DETRA NOTE

DETRA NOTE 2024-1

OPTION PRICING IN THE HESTON MODEL WITH PHYSICS INSPIRED NEURAL NETWORKS

Donatien Hainaut and Alex Casas

Detralytics

DISCLAIMER

Detralytics
Avenue du Boulevard 21 Box 5 - 1210 Bruxelles (Belgium)
1-7 Cour Valmy, Le Belvédère - 92800 Puteaux (France)
www.detralytics.com
info@detralytics.eu

Option pricing in the Heston model with Physics inspired neural networks

Donatien Hainaut [*]
UCLouvain - Detralytics
Alex Casas[†]
Detralytics

January 29, 2024

In absence of a closed form expression such as in the Heston model, the option pricing is computationally intensive when calibrating a model to market quotes. this article proposes an alternative to standard pricing methods based on physics-inspired neural networks (PINNs). A PINN integrates principles from physics into its learning process to enhance its efficiency in solving complex problems. In this article, the driving principle is the Feynman-Kac (FK) equation, which is a partial differential equation (PDE) governing the derivative price in the Heston model. We focus on the valuation of European options and show that PINNs constitute an efficient alternative for pricing options with various specifications and parameters without the need for retraining.

Keywords: neural networks, variable annuities, Feynman-Kac equation, life insurance

1 Introduction

A physics-inspired neural network (PINN) incorporates principles from physics into its learning process, enhancing its efficiency in solving complex scientific problems. Compared to existing approaches, PINNs reduce the reliance on large datasets and provide accurate predictions even with sparse or noisy measurements. Researchers have employed PINNs to address a diverse range of problems, including fluid dynamics, solid mechanics, heat transfer, and quantum mechanics.

In this study, we demonstrate that PINNs can be used for the valuation of options in the Heston model. In this framework, the stock price is governed by a geometric Brownian motion with stochastic volatility. Options are valued either by discrete Fourier transform (FFT) or by Monte Carlo simulations. During the calibration phase, the parameters of the Heston model are adjusted to best replicate market quotes of options. This step requires multiple option valuations and is therefore computationally intensive. To expedite the process, we propose a neural network model that can instantaneously price options with various features and parameters. The network is trained to solve the Feynman-Kac equation in multiple configurations and is called PINN for

[*]Postal address: Voie du Roman Pays 20, 1348 Louvain-la-Neuve (Belgium). E-mail to: donatien.hainaut(at)uclouvain.be
[†]Postal address: Rue Réaumur, 124, 75002 Paris. E-mail to: a.casas(at)detralytics.eu

this reason.

In summary, a PINN is an approximate solution of a non-linear partial differential equation (PDE) that describes the dynamics of a model. The neural network is trained by minimizing the approximation error at sampled points within the PDE domain and on its boundaries. This method originates from the work of Lee and Kang (1990) and has since been applied to various non-linear PDEs. For example, Raissi et al. (2019) used PINNs for solving two main classes of mathematical problems: data-driven solutions and data-driven discovery of partial differential equations. In the field of physics, Carleo and Troyer (2017) and Cai (2018) employed PINNs to accurately approximate quantum many-body wave functions. For further information and recent developments and applications of PINNs, we recommend referring to Cuomo et al. (2022) for a comprehensive literature review.

The literature on the valuation of financial derivatives using neural networks is relatively recent. Hejazi and Jackson (2016) developed a neural network to price and estimate the 'Greeks' for a large portfolio of variable annuities. Doyle & Groendyke (2019) priced and hedged equity-linked contracts using neural networks. They constructed a dataset of variable annuity prices with various features through Monte Carlo simulations. The neural network was then estimated by minimizing prediction errors using a scaled conjugate gradient descent. Sirignano and Spiliopoulos (2018) introduced a Deep Galerkin Method (DGM) based on a network architecture inspired by long short-term memory (LSTM) neural cells. Their approach was tested on a class of high-dimensional free boundary partial differential equations (PDEs) and on high-dimensional Hamilton-Jacobi-Bellman PDEs and Burgers' equation. Gatta et al. (2018) evaluated a suitable PINN for the pricing of American multi-asset options and proposed a novel algorithmic technique for free boundary training. Al-Aradi et al. (2022) extended the DGM in several directions to solve Fokker-Planck and Hamilton-Jacobi-Bellman equations. More recently, Jiang et al. (2023) demonstrated, under mild assumptions, the convergence of the Deep Galerkin and PINNs method for solving PDEs. Glau and Wunderlich (2022) formalized and analyzed the deep parametric PDE method for solving high-dimensional parametric partial differential equations.

A close alternative to PINN is developed in Weinan et al. (2017), where they solve parabolic PDEs in high dimensions using neural networks connected to BSDEs. This approach employs two separate networks for the solution and its gradient. They apply their algorithm to price options in a multivariate Black and Scholes model. Beck et al. (2019) introduce a similar method for solving high-dimensional PDEs based on a connection between fully nonlinear second-order PDEs and second-order backward stochastic differential equations. They illustrate the accuracy of their method with the Black and Scholes and Hamilton-Jacobi-Bellman equations. Using this principle, Barigou and Delong (2022) evaluate equity-linked life insurances with neural networks by solving a backward stochastic differential equation (BSDE). Unlike PINNs, BSDE methods rely on two networks instead of one, and any modification of contract specifications requires retraining.

Another valuation approach based on neural networks relies on simulations. Buehler et al. (2019) present a framework for pricing and hedging derivatives using neural networks that are trained on simulated sample paths of risk factors. In this case, the neural network approximates the optimal investment strategy, and the value is determined by averaging discounted payoffs obtained through simulations. Similarly, Horvath et al. (2021) studied the performance of deep hedging under rough volatility models, while Biagini et al. (2023) developed a neural network approximation using simulations for the superhedging price and replicating strategy. As with BSDE approaches, any modification of contract specifications requires retraining. For a comprehensive review of algorithms based on neural networks for stochastic control and PDEs in finance, we recommend referring to Germain et al. (2021).

In the financial and insurance industry, four dominant numerical methods are commonly used for pricing contingent claim contracts: Monte-Carlo (MC) simulations, bi- or trinomial trees, PDE solving, or Fast Fourier Transform (FFT) inversion. As the calibration of a model needs multiple valuations of options with different features, these methods are computationally intensive. In contrast, the valuation using a PINN, once it is trained, becomes nearly instantaneous. Another noteworthy feature is the ability to parameterize a PINN with model parameters and contract features such the maturity. This provides a quick solution pricing without the need to retrain the network for each setting. This presents a significant advantage compared to BSDE or deep hedging approaches.

This article makes the following contributions. Firstly, we introduce a specific type of neural network in which intermediate layers are fed with both the output of the previous layer and the initial input vector. Such a network better captures the non-linear behavior of option prices compared to classical feed-forward networks. Secondly, we develop a scaled and centered version of the Feynman-Kac equation ruling option prices in the Heston model. Thirdly, we train the network for various maturities of options and for a wide range of Heston parameters. In this sense, our model is offers a significant flexibility compared to BSDE or deep hedging methods, which require retraining for each specific setting of Heston parameters.

2 Heston model in a nutshell

We consider a financial market composed of two assets. The account earns a constant risk free rate r. The stock price, denoted by $(S_t)_{t\geq 0}$, is ruled by a geometric Brownian diffusion with a stochastic variance, $(V_t)_{t\geq 0}$. These price and variance processes are defined on a probability space $(\Omega, \mathcal{F}, \mathbb{P})$ associated to two independant Brownian motions under $\mathbb{P}$, noted $\tilde{\boldsymbol{W}}_t = \left(\tilde{W}_t^{(1)}, \tilde{W}_t^{(2)}\right)^\top_{t\geq 0}$. The state variables (S_t, V_t) are driven by the following stochastic differential equations (SDE's):

$$d\begin{pmatrix} S_t \\ V_t \end{pmatrix} = \begin{pmatrix} \left(r + \nu_S\sqrt{V_t}\right) S_t \\ \kappa\left(\gamma - \nu_V \frac{\sigma\sqrt{V_t}}{\kappa} - V_t\right) \end{pmatrix} dt + \begin{pmatrix} S_t\sqrt{V_t}\Sigma_S^\top \\ \sigma\sqrt{V_t}\Sigma_V^\top \end{pmatrix} d\tilde{\boldsymbol{W}}_t \,, \tag{1}$$

where κ, γ and σ are in $\mathbb{R}^+$. Σ_S, Σ_V are vectors such that $\Sigma = \left(\Sigma_S^\top, \Sigma_V^\top\right)$is the (upper) Choleski decomposition of the correlation matrix:

$$\Sigma = \begin{pmatrix} \Sigma_S^\top \\ \Sigma_V^\top \end{pmatrix} = \begin{pmatrix} \rho & \sqrt{1-\rho^2} \\ 0 & 1 \end{pmatrix} \quad , \quad \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix} = \Sigma\Sigma^\top \,,$$

where $\rho \in (-1, 1)$ is the correlation between the stock price and its variance. The parameters ν_S, ν_V tune the risk premiums $\left(\nu_S\sqrt{V_t}\,, -\nu_V\frac{\sigma\sqrt{V_t}}{\kappa}\right)$, of price and variance processes. We assume that costs of risk, noted $\boldsymbol{\theta} = (\theta_1, \theta_2)^\top$, are constant, i.e. the Brownian motions with drift under $\mathbb{P}$,

$$dW_t^{(j)} = d\tilde{W}_t^{(j)} + \theta_j dt \,,\; j = 1, 2.$$

are Brownian motions under the risk neutral measure, $\mathbb{Q}$. To keep identical dynamics under $\mathbb{P}$ and $\mathbb{Q}$, the risk premium parameters, stored in a vector $\boldsymbol{\theta} = (\theta_1, \theta_2)^\top$ are such that

$$\begin{aligned} \nu_S &= \rho\theta_1 + \sqrt{1-\rho^2}\theta_2 \,, \\ \nu_V &= -\theta_2 \,. \end{aligned}$$

Under the risk neutral measure $\mathbb{Q}$, the stock price and variance processes are therefore ruled by the following SDE's:

$$d\begin{pmatrix} S_t \\ V_t \end{pmatrix} = \begin{pmatrix} r\,S_t \\ \kappa\,(\gamma - V_t) \end{pmatrix} dt + \begin{pmatrix} S_t\sqrt{V_t}\Sigma_S^\top \\ \sigma\sqrt{V_t}\Sigma_V^\top \end{pmatrix} d\boldsymbol{W}_t\,, \tag{2}$$

where $\boldsymbol{W}_t = \left(W_t^{(1)}, W_t^{(2)}\right)_{t\geq 0}^\top$. In the next section, we recall the valuation equation ruling option prices.

3 Valuation equation, European payoffs

We consider a European financial derivative expiring at time T and promising a general payoff , $H(S_T)$, function of stock price and variance at maturity. The fair value, denoted by L_t is equal to the expected discounted cash-flows under the risk neutral measure $\mathbb{Q}$:

$$L_t = \mathbb{E}\left(e^{-r(T-t)}H(S_T)\,|\,\mathcal{F}_t\right).$$

In the numerical illustration, we consider a put option with a strike price K. For such a derivative, the payoff has the following form

$$H(S_T) := (K - S_T)_+ \,.$$

A physics-inspired neural network (PINN) integrates principles from physics, including partial differential equations (PDEs) governing the behavior of state variables. In a financial context, we adopt the Feynman-Kac (FK) equation as our guiding principle. The FK equation is a PDE satisfied by all assets traded in an arbitrage-free market. In this section, we construct this equation for the valuation of European options in the Heston model. In the following section, we utilize a neural network to solve it for various market parameters and contract features. To lighten future developments, we denote the vector of state variables by

$$\boldsymbol{y}_t = (S_t, V_t)^\top \,.$$

Under the risk neutral measure $\mathbb{Q}$, this multivariate process is ruled by the following SDE:

$$\mathrm{d}\boldsymbol{y}_t = \boldsymbol{\mu}_{\boldsymbol{y}}(t,\boldsymbol{y}_t)dt + \Sigma_{\boldsymbol{y}}(t,\boldsymbol{y}_t)d\boldsymbol{W}_t\,,$$

where $\boldsymbol{\mu}_{\boldsymbol{y}}(.)$ is a vector of dimension 2 and $\Sigma_{\boldsymbol{y}}(.)$ is a 2×2 matrix:

$$\boldsymbol{\mu}_{\boldsymbol{y}}(t,\boldsymbol{y}_t) = \begin{pmatrix} r\,S_t \\ \kappa\,(\gamma - V_t) \end{pmatrix}\,,\ \Sigma_{\boldsymbol{y}}(t,\boldsymbol{y}_t) = \begin{pmatrix} S_t\sqrt{V_t}\Sigma_S^\top \\ \sigma\sqrt{V_t}\Sigma_V^\top \end{pmatrix}.$$

We emphasize the dependence to parameters by denoting the contract price as follows:

$$L_t = V\left(t, \boldsymbol{y}_t\,|\,r,\kappa,\gamma,\sigma,\rho,T\right).$$

Under the assumption of arbitrage-free market, all traded securities, including derivatives, earn on average the risk free rate, i.e. :

$$\mathbb{E}\left(dL_{t+}\,|\,\mathcal{F}_t\right) = L_t\,r\,dt\,. \tag{3}$$

Let us respectively denote the gradient and the Hessian of $V(.)$ with respect to $\boldsymbol{y}$ by $\nabla_{\boldsymbol{y}}V$ and $\mathcal{H}_{\boldsymbol{y}}(V)$:

$$\nabla_{\boldsymbol{y}}V = \begin{pmatrix} \partial_S V \\ \partial_V V \end{pmatrix}\quad,\quad \mathcal{H}_{\boldsymbol{y}}(V) = \begin{pmatrix} \partial_{SS}V & \partial_{SV}V \\ \partial_{SV}V & \partial_{VV}V \end{pmatrix}.$$

Applying the Itô's lemma to $V(.)$ allows us to rewrite (3) as a partial differential valuation equation:

$$\begin{aligned} 0 =& \partial_t V - r\,V + \boldsymbol{\mu}_{\boldsymbol{y}}(t,\boldsymbol{y}_t)^\top \nabla_{\boldsymbol{y}} V \\ &+ \frac{1}{2}\mathrm{tr}\left(\Sigma_{\boldsymbol{y}}(t,\boldsymbol{y}_t)\Sigma_{\boldsymbol{y}}(t,\boldsymbol{y}_t)^\top \mathcal{H}_{\boldsymbol{y}}(V)\right) , \end{aligned} \tag{4}$$

where tr(.) is the trace operator. This last expression is called the Feynman-Kac (FK) equation. The boundary constraint on $V(.)$ at expiry is equal to

$$V\left(T, \boldsymbol{y}_T \,|\, r, \kappa, \gamma, \sigma, \rho, T\right) \;=\; H\left(S_T\right) . \tag{5}$$

For most of current European payoffs, the option value if the stock price falls to zero is simply the discounted value of a constant payoff. In this case, we have a lower boundary constraint on $V(.)$:

$$V\left(t, (0, V_t)^\top \,|\, r, \kappa, \gamma, \sigma, \rho, T\right) \;=\; e^{-r(T-t)} H\left(0\right) . \tag{6}$$

We approximate the solution of the FK equation (4) using a neural network that takes as input the time, state variables, and parameters of the Heston model. The mathematical definition of a neural network is revisited in the following section, but it's worth noting that a neural network employs bounded activation functions, such as sigmoid or hyperbolic tangents. To mitigate convergence issues associated with the vanishing gradient problem, we standardize and scale the network's inputs. This preprocessing step slightly modifies the FK equation. Let us define the vectors $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^2$ as follows

$$\begin{aligned} \boldsymbol{a} &= \left(a_S, a_V\right)^\top , \\ \boldsymbol{b} &= \left(b_S, b_V\right)^\top . \end{aligned} \tag{7}$$

$\boldsymbol{a}$ and $\boldsymbol{b}$ are chosen to normalize a random sample of state variables. This point is discussed in the next section. The vector of centered and scaled state variables is denoted by

$$\tilde{\boldsymbol{y}}_t \;=\; \boldsymbol{a} + \boldsymbol{b} \odot \boldsymbol{y}_t \,, \tag{8}$$

where $\odot$ is the elementwise product. In a similar manner, we center and scale the time with coefficients a_h and b_h that will depends on the horizon of valuation. The centered scaled time and durations are defined as follows

$$\begin{cases} \tilde{t} &= a_h + b_h t \,, \\ \tilde{T} &= a_h + b_h T \,. \end{cases} \tag{9}$$

The normalized state variables, $\tilde{\boldsymbol{y}}_t = \left(\tilde{S}_t, \tilde{V}_t\right)$ are ruled by the SDE:

$$d\tilde{\boldsymbol{y}}_t \;=\; \boldsymbol{\mu}_{\tilde{\boldsymbol{y}}}(t, \tilde{\boldsymbol{y}}_t)dt + \Sigma_{\tilde{\boldsymbol{y}}}(t, \tilde{\boldsymbol{y}}_t) d\boldsymbol{W}_t \,,$$

where $\mu_{\tilde{y}}(.)$ is a 2-vector and $\Sigma_{\tilde{\boldsymbol{y}}}(.)$ is a 2×2 matrix:

$$\boldsymbol{\mu}_{\tilde{\boldsymbol{y}}}(t,\tilde{\boldsymbol{y}}_t) = \begin{pmatrix} r\left(\tilde{S}_t - a_S\right) \\ \kappa\left(b_V\gamma + a_V - \tilde{V}_t\right) \end{pmatrix} , \; \Sigma_{\tilde{\boldsymbol{y}}}(t,\tilde{\boldsymbol{y}}_t) = \begin{pmatrix} \left(\tilde{S}_t - a_S\right)\sqrt{\frac{\tilde{V}_t - a_V}{b_V}}\Sigma_S^\top \\ \sigma\sqrt{b_V\left(\tilde{V}_t - a_V\right)}\Sigma_V^\top \end{pmatrix} .$$

The value of the contract may be seen as a function of time and rescaled state variables, $L_t = V(\tilde{t}, \tilde{\boldsymbol{y}}_t \,|\, r, \kappa, \gamma, \sigma, \rho, \tilde{T})$. The valuation equation (4) is then rewritten in rescaled form:

$$\begin{aligned} 0 =& b_h\,\partial_{\tilde{t}} V - r\,V + \boldsymbol{\mu}_{\tilde{\boldsymbol{y}}}(t,\tilde{\boldsymbol{y}}_t)^\top \nabla_{\tilde{\boldsymbol{y}}} V \\ &+ \frac{1}{2}\mathrm{tr}\left(\Sigma_{\tilde{\boldsymbol{y}}}(t,\tilde{\boldsymbol{y}}_t)\Sigma_{\tilde{\boldsymbol{y}}}(t,\tilde{\boldsymbol{y}}_t)^\top \mathcal{H}_{\tilde{\boldsymbol{y}}}(V)\right) , \end{aligned} \tag{10}$$

where $\nabla_{\tilde{\boldsymbol{y}}} V$ and $\mathcal{H}_{\tilde{\boldsymbol{y}}}(V)$ are respectively the gradient and the Hessian of V with respect to standardized state variables, $\tilde{\boldsymbol{y}}$. The rescaled version of the terminal and lower boundary constraints (10) and (6) are:

$$V\left(\tilde{T}, \tilde{\boldsymbol{y}}_T \,|\, r, \kappa, \gamma, \sigma, \rho, \tilde{T}\right) = H\left(\frac{\tilde{S}_T - a_S}{b_S}\right), \tag{11}$$

$$V\left(\tilde{t}, \left(a_S, \tilde{V}_T\right)^\top \,|\, r, \kappa, \gamma, \sigma, \rho, \tilde{T}\right) = e^{-r\frac{(\tilde{T}-\tilde{t})}{b_h}} H\left(0\right). \tag{12}$$

The next section explains how to solve Eq. (10) with a neural network.

4 Neural networks, European payoffs

We approximate the value function solving Equation (10) using a particular type of neural network. This network takes as input a vector of dimension 9, denoted by $\boldsymbol{z}$, containing the scaled time, state variables and parameters:

$$\boldsymbol{z} \;:=\; \left(\tilde{t}, \tilde{\boldsymbol{y}}_t, r, \kappa, \gamma, \sigma, \rho, \tilde{T}\right)^\top. \tag{13}$$

There is no systematic procedure for determining the optimal structure of a neural network. As proven by Hornik (1991), we know that single-layer neural networks can approximate regular functions arbitrarily well, but achieving reasonable accuracy may require a high number of neurons. An alternative approach is provided by feed-forward networks, where information flows forward through several neural layers. However, these networks often struggle to replicate non-linear functions with a limited number of layers. To address this challenge, Sirignano and Spiliopoulos (2018) proposed a variant of LSTM called the Deep Galerkin Network. While this structure replicates prices effectively, its calibration is time-consuming due to the complexity of neural cells.

In our case, we adopt a simpler architecture where intermediate layers receive inputs from the previous layer and the initial input vector. Figure 1 illustrates the structure of such networks, featuring 'skip connections.' This model falls into the category of residual neural networks, which have been successfully applied in various deep learning applications.

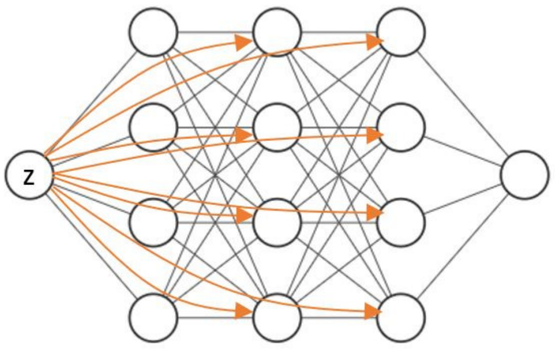

Figure 1: Feed forward network with skip connections toward intermediate layers.

Definition Let l, n_0, n_1, ..., $n_l \in \mathbb{N}$ be respectively the number of layers and neurons in each layer. n_0 is also the size of input vector. The activation function of layer $k = 1, 2, ..., l$ is noted $\phi_k(.) : \mathbb{R} \to \mathbb{R}$. Let $C_1 \in \mathbb{R}^{n_1} \times \mathbb{R}^{n_0}$,$\boldsymbol{c}_1 \in \mathbb{R}^{n_1}$, $C_k \in \mathbb{R}^{n_k} \times \mathbb{R}^{n_0+n_{k-1}}$, $\boldsymbol{c}_k \in \mathbb{R}^{n_k}$ for $k = 2, ..., l-1$,

$C_l \in \mathbb{R}^{n_l} \times \mathbb{R}^{n_{l-1}}$,$\boldsymbol{c}_l \in \mathbb{R}^{n_l}$ be neural weights defining the input, intermediate and output layers. We define the following functions

$$\begin{cases} \psi_k(\boldsymbol{x}) = \phi_k\left(C_k \boldsymbol{x} + \boldsymbol{c}_k\right) , & k = 1, l \\ \psi_k(\boldsymbol{x}, \boldsymbol{z}) = \phi_k\left(C_k \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{z} \end{pmatrix} + \boldsymbol{c}_k\right) , & k = 2, ..., l-1 \end{cases}$$

where activation functions $\phi_k(.)$ are applied componentwise. The neural network is a function $F : \mathbb{R}^{n_0} \to \mathbb{R}^{n_l}$ defined by

$$F(\boldsymbol{z}) \quad = \quad \psi_l \circ \psi_{l-1}\left(., \boldsymbol{z}\right) ... \circ \psi_2\left(., \boldsymbol{z}\right) \circ \psi_1(\boldsymbol{z}) \, .$$

After having chosen a network architecture, the model is trained by minimizing a loss function that is proportional to errors of approximation. This error is measured by replacing $V(.)$ with $F(.)$ in the scaled FK equation (10), at random points in the domain.

At time $t \in [0, T]$, the domain of the state vector, $\boldsymbol{y}_t = (S_t, V_t)^\top$is $\mathbb{R}^{+2}$. We approximate this domain by a closed convex subspace:

$$\mathcal{D}^{\boldsymbol{y}} \ : [S_l, S_u] \times [V_l, V_u] \, .$$

In order to fit the neural network, we draw a sample of $n_{\mathcal{D}}$ realizations of $\boldsymbol{y}_t$ in $\mathcal{D}^{\boldsymbol{y}}$, at random times. We also sample parameters $(r_j, \kappa_j, \gamma_j, \sigma_j, \rho_j, T_j)_{j=1,...,n_{\mathcal{D}}}$ under the constraints:

$$\begin{cases} r_j \in [r_l, r_u] \, , \, \kappa_j \in [\kappa_l, \kappa_u] \, , \\ \gamma_j \in [\gamma_l, \gamma_u] \, , \, \sigma_j \in [\sigma_l, \sigma_u] \, , \\ \rho_j \in (-1, 1) \, , \, T_j \in [0, T_{max}] \, , \\ t_j \le T_j \, . \end{cases}$$

The set of sampled state variables and parameters is noted $\mathcal{S}_{\mathcal{D}} = \left(t_j, \boldsymbol{y}_j, r_j, \kappa_j, \gamma_j, \sigma_j, \rho_j, T_j\right)_{j=1,...,n_{\mathcal{D}}}$. We next center and scale state variables and times. The mean and standard deviation of sampled state variables are computed with the standard formulas:

$$\bar{\boldsymbol{y}} = \frac{1}{n_{\mathcal{D}}} \sum_{j=1}^{n_{\mathcal{D}}} \boldsymbol{y}_j \quad , \quad \boldsymbol{S}_y = \sqrt{\frac{1}{n_{\mathcal{D}} - 1} \sum_{j=1}^{n_{\mathcal{D}}} \left(\boldsymbol{y}_j - \bar{\boldsymbol{y}}\right)^2} \, .$$

The scaling vectors (7) are defined by $\boldsymbol{a} = -\frac{\bar{\boldsymbol{y}}}{\boldsymbol{S}_{\boldsymbol{y}}}$ and $\boldsymbol{b} = \frac{1}{\boldsymbol{S}_{\boldsymbol{y}}}$. For $1, ..., n_{\mathcal{D}}$, we define $\tilde{\boldsymbol{y}_j} = \boldsymbol{a} + \boldsymbol{b} \odot \boldsymbol{y}_j$. The times, contract and bond maturities are also rescaled with relations (9) and coefficients

$$a_h = \quad -\tfrac{1}{2} \, , \quad b_h = \frac{1}{T_{max}} \, .$$

The set of sampled parameters and normalized state variables in the interior domain of the Feynman-Kac equation, is denoted by:

$$\tilde{\mathcal{S}}_{\mathcal{D}} = \left(\tilde{t}_j, \tilde{\boldsymbol{y}}_j, r_j, \kappa_j, \gamma_j, \sigma_j, \rho_j, \tilde{T}_j\right)_{j=1,...,n_{\mathcal{D}}} \, .$$

During the training phase, the error of approximation is measured for all points of this sample sets with the scaled FK equation (11). The error at expiry is measured with another sample set, denoted by $\tilde{\mathcal{S}}_T$, of n_T realizations of state variables and parameters:

$$\tilde{\mathcal{S}}_T = \left(\tilde{\boldsymbol{y}}_k, r_k, \kappa_k, \gamma_k, \sigma_k, \rho_k, \tilde{T}_k\right)_{k=1,...,n_T} \, .$$

Let us denote by $\mathbf{\Theta}$, the vector containing all neural weights $(C_k, \boldsymbol{c}_k)_{k=1,\dots,l}$. At points of $\tilde{\mathcal{S}}_{\mathcal{D}}$, we define for $j = 1, \dots, n_{\mathcal{D}}$, the error in Equation (10) when V is replaced by the neural network as follows:

$$\begin{aligned} e_j^{\mathcal{D}}(\mathbf{\Theta}) &= b_h\, \partial_{\tilde{t}_j} F - r\,F + \boldsymbol{\mu}_{\tilde{\boldsymbol{y}}}(t_j, \tilde{\boldsymbol{y}}_j)^\top \nabla_{\tilde{\boldsymbol{y}}} F \\ &\quad + \frac{1}{2} \mathrm{tr}\left(\Sigma_{\tilde{\boldsymbol{y}}}(t_j, \tilde{\boldsymbol{y}}_j) \Sigma_{\tilde{\boldsymbol{y}}}(t_j, \tilde{\boldsymbol{y}}_j)^\top \mathcal{H}_{\tilde{\boldsymbol{y}}}(F) \right) . \end{aligned} \tag{14}$$

We refer to $e_j^{\mathcal{D}}$ as the error on the inner domain since it measures the goodness of fit before expiry. The average quadratic loss on $\tilde{\mathcal{S}}_{\mathcal{D}}$ is the first component of the total loss function used to fit the neural network,

$$\mathcal{L}_D\left(\mathbf{\Theta}\right) = \frac{1}{n_{\mathcal{D}}} \sum_{j=1}^{n_{\mathcal{D}}} e_j^{\mathcal{D}}(\mathbf{\Theta})^2 . \tag{15}$$

Since the error $e_j^{\mathcal{D}}(\mathbf{\Theta})$ depends on first and second-order partial derivatives, special attention must be given to the accuracy of their calculation. Computing these derivatives using a standard finite difference method may introduce numerical instabilities. Therefore, we opt for an alternative approach known as automatic or algorithmic differentiation. Automatic differentiation leverages the fact that every computer calculation executes a sequence of elementary arithmetic operations and elementary functions, allowing us to compute partial derivatives with accuracy up to the working precision. Automatic differentiation is implemented in TensorFlow through functions such as GradientTape(.) and tape.gradient(.). For a more in-depth introduction and perspectives, we refer the reader to van Merriënboer et al. (2018).

On the terminal boundary sample set $\tilde{\mathcal{S}}_T$, we define the error $e_k^T(\mathbf{\Theta})$ for $k = 1, \dots, n_T$, as the difference between the output of the neural network and the payoff at expiry:

$$e_k^T(\mathbf{\Theta}) \quad = \quad F(\tilde{T}_k, \tilde{\boldsymbol{y}}_k, r_k, \kappa_k, \gamma_k, \sigma_k, \rho_k, \tilde{T}_k) - H\left(\frac{\tilde{S}_k - a_S}{b_S} \right) . \tag{16}$$

The average quadratic loss at expiry is the second component of the total loss function.

$$\mathcal{L}_T\left(\mathbf{\Theta}\right) = \frac{1}{n_T} \sum_{k=1}^{n_T} e_k^T(\mathbf{\Theta})^2 . \tag{17}$$

On the lower boundary sample set $\tilde{\mathcal{S}}_{low}$, we define the error $e_l^{low}(\mathbf{\Theta})$ for $l = 1, \dots, n_{low}$, as the difference between the output of the neural network and the discounted payoff at expiry when the stock price reaches its minimum:

$$e_l^{low}(\mathbf{\Theta}) \quad = \quad F(\tilde{t}_l, \left(a_S, \tilde{V}_l\right)^\top, r_l, \kappa_l, \gamma_l, \sigma_l, \rho_l, \tilde{T}_l) - e^{-r\frac{(\tilde{T}_l - \tilde{t}_l)}{b_h}} H\left(0\right) .$$

The average quadratic loss on this lower boundary is the third component of the total loss function.

$$\mathcal{L}_{low}\left(\mathbf{\Theta}\right) = \frac{1}{n_{low}} \sum_{l=1}^{n_{low}} e_l^T(\mathbf{\Theta})^2 . \tag{18}$$

The optimal network weights are found by minimizing the losses in $\tilde{\mathcal{S}}_{\mathcal{D}}$, $\tilde{\mathcal{S}}_T$ and $\tilde{\mathcal{S}}_{low}$,

$$\mathbf{\Theta}_{opt} = \arg\min_{\mathbf{\Theta}} \left[\mathcal{L}_D\left(\mathbf{\Theta}\right) + \mathcal{L}_T\left(\mathbf{\Theta}\right) + \mathcal{L}_{low}\left(\mathbf{\Theta}\right) \right] . \tag{19}$$

In practice, the minimization is performed with a gradient descent algorithm. For an introduction, we refer e.g. to Denuit et al. (2019), section 1.6.

5 Calibration and validation procedures

In the numerical illustration, we will compare put option prices computed by PINN's and Fast Fourier Transform (FFT). There are two alternatives for option pricing by FFT. The first one, proposed by Carr and Madan (2001), inverts numerically the characteristic function of option prices. A single run of this procedure returns option values for a wide range of strike prices. The accuracy of this method nevertheless depends on a tuning weight (α in the original paper), that ensures the existence of the Fourier transform. The optimal choice of this weight depends on Heston parameters and an inappropriate value generates numerical unstabilities. As we need prices for various and randomly generated Heston parameters, tuning the α is problematic. For this reason, we opt for a more robust alternative that approximates the probability density function (pdf) of the stock price by inverting its characteristic function. We briefly recall this method in the next subsection.

5.1 Option pricing by FFT

In the Heston model, the characteristic function of the log-return admits a closed-form expression provided in the next Proposition.

Proposition 1. *The characteristic function of* $\ln\left(S_s/S_0\right)|\mathcal{F}_t$ *under the risk neutral* $\mathbb{Q}$*, for* $s \geq t$ *with* $\omega \in \mathbb{C}$*, is given by the following expression*

$$\mathbb{E}^{\mathbb{Q}}\left(e^{\omega \ln(S_s/S_0)}\,|\,\mathcal{F}_t\right) = \left(\frac{S_t}{S_0}\right)^{\omega} \exp\left(A(\omega,t,s)+B(\omega,t,s)V_t\right). \tag{20}$$

Let us define the following constants:

$$\begin{cases} d = \sqrt{\left(\rho\sigma\omega-\kappa\right)^2+\sigma^2\left(\omega-\omega^2\right)}\,, \\ g = \frac{\kappa-\rho\sigma\omega+d}{\kappa-\rho\sigma\omega-d}\,. \end{cases}$$

The functions $A(\omega,t,s)$ *and* $B(\omega,t,s)$ *in Equation (20) are given by*

$$\begin{aligned} A(\omega,t,s) &= r\,\omega\,(s-t)+ \\ &\frac{\kappa\gamma}{\sigma^2}\left(\left(\kappa-\rho\sigma\omega+d\right)(s-t)-2\ln\left(\frac{1-ge^{d(s-t)}}{1-g}\right)\right), \end{aligned} \tag{21}$$

and

$$B(\omega,t,s) = \frac{\kappa-\rho\sigma\omega+d}{\sigma^2}\,\frac{1-e^{d\,(s-t)}}{1-g\,e^{d\,(s-t)}}\,. \tag{22}$$

For a proof, the reader can refer for instance, to Hainaut (2022), chapter 3, p. 65. European call or put options do not have analytical expressions. In order to evaluate these options, we calculate numerically the probability density function of the log-return, $\ln\left(S_T/S_0\right)|\mathcal{F}_t$, by a discrete Fourier transform (DFT). The characteristic function of a random variable, here $\Upsilon_{t,T}(i\omega)=\mathbb{E}^{\mathbb{Q}}\left(e^{i\,\omega\ln(S_T/S_0)}\,|\,\mathcal{F}_t\right)$, is also the inverse Fourier transform of its probability density function (pdf):

$$\begin{aligned} f_{t,T}(u) &= \frac{1}{2\pi}\int_{-\infty}^{+\infty}\Upsilon_{t,T}(i\omega)\,e^{-i\,u\,\omega}d\omega \\ &= \frac{1}{\pi}Re\left(\int_{0}^{+\infty}\Upsilon_{t,T}(i\omega)e^{-i\,u\,\omega}d\omega\right) \end{aligned} \tag{23}$$

Therefore, we can retrieve the pdf by computing numerically its Fourier transform as stated in the next proposition.

Proposition 2. *Let M be the number of steps used in the Discrete Fourier Transform (DFT) and $\Delta_u = \frac{2u_{max}}{M-1}$ be this step of discretization. Let us denote $\Delta_\omega = \frac{2\pi}{M\,\Delta_u}$* and

$$\omega_m \;=\; (m-1)\Delta_\omega,$$

for $m = 1...M$. *Let $\Upsilon_{t,T}(\omega) = \mathbb{E}^{\mathbb{Q}}\left(e^{\omega \ln(S_T/S_0)}\,|\,\mathcal{F}_t\right)$ be mgf of* $\ln(S_T/S_0)$*. The values of $f_{t,T}(\cdot)$ the pdf of* $\ln(S_T/S_0)\,|\mathcal{F}_t$ *at points $u_k = -\frac{M}{2}\Delta_u + (k-1)\Delta_u$ are approached by the sum:*

$$f(u_k) \;\approx\; \frac{2}{M\,\Delta_u} Re\left(\sum_{m=1}^{M} \varrho_m \Upsilon_{t,T}(i\,\omega_m)\,(-1)^{m-1} e^{-i\frac{2\pi}{M}(m-1)(k-1)}\right). \tag{24}$$

where $\varrho_m = \frac{1}{2}1_{\{m=1\}} + 1_{\{m\neq1\}} + \frac{1}{2}1_{\{m=M\}}$.

This result is proven by discretizing the integral (23). The value of a European option of maturity T and payoff $H(S_T)$ is then approached by the following sum

$$\mathbb{E}^{\mathbb{Q}}\left(e^{-r(T-t)}H(S_T)|\mathcal{F}_t\right) \;\approx\; \sum_{k=1}^{M} f(u_k)\,H(S_0 e^{u_k}). \tag{25}$$

Prices obtained by this method are compared to PINN prices. This method is sensitive to the choice of u_{max} and M. By construction, we have the constraint that $\Delta_\omega \Delta_u = \frac{2\pi}{M}$. For a given M, a small discretization step Δ_u for log-returns implies a large discretization step Δ_ω of frequencies and vice-versa. A reasonable accuracy can only be achieved if Δ_ω and Δ_u are small enough. In the numerical illustration, we set $M = 2^8$ and $u_{max} = 2.2$.

5.2 Learning dataset, calibration procedure and metrics

We estimate several PINN's for pricing European put options of maturities up to 5 years in various market conditions. During the training, the inner error is computed with $n_{\mathcal{D}}$ =20,000 combinations of parameters and state variables. These are randomly drawn from intervals reported in Table 1. The errors on the terminal and lower boundaries are both assessed with sample sets of sizes $n_T = 5,000$ and $n_{low} = 5,000$. The learning space, $\mathcal{S}^d$, is relatively large and cover various market conditions. Note that we have used the same data sets for all proposed neural architectures for an easy comparison of results.

Range of parameters and state variables	
$S_t \in [20, 180]$	$V_t \in [0.03^2, 0.5^2]$
$r_j \in [0.01, 0.07]$	$\kappa_j \in [0.5, 2]$
$\gamma_j \in [0.06^2, 0.4^2]$	$\sigma_j \in [0.1, 0.9]$
$\rho_j \in [-0.8, 0.8]$	$T_j \in [0, 5]$
$t_j \leq T_j$	

Table 1: Intervals from which parameters and state variables are generated.

The option strike is set to K= 100 but this is not a limitation. As the payoff is piecewise linear, we use the rule of thumb for valuing options with other strikes. Let us momentaneously denote the option price at time t by $V(S_t, K)$, for an asset value S_t and a strike K. The value of an option of strike $K^{'} \neq K$ is then equal to:

$$V(S_t, K^{'}) = \frac{K^{'}}{K} V\left(\frac{K}{K^{'}}S_t, K\right),$$

and can therefore be assessed with the network, without retraining. To compare the different neural architectures, we take care to apply a calibration protocol that is consistent across all tested configurations. It comprises four phases, with a decreasing learning rate and a variable number of iterations for each phase. The learning rates and the number of iterations per phase are detailed in Table 2. Network weights are optimized with the Adam algorithm which is a variant of the stochastic gradient descent.

Phase	Learning rate	Epochs
1	0.005	500
2	0.002	1,000
3	0.001	1,000
4	0.0001	,1000

Table 2: Learning rates and number of epochs used in the four training phases.

Decreasing the learning rate per phase reduces the number of iterations compared to a constant low learning rate. This reduction in the number of iterations shortens the model execution time, enabling relatively rapid calibration. Standardization of the calibration process allows for the comparison of errors at the end of 3,500 iterations. It is noteworthy that for some architectures, a larger number of iterations could potentially result in lower errors but we decided to use the same number of epochs for all networks. The comparison of networks is based on several performance metrics. The first ones are the total loss function and its components $\mathcal{L}_D$, $\mathcal{L}_T$, $\mathcal{L}_{low}$ after training. We also calculate the mean square error, denoted by MSE_{alea}, between PINN and FFT prices. This MSE is computed with a sample set of 5000 combinations of parameters and state variables. A third metric is the mean square error computed for a fixed set of Heston parameters, reported in Table 3, and random values $S_0 \in [20, 180]$, $T \in [0, 5]$. This error, denoted by MSE_{config}, is valued on a sample set of 5000 pairs (T, S_0). The last measure is the relative error for for in-the-money options (i.e. $S_t < K$). This error is denoted by $Err_{relative}$ and computed on a random set of 5000 combinations of Heston parameters and state variables.

MSE_{config}			
γ	0.15^2	r	0.03
κ	0.80	V_0	1.20 γ
σ	0.10	ρ	-0.40

Table 3: Market parameters for the calculation of the MSE with a single Heston configuration.

6 Numerical analysis

Tables 4 and 5 respectively report the values of losses and the performance metrics of tested network architectures. We observe that lower losses after 3500 training iterations are obtained with networks having a higher number of layers and neurons. This is consistent with the idea that increasing the number of neural weights allows greater freedom in optimizing the network. For a given number of epochs, the reduction in loss components is therefore greater for complex architectures. Based on the analysis of losses, we would prefer the network with 4 layers and 256 neurons. We also see that losses on the lower boundary are nearly null for all networks. The total loss is mainly due to errors on the inner domain and on the terminal boundary.

# of layers	# of neurons per layer	Total loss	Inner loss $\mathcal{L}_D$	Lower loss $\mathcal{L}_{low}$	T loss $\mathcal{L}_T$
2	32	0.411	0.234	0.013	0.164
3	32	0.364	0.210	0.011	0.143
4	32	0.289	0.163	0.009	0.117
2	64	0.307	0.183	0.008	0.116
3	64	0.191	0.122	0.004	0.065
4	64	0.139	0.085	0.004	0.051
2	128	0.175	0.103	0.004	0.068
3	128	0.104	0.064	0.002	0.038
4	128	0.066	0.044	0.001	0.021
2	256	0.119	0.067	0.003	0.049
3	256	0.050	0.030	0.001	0.018
4	256	0.049	0.030	0.001	0.017

Table 4: Total loss and its components

# of layers	# of neurons per layer	MSE_{alea}	MSE_{config}	$Err_{relative}$
2	32	2.611	2.477	0.046
3	32	2.892	2.505	0.043
4	32	2.195	0.924	0.040
2	64	2.877	2.977	0.045
3	64	2.583	1.274	0.045
4	64	2.622	0.907	0.043
2	128	2.583	1.199	0.043
3	128	2.509	1.031	0.044
4	128	2.462	0.700	0.044
2	256	2.206	0.394	0.042
3	256	2.373	0.651	0.044
4	256	2.346	0.689	0.044

Table 5: Mean squared errors and relative errors for the tested configurations

Table 5 reveals that the MSE_{alea}'s on a random set, range between 2.195 and 2.892. It seems that complex architectures with a low training loss do not necessary minimizes the MSE on a random sample. For instance, the network with 4 layers and 32 neurons achieves the lowest MSE_{alea} whereas its total calibration error reaches 0.289. We can partly explain this by the tendency of complex networks to overfit data. Ideally, we should increase the size of training sets proportionally to model complexity in order to limit overfitting. However, we haven't adapted the size of the training set to allow for a meaningful comparison. On the other hand, we should not forget that FFT prices are themselves numerical approximations of exact ones. This introduces a non-negligible bias in the calculation of MSE's. Furthermore for some Heston parameters, the FFT method fails to compute a price (due to an overflow encountered in the calculation of the characteristic function). The calculation of prices for these configurations requires a fine tuning of FFT parameters, case by case, which is not possible givent the size of the training set. The MSE are computed only with Heston parameters for which the FFT yields a price (i.e. for more than 90% of training data). For the Heston configuration of Table 3, we observe signification variation of MSE_{config} depending on the model complexity. According to this criterion, the best network has 2 layers of 256 neurons. the relative errors for in the money options are comparable and close to 4%. To understand from where come errors, we focus on a network with 4 layers of 256 neurons (4-256 network) and study its performance in greater depth.

6.1 Analysis of the 4-256 network, fixed market parameters

In this subsection, we consider a unique set of Heston parameters, presented in Table 6.

γ	0.20^2	r	0.04
κ	1.15	ρ	-0.40
σ	0.20		

Table 6: Market parameters for the analysis of the 4-256 network.

The left plot of Figure 2 compares FFT and PINN prices of 5 years European put when $V_0 = 0.25^2$. We observe that FFT and PINN prices are close whatever the stock value, S_0. For high value of S_0, PINN prices converges to zero at a faster pace than those computed by FFT. This is partly due to the numerical errors of the FFT algorithm.

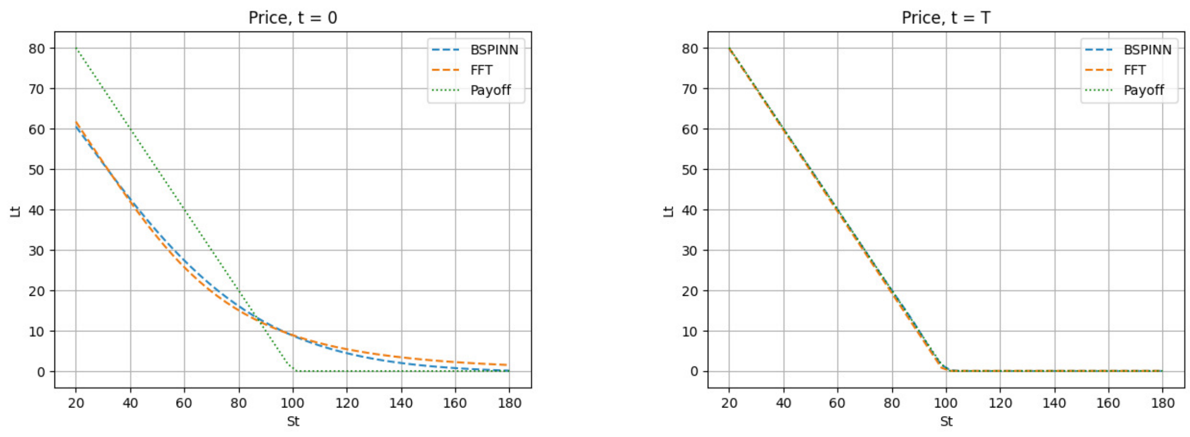

Figure 2: 5 years European put priced with PINN and FFT.

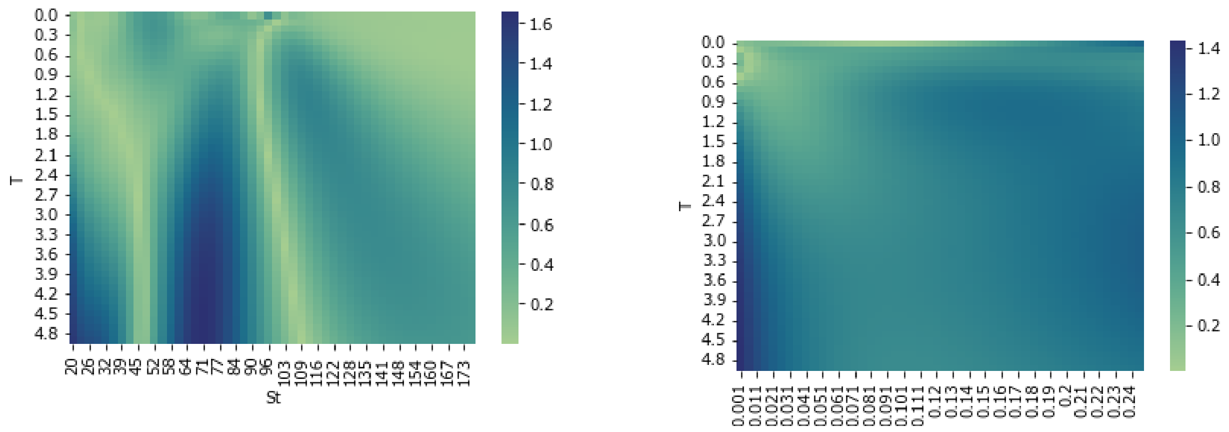

Figure 3: Heatmaps of absolute differences between PINN and FFT prices.

The right plot of Figure 2 presents option prices at expiry and reveals the terminal boundary condition is well fulfilled. The left plot of Figure 3 shows a heatmap of absolute gaps between PINN and FFT option prices for maturities between 0 and 5 years and $S_0 \in [20, 180]$. It confirms that spreads between FFT and PINN prices remain relatively small whatever the maturity and S_0. For a given maturity, the largest pricing errors are observed in areas where the convexity of the price curve is the highest. The right plot of of Figure 3 presents the heatmap of price spreads for maturities $T \in [0, 5]$ and $V_0 \in [0.001, 0.24]$. The stock value is set to $S_0 = 90$. Spreads are smalls and nearly constant. We observe a small increase at short term and for higher initial variances.

6.2 Analysis of the 4-256 network, random market parameters

In this subsection, we analyze the spreads between FFT and PINN prices for various Heston parameters. The right plot of Figure 4 shows the heatmap of absolute differences between FFT and PINN prices by maturities and stock values. To limit computational time, we consider 2500 combinations of Heston parameters and state variables simulated as follows. For each value of S_t ranging from 20 to 180 in steps of 3.2 (50 values), we randomly draw 50 Heston parameters, maturities, and variances. For each combination, we then compute the put price using both FFT and PINN methods. Some of cells of the heatmap are uncolored because the FFT algorithm fails to compute a price for the given set of Heston parameters. As mentioned, the FFT algorithm is sensitive to the number of discretization steps M and to the domain of the pdf, $[-u_{max} ; u_{max}]$, here set to $M = 2^8$ and $u_{max} = 2.2$. When the volatility is high, the FFT algorithm fails to compute a price due to an overflow encountered in the calculation of the characteristic function. Obtaining a value requires tuning M and u_{max} on a case-by-case basis without guarantee of accuracy. In comparison the PINN always yields a price regardless of the market parameters and is, from this viewpoint, much more robust than the FFT. On the other hand, we observe that for most standard market configurations, the pricing error is under control. The gap between FFT and PINN price increases with maturity but this is partly due to the numerical instability of the FFT method. The left plot of Figure 4 shows the same heatmap with respect to maturities and variances (2500 points). This confirms that the PINN pricing error is reasonable in most of configuration.

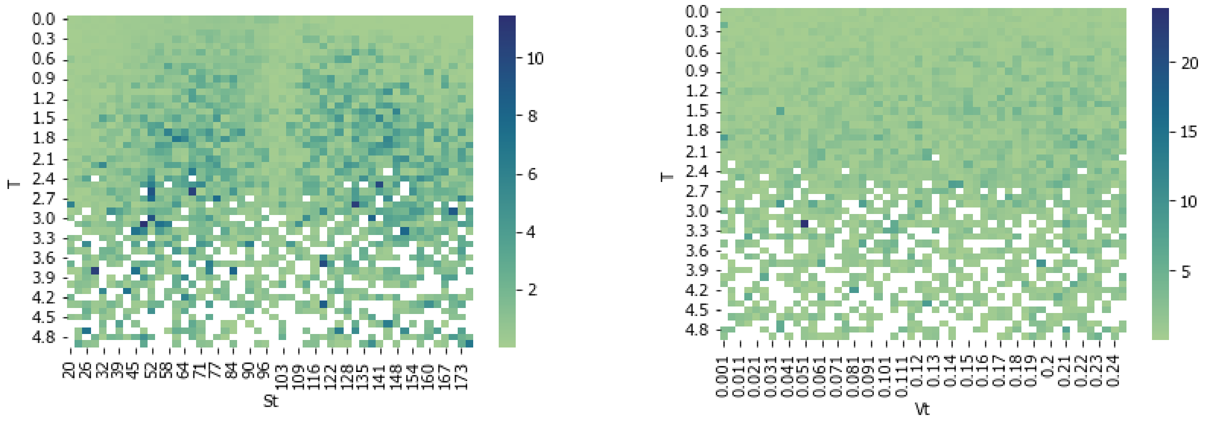

Figure 4: Heatmap of spreads between PINN and FFT prices on a random sample of parameters. Left plot : T vs S_0. Right plot: T vs V_0.

6.3 Sensitivity of the 4-256 network to training parameters

In this section, we analyze the sensitivity of performance metrics with repect to parameters of the training procedure. We focus on the network with 4 layers and 256 neurons. We perform three tests:

- Test A : Increase of the sample size $n_{\mathcal{D}}$ from 20,000 to 100,000 items and n_T, n_{low} from 5,000 to 25,000.

- Test B : Increase of the number of epochs from 3,500 to 6,500 and a smaller learning rate of 1/10,000 during the last 3,000 iterations.
- Test C : Both increases of the sample size and number of epochs (combination of tests A and B).

	Total loss	Inner loss $\mathcal{L}_D$	Lower loss $\mathcal{L}_{low}$	T loss $\mathcal{L}_T$	MSE_{alea}	MSE_{config}	$Err_{relative}$
Initial	0.049	0.030	0.001	0.017	2.346	0.696	0.044
Test A	0.058	0.034	0.001	0.023	2.418	0.667	0.042
Test B	0.037	0.023	0.001	0.014	2.381	0.702	0.043
Test C	0.040	0.024	0.0004	0.016	2.460	0.697	0.044

Table 7: Performance metrics for various training parameters

The performance metrics after training are reported in Table 7. Several conclusions can be drawn from these tests. Firstly, increasing the size of the training sample (test A: 150,000 in total vs. 30,000 considering inner and boundary data sets) does not lead to a significant reduction in the loss function. Nevertheless, a significant improvement in MSE is not observed. Secondly, increasing the number of epochs and reducing the learning rate (test B) allows for a slight reduction in the loss function, but the reductions in MSE are not truly significant. Thirdly, increasing both the sample size and the number of epochs slightly reduces the total loss, but MSE does not show improvement. Finally, we conclude that for the 4-256 network, there is no need to increase the size of the data sample or the number of epochs. With 3,500 iterations and 30,000 data items, we achieve optimal accuracy for this network.

6.4 Release of the lower boundary condition

The analysis of inner losses in Table 4 reveals that errors on the lower boundary (when $S_t = 0$) are nearly null (< 0.001). On this boundary, put option prices are set to discounted strike prices, and this constraint is well integrated by nearly all models. It raises the question of whether we can release this boundary condition. Our goal is to test if we could improve the inner and T losses by ignoring this condition and indirectly giving them more importance during training. We test this for the network with 4 layers and 256 neurons. The performance metrics are reported in Table 8 , which confirms our intuition. Releasing the lower boundary condition allows a significant reduction in the inner and T losses (-62%). Surprisingly, we do not observe a major change in MSEs, but as already mentioned, this may also be caused by the numerical inaccuracies of the FFT.

Lower boundary	Total loss	Inner loss $\mathcal{L}_D$	Lower loss $\mathcal{L}_{low}$	T loss $\mathcal{L}_T$	MSE_{alea}	MSE_{config}	$Err_{relative}$
with	0.049	0.030	0.001	0.017	2.346	0.696	0.044
without	0.019	0.011	n.a.	0.008	2.481	0.699	0.045

Table 8: Performance metrics for various training parameters

7 Conclusions

This article demonstrates that a Physics-Inspired Neural Network (PINN) is an efficient alternative to standard methods for pricing options. In the financial industry, rapid pricing is of paramount importance for the continuous recalibration of models to market quotes. Furthermore, the constant fluctuations in market parameters call for a robust tool. PINNs provide a solution to these challenges. After training, valuing an option with a PINN is indeed quasi-instantaneous and by construction, the PINN is not subject to numerical instabilities.

We focus in this work on the Heston model, a standard framework for option pricing. Utilizing Itô calculus and non-arbitrage arguments, we derive the Feynman-Kac equation governing the prices of financial contingent claims. We propose a scaled and centered version of this partial differential equation (PDE), whose solution is approximated by a neural network. We consider a specific type of feedforward neural network with skip connections to intermediate layers to capture the non-linearity of prices.

The calibration of this network does not require exact option values; instead, we minimize, via stochastic gradient descent, the errors when replacing the option value with the Physics-Inspired Neural Network (PINN) in the Feynman-Kac PDE. In comparison to existing literature, the novelty of our approach lies in the network's ability to evaluate options for a wide range of Heston parameters and maturities. Contrastingly, the Fast Fourier Transform (FFT) method needs to be rerun after every modification of model parameters or option maturity, making PINN a significantly faster pricing process.

Additionally, as seen in the numerical illustrations, the FFT method is not always numerically stable and requires fine-tuning. The PINN does not suffer from this drawback. However, our approach has practical limitations, especially regarding the structure of the neural network architecture. To the best of our knowledge, no theory on the architecture of the network has yet been formulated for solving stochastic differential equations. We have thus opted for a layer-constant network structure, although other architectural methods (top-down, bottom-up architecture, etc.) could be considered.

On the other hand, the calibration procedure (number of iterations, learning rate, choice of optimizer) relies more on empirical practices than on theoretical results. Empirical use is crucial for choosing the right calibration, and a trial-and-error phase is necessary in practice. While the performance of neural networks is usually significant, the explicability of models is low. The black-box aspect of this approach limits model comprehension and explicability. A market parameter sensitivity approach can be used to understand the main principles of the market.

The consistency and accuracy demonstrated by Physics-Inspired Neural Networks (PINNs) in valuing European options within the Heston model suggest their potential relevance for pricing more complex exotic derivatives. An extension of the proposed framework to value Bermudian options, for instance, could be explored. Handling multiple exercise dates in this context would require incorporating into the loss function intermediate errors related to early exercises. Further enhancements to the accuracy of PINNs could be achieved through various avenues, such as incorporating long-short term memory cells or employing boosting techniques.

References

[1] Al-Aradi A., Correia A., Jardim G., de Freitas Naiff D., Saporito Y. 2022. Extensions of the deep Galerkin method. Applied Mathematics and Computation.

[2] Barigou K., Delong L. 2022. Pricing equity-linked life insurance contracts with multiple risk factors by neural networks. Journal of Computational and Applied Mathematics, 404, 113922.

[3] Beck C., Weinan E., Jentzen A. 2019. Machine learning approximation algorithms for high

dimensional fully nonlinear partial differential equations and second-order backward stochastic differential equations. Journal of Nonlinear Science, 29, 1563-1519.

[4] Biagini F., Gonon L., Reitsam T., 2023. Neural network approximation for superhedging prices. Mathematical Finance 33 (1), 146-184.

[5] Buehler H., Gonon L., Teichmann J., Wood B., 2019. Deep hedging. Quantitative Finance 19 (8), 1-21.

[6] Cai Z., 2018. Approximating quantum many-body wave-functions using artificial neural networks. Physical Review B, 97, 035116.

[7] Carr P., Madan D. 2001. Option valuation using the fast Fourier tran.sform. Journal of Computational Finance 2 (4), 61-73

[8] Cuomo S., Schiano Di Cola V., Giampaolo F., Rozza G., Raissi M., Piccialli F., 2022. Scientific machine learning through Physics-Informed Neural Networks: where we are and what's next. Journal of Scientific Computing 92:88.

[9] Carleo G., Troyer M. 2017. Solving the quantum many-body problem with artificial neural networks. Science 355 (6325), 602–606.
Delong L., Dhaene J., Barigou K., 2019. Fair valuation of insurance liability cash-flow streams in continuous time: Theory. Insurance: Mathematics and Economics 88, p 196-208.

[10] Denuit M., Trufin J., Hainaut D. 2019. Effective statistical learning for actuaries III. Neural networks and extensions. Springer Nature Switzerland

[11] Doyle D.,Groendyke C. ,2019. Using neural networks to price and hedge variable annuity guarantees. Risks 7(1), 1.

[12] Gatta F., Di Cola V. S., Giampaolo F., Piccialli F. Cuomo S. 2023. Meshless methods for American option pricing through Physics-Informed Neural Networks. Engineering Analysis with Boundary Elements 151, 68-82.

[13] Germain M., Pham H., Warin X., 2021. Neural networks-based algorithms for stochastic control and PDE's in finance. Machine Learning And Data Sciences For Financial Markets: A Guide To Contemporary Practices. Edited by Agostino Capponi and Charles-Albert Lehalle, Cambridge University Press.

[14] Glau K., Wunderlich L., 2022. The deep parametric PDE method and applications to option pricing. Applied Mathematics and Computation. 432, 127355.

[15] Hainaut D. 2022. Continuous time processes for finance. Switching, Self-exciting, fractional and other recent dynamics. Springer & Bocconi Series in mathematics, statistics, finance and Economics.

[16] Hejazi S.A., Jackson K.R. 2016. A neural network approach to efficient valuation of large portfolios of variable annuities, Insurance: Mathematics and Economics 70, p 169–181.

[17] Hornik K. 1991. Approximation capabilities of multilayer feedforward networks. Neural Networks 4 (2), 251-257.

[18] Horvath B., Teichmann J., Zuric Z., 2021. Deep hedging under rough volatility. Risk 9 (7), 138.

[19] Jiang Q., Sirignano J., Cohen S., 2023. Global convergence of Deep Galerkin and PINNs method for solving PDE. arXiv preprint.

[20] Lee H., Kang I.S., 1990. Neural algorithm for solving differential equations. J. Comput. Phys. 91(1), 110-131.

[21] Raissi M., Perdikaris P., Karniadakis G.E. 2019. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. J Comput. Phys. 378, 686–707.

[22] Sirignano J., Spiliopoulos K., 2018. DGM: a deep learning algorithm for solving partial differential equations, J. Comput. Phys. 375, 1339–1364.

[23] van Merriënboer B., Breuleux O., Bergeron A. 2018. Automatic differentiation in ML: Where we are and where we should be going. 32th NeurIPS proceedings. Advances in Neural Information Processing Systems, 31, 1-11.

[24] Weinan E., Han J., Jentzen A., 2017. Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. Commun. Math. Stat. 5(4), 349– 380.

Detralytics

People drive actuarial innovation

www.detralytics.com - info@detralytics.eu