathematics: Philadelphia, PA).

Bennedsen, M., Lunde, A. and Pakkanen, M.S., Hybrid scheme for Brownian semistationary processes. *Finance Stoch.*, 2017, **21**, 931–965.

Bergomi, L., *Stochastic Volatility Modeling*, 2016 (CRC Press: Boca Raton, FL).

Black, F. and Scholes, M., The pricing of options and corporate liabilities. *J. Pol. Econ.*, 1973, **81**(3), 637–654.

Bonesini, O., Callegaro, G., Grasselli, M. and Pagès, G., Efficient simulation of a new class of Volterra-type SDEs. Preprint, arXiv:2306.02708, 2023.

Cuchiero, C. and Teichmann, J., Generalized Feller processes and Markovian lifts of stochastic Volterra processes: The affine case. *J. Evol. Equations*, 2020, **20**(4), 1301–1348.

Delft High Performance Computing Centre (DHPC), DelftBlue Supercomputer (Phase 1). https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase1, 2022.

Düring, B. and Fournié, M., High-order compact finite difference scheme for option pricing in stochastic volatility models. *J. Comput. Appl. Math.*, 2012, **236**, 4462–4473.

E, W. and Yu, B., The deep Ritz method: A deep learning-based numerical algorithm for solving variational problems. *Commun. Math. Stat.*, 2018, **6**, 1–12.

El Euch, O. and Rosenbaum, M., The characteristic function of rough Heston models. *Math. Finance*, 2019, **29**, 3–38.

Fang, F. and Oosterlee, C.W., A novel pricing method for European options based on Fourier-cosine series expansions. *SIAM J. Sci. Comput.*, 2009, **31**, 826–848.

Fukasawa, M. and Hirano, A., Refinement by reducing and reusing random numbers of the hybrid scheme for Brownian semistationary processes. *Quant. Finance*, 2021, **21**, 1127–1146.

Gatheral, J., *The Volatility Surface: A Practitioner's Guide*, 2012 (Wiley).

Gatheral, J., Efficient simulation of affine forward variance models. *Risk*, 2022, 1–6.

Gatheral, J., Jaisson, T. and Rosenbaum, M., Volatility is rough. *Quant. Finance*, 2018, **18**, 933–949.

Georgoulis, E.H., Loulakis, M. and 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.

Georgoulis, E.H., Papapantoleon, A. and Smaragdakis, C., A deep implicit-explicit minimizing movement method for option pricing in jump-diffusion models. Preprint, arXiv:2401.06740, 2024.

Glau, K. and Wunderlich, L., The deep parametric PDE method and applications to option pricing. *Appl. Math. Comput.*, 2022, **432**, 127355.

Han, J., Jentzen, A. and E, W., Solving high-dimensional partial differential equations using deep learning. *Proc. Natl. Acad. Sci.*, 2018, **115**(34), 8505–8510.

Heston, S.L., A closed-form solution for options with stochastic volatility with applications to bond and currency options. *Rev. Financ. Stud.*, 1993, **6**(2), 327–343.

Hilber, N., Matache, A.-M. and Schwab, C., Sparse wavelet methods for option pricing under stochastic volatility. *J. Comput. Finance*, 2005, **8**(4): 1–42,

Hilber, N., Reichmann, O., Schwab, C. and Winter, C., *Computational Methods for Quantitative Finance: Finite Element Methods for Derivative Pricing*, 2013 (Springer Science & Business Media: Berlin).

Horvath, B., Jacquier, A. and Lacombe, C., Asymptotic behaviour of randomised fractional volatility models. *J. Appl. Probab.*, 2019, **56**, 496–523.

Horvath, B., Muguruza, A. and Tomas, M., Deep learning volatility: A deep neural network perspective on pricing and calibration in (rough) volatility models. *Quant. Finance*, 2021, **21**, 11–27.

Jacquier, A. and Oumgari, M., Deep curve-dependent PDEs for affine rough volatility. *SIAM J. Financ. Math.*, 2023, **14**(2), 353–382.

Jacquier, A. and Pannier, A., Large and moderate deviations for stochastic Volterra systems. *Stoch. Process. Their Appl.*, 2022, **149**, 142–187.

Jacquier, A. and Zuric, Z., Random neural networks for rough volatility. Preprint, arXiv:2305.01035, 2023.

Jentzen, A., Kuckuck, B. and von Wurstemberger, P., *Mathematical Introduction to Deep Learning: Methods, Implementations, and Theory*, 2023, forthcoming. Available online at: https://arxiv.org/abs/2310.20360.

Kingma, D.P. and Ba, J., Adam: A method for stochastic optimization. Preprint, arXiv:1412.6980, 2014.

Liao, Y. and Ming, P., Deep Nitsche method: Deep Ritz method with essential boundary conditions. Preprint, arXiv:1912.01309, 2019.

McCrickerd, R. and Pakkanen, M.S., Turbocharging Monte Carlo pricing for the rough Bergomi model. *Quant. Finance*, 2018, **18**, 1877–1886.

Park, M.S., Kim, C., Son, H. and Hwang, H.J., The deep minimizing movement scheme. *J. Comput. Phys.*, 2023, **494**, 112518.

Raissi, M., Perdikaris, P. and Karniadakis, G.E., 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.

Reisinger, C. and Wittum, G., Efficient hierarchical approximation of high-dimensional option pricing problems. *SIAM J. Sci. Comput.*, 2007, **29**, 440–458.

Rømer, S.E., Empirical analysis of rough and classical stochastic volatility models to the SPX and VIX markets. *Quant. Finance*, 2022, **22**, 1805–1838.

Sirignano, J. and Spiliopoulos, K., DGM: A deep learning algorithm for solving partial differential equations. *J. Comput. Phys.*, 2018, **375**, 1339–1364.

Van der Meer, R., Oosterlee, C.W. and Borovykh, A., Optimally weighted loss functions for solving PDEs with neural networks. *J. Comput. Appl. Math.*, 2022, **405**, 113887.

Zhu, Q., Loeper, G., Chen, W. and Langrené, N., Markovian approximation of the rough Bergomi model for Monte Carlo option pricing. *Mathematics*, 2021, **9**(5), 528. https://doi.org/10.3390/math9050528.