IMPERIAL

IMPERIAL COLLEGE LONDON

DEPARTMENT OF MATHEMATICS

MSCI RESEARCH PROJECT

Option Pricing Using Physics-Informed Neural Networks

Author:
Yiyan Li

Supervisor(s):
Mikko Pakkanen

Submitted in partial fulfillment of the requirements for the MSci in Mathematics at Imperial College London

June 9, 2025

Abstract

The option pricing problems can often be formulated in PDEs. As option pricing models aim to capture realistic market dynamics or handle complex derivative features such as early exercise or path dependence, analytical solutions are often unavailable. Hence, numerical PDE methods are essential in option pricing. This work applies Physics-Informed Neural Networks (PINNs) to option pricing. In addition to prevalent MLP-based PINNs, this work explores the latest developed Kolmogorov-Arnold network (KAN) based PINNs. The PINN method for option pricing provides a flexible framework for option pricing in PDE formulation, with capabilities of option greeks computation and easy handling of early-exercise/path-dependent features. The PINN method achieves comparable accuracy to conventional numerical methods for European and American options under the Black-Scholes model, which suggests a promising potential for more complex option pricing PDE models.

Acknowledgments

First and foremost, I would like to express my deepest gratitude to my supervisor, Dr. Mikko Pakkanen, for his invaluable advice and patient guidance throughout this project.

To my dear friends, some of you I have known for more than half a decade, thank you for your friendship and support. May you prosper wherever life takes you.

Thanks to my cat, whose quiet companionship made the writing of this thesis more enjoyable.

Special thanks to Selina for her constant presence and heart-warming chats on the Queen's Lawn.

Most importantly, I would like to thank my parents for their unconditional support and sponsorship for my life and study in London.

Dedicated to all who have walked beside me, this thesis marks the end of my four years at Huxley, Imperial College London. The to-be-continued page on my academic study turns on a new chapter: one of life, work, and a sense of belonging to this city I now call my second home.

Plagiarism statement

The work contained in this thesis is my own work unless otherwise stated.

Signature: Yiyan Li
Date: June 9, 2025

Contents

Chapter 1

Introduction

An *option* is a contract that gives the holder the right, but not the obligation, to buy or sell the underlying asset at a specified price (*strike price*) on or before a specified date (*expiry* or *maturity*), depending on the type of the option. An option that conveys to the holder the right to buy (sell) is referred to as a *call* (*put*). The style of an option determines the exercise conditions. The most common styles are *European*, *American*, and *Bermudan*. A European option can only be exercised at maturity, whereas an American option allows exercise at any time up to and including the expiry. Bermudan options allow exercise on a discrete set of times. Depending on the payoff structure, options are categorised into *vanilla* or *exotic*. Vanilla options have simple payoff structures (e.g. calls and puts), while exotic options involve more complex features such as path dependence (e.g. Asian options), barrier conditions (e.g. knock-in/knock-out), or early-exercise constraints.

The concept of options can be traced back to ancient times. The Greek mathematician and philosopher Thales of Miletus used an early form of options to speculate on the future use of olive presses. He made a fortune when olive production was as abundant as he had predicted, and presses were in high demand. In modern financial markets, the outstanding notional of options and other derivatives is estimated at hundreds of trillions USD (several times the world GDP). Though long-standing in history, it was not until the 20th century that the option pricing problem was mathematically formalised. The option pricing problem is to give a fair price for an option. The term 'fair' is often defined mathematically through the concept of no-arbitrage. Louis Bachelier first introduced Brownian motion in the modern study of options in 1990. The modern theory follows the 1997 Nobel-awarded Black, Scholes and Merton.

The Black-Scholes-Merton (BSM) model [Black and Scholes, 1973] models stock prices as geometric Brownian motions and provides a closed-form formula for European options. However, the BSM model relies on several idealised assumptions often violated in practice, such as constant volatility and constant interest rate. Since then, a wide range of pricing models have been developed to relax assumptions and generalise the BSM framework in various directions. Stochastic volatility models such as the Heston model [Heston, 1993] and the SABR model [Hagan et al., 2002] treat volatility as a stochastic process to capture observed market phenomena like the volatility smile. In the context of interest rate derivatives, models are typically cate-

gorised into short rate, forward rate, and market models. These pricing models use stochastic processes to describe the dynamics of the underlying asset and other parameters, hence, option pricing problems are often formulated in terms of stochastic differential equations (SDEs). Due to the Feymann-Kac theorem, which states that the solution to a certain class of PDEs can be represented as the expected value of a functional of a stochastic process, option pricing problems can therefore be equivalently formulated as solving PDEs.

As models aim to capture realistic market dynamics or handle complex derivative features such as early exercise or path dependence, analytical solutions are often unavailable or intractable. In such cases, one resorts to numerical methods. Conventional approaches include the finite difference methods (FDM) and Monte Carlo (MC) simulations. FDM discretises and solves the option pricing PDE on a grid. MC method simulates a large number of sample paths for the underlying stochastic process to estimate the expected payoff under the risk-neutral measure. PDE-based and MC methods have their limitations:

- Grid dependence: FDM is grid-based, which can be computationally expensive and difficult to scale.
- Curse of dimensionality: Both FDM and MC become increasingly inefficient in high dimensions (e.g. basket options).
- Handling early exercise: Early-exercise feature (e.g. American options) introduces an optimal stopping problem. In PDE formulation, this leads to a free boundary problem, typically requiring iterative schemes such as the penalty method, front-fixing, or variational inequality solvers, along with high-resolution grids near the boundary to ensure stability and accuracy. In MC setting, the Least Squares Monte Carlo (LSMC) [Longstaff and Schwartz, 2001] is commonly used to approximate the option continuation value through regression. However, LSMC performance is sensitive to the choice of basis functions and suffers from high variance and slow convergence.
- Handling path dependence: In PDE formulation, incorporating path-dependent features (e.g. the running average in Asian options) often requires augmenting the state space. Though MC handles path dependence naturally, it still suffers from slow convergence.

Deep learning has recently emerged as a new paradigm of scientific computing, leveraging the expressivity of neural networks guaranteed by the universal approximation theorem. In particular, *Physics-Informed Neural Networks* (PINNs) have gained increasing popularity as a mesh-free numerical method for solving PDEs. PINNs incorporate physical laws encoded in PDEs directly into the training objective of neural networks, converting the PDE problem to an optimisation problem. The PINN framework was first introduced by Raissi et al. [2019], in which nonlinear PDEs, such as Schrödinger, Burgers, and Allen-Cahn equations, are solved using PINNs to precision. Since then, various works on the applications and theoretical studies have been developed. The literature review on PINNs by Cuomo et al. [2022] gives a comprehensive overview on PINN applications across domains, such as data collection and dynamics simulation in fluid dynamics. Despite the success in certain scenarios, PINN still face numerous

unresolved issues, including theoretical concerns on convergence and stability, and implementation issues (optimal PINN architecture design, effective enforcement of boundary conditions, and optimisation aspects). While PINNs offer a flexible, mesh-free alternative to conventional numerical methods, they are generally viewed as a complementary framework rather than a replacement. Traditional methods still dominate with high precision and stability, whereas PINNs extend the usage case to data-driven modelling and inverse problem solving.

Kolmogorov-Arnold networks [Liu et al., 2025] are the latest proposed MLP alternatives inspired by the Kolmogorov-Arnold representation theorem, which are shown to exhibit superior performance and possess faster neural scaling laws. The majority of PINNs are MLP-based. However, Wang et al. [2025] show that KAN-based PINNs significantly outperform MLPs for various PDE problems in computational solid mechanics. The existing studies on KAN-based PINNs are limited, leading us to further exploration.

In the context of option pricing, PINN is an attractive method for several reasons:

- Unified framework: PINN offers a universal framework for solving option pricing problems in PDE formulation. In particular, it handles early-exercise and path-dependent features through learning the PDE dynamics.
- Differentiability: Once trained, PINN provides a continuously differentiable solution across the option domain. The differentiability enables the computation of option greeks with little extra computational cost.
- Mesh-free: PINN can potentially overcome the curse of dimensionality.

Given the success of PINNs and these benefits, we would like to investigate the usage of PINNs in option pricing. The usage of PINNs in option pricing is not new. Note that the PINN framework falls under the category of unsupervised learning, which is a fundamentally distinct approach compared to training neural networks directly approximating option value functions. Wang et al. [2023b] implement PINNs for pricing with a nonlinear Black-Scholes equation and two-asset options using the PINN library DeepXDE [Lu et al., 2021]. Their work aims to demonstrate the capability of PINNs for pricing options with nonlinear and high-dimensional PDEs. This work focuses on the comparison between the PINN method and conventional numerical methods for option pricing, with detailed mathematical formulation and empirical error analysis. In addition, the scope includes option greeks computation, empirically optimal PINN architecture selection, early exercise handling, and comparison between prevalent MLP-based PINNs to KAN-based PINNs in the context of option pricing. Our PINN solutions are implemented from scratch using PyTorch [Paszke et al., 2019] for maximal flexibility and accurate replication of theory.

The thesis is structured as follows. Chapter 2 introduces the PINN framework for solving PDEs. Chapter 3 covers the basics of option pricing, including the derivation of the Black-Scholes PDE, the PDE formulation for American options, and conventional numerical methods. Chapter 4 details the implementation and results for pricing European and American options using PINNs with comparison to conventional numerical methods and empirical error analysis.

Chapter 2

Physics-Informed Neural Networks

Physics-Informed Neural Networks (PINNs) are a scientific computing technique for solving problems involving partial differential equations (PDEs). Raissi et al. [2019] first introduced the PINN framework for solving nonlinear PDEs. The PINN approximates PDE solutions by a neural network (NN) with a custom loss function. The custom loss function includes terms reflecting the initial and boundary conditions and the PDE residual, so that the NN is forced to fit the given data while fulfilling the initial and boundary conditions and the governing equation. The PINN method converts the problem of solving PDEs to an optimisation problem, hence, it is mesh-free and potentially scalable to high dimensions, which sets PINN apart from conventional numerical PDE methods.

2.1 Physics-Informed Neural Networks for Solving PDEs

To introduce the PINN method for solving PDEs, this section adapts the problem setup in the literature review on PINNs by Cuomo et al. [2022]. Consider partial differential equations with initial or boundary conditions in the general form:

$$\begin{aligned} \mathcal{F}(u;\lambda)(x,t) &= f(x,t), \quad (x,t) \in \Omega \subseteq \mathbb{R}^{d-1} \times \mathbb{R}, \\ \mathcal{B}(u)(x,t) &= g(x,t), \quad (x,t) \in \partial\Omega, \end{aligned} \tag{2.1}$$

where u denotes the latent solution defined on the space-time domain Ω, with spatial coordinates $x \in \mathbb{R}^{d-1}$ and time $t \in \mathbb{R}$. The function f represents the source term, and g encodes initial or boundary conditions on $\partial\Omega$. The nonlinear operator $\mathcal{F}$ governs the physical laws with parameters λ, and $\mathcal{B}$ enforces the boundary and initial conditions.

The goal is to use a neural network $\hat{u}_\theta$ (where θ denotes the neural network parameters) to approximate the PDE solution u. A physics-informed neural network $\hat{\mathcal{F}}_\theta$ approximating $\mathcal{F}$ can then be derived using *automatic differentiation* (AD). The two networks $\hat{u}_\theta$ and $\hat{\mathcal{F}}_\theta$ share the same set of parameters, and the shared parameters can be learned by minimising a custom loss function of the form

$$\mathcal{L}_\theta = \omega_{\mathcal{F}}\mathcal{L}_{\mathcal{F}} + \omega_{\mathcal{B}}\mathcal{L}_{\mathcal{B}} + \omega_{\text{data}}\mathcal{L}_{\text{data}}, \tag{2.2}$$

where the total loss is the weighted sum of three terms: the approximation error for the PDE ($\mathcal{L}_{\mathcal{F}}$), the error in describing the initial or boundary conditions ($\mathcal{L}_{\mathcal{B}}$), and the approximation error for the solution u at known data points ($\mathcal{L}_{\text{data}}$). $\mathcal{L}_{\mathcal{F}}$ enforces the PDE at the *collocation points* (a set of points sampled in the domain). A typical choice for the loss function in PINN literature is the mean square error (MSE).

2.1.1 Automatic Differentiation

This section briefly introduces automatic differentiation, as it is an indispensable component in the PINN framework.

Automatic differentiation (AD), or algorithmic differentiation, is in theory an **exact method**. AD is neither numerical nor symbolic differentiation. AD exploits the fact that all numerical computations are compositions of elementary operations (binary arithmetic operations, the unary sign switch, transcendental functions, etc.) with known derivatives. By applying the chain rule to elementary operations, AD computes the derivative of the overall composition, with accuracy at the level of machine precision (up to a small constant factor). AD gives numeric evaluations of derivatives rather than symbolic expressions [Baydin et al., 2018].

Example: Forward Mode Automatic Differentiation

We illustrate how forward mode AD works through an example of evaluating the derivative of $f(x_1, x_2) = \ln x_1 + x_1 x_2 - \sin x_2$ w.r.t. x_1 at $(x_1, x_2) = (2, 5)$. AD associates each intermediate variable v_i with its derivative $\dot{v}_i = \frac{\partial v_i}{\partial x_1}$ w.r.t. x_1. Mathematically, forward mode AD evaluates a function using dual numbers: $f(v + \dot{v}\epsilon) = f(v) + f'(v)\dot{v}\epsilon$. The corresponding derivative trace is generated by applying the chain rule to each elementary operation in the forward primal trace. The detailed process is presented in Table 2.1.

Forward Primal Trace			**Forward Derivative Trace**		
v_{-1}	$= x_1$	$= 2$	$\dot{v}_{-1}$	$= \dot{x}_1$	$= 1$
v_0	$= x_2$	$= 5$	$\dot{v}_0$	$= \dot{x}_2$	$= 0$
v_1	$= \ln v_{-1}$	$= \ln 2$	$\dot{v}_1$	$= \dot{v}_{-1}/v_{-1}$	$= 1/2$
v_2	$= v_{-1} \cdot v_0$	$= 2 \times 5$	$\dot{v}_2$	$= \dot{v}_{-1} \cdot v_0 + v_{-1}\dot{v}_0$	$= 1 \times 5 + 2 \times 0$
v_3	$= \sin v_0$	$= \sin 5$	$\dot{v}_3$	$= \cos v_0 \cdot \dot{v}_0$	$= \cos 5 \times 0$
v_4	$= v_1 + v_2$	$= \ln 2 + 10$	$\dot{v}_4$	$= \dot{v}_1 + \dot{v}_2$	$= 1/2 + 5$
v_5	$= v_4 - v_3$	$= (\ln 2 + 10) - \sin 5$	$\dot{v}_5$	$= \dot{v}_4 - \dot{v}_3$	$= 5.5 - 0$
y	$= v_5$	$= 10 + \ln 2 - \sin 5$	$\dot{y}$	$= \dot{v}_5$	$= 5.5$

Table 2.1 An example of forward mode automatic differentiation.

AD in forward accumulation mode is the conceptually simplest type. For $f : \mathbb{R}^n \to \mathbb{R}^m$ with $n \gg m$, AD in reverse accumulation mode is preferred, which propagates derivatives backwards from a given output.

2.2 Neural Network Architecture

Although all types of neural networks, including multilayer perceptrons (MLPs), convolutional neural networks (CNNs), recurrent neural networks (RNNs), and Kolmogorov–Arnold networks (KANs), can fit in the PINN framework, the majority of PINNs are MLP-based since Raissi et al. [2019].

2.2.1 Multilayer Perceptrons

This section briefly introduces *multilayer perceptrons* (MLPs). A multilayer perception, also known as a feedforward network, consists of three or more fully connected layers of mathematical neurons with nonlinear activation functions. The expressiveness of MLPs is well guaranteed by the universal approximation theorem, which states that any continuous function can be approximated arbitrarily closely by an MLP with as few as one hidden layer [Hornik et al., 1989]. The mathematical definition is given as follows [Barahona and Bravi, 2024]:

Definition 2.2.1 (Multilayer perceptron)**.** A multilayer perceptron with L hidden layers is a function $f : \mathbb{R}^{n_0} \to \mathbb{R}^{n_{L+1}}$ defined recursively as

$$\begin{aligned} \boldsymbol{h}^{(0)} &= \boldsymbol{x}, \\ \boldsymbol{h}^{(l)} &= \sigma_l \left(\mathbf{W}^{(l-1)} \boldsymbol{h}^{(l-1)} + \boldsymbol{b}^{(l-1)} \right), \quad l = 1, ..., L, \\ y_{\text{out}} &= \sigma_{\text{out}} \left(\mathbf{W}^{(L)} \boldsymbol{h}^{(L)} + \boldsymbol{b}^{(L)} \right), \end{aligned}$$

where $\left\{ \mathbf{W}^{(l)} \in \mathbb{R}^{n_{l+1} \times n_l} \right\}_{l=1}^{L}$ and $\left\{ \boldsymbol{b}^{(l)} \in \mathbb{R}^{n_{l+1}} \right\}_{l=1}^{L}$ are weights and biases, n_l is the number of units in the l-th hidden layer, and σ_l are activation functions applied element-wise.

The architecture hyperparameters, including the number of layers and neurons in each layer, are often chosen empirically for specific problems. A deep neural network architecture can be difficult and computationally expensive to train, whereas a small architecture may fail to approximate the target PDE solution effectively.

To give an example of architectures used in PINN, in the work of Raissi et al. [2019], for a one-dimensional nonlinear Schrödinger equation $ih_t + 0.5h_{xx} + |h|^2 h = 0$ with some boundary conditions, a 5-layer MLP with 100 neurons per layer and a hyperbolic tangent activation function can achieve comparable solution accuracy to conventional methods.

Activation Functions

Although the rectified linear unit (ReLU) is widely used in deep learning, smooth activation functions are favoured due to the requirement of high-order derivatives in the PINN setting. For example:

- hyperbolic tangent, $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$,

- the swish function, $\text{swish}_\beta(x) = x \cdot \text{sigmoid}(\beta x) = \frac{x}{1+e^{-\beta x}}$, where β can be constant or trainable ($\beta = 1$ is known as the sigmoid linear unit (SiLU)),
- the Gaussian-error linear unit (GELU), $f(x) = x \cdot \Phi(x)$, where $\Phi(x)$ is the standard normal cumulative distribution function.

PINN performance has been empirically shown to be highly sensitive to the choice of activation functions when solving PDEs with distinct properties [Wang et al., 2023a]. Hence, the activation function should be chosen with caution. Often, the appropriate activation function needs to be found on a trial basis.

2.2.2 Kolmogorov–Arnold Networks

Kolmogorov–Arnold networks (KANs) [Liu et al., 2025] are promising alternatives to MLPs. In contrast to the universal approximation theorem for MLPs, KANs are inspired by the Kolmogorov-Arnold representation theorem, which states that any multivariate continuous function can be reconstructed as a finite composition of continuous univariate functions and additions. More precisely, for a smooth $f : [0, 1]^n \to \mathbb{R}$,

$$f(\boldsymbol{x}) = f(x_1, \ldots, x_n) = \sum_{q=1}^{2n+1} \Phi_q \left(\sum_{p=1}^{n} \phi_{q,p}(x_p) \right),$$

where $\phi_{q,p} : [0, 1] \to \mathbb{R}$ and $\Phi_q : \mathbb{R} \to \mathbb{R}$. KANs are similar to MLPs in structure, with weights replaced by univariate functions parametrised as *splines* (Appendix A). A KAN layer with n_{in} inputs and n_{out} outputs can be defined as a matrix of 1D functions $\boldsymbol{\Phi} = \{\phi_{q,p}\}_{p=1}^{n_{\text{in}}}{}_{q=1}^{n_{\text{out}}}$. A L-layer KAN is a composition of L layers:

$$\text{KAN}(\boldsymbol{x}) = (\boldsymbol{\Phi}_{L-1} \circ \boldsymbol{\Phi}_{L-2} \circ \cdots \circ \boldsymbol{\Phi}_1 \circ \boldsymbol{\Phi}_0)\, \boldsymbol{x}.$$

For comparison,

$$\text{MLP}(\boldsymbol{x}) = (\mathbf{W}_{L-1} \circ \sigma \circ \mathbf{W}_{L-2} \circ \sigma \circ \cdots \circ \mathbf{W}_1 \circ \sigma \circ \mathbf{W}_0)\, \boldsymbol{x},$$

where $\mathbf{W}$ are affine transformations and σ are activation functions. MLPs treat linear transformations and nonlinearities separately with $\mathbf{W}$ and σ, while KANs integrate them in $\boldsymbol{\Phi}$. For a KAN of depth-L with layers of equal width N with each spline of order k on G intervals ($G + 1$ grid points), there are in total $\mathcal{O}(N^2 L(G + k)) \sim \mathcal{O}(N^2 LG)$ parameters. In contrast, an MLP with the same structure requires only $\mathcal{O}(N^2 L)$ parameters. However, KANs usually require much smaller N compared to MLPs.

The KAN activation function is implemented as $\phi(x) = w_b b(x) + w_s \text{spline}(x)$. Here, w_b and w_s are trainable weights to control the magnitude of the activation function, $b(x)$ is a basis function (e.g. SiLU), the spline is parametrised as a linear combination of *B-splines* (Appendix A): $\text{spline}(x) = \sum_i c_i B_i(x)$ where c_is are trainable. The spline grids are updated based on input activations.

It has been shown that KANs outperform MLPs in terms of accuracy and interpretability on small-scale tasks, and possess theoretically and empirically faster neural scaling laws. In the context of PINNs, Wang et al. [2025] demonstrate that KINN-based PINNs significantly outperform MLP-based PINNs in accuracy and convergence speed for various PDE problems, such as multi-scale, singularity and heterogeneity.

2.3 Training Physics-Informed Neural Networks

The training of PINNs typically requires a large number of collocation points and a relatively small number of initial and boundary points. For example, for the one-dimensional nonlinear Schrödinger equation mentioned in Section 2.2, the training data consists of 50 initial data points, 50 boundary points, and 20000 collocation points.

2.3.1 Collocation Points Sampling

While the collocation points can be distributed arbitrarily in the domain, the distribution of the collocation points can affect the quality of PINN solutions. This section introduces commonly used sampling methods for PINNs, which can be generally divided into two categories:

- non-adaptive sampling, including uniform sampling, quasi-random low-discrepancy sequences (Sobol sequences),
- adaptive sampling, including residual-based adaptive sampling.

While uniform sampling is widely used and works well for some simple PDEs, it may not be sufficient for more complicated problems. *Sobol sequences* [Sobol, 1967] are a type of quasi-random low-discrepancy sequence (Appendix B). Quasi-random numbers cover the domain of interest quickly and evenly compared to pure random numbers. Based on empirical studies by Wu et al. [2023], PINNs using low-discrepancy sequences generally perform better than uniform distribution.

Residual-based adaptive sampling can significantly improve the performance of PINNs for certain PDEs of solutions with steep gradients [Lu et al., 2021]. There are various implementations of adaptive sampling. The main idea is to dynamically increase the focus on regions with large PDE residuals during the training process. For example, residual-based adaptive distribution (RAD) proposed by Wu et al. resamples collocation points from a probability density function $p(\boldsymbol{x}) \propto \frac{\epsilon^k(\boldsymbol{x})}{\mathbb{E}\left[\epsilon^k(\boldsymbol{x})\right]} + c$, where $\epsilon^k(\boldsymbol{x})$ is the PDE residual at $\boldsymbol{x}$, $\mathbb{E}\left[\epsilon^k(\boldsymbol{x})\right]$ is approximated by a numerical integration, $k \geq 0$ and $c \geq 0$ are hyperparameters. For PDEs with complicated solutions, such as Burgers' and multi-scale wave equation, RAD is predominately effective.

Remark. In addition to various sampling methods, resampling strategies are often used in conjunction to improve PINN performance.

2.3.2 Optimisation

The PINN framework determines the parameters θ of the NN $\hat{u}_\theta$ by minimising the loss function (2.2) defined above, i.e.

$$\theta = \arg\min_\theta \mathcal{L}_\theta.$$

A typical choice for the loss function is the mean square error (MSE). Then the loss function becomes

$$\mathcal{L}_\theta = \omega_{\mathcal{F}}\mathrm{MSE}_{\mathcal{F}} + \omega_{\mathcal{B}}\mathrm{MSE}_{\mathcal{B}} + \omega_{\text{data}}\mathrm{MSE}_{\text{data}},$$

where

$$\mathrm{MSE}_{\mathcal{F}} = \frac{1}{N_{\mathcal{F}}}\sum_{i=1}^{N_{\mathcal{F}}} \left|\mathcal{F}_\theta\left(u_\theta\left(z_{\mathcal{F}}^i\right)\right) - f\left(z_{\mathcal{F}}^i\right)\right|^2,$$

$$\mathrm{MSE}_{\mathcal{B}} = \frac{1}{N_{\mathcal{B}}}\sum_{i=1}^{N_{\mathcal{B}}} \left|\mathcal{B}\left(z_{\mathcal{B}}^i\right) - g\left(z_{\mathcal{B}}^i\right)\right|^2,$$

$$\mathrm{MSE}_{\text{data}} = \frac{1}{N_{\text{data}}}\sum_{i=1}^{N_{\text{data}}} \left|u_\theta\left(z_{\text{data}}^i\right) - u_{\text{data}}^i\right|^2.$$

Here, $\left\{z_{\mathcal{F}}^i\right\}_{i=1}^{N_{\mathcal{F}}}$ denote the set of collocation points, $\left\{z_{\mathcal{B}}^i\right\}_{i=1}^{N_{\mathcal{B}}}$ are the initial and boundary data, and $\left\{z_{\text{data}}^i, u_{\text{data}}^i\right\}_{i=1}^{N_{\text{data}}}$ are known data points for the PDE solution u. $\mathrm{MSE}_{\mathcal{F}}$ penalises the PDE (2.1) not being satisfied on the collocation points. $\mathrm{MSE}_{\mathcal{B}}$ and $\mathrm{MSE}_{\text{data}}$ enforce the NN at the initial and boundary points and given data. For the weights of the loss components, the loss components should be scaled to avoid dominance of high-magnitude terms.

In PINN literature, the loss function (2.2) is often optimised using minibatch sampling using *Adam* [Kingma and Ba, 2017] and the *limited-memory Broyden–Fletcher–Goldfarb–Shanno* (L-BFGS) algorithm [Liu and Nocedal, 1989]. One common practice is to use Adam as an initial optimiser for some epochs and then continue with L-BFGS. The rationale for this combination is to quickly approach a good solution region in the initial stage using Adam, and leverage L-BFGS to refine the solution with curvature (second-order) information. Below introduces these two optimisation methods.

Remark. In addition to the loss terms defined above, penalty terms like L1 and L2 regularisation terms can be added to address overfitting.

Adam

Adam is a first-order gradient-based optimisation algorithm, which is computationally efficient with little memory requirement. The name Adam derives from adaptive moment estimation. Adam combines the advantages of two popular methods: the ability of *AdaGrad* [Duchi et al., 2011] to work with sparse gradients, and the ability of *RMSProp* [Tieleman and Hinton, 2012] to handle non-stationary objectives.

Given a noisy objective function $f(\theta)$ (a stochastic scalar function, where the stochasticity

might come from minibatch sampling) differentiable w.r.t. parameters θ, The goal is to minimise the expectation $\mathbb{E}[f(\theta)]$ w.r.t. parameters θ. Let $g_k = \nabla_\theta f(\theta_k)$ denote the gradient w.r.t. θ at timestep k, m_k and v_k denote the exponential moving averages of the gradient and the squared gradient. The moving averages are estimates of the first moment (mean) and the second raw moment (uncentred variance) of the gradient, initialised to zero. $\beta_1, \beta_2 \in [0, 1)$ are exponential decay rates for moving averages. α is the learning rate. ϵ is a small constant for numerical stability. The Adam algorithm [Kingma and Ba, 2017] is given in Algorithm 1.

Adam incorporates momentum as an estimate of the (exponentially weighted) first-order moment of the gradient. Bias correction terms are applied to the moment estimates to account for the initialisation at zero, which can be derived by considering the expectation of the exponential moving averages (detailed derivation in Kingma and Ba [2017, Section 3]).

Algorithm 1: Adam

Input: α: learning rate
Input: $\beta_1, \beta_2 \in [0, 1)$: exponential decay rates for moment estimates

$m_0 \leftarrow 0$; /* 1st moment vector */
$v_0 \leftarrow 0$; /* 2nd moment vector */
$k \leftarrow 0$;
while θ_k *not converged* **do**
  $k \leftarrow k + 1$;
  $g_k \leftarrow \nabla_\theta f(\theta_{k-1})$;
  $m_k \leftarrow \beta_1 \cdot m_{k-1} + (1 - \beta_1) \cdot g_k$; /* biased 1st moment estimate */
  $v_k \leftarrow \beta_2 \cdot v_{k-1} + (1 - \beta_2) \cdot g_k^2$; /* biased 2nd raw moment estimate */
  $\hat{m}_k \leftarrow m_k / (1 - \beta_1^k)$; /* bias-corrected 1st moment estimate */
  $\hat{v}_k \leftarrow v_k / (1 - \beta_2^k)$; /* bias-corrected 2nd raw moment estimate */
  $\theta_k \leftarrow \theta_{k-1} - \alpha \cdot \hat{m}_k / (\sqrt{\hat{v}_k} + \epsilon)$;
end

L-BFGS

The limited-memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS) algorithm approximates the Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm using limited memory. BFGS is a quasi-Newton method, which approximates (the inverse of) the Hessian matrix using a positive definite matrix in each iteration. BFGS avoids inverting the Hessian by directly approximating the inverse. Hence, the computational complexity of BFGS is only $\mathcal{O}(n^2)$ compared to $\mathcal{O}(n^3)$ of Newton's method (n is the number of parameters) [Nocedal and Wright, 2006].

There are several limits of BFGS for large-scale problems. BFGS stores the dense Hessian matrix, requiring quadratic memory cost. The quadratic computational cost is also infeasible in high dimensions. To address these issues, L-BFGS were developed. L-BFGS uses a two-loop recursion and maintains vectors in m most recent iterations to approximate the search direction directly. This avoids forming the Hessian explicitly and storing the full matrix, giving $\mathcal{O}(mn)$ computational and space complexity (m is the memory size, the number of most recent iterations to store).

Let f be the objective function to be minimised and $g_k = \nabla f(\theta_k)$ be the gradient at timestep k. The m most recent iterations of the form $s_k = \theta_{k+1} - \theta_k$ (the change in position) and $y_k = g_{k+1} - g_k$ (the change in gradient) are stored. The L-BFGS algorithm using two-loop recursion is given in Algorithm 2. The algorithm requires a stopping criterion, usually $\|g_k\| < \epsilon$ (where ϵ is the tolerance) or reaching the maximum number of iterations. The two loops essentially compute the search direction $d_k = -H_k g_k$, where H_k is the inverse Hessian approximation (initialised to a scaled identity matrix). Note that L-BFGS does not explicitly form the Hessian. Instead, the matrix-vector product $H_k g_k$ is directly computed using the stored vector pairs s_k and y_k. After the search direction computation, a line search is performed to find a step size α_k satisfying *Wolfe conditions*. A step size α_k satisfies Wolfe conditions, restricted to the descent direction d_k, if the following two inequalities hold:

$$f(\theta_k + \alpha_k d_k) \leq f(\theta_k) + c_1 \alpha_k d_k^\top \nabla f(\theta_k),$$
$$-d_k^\top \nabla f(\theta_k + \alpha_k d_k) \leq -c_2 d_k^\top \nabla f(\theta_k),$$

where $0 < c_1 < c_2 < 1$. The first inequality is known as the Armijo rule to ensure sufficient decrease in the objective function. The second inequality is the curvature condition to preserve curvature information.

Algorithm 2: L-BFGS

Input: m: memory size, the number of most recent iterations to store

$g_0 = \nabla f(\theta_0)$;
$k \leftarrow 0$;
while *stopping criterion not met* **do**
  $q \leftarrow g_k$;
  for $i = k-1, k-2, \ldots, \max(k-m, 0)$ **do**
    $\alpha_i \leftarrow \frac{s_i^T q}{y_i^T s_i}$;
    $q \leftarrow q - \alpha_i y_i$;
  end
  $H_k^0 \leftarrow \frac{s_{k-m}^T y_{k-m}}{y_{k-m}^T y_{k-m}} I$; `/* initialise inverse Hessian approximation */`
  $p \leftarrow H_k^0 q$;
  for $i = \max(k-m, 0), \ldots, k-1$ **do**
    $\beta_i \leftarrow \frac{y_i^T p}{y_i^T s_i}$;
    $p \leftarrow p + s_i (\alpha_i - \beta_i)$;
  end
  $d_k \leftarrow -p$; `/* search direction */`
  $\alpha_k \leftarrow \text{LineSearch}(\theta_k, p_k)$; `/* find step size satisfying Wolfe conditions */`
  $\theta_{k+1} \leftarrow \theta_k + \alpha_k d_k$;
  $g_{k+1} = \nabla f(\theta_{k+1})$;
  $s_k \leftarrow x_{k+1} - x_k$;
  $y_k \leftarrow g_{k+1} - g_k$;
  $k \leftarrow k + 1$;
end

2.4 Example: Black-Scholes Equation for European Put Options

This section illustrates the PINN framework through an example of the Black-Scholes equation for European put options. The Black-Scholes PDE describes the price evolution of derivatives under the Black-Scholes model [Black and Scholes, 1973]. A European put option gives the holder the right to sell the underlying asset at a strike price on an expiration date. Its payoff is $(K - S_T)^+$ at expiry, where K is the strike price and S_T is the spot price of the underlying.

For European put options, we have the following Black-Scholes PDE with initial and boundary conditions

$$\begin{aligned} \frac{\partial V}{\partial \tau} - rS\frac{\partial V}{\partial S} - \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rV = 0, \quad (\tau, S) \in (0, T] \times \mathbb{R}^+, \\ V(0, S) = (K - S)^+, \\ V(\tau, 0) = Ke^{-r\tau}, \\ V(\tau, \infty) = 0, \end{aligned}$$

where $V(\tau, S)$ is the value of the derivative at time τ with underlying asset price S at the time, K is the strike price, σ is the volatility and r is the risk-free rate. Here $\tau = T - t$ is the time to maturity T (t is the instantaneous time). τ is used instead of t to convert the terminal condition (payoff at expiry) to the initial condition for the PDE.

Fitting this example into the PINN framework, we start with a neural network $\hat{V}_\theta$ to approximate the solution V. Using automatic differentiation, an physics-informed neural network can be derived: $\hat{\mathcal{F}}_\theta = \frac{\partial \hat{V}_\theta}{\partial \tau} - rS\frac{\partial \hat{V}_\theta}{\partial S} - \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 \hat{V}_\theta}{\partial S^2} + r\hat{V}_\theta$.

To solve the PDE, we proceed to define the loss function and then solve the resulting optimisation problem. Using MSE as the loss function

$$\mathcal{L}_\theta = \omega_{\mathcal{F}} \mathrm{MSE}_{\mathcal{F}} + \omega_{\mathcal{B}} \mathrm{MSE}_{\mathcal{B}} + \omega_{\text{data}} \mathrm{MSE}_{\text{data}},$$

where

$$\begin{aligned} \mathrm{MSE}_{\mathcal{F}} &= \frac{1}{N_{\mathcal{F}}} \sum_{i=1}^{N_{\mathcal{F}}} \left| \hat{\mathcal{F}}_\theta \left(\tau_{\mathcal{F}}^i, S_{\mathcal{F}}^i \right) \right|^2, \\ \mathrm{MSE}_{\mathcal{B}} &= \frac{1}{N_{\mathcal{B}}} \sum_{i=1}^{N_{\mathcal{B}}} \left| \hat{V}_\theta \left(\tau_{\mathcal{B}}^i, S_{\mathcal{B}}^i \right) - V_{\mathcal{B}}^i \right|^2, \\ \mathrm{MSE}_{\text{data}} &= \frac{1}{N_{\text{data}}} \sum_{i=1}^{N_{\text{data}}} \left| \hat{V}_\theta \left(\tau_{\text{data}}^i, S_{\text{data}}^i \right) - V_{\text{data}}^i \right|^2. \end{aligned}$$

Here, $\left\{ \tau_{\mathcal{F}}^i, V_{\mathcal{F}}^i \right\}_{i=1}^{N_{\mathcal{F}}}$ denote the set of collocation points, $\left\{ \tau_{\mathcal{B}}^i, S_{\mathcal{B}}^i, V_{\mathcal{B}}^i \right\}_{i=1}^{N_{\mathcal{B}}}$ are the initial and boundary data, and $\left\{ \tau_{\text{data}}^i, S_{\text{data}}^i, V_{\text{data}}^i \right\}_{i=1}^{N_{\text{data}}}$ are known data points for $V(\tau, S)$. $\mathrm{MSE}_{\mathcal{F}}$ penalises the Black-Scholes PDE not being satisfied on the collocation points. $\mathrm{MSE}_{\mathcal{B}}$ and $\mathrm{MSE}_{\text{data}}$ enforce the NN on initial and boundary points and given data.

2.5 Implementation

PyTorch [Paszke et al., 2019] is used to implement PINN solutions for maximal flexibility and accurate replication of theory. The PINNs are implemented as custom PyTorch modules. Custom NN architectures, loss functions, and training loops are designed for different PDE problems. PyTorch automatic differentiation (`torch.autograd`) is used to differentiate neural networks to evaluate PDE residuals. The source code for PINN modules and training notebooks can be found on GitHub.

To efficiently train PINNs, early stopping is implemented so that the training will be terminated if the loss function does not improve for some epochs. All PINNs are trained until convergence (the loss function plateaus).

2.5.1 Evaluation and Comparison

In order to assess the PINN performance, PINN solutions are benchmarked against reference solutions (analytical solutions if available or conventional numerical methods). The root mean square error (RMSE) is used as an evaluation metric, as it is in the same unit as the function value and provides clear interpretation for PDE problems as an overall measure of the deviation between the PINN solution from the reference. In absence of ground truth, the root mean square deviation (RMSD) is used to quantify the similarity between two solutions. The PINN solutions and analytical/numerical solutions are evaluated on the same set of points (an equispaced uniform grid) for fair comparison. For grid-based numerical methods, predictions for off-grid points are made by (linear) interpolation.

Note that the PINN performance varies due to the randomness in the NN parameter initialisation and the optimisation process. To ensure the robustness of results, the training process is repeated multiple times for a PINN architecture with different random states to obtain an empirical average of the evaluation metric. The empirically best-performing PINN is selected based on the evaluation metric.

Chapter 3

Option Pricing Basics

An *option* is a contract that gives the holder the right but not the obligation to buy or sell the underlying asset at a specified price (*strike price*) on or before a specified date (*expiry* or *maturity*), depending on the type of the option. The pricing problem is to give a fair price for such an option. The arbitrage-free price can be defined as the value of a non-arbitrage self-financing trading strategy replicating the option payoff, or interpreted as the expected value of the discounted payoff under a risk-neutral measure by the Feynman-Kac theorem.

3.1 Black-Scholes PDE

This section gives a derivation of the Black-Scholes PDE [Brigo, 2024], which uses a self-financing trading strategy to replicate the option payoff to define the notion of arbitrage-free prices.

The Black-Scholes model assumes that:

- there is no trading transaction cost;
- the underlying asset pays no dividends;
- shares are infinitely divisible;
- short selling is allowed without any restriction penalty.

Consider a probability space with a right continuous filtration $(\Omega, \mathcal{F}, \{\mathcal{F}_t\}_{0\leq t\leq T}, \mathbb{P})$. In the given economy, two securities, a riskless bond (or money market account) and a risky asset (a stock), are traded from time 0 to T. The riskless bond price B_t evolves according to $\mathrm{d}B_t = B_t r \mathrm{d}t$ with $B_0 = 1$, i.e. $B_t = e^{rt}$, where r is the risk-free rate. The stock price S_t is modelled by the geometric Brownian motion $\mathrm{d}S_t = \mu S_t \mathrm{d}t + \sigma S_t \mathrm{d}W_t$ with deterministic initial condition S_0, where μ and σ are positive constants. Assume the value of the option V at time t is a function of the underlying stock S at the same time, which is once continuously differentiable w.r.t. t and twice continuously differentiable w.r.t. S, i.e. $V_t = V(t, S_t) \in C^{1,2}\left([0,T]\times\mathbb{R}^+\right)$.

Applying Itô's formula to V and substituting $\mathrm{d}S_t = \mu S_t \mathrm{d}t + \sigma S_t \mathrm{d}W_t$:

$$\mathrm{d}V(t,S_t) = \frac{\partial V}{\partial t}(t,S_t)\mathrm{d}t + \frac{\partial V}{\partial S}(t,S_t)\mathrm{d}S_t + \frac{1}{2}\frac{\partial^2 V}{\partial S^2}(t,S_t)\mathrm{d}S_t\mathrm{d}S_t \tag{3.1}$$

$$= \left(\frac{\partial V}{\partial t}(t,S_t) + \mu S_t \frac{\partial V}{\partial S}(t,S_t) + \frac{1}{2}\sigma^2 S_t^2 \frac{\partial^2 V}{\partial S^2}(t,S_t)\right)\mathrm{d}t + \sigma S_t \frac{\partial V}{\partial S}(t,S_t)\mathrm{d}W_t. \tag{3.2}$$

A *trading strategy* is a pair of stochastic processes $\phi = (\phi_t^B, \phi_t^S)$ on $\left(\Omega, \mathcal{F}, \{\mathcal{F}_t\}_{0\le t\le T}, \mathbb{P}\right)$ where the pair represents the amounts of bond and stock to be held at time t. $V_t(\phi) = \phi_t^B B_t + \phi_t^S S_t$ is the *value process* describing the value of the portfolio constructed by the trading strategy ϕ. $G_t(\phi) = \int_0^t \phi_u^B \mathrm{d}B_u + \int_0^t \phi_u^S \mathrm{d}S_u$ is the *gain process* representing the income from price movements in bond and stock. A trading strategy is *self-financing* if $\mathrm{d}V_t(\phi) = \mathrm{d}G_t(\phi)$, meaning that there is no cash inflow and outflow after the initial investment and the change in portfolio value is solely due to price movements. An arbitrage is a self-financing strategy such that $V_0(\phi) = 0$ and $\mathbb{P}\left(V_t(\phi) > 0\right) > 0$.

The idea is to replicate the option payoff with a self-financing strategy so that $V(t,S_t) = \phi_t^B B_t + \phi_t^S S_t$, that is

$$\mathrm{d}V(t,S_t) = \phi_t^B \mathrm{d}B_t + \phi_t^S \mathrm{d}S_t. \tag{3.3}$$

Comparing Equation (3.1) and (3.3) and matching the dS_t terms, we get $\phi_t^S = \frac{\partial V}{\partial S}$. Then $\phi_t^B = (V(t,S_t) - \phi_t^S S_t)/B_t$. Substituting back into (3.3):

$$\mathrm{d}V(t,S_t) = \left(V(t,S_t) - S_t \frac{\partial V}{\partial S}(t,S_t)\right) r\mathrm{d}t + \frac{\partial V}{\partial S}(t,S_t)(\mu S_t \mathrm{d}t + \sigma S_t \mathrm{d}W_t). \tag{3.4}$$

Equating (3.2) and (3.4) gives the renowned Black-Scholes PDE:

$$\frac{\partial V}{\partial t}(t,S_t) + \frac{1}{2}\sigma^2 S_t^2 \frac{\partial^2 V}{\partial S^2}(t,S_t) + rS_t \frac{\partial V}{\partial S}(t,S_t) - rV(t,S_t) = 0, \quad t < T, \tag{3.5}$$

with terminal condition $V(T,S) = g(S)$ where g is the payoff function.

Remark. A change of variable $\tau = T - t$ is often applied to convert the PDE terminal condition to the initial condition for numerical methods, where τ represents the time to expiry and t is the instantaneous time.

Remark. The Black-Scholes PDE is invariant (up to a constant) under price scaling: $\tilde{S} = \frac{S}{c}, \tilde{V} = \frac{V}{c}$. Therefore, one can scale the price range freely for normalisation.

The Black-Scholes PDE is a parabolic PDE and its solution can be interpreted as the expected value of the discounted payoff under a risk-neutral measure by the Feynman-Kac theorem. This gives a second notion of price as the expectation of the discounted payoff under a risk-neutral measure.

Theorem 3.1.1 (Feynman-Kac)**.** *Under regularity conditions, the solution of the PDE*

$$\frac{\partial V}{\partial t}(t,x) + \frac{1}{2}\sigma^2(x)\frac{\partial^2 V}{\partial x^2}(t,x) + b(x)\frac{\partial V}{\partial x}(t,x) - rV(t,x) = 0, \quad V(T,x) = f(x),$$

can be expressed as

$$V(t,x) = e^{-r(T-t)}\mathbb{E}^{\mathbb{Q}}\left[f(X_T \mid \mathcal{F}_t)\right],$$

where the diffusion process has dynamics

$$\mathrm{d}X_s = b(X_s)\mathrm{d}s + \sigma(X_s)\mathrm{d}W_s^{\mathbb{Q}}, \quad s \geq t, \quad X_t = x,$$

under the probability measure $\mathbb{Q}$*.* $W^{\mathbb{Q}}$ *is a standard Brownian motion under* $\mathbb{Q}$*.*

3.1.1 Black-Scholes Formula for European Options

A European call (put) option gives the holder the right to buy (sell) the underlying asset at a strike price on an expiration date. The payoff at the expiry is $(S-K)^+$ for call options and $(K-S)^+$ for put options.

The Black-Scholes equation has an analytical solution for the European case. For European call options, the solution is given by

$$V_{\text{CALL}}^{\text{BS}}(t) = S_t\Phi(d_1(t)) - Ke^{-r(T-t)}\Phi(d_2(t)), \tag{3.6}$$

where $d_{1,2}(t) = \frac{\ln(S_t/K)+(r\pm\sigma^2/2)(T-t)}{\sigma\sqrt{T-t}}$, Φ is standard normal cumulative distribution function. The formula for European put options can be derived via the *put-call parity*:

$$\text{PutPrice} = \text{CallPrice} - \text{ForwardPrice},$$

where ForwardPrice $V_{\text{FWD}}(t) = e^{-r(T-t)}\mathbb{E}^{\mathbb{Q}}\left[S_T - K\right] = S_t - e^{-r(T-t)}K$.

3.1.2 Crank-Nicolson Method for Black-Scholes PDE

Often, the PDEs arising in option pricing cannot be solved analytically. This section introduces the Crank-Nicolson finite difference method for solving Black-Scholes PDE based on Higham [2004].

As problems in numerical PDEs are usually specified in initial time condition form, a change of variable $\tau = T - t$ is applied to the Black-Scholes PDE (3.5):

$$\frac{\partial V}{\partial \tau} - \frac{1}{2}\sigma^2 S^2\frac{\partial^2 V}{\partial S^2} - rS\frac{\partial V}{\partial S} + rV = 0, \tag{3.7}$$

so that the $t = T$ condition becomes the $\tau = 0$ condition $V(0,S) = g(S)$. For boundary conditions, a reasonably large number L is set to truncate the unbounded domain $S \in \mathbb{R}^+$ to $S \in [0, L]$

The time and space axes are divided into $N_t + 1$ and $N_x + 1$ equally spaced points to form a grid $\{i\Delta t, j\Delta x\}_{i=0}^{N_t}{}_{j=0}^{N_x}$ where $\Delta t = T/N_t$ and $\Delta x = L/N_x$. Letting $\mathbf{V}^i = \left[V_1^i, \ldots, V_{N_x-1}^i\right]^\top \in \mathbb{R}^{N_x-1}$ denote the numerical solution at time step i, V^0 is specified by the initial condition, and the boundary values V_0^i and $V_{N_x}^i$ are specified by boundary conditions, for all $1 \leq i \leq N_t$.

To introduce Crank-Nicolson, two basic finite difference methods are required: the FTCS (forward time, centred space) and the BTCS (backward time, centred space) methods. As their names suggest, the FTCS and the BTCS methods use forward and backward differences to approximate the time derivative, and full central and second-order central differences to approximate the space derivatives.

The FTCS scheme for the Black-Scholes PDE (3.7) is

$$\frac{V_j^{i+1}-V_j^i}{\Delta t}-\frac{1}{2}\sigma^2(j\Delta x)^2\frac{V_{j+1}^i-2V_j^i+V_{j-1}^i}{(\Delta x)^2}-r(j\Delta x)\frac{V_{j+1}^i-V_{j-1}^i}{2\Delta x}+rV_j^i=0,$$

which can be written in matrix form

$$\mathbf{V}^{i+1}=\mathbf{F}\mathbf{V}^i+\boldsymbol{p}^i,\quad 0\le i\le N_t-1, \tag{3.8}$$

where

$$\boldsymbol{p}^i=\left[\frac{1}{2}\Delta t(\sigma^2-r)V_0^i,0,...,0,\frac{1}{2}\Delta t(N_x-1)(\sigma^2(N_x-1)+r)V_{N_x}^i\right]^\top,$$
$$\mathbf{F}=(1-r\Delta t)\mathbf{I}+\frac{1}{2}\sigma^2\Delta t\mathbf{D}_2\mathbf{T}_2+\frac{1}{2}r\Delta t\mathbf{D}_1\mathbf{T}_1,$$

$$\mathbf{D}_1=\begin{bmatrix}1&0&\dots&\dots&0\\0&2&0&\ddots&\vdots\\\vdots&0&3&\ddots&\vdots\\\vdots&\ddots&\ddots&\ddots&0\\0&\dots&\dots&0&N_x-1\end{bmatrix},\quad \mathbf{D}_2=\begin{bmatrix}1^2&0&\dots&\dots&0\\0&2^2&0&\ddots&\vdots\\\vdots&0&3^2&\ddots&\vdots\\\vdots&\ddots&\ddots&\ddots&0\\0&\dots&\dots&0&(N_x-1)^2\end{bmatrix},$$

$$\mathbf{T}_1=\begin{bmatrix}0&1&0&\dots&\dots&0\\-1&0&1&\ddots&\ddots&\vdots\\0&\ddots&\ddots&\ddots&\ddots&\vdots\\\vdots&\ddots&\ddots&\ddots&\ddots&0\\\vdots&\ddots&\ddots&-1&0&1\\0&\dots&\dots&0&-1&0\end{bmatrix},\quad \mathbf{T}_2=\begin{bmatrix}-2&1&0&\dots&\dots&0\\1&-2&1&\ddots&\ddots&\vdots\\0&1&\ddots&\ddots&\ddots&\vdots\\\vdots&\ddots&\ddots&\ddots&\ddots&0\\\vdots&\ddots&\ddots&1&-2&1\\0&\dots&\dots&0&1&-2\end{bmatrix}.$$

Similarly, the matrix form of the BTCS scheme is given by

$$\mathbf{B}\mathbf{V}^{i+1}=\mathbf{V}^i+\boldsymbol{q}^i,\quad 0\le i\le N_t-1, \tag{3.9}$$

where

$$\boldsymbol{q}^i=\left[\frac{1}{2}\Delta t(\sigma^2-r)V_0^{i+1},0,\dots,0,\frac{1}{2}\Delta t(N_x-1)(\sigma^2(N_x-1)+r)V_{N_x}^{i+1}\right]^\top,$$
$$\mathbf{B}=(1+r\Delta t)\mathbf{I}-\frac{1}{2}\sigma^2\Delta t\mathbf{D}_2\mathbf{T}_2-\frac{1}{2}r\Delta t\mathbf{D}_1\mathbf{T}_1.$$

Both FTCS and BTCS are of local accuracy $\mathcal{O}(\Delta t) + \mathcal{O}((\Delta x)^2)$. The $\mathcal{O}(\Delta t)$ accuracy in time arises from first-order forward and backward difference for the time derivative approximation. The idea of the Crank-Nicolson method is to take the average of the FTCS and the BTCS formulae (3.8) and (3.9) to achieve second-order in time:

$$\frac{1}{2}(\mathbf{I} + \mathbf{B})\mathbf{V}^{i+1} = \frac{1}{2}(\mathbf{I} + \mathbf{F})\mathbf{V}^{i} + \frac{1}{2}(\boldsymbol{p}^i + \boldsymbol{q}^i). \tag{3.10}$$

Like BTCS, Crank-Nicolson is an implicit method, which essentially requires a tridiagonal system of linear equations to be solved.

3.2 PDE Formulation for American Options

In contrast to a European option, an American option can be exercised at any time before or at the expiry date. This early-exercise feature adds extra complexity to the American option pricing problem, leading to an optimal stopping problem. This section briefly introduces two formulations of the American option pricing problem: an optimal stopping problem via free-boundary approach and in terms of variational inequalities [Musiela and Rutkowski, 2005].

Let $\left(\Omega, \mathcal{F}, \{\mathcal{F}_t\}_{0\leq t\leq T}, \mathbb{P}\right)$ be a filtered probability space. A *stopping time* is a random variable $\tau : \Omega \to \mathbb{R}^+$ such that $\{\tau \leq t\} \in \mathcal{F}_t$ for all t. Let $\mathcal{T}_{[t,T]}$ denote the set of admissible stopping times in $[t, T]$.

Observe that the value of an American option is at least as valuable as its European counterpart (otherwise there exists an arbitrage opportunity). This motivates the definition of American option price in the risk-neutral pricing setting. The price of an American option at time t can be defined as $V(t, S_t) = \sup_{\tau\in\mathcal{T}_{[t,T]}} \mathbb{E}^{\mathbb{Q}}\left[e^{-r(T-t)}g(S_\tau) \mid S_\tau\right]$, where $\mathbb{Q}$ is the risk-neutral measure and g is the payoff function. This definition suggests the optimal exercise policy for American options. Define the *continuation region* $\mathcal{C} = \{(t, S) \in [0, T] \times \mathbb{R}^+ \mid V(t, S) > g(S)\}$ and the *stopping region* $\mathcal{D} = \{(t, S) \in [0, T] \times \mathbb{R}^+ \mid V(t, S) = g(S)\}$. The American option holder should continue to hold the option in the continuation region and exercise the option in the stopping region. The critical stock price at time t is $c^*(t) = \sup\{S \in \mathbb{R}^+ \mid V(t, S) = g(S)\}$ and the optimal stopping time after time t is $\tau^* = \inf\{u \in [t, T] \mid S_u \leq c^*(u)\}$.

For an American call option written on a non-dividend-paying stock, it can be shown using an arbitrage argument that its value coincides with its European counterpart [Higham, 2004]. Let S_t be the underlying asset price at time t, K be the strike price, and T be the expiry date. Early exercise only makes sense if $S_t > K$ at some $t < T$, giving payoff $S_t - K$. However, one can potentially realise an even larger payoff by the following strategy:

1. At time t, short sell the underlying at the market price S_t
2. At expiry T,
   (a) exercise the call option if $S_T > K$,
   (b) buy at the spot price S_T otherwise.

The former case (a) recovers the early-exercise payoff and the latter (b) yields a larger or equal payoff. We conclude that early exercise for such an American call option is never optimal, hence an American call option must have the same value as its corresponding European call option.

For an American put option, we state the following results on the PDE formulation (the proof is beyond the scope of this study). The value function of an American put option can be characterised as the solution to a free boundary problem, which satisfies the Black-Scholes PDE in the continuation region and is greater or equal to the option's intrinsic value.

Theorem 3.2.1. *Let $\mathcal{G} \subseteq [0,T] \times \mathbb{R}^+$ be an open domain with continuously differentiable boundary c^*. Assume that $V(t,S) \in C^{1,2}(\mathcal{G})$ with certain growth conditions satisfies*

$$\begin{aligned} \frac{\partial V}{\partial t}(t,S) + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2}(t,S) + rS\frac{\partial V}{\partial S}(t,S) - rV(t,S) = 0, \qquad & (t,S) \in \mathcal{G}, \\ V(t,S) > g(S), \qquad & (t,S) \in \mathcal{G}, \\ V(t,S) = g(S), \qquad & (t,S) \in \mathcal{G}^c, \\ V(T,S) = g(S), \qquad & S \in \mathbb{R}^+. \end{aligned}$$

Then V is the value function of the American option with payoff function g. Moreover, the sets $\mathcal{G} = \mathcal{C}$ and $\mathcal{G}^c = \mathcal{D}$ are the continuation and stopping regions of the option, and the boundary c^* is the critical stock price.

In addition to the free-boundary approach, the general theory of variational inequalities gives a formulation without an explicit optimal stopping boundary, allowing us to treat the option domain as one entire region [Jaillet et al., 1990].

Theorem 3.2.2. *Assume $V(t,S) \in C^{1,2}([0,T] \times \mathbb{R}^+)$ satisfies certain growth conditions. Let $\mathcal{L} = \frac{\partial}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2}{\partial S^2} + rS\frac{\partial}{\partial S} - r$ be the differential operator for the Black-Scholes PDE. Suppose for $(t,S) \in [0,T] \times \mathbb{R}^+$ we have*

$$\begin{aligned} \mathcal{L}V(t,S) &\leq 0, \\ V(t,S) &\geq g(S), \\ V(T,S) &= g(S), \\ (V(t,S) - g(S))\,\mathcal{L}V(t,S) &= 0. \end{aligned}$$

Equivalently,

$$\begin{aligned} \min\{-\mathcal{L}V(t,S), V(t,S) - g(S)\} &= 0, \\ V(T,S) &= g(S). \end{aligned}$$

Then V is the value function of the American option with payoff function g.

Theorem 3.2.2 essentially states that an American option at time t either satisfies the Black-Scholes PDE or its value is greater than its intrinsic value, which provides an adequate framework for the study of numerical methods.

3.2.1 Numerical Methods for Pricing American Options

In the absence of analytical solutions for American options, this section introduces two conventional numerical methods: the *Cox-Ross-Rubinstein* (CRR) binomial tree and a PDE-based approach.

CRR Binomial Tree

The Cox-Ross-Rubinstein (CRR) model [Cutland and Roux, 2013] is a binomial model in discrete time. Consider a probability space with a natural filtration $(\Omega, \mathcal{P}(\Omega), \{\mathcal{F}_n\}_{0\leq n\leq N}, \mathbb{P})$, where $\Omega = \{\omega = (\omega_1, ..., \omega_N) : \omega_n \in \{H, T\}\}$. The CRR model assumes the following:

- A riskless bond and a stock is traded in a finite set of times $\mathbb{T} = \{0, 1, ..., N\}$.
- The bond price at time step n is $B_n = (1 + r')^n B_0$ with $B_0 = 1$ (the rate of discrete compounding $r' > -1$).
- The stock price is modelled by $S_{n+1}(\omega) = \begin{cases} uS_n(\omega), & \omega_{n+1} = H \\ dS_n(\omega), & \omega_{n+1} = T \end{cases}$, with known S_0.

The CRR model has the following results:

- The market (B, S) is complete if and only if $d < 1+r' < u$. A CRR model with $d < 1+r' < u$ is said to be viable.
- A viable CRR model admits a unique fair price $V_n = (1 + r')^{n-N}\mathbb{E}^{\mathbb{Q}}[V_N \mid \mathcal{F}_n]$ at time step n for any derivative with payoff V_N at time step N, where $\mathbb{Q}$ is the unique equivalent martingale measure (risk-neutral measure).
- The one-step conditional risk-neutral probabilities are the same for a viable CRR model, given by $(q, 1-q) = \left(\frac{(1+r')-d}{u-d}, \frac{u-(1+r')}{u-d}\right)$. The risk-neutral measure is given by

$$\mathbb{Q}\left[\left\{\omega : S_N(\omega) = S_0 u^k d^{N-k}\right\}\right] = \binom{N}{k} q^k (1-q)^{N-k}.$$

For a CRR model with parameter choice:

$$u = e^{(r-\frac{1}{2}\sigma^2)\Delta t+\sigma\sqrt{\Delta t}}, d = e^{(r-\frac{1}{2}\sigma^2)\Delta t-\sigma\sqrt{\Delta t}}, r' = e^{r\Delta t} - 1, q = \frac{(1+r')-d}{u-d} = \frac{e^{\frac{1}{2}\sigma^2\Delta t} - e^{-\sigma\sqrt{\Delta t}}}{e^{\sigma\sqrt{\Delta t}} - e^{-\sigma\sqrt{\Delta t}}},$$

where $\Delta t = \frac{T}{N}$, T is the option, N is the number of CRR time steps, r' and r is the rate of discrete/continuous compounding. As the number of time steps increases, we have

$$S_{n+1}(\omega) = S_n e^{(r-\frac{1}{2}\sigma^2)\Delta t\pm\sqrt{\Delta t}},$$

converging to log normal distribution. For an option with a bounded payoff function g,

$$\lim_{N\to\infty}(1+r')^{-N}\sum_{k=0}^{N}q^k(1-q)^{N-k}g(S_N)=e^{-rT}\int_{-\infty}^{\infty}\frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}z^2}g(S_0\exp(r-\tfrac{1}{2}\sigma^2)T+\sigma\sqrt{T}z)dz,$$

i.e. the CRR model converges to the Black-Scholes in the sense that CRR option price formula converges to the general Black-Scholes formula.

For American options, the option value at time step n is given by backward induction incorporating the continuation value (discounted expected payoff) and the early-exercise payoff:

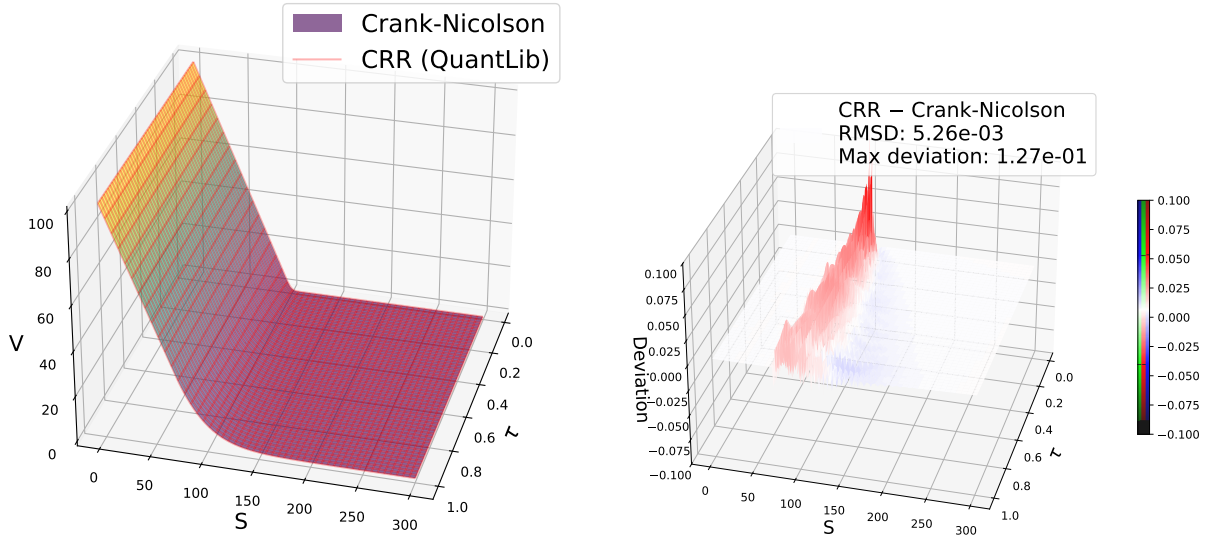
$$V_n=\max\left(\mathbb{E}_n^{\mathbb{Q}}\left[\frac{V_{n+1}}{1+r'}\right],\ g(S_n)\right),$$

where $g(S_n)$ is the intrinsic value of the option at time step n and $V_N=g(S_N)$.

PDE Approach

The variational formulation of American option pricing problem maps the free boundary problem to a discrete form, which can be solved by finite difference methods. The Crank-Nicolson scheme (3.10) can be modified such that the solution at every time step is equal to or greater than the intrinsic value of the option, $g(S)$.

An alternative is the penalty method, deriving from the viscosity solution approach. The idea is to add a penalty term $f(V)$ to the Black-Scholes PDE to remove the free boundary, producing a solution satisfying the constraint $V(t,S)\geq g(S)$. An example penalty function is $f(V)=\frac{1}{\epsilon}(g(S)-V)^+$, where ϵ is a small constant to be tuned [Duffy, 2006].



(a) American put option value: Crank-Nicolson (colored surface) and CRR (red mesh) solutions.

(b) Pointwise deviation between Crank-Nicolson and CRR solutions.

Figure 3.1 Comparison of Crank-Nicolson and CRR solutions for American put option pricing.

Instead of a theoretical proof, we numerically verify the correctness of the modified Crank-Nicolson scheme using the QuantLib implementation of the CRR model [The QuantLib contributors, 2021]. We price an American put option with the following parameters:

$$K = 100, \sigma = 0.3, r = 0.05, T = 1, S_{\text{inf}} = 3 \cdot K,$$

where K is the strike price, σ is the volatility, r is the risk-free rate, T is the expiry and S_{inf} is the upper bound of the truncated domain. The results for the modified Crank-Nicolson scheme with a 100×100 grid and QuantLib CRR with 200 steps are interpolated on a 1000×1000 equispaced uniform grid on the option domain, presented in Figure 3.1. The Crank-Nicolson and CRR solutions are highly consistent, with minor deviations near the strike price $K = 100$.

3.3 Greeks

Greeks are partial derivatives of the price of a financial derivative instrument w.r.t. one or more underlying parameters, which measure the sensitivity of the price to changes in underlying parameters.

Below lists some of the most important greeks used in practice when hedging derivatives:

- Delta, $\Delta = \frac{\partial V}{\partial S}$, measures how much the option price V changes when there is a small change in the underlying asset price S;
- Theta (time decay), $\Theta = -\frac{\partial V}{\partial \tau}$, is the negative sensitivity to time to expiry;
- Gamma, $\Gamma = \frac{\partial \Delta}{\partial S} = \frac{\partial^2 V}{\partial S^2}$, is the sensitivity of delta to the underlying asset price.

Formulae for greeks of European options under the Black-Scholes model are available:

- $\Delta^{\text{BS}}_{\text{CALL}} = \Phi(d_1), \quad \Delta^{\text{BS}}_{\text{PUT}} = -\Phi(-d_1),$
- $\Theta^{\text{BS}}_{\text{CALL}} = -\frac{S\varphi(d_1)\sigma}{2\sqrt{\tau}} - rKe^{-r\tau}\Phi(d_2), \quad \Theta^{\text{BS}}_{\text{PUT}} = -\frac{S\varphi(d_1)\sigma}{2\sqrt{\tau}} + rKe^{-r\tau}\Phi(-d_2),$
- $\Gamma^{\text{BS}}_{\text{CALL}} = \Gamma^{\text{BS}}_{\text{PUT}} = \frac{\varphi(d_1)}{S\sigma\sqrt{\tau}},$

where Φ is the standard normal cumulative distribution function, φ is the standard normal probability density function, $d_{1,2}(t) = \frac{\ln(S_t/K) + (r \pm \sigma^2/2)(T-t)}{\sigma\sqrt{T-t}}$.

Chapter 4

Option Pricing Using Physics-Informed Neural Networks

4.1 European Options

The Black-Scholes PDE for option pricing is derived in Section 3.1. A change of variable $\tau = T - t$ is applied to convert the terminal condition to the initial condition. Here, τ represents the time to expiry and t is the instantaneous time.

The initial condition is the option payoff: $g(S) = (S - K)^+$ for European call options and $g(S) = (K - S)^+$ for European put options. Consider boundary conditions when the stock price is zero and goes to infinity. Since the stock price is modelled by the geometric Brownian motion, zero is an absorbing boundary, the stock price at expiry is guaranteed to be zero if the stock price reaches zero at some time. In this case, the option price is the discounted final payoff with zero stock price. When the stock price goes to infinity, options are infinitely deep in/out of the money. Hence, the option value is infinity for calls and zero for puts. In practice, we use a reasonably large number S_{inf} to cap the unbounded domain. To summarise, the boundary conditions are:

$$\begin{cases} V_{\text{CALL}}(\tau, 0) = 0, \\ V_{\text{CALL}}(\tau, S_{\text{inf}}) \approx S_{\text{inf}} - Ke^{-r\tau}, \end{cases} \qquad \begin{cases} V_{\text{PUT}}(\tau, 0) = Ke^{-r\tau}, \\ V_{\text{PUT}}(\tau, S_{\text{inf}}) = 0. \end{cases}$$

Note that the approximation in the truncated boundary in the call option case $V_{\text{CALL}}(\tau, S_{\text{inf}}) \approx S_{\text{inf}} - Ke^{-r\tau}$ can result in large error in PINN solutions near S_{inf}. To mitigate this issue, one needs to use very large S_{inf}. It is more advisable to train PINN solutions for put options and then obtain call option prices using the put-call parity.

As an example, we implement a PINN solution for the European put option with the following parameters:

$$K = 100,\ \sigma = 0.3,\ r = 0.05,\ T = 1,\ S_{\text{inf}} = 3 \cdot K,$$

where K is the strike price, σ is the volatility, r is the risk-free rate, T is the expiry and S_{inf} is the

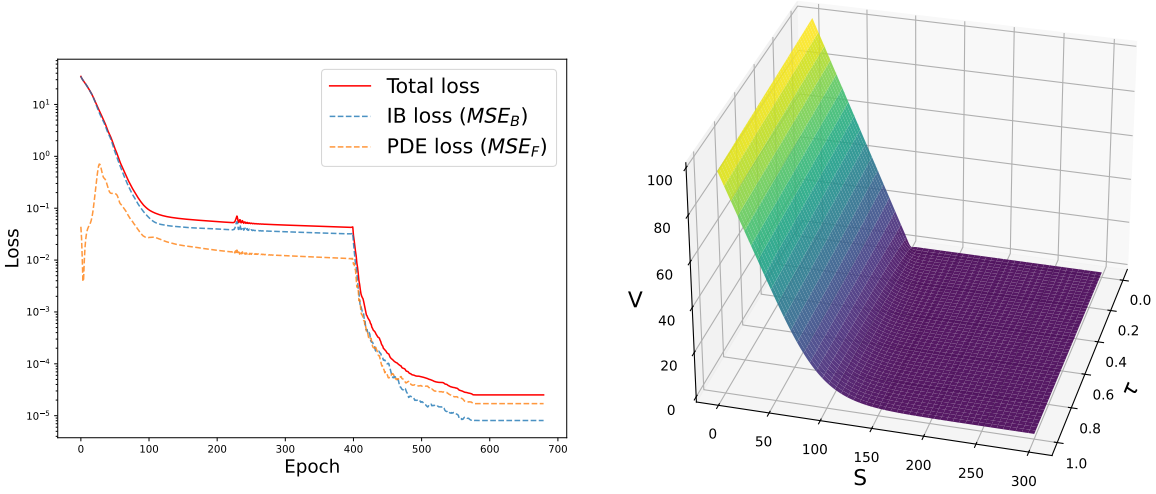
upper bound of the truncated domain. The detailed description of the PINN framework setting for European put options has been given in Section 2.4.

The training data comprise 100 initial data points, 100 boundary points, and 2000 collocation points ($N_{\mathcal{B}} = 100 + 100$, $N_{\mathcal{F}} = 2000$, $N_{\text{data}} = 0$). The data points are generated using Sobol sequences for better coverage of the option domain as mentioned in Section 2.3.1. Apart from the initial and boundary points, no observed data are included in the training set. As we will see in later results, the PINN learns the underlying Black-Scholes dynamics with only a small number of initial and boundary points and a relatively large number of collocation points.

For the NN architecture, an MLP with 4 hidden layers each with 10 neurons followed by a hyperbolic tangent activation is used. This MLP has 371 parameters and is a relatively small architecture. This choice of architecture has been empirically experimented to produce a desired level of RMSE on the evaluation set. Details on NN architecture selection will be discussed in Section 4.1.3.

As a common practice in deep learning, normalisation is essential for model stability. The asset price range is scaled by $\frac{K}{10}$: $\tilde{S} = 10 \cdot \frac{S}{K}, \tilde{V} = 10 \cdot \frac{V}{K}$. The ratio $\frac{S}{K}$ is a common measure of the option moneyness. Under the Black-Scholes model, the option pricing PDE is invariant (up to a constant $\frac{K}{10}$) under this transformation. Thus, the PINN is theoretically able to price for any strike price once trained, i.e. the PINN solution is essentially an approximator $\hat{V}_\theta(\tau, S, K)$.

The PINN is first optimised using Adam and then by L-BFGS. It can be observed in the loss history 4.1a that the two loss components ($\text{MSE}_{\mathcal{B}}$ and $\text{MSE}_{\mathcal{F}}$) is stabilising after 100 epochs of Adam optimisation. The loss curve exhibits a significant descent when switching to L-BFGS at the 400th epoch. As discussed in Section 2.3.2, L-BFGS is a second-order method providing curvature information whereas Adam uses only gradient information. The resulting PINN solution for this European put option is shown in Figure 4.1b.



(a) Loss history: total loss, initial and boundary loss ($\text{MSE}_{\mathcal{B}}$), and PDE loss ($\text{MSE}_{\mathcal{F}}$).

(b) PINN solution surface.

Figure 4.1 European put option PINN solution.

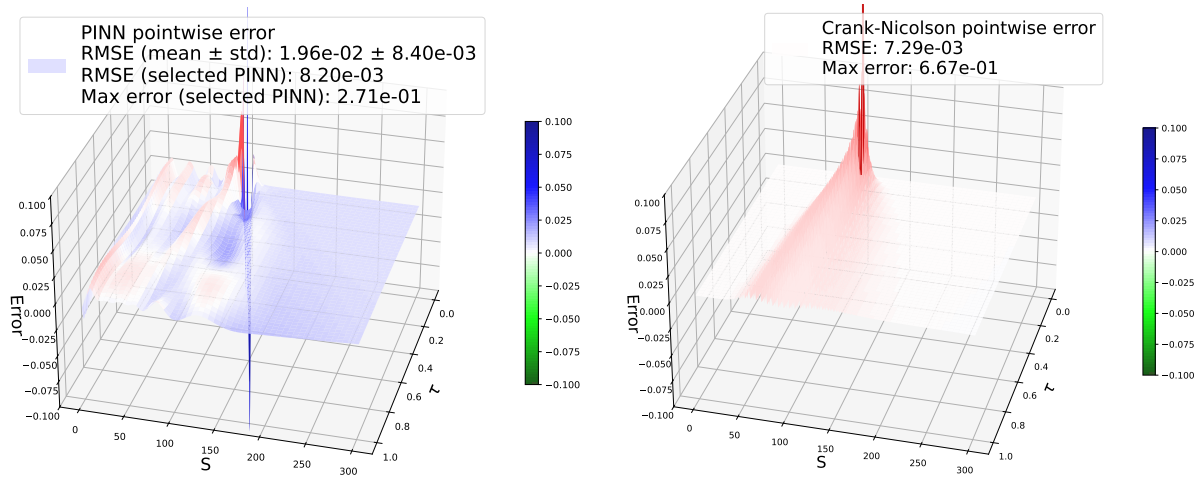
Remark. The training process is repeated 100 times with different random states. The presented figures correspond to the empirically best-performing PINN in terms of RMSE on evaluation set.

4.1.1 PINN Solution Evaluation and Comparison with Crank-Nicolson Method

The root mean squared error (RMSE) is used as an evaluation metric as discussed in Section 2.5.1. The RMSE is evaluated on an equispaced uniform grid of size 1000×1000 using the Black-Scholes formula (3.6).

In addition to the analytical solution, the PINN solution is compared to the Crank-Nicolson method illustrated in Section 3.1.2. The Crank-Nicolson method is a finite difference method for solving heat equations and similar PDEs, which is second-order in time. A Crank-Nicholson scheme with a 100×100 grid on the same domain as the PINN is linearly interpolated on the same evaluation set.

To ensure the robustness of results, the training process is repeated 100 times to calculate the mean and the standard deviation of RMSE. The trained model with the smallest RMSE is selected to be the empirically best-performing PINN solution.



(a) PINN error surface.

(b) Crank-Nicolson error surface.

Figure 4.2 European put option PINN and Crank-Nicolson solution pointwise absolute error.

The pointwise absolute error surfaces for the empirically best-performing PINN and Crank-Nicolson solutions are given in Figure 4.2. The mean of RMSE for PINN is 1.96×10^{-2} with standard deviation 8.40×10^{-3}. The RMSE for the empirically best-performing PINN is 8.20×10^{-3} (3 s.f.), very close to 7.29×10^{-3} (3 s.f.) for Crank-Nicolson. The maximum pointwise error magnitude is 0.271 (3 s.f.) for PINN and 0.667 (3 s.f.) for Crank-Nicolson.

Both solutions show increasing pointwise error near the strike price ($S = 4$) with time approaching the expiry ($\tau = 0$), where the maximum pointwise error occurs. The option value surface along the strike price gradually transits from smooth to non-differentiable as the time to expiry decreases. The option value at expiry is not differentiable at the strike price. As it is difficult for neural networks with smooth activation functions to capture non-differentiability,

the increasing pointwise error in this region makes sense.

It is also noticeable that the PINN pointwise error along the boundary $S = 0$ is higher. The boundary is a tricky area for PINN to learn and only 100 boundary points are fed into the PINN. There are several potential methods to improve PINN performance near the boundary. One can add sampling weight to collocation points near the boundary or increase the weight for the boundary loss term. In contrast, Crank-Nicolson does not suffer along the boundary as the boundary condition is directly encoded into the Crank-Nicolson solution grid.

In this scenario, the PINN performance is comparable to the Crank-Nicolson method with a 100×100 grid size in terms of RMSE with even smaller maximum pointwise error, though Crank-Nicolson gives less oscillatory pointwise error distribution and does not suffer near the boundary. One should also be aware that this is not a perfectly fair comparison. The PINN learns the PDE dynamics with only 200 observed initial and boundary data, whereas everything is hard-coded for Crank-Nicolson.

4.1.2 Greeks Computation

One of the advantages of the PINN method is the solution differentiability. This enables the calculation of the option greeks with little extra effort using automatic differentiation. To further access how well the PINN solution captures the option dynamics, the option greeks (delta, theta, and gamma) are evaluated and compared to analytical results in Section 3.3. The PINN greeks value and pointwise absolute error surfaces are given in Figure 4.3.

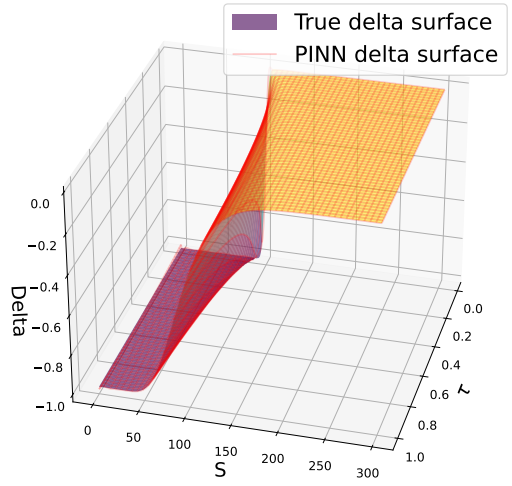
The PINN solution approximates the greeks well, with RMSE at the same level of the option value RMSE (except theta due to numerical stability). Note that in Figure 4.3d and 4.3f, the error near the expiry ($\tau = 0$) and the strike price ($S = 4$) is high, as theta and gamma are numerically unstable in these areas.

Remark. In our PINN implementation, the neural network takes input (τ, S) and treats other option parameters as constants. Hence, we are only able to differentiate w.r.t. τ and S. To enable calculation of other greeks, e.g. vega, $\mathcal{V} = \frac{\partial V}{\partial \sigma}$, one needs to make σ an explicit variable in the NN and then input constant values for σ during training. However, this would increase the complexity of the optimisation and may require more complex architecture to achieve the same level of accuracy.

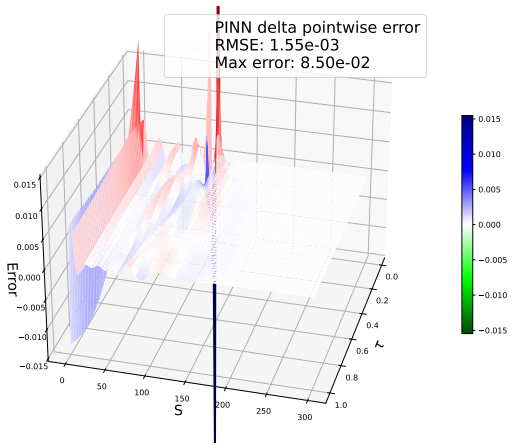
4.1.3 PINN Architecture Selection

The goal is to empirically find a light PINN architecture matching the RMSE for Crank-Nicolson scheme with a 100×100 grid size. PINN solutions with different NN architectures are trained multiple times to calculate the mean and the standard deviation of RMSE. The empirical results are presented in Table 4.1.

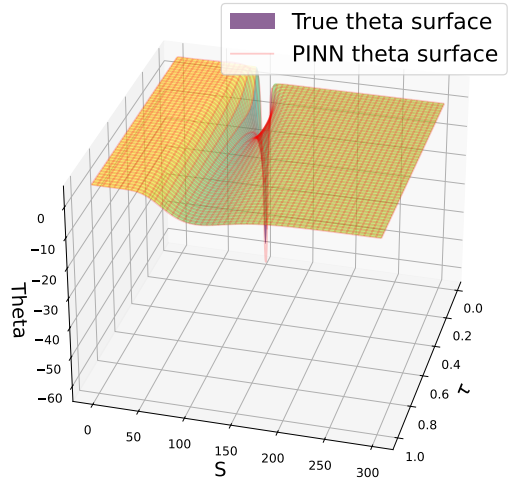
For convenience, we denote a PINN solution using an MLP consisting of L hidden layers with n_l neurons for l-th hidden layer followed by an activation function σ trained using collocation sampling method M as $\mathrm{MLP}(n_1, n_2, \ldots, n_l)\sigma$_M. For example, $\mathrm{MLP}(10, 10, 10)\mathrm{tanh}$_sobol is a



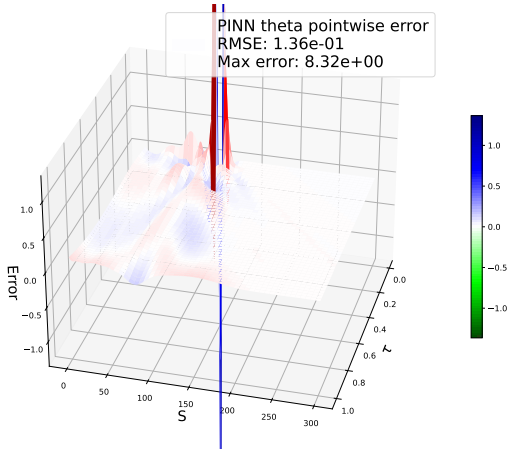
(a) Delta surface.



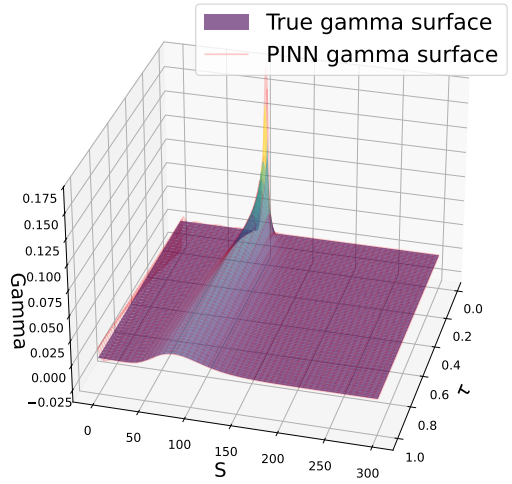
(b) Delta error surface.



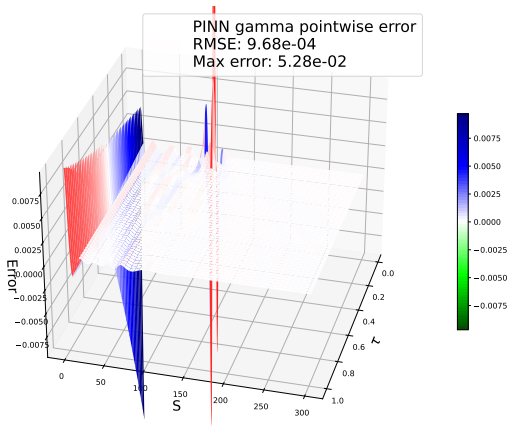
(c) Theta surface.



(d) Theta error surface.



(e) Gamma surface.



(f) Gamma error surface.

Figure 4.3 European put option PINN greeks value and pointwise absolute error surfaces.

PINN solution using an MLP with 3 hidden layers each with 10 neurons followed by a $\tanh$ activation, trained using Sobol sequence collocation points sampling.

PINN architecture	# parameters	RMSE mean	RMSE std	RMSE min
$\mathrm{MLP}(5,5)\tanh\_\mathrm{sobol}$	51	1.00×10^{-1}	4.19×10^{-2}	4.45×10^{-2}
$\mathrm{MLP}(10,10)\tanh\_\mathrm{sobol}$	151	2.39×10^{-2}	6.72×10^{-3}	1.33×10^{-2}
$\mathrm{MLP}(10,10)\tanh\_\mathrm{uniform}$	151	6.63×10^{-2}	1.89×10^{-2}	3.69×10^{-2}
$\mathrm{MLP}(10,10)\mathrm{SiLU}\_\mathrm{sobol}$	151	1.22×10^{-1}	8.14×10^{-2}	1.67×10^{-2}
$\mathrm{MLP}(10,10)\mathrm{GELU}\_\mathrm{sobol}$	151	1.30×10^{-1}	9.54×10^{-2}	2.48×10^{-2}
$\mathrm{MLP}(10,10)\mathrm{ReLU}\_\mathrm{sobol}$	151	1.90×10^{0}	7.80×10^{-2}	1.84×10^{0}
$\mathrm{MLP}(10,10,10)\tanh\_\mathrm{sobol}$	261	1.76×10^{-2}	6.54×10^{-3}	1.04×10^{-2}
$\mathrm{MLP}(10,10,10,10)\tanh\_\mathrm{sobol}$	371	1.96×10^{-2}	8.40×10^{-3}	8.20×10^{-3}
$\mathrm{MLP}(40,20,10,5)\tanh\_\mathrm{sobol}$	1211	1.47×10^{-2}	4.28×10^{-3}	7.89×10^{-3}

Table 4.1 Empirical results for various PINN architectures: the mean and the standard deviation of RMSE (RMSE mean and RMSE std), and the smallest RMSE achieved (RMSE min).

Empirical results suggest that increasing the depth and width of the NN architecture generally reduces the mean and standard deviation of RMSE, indicating enhanced performance and stability. However, the performance improvement is marginal to the number of parameters (computational cost). Notably, light PINN architectures with as few as 51 parameters are able to produce 10^{-2} level RMSE for the basic European case. For the activation function, $\tanh$ outperforms SiLU (Sigmoid Linear Unit), GELU (Gaussian Error Linear Unit), and ReLU (Rectified Linear Unit). In particular, smooth activation functions produce superior performance compared to ReLU (non-differentiable at zero). For the collocation points sampling method, Sobol sequences with better domain coverage improve the solution quality compared to uniform sampling, as expected.

Therefore, the PINN architecture $\mathrm{MLP}(10,10,10,10)\tanh\_\mathrm{sobol}$ is selected for the European put option as a good trade-off between performance and computational cost.

4.2 American Options

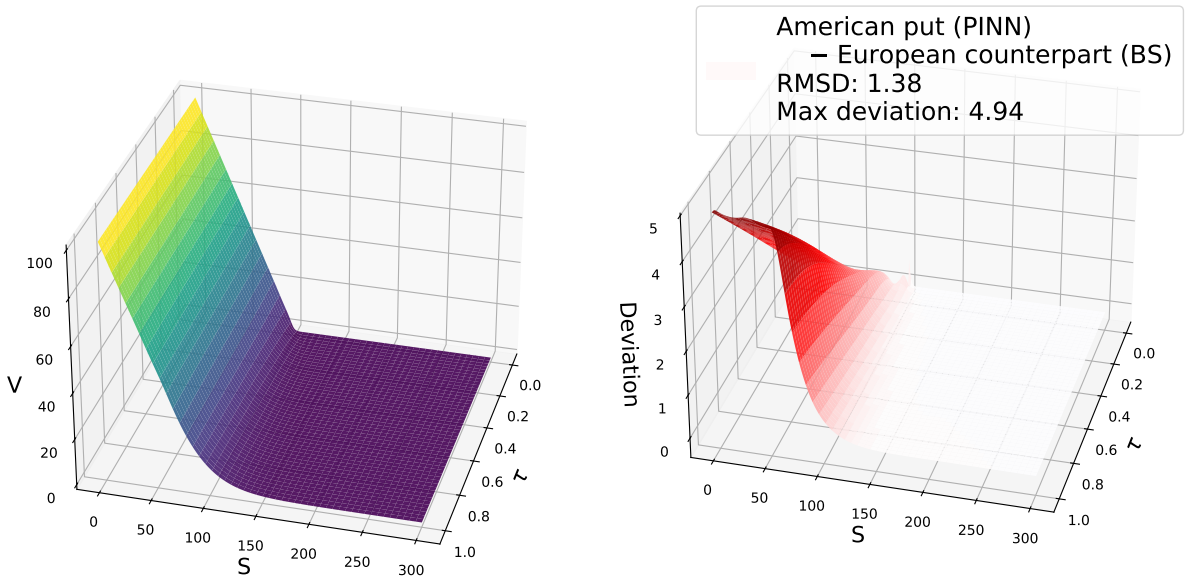
We adopt the variational formulation (Theorem 3.2.2, under change of variable $\tau = T - t$). Let $\mathcal{L} = \frac{\partial}{\partial \tau} - rS\frac{\partial}{\partial S} - \frac{1}{2}\sigma^2 S^2 \frac{\partial^2}{\partial S^2} + r$ be the differential operator for the Black-Scholes PDE, we have the governing equation with initial condition:

$$\begin{aligned} \min\{\mathcal{L}V(\tau,S), V(\tau,S) - g(S)\} &= 0, \\ V(0,S) &= g(S), \end{aligned}$$

where g is the option payoff function. With the early-exercise feature, the boundary conditions are without discounting factors:

$$\begin{cases} V_{\text{CALL}}(\tau, 0) = 0, \\ V_{\text{CALL}}(\tau, S_{\text{inf}}) \approx S_{\text{inf}} - K, \end{cases} \qquad \begin{cases} V_{\text{PUT}}(\tau, 0) = K, \\ V_{\text{PUT}}(\tau, S_{\text{inf}}) = 0. \end{cases}$$

A PINN solution for the American counterpart of Section 4.1 is implemented, i.e. an American put option with parameters: $K = 100$, $\sigma = 0.3$, $r = 0.05$, $T = 1$, $S_{\text{inf}} = 3 \cdot K$. The same setup for training data is used: 100 initial data points, 100 boundary points, and 2000 collocation points ($N_{\mathcal{B}} = 100 + 100$, $N_{\mathcal{F}} = 2000$, $N_{\text{data}} = 0$), generated by Sobol sequences. The same NN architecture in the European case (4 hidden layer MLP) is applied here.



(a) American put PINN solution.

(b) Pointwise deviation between American put PINN and European counterpart (BS formula).

Figure 4.4 American put option PINN and difference to European counterpart.

The PINN solution for the American put option is shown in Figure 4.4a. Figure 4.4b shows the difference in value between the American put option (PINN solution) and its European counterpart (Black-Scholes formula). An American put option is at least as expensive as its European counterpart due to the early-exercise feature.

Remark. The training process is repeated 100 times with different random states. The presented figures correspond to the empirically best-performing PINN in terms of RMSE on evaluation set.

4.2.1 PINN Solution Evaluation and Comparison with Conventional Methods

In absence of ground truth, the root mean squared deviation (RMSD) is used to measure the similarity between the PINN solution and numerical solutions. As in the European case, the training process is repeated 100 times to calculate the mean and the standard deviation of RMSD. The trained model with the smallest RMSD is selected to be the empirically best-

performing PINN solution.

The pointwise deviation of the PINN solution from the modified Crank-Nicolson solution (evaluated on a 1000×1000 equispaced uniform grid) is shown in Figure 4.5. As seen earlier in Section 3.2.1, the Crank-Nicolson and the CRR solutions are highly consistent with 10^{-3} level RMSD. Hence, this section only presents a comparison with the Crank-Nicolson solution here. Overall, the PINN solution is very close to the Crank-Nicolson solution, with 3.29×10^{-2} RMSD. The error distribution in the American case is more oscillatory with higher maximum pointwise error compared to the European case, particularly in the *in-the-money* region of the option (an option is in-the-money if it has positive value).

This example of an American put option written on a non-dividend-paying stock illustrates the potential of the PINN method as a good alternative to conventional PDE-based and tree-based methods for options with early-exercise feature. The PINN method handles early exercise effortlessly through the PDE formulation.

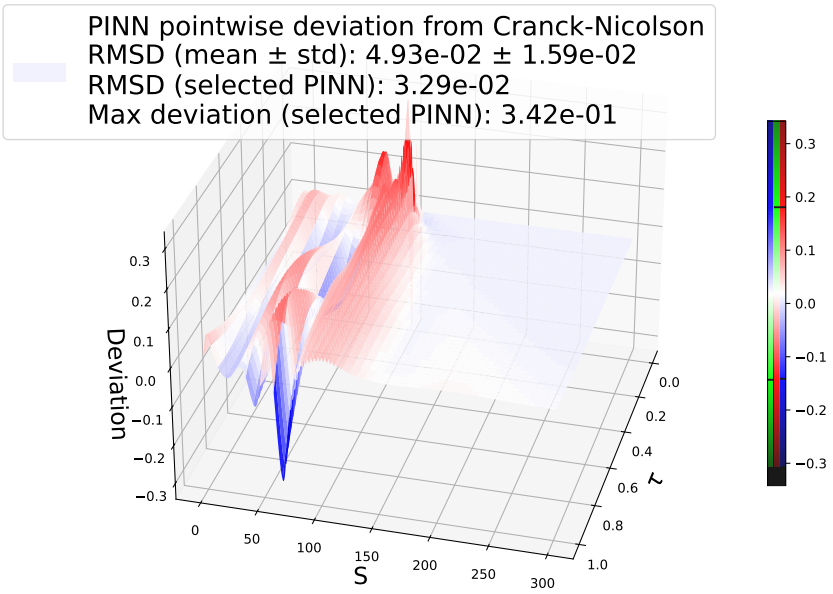


Figure 4.5 American put PINN solution error surface w.r.t Crank-Nicolson solution

4.3 KAN-Based PINNs

We implement KAN-based PINNs for vanilla options covered in Section 4.1 and 4.2. Our KAN-based PINNs are based on the code from the original KAN paper [Liu et al., 2025]. Due to limited optimisation and support for existing KAN implementations, our KAN-based PINNs are not fully optimised. The goal is to match (rather than outperform) MLP-based PINN performance with empirically minimal KAN architecture.

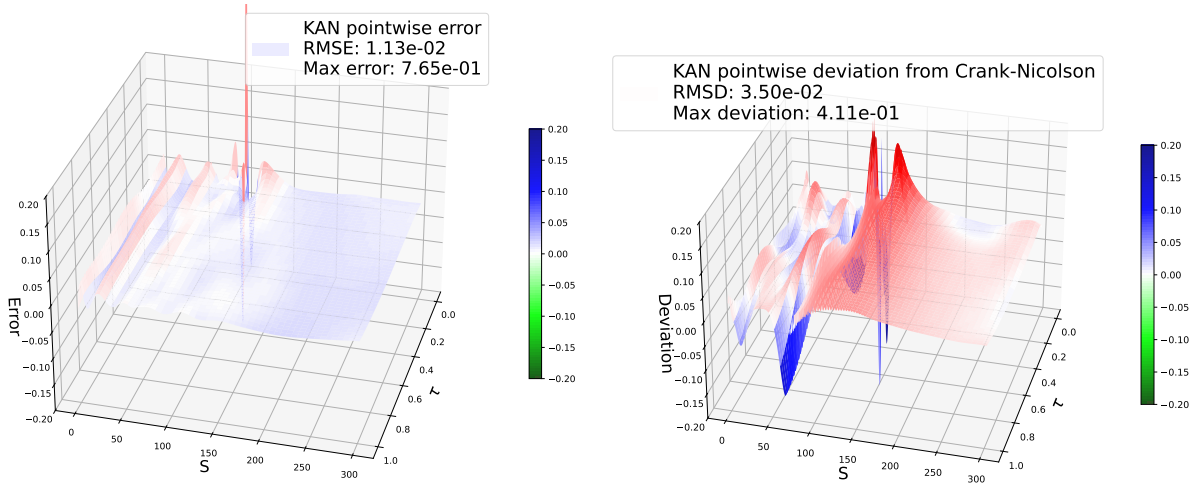
Following notation in Section 4.1.3 ($\mathrm{MLP}(10, 10, 10)\tanh$_sobol is an MLP-based PINN with 3 hidden layers each with 10 neurons followed by a $\tanh$ activation, trained using Sobol sequence collocation points sampling), $\mathrm{KAN}(8, 4, 2)$_spline3_grid16_sobol is a KAN-based PINN

with 3 hidden layers with 8, 4, 2 neurons, using cubic splines (order 3) with a grid size 16, trained with Sobol sequence collocation points sampling.

$\mathrm{KAN}(8,4,2)$_spline3_grid16_sobol is applied to price the same European and American put options. The results are presented in Figure 4.6. $\mathrm{KAN}(8,4,2)$_spline3_grid16_sobol (1218 parameters) falls in the average performance range of $\mathrm{MLP}(40,20,10,5)\tanh$_sobol (1211 parameters) in the European case, and closely matches $\mathrm{MLP}(10,10,10,10)\tanh$_sobol (371 parameters) in the American case (more complex solutions not implemented). Note that the parameter count of KANs and MLPs is not exactly comparable, as KANs use B-splines to parametrise activation functions. We conclude that the KAN-based PINN achieves the same level of performance with a more compact layer structure.

Here, we aim to produce comparable performance to MLP-based PINN with empirically minimal KAN architecture. Section 4.1.3 suggests that the MLP-based PINN performance improvement is marginal to the number of parameters, $\mathrm{MLP}(40,20,10,5)\tanh$_sobol (1211 parameters) improves less than 4% in the best case compared to $\mathrm{MLP}(10,10,10,10)\tanh$_sobol (371 parameters). In contrast, KANs can easily match MLP performance by using a similar or even more compact architecture. It is fairly easy for KANs to outperform MLPs by using more complex layers and finer grids. However, the optimal performance for KAN-based PINNs is left unexplored due to limited optimisation and support for existing KAN implementations.

Overall, KAN-based PINNs are promising for superior expressiveness inherited from splines and faster neural scaling laws.



(a) European put KAN-based PINN error surface. **(b)** American put KAN-based PINN error surface.

Figure 4.6 KAN-based PINN for European and American put options.

Chapter 5

Conclusion

PINNs offer a flexible, unified framework for option pricing through PDE formulation, producing continuously differentiable solutions across the entire option domain. As seen in Chapter 4, the PINN method is able to produce solutions below 10^{-2} level RMSE with light MLP architectures. (All MLP-based PINNs can be trained within 90 seconds using MacBook M3 Pro on CPU.) Once trained, the PINN solution is able to give price predictions on the entire domain within milliseconds, and allows option greeks computation at little computational cost. In addition, the PINN method easily handles the early-exercise feature of American-style options via the variational formulation. Compared to conventional numerical methods, PINNs can be easily implemented and flexibly adapted to different option pricing PDE models. Overall, the PINN method offers an alternative option pricing framework, with capabilities of greeks computation and easy handling of early-exercise and (potentially) path-dependent features, at the trade-off of solution accuracy.

5.1 Limitations and Further Work

Although the PINN method is theoretically applicable to arbitrary model dynamics in PDE formulation, there is currently no theoretical guarantee of convergence and stability, due to the limited development in PINN learning theory.

The management of boundary conditions remains an active topic of research in PINNs, with variations such as hard constraints being proposed. PINN solutions exhibit increasing errors along boundaries and critical regions (at-the-money, near expiry, and early-exercise boundary).

In greeks computation, delta, theta, and gamma are well captured with numerical instabilities in some regions. As noted in Section 4.1.2 Remark, our current PINN implementation takes input (τ, S) and treats other option parameters as constants. To enable the greeks computation involving other parameters, one needs to make other parameters as explicit variables in the NN (set to constant values during training). Complex architectures and optimisation techniques are required to achieve desired solution accuracy. Due to the time constraint, this direction is left for future work.

Section 4.2 demonstrates the capability of the PINN method to handle early exercise through an American put option written on a non-dividend-paying stock. Due to the time constraint, the handling of path dependence is left unexplored. The idea is to introduce an auxiliary process, for example, the cumulative sum of the underlying asset price, $I(t) = \int_0^t S(u)\mathrm{d}u$. The Asian option price is a function of (t, S, I).

The optimal PINN RMSE performance is left unexplored. Section 4.1.3 suggests that the MLP-based PINN performance improvement is getting marginal as the number of parameters (width and depth) increases, indicating a performance limit. On the other hand, KAN-based PINNs can easily match MLP performance by using similar or even more compact architectures, which suggests the potential to improve the MLP-based PINN performance limit.

Appendix A

Splines

This section introduces splines based on Hastie et al. [2001].

Given feature vector $X \in \mathbb{R}^p$, denote by $h_m(X) : \mathbb{R}^p \to \mathbb{R}$ the mth transformation of X, $m = 1, \ldots, M$. The *linear basis expansion* in X is $f(X) = \sum_{m=1}^{M} \beta_m h_m(X)$. This approach moves beyond linearity, while the models are still linear in basis functions (hence fitting proceeds as in linear models). The basis expansions derive families of *piecewise-polynomials*, *splines*, and *wavelets*, producing a *dictionary* $\mathcal{D}$ consisting of a very large number of basis functions. Examples of basis functions are:

- $h_m(X) = X_m$, $m = 1, \ldots, p$ recovers the linear model $f(X) = \mathrm{E}[Y \mid X]$.
- $h_m(X) = \mathbb{1}_{\{L_m \leq X_k \leq U_m\}}$ results in a piecewise constant model (regional means). A piecewise linear model can be obtained by adding basis functions of the form $h_m(X)X$.

In addition to continuity restriction at knots, smoothness can be achieved by increasing the order of local polynomials. A *spline* is a function defined piecewise by polynomials. The mathematical definition is given as follows:

Definition A.0.1 (Splines)**.** An order-M spline with knots ξ_j, $j = 1, \ldots, K$ is a piecewise-polynomial of order M, and has continuous derivatives up to order $M - 2$. This spline can be represented in the truncated-power basis:

$$h_j(X) = X^{j-1}, \quad j = 1, \ldots, M,$$
$$h_{M+\ell}(X) = (X - \xi_\ell)_+^{M-1}, \quad \ell = 1, \ldots, K,$$

where $(\cdot)_+ = \max(\cdot, 0)$.

The spline functions for a given order and knot sequence form a vector space. Hence, there exist equivalent bases other than the truncated-power basis in Definition A.0.1. A *B-spline* is a spline function constructed with minimal support for a given order and knot sequence, which acts as a numerically stable and computationally efficient basis.

Appendix B

Sobol Sequences

The *discrepancy* of a set of vectors $P = \left\{\boldsymbol{x}_i \in \mathbb{R}^d\right\}_{1 \leq i \leq N}$ is a measure of inhomogeneity of distribution in the unit hypercube. Pseudo-randomness seeks to approximate true randomness in statistical properties, while quasi-randomness deliberately avoids randomness to ensure even coverage. Consider a set of N uniform samples from P and a sub-hypercube delimiting the upper right corner of the hyper-rectangular domain from $\mathbf{0}$ to $\boldsymbol{y}$, $S(\boldsymbol{y}) = \prod_{i=1}^{d}[0, y_i)$. Let $n_{S(\boldsymbol{y})} = \sum_{i=1}^{N} \mathbb{1}_{\{\boldsymbol{x}_i \in S(\boldsymbol{y})\}}$ denote the number of samples in $S(\boldsymbol{y})$. The discrepancy of P is defined as $D(P) = \sup_{\boldsymbol{y} \in [\mathbf{0},\mathbf{1}]^{\boldsymbol{d}}} \left| \frac{n_{S(\boldsymbol{y})}}{N} - \prod_{k=1}^{d} y_k \right|$. In particular, low discrepancy indicates even and uniform coverage of space [Jäckel, 2002].

Low-discrepancy sequences, or *quasi-random sequences*, are deterministic sequences constructed to achieve uniform coverage of a space. Low-discrepancy sequences are used in quasi-Monte Carlo method for numerical integration to achieve convergence rate of $\mathcal{O}(1/N)$ compared to $\mathcal{O}(1/\sqrt{N})$ for standard Monte Carlo method [Asmussen and Glynn, 2007]. *Sobol sequences* [Sobol, 1967] are a type of low-discrepancy sequence, constructed using a base of two to form successively finer uniform partitions of the unit interval.

Remark. Though Sobol sequences (low-discrepancy sequences) mostly appear in quasi-Monte Carlo, this work uses them in PDE collocation points sampling for better domain coverage.

Bibliography

S. Asmussen and P. W. Glynn. *Stochastic simulation: algorithms and analysis*, volume 57. Springer, 2007.

M. Barahona and B. Bravi. Methods for data science, 2024. Lecture Notes, Imperial College London.

A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind. Automatic differentiation in machine learning: a survey. *Journal of Machine Learning Research*, 18(153):1–43, 2018.

F. Black and M. Scholes. The pricing of options and corporate liabilities. *Journal of Political Economy*, 81(3):637–654, 1973.

D. Brigo. Stochastic differential equations in financial modelling, 2024. Lecture Notes, Imperial College London.

S. Cuomo, V. S. D. Cola, F. Giampaolo, et al. Scientific machine learning through physics-informed neural networks: Where we are and what's next. *Journal of Scientific Computing*, 92:88, 2022. doi: 10.1007/s10915-022-01939-z.

N. J. Cutland and A. Roux. *The Cox-Ross-Rubinstein Model*, pages 177–209. Springer London, London, 2013.

J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(7), 2011.

D. J. Duffy. *Viscosity Solutions and Penalty Methods for American Option Problems*, chapter 28, pages 307–314. John Wiley Sons, Ltd, 2006.

P. S. Hagan, D. Kumar, A. S. Lesniewski, and D. E. Woodward. Managing smile risk. *The Best of Wilmott*, 1:249–296, 2002.

T. Hastie, J. Friedman, and R. Tibshirani. *Basis Expansions and Regularization*, pages 115–163. Springer New York, New York, NY, 2001. doi: 10.1007/978-0-387-21606-5_5.

S. L. Heston. A closed-form solution for options with stochastic volatility with applications to bond and currency options. *The Review of Financial Studies*, 6(2):327–343, 1993.

D. J. Higham. *An Introduction to Financial Option Valuation: Mathematics, Stochastics and Computation*. Cambridge University Press, 2004.

K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

P. Jäckel. *Monte Carlo methods in finance*. John Wiley & Sons, 2002.

P. Jaillet, D. Lamberton, and B. Lapeyre. Variational inequalities and the pricing of American options. *Acta Applicandae Mathematica*, 21:263–289, 1990.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2017.

D. C. Liu and J. Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45(1):503–528, 1989.

Z. Liu, Y. Wang, S. Vaidya, F. Ruehle, J. Halverson, M. Soljačić, T. Y. Hou, and M. Tegmark. KAN: Kolmogorov-Arnold networks, 2025. URL https://arxiv.org/abs/2404.19756.

F. A. Longstaff and E. S. Schwartz. Valuing American options by simulation: a simple least-squares approach. *The Review of Financial Studies*, 14(1):113–147, 2001.

L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis. DeepXDE: A deep learning library for solving differential equations. *SIAM Review*, 63(1):208–228, 2021.

M. Musiela and M. Rutkowski. *American Options*, pages 205–228. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005. doi: 10.1007/3-540-26653-4_5.

J. Nocedal and S. Wright. *Numerical Optimization*. Springer New York, 2006.

A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. PyTorch: An imperative style, high-performance deep learning library, 2019. URL https://arxiv.org/abs/1912.01703.

M. Raissi, P. Perdikaris, and G. 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. doi: https://doi.org/10.1016/j.jcp.2018.10.045.

I. M. Sobol. The distribution of points in a cube and the approximate evaluation of integrals. *USSR Computational Mathematics and Mathematical Physics*, 7:86–112, 1967.

The QuantLib contributors. QuantLib: A free/open-source library for quantitative finance. https://github.com/lballabio/QuantLib, 2021. URL https://www.quantlib.org/. Version 1.23.

T. Tieleman and G. Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, 4(2):26–31, 2012.

H. Wang, L. Lu, S. Song, and G. Huang. Learning specialized activation functions for physics-informed neural networks. *arXiv Preprint arXiv:2308.04073*, 2023a.

X. Wang, J. Li, and J. Li. A deep learning based numerical PDE method for option pricing. *Computational Economics*, 62(1):149–164, 2023b.

Y. Wang, J. Sun, J. Bai, C. Anitescu, M. S. Eshaghi, X. Zhuang, T. Rabczuk, and Y. Liu. Kolmogorov–Arnold-Informed neural network: A physics-informed deep learning framework for solving forward and inverse problems based on Kolmogorov–Arnold networks. *Computer Methods in Applied Mechanics and Engineering*, 433:117518, Jan. 2025. doi: 10.1016/j.cma.2024.117518.

C. Wu, M. Zhu, Q. Tan, Y. Kartha, and L. Lu. A comprehensive study of non-adaptive and residual-based adaptive sampling for physics-informed neural networks. *Computer Methods in Applied Mechanics and Engineering*, 403:115671, 2023. doi: https://doi.org/10.1016/j.cma.2022.115671.