

Faculté des sciences

Options pricing with Physics-Informed Neural Networks

Author: **Marie THIBEAU**
Supervisor: **Donatien HAINAUT**
Reader: **Jérôme BARBARIN**
Academic year 2023–2024
Master [120] en sciences actuarielles

Abstract

This thesis explores the use of Physics-Informed Neural Networks (PINNs) for option pricing in complex financial contexts. We developed models to evaluate European call and Bermudian put options, incorporating stochastic interest rates modelled by Hull & White. For European options, the results were validated with a closed-form solution, while validation for Bermudian puts has been more challenging. Numerical analyses demonstrate that PINNs offer a robust parametric tool that can be able of well capturing the price dynamics, outperforming traditional methods in scenarios with multiple risk factors and requiring minimal recalibration across different market conditions. However, challenges remain in optimizing network architecture selection and calibration procedure, which currently rely on trial and error. Future work could enhance these models by exploring alternative neural network architectures, expanding to more complex frameworks like Svensson or Heston models, and applying these methods to life insurance. This research contributes to advancing PINNs as a versatile tool in financial modelling.

Key words: European and Bermudan Options pricing, Physics-Informed Neural Networks (PINNs), Stochastic interest rate, Feynman-Kac equation.

Acknowledgments

First and foremost, I would like to thank my supervisor, Prof. Donatien Hainaut, for his availability and guidance. His confidence in my work was appreciated, and I am sincerely grateful for his insight and encouragement.

I am profoundly grateful to my partner, Tiago, whose constant presence and support made this journey possible. His management of our daily life together and his constant encouragement allowed me to focus on my research, knowing that I had someone by my side every step of the way.

I owe a special thanks to my Mum, Valérie, for her careful proofreading and her generosity in sharing knowledge. Her dedication has always inspired me to surpass myself.

I don't forget my Dad, François, for checking my English, thank you very much for your time.

Finally, my immense gratitude to my friends for their support in this challenge and in everyday life.

Contents

Abstract	i
Acknowledgements	ii
1 Introduction	1
1.1 State-of-the-art	1
1.2 Contributions and overview	3
2 General problem statement	4
2.1 Financial market with stochastic interest rate	4
2.1.1 Expression of $\gamma(t)$	5
2.2 Scaled Feynman-Kac equation	6
2.3 Neural-networks generalities	8
2.3.1 Definition and applications	8
2.3.2 Vanishing gradient effect	9
2.3.3 Proposed architectures	10
2.4 Calibration	12
2.4.1 Domain and range of state variables and parameters	13
2.4.2 Training points	14
2.4.3 Learning phases and metrics	15
2.4.4 Implementation	17
3 European call option pricing with a single interest rate curve	18
3.1 Boundary constraints	18
3.2 Loss function	19
3.3 Validation procedure	21
3.4 Numerical analysis	23
3.4.1 Loss function weights selection	23
3.4.2 Network architecture and hyperparameters selection	24
3.4.3 Performances of the 3-256-DGM network	27
3.4.4 Sensitivity of the 3-256-DGM network to loss function	31
3.4.5 Sensitivity of the 3-256-DGM network to sampling method of training points	32
3.4.6 Sensitivity of the 3-256-DGM network to training parameters	33
4 Bermudan put option pricing with a single interest rate curve	35
4.1 Boundary constraints	36
4.2 Loss function	37
4.3 Validation procedure	39
4.4 Numerical analysis	43
4.4.1 Loss function weights selection	43
4.4.2 Network architecture and hyperparameters selection	44

4.4.3	Performances of the 3-256-ResNN network	45
4.4.4	Sensitivity of the 3-256-ResNN network to training parameters	47
5	Extending the model: Option pricing with multiple interest rate curves	48
5.1	Model adaptations	48
5.2	European call option case	49
5.3	Bermudan put option case	50
5.4	Conclusion	50
6	Perspectives for life insurance applications	51
7	Conclusion	52
A	Interest rate modelling: Hull & White model	54
B	Feynman-Kac equation	58
C	Closed-form expression for the European call option	60
	Bibliography	64

Chapter 1

Introduction

The rapid advancements in machine learning and artificial intelligence have opened new avenues for tackling complex problems in finance, particularly in option pricing. Among these innovations, Physics-Informed Neural Networks (PINNs) stand out as a powerful tool that integrates the underlying physical and financial principles into the neural network framework. This capability allows PINNs to efficiently solve partial differential equations (PDEs), which are at the core of option pricing models. Unlike traditional approaches, PINNs reduce the dependency on large datasets and deliver accurate predictions even with limited or noisy data. It also offers a flexible and adaptive approach.

In this master thesis, we explore the application of PINNs in the context of option pricing, with a particular focus on improving the accuracy and flexibility of the model. Our work extends existing frameworks by incorporating advanced neural network architectures to tackle both European and Bermudan options under stochastic interest rate and various economic conditions. This work also mentions some future applications for the use of PINNs in financial contexts such as life insurance, where traditional methods are often insufficient.

The following sections will explore the current state-of-the-art in option pricing using neural network and the contributions of this master thesis to the field.

1.1 State-of-the-art

Option pricing has been widely studied in the literature, and allow us to provide a thorough overview of the latest and most pertinent work, including that using neural networks. Option pricing has traditionally relied on several widely used methods, each with its strengths and limitations. Monte Carlo simulations estimate option prices by averaging payoffs from numerous simulated asset price paths, offering flexibility but at a high computational cost. Binomial and trinomial trees model provide price evolution in discrete time, balancing simplicity with accuracy, which depends on the number of time steps. PDE-based methods, derived from the Black-Scholes equation, provide precise pricing for vanilla options through continuous modelling, though they become complex when dealing with stochastic factors or exotic derivatives. These established techniques, while effective, often face challenges with more complex scenarios, leading to the exploration of advanced methods like Physics-Informed Neural Networks in option pricing.

PINNs are an innovative approach that integrates physical laws directly into the neural network training process, making them particularly useful for solving complex problems where traditional methods struggle. PINNs can efficiently approximate the solution of non-linear partial differential equations, explaining the dynamics of a model. The neural network is

trained by minimizing the approximation error at sampled points within the PDE domain and along its boundaries. Initially proposed by Lee and Kang (1990) and later used for solving non-linear PDEs, PINNs have been applied across various fields. Raissi et al. (2019) used them for data-driven solutions and discovery of PDEs, while Carleo and Troyer (2017) and Cai and Liu (2018) applied PINNs to accurately approximate quantum many-body wave functions. Some proposed other version of PINN such as Jagtap and Karniadakis (2020) who proposed Extended PINNs (XPINNs) for general space-time domain decomposition for nonlinear PDEs and Hu et al. (2023) who proposed Augmented PINNs (APINNs) which adopt soft and trainable domain decomposition and flexible parameter sharing. Cuomo et al. (2022) proposed a comprehensive review of recent developments and applications of PINNs, for further readings.

Research on using PINNs for valuing financial derivatives or in the insurance field is relatively recent. Hejazi and Jackson (2016) developed a neural network model for pricing and estimating the Greeks of large portfolios of variable annuities. Doyle and Groendyke (2018) extended this approach by building a neural network to price and hedge equity-linked contracts, using a dataset generated through Monte-Carlo simulations. Sirignano and Spiliopoulos (2018) introduced the Deep Galerkin Method (DGM), a deep learning algorithm inspired by LSTM neural cells, to solve high-dimensional free boundary and Hamilton-Jacobi-Bellman partial differential equations. Building on this, Al-Aradi et al. (2018) tested the DGM model on various financial problems, including the pricing of European and American options, while Al-Aradi et al. (2022) later extended it for solving Fokker-Planck and Hamilton-Jacobi-Bellman equations. Gatta et al. (2023) proposed a PINN specifically for evaluating American multi-asset options, introducing a novel technique for training free boundaries. Glau and Wunderlich (2022) formalized the deep parametric PDE method, applying it to high-dimensional parametric PDEs relevant in option pricing. Dhiman and Hu (2023) utilized a physics-informed deep learning approach for pricing American and European options under the Black-Scholes model. In the context of life insurance, Hainaut (2023) proposed a PINN-based method for valuing Guaranteed Minimum Accumulation Benefits (GMAB) under multiple financial and biometric risks. Most recently, Hainaut and Casas (2024) extended the application of PINNs to option pricing within the Heston model, demonstrating their versatility in complex financial environments.

In addition to Physics-Informed Neural Networks, other neural network-based approaches have been developed for the valuation of financial derivatives. A close alternative to PINNs is the method proposed by Han et al. (2017), which leverages the connection between high-dimensional parabolic PDEs and backward stochastic differential equations (BSDEs). This approach uses two separate neural networks, one for the solution and another for its gradient, and has been applied to pricing options in a multivariate Black-Scholes model. Similarly, Beck et al. (2019) introduced a method for solving high-dimensional PDEs by exploiting the relationship between fully nonlinear second-order PDEs and second-order BSDEs, demonstrating its accuracy with the Black-Scholes and Hamilton-Jacobi-Bellman equations. Building on this principle, Barigou and Delong (2022) used neural networks to solve BSDEs for pricing equity-linked life insurance contracts under multiple stochastic risk factors.

Another approach involves using supervised neural networks, which require a training set of exact prices to calibrate the network by minimizing the difference between predicted and actual prices. In the literature, these training sets are derived from either simulations or observed market prices. For instance, Buehler et al. (2019) present a framework where

neural networks are used to price and hedge derivatives based on simulated sample paths of risk factors, approximating optimal investment strategies by averaging discounted simulated payoffs. Similarly, Horvath et al. (2021) evaluate deep hedging performance under rough volatility models, while Biagini et al. (2023) use simulations to develop a neural network approximation for superhedging prices and replicating strategies. Kohler et al. (2010) apply the least squares neural network regression for pricing American options in discrete time by estimating continuation values. In contrast, other researchers, such as Anders et al. (1998), Yao et al. (2000), Gaspar et al. (2020), and Du Plooy and Venter (2021), train their neural networks using market-observed prices for the pricing of call options written on the German stock index DAX, of Nikkei 225 index futures, of American put options and of European call options using a constructed implied volatility surface, respectively.

For these methods based on neural networks, models are trained for a specific set of parameters and must be recalibrated whenever contract specifications change. This contrasts with the flexibility of PINNs, which can be trained over a range of parameter values, as will be further discussed in the following sections of this master thesis.

1.2 Contributions and overview

The contributions of this master thesis are multiple. In Chapter 2, we develop a pricing framework based on the notations used by Hainaut (2023), within the context of stochastic interest rate modelling using the Hull & White (HW) model. After giving a general overview of neural networks, we explore different architectures, including a classic feedforward network, a feed-forward network with intermediate layers also receiving the input as proposed by Hainaut (2023), and a network incorporating DGM layers as introduced by Sirignano and Spiliopoulos (2018).

In Chapter 3, we apply this framework to price European call options, deriving a closed-form expression that enables us to assess the accuracy of the PINN's predictions across a wide range of market parameters. Similarly, in Chapter 4, we use the framework to price Bermudan put options, proposing a validation procedure based on a hybrid bi-trinomial tree. However, this procedure requires further development to be fully operational. In both chapters, we conduct numerical analyses by training multiple networks under various configurations. In Chapter 5, we present an extended model that facilitates the pricing of derivatives within the context of multiple interest rate curves. Our models offer greater flexibility compared to traditional pricing methods, BDSE, or deep hedging, which each require retraining for new parameter sets.

In Chapter 6, we mention potential applications of our tool in life insurance contexts. The master thesis ends with discussions, conclusions, and perspectives for future work in Chapter 7.

Chapter 2

General problem statement

The general problem developed in this chapter is applied to both European case and Bermudian case of vanilla options (call or put) considering a financial market with a stochastic interest, as described in Section 2.1. We then derive the stochastic differential equation (SDE) satisfied by all derivatives present on this financial market in Section 2.2. This SDE will be used to train the model, a neural network, to estimate option prices. Then, in Section 2.3, we give a general description of neural networks and the architectures proposed in this thesis. Finally, we discuss the elements required to calibrate the model in Section 2.4. These include the determination of learning points and a description of the process for comparing results.

2.1 Financial market with stochastic interest rate

We consider a financial market composed of two assets. The account earns a stochastic risk-free rate $(r_t)_{t \geq 0}$. The stock price, noted $(S_t)_{t \geq 0}$, is ruled by a geometric Brownian motion. These price and rate processes are defined on a probability space $(\Omega, \mathcal{F}, \mathbb{P})$ associated to two independent Brownian motions under $\mathbb{P}$, $\tilde{\mathbf{W}}_t = (\tilde{W}_t^{(1)}, \tilde{W}_t^{(2)})_{t \geq 0}^\top$. By considering the model given by Hull and White (1990), called the Hull & White (HW) model, for the risk-free stochastic interest rate, the state variables (r_t, S_t) are driven by the following SDE's:

$$\begin{pmatrix} dr_t \\ dS_t \end{pmatrix} = \begin{pmatrix} \kappa \left(\gamma(t) - \nu_r \frac{\sigma_r}{\kappa} - r_t \right) \\ (r_t + \nu_S \sigma_S) S_t \end{pmatrix} dt + \begin{pmatrix} \sigma_r \Sigma_r^\top \\ \sigma_S S_t \Sigma_S^\top \end{pmatrix} d\tilde{\mathbf{W}}_t, \quad (2.1)$$

where κ, σ_r and σ_S are in $\mathbb{R}^+$ and $\gamma(t)$ is a positive function of time:

- $\gamma(t)$ is fitted to term structure of interest rates and corresponds to a mean reversion level towards which r_t converges;
- κ describes the speed of reversion towards $\gamma(t)$;
- σ_r is the standard deviation of r_t ;
- σ_S is the standard deviation of S_t .

The parameters ν_r and ν_S are market prices of risk, and they tune the risk premiums of processes and allow finding risk-neutral expressions of these processes. Σ_r and Σ_S are vectors such that the following matrix:

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

is the (lower) Cholesky decomposition of the correlation matrix: $\begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix} = \Sigma \Sigma^\top$ where $\rho \in (-1, 1)$ is the correlation between the stochastic risk-free rate and the stock price. The processes (2.1) can be written in developed form as follows:

$$\begin{aligned} dr_t &= \kappa \left(\gamma(t) - \nu_r \frac{\sigma_r}{\kappa} - r_t \right) dt + \sigma_r d\tilde{W}_t^{(1)}, \\ dS_t &= (r_t + \nu_S \sigma_S) S_t dt + \sigma_S \rho S_t d\tilde{W}_t^{(1)} + \sigma_S \sqrt{1 - \rho^2} S_t d\tilde{W}_t^{(2)}. \end{aligned}$$

We assume that costs of risk, noted $(\lambda_j)_{j=1,2}$, are constant. In other words, by the theorem given by Girsanov (1960), the Brownian motions with drift under $\mathbb{P}$,

$$dW_t^{(j)} = d\tilde{W}_t^{(j)} + \lambda_j dt, \quad (j = 1, 2),$$

are Brownian motions under $\mathbb{Q}$, the risk neutral measure. To keep identical dynamics under $\mathbb{P}$ and $\mathbb{Q}$, the risk premium parameters, ν_r and ν_S , are set to

$$\begin{aligned} \nu_r &= -\lambda_1, \\ \nu_S &= \rho \lambda_1 + \sqrt{1 - \rho^2} \lambda_2. \end{aligned}$$

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

$$\begin{pmatrix} dr_t \\ dS_t \end{pmatrix} = \begin{pmatrix} \kappa (\gamma(t) - r_t) \\ r_t S_t \end{pmatrix} dt + \begin{pmatrix} \sigma_r \Sigma_r^\top \\ \sigma_S S_t \Sigma_S^\top \end{pmatrix} d\mathbf{W}_t, \quad (2.2)$$

where $\mathbf{W}_t = (W_t^{(1)}, W_t^{(2)})_{t \geq 0}^\top$.

The pure Brownian model set up allows to find closed form prices in specific cases, as it will be seen for the European case in Section 3.3.

2.1.1 Expression of $\gamma(t)$

For numerical illustration, we choose to model the initial yield curve with the Nelson-Siegel (NS) model. This model expressed the initial instantaneous forward rates with the following function:

$$f(0, t) := -\frac{\partial \ln P(0, t)}{\partial t} = b_0 + (b_{10} + b_{11}t) \exp(-c_1 t),$$

where $P(0, t)$ is the price of a zero-coupon bond of maturity t . This price can be expressed in terms of instantaneous forward rates, $f(0, t)$ or in terms of the continuous zero-coupon rates, $r_0(t)$:

$$P(0, t) = \exp \left(- \int_0^t f(0, s) ds \right), \quad \text{or,} \quad P(0, t) = \exp(-r_0(t)t).$$

Then, by identification, the initial zero-coupon yields depend on the Nelson-Siegel model:

$$r_0(t) = \frac{1}{t} \int_0^t f(0, s) ds = b_0 + \frac{1}{t} \frac{b_{10}}{c_1} (1 - e^{-c_1 t}) + \frac{1}{t} \frac{b_{11}}{c_1^2} (1 - (c_1 t + 1) e^{-c_1 t}). \quad (2.3)$$

We fit the NS parameters to the yield curve of Belgian state bonds observed on the 28th of June 2024¹ and obtain estimates given in Table [2.1].

¹National Bank of Belgium Online statistics, *Yield of Belgian loans on the secondary market*: <https://stat.nbb.be/Index.aspx?DataSetCode=IROLOBE2>, consulted on 28-06-2024.

Parameter	Value
b_0	0.039749
b_{10}	0.001472
b_{11}	-0.016936
c_1	0.422290

Table 2.1: Nelson-Siegel parameters corresponding to the Belgian state bonds on the 28th of June 2024.

The dynamic of the risk-free interest rate is given by the HW model, see Equation (2.2), where the function $\gamma(t)$ is chosen to ensure the fit of observed yields. These observed yields are given by the NS model (2.3) with calibrated parameters of Table [2.1]. The function $\gamma(t)$ is obtained in Hull and White (1990) and Hull and White (1994) and is given, see Proposition A.2 in Appendix A, by:

$$\gamma(t) = \frac{1}{\kappa} \frac{\partial f(0, t)}{\partial t} + f(0, t) + \frac{\sigma_r^2}{2\kappa^2} (1 - e^{-2\kappa t}).$$

Which, with the Nelson-Siegel model, gives:

$$\gamma(t) = \frac{1}{\kappa} (b_{11} - c_1(b_{10} + b_{11}t)) e^{-c_1 t} + b_0 + (b_{10} + b_{11}t)e^{-c_1 t} + \frac{\sigma_r^2}{\kappa^2} (1 - e^{-c_1 t}). \quad (2.4)$$

We will see in the Chapter 5 that we can either use a unique interest rate curve based on the observed data at a given time or consider several market conditions and therefore a set of interest rate curves. In the first case, we use the calibrated parameters of Table [2.1]. In the second case, we consider several values within a range for the NS parameters. However, we will initially focus only on the first case described above.

2.2 Scaled Feynman-Kac equation

In this financial market, we consider a derivative (either a put or a call) that expires at time T and has a payoff, $H(S_T)$, which depends on the stock price and interest rate at maturity. This derivative can be either European or Bermudian. The Bermudian derivative can also be exercised on a series of intermediate dates $I = \{t_1, t_2, \dots, t_n = T\}$, where the payoff is $H(S_{t_i})$. These intermediate dates are also called exercise dates. These two scenarios will be discussed in detail later in Chapter 3 and Chapter 4, respectively.

A physics-inspired neural network (PINN) incorporates principles from physics, such as partial differential equations (PDEs) that govern the behaviour of state variables. In the context of finance, we apply the Feynman-Kac (FK) equation, a PDE that is satisfied by all assets traded in an arbitrage-free market.

Let the vector of state variables be

$$\mathbf{y}_t = (r_t, S_t)^\top.$$

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

$$d\mathbf{y}_t = \mu_{\mathbf{y}}(t, \mathbf{y}_t)dt + \Sigma_{\mathbf{y}}(t, \mathbf{y}_t)d\mathbf{W}_t,$$

where $\mu_{\mathbf{y}}(\cdot)$ is a vector of dimension 2 and $\Sigma_{\mathbf{y}}(\cdot)$ is a 2×2 matrix. By identification with (2.2):

$$\mu_{\mathbf{y}}(t, \mathbf{y}_t) = \begin{pmatrix} \kappa(\gamma(t) - r_t) \\ r_t S_t \end{pmatrix}, \quad \Sigma_{\mathbf{y}}(t, \mathbf{y}_t) = \begin{pmatrix} \sigma_r \Sigma_r^\top \\ \sigma_S S_t \Sigma_S^\top \end{pmatrix}.$$

Denote the derivative price (or fair value) as follows:

$$F_t = V(t, \mathbf{y}_t | \kappa, \sigma_r, \sigma_S, \rho, T).$$

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

$$\mathbb{E}(dF_{t+} | \mathcal{F}_t) = F_t r_t dt. \quad (2.5)$$

The gradient and the Hessian of $V(\cdot)$ with respect to $\mathbf{y}$ are respectively denoted as follows:

$$\nabla_{\mathbf{y}} V = \begin{pmatrix} \partial_r V \\ \partial_S V \end{pmatrix}, \quad \mathcal{H}_{\mathbf{y}}(V) = \begin{pmatrix} \partial_{rr} V & \partial_{rS} V \\ \partial_{Sr} V & \partial_{SS} V \end{pmatrix}.$$

Applying Itô's lemma to $V(\cdot)$ allows us to write the condition (2.5) as a partial differential valuation equation, see developments in Appendix B:

$$0 = \partial_t V - r_t V + \mu_{\mathbf{y}}(t, \mathbf{y}_t)^\top \nabla_{\mathbf{y}} V + \frac{1}{2} \text{tr} \left(\Sigma_{\mathbf{y}}(t, \mathbf{y}_t) \Sigma_{\mathbf{y}}(t, \mathbf{y}_t)^\top \mathcal{H}_{\mathbf{y}}(V) \right). \quad (2.6)$$

This last expression is called the "Feynman-Kac (FK) equation".

To avoid convergence issues related to the vanishing gradient effect in neural network learning, we need to centre and scale the inputs of the network, as explained in Section 2.3.2. This pretreatment slightly modifies the FK equation. Let us define the vectors $\mathbf{a}, \mathbf{b} \in \mathbb{R}^2$ as follows,

$$\begin{aligned} \mathbf{a} &= (a_r, a_S)^\top, \\ \mathbf{b} &= (b_r, b_S)^\top. \end{aligned}$$

$\mathbf{a}$ and $\mathbf{b}$ are chosen to normalize a random sample of state variables. The vector of centred and scaled state variables is denoted by

$$\tilde{\mathbf{y}}_t = \mathbf{a} + \mathbf{b} \odot \mathbf{y}_t,$$

where $\odot$ is the elementwise product. Similarly, we centre and scale the time with coefficients a_h and b_h that will depend on the horizon of valuation. The centred scaled time and duration are defined as follows

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

The normalized state variables are ruled by the following SDE:

$$d\tilde{\mathbf{y}}_t = \mu_{\tilde{\mathbf{y}}}(t, \tilde{\mathbf{y}}_t) dt + \Sigma_{\tilde{\mathbf{y}}}(t, \tilde{\mathbf{y}}_t) d\mathbf{W}_t,$$

where $\mu_{\tilde{\mathbf{y}}}(\cdot)$ is a 2-vector and $\Sigma_{\tilde{\mathbf{y}}}(\cdot)$ is a 2×2 matrix. By identification with the dynamics of $\tilde{r}_t$ and $\tilde{S}_t$:

$$\mu_{\tilde{\mathbf{y}}}(t, \tilde{\mathbf{y}}_t) = \begin{pmatrix} \kappa(b_r \gamma(t) + a_r - \tilde{r}_t) \\ (\tilde{S}_t - a_S) \frac{\tilde{r}_t - a_r}{b_r} \end{pmatrix}, \quad \Sigma_{\tilde{\mathbf{y}}}(t, \tilde{\mathbf{y}}_t) = \begin{pmatrix} b_r \sigma_r \Sigma_r^\top \\ (\tilde{S}_t - a_S) \sigma_S \Sigma_S^\top \end{pmatrix}.$$

The value of the contract can be seen as a function of scaled time and scaled state variables,

$$F_t = V(\tilde{t}, \tilde{\mathbf{y}}_t | \kappa, \sigma_r, \sigma_S, \rho, \tilde{T}).$$

The valuation Equation (2.6) is rewritten in scaled form:

$$0 = b_h \partial_{\tilde{t}} V - \frac{\tilde{r}_t - a_r}{b_r} V + \mu_{\tilde{\mathbf{y}}}(t, \tilde{\mathbf{y}}_t)^\top \nabla_{\tilde{\mathbf{y}}} V + \frac{1}{2} \text{tr} \left(\Sigma_{\tilde{\mathbf{y}}}(t, \tilde{\mathbf{y}}_t) \Sigma_{\tilde{\mathbf{y}}}(t, \tilde{\mathbf{y}}_t)^\top \mathcal{H}_{\tilde{\mathbf{y}}}(V) \right), \quad (2.7)$$

where $\nabla_{\tilde{\mathbf{y}}} V$ and $\mathcal{H}_{\tilde{\mathbf{y}}}(V)$ are respectively the gradient and the Hessian of V with respect to standardized state variables, $\tilde{\mathbf{y}}$. This expression is called the "Scaled Feynman-Kac equation". This is the PDE that will be solved using a neural network.

2.3 Neural-networks generalities

Neural networks are computational models inspired by the human brain's structure and function. They consist of interconnected layers of nodes (neurons), each performing simple computations. Neural networks can have several different architecture and learn patterns and relationships within data through a process called training or calibration. During training, they adjust the connections (weights) between neurons based on a given set of input-output examples and/or based on a loss function to minimize.

2.3.1 Definition and applications

The basic structure of a neural network includes an input layer, one or more hidden layers, and an output layer. The Figure [2.1] shows an example of the architecture of a neural network.

The input layer receives the raw input data, with each neuron corresponding to a feature in the dataset. The hidden layers perform computations and extract features from the input data. Each neuron in the hidden layers applies an activation function to its inputs, transforming the data into a form that can be more easily processed in subsequent layers. Common activation functions include the sigmoid function, the ReLU (Rectified Linear Unit) and the hyperbolic tangent (tanh).

The output layer produces the final output of the network. In a classification task, the output layer might use a softmax activation function to produce probabilities for each class, whereas in regression tasks, the output layer typically has no activation function, providing a continuous output. Mathematically speaking, we have the following definition for a classic feedforward network, a network where all neurons of a layer receives information from all neurons of the previous layer.

Definition 2.1. A FEEDFORWARD NEURAL NETWORK, with $L+1 \in \mathbb{N}$ layers and $n_0, n_1, \dots, n_L \in \mathbb{N}$ neurons by layer, is a function $N : \mathbb{R}^{n_0} \rightarrow \mathbb{R}^{n_L}$ defined by

$$N(\mathbf{z}) = \psi_L \circ \psi_{L-1} \circ \dots \circ \psi_2 \circ \psi_1(\mathbf{z}),$$

where $\mathbf{z}$ is the input vector of the network, n_0 and n_L are respectively the size of input vector and of output vector, and the functions ψ_l are defined as follows:

$$\psi_l(\mathbf{x}) = \phi_l(\mathbf{W}_l \mathbf{x} + \mathbf{b}_l), \quad l = 1, 2, \dots, L,$$

where $\phi_l(\cdot) : \mathbb{R} \rightarrow \mathbb{R}$ is the activation function of layer l , $\mathbf{W}_l \in \mathbb{R}^{n_l \times \mathbb{R}^{n_{l-1}}}$ is the weight matrix of layer l and $\mathbf{b}_l \in \mathbb{R}^{n_l}$ is the bias vector of layer l . The entire set of neural weights ($\mathbf{W}_l$ and $\mathbf{b}_l$) is denoted by the vector Θ .

The training process involves adjusting the weights of the connections between neurons to minimize a loss function, which measures the difference between the predicted and actual values. Optimization algorithms such as stochastic gradient descent (SGD) and its variants, including Adam and RMSprop, are commonly used for this purpose.

Input Layer Hidden Layer Output Layer

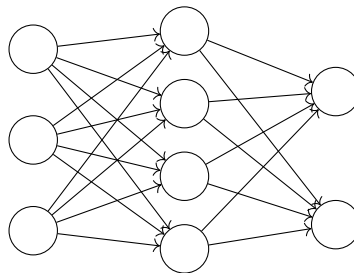


Figure 2.1: Feedforward neural network with three input neurons, four hidden neurons, and two output neurons.

Neural networks have been successfully applied across various fields, such as image and speech recognition, where they identify objects in images and understand spoken language. In natural language processing, they are used for language translation, sentiment analysis, and text generation. In the financial sector, neural networks aid in predicting stock prices, managing risks, and pricing derivatives. Additionally, in the medical field, they assist in detecting diseases from medical images and predicting patient outcomes.

In this master thesis, we use Physics-Informed Neural Networks (PINNs). They are a particular case of neural network where the loss function to minimize is inspired by physical law that can be described by partial differential equations (PDE). In the field of finance where derivatives depend upon stochastic process, we use a stochastic differential equation (SDE) such as the Feynman-Kac equation.

2.3.2 Vanishing gradient effect

The vanishing gradient effect is a phenomenon that occurs during the training of deep neural networks, particularly those with many layers. It manifests when the gradients of the loss function with respect to the network parameters become very small as they propagate back through the network during the backpropagation process. This issue is more pronounced in networks using activation functions like the sigmoid or hyperbolic tangent (tanh), where the gradients can shrink exponentially with each layer.

When the gradients are close to zero, the weights in the earlier layers of the network do not get updated effectively, which means the network learns very slowly and may even stop learning altogether. This impedes the ability of the network to capture and learn the complex patterns in the data, limiting its performance and making it difficult to train deep architectures successfully.

To address the vanishing gradient problem, several techniques have been developed such as proper weight initialization, data normalization, skip connections, etc. Normalizing the inputs of the neural network stabilizes the learning process and allows for higher learning rates. Introducing skip connections between layers allows gradients to flow directly through the network, bypassing one or more layers. These techniques help in maintaining gradient magnitude and alleviates the vanishing gradient problem.

2.3.3 Proposed architectures

It is important to test several architectures in order to find the best one. However, there is no standardized method for identifying the optimal structure of a neural network, so we have to rely on trial and error. As showed by Hornik (1991), any continuous function can be approximated by a single-layer neural network, provided it has a sufficient number of hidden neurons. As this number of neurons has to be large enough to ensure good accuracy, there are alternative methods that are sometimes simpler. In this master thesis, we consider these alternative methods and explore several neural network architectures to solve the Feynman-Kac equation. These architectures include:

Feedforward neural network (FNN): it consists of an input layer, several hidden layers, and an output layer. Each layer is fully connected to the next, and information flows unidirectionally from the input to the output, see the mathematical Definition 2.1. This simple structure is the foundation for more complex architectures. However, these networks often have difficulty replicating non-linear functions with a limited number of layers. The Figure [2.1] gives an example of a feedforward neural network.

Neural network with Deep Galerkin Method (DGM) layers: tackles the issue of FNN by introducing a variant of long short-term memory (LSTM) neural cells known as the Deep Galerkin Method (DGM) network, as introduced by Sirignano and Spiliopoulos (2018). Although this structure effectively replicates prices, its calibration is time-consuming due to the complexity of the neural cells. The mathematical definition of such a network is described in Definition 2.2. Illustrations of the networks and DGM layers are given in Figures [2.2] and [2.3] respectively.

Definition 2.2. A NEURAL NETWORK WITH DEEP GALERKIN METHOD LAYERS, with $L \in \mathbb{N}$ DGM layers and $n_0, n_{L+1} \in \mathbb{N}$ neurons in the input and output layer respectively, is a function $N : \mathbb{R}^{n_0} \rightarrow \mathbb{R}^{n_{L+1}}$ defined by

$$\begin{aligned}
 S^1 &= \phi_1 (\mathbf{W}_1 \mathbf{z} + \mathbf{b}_1), \\
 Z^l &= \phi_{z,l} (\mathbf{U}_{z,l} \mathbf{z} + \mathbf{W}_{z,l} S^l + \mathbf{b}_{z,l}), & l = 1, \dots, L, \\
 G^l &= \phi_{g,l} (\mathbf{U}_{g,l} \mathbf{z} + \mathbf{W}_{g,l} S^l + \mathbf{b}_{g,l}), & l = 1, \dots, L, \\
 R^l &= \phi_{r,l} (\mathbf{U}_{r,l} \mathbf{z} + \mathbf{W}_{r,l} S^l + \mathbf{b}_{r,l}), & l = 1, \dots, L, \\
 H^l &= \phi_{h,l} (\mathbf{U}_{h,l} \mathbf{z} + \mathbf{W}_{h,l} (S^l \odot R^l) + \mathbf{b}_{h,l}), & l = 1, \dots, L, \\
 S^{l+1} &= (1 - G^l) \odot H^l + Z^l \odot S^l, & l = 1, \dots, L, \\
 N(\mathbf{z}) &= \mathbf{W}_{L+1} S^{L+1} + \mathbf{b}_{L+1},
 \end{aligned}$$

where $\odot$ denotes Hadamard (element-wise) multiplication, $\phi(\cdot) : \mathbb{R} \rightarrow \mathbb{R}$ are the activation functions and $\mathbf{W}$, $\mathbf{U}$, and $\mathbf{b}$ are the weights of the neural network.

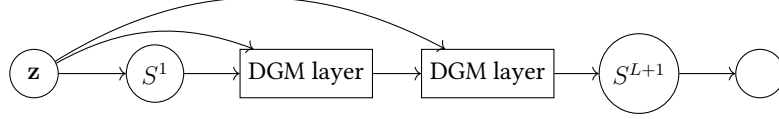


Figure 2.2: Neural network with Deep Galerkin Method layers.

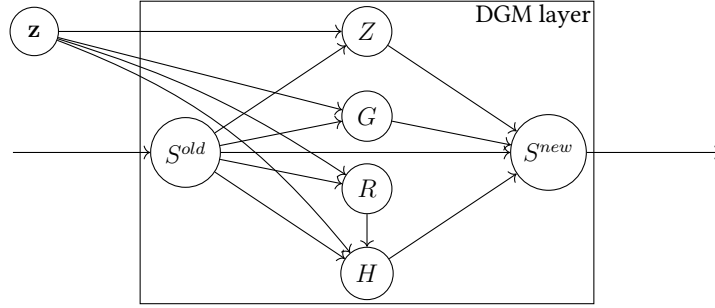


Figure 2.3: Deep Galerkin Method layer.

Feedforward neural network with skip connections: it is a FNN that includes direct connections between hidden layers and the input layer. These skip connections help mitigate the vanishing gradient problem and enable the training of deeper networks. By allowing gradients to flow more easily through the network, skip connections enhance the network's ability to learn complex functions. Mathematically speaking, this type of neural network corresponds to the Definition 2.1 where we add n_0 weights for layers $l = 2, \dots, L - 1$ corresponding to the skip connections between the input and the hidden layers, leading to Definition 2.3. The entire set of neural weights is denoted by the vector Θ .

Definition 2.3. A FEEDFORWARD NEURAL NETWORK WITH SKIP CONNECTIONS, with $L + 1 \in \mathbb{N}$ layers and $n_0, n_1, \dots, n_L \in \mathbb{N}$ neurons by layer, is a function $N : \mathbb{R}^{n_0} \rightarrow \mathbb{R}^{n_L}$ defined by

$$N(\mathbf{z}) = \psi_L \circ \psi_{L-1}(\cdot, \mathbf{z}) \circ \dots \circ \psi_2(\cdot, \mathbf{z}) \circ \psi_1(\mathbf{z}),$$

where $\mathbf{z}$ is the input vector of the network, n_0 and n_L are respectively the size of input vector, and of output vector and the functions ψ_l are defined as follows:

$$\begin{cases} \psi_l(\mathbf{x}) = \phi_l(\mathbf{W}_l \mathbf{x} + \mathbf{b}_l), & l = 1, L, \\ \psi_l(\mathbf{x}, \mathbf{z}) = \phi_l \left(\mathbf{W}_l \begin{pmatrix} \mathbf{x} \\ \mathbf{z} \end{pmatrix} + \mathbf{b}_l \right), & l = 2, \dots, L - 1, \end{cases}$$

where $\phi_l(\cdot) : \mathbb{R} \rightarrow \mathbb{R}$ is the activation function of layer l (applied componentwise), $\mathbf{W}_1 \in \mathbb{R}^{n_1 \times \mathbb{R}^{n_0}}$, $\mathbf{W}_L \in \mathbb{R}^{n_L \times \mathbb{R}^{n_{L-1}}}$, $\mathbf{W}_l \in \mathbb{R}^{n_l \times \mathbb{R}^{n_{l-1} + n_0}}$ are respectively the weight matrix of layers 1, L and $l = 2, \dots, L - 1$, and $\mathbf{b}_l \in \mathbb{R}^{n_l}$ is the bias vector of layer $l = 1, \dots, L$. The entire set of neural weights ($\mathbf{W}_l$ and $\mathbf{b}_l$) is denoted by the vector Θ .

The Figure [2.4] gives an example of a neural network with skip connections. This kind of network is also known as a residual neural network (ResNN).

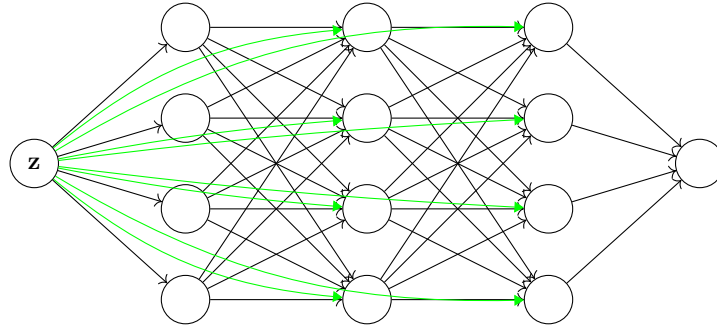


Figure 2.4: Neural network with skip connections (in green).

Other neural network architectures may also be considered, such as Recurrent Neural Networks (RNNs) or Convolutional Neural Networks (CNNs).

In our context, we consider a neural network with one neuron on the input layer taking the state variables and the parameters as a vector in input. We also consider one neuron on the output layer that gives the derivative price. For each of the tested architecture, we can choose hyperparameters such as the number of hidden layers and the number of neurons by hidden layer. We could also choose the activation function after each hidden neuron, however we always use the tanh function because it has a good behaviour and stays general. For the output layer, as we are in a regression task, we use the identity function as activation function.

The network takes as input a vector of dimension 8, denoted $\mathbf{z}$, containing scaled time and state variables, parameters and scaled maturity:

$$\mathbf{z} := \left(\tilde{t}, \tilde{\mathbf{y}}_t, \kappa, \sigma_r, \sigma_S, \rho, \tilde{T} \right)^\top.$$

The neural network, denoted $N(\cdot)$, approximates the price of the derivative, i.e.:

$$N(\mathbf{z}) \approx V(\tilde{t}, \tilde{\mathbf{y}}_t | \kappa, \sigma_r, \sigma_S, \rho, \tilde{T}).$$

2.4 Calibration

Once a network architecture has been chosen, the network needs to be calibrated. This stage is also known as the training phase. The aim of each network is to approximate the value function solving the scaled FK Equation (2.7). The model is trained by minimizing a loss function that is proportional to errors of approximation. This error is measured by replacing $V(\cdot)$ with $N(\cdot)$ in the scaled FK equation, at random points in the domain. It is also possible to minimize the error on boundary conditions, the network must give the value of the payoff at maturity for example. Finally, we can also add the error between the value given by the network and exact prices found on the market. These loss terms and the general loss function will be developed for European and Bermudan cases in the following chapters.

The error being measured at random points in the domain, we must first determine the domain of state variables and parameters. Then, we discuss how to select random training points. Finally, we describe the method used to save the results, compare them and select the best neural network.

2.4.1 Domain and range of state variables and parameters

The aim is to train the neural network for several values of the state variables and also for several values of the parameters. It is interesting to have a model that provides an estimate when the parameter values are modified and not just for fixed parameter values. This is possible with neural networks. To do it, input the parameter range to the neural network, and it will be calibrated for several parameter values. We therefore need to determine both the domains of the state variables and of the parameters.

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

$$\mathcal{D}^y : [r_l, r_u] \times [S_l, S_u]. \quad (2.8)$$

The parameters are subject to the following constraints:

$$\begin{cases} \kappa & \in [\kappa_l, \kappa_u], \\ \sigma_r & \in [\sigma_{rl}, \sigma_{ru}], \\ \sigma_S & \in [\sigma_{Sl}, \sigma_{Su}], \\ \rho & \in (-1, 1), \\ T & \in [0, T_{max}], \\ t & \leq T. \end{cases} \quad (2.9)$$

There is no clearly defined method for determining the ranges, low (l) and high (u) values, for state variables and parameters. Consequently, we must rely on "expert judgment" and establish these ranges in a coherent manner. Generally, we aim to have a broad range for state variables to cover as many scenarios as possible. For parameters, we also seek to encompass various scenarios while staying close to market realities. A possible approach is to use the standard value of a parameter and set the low and high values as the standard value $\pm 40\%$. With this in mind and considering the parameter values used in other studies (Hainaut (2023), Hainaut and Casas (2024)), we have chosen the following values:

$$\begin{cases} r_t & \in [0.005, 0.07], \\ S_t & \in [20, 250], \end{cases} \quad \begin{cases} \kappa & \in [0.5, 2], \\ \sigma_r & \in [0.002, 0.02], \\ \sigma_S & \in [0.02, 0.3], \\ \rho & \in [-0.8, 0.8], \\ T & \in [0, 10]. \end{cases} \quad (2.10)$$

It should also be noted that the payoff of the derivative, $H(S_T)$, may depend on additional parameters such as a strike price K . In such cases, it would be beneficial to include this parameter in the network input and calibrate the network for various values of this parameter. However, for the scenarios we will consider (call or put options), where the payoff is piecewise linear, we can use a heuristic approach to evaluate derivatives with different strike values. Let us temporarily denote the option price at time t by $V(S_t, K)$ for an asset value S_t and a strike price K . The value of an option with a strike price $K' \neq K$ can then be calculated as:

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

and can thus be assessed using the network without the need for retraining. When a strike value is required later, we take $K = 100$ without loss of generality.

2.4.2 Training points

The objective is to train the neural network using values within the ranges defined in the previous section. Given the infinite number of potential values, we will train the network on a finite set of randomly selected points within this domain. As explored by Al-Aradi et al. (2018) in the context of DGM networks, the methodology for sampling training points is crucial. They attempted to sample according to the distribution of state variables (such as the log-normal distribution for stock prices), but this approach yielded poor results. The best outcomes were achieved by sampling uniformly beyond the region of interest. For example, if we are interested in derivative prices for stock prices up to 100, we should train the network using stock prices greater than 100. These observations are validated with other network architectures. Given that the domain of state variables and parameters has been arbitrarily chosen and is relatively broad (meaning we cover various market conditions), we will not consider training points outside this domain. Consequently, it is important to remain cautious when estimating the network's performance at the boundary values (l or u) or beyond them.

As explained in the beginning of the Calibration section, the loss function to minimize can be composed of different kind of loss: residual loss in the scaled FK equation on the inner domain, loss on some boundary conditions and loss with some exact prices. For each type of loss, we must have a set of training points:

- Inner sample set: both variables and parameters are uniformly sampled from the entire domain, between l and u values;
- Boundary sample sets: one variable is fixed (for example, $t = T$ for an expiry condition or $S_t = S_l \forall t$ for a lower condition), the other variables and parameters are uniformly sampled;
- Exact price sample set: the variables and parameters are chosen and fixed to correspond to the exact prices selected on the market.

During the training, we use $n_{\mathcal{D}} = 20,000$ combinations of state variables and parameters to compute the inner error. Each boundary error is assessed with a sample set of size $n_{bound} = 5,000$. For the exact prices' error, the number of points considered is $n_{exact} = 400$.

To ensure stabilized learning process and good convergence, it is better to normalize the inputs of the network. We start by drawing a uniform sample of $n_{\mathcal{D}}$ realizations of $\mathbf{y}_t$ in $\mathcal{D}^y$, (2.8), at random times. We also sample uniformly parameters $(\kappa_j, \sigma_{rj}, \sigma_{Sj}, \rho_j, T_j)_{j=1, \dots, n_{\mathcal{D}}}$ under the constraints given in (2.9). The set of samples state variables and parameters, called the inner sample set, is noted $\mathcal{S}_{\mathcal{D}} = (t_j, \mathbf{y}_j, \kappa_j, \sigma_{rj}, \sigma_{Sj}, \rho_j, T_j)_{j=1, \dots, n_{\mathcal{D}}}$. We next centre and scale state variables and times. The mean and standard deviation of sampled state variables are computed respectively with the following standard formulas:

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

The scaling vectors are defined by $\mathbf{a} = -\frac{\bar{\mathbf{y}}}{\mathbf{S}_y}$ and $\mathbf{b} = \frac{1}{\mathbf{S}_y}$. For $j = 1, \dots, n_{\mathcal{D}}$, we define $\tilde{\mathbf{y}}_j = \mathbf{a} + \mathbf{b} \odot \mathbf{y}_j$, the scaled or normalized state variables. The times and derivative maturity

are also rescaled with coefficients $a_h = -\frac{1}{2}$ and $b_h = \frac{1}{T_{max}}$. The set of sampled normalized state variables and parameters is denoted by:

$$\tilde{\mathcal{S}}_{\mathcal{D}} = \left(\tilde{t}_j, \tilde{y}_j, \kappa_j, \sigma_{rj}, \sigma_{Sj}, \rho_j, \tilde{T}_j \right)_{j=1, \dots, n_{\mathcal{D}}} . \quad (2.11)$$

During the training phase, the error of approximation is measured for all points of this sample sets with the scaled FK equation. We also consider the error at expiry, $t = T$, which is a boundary error and is measured with another sample set, denoted by $\tilde{\mathcal{S}}_T$, of $n_T = n_{bound}$ new realizations of state variables and parameters:

$$\tilde{\mathcal{S}}_T = \left(\tilde{y}_k, \kappa_k, \sigma_{rk}, \sigma_{Sk}, \rho_k, \tilde{T}_k \right)_{k=1, \dots, n_T} . \quad (2.12)$$

We can also consider a lower boundary constraint corresponding to the case where the stock price falls to zero ($S_t = 0$). In this case, the error is measured with another sample set, denoted by $\tilde{\mathcal{S}}_{low}$, of $n_{low} = n_{bound}$ new realizations of state variables and parameters:

$$\tilde{\mathcal{S}}_{low} = \left(\tilde{t}_l, (\tilde{r}_l, a_S)^\top, \kappa_l, \sigma_{rl}, \sigma_{Sl}, \rho_l, \tilde{T}_l \right)_{l=1, \dots, n_{low}} . \quad (2.13)$$

For these two boundary constraints, the realizations of state variables are scaled with the same coefficients **a** and **b** computed from the inner sample set. The construction of the exact price sample set will be discussed separately for European and Bermudan cases because the way in which it is obtained differs from case to case.

The determination of the loss function will also be carried out separately for the European and Bermudian cases in the following chapters. Indeed, we consider the residual loss in the scaled FK equation (also called the inner loss) in both cases. The loss at maturity is also considered in each case but does not have exactly the same formulation (we talk about Exercise loss in the case of Bermudan options). Whereas, the loss on the lower sample set and the loss with exact prices will depend on the case considered.

2.4.3 Learning phases and metrics

To train the network, we use a stochastic gradient descent algorithm. It allows minimizing the loss function at each step of the algorithm. We also use three random batches at each epoch, where a batch refers to a subset of the training data processed at once, and an epoch represents a full cycle through the entire dataset. The use of batches improves convergence and reduces overfitting. To ensure a fair comparison of different neural architectures, we adhere to a consistent calibration protocol across all tested configurations. This protocol consists of four phases with a decreasing learning rate and a variable number of iterations per phase, as detailed in Table [2.2]. Reducing the learning rate per phase decreases the number of iterations compared to maintaining a constant low learning rate, thereby shortening the model execution time and enabling faster calibration. Although more iterations could potentially yield lower errors for some architectures, we use the same number of epochs for all networks to facilitate error comparison after 3,500 iterations. Standardizing the calibration process allows for a fair and accurate evaluation of each network's performance.

Phase	Learning rate	Number of epochs
1	0.005	500
2	0.002	1,000
3	0.001	1,000
4	0.0001	1,000

Table 2.2: Learning rates and number of epochs used in the four training phases.

Training time also depends on the network architecture, the more layers and/or neurons per layer, the longer it takes to calibrate. A more complex architecture such as DGM layers also takes longer to calibrate than a ResNN or FNN architecture. For a single neural network (Bermudan case requiring several neural networks), the total calibration time can take up to 7 hours for a DGM network with 5 layers and 256 neurons per layer. The equivalent 5-256-ResNN and 5-256-FNN networks have a very similar training time and take less than 2 hours to be calibrated. DGM networks being more accurate, there is a tradeoff between accuracy and training time. Furthermore, DGM networks having much more parameters, they can overfit the data. Overfitting occurs when a model learns the training data too well, including its noise and outliers, resulting in poor generalization to new, unseen data. This leads to high accuracy on training data but low accuracy on validation or test data. We therefore need to consider metrics that quantify overfitting.

Once calibrated, the comparison of networks is based on several performance metrics: the total loss function and its components after training, the mean square error (MSE) between the price given by the PINN and the price given by a validation procedure. We consider two type of MSE, the MSE_{alea} which is computed on a new sample set (other than the sets used for training) of 5,000 combinations of state variables and parameters, and MSE_{conf} which is computed for a fixed set of parameters, given in Table [2.3], with random values of r_t and S_t between low and high values, also on a sample set of 5,000 observations.

κ	1.15	σ_r	0.01
σ_S	0.025	ρ	-0.4

 Table 2.3: Fixed market parameters for the computation of MSE_{conf} .

Additionally, we measure the relative error for in-the-money options ($S_t < K$ for a put and $S_t > K$ for a call), denoted as Err_{rel} , calculated on a random set of 5,000 combinations of parameters and state variables. Mean square error and relative error are computed as follows:

$$MSE = \frac{1}{n} \sum_{i=1}^n \left(N(\tilde{t}_i, \tilde{\mathbf{y}}_i, \kappa_i, \sigma_{ri}, \sigma_{Si}, \rho_i, \tilde{T}_i) - P^{exact}(t_i, \mathbf{y}_i, \kappa_i, \sigma_{ri}, \sigma_{Si}, \rho_i, T_i) \right)^2,$$

$$Err_{rel} = \frac{1}{n} \sum_{i=1}^n \frac{|N(\tilde{t}_i, \tilde{\mathbf{y}}_i, \kappa_i, \sigma_{ri}, \sigma_{Si}, \rho_i, \tilde{T}_i) - P^{exact}(t_i, \mathbf{y}_i, \kappa_i, \sigma_{ri}, \sigma_{Si}, \rho_i, T_i)|}{|P^{exact}(t_i, \mathbf{y}_i, \kappa_i, \sigma_{ri}, \sigma_{Si}, \rho_i, T_i)|},$$

where $N(\cdot)$ is the output of the neural network and $P^{exact}(\cdot)$ is the price given by the validation procedure. For the relative error, we consider the case in-the-money, so we only take stock price S_t above (resp. under) K for a call (resp. put) option. The losses compute how well the

model fit to the training data, whereas MSE and relative error allow quantifying how well the model generalized to new data. Metrics close to zero indicate good model performance. Using the same datasets across all proposed neural architectures facilitates easy comparison of results.

The validation procedure depends on the case considered. For example, in the European case, there exists a closed-form valuation while for the Bermudan case, we must rely on another estimation procedure because no closed-form solutions exist. For this reason, the validation procedure is described for each case in the following chapters.

It is important to note that the calibration procedure, including the number of iterations, learning rate, and choice of optimizer, relies heavily on empirical practices rather than theoretical results. Empirical use is crucial for selecting the right calibration, necessitating a trial-and-error phase in practice.

2.4.4 Implementation

The calibration procedure has been implemented in *Python* using the `tensorflow` library and the `keras` neural networks API. This library allows to define the neural networks by creating new classes that depend on the class `keras.Model`. Also, the partial derivatives needed for evaluating the inner loss are computed by automatic differentiation using `GradientTape(.)` and `tape.Gradient(.)` of `tensorflow`. This technique is not based on finite differences and provides high accuracy. Finally, it provides optimizers to train the networks. Here, we use the ADAM optimizer (`keras.optimizers.Adam(.)`), which is a variant of the stochastic gradient descent.

The computer used for the execution is equipped with an *Intel Core i7-10700K CPU (3.8G Hz/3.79 GHz)* processor, with 32 GB of RAM and with an *AMD Radeo RX 5700 XT GPU*.

A *Python* script has been developed for each case of Chapters 3 and 4. Two other scripts are the original versions of the scripts adapted with the model extension from Chapter 5. All these scripts as well as the one to calibrate the Nelson-Siegel parameters and the one for the validation process of the Bermudian case are available on the following GitHub repository https://github.com/Marie-Thibeaue/LACTU2900_research_master_thesis.

Chapter 3

European call option pricing with a single interest rate curve

Let's consider the case of a European derivative expiring at time T and with payoff, $H(S_T)$, function of stock price at maturity. Here we are interested in a call option because Bermudan call or even American call have the same value as the European call. We could also develop the case of the put option in an equivalent way, by considering another payoff. In the case of a call, the payoff with strike K is

$$H(S_T) = (S_T - K)^+.$$

And the fair value of the derivative, denoted $F(\cdot)$, is equal to the expected discounted cash-flows under the risk neutral measure $\mathbb{Q}$:

$$F(t, T, r_t, S_t) = \mathbb{E}^{\mathbb{Q}} \left(e^{-\int_t^T r_s ds} (S_T - K)^+ | \mathcal{F}_t \right).$$

The aim is to estimate the price of this derivative, which corresponds to the fair value evaluated in $t = 0$, by using a neural network. We already know thanks to the Chapter 2 that the derivative value respects the scaled FK Equation (2.7). Moreover, European derivatives satisfy some additional boundary constraints developed here after in Section 3.1. These constraints allow obtaining a loss function that has to be minimized by the neural network in order to well estimate the price of the derivative, see Section 3.2. To ensure that the estimation by the neural network is accurate, we develop a validation procedure in Section 3.3. Finally, we perform numerical analysis with different type and architecture of neural networks in Section 3.4 and perform sensitivity testing on the best network selected.

3.1 Boundary constraints

Recall that the derivative price is noted $V(t, \mathbf{y}_t | \kappa, \sigma_r, \sigma_S, \rho, T)$. Then, the boundary constraint on $V(\cdot)$ at expiry depends on the payoff:

$$V(T, \mathbf{y}_T | \kappa, \sigma_r, \sigma_S, \rho, T) = F(T, T, r_T, S_T) = H(S_T) = (S_T - K)^+.$$

For most current European payoffs, the option value if the stock price falls to zero ($S_t = 0$) is simply the discounted value of a constant payoff. In this case, we consider a lower boundary constraint on $V(\cdot)$:

$$V(t, (r_t, 0)^\top | \kappa, \sigma_r, \sigma_S, \rho, T) = F(t, T, r_t, 0) = \mathbb{E}^{\mathbb{Q}} \left(e^{-\int_t^T r_s ds} H(0) | \mathcal{F}_t \right) = 0.$$

This lower bound is not mandatory, and we can test in the numerical analysis whether adding it gives better results.

Since we use a scaled version of the FK equation, the boundary constraints must be adapted accordingly. Recall that the value of the contract can also be seen as a function of scaled time and state variables, $V(\tilde{t}, \tilde{\mathbf{y}}_t | \kappa, \sigma_r, \sigma_S, \rho, \tilde{T})$. The rescaled version of the terminal and lower boundary conditions are, respectively:

$$V(\tilde{T}, \tilde{\mathbf{y}}_T | \kappa, \sigma_r, \sigma_S, \rho, \tilde{T}) = H\left(\frac{\tilde{S}_T - a_S}{b_S}\right) = \left(\frac{\tilde{S}_T - a_S}{b_S} - K\right)^+,$$

$$V(\tilde{t}, (\tilde{r}_t, a_S)^\top | \kappa, \sigma_r, \sigma_S, \rho, \tilde{T}) = \mathbb{E}^\mathbb{Q}\left(e^{-\int_{(\tilde{t}-a_h)/b_h}^{(\tilde{T}-a_h)/b_h} \frac{\tilde{r}_s - a_r}{b_r} ds} H(0) | \mathcal{F}_t\right) = 0.$$

3.2 Loss function

We will determine the loss function to be minimized by the neural network. Recall that the neural network approximates the value function solving the scaled FK equation by taking as input a vector of dimension 8, denoted $\mathbf{z}$, containing scaled time and state variables, parameters and scaled maturity:

$$\mathbf{z} := (\tilde{t}, \tilde{\mathbf{y}}_t, \kappa, \sigma_r, \sigma_S, \rho, \tilde{T})^\top.$$

The neural network, denoted $N(\cdot)$, approximates the fair value of the European derivative, i.e. $N(\mathbf{z}) \approx V(t, \tilde{\mathbf{y}}_t | \kappa, \sigma_r, \sigma_S, \rho, \tilde{T})$.

Regardless of the chosen architecture, the neural network is trained by minimizing a loss function. This loss function comprises multiple terms that represent the desired approximations of the network's output. For the European case, the network must satisfy three conditions: the scaled FK equation, the maturity boundary condition, and the lower boundary condition. Consequently, the loss function includes three terms, each calculated on distinct sample sets of training points as described in Section 2.4.2.

Recall that a neural network is characterized by a set of weights, denoted Θ . At points of $\tilde{\mathcal{S}}_\mathcal{D}$, defined in (2.11), we define for $j = 1, \dots, n_\mathcal{D}$, the error in scaled FK Equation (2.7) when V is replaced by the neural network as follows:

$$e_j^\mathcal{D}(\Theta) = b_h \partial_{t_j} N - \frac{\tilde{r}_j - a_r}{b_r} N + \mu_{\tilde{\mathbf{y}}}(t_j, \tilde{\mathbf{y}}_j)^\top \nabla_{\tilde{\mathbf{y}}} N + \frac{1}{2} \text{tr}\left(\Sigma_{\tilde{\mathbf{y}}}(t_j, \tilde{\mathbf{y}}_j) \Sigma_{\tilde{\mathbf{y}}}(t_j, \tilde{\mathbf{y}}_j)^\top \mathcal{H}_{\tilde{\mathbf{y}}}(N)\right).$$

We refer to $e_j^\mathcal{D}$ as the error on the interior 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,

$$\mathcal{L}_\mathcal{D}(\Theta) = \frac{1}{n_\mathcal{D}} \sum_{j=1}^{n_\mathcal{D}} e_j^\mathcal{D}(\Theta)^2.$$

On the maturity boundary sample set $\tilde{\mathcal{S}}_T$, defined in Equation (2.12), we define the error $e_k^T(\Theta)$ for $k = 1, \dots, n_T$, as the difference between the output of the neural network and the payoff at expiry:

$$e_k^T(\Theta) = N(\tilde{T}_k, \tilde{\mathbf{y}}_k, \kappa_k, \sigma_{rk}, \sigma_{Sk}, \rho_k, \tilde{T}_k) - \left(\frac{\tilde{S}_T - a_S}{b_S} - K\right)^+.$$

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

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

On the lower boundary sample set $\tilde{\mathcal{S}}_{low}$, defined in Equation (2.13), we define the error $e_l^{low}(\Theta)$ 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:

$$e_l^{low}(\Theta) = N(\tilde{t}_l, (\tilde{r}_l, a_S)^\top, \kappa_l, \sigma_{rl}, \sigma_{Sl}, \rho_l, \tilde{T}_l) - 0.$$

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

$$\mathcal{L}_{low}(\Theta) = \frac{1}{n_{low}} \sum_{l=1}^{n_{low}} e_l^{low}(\Theta)^2.$$

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

$$\Theta_{opt} = \arg \min_{\Theta} [\mathcal{L}_{\mathcal{D}}(\Theta) + \mathcal{L}_T(\Theta) + \mathcal{L}_{low}(\Theta)].$$

In practice, we assign weights to each component of the loss function to explore different configurations, prioritizing some components. The minimization problem becomes,

$$\Theta_{opt} = \arg \min_{\Theta} [w_{\mathcal{D}} \mathcal{L}_{\mathcal{D}}(\Theta) + w_T \mathcal{L}_T(\Theta) + w_{low} \mathcal{L}_{low}(\Theta)]. \quad (3.1)$$

In a first time, we do not take into account the loss on exact prices. However, we will analyse the effect of adding it. To do so, we need to start by constructing a set of exact prices. We could look for prices directly on the market but, as we will see in the validation procedure, Section 3.3, there is a closed-form formula for the price of European options in a financial market as described in Section 2.1. We therefore construct a sample set, denoted $\mathcal{S}_{exact}$, with $n_{exact} = 400$ new realizations of state variables and parameters:

$$\mathcal{S}_{exact} = (t_m, \mathbf{y}_m, \kappa_m, \sigma_{rm}, \sigma_{Sm}, \rho_m, T_m)_{m=1, \dots, n_{exact}}.$$

The closed-form formula, $P^{exact}(\cdot)$, is used to obtain a price for each realization in this set. This set can be normalised with the same scaling parameters **a** and **b** obtained in Section 2.4.2:

$$\tilde{\mathcal{S}}_{exact} = (\tilde{t}_m, \tilde{\mathbf{y}}_m, \kappa_m, \sigma_{rm}, \sigma_{Sm}, \rho_m, \tilde{T}_m)_{m=1, \dots, n_{exact}},$$

and we obtain an estimate of the price given by the neural network for each realization of $\tilde{\mathcal{S}}_{exact}$. Finally, we define the error $e_m^{exact}(\Theta)$ for $m = 1, \dots, n_{exact}$, as the difference between the output of the neural network and the corresponding exact price:

$$e_m^{exact}(\Theta) = N(\tilde{t}_m, \tilde{\mathbf{y}}_m, \kappa_m, \sigma_{rm}, \sigma_{Sm}, \rho_m, \tilde{T}_m) - P^{exact}(t_m, \mathbf{y}_m, \kappa_m, \sigma_{rm}, \sigma_{Sm}, \rho_m, T_m).$$

The average quadratic loss for this condition can be added to the total loss function:

$$\mathcal{L}_{exact}(\Theta) = \frac{1}{n_{exact}} \sum_{m=1}^{n_{exact}} e_m^{exact}(\Theta)^2. \quad (3.2)$$

3.3 Validation procedure

To ensure the accuracy and reliability of the neural network model developed for approximating the price of a European call option, we will validate the results using a closed-form formula. Validation is a crucial step in the modelling process, as it allows us to compare the neural network's output against established analytical solutions, thereby confirming its correctness and robustness. By comparing the approximated prices generated by the neural network with those obtained from a closed-form solution, we can quantitatively assess the performance and accuracy of our model. This procedure involves calculating the option prices using the closed-form formula for various sets of input parameters and comparing these values with the corresponding outputs from the neural network. Discrepancies between the two sets of results will help identify areas where the neural network may need further tuning or improvement. This step not only provides confidence in the model's predictions but also highlights the neural network's capability to generalize and accurately replicate known solutions under different conditions.

To obtain the closed-form solution for the fair value, we start from its expression as the expected discounted cash-flows under the risk-neutral measure $\mathbb{Q}$. Recall that we have a function that depends on all state variables:

$$F(t, T, r_t, S_t) = \mathbb{E}^{\mathbb{Q}} \left(e^{-\int_t^T r_s ds} (S_T - K)^+ \middle| \mathcal{F}_t \right).$$

We then perform a change of measure using as Radon-Nikodym density:

$$\frac{d\mathbb{F}}{\mathbb{Q}} \bigg|_T = \mathbb{E}^{\mathbb{Q}} \left(\frac{d\mathbb{F}}{\mathbb{Q}} \middle| \mathcal{F}_T \right) = \frac{e^{-\int_0^T r_s ds}}{\mathbb{E}^{\mathbb{Q}} \left(e^{-\int_0^T r_s ds} \middle| \mathcal{F}_0 \right)}.$$

From Equation (A.1), this density can be rewritten as follows:

$$\frac{d\mathbb{F}}{\mathbb{Q}} \bigg|_T = \exp \left[-\int_0^T \sigma_r B(u, T) dW_u^{(1)} - \frac{1}{2} \int_0^T \sigma_r^2 B^2(u, T) du \right], \quad (3.3)$$

where we recognize a Doleans-Dade exponential. As a consequence, under the measure $\mathbb{F}$, we denote $\mathbf{W}_t^{\mathbb{F}}$, the $\mathbb{F}$ -Brownian motions $(W_t^{(1)\mathbb{F}}, W_t^{(2)\mathbb{F}})$, defined by

$$d\mathbf{W}_t^{\mathbb{F}} = d\mathbf{W}_t + \sigma_r B(t, T) \Sigma_r^\top dt, \quad \Rightarrow \quad \begin{cases} dW_t^{(1)\mathbb{F}} = dW_t^{(1)} + \sigma_r B(t, T) dt, \\ dW_t^{(2)\mathbb{F}} = dW_t^{(2)}. \end{cases}$$

The dynamic of the stock S_t under the measure $\mathbb{F}$ is then given by

$$\frac{dS_t}{S_t} = (r_t - \rho \sigma_r \sigma_S B(t, T)) dt + \sigma_S \rho dW_t^{(1)\mathbb{F}} + \sigma_S \sqrt{1 - \rho^2} dW_t^{(2)\mathbb{F}}.$$

As $S_T = S_T/P(T, T)$, we show that $S_t/P(t, T)$ follows a log-normal distribution, the proof can be found in Appendix C.

Proposition 3.1. *The quantity $\frac{S_t}{P(t,T)}$ has a log-normal distribution,*

$$\ln \frac{S_T/P(T,T)}{S_t/P(t,T)} \sim \mathcal{N}(\mu_{\mathbb{F}}(t), v_{\mathbb{F}}(t)),$$

with time-dependent variance and mean equal to

$$v_{\mathbb{F}}(t) = \int_t^T (\sigma_r B(u, T) \Sigma_r^\top + \sigma_S \Sigma_S^\top) (\sigma_r B(u, T) \Sigma_r + \sigma_S \Sigma_S) du, \quad (3.4)$$

$$\mu_{\mathbb{F}}(t) = -\frac{1}{2} v_{\mathbb{F}}(t). \quad (3.5)$$

The variance (3.4) and mean (3.5) of the above proposition can be rewritten as follows:

$$\begin{aligned} v_{\mathbb{F}}(t) &= \int_t^T \sigma_r^2 B^2(u, T) + 2\rho\sigma_r\sigma_S B(u, T) + \sigma_S^2 du, \\ &= \frac{\sigma_r^2}{\kappa^2} \left[(T-t) - 2B(t, T) + \frac{(1 - e^{-2\kappa(T-t)})}{2\kappa} \right] + \frac{2\rho\sigma_r\sigma_S}{\kappa} \left[(T-t) - \frac{1}{\kappa} B(t, T) \right] \\ &\quad + \sigma_S^2 (T-t), \\ \mu_{\mathbb{F}}(t) &= -\frac{1}{2} v_{\mathbb{F}}(t), \\ &= \frac{-\sigma_r^2}{2\kappa^2} \left[(T-t) - 2B(t, T) + \frac{(1 - e^{-2\kappa(T-t)})}{2\kappa} \right] - \frac{\rho\sigma_r\sigma_S}{\kappa} \left[(T-t) - \frac{1}{\kappa} B(t, T) \right] \\ &\quad - \frac{\sigma_S^2}{2} (T-t). \end{aligned}$$

As $\frac{S_t}{P(t,T)}$ is log-normal under the $\mathbb{F}$ -measure, we can use a standard Black & Scholes development to price the European call option and obtain the following corollary.

Corollary 3.2. *The expected positive difference between S_T and K under the $\mathbb{F}$ -measure is given by*

$$\mathbb{E}^{\mathbb{F}} \left((S_T - K)^+ \middle| \mathcal{F}_t \right) = \frac{S_t}{P(t, T)} \Phi(h_1(t)) - K \Phi(h_2(t)),$$

where $\Phi(\cdot)$ is the cumulative distribution function of a $\mathcal{N}(0, 1)$ and,

$$h_{1,2}(t) = \frac{\ln \left(\frac{S_t}{KP(t,T)} \right) \pm \frac{1}{2} v_{\mathbb{F}}(t)}{\sqrt{v_{\mathbb{F}}(t)}}.$$

Then, the closed-form expression of the fair value is given by,

$$F(t, T, r_t, S_t) = P(t, T) \mathbb{E}^{\mathbb{F}} \left((S_T - K)^+ \middle| \mathcal{F}_t \right) = S_t \Phi(h_1(t)) - KP(t, T) \Phi(h_2(t)),$$

where the expression of the price of the zero coupon bond, $P(t, T)$, is given in Equation (A.6). To emphasize that this exact price depends also on the market parameters, we use the following expression:

$$P^{exact}(t, r_t, S_t, \kappa, \sigma_r, \sigma_S, \rho, T) := F(t, T, r_t, S_t)|_{\kappa, \sigma_r, \sigma_S, \rho}.$$

3.4 Numerical analysis

As already mentioned, the calibration procedure and the choice of the best model are based on trial and error. In this analysis, we take a step-by-step approach. To begin with, we consider the calibration procedure described in Section 2.4 and are interested in the optimal value of the weights of the loss function, see Equation (3.1). Once these weights have been determined, we select the best model by considering the architectures proposed in Section 2.3.3 and tuning the network hyperparameters (number of layers and number of neurons per layer). We then carry out more detailed analyses with the selected model in order to understand where the model's errors come from. Finally, still based on the selected model, we analyse the impact of a change in the calibration procedure by considering a different loss function, different learning rates or different training points.

3.4.1 Loss function weights selection

To determine the weights for each loss component in the total loss function, see Equation (3.1), we test several combinations of weights for a given network and try it with several networks (with different architecture or different hyperparameters). The Table [3.1] and Table [3.2] give the value of the metrics (losses and MSE/relative error, respectively) for 3 different networks (4-256-FNN, 4-256-ResNN and 3-128-DGM) and six combinations of loss weights for each network.

Type	# layers	# neurons per layer	Loss weights (w_D, w_T, w_{low})	Inner loss $\mathcal{L}_D$	Maturity loss $\mathcal{L}_T$	Lower loss $\mathcal{L}_{low}$
FNN	4	256	(1, 1, 0)	0.008576	0.003976	-
FNN	4	256	(1, 1, 1)	0.011137	0.005696	0.000013
FNN	4	256	(10, 1, 0)	0.003093	0.024504	-
FNN	4	256	(10, 10, 1)	0.068597	0.042205	0.006509
FNN	4	256	(100, 10, 1)	0.003190	0.020046	0.002541
FNN	4	256	(100, 100, 1)	0.056648	0.023692	0.010980
ResNN	4	256	(1, 1, 0)	0.005427	0.002659	-
ResNN	4	256	(1, 1, 1)	0.006380	0.002170	0.000093
ResNN	4	256	(10, 1, 0)	0.002007	0.011890	-
ResNN	4	256	(10, 10, 1)	0.006117	0.002793	0.000366
ResNN	4	256	(100, 10, 1)	0.002965	0.016705	0.001654
ResNN	4	256	(100, 100, 1)	0.005437	0.003412	0.006378
DGM	3	128	(1, 1, 0)	0.012568	0.005204	-
DGM	3	128	(1, 1, 1)	0.011970	0.006608	0.000170
DGM	3	128	(10, 1, 0)	0.002453	0.016412	-
DGM	3	128	(10, 10, 1)	0.014159	0.007714	0.001915
DGM	3	128	(100, 10, 1)	0.002912	0.019041	0.004295
DGM	3	128	(100, 100, 1)	0.008269	0.003686	0.008973

Table 3.1: Components of the loss for different combinations of loss weights.

Type	# layers	# neurons per layer	Loss weights $(w_{\mathcal{D}}, w_T, w_{low})$	MSE_{alea}	MSE_{conf}	Err_{rel}
FNN	4	256	(1, 1, 0)	48.291386	16.042750	0.047109
FNN	4	256	(1, 1, 1)	57.392666	15.702306	0.038927
FNN	4	256	(10, 1, 0)	66.364200	17.157763	0.051943
FNN	4	256	(10, 10, 1)	28.268341	15.323506	0.038823
FNN	4	256	(100, 10, 1)	63.804040	16.779830	0.045632
FNN	4	256	(100, 100, 1)	24.151052	17.465347	0.037802
ResNN	4	256	(1, 1, 0)	87.855515	20.281912	0.060843
ResNN	4	256	(1, 1, 1)	83.732350	19.250607	0.059610
ResNN	4	256	(10, 1, 0)	64.535230	17.918983	0.051823
ResNN	4	256	(10, 10, 1)	84.467560	18.978903	0.059233
ResNN	4	256	(100, 10, 1)	33.825924	16.829197	0.041081
ResNN	4	256	(100, 100, 1)	60.578377	17.650124	0.051103
DGM	3	128	(1, 1, 0)	19.204552	16.007290	0.033350
DGM	3	128	(1, 1, 1)	18.132400	15.296609	0.032595
DGM	3	128	(10, 1, 0)	19.690456	15.730644	0.033324
DGM	3	128	(10, 10, 1)	15.737797	16.096960	0.031239
DGM	3	128	(100, 10, 1)	24.078693	15.964383	0.035540
DGM	3	128	(100, 100, 1)	22.063920	15.439319	0.034640

Table 3.2: MSE and relative error for different combinations of loss weights.

It can be seen that there is no one combination of loss weights $(w_{\mathcal{D}}, w_T, w_{low})$ that stands out from the rest. The best combination changes depending on the network under consideration. Indeed, for the 4-256-FNN network, the best combination of weights is (10, 10, 1) where the loss components may be high but has the lowest MSE and relative error. For this network, the worst combination is (10, 1, 0) which has the highest MSE_{alea} and Err_{rel} , which could mean more overfitting. For the 4-256-ResNN network, the best combination is (100, 10, 1) and the worst is (1, 1, 0) with the same justification. Finally, for the 3-128-DGM network, the combinations (1, 1, 1) and (10, 10, 1) all have fairly low metrics and are therefore the best combinations. The worst weight combination is (100, 10, 1), which has the highest $\mathcal{L}_T$, MSE_{alea} and Err_{rel} .

It is therefore not easy to choose the best combination of weights for all networks. In fact, it would be ideal to choose the same weights in order to have the same basis for comparison between the models. To help us, we can see that the lower loss is almost always lower (or even much lower) than the other two error components (inner loss and maturity loss). Moreover, this error is not an intrinsic behaviour of the option price. It would therefore be interesting to test removing this loss component (see Section 3.4.4). To remain general, we will consider weights that place more emphasis on the inner loss and the maturity loss. We therefore choose to keep $(w_{\mathcal{D}}, w_T, w_{low}) = (10, 10, 1)$ for the rest of this analysis.

3.4.2 Network architecture and hyperparameters selection

Now that we've fully fixed the calibration procedure, we need to find the best model. To do so, we estimate several PINN's for pricing European call options in various market conditions. To begin with, we are interested in the architecture of the neural network: FNN, ResNN or DGM,

see their definition in Section 2.3.3. Once the architecture has been selected, we determine the hyperparameters of the network (number of layers and number of neurons per layer). Tables [3.3] and [3.4] show the values of the metrics (losses and MSE/relative error, respectively) for the three architectures and some combinations of hyperparameters.

Type	# layers	# neurons per layer	Inner loss $\mathcal{L}_{\mathcal{D}}$	Maturity loss $\mathcal{L}_T$	Lower loss $\mathcal{L}_{low}$
FNN	3	256	0.010741	0.005911	0.002468
ResNN	3	256	0.008968	0.004501	0.000663
DGM	3	256	0.007297	0.004211	0.000959
FNN	4	256	0.068597	0.042205	0.006509
ResNN	4	256	0.006117	0.002793	0.000366
DGM	4	256	0.005194	0.002976	0.000438
FNN	5	256	0.238345	0.198394	0.020351
ResNN	5	256	0.024113	0.015166	0.006847
DGM	5	256	0.006960	0.003419	0.000567

Table 3.3: Components of the loss for different architecture of networks.

Type	# layers	# neurons per layer	MSE_{alea}	MSE_{conf}	Err_{rel}	Training time [sec]
FNN	3	256	61.708527	18.274828	0.051735	0h45
ResNN	3	256	78.660390	19.929338	0.057846	0h53
DGM	3	256	15.103012	14.321458	0.030593	3h11
FNN	4	256	28.268341	15.323506	0.038823	0h57
ResNN	4	256	84.467560	18.918903	0.059233	1h08
DGM	4	256	15.592097	15.365858	0.030777	4h41
FNN	5	256	18.669065	36.561203	0.041578	1h16
ResNN	5	256	31.045568	15.481806	0.039722	1h21
DGM	5	256	15.556396	14.787518	0.031012	6h16

Table 3.4: MSE, relative error and training time for different architecture of networks.

It is easily observed that the DGM architecture generally has lower metrics compared to the other two architectures. This model being more complex, it better captures the dynamics of option pricing. However, a disadvantage of this complexity is that it takes significantly longer to calibrate, up to four times longer than the FNN and ResNN architectures. In some contexts, an excessively long calibration time could be limiting, making FNN or ResNN architectures preferable. Between these two, ResNN achieves lower loss components, while the other metrics are comparable to those of the FNN network. For the future, we will continue using a DGM network, keeping in mind that an ResNN network offers a shorter calibration time at the expense of some accuracy.

Now we perform the same analysis by varying the hyperparameters of the DGM architecture network. We consider the number of layers ranging from 2 to 5 and the number of neurons

per layer ranging from 32 to 256. Tables [3.5] and [3.6] show the values of the metrics (losses and MSE/relative error, respectively) for the sixteen combinations of hyperparameters for the DGM network.

Type	# layers	# neurons per layer	Inner loss $\mathcal{L}_{\mathcal{D}}$	Maturity loss $\mathcal{L}_T$	Lower loss $\mathcal{L}_{low}$
DGM	2	32	0.055600	0.049015	0.003404
DGM	2	64	0.058686	0.033184	0.001825
DGM	2	128	0.036678	0.020902	0.002037
DGM	2	256	0.010875	0.006178	0.002090
DGM	3	32	0.032745	0.019944	0.003076
DGM	3	64	0.017557	0.009805	0.002070
DGM	3	128	0.014159	0.007714	0.001915
DGM	3	256	0.007297	0.004211	0.000959
DGM	4	32	0.018398	0.012264	0.003435
DGM	4	64	0.014121	0.010352	0.002748
DGM	4	128	0.008296	0.004773	0.001668
DGM	4	256	0.005194	0.002976	0.000438
DGM	5	32	0.047036	0.031685	0.004759
DGM	5	64	0.014000	0.010479	0.002184
DGM	5	128	0.006570	0.003999	0.001173
DGM	5	256	0.006960	0.003419	0.000567

Table 3.5: Components of the loss for different hyperparameter of the DGM network.

We observe that networks with a higher number of layers and neurons achieve lower losses. This aligns with the notion that increasing the number of neural weights provides greater flexibility in optimizing the network. Consequently, for a given number of epochs, complex architectures exhibit a more significant reduction in loss components. However, with a complex network, it's important to avoid overfitting the training data by not excessively increasing the number of parameters (i.e., the number of layers and neurons per layer). Therefore, we use a validation process and calculate metrics such as MSE and relative error. We prefer a model with slightly higher loss components if it has lower MSE and relative error. Here, the lowest losses are achieved with the 4-256-DGM network, however it does not necessarily minimize the other metrics which indicates overfitting. We observe that the MSE_{conf} , that corresponds to a fixed configuration, presents signification variations. As for the Err_{rel} , it ranges between 3% and 5%. The 3-256-DGM network has the lowest MSE_{conf} , while the 2-256-DGM network has the lowest MSE_{alea} and Err_{rel} , though their losses are higher than the other two mentioned networks. A good compromise is to use the 3-256-DGM network. A more detailed analysis of this network's performance allows for a better understanding of the sources of errors.

Type	# layers	# neurons per layer	MSE_{alea}	MSE_{conf}	Err_{rel}	Training time
DGM	2	32	29.858930	18.405727	0.039008	0h17
DGM	2	64	21.262897	19.752580	0.035819	0h25
DGM	2	128	14.628018	19.345587	0.031930	0h43
DGM	2	256	12.596724	15.780653	0.029342	1h52
DGM	3	32	59.751500	18.918524	0.051104	0h38
DGM	3	64	42.663340	16.695179	0.044041	0h47
DGM	3	128	15.737797	16.096960	0.031239	1h26
DGM	3	256	15.103012	14.321458	0.030593	3h11
DGM	4	32	54.704956	18.109053	0.049406	0h46
DGM	4	64	58.595932	20.258114	0.050564	1h09
DGM	4	128	24.860506	16.079405	0.035807	2h08
DGM	4	256	15.592097	15.365858	0.030777	4h41
DGM	5	32	59.506424	17.603191	0.050631	1h01
DGM	5	64	63.273700	21.237148	0.052064	1h33
DGM	5	128	37.351090	17.160204	0.041608	3h29
DGM	5	256	15.556396	14.787518	0.031012	6h16

Table 3.6: MSE, relative error and training time for different hyperparameter of the DGM network.

3.4.3 Performances of the 3-256-DGM network

Firstly, we consider that the market is fixed, and we have a unique set of market parameters, shown in Table [3.7]. We then compare the price estimated by the PINN and the exact price given by the closed-form expression.

κ	1.15	σ_r	0.01
σ_S	0.025	ρ	-0.4

Table 3.7: Fixed market parameters.

The Figure [3.1] compares exact price and PINN price of the European call option of maturity $T = 5$ for several values of the interest rate, r_t . The left plots show an out-of-the-money case ($S_t < K$) and the right plots an in-the-money case ($K_t > K$). The upper plots show the difference at inception ($t = 0$) and the lower plots at maturity ($t = T$) with an intermediate time for middle plots. The Figure [3.2] compares exact price and PINN price of the European call option of maturity $T = 5$ for several values of the stock price, S_t , with fixed interest rate, $r_t = 0.03$. The left plot shows the difference at inception ($t = 0$) whereas the right plot shows it at maturity ($t = T$).

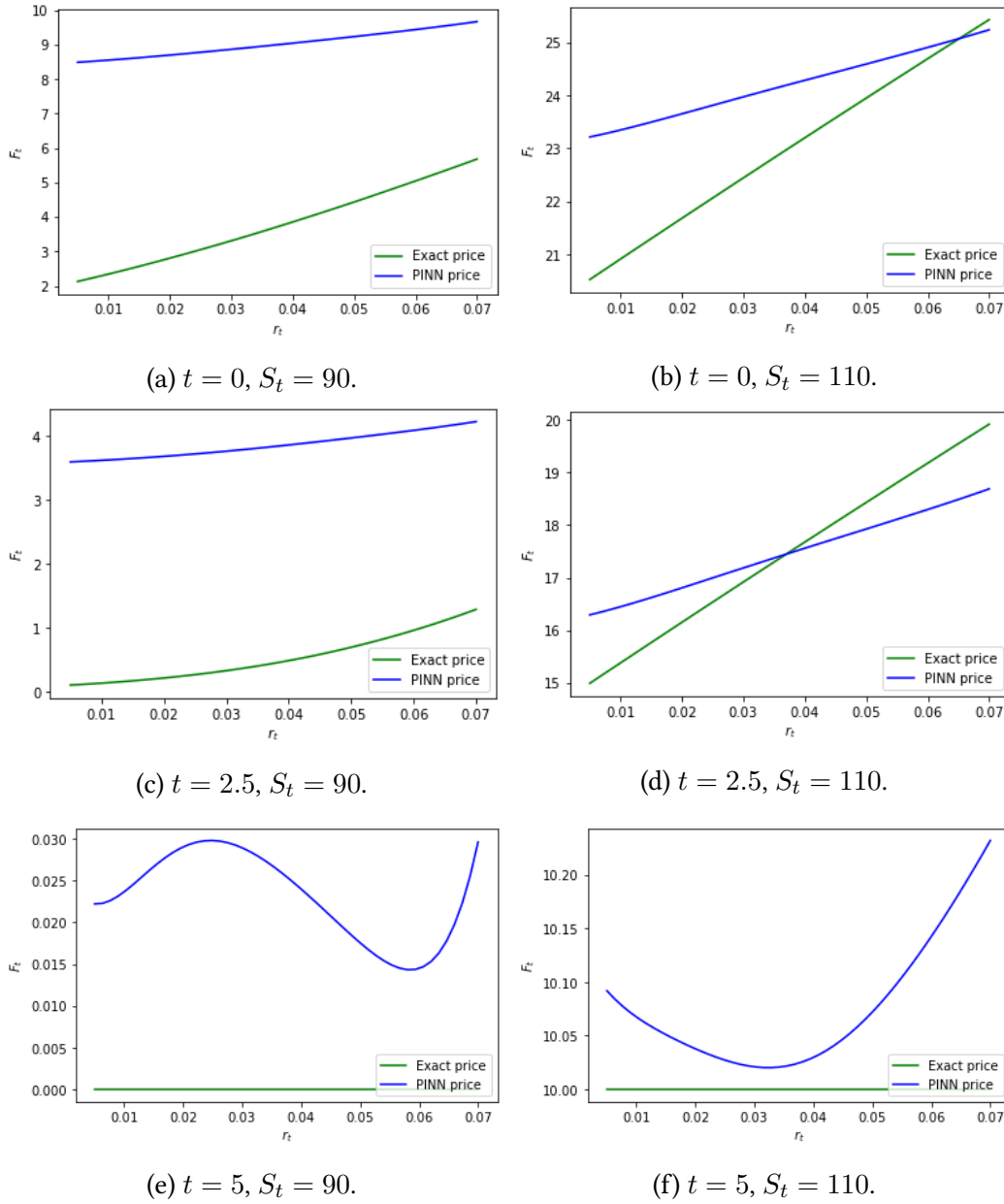


Figure 3.1: European call priced with PINN and close-form expression for $r_t \in [0.005, 0.07]$, $T = 5$.

Thanks to Figure [3.1], we observe that the difference between PINN price and exact price is getting close to zero as time goes to maturity. We also notice that the error is bigger in the out-of-the-money case, when $S_t < K$. The mean differences are $\{5.24, 3.33, 0.02\}$ and $\{1.22, 0.64, 0.08\}$ for $t = \{0, 2.5, 5\}$ for the out-of-the-money and in-the-money cases, respectively. The left plot of Figure [3.2] shows that the PINN price converges to the exact price for high value of S_0 . For $S_0 < K$, the PINN has troubles to replicate the exact price. The right plot of Figure [3.2] shows option prices at maturity, demonstrating that the maturity boundary condition is well satisfied. Between inception and maturity, the PINN price is getting closer to the closed-form solution as time passes.

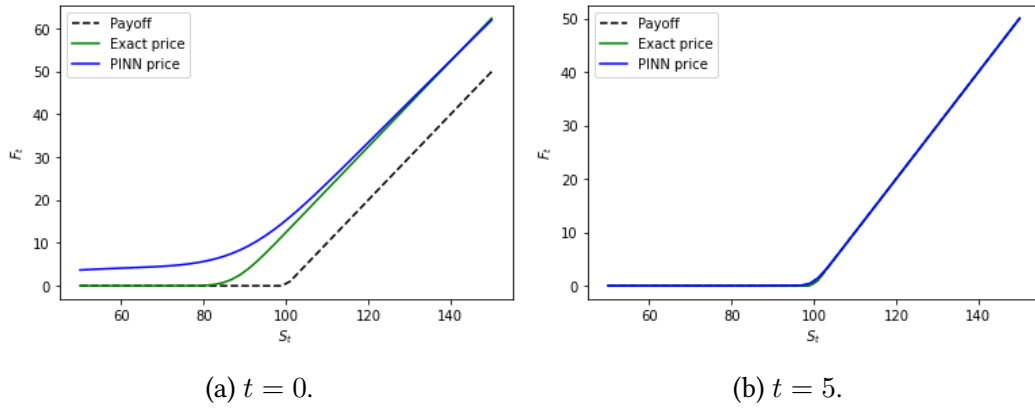


Figure 3.2: European call priced with PINN and close-form expression for $S_t \in [50, 150]$, $T = 5$, $r_t = 0.03$.

To include the maturity of the contract in our analysis, we consider the Figure [3.3] that gives heatmaps of absolute differences between PINN price and closed-form expression for maturities between 0 and 10. The above plots are for several values of the interest rate, $r_t \in [0.005, 0.07]$, and $S_t = 90$ (resp. $S_t = 110$) for the left (resp. right) plot. The lower plot is for several values of the stock price, $S_t \in [20, 250]$, and $r_t = 0.03$.

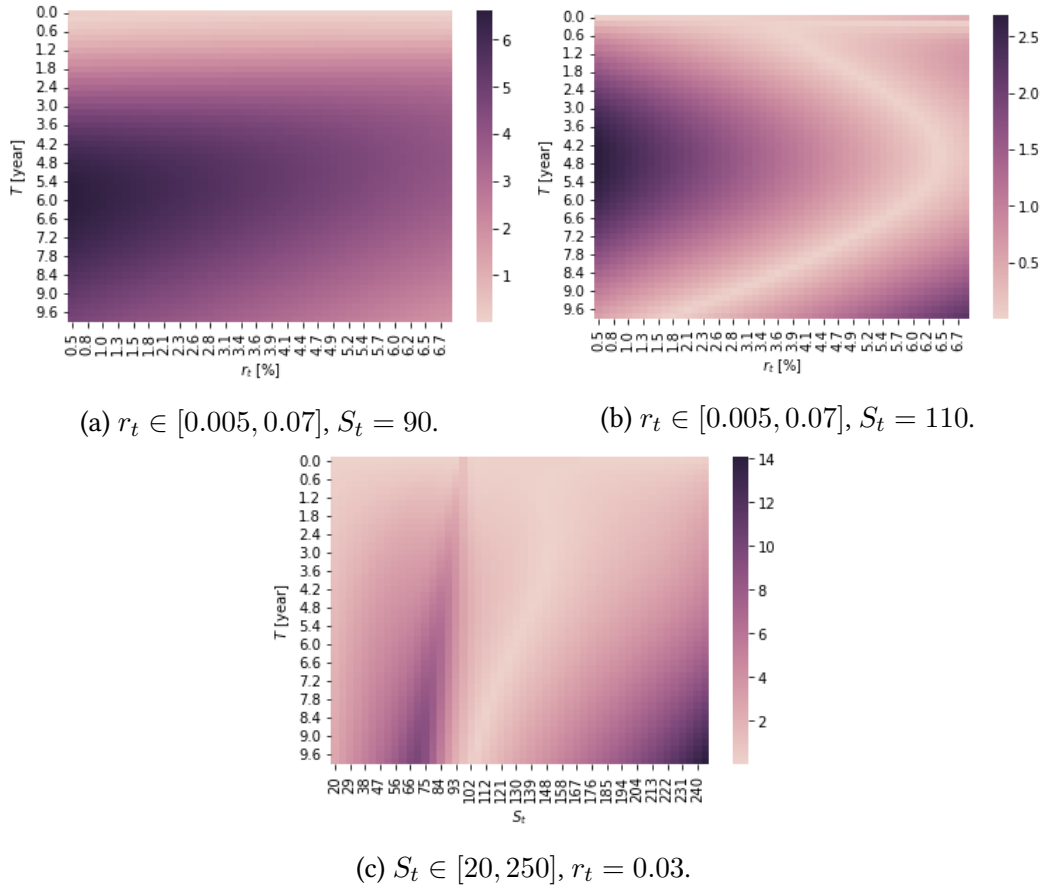


Figure 3.3: Heatmaps of absolute differences between PINN price and closed-form expression in $t = 0$ for $T \in [0, 10]$ and fixed market parameters.

The Figures [3.3a] and [3.3b] confirms that the error in the out-of-the-money case (upper left plot) is bigger than in the in-the-money case (upper right plot). The absolute difference goes up to 6.5 in the first case, whereas it stays below 3 in the second. For $S_0 < K$, the error is quite homogeneous, and we observe a decrease for short maturity. For $S_0 > K$, the biggest error is observed for extreme values of the interest rate. The Figure [3.3c] confirms that for not too high maturity, the PINN price converges to the exact price for $S_0 > K$. However, we observe that the largest errors occur at the highest stock price values. Excluding these extreme values, the error is relatively uniform across different maturities. For a given maturity, the greatest price deviations are found in regions where the price curve exhibits the highest convexity.

Now, we consider various market parameters. We produce similar heatmaps than Figure [3.3] but with random value of parameters (between low and high values given in (2.10)) instead of fixed value of parameters. The Figure [3.4] shows heatmaps of absolute differences between PINN price and closed-form expression for maturities between 0 and 10. The left plot is for several values of the interest rate, $r_t \in [0.005, 0.07]$ and, the right plot is for several values of the stock price, $S_t \in [20, 250]$.

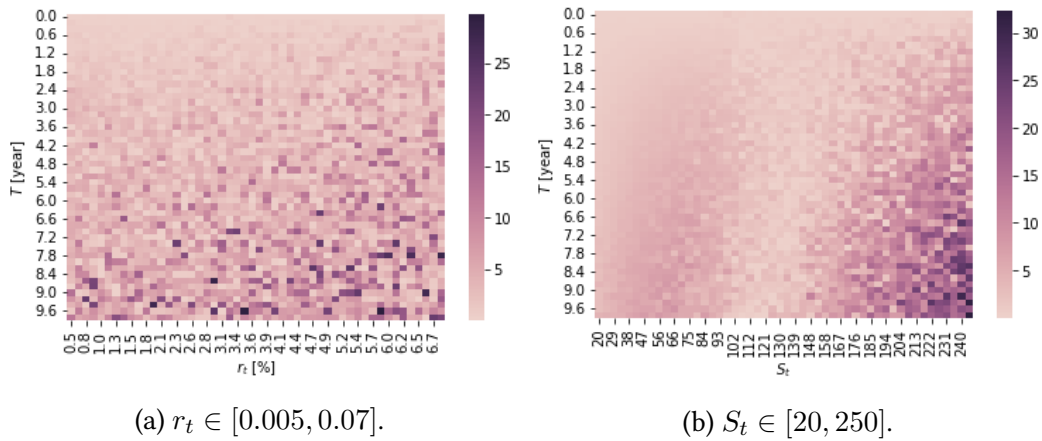


Figure 3.4: Heatmaps of absolute differences between PINN price and closed-form expression in $t = 0$ for $T \in [0, 10]$ and random market parameters.

We observe that the maximum spread between PINN price and closed-form expression increases compared to the case with fixed market parameters. The pricing error is still reasonable most of the time and is under control. Figure [3.4a] illustrates that the difference between PINN and exact prices grows with maturity. However, due to the significant variability in errors between consecutive interest rate values for the same maturity, this increase might also be influenced by other factors, such as the low asset values associated with each interest rate corresponding to an out-of-the-money case. In Figure [3.4b], we generally observe consistent and low pricing errors. However, larger discrepancies appear for higher stock prices and longer maturities. We can therefore conclude that the PINN pricing error is reasonable in most scenarios, though it exhibits greater deviation near inception.

3.4.4 Sensitivity of the 3-256-DGM network to loss function

As discussed above in the selection of the loss function weights, the lower losses (when S_t falls to 0) are very small, see Table [3.1]. This means that the models integrate well the lower boundary condition: call option prices equal zero when $S_t = 0$. It would then be interesting to remove the lower boundary condition and see if it improves the other metrics. We test this for the 3-256-DGM network. Table [3.8] gives the values of the metrics for the two cases. Releasing the lower boundary condition allows a reduction in the inner and maturity losses (about -20%) without too much increase of the MSE and relative error ($+12.7\%$, $+5.1\%$ and $+4.6\%$ for MSE_{alea} , MSE_{conf} and Err_{rel} , respectively).

Lower bound condition	Inner loss $\mathcal{L}_{\mathcal{D}}$	Maturity loss $\mathcal{L}_T$	Lower loss $\mathcal{L}_{low}$	MSE_{alea}	MSE_{conf}	Err_{rel}
with	0.007297	0.004211	0.000959	15.103012	14.321458	0.030593
without	0.005637	0.003403	-	17.014233	15.052166	0.032015

Table 3.8: Performance metrics of the 3-256-DGM network with and without lower boundary condition.

Now we consider adding the loss term (3.2) that gives the error between the PINN price and the closed-form price, and do not take lower loss into account. We take $(w_{\mathcal{D}}, w_T, w_{exact}) = (1, 1, 1)$ as loss function weights. We consider $n_{exact} = 400$ sampling points as a first case and try also with 200 sampling points. These points are sampling uniformly between low and high values, given in (2.10), of parameters and state variables. Table [3.9] gives the values of the metrics for the three cases. Incorporating this condition relative to exact prices significantly increases the inner and maturity loss components. However, it greatly improves the other metrics, indicating that the model becomes much more accurate in predicting option prices. We see that even 200 sampling points improve well the prediction performance. With exact boundary condition, the PINN price behaviour is the same as described in Section 3.4.3 but with much lower spreads between PINN price and closed-form expression. Furthermore, as illustration, the Figure [3.5] shows that the PINN price replicates more accurately the price at inception than in Figure [3.2a]. We have uniformly sampled these points across the domain, though alternative sampling methods could have been used. For instance, we might have focused on adding more exact prices in regions where the PINN price shows the largest errors. However, we will not explore this approach here, as uniform sampling alone significantly enhances the model's accuracy. Additionally, our primary goal is to assess the model's performance without excessive adjustments.

Exact prices condition	Inner loss $\mathcal{L}_{\mathcal{D}}$	Maturity loss $\mathcal{L}_T$	Exact loss $\mathcal{L}_{exact}$	MSE_{alea}	MSE_{conf}	Err_{rel}
without	0.005637	0.003403	-	17.014233	15.052166	0.032015
with 200 points	0.128961	0.035281	0.031648	1.896905	4.245562	0.015228
with 400 points	0.181742	0.041377	0.08358	1.473128	6.775856	0.013046

Table 3.9: Performance metrics of the 3-256-DGM network with and without exact prices condition (no lower bound condition).

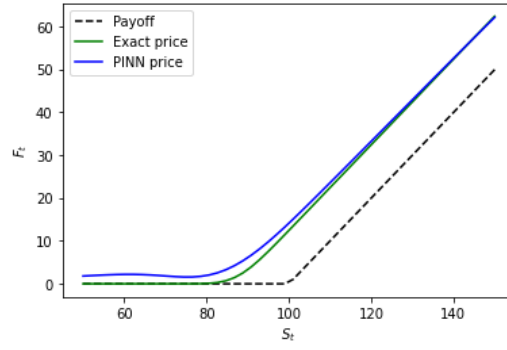


Figure 3.5: European call priced with PINN and closed-form expression for $S_t \in [50, 150]$, $T = 5$, $r_t = 0.03$ and with 400 sampling points for exact prices condition.

We can conclude that, unlike the lower boundary condition, which has a marginal impact, incorporating the exact price condition significantly enhances the model's performance, even with relatively fewer training points compared to those used for inner and maturity losses. For more complex cases like Bermudian options, it would be advisable to focus on the exact price loss without considering the lower loss. However, accurately assessing exact prices necessitates a robust validation procedure capable of providing these exact values. In the European case, we have a closed-form solution, but this is not always available, making the acquisition of exact prices more challenging, as we will discuss in the next chapter for the Bermudian case.

3.4.5 Sensitivity of the 3-256-DGM network to sampling method of training points

Let's reconsider the 3-256-DGM network with the lower boundary condition (loss weights, $(w_D, w_T, w_{low}) = (10, 10, 1)$). Recall that the baseline case assumes uniform sampling of the training points (see Section 2.4.2). As tested by Al-Aradi et al. (2018), we try to use a log-normal distribution for sampling the stock price, S_t . The goal is to have representative stock prices that include the strike price $K = 100$. We consider two distributions: $S_t \sim \log \mathcal{N}(4.70, 0.07)$ with quite small variance and $S_t \sim \log \mathcal{N}(4.66, 0.3)$ with bigger variance. Figure [3.6] shows a representation of each sampling method.

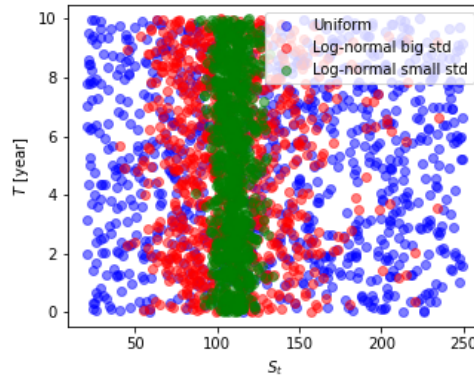


Figure 3.6: Different sampling scheme of the stock price: Uniform in blue, Log-normal with large variance in red and Log-normal with small variance in green.

The Table [3.10] gives the values of the metrics for the three sampling methods. We observe that log-normal sampling with a small variance significantly reduces the loss components. However, it also substantially increases other metrics, indicating signs of overfitting. This is expected, as all S_t values are sampled from a narrow range, causing the network to overfit this limited set of similar data. Additionally, the network is trained on a narrow range of S_t , while $MSEs$ and Err_{rel} are calculated over a broader range, forcing the network to extrapolate the price, which can result in inaccuracies. For the log-normal sampling with a bigger variance, we achieve even smaller loss components, and both $MSEs$ and Err_{rel} are lower than in the small variance case. However, there are still signs of overfitting, as these metrics remain significantly higher compared to the uniform sampling case. Moreover, we can add, thanks to the plots in Figure [3.7], that the log-normal sampling methods fail completely to reproduce the dynamics of the option price even for points close to the strike price, where there are many sampling points.

Sampling of S_t	Inner loss $\mathcal{L}_{\mathcal{D}}$	Maturity loss $\mathcal{L}_T$	Exact loss $\mathcal{L}_{exact}$	MSE_{alea}	MSE_{conf}	Err_{rel}
$\mathcal{U}(20, 250)$	0.007297	0.004211	0.000959	15.103012	14.321458	0.030593
$\log \mathcal{N}(4.7, 0.07)$	0.000302	0.000008	0.000026	8036.6330	8248.3380	1.005010
$\log \mathcal{N}(4.7, 0.3)$	0.000015	0.000001	0.000009	5001.6855	4140.357	0.8784

Table 3.10: Performance metrics of the 3-256-DGM network with uniform and log-normal sampling method for S_t .

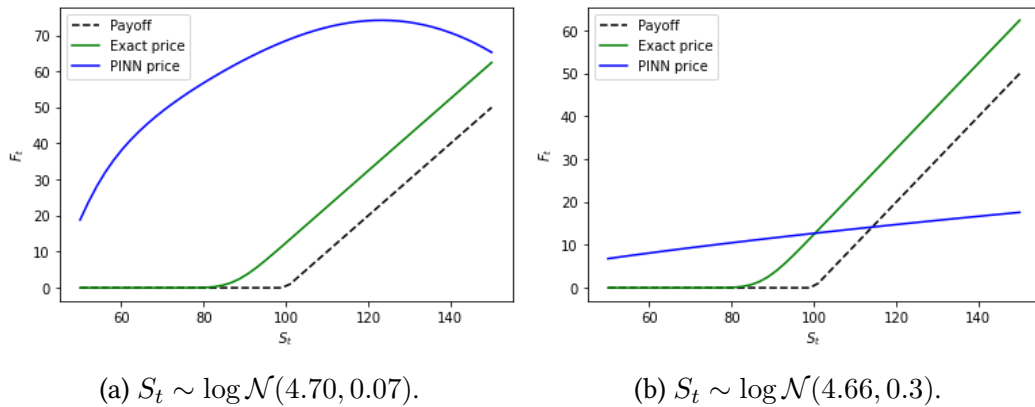


Figure 3.7: European call priced with PINN and closed-form expression for $S_t \in [50, 150]$, $T = 5$, $r_t = 0.03$ and with log-normal sampling methods for the stock price.

In conclusion, we confirm the findings of Al-Aradi et al. (2018) that the sampling method significantly influences the results, with uniform sampling proving to be the most effective approach for the stock price.

3.4.6 Sensitivity of the 3-256-DGM network to training parameters

We examine how performance metrics of the 3-256-DGM network are affected by variations in the training parameters. We conduct three tests:

- Test 1: Increase the sample size $n_{\mathcal{D}}$ from 20,000 to 40,000 items and n_T, n_{low} from 5,000 to 10,000;
- Test 2: Increase the total number of epochs from 3,500 to 5,500 at a smaller learning rate of 0.00005 during the last 2,000 iterations;
- Test 3: A combination of both increasing the sample size and the number of epochs (tests 1 and 2).

The Table [3.11] reports the values of the metrics and the calibration time for the base case and the three tests. We observe that in Test 1, doubling the number of training points leads to a slight decrease in both the loss components and other metrics (MSE and Err_{rel}), with the training time also doubling. In Test 2, adding 2,000 epochs with a lower learning rate results in a more significant reduction in loss components, a slight increase in the other metrics, and an 50% increase in training time. When combining both approaches in Test 3, the loss components decrease (though slightly less than in Test 2), while the other metrics remain consistent with the base case. However, the training time triples compared to the base case.

Case	Inner loss $\mathcal{L}_{\mathcal{D}}$	Maturity loss $\mathcal{L}_T$	Exact loss $\mathcal{L}_{exact}$	MSE_{alea}	MSE_{conf}	Err_{rel}	Training time
Base	0.007297	0.004211	0.000959	15.103012	14.321458	0.030593	3h11
Test 1	0.007144	0.003384	0.000818	14.501055	14.234378	0.030500	6h30
Test 2	0.004391	0.002285	0.000477	16.280869	14.852486	0.031370	4h40
Test 3	0.004182	0.003292	0.000516	15.057356	14.62269	0.030694	9h59

Table 3.11: Performance metrics and training time of the 3-256-DGM network for various training parameters.

In conclusion, for the 3-256-DGM network, adding more epochs increases the risk of overfitting, and doubling the number of training points does not significantly enhance model performance. The optimal accuracy for this network is achieved with the training parameters used in the base case.

Chapter 4

Bermudan put option pricing with a single interest rate curve

Let's consider the case of a Bermudan derivative expiring at time T and with payoff, $H(S_T)$, function of stock price and interest rate at maturity. We are interested in a put option because a Bermudian call is equivalent to a European call, the case developed in Chapter 3. In the case of a put, the payoff with strike K is

$$H(S_T) = (K - S_T)^+.$$

The Bermudan derivative can also be exercised at a series of exercise dates $I = \{t_1, t_2, \dots, t_n = T\}$ where the payoff is $H(S_{t_i})$, $i = 1, 2, \dots, n$. The price in $t = 0$ of such a derivative is given by the expected discounted cash-flows under the risk neutral measure $\mathbb{Q}$:

$$P(0) = \max_{\tau \in I} \mathbb{E}^{\mathbb{Q}} \left[e^{-\int_0^{\tau} r_s ds} (K - S_{\tau})^+ \right],$$

where τ is a stopping time that is part of I but is unknown. The fair value of the derivative, denoted $F(\cdot)$, follows a recursive formula:

$$F(t, I, R_t, S_t) = \begin{cases} H(S_T), & \text{for } t = T; \\ \mathbb{E}^{\mathbb{Q}} \left[e^{-\int_t^{t_{i+1}} r_s ds} F(t_{i+1}, I, r_{t_{i+1}}, S_{t_{i+1}}) \middle| \mathcal{F}_t \right], & \text{for } t_i < t < t_{i+1}, \\ & i = 0, \dots, n-1, \\ & t_0 = 0; \\ \max \left\{ H(S_{t_i}), \mathbb{E}^{\mathbb{Q}} \left[e^{-\int_{t_i}^{t_{i+1}} r_s ds} F(t_{i+1}, I, r_{t_{i+1}}, S_{t_{i+1}}) \middle| \mathcal{F}_{t_i} \right] \right\}, & \text{for } t = t_i \in I, \\ & i = 1, \dots, n-1; \\ \mathbb{E}^{\mathbb{Q}} \left[e^{-\int_0^{t_1} r_s ds} F(t_1, I, r_{t_1}, S_{t_1}) \middle| \mathcal{F}_0 \right], & \text{for } t = 0. \end{cases} \quad (4.1)$$

This means that the holder will exercise its derivative at time t_i if the payoff $H(t_i)$ is bigger than the expected discounted fair value of time t_{i+1} .

It may not seem like it, but this problem is already more complicated than the European case and, for simplicity's sake, we're going to set the number of years to maturity and not use it as the input to the neural network. Since we need to use a neural network between each exercise date and in order to limit the calibration time, we consider only a few years, $T = 5$ for example, and annual exercise dates. In this case, the vector of exercise dates is $I = \{1, 2, 3, 4, 5\}$ and we need 5 neural networks to estimate the derivative price.

So in the Bermudan case, each network takes as input a vector of dimension 7, denoted $\mathbf{z}$, containing scaled time and state variables and parameters:

$$\mathbf{z} := \left(\tilde{t}, \tilde{\mathbf{y}}_t, \kappa, \sigma_r, \sigma_S, \rho \right)^\top. \quad (4.2)$$

Thanks to the Chapter 2, we know that the derivative value respects the scaled FK Equation (2.7). Moreover, Bermudan derivatives satisfy some additional boundary constraints developed here after in Section 4.1. These constraints allow obtaining a loss function that has to be minimized by the neural network in order to well estimate the price of the derivative, see Section 4.2. To ensure that the estimation by the neural network is accurate, we develop a validation procedure in Section 4.3. Finally, we perform numerical analysis with different type and architecture of neural networks in Section 4.4 and perform sensitivity testing with respect to training parameters on the best network selected.

4.1 Boundary constraints

Recall that the derivative price is noted $V(t, \mathbf{y}_t | \kappa, \sigma_r, \sigma_S, \rho)$. This function has to respect multiple boundary constraints which depends on the payoff:

- a maturity boundary condition:

$$V(T, \mathbf{y}_T | \kappa, \sigma_r, \sigma_S, \rho) = F(T, I, r_T, S_T) = H(S_T) = (K - S_T)^+.$$

- $n - 1$ exercise boundary conditions:

$$V(t_i, \mathbf{y}_{t_i} | \kappa, \sigma_r, \sigma_S, \rho) = F(t_i, I, r_{t_i}, S_{t_i}) = \max \left\{ H(S_{t_i}), F^d(t_i) \right\},$$

where $F^d(t_i) := \mathbb{E}^\mathbb{Q} \left[e^{-\int_{t_i}^{t_{i+1}} r_s ds} F(t_{i+1}, I, r_{t_{i+1}}, S_{t_{i+1}}) \middle| \mathcal{F}_{t_i} \right]$ is the expected discounted fair value, for $i = 1, \dots, n - 1$.

We do not consider lower boundary constraint here because we have seen in the European case that it has only a marginal impact. We also have seen that adding a condition on the exact prices can improve a lot the performance of the model. So we consider exact prices conditions on $V(\cdot)$:

$$V(t, \mathbf{y}_t | \kappa, \sigma_r, \sigma_S, \rho) = P^{exact}(t, \mathbf{y}_t, \kappa, \sigma_r, \sigma_S, \rho, I),$$

where $P^{exact}(\cdot)$ represents the exact prices of the derivative, calculated using the same pricing tool as the validation procedure (see Section 4.3). However, as we will discuss in this section, implementing an effective validation procedure for the Bermudan case described here is challenging.

Since we use a scaled version of the FK equation, the above constraints must be adapted accordingly. Recall that the value of the contract can also be seen as a function of scaled time and state variables, $V(\tilde{t}, \tilde{\mathbf{y}}_t | \kappa, \sigma_r, \sigma_S, \rho)$. The rescaled version of the terminal and exercise boundary conditions and of the exact prices conditions are, respectively:

$$V(\tilde{T}, \tilde{\mathbf{y}}_T | \kappa, \sigma_r, \sigma_S, \rho) = H \left(\frac{\tilde{S}_T - a_S}{b_S} \right) = \left(K - \frac{\tilde{S}_T - a_S}{b_S} \right)^+,$$

$$\begin{aligned}
 V(\tilde{t}_i, \tilde{\mathbf{y}}_i | \kappa, \sigma_r, \sigma_S, \rho) &= \max \left\{ H \left(\frac{\tilde{S}_{t_i} - a_S}{b_S} \right), \tilde{F}^d(\tilde{t}_i) \right\}, \\
 V(\tilde{t}, \tilde{\mathbf{y}}_t | \kappa, \sigma_r, \sigma_S, \rho) &= P^{exact}(t, \mathbf{y}_t, \kappa, \sigma_r, \sigma_S, \rho, I), \\
 \text{where } \tilde{F}^d(\tilde{t}_i) &:= \mathbb{E}^{\mathbb{Q}} \left[e^{-\int_{(\tilde{t}_i - a_h)/b_h}^{(\tilde{t}_{i+1} - a_h)/b_h} \frac{\tilde{r}_s - a_r}{b_r} ds} F \left(\frac{\tilde{t}_{i+1} - a_h}{b_h}, \frac{\tilde{I} - a_h}{b_h}, \frac{\tilde{r}_{t_{i+1}} - a_r}{b_r}, \frac{\tilde{S}_{t_{i+1}} - a_S}{b_S} \right) \middle| \mathcal{F}_{\tilde{t}_i} \right].
 \end{aligned}$$

4.2 Loss function

We will determine the loss function to be minimized by the neural networks. In the Bermudan case with maturity T , we approximate the value function solving the scaled FK equation by T neural networks, one for each year. The set of neural networks is denoted $\mathcal{N} = \{N_1, N_2, \dots, N_n\}$ where n is such that $t_n = T$. The neural network N_i covers a period from t_{i-1} to t_i (for $i = 1, 2, \dots, n$ with $t_0 = 0$) which corresponds to one year. Each of these neural networks takes as input the vector $\mathbf{z}$ of dimension 7 defined in (4.2). The neural network $N_i(\cdot)$ approximates the fair value of the Bermudan derivative, i.e. $N_i(\mathbf{z}) \approx V(t, \tilde{\mathbf{y}}_t | \kappa, \sigma_r, \sigma_S, \rho)$ for $t \in (t_{i-1}, t_i]$ for $i = 2, 3, \dots, n$ and for $t \in [0, t_1]$ for $i = 1$. Here, we consider the case where we fix $T = 5$ and $I = \{1, 2, 3, 4, 5\}$, so we have 5 neural networks to calibrate.

Similar to the recursive scheme for determining the fair value of the derivative, neural network estimation is initially performed at maturity T , where the value is determined by the payoff $H(S_T)$. We then move backward in time, using the N_n network to estimate the value from T to t_{n-1} . Next, the N_{n-1} network begins with a payoff that depends on the value provided by the N_n network at t_{n-1} and estimates the value from t_{n-1} to t_{n-2} . This process continues until $t_0 = 0$, where the N_1 network is used, allowing for the estimation of the price at $t = 0$ by approximating it for $t \in [0, t_1]$. Therefore, calibration of the n neural networks must be done in reverse order, starting at maturity (N_n) and ending at inception (N_1).

Regardless of the chosen architecture, the model is trained by minimizing a loss function for each neural network. This loss function is proportional to errors of approximation and comprises multiple terms that represent the desired approximations of the network's output. For the Bermudan case, each network must satisfy three types of conditions: the scaled FK equation, the exercise boundary condition (condition in t_i for the network N_i , previously called maturity boundary condition for $i = n$), and the exact prices conditions. Consequently, the loss function includes three terms. The first two conditions are each calculated on distinct sample sets of training points, as described in Section 2.4.2. For the exact prices conditions, we compute the error between a set of n_{exact} exact prices and compare them to the output of the network. The determination of these exact prices is explained in Section 4.3, where another pricing tool for the Bermudan put option is proposed.

Recall that a neural network is characterized by a set of weights, denoted Θ^i for network N_i . At points of $\tilde{\mathcal{S}}_{\mathcal{D}}^i$, corresponding to the set defined in Equation (2.11) where the time is restrained between t_{i-1} and t_i and T is fixed to 5, we define for $j = 1, \dots, n_{\mathcal{D}}$, the error in scaled FK Equation (2.7) when V is replaced by the neural network as follows:

$$e_j^{\mathcal{D},i}(\Theta^i) = b_h \partial_{t_j} N_i - \frac{\tilde{r}_j - a_r}{b_r} N_i + \mu_{\tilde{\mathbf{y}}}(t_j, \tilde{\mathbf{y}}_j)^\top \nabla_{\tilde{\mathbf{y}}} N_i + \frac{1}{2} \text{tr} \left(\Sigma_{\tilde{\mathbf{y}}}(t_j, \tilde{\mathbf{y}}_j) \Sigma_{\tilde{\mathbf{y}}}(t_j, \tilde{\mathbf{y}}_j)^\top \mathcal{H}_{\tilde{\mathbf{y}}}(N_i) \right).$$

We refer to $e_j^{\mathcal{D},i}$ as the error on the interior domain since it measures the goodness of fit before expiry, between the exercise dates. The average quadratic loss on $\tilde{\mathcal{S}}_{\mathcal{D}}^i$ is the first component of

the total loss function used to fit the neural network N_i ,

$$\mathcal{L}_{\mathcal{D}}^i(\Theta^i) = \frac{1}{n_{\mathcal{D}}} \sum_{j=1}^{n_{\mathcal{D}}} e_j^{\mathcal{D},i}(\Theta^i)^2.$$

We define the exercise boundary sample set, denoted by $\tilde{\mathcal{S}}_I^i$, as the set of $n_I = n_{bound}$ realizations of state variables and parameters. This set corresponds to the set defined in Equation (2.12) where T is not sampled and is fixed to t_i ($\tilde{T} = a_h + b_h t_i$) for network N_i . On this set $\tilde{\mathcal{S}}_I^i$, we define the error $e_k^{I,i}(\Theta^i)$ for $k = 1, \dots, n_I$, as the difference between the output of the neural network N_i and the value of the derivative at the exercise date t_i :

$$e_k^{I,i}(\Theta^i) = \begin{cases} N_i(\tilde{t}_i, \tilde{\mathbf{y}}_k, \kappa_k, \sigma_{rk}, \sigma_{Sk}, \rho_k) - H\left(\frac{S_{T-as}}{b_S}\right), & \text{for } i = n, t_i = T; \\ N_i(\tilde{t}_i, \tilde{\mathbf{y}}_k, \kappa_k, \sigma_{rk}, \sigma_{Sk}, \rho_k) - \max\left\{H\left(\frac{S_{t_i-as}}{b_S}\right), N_{i+1}(\tilde{t}_i, \tilde{\mathbf{y}}_k, \kappa_k, \sigma_{rk}, \sigma_{Sk}, \rho_k)\right\}, & \text{for } i = 0, \dots, n-1. \end{cases}$$

where $N_{i+1}(\tilde{t}_i, \tilde{\mathbf{y}}_k, \kappa_k, \sigma_{rk}, \sigma_{Sk}, \rho_k)$ is the output of the neural network N_{i+1} (approximating V between t_i and t_{i+1}) at t_i . The average quadratic loss at exercise date is the second component of the total loss function:

$$\mathcal{L}_I^i(\Theta^i) = \frac{1}{n_{t_i}} \sum_{k=1}^{n_{t_i}} e_k^{I,i}(\Theta^i)^2.$$

Finally, for the exact prices, we construct a sample set, denoted $\tilde{\mathcal{S}}_{exact}$, with $n_{exact} = 400$ new realizations of state variables and parameters, which are normalised with the same scaling parameters $\mathbf{a}$ and $\mathbf{b}$ obtained in Section 2.4.2:

$$\tilde{\mathcal{S}}_{exact} = \left(\tilde{t}_m, \tilde{\mathbf{y}}_m, \kappa_m, \sigma_{rm}, \sigma_{Sm}, \rho_m \right)_{m=1, \dots, n_{exact}}.$$

Then, we define the error $e_m^{exact,i}(\Theta^i)$ for $m = 1, \dots, n_{exact}$, as the difference between the output of the neural network and the corresponding exact price:

$$e_m^{exact,i}(\Theta^i) = N_i(\tilde{t}, \tilde{\mathbf{y}}_t, \kappa, \sigma_r, \sigma_S, \rho) - P^{exact}(t_m, \mathbf{y}_m, \kappa_m, \sigma_{rm}, \sigma_{Sm}, \rho_m, I).$$

The average quadratic loss for this condition is the third component of the total loss function:

$$\mathcal{L}_{exact}^i(\Theta^i) = \frac{1}{n_{exact}} \sum_{m=1}^{n_{exact}} e_m^{exact,i}(\Theta^i)^2.$$

The optimal network weights are found by minimizing all those losses,

$$\Theta_{opt}^i = \arg \min_{\Theta^i} [\mathcal{L}_{\mathcal{D}}^i(\Theta^i) + \mathcal{L}_I^i(\Theta^i) + \mathcal{L}_{exact}^i(\Theta^i)].$$

In practice, we assign weights to each component of the loss function to explore different configurations, prioritizing some components. The minimization problem becomes,

$$\Theta_{opt}^i = \arg \min_{\Theta^i} [w_{\mathcal{D}} \mathcal{L}_{\mathcal{D}}^i(\Theta^i) + w_I \mathcal{L}_I^i(\Theta^i) + w_{exact} \mathcal{L}_{exact}^i(\Theta^i)].$$

As already mentioned, the exercise boundary error of N_i depends on the output of the neural network N_{i+1} for $i = 1, 2, \dots, n-1$. Thus, we must start by calibrating the neural network N_n then N_{n-1} until N_1 . This final neural network allows estimating the price of the derivative at $t = 0$ since it approximates it for $t \in [0, t_1]$.

4.3 Validation procedure

As in the European case, a validation procedure is of utmost importance in order to ensure the accuracy and reliability of the model developed for approximating the price of a derivative, here a Bermudan put option. Unfortunately, there is no closed-form expression in the Bermudan case, and we must rely on another pricing technique that will also estimate the price of the derivative. As discussed in the state-of-the-art review, other numerical methods include Monte Carlo simulations or bi- and trinomial trees. Although the Least Squares Monte Carlo (LSMC) technique is commonly used for American option valuation and could certainly be applied to Bermudian options, it can be prone to instabilities, which is why we choose not to use it. Additionally, any chosen technique would require re-estimating the price for each combination of initial risk factors and parameters. This would be impractical for our purposes, where we aim to validate the PINN price across a wide range of parameters and state variables. With this in mind, we opt for a tree-based method, which we will develop in the following of this section.

The first step is to construct a trinomial tree for the dynamics of the interest rate r_t . The trinomial tree is an effective technique for pricing exotic options, bonds, and American or Bermudan options within interest rate models. Recall that its dynamic is governed by the Hull & White model. Under the risk-neutral measure $\mathbb{Q}$, the short-term rate follows the stochastic differential equation (SDE):

$$dr_t = \kappa(\gamma(t) - r_t)dt + \sigma_r dW_t^{(1)},$$

where $\gamma(t)$ is a time-dependent deterministic function. We prefer to reformulate this dynamic to isolate the random component of r_t and get rid of the time-dependent drift as follows:

$$r_t := x_t + \nu(t),$$

where $(x_t)_{t \geq 0}$ is a Gaussian process and $\nu(t)$ is a deterministic function of time. The dynamic of x_t is given by:

$$dx_t = -\kappa x_t dt + \sigma_r dW_t^{(1)},$$

which corresponds to a mean-reverting process to level zero. We can show, similarly to the case of r_t in the proof of Proposition A.1, that the solution to this SDE is:

$$x_t = x_s e^{-\kappa(t-s)} + \sigma_r \int_s^t e^{-\kappa(t-u)} dW_u^{(1)},$$

and, as a result, $\nu(t) = f(0, t) + \frac{\sigma_r^2}{2\kappa^2}(1 - e^{-\kappa t})^2$. To build the tree, we first construct it for x_t , and then adjust each node by adding $\nu(t)$ to obtain the corresponding r_t .

We consider a finite set of times, $J = \{t_0, t_1, t_2, \dots, t_n\}$, with $0 = t_0 < t_1 < \dots < t_n = T$ and with time increments $\Delta t_i = t_{i+1} - t_i$. Be aware that this set of times does not correspond to the set of exercise times, I , defined previously in this chapter. However, the times of the set I must be part of the set defined for the construction of the trinomial tree ($I \in J$). At each time t_i , the process x_t can take values at equidistant points $x_{i,j} = j\Delta x_i$, where j indexes the nodes. The corresponding interest rates at each node are $r_{i,j} = x_{i,j} + \nu(t_i)$. At time t_i , if the process is at node j with value $x_{i,j}$, it can transition to one of three possible states at t_{i+1} : $x_{i+1,k+1}$, $x_{i+1,k}$, or $x_{i+1,k-1}$. The probabilities of these transitions are p_u^r , p_m^r , and p_d^r , respectively.

The trinomial tree is calibrated using the method of moments, meaning that we try to match the theoretical mean and variance of the process, which are given by:

$$\begin{aligned}\mathbb{E}^{\mathbb{Q}}(x_t | \mathcal{F}_s) &= x_s e^{-\kappa(t-s)}, \\ \mathbb{V}^{\mathbb{Q}}(x_t | \mathcal{F}_s) &= \int_s^t \sigma_r^2 e^{-2\kappa(t-u)} du = \frac{\sigma_r^2}{2\kappa} (1 - e^{-2\kappa(t-s)}),\end{aligned}$$

which are defined as follows with the lattice components:

$$\begin{aligned}M_{i,j} &:= \mathbb{E}(x_{t_{i+1}} | x_{t_i} = x_{i,j}) = x_{i,j} e^{-\kappa \Delta t_i}, \\ V_i^2 &:= \mathbb{V}(x_{t_{i+1}} | x_{t_i} = x_{i,j}) = \frac{\sigma_r^2}{2\kappa} (1 - e^{-2\kappa \Delta t_i}).\end{aligned}$$

The corresponding moments of the tree are:

$$\begin{aligned}p_u^r x_{i+1,k+1} + p_m^r x_{i+1,k} + p_d^r x_{i+1,k-1} &= M_{i,j}, \\ p_u^r x_{i+1,k+1}^2 + p_m^r x_{i+1,k}^2 + p_d^r x_{i+1,k-1}^2 &= V_i^2 + M_{i,j}^2.\end{aligned}$$

In practice, the step size is chosen as $\Delta x_{i+1} = V_i \sqrt{3}$, and the level k is selected such that $x_{i+1,k}$ is the closest to $M_{i,j}$:

$$k = \text{round} \left(\frac{M_{i,j}}{\Delta x_{i+1}} \right).$$

Let $\eta_{j,k} = M_{i,j} - x_{i+1,k}$. The transition probabilities are then computed as follows:

$$\begin{aligned}p_u^r &= \frac{1}{6} + \frac{\eta_{j,k}^2}{6V_i^2} + \frac{\eta_{j,k}}{2\sqrt{3}V_i}, \\ p_m^r &= \frac{2}{3} - \frac{\eta_{j,k}^2}{3V_i^2}, \\ p_d^r &= \frac{1}{6} + \frac{\eta_{j,k}^2}{6V_i^2} - \frac{\eta_{j,k}}{2\sqrt{3}V_i}.\end{aligned}$$

From these definitions, we can construct the tree for x_t recursively from $t = 0$ to $t = T$. Then we evaluate $\nu(t)$ for each node and obtain the tree for r_t by adding it to x_t . Technically speaking, at the end we obtain an array of size $(m+1) \times (n+1)$ where m is the maximum number of possible states at a given time, n being the number of time steps. Each entry in this table corresponds to a node where we have the following data: the value of r_t , the transition probabilities (p_d^r, p_m^r, p_u^r) and the indices of the nodes to which the transition is made (k_d^r, k_m^r, k_u^r).

The second step is to construct a binomial tree for the dynamics of the stock price S_t that relies on the trinomial tree of the interest rate. We consider the same set of times, J , and time increments, Δt_i , defined for the trinomial tree of r_t . At each time t_i , if the stock price process is at node j with value $S_{i,j}$, it can move up ($S_{i+1,k+1} = u_{i,j} S_{i,j}$) or down ($S_{i+1,k+1} = d_{i,j} S_{i,j}$). The corresponding transition probabilities are p_u^S and $p_d^S = 1 - p_u^S$ and are given by:

$$\begin{aligned}p_u^S &= \frac{e^{r_{i,j} \Delta t_i} - d_{i,j}}{u_{i,j} - d_{i,j}}, \\ \text{where} \quad u_{i,j} &= \exp \left(\left(r_{i,j} - \frac{\sigma_S^2}{2} \right) \Delta t_i + \sigma_S \sqrt{\Delta t_i} \right), \\ d_{i,j} &= \exp \left(\left(r_{i,j} - \frac{\sigma_S^2}{2} \right) \Delta t_i - \sigma_S \sqrt{\Delta t_i} \right).\end{aligned}$$

To ensure that the evolution of S_t is coherent with the evolution of r_t , we are basing our development on the work of De Simone (2013). In fact, we consider a hybrid tree such that at each node the interest rate can move up, down or stay in the middle and the stock price can move up or down, resulting in a combination of six possibilities. The transition probabilities are then p_{uu} , p_{ud} , p_{mu} , p_{md} , p_{du} and p_{dd} . If the two processes were independent, it would be sufficient to multiply the marginal transition probabilities of each process. However, the two processes are correlated (with a correlation of ρ). Therefore, we start with the probabilities from the independent case and adjust them to account for the correlation. If the correlation is negative, we reduce the probabilities of transitions where S and r move in the same direction (i.e., both up or both down). Conversely, if the correlation is positive, we reduce the probabilities of transitions where S and r move in opposite directions (i.e., one up and the other down). Since the transition probabilities must sum to 1, the penalized amount is redistributed to the other probabilities. The transition probabilities for the hybrid tree are thus given by:

if $\rho < 0$:

$$\begin{aligned} p_{uu} &= p_u^r p_u^S (1 + \rho), \\ p_{ud} &= p_u^r p_d^S - \frac{\rho}{4} (p_u^r p_u^S + p_d^r p_d^S), \\ p_{mu} &= p_m^r p_u^S - \frac{\rho}{4} (p_u^r p_u^S + p_d^r p_d^S), \\ p_{md} &= p_m^r p_d^S - \frac{\rho}{4} (p_u^r p_u^S + p_d^r p_d^S), \\ p_{du} &= p_d^r p_u^S - \frac{\rho}{4} (p_u^r p_u^S + p_d^r p_d^S), \\ p_{dd} &= p_d^r p_d^S (1 + \rho), \end{aligned}$$

if $\rho \geq 0$:

$$\begin{aligned} p_{uu} &= p_u^r p_u^S + \frac{\rho}{4} (p_d^r p_u^S + p_u^r p_d^S), \\ p_{ud} &= p_u^r p_d^S (1 - \rho), \\ p_{mu} &= p_m^r p_u^S + \frac{\rho}{4} (p_d^r p_u^S + p_u^r p_d^S), \\ p_{md} &= p_m^r p_d^S + \frac{\rho}{4} (p_d^r p_u^S + p_u^r p_d^S), \\ p_{du} &= p_d^r p_u^S (1 - \rho), \\ p_{dd} &= p_d^r p_d^S + \frac{\rho}{4} (p_d^r p_u^S + p_u^r p_d^S). \end{aligned}$$

Technically, we end up with an array of size $(6^n) \times (n + 1)$, where n is the number of time steps. In the i^{th} column of this table, the first 6^i entries represent the nodes at time t_i . For each node, the following data is stored: the values of r_t and r_t , the transition probabilities (p_{uu} , p_{ud} , p_{mu} , p_{md} , p_{du} , p_{dd}), the indices of the nodes to which the transition is made (k_{uu} , k_{ud} , k_{mu} , k_{md} , k_{du} , k_{dd}), and the index of the mother node (l). The mother node refers to the node at the previous time step (in the previous column) from which the current node is computed.

Finally, now that the hybrid tree has been constructed, we can evaluate Bermudian options. We start with the terminal nodes, at $t = t_n = T$, where the option is equal to the payoff:

$$F_{n,j} = (S_{n,j} - K)^+.$$

We then go backwards through the tree and evaluate the option as follows, for $i = n - 1, \dots, 0$:

$$F_{i,j} = \begin{cases} \max[(S_{i,j} - K)^+; f_{i,j}], & \text{if } t_i \in I, \\ f_{i,j}, & \text{otherwise,} \end{cases}$$

where $f_{i,j} = e^{-r_{i,j} \Delta t_i} (p_{uu} F_{i+1,k_{uu}} + p_{ud} F_{i+1,k_{ud}} + p_{mu} F_{i+1,k_{mu}} + p_{md} F_{i+1,k_{md}} + p_{du} F_{i+1,k_{du}} + p_{dd} F_{i+1,k_{dd}}).$

The price in $t = 0$ of the derivative is then given by $F_{0,0}$. This recursive formulation translates the fact that a Bermudan derivative can be exercised at a series of exercise dates, I . Then, the

exact prices used for the exact prices condition or for validation of the PINN is given as follows:

$$P^{exact}(t, r_t, S_t, \kappa, \sigma_r, \sigma_S, \rho, I) = F_{i,j}|_{\kappa, \sigma_r, \sigma_S, \rho, I},$$

where i and j are such that $t_i \approx t$, $S_{i,j} \approx S_t$ and $r_{i,j} \approx r_t$.

This approach should theoretically offer an accurate method for pricing Bermudan derivatives with stochastic interest rates, provided the number of time steps between inception and maturity approaches infinity. However, from a computational perspective, the array used for the hybrid tree contains many empty entries (only 6^i non-empty entries in column i , while all columns are sized at 6^n elements). To address this, we implemented a sparse version of the code. Despite this optimization, achieving the desired accuracy requires a very large number of time steps, which in turn necessitates declaring arrays of immense size (e.g., $60M \times 11$ for $n = 10$). Consequently, regardless of the implementation, we face significant memory issues (even with sparse implementation), making the validation procedure unusable.

One potential solution is to attempt constructing a recombining hybrid tree, where the states at time t_i can be reached from multiple states at time t_{i-1} . This concept is illustrated with binomial and trinomial trees in Figure 4.1. However, in our case, each node has six possible transitions, unlike the two or three found in binomial or trinomial trees. This solution would decrease significantly the maximum number of states to $5n + 1$ (e.g., arrays of size 51×11 for $n = 10$). If the recombining approach proves ineffective, we could explore using dynamic programming. This would allow us to compute and store only the essential parts of the tree as needed, rather than keeping the entire tree in memory.

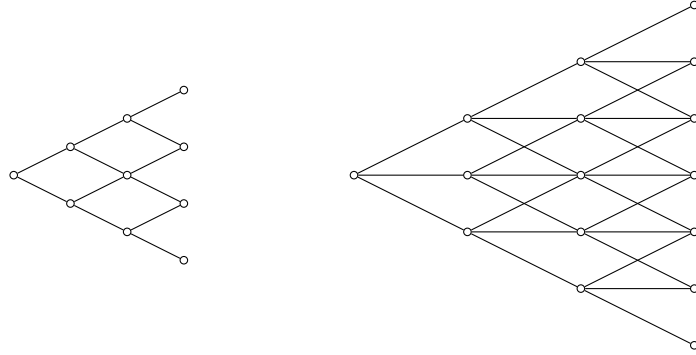


Figure 4.1: Example of recombining trees, binomial tree (left) and trinomial tree (right).

Finally, we could consider a completely different numerical validation method, such as Monte Carlo simulations. There also exist techniques that use directly prices found on the market to validate their model. For example, Siswanah et al. (2023) explains that Bermudan options are traded bilaterally in the over-the-counter market, where the seller and the buyer based their price on the option price listed on the stock exchange. So they use data obtained from Yahoo Finance in their research. This approach could enable the collection of sufficient data to create the set of sampling points needed for the exact price condition (recall that we consider $n_{exact} = 400$ sampling points). However, gathering additional data to validate the model over an extended set of points might pose a challenge, as a total of 15,000 observations are required to calculate the metrics MSE_{alea} , MSE_{conf} , and Err_{rel} .

4.4 Numerical analysis

As discussed in the previous section, we were unable to establish an effective validation procedure, which leads to two key consequences. First, we cannot generate exact prices for the exact prices loss component, meaning the total loss function is now limited to the inner loss and exercise loss components, with $\mathcal{L}_{exact} = 0$ set to zero. Note that the code developed works with the condition on the exact prices, if they were available (see comments in the code).

The second consequence is that we are unable to compare the PINN price with another pricing method to evaluate the model's accuracy. We can only rely on the components of the loss function, $\mathcal{L}_{\mathcal{D}}$ and $\mathcal{L}_I$. With only these metrics, we cannot detect potential overfitting in the model. Therefore, our goal is to minimize the loss components, but not to the extent that it risks overfitting. Note that the metrics provided represent the sum of the metrics across all networks.

For this numerical analysis, we examine a Bermudan put option with a maturity of $T = 5$ years and annual exercise dates. This setup requires the calibration of five neural networks, making the process five times longer than in the European case for an equivalent network. As with the European case, we use a trial-and-error approach to identify the best model. Initially, we apply the standard calibration procedure and find the optimal weights for the loss function components, $w_{\mathcal{D}}$ and w_I , see Equation (4.2). After determining these weights, we choose the best model by evaluating the ResNN and DGM architectures described in Section 2.3.3 and tuning the network's hyperparameters (number of layers and neurons per layer). Once the model is selected, we conduct more in-depth analyses to understand how it predicts the derivative price. Finally, still using the selected model, we assess the impact of changes in the calibration procedure by testing different learning rates and training points. We do not explore modifications to the loss function, as adding the lower loss showed minimal benefit in the European call case, a finding that aligns with the conclusions drawn for the Heston model by Hainaut and Casas (2024). Similarly, we do not consider altering the sampling method for the stock price, as this approach has proven ineffective in the European case.

4.4.1 Loss function weights selection

We will explore whether varying the emphasis on different loss components could be beneficial. Given that the inner loss is a key component, we experiment with increasing its relative importance compared to the exercise loss. We evaluate these configurations using a general network, the 3-256-ResNN, which does not require excessive calibration time. Table [4.1] presents the values of the loss components for the three different scenarios we analysed.

Type	# layers	# neurons per layer	Loss weights ($w_{\mathcal{D}}, w_I$)	Inner loss $\mathcal{L}_{\mathcal{D}}$	Exercise loss $\mathcal{L}_I$
ResNN	3	256	(1,1)	0.010556	0.014485
ResNN	3	256	(10,1)	0.003447	0.043398
ResNN	3	256	(100,1)	0.001850	0.239017

Table 4.1: Components of the loss for different combinations of loss weights.

As expected, placing greater emphasis on the inner loss component leads to a larger

reduction in this area compared to the exercise loss component. However, the decrease in the inner loss doesn't offset the increase in the exercise loss, resulting in a higher overall loss (when ignoring the weights) as w_D increases. Additionally, while the inner loss is crucial, the exercise loss is equally important because this boundary condition is an intrinsic part of the derivative's pricing. Therefore, we conclude that it is best to maintain equal weights, $(w_D, w_I) = (1, 1)$, for the rest of this analysis.

4.4.2 Network architecture and hyperparameters selection

Firstly, we determined in the European case that the DGM architecture is the most effective for capturing complex dependencies. However, its lengthy calibration time suggests that, in certain contexts, the faster-calibrating ResNN architecture might be more advantageous. Given that we need to calibrate five neural networks here, the reduction in training time could be appealing. Consequently, we evaluate both the DGM and ResNN architectures, varying the number of layers from 2 to 4 and the number of neurons per layer from 64 to 256. Table [4.2] presents the loss values for the nine different hyperparameter combinations for the two architectures considered.

Type	# layers	# neurons per layer	Inner loss $\mathcal{L}_D$	Exercise loss $\mathcal{L}_I$	Training time
ResNN	2	64	0.052697	0.063397	0h52
ResNN	2	128	0.031047	0.041455	1h30
ResNN	2	256	0.022836	0.033683	3h25
ResNN	3	64	0.036783	0.040280	1h17
ResNN	3	128	0.019120	0.023839	2h00
ResNN	3	256	0.010556	0.014485	4h18
ResNN	4	64	0.030290	0.035099	1h24
ResNN	4	128	0.012800	0.014538	2h49
ResNN	4	256	0.007133	0.009059	4h52
DGM	2	64	0.077686	0.094731	2h02
DGM	2	128	0.050522	0.062704	3h33
DGM	2	256	0.031745	0.039429	7h30
DGM	3	64	0.035023	0.034268	4h08
DGM	3	128	0.015209	0.016803	7h55
DGM	3	256	0.010939	0.014629	15h18
DGM	4	64	0.024998	0.028913	5h20
DGM	4	128	0.013583	0.015025	9h49
DGM	4	256	0.006573	0.009383	22h18

Table 4.2: Components of the loss and training time for different networks.

As anticipated, the more complex the network – those with a higher number of layers and neurons per layer (resulting in more parameters) – the lower the loss components tend to be. This trend was already observed in the European case. Notably, for a given set of hyperparameters, the ResNN and DGM architectures yield very similar losses. In some instances,

ResNN even achieves lower losses than DGM, despite DGM requiring significantly more time for calibration. Consequently, we have decided not to continue with the DGM architecture.

When focusing on the ResNN architecture, the 4-256-ResNN network achieves the best results in terms of loss minimization. However, without additional metrics, we can't confirm whether this network is overfitting the training data. To be cautious, we opt for the second-best network in terms of loss minimization, which also has fewer parameters: the 3-256-ResNN network. Interestingly, this network configuration aligns with the hyperparameters of the network chosen in the European case, though it uses a different architecture. The next section gives a more detailed analysis of the results of the 3-256-ResNN network.

4.4.3 Performances of the 3-256-ResNN network

Since we are not able to show the absolute difference between PINN price and the price given by another pricing method, we only show the value of the PINN price in various context. First, we consider that the market is fixed and have a unique set of market parameters, shown in Table [4.3].

κ	1.15	σ_r	0.01
σ_S	0.025	ρ	-0.4

Table 4.3: Fixed market parameters.

The Figure [4.2] compares PINN prices of the Bermudan put option for several values of the stock price, S_t , with fixed interest rate, $r_t = 0.03$. These plots show the price given by all the calibrated network, at start date of each network, N_i , for the upper left plot ($t = I_{i-1}$) and at end date of each network for the lower plot ($t = I_i$). The upper right plot shows the prices at an intermediate time.

We observe that at the option's maturity, the price provided by the network N_5 at $t = 5$, shown in Figure [4.2c], closely aligns with the payoff but shows the largest error around $S_t = K$, where the price curve's convexity is at its peak. More broadly, this plot illustrates the PINN prices at the points where the exercise boundary condition is applied for each network. Recall that this condition compels the prices to be the maximum between the current fair value and the payoff: $F_t = \max(H(S_t), F_t^d) = H(S_t) + (F_t^d - H(S_t))^+$, for networks $N_1, \dots, N_4$, using simplified notations. As a result, the price is automatically above the payoff. When $S_t > K$, the payoff is zero, and the option is not exercised, so the price equals the discounted fair value. Conversely, when $S_t < K$, there is a high probability that the option will be exercised, so the price should approach the payoff. The prices generated by the PINN at exercise dates appear to adhere to these principles. In Figures [4.2a] and [4.2b], we see that the price gradually diverges from the payoff, which is not an unexpected behaviour.

Another way to visualize the results and obtain PINN prices for more intermediate time points is through a heatmap. Figure [4.3] presents heatmaps of PINN prices for times between 0 and 5. The left plot displays data for various interest rate values, $r_t \in [0.005, 0.07]$, with $S_t = 90$, representing an in-the-money scenario. The right plot shows data for different stock prices, $S_t \in [20, 250]$, with $r_t = 0.03$.

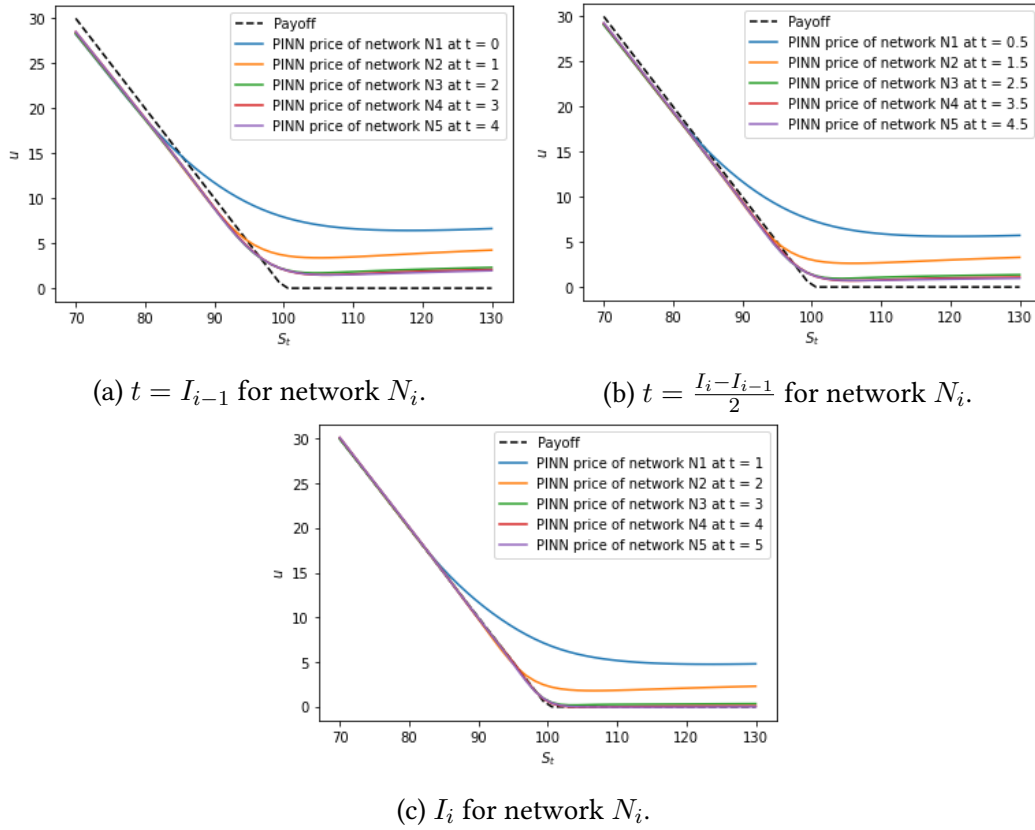


Figure 4.2: Bermudan put priced with PINN for $S_t \in [70, 130]$, $r_t = 0.03$.

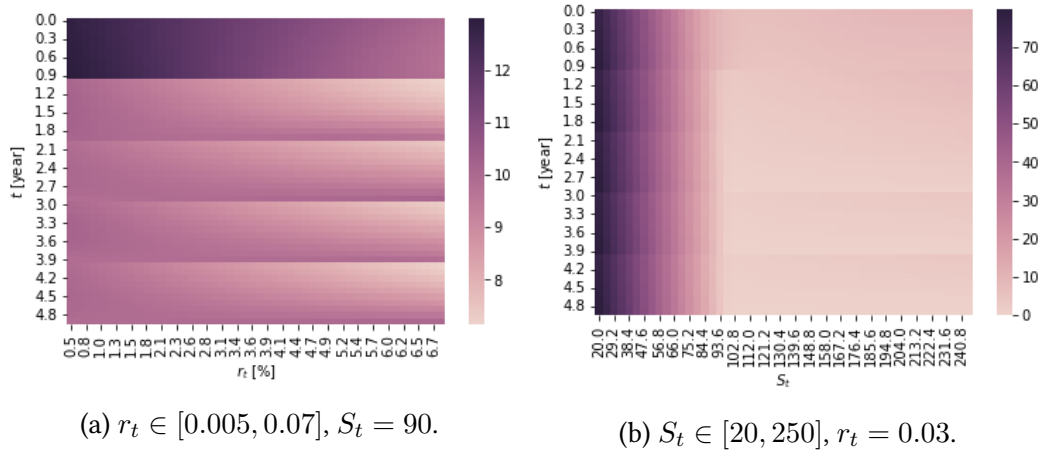


Figure 4.3: PINN price for $t \in [0, 5]$ and fixed market parameters.

We observe a value jump at each exercise date, as anticipated, due to the exercise boundary conditions. Figure [4.3a] demonstrates that the PINN price remains relatively stable concerning the interest rate. Additionally, Figure [4.3b] shows that the PINN price consistently follows the same trend as payoff.

We do not present the results for random market parameters, as they do not provide any additional insights compared to the scenario where the parameters are fixed in a specific configuration.

4.4.4 Sensitivity of the 3-256-ResNN network to training parameters

We examine how the 3-256-ResNN network is impacted by a change in the training parameters. We conduct three tests that are similar as for the European case:

- Test 1: Increase the sample size n_D from 20,000 to 40,000 items and n_T from 5,000 to 10,000;
- Test 2: Increase the total number of epochs from 3,500 to 4,500 at a learning rate of 0.0001 during the last 1,000 iterations;
- Test 3: A combination of both increasing the sample size and the number of epochs (tests 1 and 2).

The Table [4.4] reports the value of the loss components and the calibration time for the base case and the tree tests. We observe that increasing the number of training points (Test 1) slightly raises the loss components, while adding more epochs (Test 2) reduces them. Combining both approaches (Test 3) leads to a smaller reduction in losses compared to Test 2 alone.

Case	Inner loss $\mathcal{L}_D$	Exercise loss $\mathcal{L}_I$	Training time
Base	0.010556	0.014485	4h18
Test 1	0.011555	0.015475	8h58
Test 2	0.008215	0.011146	5h45
Test 3	0.009529	0.012677	11h36

Table 4.4: Loss components and training time of the 3-256-ResNN network for various training parameters.

Since increasing the number of training points should improve model predictions, while adding epochs alone may lead to overfitting, Test 3 appears to be the best option for improving performance. However, Test 3 significantly increases calibration time without guaranteeing substantial improvements. Therefore, we choose to maintain the calibration parameters from the base case for the 3-256-ResNN network, as this approach has proven effective for optimizing this type of network in the European case.

Chapter 5

Extending the model: Option pricing with multiple interest rate curves

In the previous chapters, we developed an option pricing framework using a single interest rate curve for both European call and Bermudan put options. This approach, based on a physics-informed neural network, allows for price estimation across a wide range of market parameters (with the model calibrated for $\kappa \in [0.5, 2]$, $\sigma_r \in [0.002, 0.02]$, $\sigma_S \in [0.02, 0.3]$, and $\rho \in [-0.8, 0.8]$). However, it does not account for different interest rate curve scenarios, as the Nelson-Siegel parameters are fixed to the values listed in Table [2.1]. Incorporating multiple interest rate curve contexts would be particularly valuable in risk management and asset-liability management (ALM).

Let's extend our model to consider a broader scenario where multiple interest rate curves are calibrated. This extension involves integrating the Nelson-Siegel parameters as additional inputs into our physics-informed neural network (PINN). By doing so, we can account for a range of possible interest rate environments, thereby enhancing the model's flexibility. This approach allows us to simulate and evaluate options price under various economic conditions, providing a more comprehensive tool for practitioners.

We start by revisiting the frameworks for European call and Bermudan put options established in Chapters 3 and 4, and adapt them to incorporate the new range of Nelson-Siegel parameters in Section 5.1. Next, in Section 5.2, we examine how this adaptation affects the performance of the optimal network for European call options, as identified in Section 3.4. Lastly, in Section 5.3, we evaluate the impact on the PINN pricing derived from the optimal network for Bermudan put options, as discussed in Section 4.4.

5.1 Model adaptations

The primary adjustment to the model involves incorporating the Nelson-Siegel parameters (b_0 , b_{10} , b_{11} , and c_1) as additional inputs to the model. We first examine the impact on the scaled Feynman-Kac Equation (2.7). The Nelson-Siegel parameters are embedded in the expression of $\gamma(t)$, see Equation (2.4), but they do not impact the partial derivatives of the FK equation. The only modification is in the derivative value expression, which now depends on scaled time and state variables, but with additional parameters:

$$F_t = V(\tilde{t}, \tilde{\mathbf{y}}_t \mid \kappa, \sigma_r, \sigma_S, \rho, b_0, b_{10}, b_{11}, c_1, \tilde{T}).$$

Consequently, the neural network now takes a vector of dimension 12, instead of 8:

$$\mathbf{z} := \left(\tilde{t}, \tilde{\mathbf{y}}_t, \kappa, \sigma_r, \sigma_S, \rho, b_0, b_{10}, b_{11}, c_1, \tilde{T} \right)^\top.$$

Thus, the neural network, denoted $N(\cdot)$, approximates the derivative price as:

$$N(\mathbf{z}) \approx V(\tilde{t}, \tilde{\mathbf{y}}_t \mid \kappa, \sigma_r, \sigma_S, \rho, b_0, b_{10}, b_{11}, c_1, \tilde{T}).$$

With the Nelson-Siegel parameters now incorporated into the model, we must define their ranges. As detailed in Section 2.4.1, there is no systematic procedure for selecting these ranges. We start with the calibrated Nelson-Siegel parameters from Table [2.1] and set the low and high values to $\pm 40\%$ of these values. This results in the following ranges:

$$\begin{cases} b_0 & \in [0.023849, 0.055648], \\ b_{10} & \in [0.000883, 0.002061], \\ b_{11} & \in [-0.010162, -0.023711], \\ c_1 & \in [0.253374, 0.591206]. \end{cases}$$

Next, we adjust the training points. As for the other parameters of the neural network, we sample uniformly values of NS parameters between their low and high values. This approach is consistent across all sampling sets. The loss function remains unchanged, the Nelson-Siegel parameters are only used in calculating $\gamma(t)$ for the inner loss, and $\gamma(t)$ can be computed for various parameter values.

Finally, the metrics used in numerical analysis (MSE_{alea} , MSE_{conf} , and Err_{rel}) remain globally unaffected. We just need to make sure that the exact price can be calculated using various values for the NS parameters. And the NS parameters values used for the computation of MSE_{conf} are fixed to the following values: $b_0 = 0.04$, $b_{10} = 0.002$, $b_{11} = -0.018$ and $c_1 = 0.45$.

5.2 European call option case

To analyse the impact of adding new parameters into the model in the European call option case, we test the adapted model given in the above section with the same network selected in the numerical analysis of Section 3.4, the 3-256-DGM network.

The Table [5.1] gives the value of the metrics and the training time for the base case and the case where the NS parameters are added as inputs to the network. We observe an increase in the loss components in the extended case compared to the base case. The inner loss nearly quadruples, and the losses at maturity double. The lower loss is less impacted, rising by about 50%. Other metrics also show a slight increase, as does calibration time.

Case	Inner loss $\mathcal{L}_{\mathcal{D}}$	Maturity loss $\mathcal{L}_T$	Exact loss $\mathcal{L}_{exact}$	MSE_{alea}	MSE_{conf}	Err_{rel}	Training time
Base	0.007297	0.004211	0.000959	15.103012	14.321458	0.030593	3h11
NS	0.028630	0.008226	0.001465	16.617712	16.319382	0.035448	3h46

Table 5.1: Performance metrics and training time of the 3-256-DGM network. NS case being the case where the Nelson-Siegel parameters have been added as inputs to the network.

Incorporating additional parameters into the neural network's input reduces the model's accuracy for given hyperparameters and network's architecture. Therefore, the adapted model requires a new numerical analysis to identify the new optimal configuration. As the number of model parameters increases, the network's complexity can also be increased slightly without a significant risk of overfitting. Note that a more complex network would take longer to train.

5.3 Bermudan put option case

To analyse the impact of adding new parameters into the model in the Bermudan put option case, we test the adapted model given in Section 5.1 with the same network selected in the numerical analysis of Section 4.4, the 3-256-ResNN network.

The Table [5.2] gives the value of the loss components and the training time for the base case and the case where the NS parameters are added as inputs to the network. Similar to the European case, the loss components have increased, though with a smaller relative impact (inner loss rises by 89%, and exercise loss by less than 1%). The training time is also only merely affected.

Case	Inner loss $\mathcal{L}_{\mathcal{D}}$	Exercise loss $\mathcal{L}_I$	Training time
Base	0.010556	0.014485	4h18
NS	0.019972	0.014565	4h47

Table 5.2: Loss components and training time of the 3-256-ResNN network. NS case being the case where the Nelson-Siegel parameters have been added as inputs to the network.

Even without a validation procedure in the Bermudan case, we can reasonably assume that the conclusions drawn from adding new parameters to the neural network are similar to those in the European case. Introducing these additional parameters should be paired with a fresh numerical analysis to identify the optimal network, which will likely need to be more complex than the 3-256-ResNN network.

5.4 Conclusion

We have successfully extended our option pricing model to incorporate multiple interest rate curves by integrating the Nelson-Siegel parameters as additional inputs into the physics-informed neural network (PINN). This enhancement allows the model to account for a broader range of economic scenarios, increasing its flexibility in pricing both European call and Bermudan put options.

The results indicate that while the inclusion of these additional parameters enhances the model's adaptability, it does not significantly affect the performance of the previously selected optimal model. However, given the increased input dimensionality, we recommend doing a new model selection. A slightly more complex network might better handle the expanded input space and improve overall performance.

In summary, this extension marks a meaningful advancement in creating a more comprehensive and adaptable option pricing framework.

Chapter 6

Perspectives for life insurance applications

Now that we have a model for pricing European call options and Bermudan put options under a stochastic interest rate, we can explore life insurance applications, focusing on the Bermudan case since the European call option is a well-established and widely used case with a known closed-form solution.

A first application is the valuation of embedded options such as Guaranteed Minimum Withdrawal Benefits (GMWB) which is a variable annuity where minimum periodical withdrawals are guaranteed. The option embedded here is the policyholder's ability to withdraw funds even if the value of the investment fund decreases. When disregarding the insured's survival probabilities (treating the product purely as an investment product) the value of a GMWB can be modelled similarly to a Bermudan put option. In this framework, the policyholder has multiple predetermined opportunities to make withdrawals, reflecting the early exercise feature of Bermudan options. The option's value is determined through a recursive relationship akin to Equation (4.1), where the payoff depends on the difference between the guaranteed withdrawal amount, c , and the value of the investment fund, A_t . For example, the payoff can be represented as $H(S_t) = (c - A_t)^+$, with the investment fund $(A_t)_{t \geq 0}$ invested in the asset $(S_t)_{t \geq 0}$ and evolving according to the relation $A_t = (A_{t-1} - c) \frac{S_t}{S_{t-1}}$.

Another application is modelling policy surrenders in life insurance. Many life insurance policies include embedded options, one of which is the surrender option, allowing the policyholder to terminate the contract in exchange for a surrender value. This option can be analogized to Bermudan put options, where the policyholder has multiple opportunities (typically on policy anniversaries) to "exercise" the option by surrendering the policy. The policyholder is likely to surrender the contract if the surrender value exceeds the value of the contract, taking into account a margin that reflects their preference to wait until the surrender value surpasses the contract value by a certain percentage before taking action. In this context, the value of the surrender option can be determined using a backward induction process similar to the one in Equation (4.1). The payoff is defined as the surrender value multiplied by an indicator function that translates whether the policyholder surrenders: $\text{Payoff} = P \mathbf{1}_{\{(1-\beta)P > V_t\}} = S \frac{((1-\beta)P - V_t)^+}{(1-\beta)P - V_t}$, where P represents the surrender value, β is a pre-determined margin (e.g., $(1 - \beta) = 90\%$), and V_t is the contract value at time t , calculated as $V_t = V_{t-1} \frac{S_t}{S_{t-1}}$ in the case of an investment product linked to the stock $(S_t)_{t \geq 0}$.

This tool can have broader applications that can improve life insurers' practices in various areas. It can aid in risk-neutral valuation, compliance with Solvency II and IFRS 17, stress testing, and product development by accurately pricing guarantees and embedded options, ensuring competitiveness while managing risk effectively.

Chapter 7

Conclusion

The aim of this thesis was to use PINNs to price options in unexplored configurations. We developed a model based on the Deep Galerkin Method (DGM) with three layers of 256 neurons each to price European call options. We then obtained a model for evaluating a Bermudian put. This model is a neural network with skip connections, also featuring three layers of 256 neurons each. Both models account for stochastic interest rates using the Hull & White model and a stock following geometric Brownian motion. On this financial market, we were able to develop a scaled version of the Feynman-Kac equation, which was used in neural networks training. For European options, we validated our results with a closed-form formula, while validation for Bermudian puts faced challenges with the hybrid bi-trinomial tree method. Finally, We extended our models to handle various interest rate scenarios and mentioned the potential benefits of using PINNs for evaluating embedded options and simulating contract surrenders.

Numerical analyses demonstrate that PINNs are a robust parametric tool that can be able of well capturing the price dynamics, at least in the European case, where results were validated. The largest errors occur in out-of-the-money scenarios or where the price curve exhibits high convexity. More refine analysis could be done to determine where these spreads between PINN price and exact price come from, especially out-of-the-money. However, we showed that these deviations can be minimized by including exact price conditions during the network's training phase. We also observed deviations for extreme values of state variables, likely due to boundary effects. One solution would be to train the network for a wider range of state variables than the range considered for pricing. Additionally, adding more parameters to the network's input increased flexibility without sacrificing accuracy. Although, a more tailored model could further improve accuracy by better adapting to the increased complexity of the problem.

PINNs offer several advantages over traditional methods in option pricing. First, it handles easily multiple risk factors while, as seen in the validation procedure of the Bermudan case where a hybrid tri-binomial tree was used, traditional techniques can be difficult to implement and time-consuming. Another key benefit of PINNs is their flexibility as a parametric model. Unlike Monte Carlo simulations or tree-based methods that require recalibration for each new set of market parameters, PINNs can be trained once and then quickly used for pricing across different scenarios. Moreover, once calibrated, the valuation process with PINNs is remarkably fast, taking less than a minute.

However, there are also some challenges associated with using PINNs. One of the main drawbacks is that the optimal calibration procedure often requires trial and error, as there is no standardized approach. Similarly, selecting the best network architecture and hyperparameters is also largely dependent on trial and error, and could benefit from more systematic methods, such as top-down or bottom-up approaches. Furthermore, while calibration time can be lengthy,

this could be mitigated by using more powerful computing resources or by initializing network weights with those from a similar, already-optimized network. Finally, clearly explaining the relationship between a neural network's input and output can be challenging, which may make it difficult to understand certain discrepancies.

Looking ahead, there are several exciting avenues for future work. One possibility is to explore alternative neural network architectures that might offer even better performance. Additionally, the pricing of Bermudan options could be extended to other frameworks, for example by incorporating more risk factors, such as mortality rates or stochastic variance (e.g., the Heston model). The Svensson model for interest rate modelling, which introduces two additional parameters compared to the Nelson-Siegel model, could also be considered. However, this would first require the development of an effective validation procedure for the Bermudan case, which remains an open challenge from this master thesis. Another possible extension of this work is to consider the American option case, where options can be exercised at any time. This would require a complete redesign of the current model, which, as a reminder, uses a neural network between each exercise date in the Bermudan case. Finally, the models developed here could be directly applied to real-world scenarios in life insurance, opening up new possibilities for practical application. The journey of refining PINNs for option pricing is far from over, with each step bringing us closer to more robust and versatile financial tools.

Appendix A

Interest rate modelling: Hull & White model

With the Hull & White model, the short term rate is ruled under the risk neutral measure $\mathbb{Q}$ by the following SDE:

$$dr_t = \kappa(\gamma(t) - r_t)dt + \sigma_r dW_t^{(1)}.$$

Proposition A.1. *The price of a zero coupon bond under the Hull & White model is given by,*

$$P(t, T) = \exp(A(t, T) - r_t B(t, T)),$$

where

$$B(t, T) = \frac{1}{\kappa} \left(1 - e^{-\kappa(T-t)}\right),$$

$$A(t, T) = - \int_t^T \gamma(u) \kappa B(u, T) du + \frac{1}{2} \int_t^T \sigma_r^2 B^2(u, T) du,$$

and $f(0, t)$ is the initial instantaneous forward rate.

Proof: If the market is arbitrage free and complete, there exists a risk neutral measure $\mathbb{Q}$ such that the discounted value of $P(t, T)$ is a martingale under $\mathbb{Q}$,

$$\mathbb{E}^{\mathbb{Q}} \left(\frac{P(T, T)}{B_T} \middle| \mathcal{F}_t \right) = \frac{P(t, T)}{B_t},$$

where $B_t = \exp \left(\int_0^t r_s ds \right)$ is the cash account. The zero coupon bond price is then equal to the following expectation:

$$P(t, T) = \mathbb{E}^{\mathbb{Q}} \left(\frac{B_t}{B_T} \underbrace{P(T, T)}_{=1} \middle| \mathcal{F}_t \right) = \mathbb{E}^{\mathbb{Q}} \left(\exp \left(- \int_t^T r_s ds \right) \middle| \mathcal{F}_t \right).$$

The short term rate $r_s | \mathcal{F}_t$ and its integral can be expressed as follows:

$$r_s = r_t e^{-\kappa(s-t)} + \int_t^s \gamma(u) \kappa e^{-\kappa(s-u)} du + \int_t^s \sigma_r e^{-\kappa(s-u)} dW_u^{(1)},$$

$$\int_t^T r_s ds = r_t B(t, T) + \int_t^T \gamma(u) \kappa B(u, T) du + \int_t^T \sigma_r B(u, T) dW_u^{(1)}. \quad (\text{A.1})$$

and thus have both a normal distribution. Indeed, let us define Z_t as follows:

$$Z_t = e^{\kappa t} (\gamma(t) - r_t). \quad (\text{A.2})$$

By Itô's lemma, the differential of Z_t is equal to

$$\begin{aligned} dZ_t &= \frac{\partial Z_t}{\partial t} dt + \frac{\partial Z_t}{\partial r_t} dr_t + \frac{1}{2} \frac{\partial^2 Z_t}{\partial r_t^2} \sigma_r^2 dt, \\ &= \left[\kappa e^{\kappa t} (\gamma(t) - r_t) + e^{\kappa t} \gamma'(t) \right] dt - e^{\kappa t} dr_t + 0 dt, \\ &= e^{\kappa t} \gamma'(t) dt - e^{\kappa t} \sigma_r dW_t^{(1)}. \end{aligned}$$

where $\gamma'(t) = \frac{d\gamma(t)}{dt}$. Integrating from t to s ,

$$Z_s = Z_t + \int_t^s e^{\kappa u} \gamma'(u) du - \int_t^s e^{\kappa u} \sigma_r dW_u^{(1)}. \quad (\text{A.3})$$

Integrating by parts the second term, we infer that

$$\int_t^s e^{\kappa u} \gamma'(u) du = [e^{\kappa u} \gamma(u)]_t^s - \int_t^s \kappa e^{\kappa u} \gamma(u) du = e^{\kappa s} \gamma(s) - e^{\kappa t} \gamma(t) - \int_t^s \kappa e^{\kappa u} \gamma(u) du.$$

Combining (A.2) and (A.3), we have that

$$\begin{aligned} e^{\kappa s} (\gamma(s) - r_s) &= e^{\kappa t} (\gamma(t) - r_t) + e^{\kappa s} \gamma(s) - e^{\kappa t} \gamma(t) - \int_t^s \kappa e^{\kappa u} \gamma(u) du - \int_t^s e^{\kappa u} \sigma_r dW_u^{(1)}, \\ \Leftrightarrow \quad r_s &= r_t e^{-\kappa(s-t)} + \int_t^s \kappa e^{-\kappa(s-u)} \gamma(u) du + \int_t^s e^{-\kappa(s-u)} \sigma_r dW_u^{(1)}. \end{aligned}$$

And the integral of r_s is obtained as follows:

$$\begin{aligned} \int_t^T r_s ds &= r_t \int_t^T e^{-\kappa(s-t)} ds + \int_t^T \int_t^s \gamma(u) \kappa e^{-\kappa(s-u)} du ds + \int_t^T \int_t^s \sigma_r e^{-\kappa(s-u)} dW_u^{(1)} ds, \\ &= r_t \int_t^T e^{-\kappa(s-t)} ds + \int_t^T \int_u^T \gamma(u) \kappa e^{-\kappa(s-u)} ds du + \int_t^T \int_u^T \sigma_r e^{-\kappa(s-u)} ds dW_u^{(1)}, \\ &= r_t B(t, T) + \int_t^T \gamma(u) \kappa B(u, T) du + \int_t^T \sigma_r B(u, T) dW_u^{(1)}. \end{aligned}$$

which corresponds to a normal distribution with

$$\begin{aligned} \mathbb{E}^{\mathbb{Q}} \left(\int_t^T r_s ds \middle| \mathcal{F}_t \right) &= r_t B(t, T) + \int_t^T \gamma(u) \kappa B(u, T) du, \\ \mathbb{V}^{\mathbb{Q}} \left(\int_t^T r_s ds \middle| \mathcal{F}_t \right) &= \int_t^T \sigma_r^2 B^2(u, T) du. \end{aligned}$$

Let us define $Y = \exp \left(- \int_t^T r_s ds \right)$. Then Y is lognormal, $\ln Y \sim \mathcal{N}(\mu_Y, \sigma_Y)$ with $\mu_Y = -\mathbb{E}^{\mathbb{Q}} \left(\int_t^T r_s ds \middle| \mathcal{F}_t \right)$ and $\sigma_Y^2 = \mathbb{V}^{\mathbb{Q}} \left(\int_t^T r_s ds \middle| \mathcal{F}_t \right)$. The zero coupon bond price is given by the expectation of Y which for a lognormal distribution is given by:

$$\mathbb{E}(Y) = e^{\mu_Y + \frac{\sigma_Y^2}{2}}.$$

□

Proposition A.2. *The function $\gamma(t)$ matches the initial yield curve of interest rate if*

$$\gamma(t) = \frac{1}{\kappa} \frac{\partial f(0, t)}{\partial t} + f(0, t) + \frac{\sigma_r^2}{2\kappa^2} (1 - e^{-2\kappa t}),$$

where $f(0, t)$ is the instantaneous forward rate.

Proof: By Proposition A.1, we have the expression of the price of a zero coupon bond. The initial price of a zero coupon bond of maturity t can also be expressed in terms of instantaneous forward rate:

$$P(0, t) = \exp \left(- \int_0^t f(0, u) du \right).$$

Then, by identification, we have

$$\int_0^t \gamma(u) \kappa B(u, t) du = -r_0 B(0, t) + \int_0^t f(0, u) du + \frac{1}{2} \int_0^t \sigma_r^2 B^2(u, t) du.$$

By using the Leibniz integral rule and deriving twice:

$$\frac{d}{dt} : \quad \int_0^t \gamma(u) \kappa e^{-\kappa(t-u)} du = f(0, t) + \frac{\sigma_r^2}{2\kappa^2} (1 - 2e^{-\kappa t} + e^{-2\kappa t}), \quad (\text{A.4})$$

$$\frac{d^2}{dt^2} : \quad \kappa \gamma(t) - \kappa \int_0^t \gamma(u) \kappa e^{-\kappa(t-u)} du = \frac{\partial}{\partial t} f(0, t) + \frac{\sigma_r^2}{\kappa} (e^{-\kappa t} - e^{-2\kappa t}). \quad (\text{A.5})$$

Putting (A.4) in (A.5), we obtain the following useful reformulation of $\gamma(t)$:

$$\begin{aligned} \gamma(t) &= f(0, t) + \frac{\sigma_r^2}{2\kappa^2} (1 - 2e^{-\kappa t} + e^{-2\kappa t}) + \frac{1}{\kappa} \frac{\partial}{\partial t} f(0, t) + \frac{\sigma_r^2}{\kappa^2} (e^{-\kappa t} - e^{-2\kappa t}), \\ &= f(0, t) + \frac{1}{\kappa} \frac{\partial}{\partial t} f(0, t) + \frac{\sigma_r^2}{2\kappa^2} (1 - e^{-2\kappa t}). \end{aligned}$$

□

Then, knowing the expression of $\gamma(t)$, we can rewrite the price of a zero coupon bond of Proposition A.1 by computing the following integral:

$$\begin{aligned} \int_t^T \gamma(u) \kappa B(u, T) du &= -\ln \frac{P(0, T)}{P(0, t)} - f(0, t) B(t, T) + \frac{\sigma_r^2}{2\kappa^2} (T - t) \\ &\quad - \frac{\sigma_r^2}{2\kappa^2} B(t, T) - \frac{\sigma_r^2}{4\kappa} e^{-2\kappa t} B^2(t, T). \end{aligned}$$

The price of a zero coupon bond is given by:

$$P(t, T) = \exp \left(-r_t B(t, T) + f(0, t) B(t, T) + \ln \frac{P(0, T)}{P(0, t)} - \frac{\sigma_r^2}{4\kappa^3} (1 - e^{-2\kappa t}) B^2(t, T) \right). \quad (\text{A.6})$$

And the following proposition gives the dynamic of the zero coupon bond price.

Proposition A.3. *The dynamic of the price of a zero coupon bond is:*

$$\frac{dP(t, T)}{P(t, T)} = r_t dt - \sigma_r B(t, T) dW_t^{(1)}.$$

Proof: The price depends on t and r_t , so from Itô's lemma:

$$dP(t, T) = \frac{\partial P(t, T)}{\partial t} dt + \frac{\partial P(t, T)}{\partial r_t} \kappa(\gamma(t) - r_t) dt + \frac{\partial P(t, T)}{\partial r_t} \sigma_r dW_t^{(1)} + \frac{1}{2} \frac{\partial^2 P(t, T)}{\partial r_t^2} \sigma_r^2 dt, \quad (\text{A.7})$$

where $\frac{\partial P(t, T)}{\partial r_t} = -P(t, T)B(t, T)$, $\frac{\partial^2 P(t, T)}{\partial r_t^2} = P(t, T)B^2(t, T)$ and,

$$\begin{aligned} \frac{\partial P(t, T)}{\partial t} &= P(t, T) \left[-r_t \frac{\partial B(t, T)}{\partial t} + f(0, t) \frac{\partial B(t, T)}{\partial t} + \frac{\partial f(0, t)}{\partial t} B(t, T) - \frac{\partial}{\partial t} (\ln P(0, t)) \right. \\ &\quad \left. - \frac{\sigma_r^2}{4\kappa^3} \frac{\partial(1 - e^{-2\kappa t})}{\partial t} B^2(t, T) - \frac{\sigma_r^2}{4\kappa^3} (1 - e^{-2\kappa t}) \frac{\partial B^2(t, T)}{\partial t} \right], \\ &= P(t, T) \left[r_t e^{-\kappa(T-t)} + f(0, t)(1 - e^{-\kappa(T-t)}) + \frac{\partial f(0, t)}{\partial t} B(t, T) \right. \\ &\quad \left. - \frac{\sigma_r^2}{2\kappa^2} (e^{-2\kappa t}(1 - e^{-\kappa(T-t)})^2 - (1 - e^{-2\kappa t})(e^{-2\kappa(T-t)} - e^{-\kappa(T-t)})) \right]. \end{aligned}$$

Injecting the partial derivatives and the value of $\gamma(t)$ given in Proposition A.2 in (A.7), we have:

$$\begin{aligned} dP(t, T) &= P(t, T) \left[r_t e^{-\kappa(T-t)} + f(0, t)(1 - e^{-\kappa(T-t)}) + \frac{\partial f(0, t)}{\partial t} B(t, T) \right. \\ &\quad \left. - \frac{\sigma_r^2}{2\kappa^2} (e^{-2\kappa t}(1 - e^{-\kappa(T-t)})^2 - (1 - e^{-2\kappa t})(e^{-2\kappa(T-t)} - e^{-\kappa(T-t)})) \right. \\ &\quad \left. - (1 - e^{-\kappa(T-t)}) \left(\frac{1}{\kappa} \frac{\partial f(0, t)}{\partial t} + f(0, t) + \frac{\sigma_r^2}{2\kappa^2} (1 - e^{-2\kappa t}) \right) \right. \\ &\quad \left. + (1 - e^{-\kappa(T-t)}) r_t + \frac{1}{2} \sigma_r^2 B^2(t, T) \right] dt - \sigma_r B(t, T) P(t, T) dW_t^{(1)}, \\ &= P(t, T) \left[r_t - \frac{\sigma_r^2}{2\kappa^2} (1 - e^{-\kappa(T-t)})(1 - e^{-2\kappa t}) + \frac{\sigma_r^2}{2\kappa^2} (1 - e^{-\kappa(T-t)})^2 \right. \\ &\quad \left. - \frac{\sigma_r^2}{2\kappa^2} (e^{-2\kappa t}(1 - e^{-\kappa(T-t)})^2 - (1 - e^{-2\kappa t})(e^{-2\kappa(T-t)} - e^{-\kappa(T-t)})) \right] dt \\ &\quad - \sigma_r B(t, T) P(t, T) dW_t^{(1)}, \\ &= P(t, T) [r_t dt - \sigma_r B(t, T) dW_t^{(1)}]. \end{aligned}$$

□

Appendix B

Feynman-Kac equation

To obtain the Feynman-Kac Equation (2.6), we apply the Itô's lemma to the derivative price, $V(\cdot)$, and rewrite the absence of arbitrage condition, Equation (2.5).

Itô's Lemma is a fundamental result in stochastic calculus, which plays a critical role in the mathematical modelling of financial derivatives. First introduced in Itô (1941), the lemma provides a way to find the differential of a function that depends on a stochastic process, specifically an Ito process. In the context of derivative pricing, Itô's Lemma is used to derive the dynamics of the price of a derivative, which is a function of the underlying asset's price, which is stochastic, and time. Here, the derivative price depends on three variables: the time and two stochastic state variables (r_t and S_t). The Itô's lemma gives:

$$\begin{aligned} dV &= \frac{\partial V}{\partial t} dt + \frac{\partial V}{\partial r_t} dr_t + \frac{\partial V}{\partial S_t} dS_t + \frac{1}{2} \frac{\partial^2 V}{\partial r_t^2} \sigma_r^2 \Sigma_r^\top \Sigma_r dt + \frac{1}{2} \frac{\partial^2 V}{\partial S_t^2} \sigma_S^2 S_t^2 \Sigma_S^\top \Sigma_S dt \\ &\quad + \frac{1}{2} \frac{\partial^2 V}{\partial r_t \partial S_t} \sigma_r \sigma_S S_t \Sigma_r^\top \Sigma_S dt + \frac{1}{2} \frac{\partial^2 V}{\partial S_t \partial r_t} \sigma_r \sigma_S S_t \Sigma_S^\top \Sigma_r dt, \\ &= \frac{\partial V}{\partial t} dt + \frac{\partial V}{\partial r_t} [\kappa(\gamma(t) - r_t) dt + \sigma_r \Sigma_r^\top d\mathbf{W}_t] + \frac{\partial V}{\partial S_t} [r_t S_t dt + \sigma_S S_t \Sigma_S^\top d\mathbf{W}_t] \\ &\quad + \frac{1}{2} \frac{\partial^2 V}{\partial r_t^2} \sigma_r^2 \Sigma_r^\top \Sigma_r dt + \frac{1}{2} \frac{\partial^2 V}{\partial S_t^2} \sigma_S^2 S_t^2 \Sigma_S^\top \Sigma_S dt \\ &\quad + \frac{1}{2} \frac{\partial^2 V}{\partial r_t \partial S_t} \sigma_r \sigma_S S_t \Sigma_r^\top \Sigma_S dt + \frac{1}{2} \frac{\partial^2 V}{\partial S_t \partial r_t} \sigma_r \sigma_S S_t \Sigma_S^\top \Sigma_r dt, \\ &= \frac{\partial V}{\partial t} dt + \frac{\partial V}{\partial r_t} [\kappa(\gamma(t) - r_t) dt + \sigma_r dW_t^{(1)}] \\ &\quad + \frac{\partial V}{\partial S_t} [r_t S_t dt + \sigma_S S_t \rho dW_t^{(1)} + \sigma_S S_t \sqrt{1 - \rho^2} dW_t^{(2)}] \\ &\quad + \frac{1}{2} \frac{\partial^2 V}{\partial r_t^2} \sigma_r^2 dt + \frac{1}{2} \frac{\partial^2 V}{\partial S_t^2} \sigma_S^2 S_t^2 dt + \frac{1}{2} \frac{\partial^2 V}{\partial r_t \partial S_t} \sigma_r \sigma_S S_t \rho dt + \frac{1}{2} \frac{\partial^2 V}{\partial S_t \partial r_t} \sigma_r \sigma_S S_t \rho dt, \\ &= \left[\frac{\partial V}{\partial t} + \frac{\partial V}{\partial r_t} \kappa(\gamma(t) - r_t) + \frac{\partial V}{\partial S_t} r_t S_t + \frac{1}{2} \frac{\partial^2 V}{\partial r_t^2} \sigma_r^2 + \frac{1}{2} \frac{\partial^2 V}{\partial S_t^2} \sigma_S^2 S_t^2 + \frac{1}{2} \frac{\partial^2 V}{\partial r_t \partial S_t} \sigma_r \sigma_S S_t \rho \right. \\ &\quad \left. + \frac{1}{2} \frac{\partial^2 V}{\partial S_t \partial r_t} \sigma_r \sigma_S S_t \rho \right] dt + \left[\frac{\partial V}{\partial r_t} \sigma_r + \frac{\partial V}{\partial S_t} \sigma_S S_t \rho \right] dW_t^{(1)} + \frac{\partial V}{\partial S_t} \sigma_S S_t \sqrt{1 - \rho^2} dW_t^{(2)}. \end{aligned}$$

The absence of arbitrage condition, saying that all traded securities earn on average the

risk-free rate, can be rewritten as follows:

$$\begin{aligned}
 \mathbb{E}(dV|\mathcal{F}_t) &= V r_t dt, \\
 \Leftrightarrow \quad &\frac{\partial V}{\partial t} + \frac{\partial V}{\partial r_t} \kappa(\gamma(t) - r_t) + \frac{\partial V}{\partial S_t} r_t S_t + \frac{1}{2} \frac{\partial^2 V}{\partial r_t^2} \sigma_r^2 + \frac{1}{2} \frac{\partial^2 V}{\partial S_t^2} \sigma_S^2 S_t^2 + \frac{1}{2} \frac{\partial^2 V}{\partial r_t \partial S_t} \sigma_r \sigma_S S_t \rho \\
 &\quad + \frac{1}{2} \frac{\partial^2 V}{\partial S_t \partial r_t} \sigma_r \sigma_S S_t \rho = V r_t, \\
 \Leftrightarrow \quad &0 = \partial_t V - r_t V + \mu_{\mathbf{y}}(t, \mathbf{y}_t)^\top \nabla_{\mathbf{y}} V + \frac{1}{2} \text{tr} \left(\Sigma_{\mathbf{y}}(t, \mathbf{y}_t) \Sigma_{\mathbf{y}}(t, \mathbf{y}_t)^\top \mathcal{H}_{\mathbf{y}}(V) \right),
 \end{aligned}$$

where,

$$\mu_{\mathbf{y}}(t, \mathbf{y}_t) = \begin{pmatrix} \kappa(\gamma(t) - r_t) \\ r_t S_t \end{pmatrix}, \quad \Sigma_{\mathbf{y}}(t, \mathbf{y}_t) = \begin{pmatrix} \sigma_r \Sigma_r^\top \\ \sigma_S S_t \Sigma_S^\top \end{pmatrix},$$

and

$$\nabla_{\mathbf{y}} V = \begin{pmatrix} \partial_r V \\ \partial_S V \end{pmatrix}, \quad \mathcal{H}_{\mathbf{y}}(V) = \begin{pmatrix} \partial_{rr} V & \partial_{rS} V \\ \partial_{Sr} V & \partial_{SS} V \end{pmatrix}.$$

Appendix C

Closed-form expression for the European call option

This appendix contains the proof of the proposition that the quantity $\frac{S_t}{P(t,T)}$ follows a log-normal distribution.

Proof of Proposition 3.1: We start from the dynamic of $dP(t, T)$ given in Proposition A.3 and, by Itô's lemma, we obtain the dynamic of $d\frac{1}{P(t,T)}$ under the risk-neutral measure $\mathbb{Q}$:

$$\begin{aligned} d\frac{1}{P(t,T)} &= \frac{-1}{P^2(t,T)}dP(t,T) + \frac{1}{2}\frac{2}{P^3(t,T)}\sigma_r^2 B^2(t,T)P^2(t,T)dW_t^{(1)}, \\ &= \frac{-r_t}{P(t,T)}dt + \frac{\sigma_r^2 B^2(t,T)}{P(t,T)}dt + \frac{\sigma_r B(t,T)}{P(t,T)}dW_t^{(1)}. \end{aligned}$$

And under the $\mathbb{F}$ -measure defined in Equation (3.3):

$$\begin{aligned} d\frac{1}{P(t,T)} &= \frac{1}{P(t,T)} \left[-r_t + \sigma_r^2 B^2(t,T) - \sigma_r^2 B^2(t,T) \right] dt + \frac{\sigma_r B(t,T)}{P(t,T)}dW_t^{(1)\mathbb{F}}, \\ &= \frac{-r_t}{P(t,T)}dt + \frac{\sigma_r B(t,T)}{P(t,T)}dW_t^{(1)\mathbb{F}}. \end{aligned}$$

Then we can find the dynamic of $d\left(\frac{S_t}{P(t,T)}\right)$ under $\mathbb{F}$, thanks to Itô's lemma:

$$\begin{aligned} d\left(\frac{S_t}{P(t,T)}\right) &= S_t d\frac{1}{P(t,T)} + \frac{dS_t}{P(t,T)} + \sigma_r \sigma_S \rho S_t \frac{B(t,T)}{P(t,T)}dt, \\ &= \frac{S_t}{P(t,T)} \left[(\sigma_r B(t,T) + \sigma_S \rho) dW_t^{(1)\mathbb{F}} + \sigma_S \sqrt{1 - \rho^2} dW_t^{(2)\mathbb{F}} \right], \\ &= \frac{S_t}{P(t,T)} \left(\sigma_r B(t,T) \Sigma_r^\top + \sigma_S \Sigma_S^\top \right) d\mathbf{W}_t^\mathbb{F}. \end{aligned}$$

Applying Itô's lemma to $\ln(S_t/P(t,T))$, leads to the following result:

$$\begin{aligned} d\ln\left(\frac{S_t}{P(t,T)}\right) &= \frac{P(t,T)}{S_t} d\left(\frac{S_t}{P(t,T)}\right) \\ &\quad - \frac{1}{2} \frac{P^2(t,T)}{S_t^2} \frac{S_t^2}{P^2(t,T)} \left(\sigma_r B(t,T) \Sigma_r^\top + \sigma_S \Sigma_S^\top \right) \left(\sigma_r B(t,T) \Sigma_r + \sigma_S \Sigma_S \right) dt, \\ &= \frac{-1}{2} \left(\sigma_r^2 B^2(t,T) + 2\sigma_r \sigma_S \rho B(t,T) + \sigma_S^2 \right) dt + \left(\sigma_r B(t,T) \Sigma_r^\top + \sigma_S \Sigma_S^\top \right) d\mathbf{W}_t^\mathbb{F}. \end{aligned}$$

This last expression emphasizes that $\ln \frac{S_T/P(T,T)}{S_t/P(t,T)} \sim \mathcal{N}(\mu_{\mathbb{F}}(t), v_{\mathbb{F}}(t))$ with:

$$v_{\mathbb{F}}(t) = \int_t^T \left(\sigma_r B(u, T) \Sigma_r^\top + \sigma_S \Sigma_S^\top \right) \left(\sigma_r B(u, T) \Sigma_r + \sigma_S \Sigma_S \right) du,$$

$$\mu_{\mathbb{F}}(t) = \frac{-1}{2} \int_t^T \left(\sigma_r^2 B^2(u, T) + 2\sigma_r \sigma_S \rho B(u, T) + \sigma_S^2 \right) du = \frac{-1}{2} v_{\mathbb{F}}(t).$$

□

Bibliography

- Ali Al-Aradi, Adolfo Correia, Danilo Naiff, Gabriel Jardim, and Yuri Saporito. Solving non-linear and high-dimensional partial differential equations via deep learning. *arXiv preprint arXiv:1811.08782*, 2018.
- Ali Al-Aradi, Adolfo Correia, Gabriel Jardim, Danilo de Freitas Naiff, and Yuri Saporito. Extensions of the deep galerkin method. *Applied Mathematics and Computation*, 430:127287, 2022.
- Ulrich Anders, Olaf Korn, and Christian Schmitt. Improving the pricing of options: A neural network approach. *Journal of forecasting*, 17(5-6):369–388, 1998.
- Karim Barigou and Łukasz Delong. Pricing equity-linked life insurance contracts with multiple risk factors by neural networks. *Journal of Computational and Applied Mathematics*, 404:113922, 2022.
- Christian Beck, Weinan E, and Arnulf Jentzen. Machine learning approximation algorithms for high-dimensional fully nonlinear partial differential equations and second-order backward stochastic differential equations. *Journal of Nonlinear Science*, 29:1563–1619, 2019.
- Francesca Biagini, Lukas Gonon, and Thomas Reitsam. Neural network approximation for superhedging prices. *Mathematical Finance*, 33(1):146–184, 2023.
- Hans Buehler, Lukas Gonon, Josef Teichmann, and Ben Wood. Deep hedging. *Quantitative Finance*, 19(8):1271–1291, 2019.
- Zi Cai and Jinguo Liu. Approximating quantum many-body wave functions using artificial neural networks. *Physical Review B*, 97(3):035116, 2018.
- Giuseppe Carleo and Matthias Troyer. Solving the quantum many-body problem with artificial neural networks. *Science*, 355(6325):602–606, 2017.
- Salvatore Cuomo, Vincenzo Schiano Di Cola, Fabio Giampaolo, Gianluigi Rozza, Maziar Raissi, and Francesco Piccialli. Scientific machine learning through physics-informed neural networks: Where we are and what’s next. *Journal of Scientific Computing*, 92(3):88, 2022.
- Antonio De Simone. *A Two-Factor Binomial Model for Pricing Hybrid Securities: A Simplified Approach*. PhD thesis, University of Napoli “Federico II”, 2013.
- Ashish Dhiman and Yibei Hu. Physics informed neural network for option pricing. *arXiv preprint arXiv:2312.06711*, 2023.
- Daniel Doyle and Chris Groendyke. Using neural networks to price and hedge variable annuity guarantees. *Risks*, 7(1):1, 2018.
- Ryno Du Plooy and Pierre J Venter. Pricing vanilla options using artificial neural networks: Application to the south african market. *Cogent Economics & Finance*, 9(1):1914285, 2021.

- Raquel M Gaspar, Sara D Lopes, and Bernardo Sequeira. Neural network pricing of american put options. *Risks*, 8(3):73, 2020.
- Federico Gatta, Vincenzo Schiano Di Cola, Fabio Giampaolo, Francesco Piccialli, and Salvatore Cuomo. Meshless methods for american option pricing through physics-informed neural networks. *Engineering Analysis with Boundary Elements*, 151:68–82, 2023.
- Igor Vladimirovich Girsanov. On transforming a certain class of stochastic processes by absolutely continuous substitution of measures. *Theory of Probability & Its Applications*, 5(3): 285–301, 1960.
- Kathrin Glau and Linus Wunderlich. The deep parametric pde method and applications to option pricing. *Applied Mathematics and Computation*, 432:127355, 2022.
- Donatien Hainaut. Valuation of guaranteed minimum accumulation benefits (gmabs) with physics-inspired neural networks. *Annals of Actuarial Science*, pages 1–32, 2023.
- Donatien Hainaut and Alex Casas. Option pricing in the heston model with physics inspired neural networks. Technical report, Université catholique de Louvain, Institute of Statistics, Biostatistics and Actuarial Sciences, 2024.
- Jiequn Han, Arnulf Jentzen, et al. Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. *Communications in mathematics and statistics*, 5(4):349–380, 2017.
- Seyed Amir Hejazi and Kenneth R Jackson. A neural network approach to efficient valuation of large portfolios of variable annuities. *Insurance: Mathematics and Economics*, 70:169–181, 2016.
- Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.
- Blanka Horvath, Josef Teichmann, and Žan Žurič. Deep hedging under rough volatility. *Risks*, 9(7):138, 2021.
- Zheyuan Hu, Ameya D Jagtap, George Em Karniadakis, and Kenji Kawaguchi. Augmented physics-informed neural networks (apinns): A gating network-based soft domain decomposition methodology. *Engineering Applications of Artificial Intelligence*, 126:107183, 2023.
- John Hull and Alan White. Pricing interest-rate-derivative securities. *The review of financial studies*, 3(4):573–592, 1990.
- John Hull and Alan White. Numerical procedures for implementing term structure models i: Single-factor models. *Journal of derivatives*, 2(1):7–16, 1994.
- Kiyosi Itô. On stochastic processes (i) infinitely divisible laws of probability. In *Japanese journal of mathematics: transactions and abstracts*, volume 18, pages 261–301. The Mathematical Society of Japan, 1941.

- Ameya D Jagtap and George Em Karniadakis. Extended physics-informed neural networks (xpinns): A generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations. *Communications in Computational Physics*, 28(5), 2020.
- Michael Kohler, Adam Krzyżak, and Nebojsa Todorovic. Pricing of high-dimensional american options by neural networks. *Mathematical Finance: An International Journal of Mathematics, Statistics and Financial Economics*, 20(3):383–410, 2010.
- Hyuk Lee and In Seok Kang. Neural algorithm for solving differential equations. *Journal of Computational Physics*, 91(1):110–131, 1990.
- Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, 378:686–707, 2019.
- Justin Sirignano and Konstantinos Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. *Journal of computational physics*, 375:1339–1364, 2018.
- Emy Siswanah, Ahmad Mutawaslih Idrus, and Muhammad Malik Hakim. Binomial method in bermudan option. *Journal of Multidisciplinary Applied Natural Science*, 3(2):161–171, 2023.
- Jingtao Yao, Yili Li, and Chew Lim Tan. Option price forecasting using neural networks. *Omega*, 28(4):455–466, 2000.

