CHALMERS

# Neural Network Surrogates for Optimizing Heston Model Calibration

## A Comparative Study on Design Choices for Deep Differential Pricing Surrogates

Bachelor's thesis in Industrial Engineering and Management


ALEX ANDERSSON
LINNEA DAVIDSON
MARKUS ELIASSEN
JACOB HANSÉN
ISAK STRIDH
MÅNS WESTMAN

DEPARTMENT OF TECHNOLOGY MANAGEMENT AND ECONOMICS

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2026
www.chalmers.se


Bachelor's thesis TEKX18-VT26-11A

Bachelor's thesis TEKX18-VT26-11A

# Neural Network Surrogates for Optimizing Heston Model Calibration

## A Comparative Study on Design Choices for Deep Differential Pricing Surrogates

# Neurala nätverk som surrogatmodeller för effektivare kalibrering av Hestonmodellen

## En jämförande studie av designval för Deep Differential-prissättningssurrogat

ALEX ANDERSSON
LINNEA DAVIDSON
JACOB HANSÉN
MARKUS ELIASSEN
ISAK STRIDH
MÅNS WESTMAN

DEPARTMENT OF TECHNOLOGY MANAGEMENT AND ECONOMICS
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2026

Neural Network Surrogates for Optimizing Heston Model Calibration
A Comparative Study on Design Choices for Deep Differential Pricing Surrogates

ALEX ANDERSSON
LINNEA DAVIDSON
MARKUS ELIASSEN
JACOB HANSÉN
ISAK STRIDH
MÅNS WESTMAN

© ALEX ANDERSSON, 2026
© LINNEA DAVIDSON, 2026
© MARKUS ELIASSEN, 2026
© JACOB HANSÉN, 2026
© ISAK STRIDH, 2026
© MÅNS WESTMAN, 2026

Bachelor's thesis TEKX18-VT26-11A
Department of Technology Management and Economics
Chalmers University of Technology
SE-412 96 Gothenburg
Sweden
Telephone + 46 (0)31-772 1000

Gothenburg, Sweden 2026

Neural Network Surrogates for Optimizing Heston Model Calibration
A Comparative Study on Design Choices for Deep Differential Pricing Surrogates

ALEX ANDERSSON
LINNEA DAVIDSON
MARKUS ELIASSEN
JACOB HANSÉN
ISAK STRIDH
MÅNS WESTMAN

Department of Technology Management and Economics
Chalmers University of Technology

SUMMARY

**Problem** Calibrating stochastic volatility models means estimating parameters that allow model prices to match observed market prices. For the Heston model, this process is computationally expensive, as it requires repeated numerical evaluations of a semi-analytical pricing formula. In practical settings where computational speed is critical, this slowdown limits the applicability of traditional calibration methods. As a result, faster yet accurate surrogate models are needed.

**Aim** This study sets out to investigate the extent to which neural network surrogate models can match the accuracy and stability of semi-analytical option pricing, while offering substantial gains in computational speed. Furthermore, the aim is to examine how model design choices affect performance by comparing the models with each other and with the COS method. The purpose of this is to identify trade-offs between approaches and thereby guide the choice of a surrogate model that best meets the requirements of the intended application.

**Theoretical Framework** This thesis draws on Black-Scholes option pricing theory (Black & Scholes, 1973), the Heston (1993) stochastic volatility model, and its semi-analytical pricing via the COS method, developed by Fang & Oosterlee (2008). Surrogate models are trained based on the DML framework of Huge & Savine (2020), where parameter gradients, i.e., Greeks, are incorporated through Sobolev training.

**Method** Synthetic training data was generated by sampling Heston parameters from a broad parameter space using Sobol sequencing. Option prices and parameter Greeks were computed via the COS method, with derivatives obtained through automatic differentiation. Additionally, the corresponding implied volatility (IV) and IV Greeks were computed via the "Let's Be Rational" method. Four surrogate models were then trained: a multilayer perceptron trained on prices (MLP_P), and three differential neural networks trained on prices (DDN_P), prices normalized by strike (DDN_PdivK), and IVs (DDN_IV), respectively. The models were then evaluated on a separate test set. Performance was assessed in terms of pricing accuracy, Greek stability, computational throughput and no-arbitrage compliance, using a COS method benchmark.

**Results and Implications** All neural networks achieved computational speedups of two to three orders of magnitude over the COS benchmark. DDN_IV reached a 764 times higher throughput on the full IV and IV-Greeks pipeline. The DDN_IV network also achieved the lowest pricing and Greek errors, with near-zero arbitrage violations, outperforming all other models. Furthermore, the DDN models produced smoother parameter derivatives than the

COS benchmark, with stable behavior across both typical and extreme parameter combinations. Taken together, these results indicate that DDN_IV, a differential neural network trained on implied volatility, is the most promising surrogate for optimizing Heston model calibration.

Keywords: Heston model, Gradient-based Calibration, Parameter-Greeks, Deep Differential Learning, Neural network surrogate, Implied volatility, Fourier-Cosine (COS) method

Note: This report is written in English.

SAMMANDRAG

**Problem** Kalibrering av stokastiska volatilitetsmodeller innebär att parametrar skattas så att modellpriser stämmer överens med observerade marknadspriser. För Hestonmodellen är denna process beräkningsmässigt krävande, eftersom den kräver upprepade numeriska utvärderingar av en semi-analytisk prissättningsformel. I praktiska sammanhang där beräkningshastighet är avgörande begränsas därmed tillämpligheten av traditionella kalibreringsmetoder. Detta skapar ett behov av snabbare, men fortsatt noggranna surrogatmodeller.

**Syfte** Studien undersöker i vilken utsträckning neurala nätverkssurrogat kan uppnå samma noggrannhet och stabilitet som semi-analytisk optionsprissättning, samtidigt som de ger betydande förbättringar i beräkningstid. Vidare undersöker studien även hur olika modelldesignval påverkar prestandan genom att jämföra modellerna både med varandra och med COS-metoden. Målet är att identifiera avvägningar mellan olika tillvägagångssätt och därigenom vägleda vilken surrogatmodell som bäst uppfyller kraven i olika tillämpningar.

**Teoretiskt ramverk** Studien bygger på Black-Scholes optionsprissättningsteori (Black & Scholes, 1973), Hestonmodellen (Heston, 1993) och dess semi-analytiska prissättning via COS-metoden (Fang & Oosterlee, 2008). Surrogatmodellerna tränas utifrån DML-ramverket av Huge & Savine (2020), vilket innebär att parameterderivator, så kallade ”Greeks”, inkluderas i träningen genom Sobolev-träning.

**Metod** Syntetisk träningsdata genererades genom att välja ut Heston-parametrar från ett brett parameterrum med hjälp av Sobol-sekvenser. Optionspriser och Greeks beräknades därefter via COS-metoden, med derivator erhållna via automatisk differentiering. Dessutom erhölls motsvarande implicita volatiliteter (IV) och IV-Greeks via ”Let's Be Rational”-metoden. Totalt tränades därefter fyra surrogatmodeller: en multilayer perceptron tränad på priser (MLP_P), samt tre differentiella neurala nätverk tränade på priser (DDN_P), priser normaliserade med lösenpris (DDN_PdivK) respektive IVs (DDN_IV). Modellerna utvärderades sedan på ett separat testdataset. Prestandan bedömdes med avseende på noggrannhet i prissättning, Greek-stabilitet, beräkningshastighet och förekomst av arbitragebrott, med COS-metoden som referensmetod.

**Resultat och implikationer** Alla neurala nätverk uppnådde två till tre storleksordningar snabbare beräkningar jämfört med COS-referensen. DDN_IV var 764 gånger snabbare över hela beräkningskedjan för IV och IV-Greeks. Modellen gav även lägst fel i både prissättning och Greeks, nästan helt utan arbitragebrott, vilket innebar att den överträffade både MLP_P

och övriga DDN-modeller inom både pris- och IV-domänen. Alla DDN-modeller producerade dessutom jämnare parameterderivator än COS-referensen och uppvisade stabilt beteende över både typiska och extrema parameterkombinationer. Sammantaget visar resultaten att DDN_IV, som tränas på implicita volatiliteter, är den mest lovande surrogatmodellen för att effektivisera kalibrering av Hestonmodellen.

Nyckelord: Hestonmodellen, Gradient-baserad kalibrering, Parameter-Greeks, Deep Differential Learning, Neurala nätverkssurrogat, Implicit volatilitet, Fourier-Cosinus (COS) metoden

Notera: Denna rapport är skriven på engelska.

# Preface

This bachelor's thesis was carried out in the spring of 2026 at the Department of Technology Management and Economics at Chalmers University of Technology in Gothenburg, Sweden. This thesis constitutes 15 credits and concludes our bachelor's studies in Industrial Engineering and Management.

We would like to express our gratitude to our supervisor, Carl Lindberg. His teaching in courses related to financial and quantitative mathematics served as a key inspiration for this work. Having him as our supervisor, and for several of us also as a teacher during our time at Chalmers, has truly been a privilege.

Gothenburg, May 2026

Alex Andersson
Linnea Davidson
Markus Eliassen
Jacob Hansén
Isak Stridh
Måns Westman

# Contents

# 1 Introduction

## 1.1 Background

Over the years, stochastic models have been developed to improve the accuracy of pricing and hedging of financial derivatives (Hull, 2022). In the context of financial options, the Black–Scholes formula remains a widely used benchmark in option pricing theory. The method, published in 1973, is especially valuable, as option prices can be expressed in terms of the price distribution of the underlying asset (Black & Scholes, 1973). However, the Black–Scholes model assumes that the underlying asset price follows a geometric Brownian motion with constant volatility. Heston (1993) extended the model by introducing a stochastic variance process that follows a mean-reverting Cox–Ingersoll–Ross process, thereby implying stochastic volatility (Cox et al., 1985). A more detailed derivation of Heston's stochastic volatility model can be found in Section 2.2.

However, the stochastic variance feature prevents a closed-form expression for option prices in the original price domain. Specifically, the probability density function required for integration is not explicitly known for the relevant asset prices. Nevertheless, the characteristic function for the Heston model is known (Heston, 1993), which enables option prices to be computed via one-dimensional numerical integration.

This implies that the Heston PDE has well-defined solutions in theory, but they cannot be solved analytically. Consequently, numerical methods are typically required to compute option prices. In computational finance, such methods are frequently applied in the valuation of financial derivatives as well as contemporary risk management practices (Liu, Oosterlee, & Bohte, 2019). For instance, such numerical methods typically rely on solving the PDE directly, using Monte Carlo simulations or evaluating numerical integrals in the Fourier space, as in the COS method (Fang & Oosterlee, 2008).

In practical derivative pricing, asset models must often be calibrated to market data, which means that the open parameters in the asset price model need to be fitted (Liu, Oosterlee, & Bohte, 2019). The Heston model itself contains several parameters that make up the dynamics of the stochastic variance process, for example the volatility of the variance $\sigma$, initial variance $v_0$, mean-reversion level $\theta$, and mean-reversion speed $\kappa$ (Heston, 1993). These parameters are not directly observable from the market data, which means that they must be calibrated by adjusting them to match the prices of options that are liquid and actively traded in the market (Liu, Oosterlee, & Bohte, 2019).

Since option prices have no closed-form solutions in the Heston model, calibrating the parameters requires repeated numerical evaluation for thousands of option strikes and maturities. For each set of Heston parameters, option prices must then be compared to the market prices, which can quickly become computationally expensive. This is particularly relevant in practical settings such as high-frequency trading, real-time risk management, or dealing with counterparty credit risk issues, where the requirement for highly efficient computation is strong. The described computational burden often results in a trade-off between computational efficiency and precision (Liu, Oosterlee, & Bohte, 2019).

Consequently, efficient calibration has been a persistent problem in certain financial settings. Recent research has turned towards a more data-driven approach, utilizing deep learning techniques to minimize the computational burden associated with calibrating stochastic volatility models. These techniques strive to construct approximations to the calibration problem with fast surrogate models. The strategy can, according to Sridi and Bilokon (2023), be divided into two principal approaches.

The first, often referred to as the "*one-step approach*" or "*Deep Calibration*", was developed by Hernandez (2016), where an Artificial Neural Network (ANN) was developed to directly map market data to calibrated model parameters. The second approach, known as the "*two-step approach*", is more com-

monly adopted, and applies deep learning to approximate the option prices first, typically for vanilla European contracts. The model parameters are then calibrated by minimizing pricing errors computed through the trained neural network, using traditional optimization techniques (Sridi & Bilokon, 2023). The "two-step approach" has, for example, been used by Liu, Oosterlee, and Bohte (2019) to calibrate stochastic volatility models, such as the Heston and Bates models.

Furthermore, Huge and Savine (2020) developed a Differential Machine Learning (DML) approach, which serves as an extension of standard supervised learning models. Unlike traditional models that train a neural network solely on input-output pairs, DML also includes the derivatives of the outputs with respect to the inputs in the loss function. By learning from the additional differential information during training, DML allows neural networks to compute both function values and their sensitivities. This leads to improved accuracy compared to non-differential neural networks.

While DML has shown strong results in pricing and risk management, its use has been comparatively limited in terms of calibrating stochastic volatility models. One such study is Sridi and Bilokon (2023), who apply DML to the Heston model, showing its potential for reducing calibration cost. Even more recently, C. Zhang et al. (2025) proposed a deep differential network (DDN) that learns both Heston prices and its parameter derivatives. By including these derivatives directly into the training loss, their approach enabled fast and stable gradient-based calibration, outperforming standard feedforward networks in accuracy. These derivatives, commonly known as parameter Greeks, measure the sensitivity of the option price to changes in the Heston parameters. Stable and accurate Greeks are vital for gradient-based calibration methods, since they determine how reliably an optimizer can navigate the parameter space (Huge & Savine, 2020; C. Zhang et al., 2025).

Since the application of differential machine learning for Heston model calibration remains at an early stage, several design choices that may affect the practical usefulness and performance of such surrogate models are still not fully explored. This study therefore builds on previous work and addresses the first step of a two-step calibration approach. It does so by examining how certain network design choices affect the accuracy, derivative stability, and computational efficiency of differential neural network surrogates.

## 1.2 Purpose

The purpose of this study is to evaluate the performance of differential neural network surrogates, which incorporate parameter Greeks during training, intended for calibrating the Heston model. The study focuses on how different output representations affect pricing accuracy, Greek stability and computational efficiency.

To put the results in context, the aim is to compare the differential surrogates against both the COS method and a standard non-differential neural network trained only on prices. These benchmarks are used both to further assess the computational efficiency of differential neural networks and to provide reference points for comparing differential surrogate designs.

## 1.3 Research Questions

Accordingly, the main research questions are as follows:

(i) How do output representation choices influence pricing accuracy, Greek stability, and computational efficiency in differential neural network surrogate models intended for calibrating the Heston model?

(ii) How do the surrogate models compare among themselves as well as against the non-differential

neural network and the COS method, and what are the trade-offs between the approaches?

## 1.4 Methodology

To address these questions, we generate theoretical prices under Heston dynamics using synthetic parameter samples across a wide range of market regimes. The prices and their derivatives are generated using a semi-analytical pricing model, giving us access to ground truth Greeks that would be unavailable from market data alone. We implement two types of neural network models, a Deep Differential Neural Network (DDN), trained using both price and derivative information, and a standard Multilayer Perceptron (MLP), trained using price information only. We then develop variants of these to accommodate different types of output, such as implied volatility rather than raw prices. The models are trained on the synthetic data after beneficial feature engineering and scaling. Finally, we evaluate the models across a comprehensive set of tests with a focus on precision, numerical stability, and computational speed.

## 1.5 Delimitations

Throughout this study, certain delimitations are made. For one, we only train the neural networks on European put options. The reason for this is that it ensures greater stability in the COS pricing method. When calculating the price of an option, the COS method evaluates two coefficients, $\chi_k$ and $\psi_k$, over a truncated domain [a,b]. For call options the integration boundaries are from 0 to b. The analytic solution, as derived by Fang and Oosterlee (2008), contains the term $e^b$. To ensure accurate prices, $b$ must be large enough to capture the tails of the distribution. However, this introduces very large values to the summation, causing instability in pricing due to cancellation errors. On the other hand, for put options, $\chi_k$ and $\psi_k$ are evaluated through integration from $a$ to 0. Because $a$ is a negative number, the exponential terms are bounded between 1 and a value close to 0. This in turn makes the put pricing formula inherently more stable by removing the calculation errors (Fang & Oosterlee, 2008). The put-call parity is then used to obtain call prices, if needed. However, during calibration observed call options can be converted to puts and then form the basis of calibration. Furthermore, the restriction to European options is made to align with recent research. As emphasized by Sridi and Bilokon (2023), differential machine learning techniques can in principle be extended to other option types, but this requires more involved pricing procedures.

The dividend yield, $q$, is assumed to be zero with no loss of generality regarding the option pricing dynamics. This is because, in the risk-neutral measure of the stock model derived by Heston (1993), the drift of the underlying asset is determined by the term ($r$-$q$), where $r$ is the risk-free rate. As shown by Rouah (2013), if it is necessary to introduce dividend yield into the model, the process is straightforward if it can be assumed that the dividend payment is a continuous yield.

Furthermore, while motivated by the problem of calibrating the Heston model, this study focuses on the surrogate pricing implementation. Previous studies have shown how surrogate pricing methods providing the parameter sensitivities can be integrated into a gradient-based calibration framework (Huge & Savine, 2020; Sridi & Bilokon, 2023; C. Zhang et al., 2025). This study, on the other hand, examines the absolute pricing and sensitivity performance of the core surrogate models, allowing for a deeper analysis of the different models' behavior. Hence, the focus lies on thoroughly examining different aspects of performance such as stability, arbitrage, and extrapolation. The implications of these results for different calibration frameworks are discussed from a theoretical perspective.

# 2 Theoretical Background

This chapter presents the theoretical foundations that are required to follow the study. An explanation of fundamental concepts in option pricing theory is followed by a description of the Heston stochastic volatility model. The numerical pricing methods, particularly Fourier-based techniques, are then introduced together with the theory regarding model calibration and the methods for computing implied volatility. The section ends by considering the principles of neural networks, differential machine learning, and neural network-based pricing approximations for the Heston model.

## 2.1 Financial Derivatives and Option Pricing Theory

This section introduces relevant fundamental theory regarding financial derivatives and arbitrage-free option pricing.

### 2.1.1 European Options and Payoff Structures

A derivative is a financial contract whose value is determined by, or related to, the value of one or more underlying assets (Hull, 2012). The derivatives considered in this investigation are commonly referred to as European vanilla options. An option gives the holder the right, but not the obligation, to buy or sell the underlying asset by a predetermined date, called the *exercise date* or *maturity*, and for a predetermined price, called the *strike price*. If the option gives the holder the right to buy, it is referred to as a call option, while an option giving the holder the right to sell is referred to as a put option. A European option can only be exercised at its maturity date, distinguishing it from American-style contracts which allow early exercise (Hull, 2012). With $K$ denoting the strike price and $S_T$ denoting the underlying asset price at maturity $T$, the payoff of a European vanilla call option is given by:

$$C(S_T) = \max(S_T - K, 0) \tag{1}$$

and for a European put option:

$$P(S_T) = \max(K - S_T, 0), \tag{2}$$

as defined in Hull (2022, p. 231). An important relation between the European put and call prices is the standard put-call parity, which states that:

$$C + Ke^{-r\tau} = P + S_t, \tag{3}$$

where $\tau = T - t$ is the time to maturity. $S_t$ is the value of the underlying asset at time t, $C$ is the price of a call option, $P$ is the price of a put option, $r$ is the risk-free interest rate, and $e^{-r\tau}$ is the discount factor. The relationship shows that the value of a European call with a given strike price and maturity can be derived from a European put with the same strike and maturity, and vice versa. (Hull, 2022, pp. 255-256)

### 2.1.2 Arbitrage and Risk-Neutral Measure

An arbitrage opportunity is a self-financed trading strategy that does not require an initial investment, and has a positive probability of a strictly positive payoff without any risk of loss (Shreve, 2004, p. 231). The following pricing theories build upon the assumption that such arbitrage opportunities are non-existent.

The zero-arbitrage assumption, combined with the ability to replicate derivative payoffs, implies that prices do not depend on investors' risk preferences. If the payoff of a derivative, such as a European option, can be replicated by a self-financed trading strategy in the underlying asset and a risk-free money market account, then the price of the option must equal the cost of constructing this replicating portfolio. Otherwise, there would be arbitrage opportunities. Hence, derivative prices are solely determined by traded asset prices and therefore do not depend on investors' risk preferences (Hull, 2022, p. 351-352).

This allows us to express derivative prices under the risk-neutral measure $\mathbb{Q}$, under which all assets earn the risk-free rate in expectation. Pricing therefore becomes a matter of taking discounted expectations under $\mathbb{Q}$, without requiring a model of investors' risk preferences. Under $\mathbb{Q}$, following the notation of Shreve (2004, p. 218) who denotes this measure $\tilde{\mathbb{P}}$, the price of a derivative security with payoff $V(T)$ at maturity $T$ is given by the *risk-neutral pricing formula*:

$$V(t) = \mathbb{E}^{\mathbb{Q}}\Big[e^{-\int_t^T r(u)\,du}\,V(T)\;\Big|\;\mathcal{F}(t)\Big], \quad 0 \le t \le T, \tag{4}$$

where $r(u)$ is the risk-free rate at time $u$. Throughout this thesis, $r$ is assumed constant, so $e^{-\int_t^T r(u)\,du} = e^{-r\tau}$ where $\tau = T - t$. It is important to emphasize that when moving from the real-world probability measure to the risk-neutral measure, both the expected payoff and the appropriate discount rate change. These effects exactly offset each other, so the resulting derivative price remains unchanged (Hull, 2022, pp. 351-352).

### 2.1.3 The Black–Scholes Model

Introduced by Black and Scholes (1973), the Black–Scholes framework remains one of the most used methods for option pricing. The model views the underlying stock price as a continuous-time stochastic process with one deterministic trend, and one part to capture the random fluctuations in the stock:

$$dS_t = \mu S_t\,dt + \sigma S_t\,dW_t \tag{5}$$

Here, $S_t$ is the stock price at time $t$, $\mu$ is the stock's expected return, $\sigma$ is the constant volatility, and $(W_t)_{t\ge 0}$ is a standard Brownian motion.

The drift $\mu$ reflects investors' risk appetite and expected growth of the stock, hence it is neither an objective measure nor directly observable. In an arbitrage-free market, the option price cannot depend on $\mu$. The replication argument instead implies that the hedged position, which locally replicates the option price, is risk-free and must grow at the risk-free rate $r$ instead of $\mu$. Hence, the risk-neutral dynamics of the stock price become:

$$dS_t = rS_t\,dt + \sigma S_t\,dW_t \tag{6}$$

Having specified the risk-neutral dynamics of the stock, we can consider a European option on the same stock with price $V(S,t)$ as a function of the current stock price $S$ and time $t$. The key here is to locally replicate the option payoff by dynamically trading in the underlying stock and a risk-free money market

account. This is done by forming a self-financing portfolio, which consists of a position in the underlying stock as well as the option.

In order to construct this strategy, we let $X_t$ be the value of the self-financing portfolio that holds $\Delta_t$ shares of the underlying stock, and invests the remaining wealth into the risk-free money market account. This yields that:

$$X_t = \Delta_t S_t + B_t \tag{7}$$

where $B_t$ is equivalent to the cash position held in the money market account. Since the strategy itself is self-financing, changes in the portfolio only come from gains and losses in the stock position and interest from the cash position. Hence, the dynamics of the portfolio become:

$$dX_t = \Delta_t dS_t \ + r(X_t - \Delta_t S_t)dt \tag{8}$$

To find the PDE of the option price, we first state *Itô's lemma* (Hull, 2022, p. 327):

$$dG = \left(\frac{\partial G}{\partial x}a + \frac{\partial G}{\partial t} + \frac{1}{2}\frac{\partial^2 G}{\partial x^2}b^2\right)dt + \frac{\partial G}{\partial x}b\,dz \tag{9}$$

Applying Equation 9 to the option price $V(S_t, t)$, where $S_t$ follows Equation 6, yields:

$$dV = \frac{\partial V}{\partial t}\,dt + \frac{\partial V}{\partial S}\,dS_t + \frac{1}{2}\frac{\partial^2 V}{\partial S^2}\,\sigma^2 S_t^2\,dt \tag{10}$$

By choosing $\Delta_t = \frac{\partial V}{\partial S}$, the portfolio's exposure to the stock's random fluctuations ($dW_t$) exactly matches that of the option. This choice is known as delta-neutral hedging, as it eliminates the portfolio's instantaneous exposure to the stock price. This makes the portfolio locally risk-free. Since the hedged portfolio now earns the risk-free rate, equating the deterministic parts that are left yields the Black–Scholes PDE (Hull, 2022):

$$\frac{\partial V}{\partial t} + rS\frac{\partial V}{\partial S} + \frac{1}{2}\sigma^2 S^2\frac{\partial^2 V}{\partial S^2} - rV = 0 \tag{11}$$

subject to the terminal condition

$$V(S_T) = \begin{cases} \max(S_T - K, 0), & \text{Call option} \\ \max(K - S_T, 0), & \text{Put option} \end{cases} \tag{12}$$

Here, $V(S_T)$ represents the terminal payoff at maturity $T$.

For a European call option, $V(S_T) = \max(S_T - K, 0)$, solving Equation 11 yields the Black–Scholes formula (Black & Scholes, 1973):

$$C(S, t) = S\mathcal{N}(d_1) - Ke^{-r\tau}\mathcal{N}(d_2) \tag{13}$$

where

$$d_1 = \frac{\ln(S/K) + \left(r + \frac{1}{2}\sigma^2\right)\tau}{\sigma\sqrt{\tau}}, \qquad d_2 = d_1 - \sigma\sqrt{\tau}, \tag{14}$$

and $\mathcal{N}(\cdot)$ denotes the standard normal cumulative distribution function.

The corresponding European put price can be obtained via the put-call parity shown in Equation 3.

### 2.1.4 Implied Volatility

As noted above, the Black–Scholes model assumes a constant volatility parameter $\sigma$. This volatility is also the one parameter that cannot be directly observed in the market. Therefore, it is common to instead work with so-called *implied volatilities*, that is, the volatilities implied by option prices in the market (Hull, 2022, p. 358).

When the Black–Scholes formula is inverted to recover the volatility from observed market prices, the resulting value is known as the *implied volatility*. Formally, the implied volatility $\sigma^{IV}(K,T)$ for a given market price $V^{\text{mkt}}$ is defined as the unique $\sigma > 0$ that satisfies

$$V^{\text{BS}}(S,K,T,r,\sigma) = V^{\text{mkt}}(K,T), \tag{15}$$

where $V^{\text{BS}}$ denotes the Black–Scholes pricing formula as derived above. The existence and uniqueness of $\sigma^{IV}$ follow from the strict monotonicity of the Black–Scholes price with respect to volatility, i.e., from the positivity of the option Vega (Hull, 2022, pp. 434-436).

The pattern of implied volatilities across strikes for a fixed maturity is commonly referred to as the *volatility smile* or *volatility skew*. The full surface of $\sigma^{IV}(K,T)$ across different strikes and maturities is called the *implied volatility surface*. The existence of a non-flat implied volatility surface contradicts the constant-volatility assumption of Black–Scholes and is one of the main motivations for stochastic volatility models such as the Heston model (Heston, 1993).

### 2.1.5 Black–Scholes Greeks

The sensitivities of the Black–Scholes option price with respect to its inputs are known as the *Greeks*. For this study, the most relevant Greeks are:

- **Delta ($\Delta$):** the sensitivity of the option price to the underlying asset price,

$$\Delta_{\text{call}} = \frac{\partial V_{\text{call}}}{\partial S} = \mathcal{N}(d_1), \tag{16}$$

$$\Delta_{\text{put}} = \frac{\partial V_{\text{put}}}{\partial S} = \mathcal{N}(d_1) - 1, \tag{17}$$

  where $\mathcal{N}(\cdot)$ is the standard normal cumulative distribution function. Delta measures how much the option price changes for a small change in the underlying asset. Since $\mathcal{N}(d_1) \in (0,1)$, the call Delta is bounded in $(0,1)$ and the put Delta in $(-1,0)$. The put price moves inversely to the underlying but never by more than the underlying asset's own price change, in line with the no-arbitrage constraints in Section 2.1.6.

- **Vega ($\mathcal{V}$):** the sensitivity of the option price to volatility. Starting from the Black–Scholes call price in Equation 13, differentiating with respect to $\sigma$ and applying the chain rule gives

$$\frac{\partial C}{\partial \sigma} = S\,\mathcal{N}'(d_1)\frac{\partial d_1}{\partial \sigma} - Ke^{-r\tau}\,\mathcal{N}'(d_2)\frac{\partial d_2}{\partial \sigma}. \tag{18}$$

  Since $S\,\phi(d_1) = Ke^{-r\tau}\phi(d_2)$ (which follows from the fact that $d_2 = d_1 - \sigma\sqrt{\tau}$ and the log-normal structure of the model). Also note that $\phi(\cdot) = \mathcal{N}'(\cdot)$ is the standard normal density. Since $\frac{\partial d_2}{\partial \sigma} = \frac{\partial d_1}{\partial \sigma} - \sqrt{\tau}$, the expression simplifies to

$$\mathcal{V} = \frac{\partial C}{\partial \sigma} = S\,\phi(d_1)\sqrt{\tau}, \tag{19}$$

Vega measures how much the option price changes in response to changes in volatility and is identical for calls and puts.

- **Theta ($\theta$):** the sensitivity of the option price to time,

$$\theta_{\text{call}} = -\frac{S\,\phi(d_1)\,\sigma}{2\sqrt{\tau}} - rKe^{-r\tau}\,\mathcal{N}(d_2), \tag{20}$$

$$\theta_{\text{put}} = -\frac{S\,\phi(d_1)\,\sigma}{2\sqrt{\tau}} + rKe^{-r\tau}\,\mathcal{N}(-d_2). \tag{21}$$

  Theta represents the rate of time decay of the option price.

- **Gamma ($\Gamma$):** the second-order sensitivity to the underlying price,

$$\Gamma = \frac{\partial^2 V}{\partial S^2} = \frac{\phi(d_1)}{S\sigma\sqrt{\tau}}. \tag{22}$$

  Gamma measures how delta changes with the underlying price, representing its mathematical curvature, and is identical for calls and puts. Due to Delta being strictly increasing, Gamma must be non-negative across the entire domain. A violation of this property implies local concavity in the pricing function, which introduces static arbitrage opportunities as explained in Section 2.1.6 (Hull, 2022, pp. 429–433).

Closed-form expressions for all Black–Scholes Greeks follow directly from differentiating the Black–Scholes formula. A full overview is given in Hull (2022, pp. 420–439).

### 2.1.6 No-Arbitrage Constraints for Vanilla Options

The price of a European put is bounded by two criteria. It cannot be worth more than the discounted strike, since the maximum payoff $\max(K - S_T, 0)$ cannot exceed $K$, giving (Hull, 2022, p. 253):

$$P \leq Ke^{-r\tau}. \tag{23}$$

It also cannot be worth less than zero or less than the amount by which the strike exceeds the current stock price (Hull, 2022, p. 255):

$$P \geq \max\left(Ke^{-r\tau} - S_t,\, 0\right). \tag{24}$$

Beyond individual price bounds, no-arbitrage also enforces constraints on how option prices must relate across different strikes and maturities. Note that the following constraints are stated for call options but apply for puts via the put-call parity in Equation 3.

The first constraint concerns monotonicity in the strike price, often referred to as vertical spread arbitrage. If a call option is exercised in the future, its payoff equals the difference between the stock price and the strike price, provided that this difference is positive. As a result, call options increase in value as the stock price rises and are less valuable when the strike price is higher. Merton (1973) formalizes this as the inequality:

$$C(K_1, T) \geq C(K_2, T) \tag{25}$$

for any $K_1 < K_2$.

The second constraint requires that the call price is convex in the strike, also called butterfly spread. Again, Merton (1973) proves that this holds using a dominance argument. Consider buying one call at a low strike, $K_1$, and another call at a higher strike $K_3$, and selling two calls at a strike in between, $K_2$. Taking this position is known as the butterfly spread, and will always pay off a non-negative amount at

expiry, so it cannot have a negative cost. For equally spaced strikes ($K_2 - K_1 = K_3 - K_2$), the condition is formalized as:

$$C(K_1, T) - 2C(K_2, T) + C(K_3, T) \geq 0 \tag{26}$$

The third constraint considers monotonicity in maturity. Hull (2022, p. 248) notes that a longer-dated call must be worth at least as much as an identical shorter-dated call, which gives the constraint:

$$C(K, T_2) \geq C(K, T_1) \tag{27}$$

for any $T_1 < T_2$. The longer-dated option offers strictly more optionality. However, it is important to note that this condition only holds under the assumption of non-negative interest rates and dividends.

The calendar spread condition can equivalently be expressed in terms of the total implied variance surface. With $m = \ln(S/K)$ denoting the log-moneyness, the total implied variance is defined as

$$w(m, \tau) = \tau\big(\sigma^{\mathrm{IV}}(m, \tau)\big)^2 \tag{28}$$

For a fixed $m$, $w$ must be non-decreasing in $\tau$ to avoid calendar arbitrage (Gatheral, 2006). If this condition is violated, an option with a longer time to maturity could have a lower call price than an otherwise identical option with a shorter time to maturity (Roper, 2010).

To summarize, Carr and Madan (2005) combine these constraints and state that they are jointly sufficient for a finite grid of European call quotes to be free of static arbitrage. Hence, if every vertical, butterfly, and calendar spread formed on the quoted prices is non-negatively priced, it means that no further tests are needed to certify that the grid is free of static arbitrage.

## 2.2 Stochastic Volatility Modeling

### 2.2.1 Motivation for Stochastic Volatility

Under the constant-volatility assumption of the Black–Scholes model, the implied volatility surface would essentially be flat across all strikes and maturities. However, as shown in Section 2.1.4, this does not align with what is observed in practice. The implied volatilities observed in markets vary between different strikes and maturities, which directly opposes the assumption and produces patterns such as volatility smiles (Hull, 2022, p. 358). This suggests that volatility itself varies over time in a way that is uncertain. Stochastic volatility models account for this by allowing the volatility of the underlying asset to follow its own stochastic process.

### 2.2.2 The Heston Model and its Dynamics

Heston (1993, p. 328) was one of many to argue that the constant volatility assumption is unreliable. Earlier stochastic volatility models lacked direct analytical solutions without known characteristic functions and often relied on heavy numerical computations to solve two-dimensional PDEs.

Heston (1993) therefore proposed a model which computationally is somewhere in between the Black–Scholes formula and earlier proposed stochastic volatility models. Heston's approach does not yield a direct formula for the European call price, but since the characteristic function is known, pricing is reduced to a one-dimensional numerical integration. This is substantially cheaper than solving a two-dimensional PDE.

The model introduces a stochastic variance process of Cox–Ingersoll–Ross (CIR) type (Cox et al., 1985),

correlated with the underlying asset price. Hence, the Heston model can be represented by a system of two stochastic differential equations (SDEs): one for the asset price $S_t$ and one for its variance $v_t$. Under the physical real-world measure $\mathbb{P}$, they evolve according to the following dynamics (Rouah, 2013, p. 1):

$$\begin{cases} dS_t = \mu S_t \, dt + \sqrt{v_t} \, S_t \, dW_{1,t} \\ dv_t = \kappa(\theta - v_t) \, dt + \sigma \sqrt{v_t} \, dW_{2,t} \end{cases} \tag{29}$$

Here, $W_{1,t}$ and $W_{2,t}$ are two correlated Brownian motions such that $\mathbb{E}^{\mathbb{P}}[dW_{1,t} \, dW_{2,t}] = \rho \, dt$, with $\rho \in [-1, 1]$. The parameters are: $\mu$, the drift of the stock price; $\kappa > 0$, the mean reversion speed of the variance; $\theta > 0$, the mean reversion level of the variance; $\sigma > 0$, the volatility of the variance and $v_0 > 0$, the initial variance level (Rouah, 2013, pp. 1-2).

However, for pricing financial derivatives, and especially for pricing options, it is important to work with $S_t$ and $v_t$ under the risk-neutral measure $\mathbb{Q}$, as explained in Section 2.1.2. Hence, the real-world drift $\mu$, is now replaced with the risk-free rate $r$. The SDEs are then modified by applying Girsanov's theorem found in Shreve (2004, Thm. 5.4.1). Girsanov's theorem allows a change of probability measure $\mathbb{P}$ to $\mathbb{Q}$, which modifies the drift terms of the SDEs while leaving volatility and correlation structure intact. After the theorem is applied, it yields the following SDEs:

$$\begin{cases} dS_t = r S_t \, dt + \sqrt{v_t} \, S_t \, dW_{1,t}^{\mathbb{Q}} \\ dv_t = [\kappa(\theta - v_t) - \lambda(S_t, v_t, t)] \, dt + \sigma \sqrt{v_t} \, dW_{2,t}^{\mathbb{Q}} \end{cases} \tag{30}$$

where the function $\lambda(S_t, v_t, t)$ is the volatility risk premium. The drift of the stock price $S_t$ is forced to equal the risk-free rate $r$ under $\mathbb{Q}$. However, the variance process $v_t$ is not directly tradeable, which means that there is no corresponding arbitrage argument under $\mathbb{Q}$ that uniquely determines its drift. An additional assumption is therefore needed. Hence, Heston (1993) assumes that the volatility risk premium is proportional to the variance, $\lambda(S_t, v_t, t) = \lambda v_t$, where $\lambda$ is a constant.

Substituting $\lambda(S_t, v_t, t) = \lambda v_t$ and expanding the dynamics of the variance will yield:

$$dv_t = [\kappa(\theta - v_t) - \lambda v_t] \, dt + \sigma \sqrt{v_t} \, dW_{2,t}^{\mathbb{Q}} = [\kappa\theta - (\kappa + \lambda) v_t] \, dt + \sigma \sqrt{v_t} \, dW_{2,t}^{\mathbb{Q}} \tag{31}$$

Defining $\kappa^* = \kappa + \lambda$ and $\theta^* = \frac{\kappa\theta}{\kappa + \lambda}$ restores the earlier mentioned CIR mean-reverting form, and all properties of the CIR process carry over to the newly defined risk-neutral dynamics (Rouah, 2013).

The complete risk-neutral dynamics of the Heston model are hence given by (Rouah, 2013, p. 3):

$$\begin{cases} dS_t = r S_t \, dt + \sqrt{v_t} \, S_t \, dW_{1,t}^{\mathbb{Q}} \\ dv_t = \kappa^*(\theta^* - v_t) \, dt + \sigma \sqrt{v_t} \, dW_{2,t}^{\mathbb{Q}} \end{cases} \tag{32}$$

where $\mathbb{E}^{\mathbb{Q}}[dW_{1,t}^{\mathbb{Q}} \, dW_{2,t}^{\mathbb{Q}}] = \rho \, dt$. Note that when $\lambda = 0$, we have that $\kappa^* = \kappa$ and $\theta^* = \theta$. Since $\lambda$ is embedded in the risk-neutral parameters $\kappa^*$ and $\theta^*$, it does not need to be estimated separately when calibrating with option prices. This thesis therefore sets $\lambda = 0$ throughout following Rouah (2013, p. 3). All parameters in the remainder of this chapter are risk-neutral quantities, and the $*$-notation is dropped.

### 2.2.3 Feller Condition and Structural Limitations of the Variance Process

As noted above, the variance process follows a CIR-type SDE. For such a process, the Feller condition restricts the model parameters according to:

$$2\kappa\theta > \sigma^2 \tag{33}$$

If the condition holds, the mean-reverting drift is essentially strong enough to prevent the variance from ever reaching zero, meaning that $v_t > 0$ for all $t$. If a violation of the Feller condition occurs, the variance can just touch zero but bounces right back, remaining non-negative (Cox et al., 1985; Rouah, 2013, p. 4). However, in practice, Feller violations often occur because the steep implied volatility skew observed in equity markets typically needs a large $\sigma$. This makes the Feller condition $2\kappa\theta > \sigma^2$ harder to satisfy, which explains why calibrated parameters often violate it (Lin et al., 2017). Such violations can in turn cause numerical instability when evaluating the characteristic function, which is addressed in Section 2.2.6.

Beyond the numerical issues, the assumption that variance moves smoothly and continuously according to the CIR process poses a structural challenge. It means that the model cannot produce the rapid and large shifts in volatility that would be needed to match steep skews observed in short maturity, deep out-of-the-money (OTM) options (Bakshi et al., 1997, p. 2005). In other words, the market usually prices such options to reflect the possibility of sudden price drops, but a continuous variance process may not generate enough extreme behavior over a short period of time to reflect this.

Hence, this is not an issue in calibration, but rather a property of the model. Therefore, large pricing errors in deep OTM options with short maturities may reflect a theoretical limitation of the Heston model itself, rather than a flaw in the pricing model or the neural network trying to approximate it.

### 2.2.4 The Heston PDE and the Call Price Formula

With the risk-neutral dynamics and variance process established, one needs to determine how to actually price options or derivatives under the Heston model. First, we need to derive the Heston PDE.

To simplify, let us follow the same approach as for deriving the Black–Scholes PDE in Section 2.1.3. We apply Itô's lemma to a derivative $U(S_t, v_t, t)$, which depends on both the variance and the stock price:

$$dU = \frac{\partial U}{\partial t}\,dt + \frac{\partial U}{\partial S}\,dS_t + \frac{\partial U}{\partial v}\,dv_t + \frac{1}{2}\frac{\partial^2 U}{\partial S^2}\,(dS_t)^2 + \frac{1}{2}\frac{\partial^2 U}{\partial v^2}\,(dv_t)^2 + \frac{\partial^2 U}{\partial S \partial v}\,dS_t\,dv_t \tag{34}$$

In Black–Scholes, only the $(dS_t)^2$ second-order enters Itô's formula, since volatility is constant. In this case, because $v_t$ is also stochastic, the $(dv_t)^2$ and $dS_t\,dv_t$ terms also appear. Substituting the risk-neutral SDEs from Equation 32 and applying the Itô multiplication rules ($dW^{\mathbb{Q}}_{i,t} \cdot dW^{\mathbb{Q}}_{i,t} = dt$, $dW^{\mathbb{Q}}_{1,t} \cdot dW^{\mathbb{Q}}_{2,t} = \rho\,dt$, and $dt \cdot dW^{\mathbb{Q}}_{i,t} = 0$) separates $dU$ into a deterministic part (terms in $dt$) and two random parts (terms in $dW^{\mathbb{Q}}_{1,t}$ and $dW^{\mathbb{Q}}_{2,t}$).

Once again, it is necessary to make the portfolio locally risk-free. This is done by eliminating both sources of randomness, and the replicating portfolio therefore requires a position both in the underlying stock and a second derivative that depends on $v_t$. Remember that in Black–Scholes, only hedging the stock is sufficient, since that is the only source of randomness. Once both random terms in the Heston model are hedged away, the portfolio is locally risk-free and must, as before, earn the risk-free rate $r$. Rearranging terms yields the Heston PDE, which expressed in terms of the log-price $x = \ln S_t$ takes the form (Heston,

1993, p. 329; Rouah, 2013, p. 8):

$$\frac{\partial U}{\partial t} + \frac{1}{2}v\frac{\partial^2 U}{\partial x^2} + \left(r - \tfrac{1}{2}v\right)\frac{\partial U}{\partial x} + \rho\sigma v\frac{\partial^2 U}{\partial v \partial x} + \frac{1}{2}\sigma^2 v\frac{\partial^2 U}{\partial v^2} + \kappa(\theta - v)\frac{\partial U}{\partial v} - rU = 0 \tag{35}$$

subject to the terminal condition $U(x, v_T, T) = \max(e^x - K, 0)$ for a European call option, where $e^x = S_T$.

This is the Heston equivalent of the Black–Scholes PDE in Equation 11. However, since it is two-dimensional, solving it numerically is computationally expensive, making it difficult to produce accurate prices sufficiently fast for real-time applications. So, Heston (1993) found a way to avoid this by deconstructing the European call price into two probabilities, and then computing these probabilities through the use of characteristic functions.

The European call price via the Heston model can with these probabilities be expressed as (Heston, 1993, p. 330; Rouah, 2013, p. 4):

$$C(K) = S_t P_1 - K e^{-r\tau} P_2 \tag{36}$$

This has the same structure as the Black–Scholes formula in Equation 13, with $P_1$ and $P_2$ replacing $\mathcal{N}(d_1)$ and $\mathcal{N}(d_2)$. Here $P_2 = \mathbb{Q}(S_T > K)$ is the risk-neutral probability of the option expiring in the money. $P_1 = \mathbb{Q}_S(S_T > K)$ is also a probability of finishing in the money, but computed under a measure where the fact that higher stock prices contribute more to the option's value is taken into account. By substituting Equation 36 back into the Heston PDE (Equation 35), one can show that each $P_j$ satisfies its own PDE (Rouah, 2013, pp. 9-10).

The Feynman-Kac theorem (Shreve, 2004, Thm. 6.4.1) states that the solution to each of these PDEs can be rewritten as a conditional expectation. A key insight in Heston (1993) is that, due to the affine structure of the Heston dynamics, this can be expressed as a characteristic function of $\ln S_T$. Rather than solving the two-dimensional PDEs numerically, $P_1$ and $P_2$ can then be recovered from these characteristic functions via Fourier inversion. The characteristic function and the Fourier inversion procedure are presented in more detail in the following Section 2.2.5.

### 2.2.5 Characteristic Function and Pricing via Fourier Inversion

The problem that arises in Heston's call price formula (Equation 36) is that $P_1$ and $P_2$ are needed to price the call. In Black–Scholes, one would normally integrate over the known normal density. However, in Heston, the density is unknown. Fortunately, each $P_j$ can be recovered from the corresponding characteristic function $f_j(\phi) = \mathbb{E}[\, e^{i\phi \ln S_T}]$, which contains the same information as the density and is available in closed-form. Heston (1993, p. 331) showed that it takes the log-linear form

$$f_j(\phi, \tau) = \exp\big(C_j(\phi, \tau) + D_j(\phi, \tau)\, v_0 + i\phi\, x\big), \tag{37}$$

where $x = \ln S_0 + (r - q)\tau$, and $C_j$ and $D_j$ are functions of $\phi$ and $\tau$ that depend on the model parameters $\kappa$, $\theta$, $\sigma$, and $\rho$. The variable $q$ is the dividend yield, and is set to 0, as established in Section 1.5.

Substituting this ansatz into the PDE that each $P_j$ satisfies reduces the two-dimensional problem to a pair of ordinary differential equations. This includes a quadratic ODE for $D_j$, known as a Riccati equation, and a direct integration for $C_j$, both of which can be solved in closed form (Heston, 1993, pp. 341-342). The closed-form solutions require the following intermediate definitions (Rouah, 2013, pp. 13-14):

$$d_j = \sqrt{(\rho\sigma i\phi - b_j)^2 - \sigma^2(2u_j i\phi - \phi^2)}, \qquad g_j = \frac{b_j - \rho\sigma i\phi + d_j}{b_j - \rho\sigma i\phi - d_j}, \tag{38}$$

where $u_1 = \frac{1}{2}$, $u_2 = -\frac{1}{2}$, $b_1 = \kappa - \rho\sigma$, and $b_2 = \kappa$. The solutions are then

$$D_j(\phi, \tau) = \frac{b_j - \rho\sigma i\phi + d_j}{\sigma^2} \left( \frac{1 - e^{d_j \tau}}{1 - g_j \, e^{d_j \tau}} \right), \tag{39}$$

$$C_j(\phi, \tau) = r \, i\phi \, \tau + \frac{\kappa\theta}{\sigma^2} \left[ (b_j - \rho\sigma i\phi + d_j) \, \tau - 2 \ln\!\left( \frac{1 - g_j \, e^{d_j \tau}}{1 - g_j} \right) \right]. \tag{40}$$

The complex square root $d_j$ has two valid values, $+d_j$ and $-d_j$, both of which yield algebraically equivalent characteristic functions. However, it turns out that the choice between them has significant numerical consequences, which is the subject of Section 2.2.6.

Now, given the characteristic functions in Equation 37, the probabilities $P_1$ and $P_2$ are recovered via a theorem called the Gil-Pelaez inversion theorem used by Heston (1993, p. 331, Equation 18):

$$P_j = \frac{1}{2} + \frac{1}{\pi} \int_0^\infty \mathrm{Re}\left[ \frac{e^{-i\phi \ln K} \, f_j(\phi)}{i\phi} \right] d\phi, \tag{41}$$

where $K$ is the strike price. Together with Equation 36, this provides a semi-analytical pricing formula for European options under the Heston model, requiring only the numerical evaluation of two one-dimensional integrals.

### 2.2.6 Numerical Stability and the Little Heston Trap

The expressions for $D_j$ and $C_j$ derived in Section 2.2.5 both contain the complex square root $d_j$ and a complex logarithm. In the complex plane, these have what is called a *branch cut* along the negative real axis, a line across which the function jumps discontinuously (Albrecher et al., 2007). When numerically evaluating the integral in Equation 41, the complex number inside the logarithm in $C_j$ moves through the complex plane as the integration variable $\phi$ increases from 0 to infinity. It is especially the branch cut of the logarithmic function that causes the problem. If it crosses the negative real axis, the output of the logarithm jumps, making the integrand discontinuous, returning an incorrect price without signaling any error (Albrecher et al., 2007, pp. 5-6).

Whether this crossing occurs depends on which of the two square roots, $+d_j$ or $-d_j$, is used. Although both choices yield equivalent characteristic functions, they lead to different algebraic representations of the argument inside the logarithm in $C_j$ (Albrecher et al., 2007, pp. 3-4).

The original formulation from Heston (1993) uses the positive square root $+d_j$. Albrecher et al. (2007) denote this formulation by $\varphi_1$. Following that, the argument of the logarithm inside $C_j$ becomes:

$$G_1(\phi) = \frac{1 - g_j \, e^{\, d_j \tau}}{1 - g_j}. \tag{42}$$

Since $d_j$ is complex, $e^{d_j \tau}$ rotates and grows in magnitude simultaneously as $\tau$ grows. As $\phi$ increases in the integration, $G_1$ therefore spirals outward in the complex plane. For short maturities, the spiral is small and stays away from the negative real axis, but for long maturities, it eventually crosses the earlier mentioned branch cut, breaking the logarithm. Albrecher et al. (2007, p. 9) show that this turns out to be the case for nearly any of the parameters in the Heston model if the maturity is sufficiently large.

The second formulation, denoted $\varphi_2$, is obtained by selecting $-d_j$ instead. Replacing $g_j$ with $c_j = 1/g_j$ and the growing exponential with a decaying one, $e^{-d_j \tau}$, the spiral will shrink instead of expand (Rouah, 2013, pp. 31-32):

$$G_2(\phi) = \frac{1 - c_j \, e^{-d_j \tau}}{1 - c_j}. \tag{43}$$

Since $e^{-d_j\tau}$ decays, $G_2$ never crosses the negative real axis. Albrecher et al. (2007, pp. 12-18) prove that this holds for the full parameter space. Therefore, it is common to use $\varphi_2$ as the characteristic function evaluation. This ensures that both option prices and parameter derivatives are numerically stable across the full maturity and parameter ranges.

## 2.3 Numerical Pricing Methods for the Heston Model

### 2.3.1 Fourier-Based Pricing and the Fast Fourier Transform

Fourier-based pricing methods are essentially a numerical integration approach used to calculate the prices of options based on the Fourier transform of the risk-neutral density of the underlying asset, which is referred to as the characteristic function (Carr & Madan, 1999). This technique is extremely efficient if the characteristic function of the log price is known in analytical form, which holds true in many advanced financial models such as the Heston stochastic volatility model. The traditional approach to implementing Fourier-based pricing involves calculating the value of the option based on the value of its delta ($P_1$) and the risk-neutral probability of finishing in-the-money ($P_2$) as shown in section 2.2.5. The use of the Fast Fourier Transform (FFT) algorithm for pricing options, proposed by Carr and Madan (1999), can be used to dramatically speed up this integration process, calculating multiple option prices simultaneously, reducing the computational complexity significantly. This allows for real-time pricing and hedging of massive portfolios.

The FFT approach suffers from a structural limitation, the relationship between the integration grid ($\Delta w$) and the strike price grid ($\Delta k$) (Fang & Oosterlee, 2008). The algorithm requires the step size of the integration domain (size between discrete integration points) and the step size of the log-strike grid (size between strike prices) to satisfy the Nyquist relation ($\Delta k \Delta w = 2\pi/N$), where $N$ is the number of grid points. This connection creates a restrictive trade-off. Since $\Delta w$ needs to be low to achieve a high accuracy, simultaneously as $\Delta k$ needs to be low to avoid interpolation errors, this forces $N$ to massivly increase, which slows down the computation.

### 2.3.2 The COS Method

The COS method developed by Fang and Oosterlee (2008) is an extremely efficient numerical semi-analytical approach to valuing European options. The method is especially useful when the distribution of the underlying asset is not explicitly known but the characteristic function (Fourier transform of the PDF) can be computed analytically, as in the Heston model. The major innovation of the COS method is its ability to approximate the unknown PDF through a Fourier-cosine series expansion. Through the connection between the cosine series coefficients and the characteristic function, the valuation problem becomes an easy algebraic sum rather than a complex integration problem.

For pricing European options the risk-neutral valuation formula is the fundamental starting point. Let $x$ represent the log-moneyness of the underlying asset at the initial time $t_0$ ($x = \ln(S_0/K)$), and let $y$ represent the log-moneyness of the underlying asset at the maturity date $T$ ($y = \ln(S_T/K)$). The present value of the option, $V(x, t_0)$, is given by the discounted expected payoff under the risk-neutral measure:

$$V(t_0, x) = e^{-r\tau}\mathbb{E}^{\mathbb{Q}}\big[V(T, y) \mid x\big] = e^{-r\tau}\int_{\mathbb{R}} V(T, y) f(y \mid x)\, dy \tag{44}$$

Where:

- $r$ is the constant risk-free interest rate.

- $\tau = T - t_0$ is the time to maturity.
- $V(T, y)$ is the payoff function at maturity.
- $f(y \mid x)$ is the conditional probability density function of $y$ given the initial state $x$.

Because the true density function $f(y \mid x)$ decays rapidly toward zero in its extreme tails, the infinite integration domain $(-\infty,\infty)$ seen in Equation 44 can be truncated to a finite interval $[a, b]$ without introducing significant approximation error. The truncated valuation formula $V_1$ then becomes:

$$V_1(t_0, x) = e^{-r\tau} \int_a^b V(T, y) f(y \mid x)\, dy \tag{45}$$

Functions with finite support can be optimally approximated using Fourier-cosine series expansions (Oosterlee & Grzelak, 2019). The density function on the truncated interval is approximated by its cosine series expansion:

$$\hat{f}_X(y) = \sum_{k=0}^{+\infty}{}' \bar{A}_k(x)\, \cos\left(k\pi \frac{y-a}{b-a}\right) \tag{46}$$

The prime notation ($'$) indicates that the first term in the summation is multiplied by one-half. The coefficients of this expansion, $\bar{A}_k(x)$, are defined as:

$$\bar{A}_k(x) := \frac{2}{b-a} \int_a^b \hat{f}_X(y) \cos\left(k\pi \frac{y-a}{b-a}\right) dy \tag{47}$$

The core innovation of the COS method is that it recovers the cosine series coefficients directly from the characteristic function, bypassing the need for the unknown density (Fang & Oosterlee, 2008). The characteristic function $\phi_X(u)$ and the density function $f_X(y)$ form a Fourier pair (Oosterlee & Grzelak, 2019):

$$\phi_X(u) = \int_{\mathbb{R}} e^{iuy} f_X(y)\, dy \tag{48}$$

Assuming the truncated integration over $[a, b]$ provides a highly accurate approximation of the infinite integral, we have:

$$\int_a^b e^{iuy} f_X(y)\, dy \approx \phi_X(u) \tag{49}$$

By substituting the specific frequency argument $u = \frac{k\pi}{b-a}$ and multiplying both sides by $\exp\left(-ik\pi \frac{a}{b-a}\right)$, we obtain:

$$\phi_X\left(\frac{k\pi}{b-a}\right) \exp\left(-ik\pi \frac{a}{b-a}\right) \approx \int_a^b \exp\left(iy\frac{k\pi}{b-a} - ik\pi \frac{a}{b-a}\right) f_X(y)\, dy \tag{50}$$

Taking the real part $Re(\cdot)$ of both sides and utilizing Euler's formula ($e^{iu} = \cos(u) + i\sin(u)$) directly yields the definition of the cosine expansion coefficients:

$$\bar{F}_k(x) := \frac{2}{b-a} \operatorname{Re}\left[\phi_X\left(\frac{k\pi}{b-a}\right) \exp\left(-ik\pi \frac{a}{b-a}\right)\right] \tag{51}$$

Therefore, the unknown coefficients $\bar{A}_k(x)$ in (47) are accurately approximated by the analytically known values $\bar{F}_k(x)$. Now replacing the conditional density function $f(y \mid x)$ in the truncated valuation formula (45) with its Fourier-cosine expansion (46) gives:

$$V_1(t_0, x) = e^{-r\tau} \int_a^b V(y, T) \sum_{k=0}^{+\infty}{}' \bar{A}_k(x) \cos\left(k\pi \frac{y-a}{b-a}\right)\, dy \tag{52}$$

The cosine series coefficients of the payoff function, $V(y, T)$, are analytically known for standard options

as $H_k$:

$$H_k := \frac{2}{b-a}\int_a^b V(T,y)\cos\left(k\pi\frac{y-a}{b-a}\right)\,dy, \tag{53}$$

Using Fubini's theorem to interchange the summation and integration in (52) then inserting (53) yields:

$$V_1(t_0,x) = \frac{b-a}{2}e^{-r\tau}\sum_{k=0}^{\infty}{}'\bar{A}_k(x)\cdot H_k \tag{54}$$

Since the series coefficients rapidly decay, the infinite summation can be truncated to $N$ terms without significant loss in precision (Fang & Oosterlee, 2008). This together with substituting $\bar{A}_k(x)$ for the characteristic function approximation $\bar{F}_k(x)$ in Equation 51 gives the final COS formula for general underlying processes (Oosterlee & Grzelak, 2019):

$$V(t_0,x) \approx e^{-r\tau}\sum_{k=0}^{N-1}{}'\,\mathrm{Re}\Big\{\phi_X\left(\frac{k\pi}{b-a}\right)\exp\left(-ik\pi\frac{a}{b-a}\right)\Big\}\cdot H_k, \tag{55}$$

Note that the factor $\frac{b-a}{2}$ appearing in Equation 54 is not missing, it cancels exactly with the factor $\frac{2}{b-a}$ in the definition of $\bar{F}_k(x)$. Using the characteristic function for the Heston model from (Equation 37) Section 2.2.5 in the COS formula for general underlying processes gives:

$$V(t_0,x) \approx e^{-r\tau}\sum_{k=0}^{N-1}{}'\,\mathrm{Re}\Big\{\phi_{\text{Heston}}\big(\frac{k\pi}{b-a};\,x,v_0,\tau\big)\exp\left(-ik\pi\frac{a}{b-a}\right)\Big\}\cdot H_k, \tag{56}$$

Before this result can be used for pricing options the cosine series coefficient ($H_k$) needs to be recovered. This can be obtained analytically for plain European options. The payoff for European options under log-moneyness is given by:

$$V(T,y) \equiv \big[\alpha\cdot K(e^y-1)\big]^+ \quad \text{with} \quad \alpha = \begin{cases} 1 & \text{for a call,} \\ -1 & \text{for a put.} \end{cases} \tag{57}$$

To derive $H_k$ from its definition (52) the cosine series coefficients, $\chi_k$ for $g(y)=e^y$ on $[c,d]\subset[a,b]$ and $\psi_k$ for $g(y)=1$ on $[c,d]\subset[a,b]$ are needed.

$$\chi_k(c,d) := \int_c^d e^y\cos\left(k\pi\frac{y-a}{b-a}\right)\,dy, \qquad \psi_k(c,d) := \int_c^d \cos\left(k\pi\frac{y-a}{b-a}\right)\,dy. \tag{58}$$

Using (57) and (58) for a European option in the definition for $H_k$ (53) yields:

$$H_k^{call} = \frac{2}{b-a}\int_0^b K(e^y-1)\cos\left(k\pi\frac{y-a}{b-a}\right)\,dy = \frac{2}{b-a}K\big(\chi_k(0,b)-\psi_k(0,b)\big), \tag{59}$$

$$H_k^{put} = \frac{2}{b-a}K\big(-\chi_k(a,0)+\psi_k(a,0)\big). \tag{60}$$

Note that since integrating over the payoff function at maturity $V(T,y)$ the integral only needs to include the areas where the payoff is strictly larger than 0 (i.e., in-the-money). This result in the integration range being [0, b] for call options and [a, 0] for put options.

To ensure the utility of the COS method within the Heston framework, the truncation interval [a,b] must

be chosen with care to include the significant mass of the conditional density $f(y \mid x)$. The boundaries are determined using the cumulants $c_n$ of the log-asset price distribution:

$$a = c_1 - L\sqrt{c_2 + \sqrt{c_4}}, \quad b = c_1 + L\sqrt{c_2 + \sqrt{c_4}} \tag{61}$$

Where $c_1$ represents the mean, $c_2$ the variance of the process, $c_4$ the excess kurtosis (how much of the probability mass is located in the extreme tails of the distribution compared to a normal distribution) and $L$ is a truncation constant.

### 2.3.3 Monte Carlo Simulation

Another widely used numerical approach for option pricing is Monte Carlo simulation. The method has the capacity to accurately determine the prices of very complicated derivatives, in this case options (Glasserman, 2004). The fundamental idea is the use of the fact that the price of the derivative can be viewed as the expected discounted payoff of the option. This expectation is estimated by simulating random paths of changes in the price of the underlying asset over time, discounting the payoffs associated with each of them and then taking their average to estimate the option price. When applying Monte Carlo to the Heston model, the method typically relies on discrete-time approximations because the asset price and its variance evolve together as a complex system of stochastic differential equations.

One major problem with the Monte Carlo simulation is its slow convergence rate where the standard error decreases at a rate of $\mathcal{O}(N^{-1/2})$, meaning that to cut the pricing error in half, one must quadruple the number of simulated paths. Adding a single decimal place of precision requires 100 times as much computational effort. Another major problem is its difficulty for estimating Greeks. Calculating Greeks requires estimating the derivatives of the option price. Using finite-difference approximations (simulating the model at parameter value $\vartheta$ and $\vartheta + h$) introduces a difficult trade-off since a small h minimizes bias but inflates the variance (noise) of the estimate, while a large $h$ reduces variance but increases bias.

## 2.4 Model Calibration Theory and Surrogate Modeling Approaches

### 2.4.1 Calibration as an Optimization Problem

As discussed in the introduction, calibrating the Heston model means choosing parameter values such that the model reproduces observed market option data as closely as possible. This can be formulated more precisely as an inverse problem, usually written in the nonlinear least-squares form

$$\min_{\phi \in \Theta} \|r(\phi)\|^2 \tag{62}$$

(Cui et al., 2017). Here, $\phi \in \Theta \subset \mathbb{R}^m$ denotes the parameter vector to be calibrated, where $\Theta$ is the set of allowed parameter values and $m = 5$ indicates the dimension. $r(\phi)$ is the residual vector

$$r(\phi) = [r_1(\phi), r_2(\phi), \ldots, r_n(\phi)]^{\mathsf{T}}, \tag{63}$$

where $n$ is the number of options to be calibrated and

$$r_i(\phi) = V_i^{\text{model}}(\phi) - V_i^{\text{market}}. \tag{64}$$

In other words, the residuals are the differences between model-implied and observed market option prices. Since the Heston pricing model depends nonlinearly on the parameters, the resulting calibration

problem is a nonlinear least-squares problem (Cui et al., 2017). Unlike linear least-squares, nonlinear least-squares problems require the use of iterative optimization procedures (NIST/SEMATECH, 2012). In practice, this requires repeated pricing of a large number of options. As a result, calibration becomes computationally expensive, motivating the use of surrogate models.

### 2.4.2 Implied Volatility-based Calibration

Apart from prices, it is also common to calibrate the model using an implied volatility-based objective function. Both choices rely on the same underlying market information and the objective is still to match model outputs to observed market values. Hence, the residual in the IV domain can be written as

$$r_i(\phi) = \sigma_i^{\text{IV model}}(\phi) - \sigma_i^{\text{IV market}}. \tag{65}$$

An implied volatility surface provides a more compact representation of all vanilla option prices, which is one reason why implied volatility is often preferred in practice. A further motivation for calibrating in the IV domain is that it may be numerically advantageous, as it reduces the ratio between different Hessian entries compared to calibration in the price domain. This can improve both calibration efficiency and accuracy when using gradient-based calibration methods (Liu, Borovykh, et al., 2019). Such gradient-based calibration methods are discussed in more detail in the next section.

### 2.4.3 Gradient-Based Calibration and Parameter Greeks

A wide range of optimization procedures can be used for model calibration, and determining which one is the most suitable in this setting is not always straightforward. This has therefore become an important research topic within finance.

One group of methods consists of global optimization techniques. These are broadly applicable methods that do not require gradients of the objective function (C. Zhang et al., 2025). In the present setting this means the partial derivatives with respect to the model parameters, i.e., the parameter Greeks. As discussed by C. Zhang et al. (2025), another possible advantage of these techniques is that their convergence to the global optimum is also less sensitive to the starting values. However, this flexibility often requires a large number of iterations, making the search for an appropriate parameter set time-consuming.

Another group contains the multistart optimization methods. These methods search for local optima from a selected set of starting values and then choose the best candidate among them. To do this, many of these algorithms instead make use of the derivatives of the objective function, again in this case the parameter Greeks.

The calculation of the price derivatives w.r.t the parameters $\phi \in \{\kappa, \theta, \sigma, \rho, v_0\}$,

$$\frac{\partial P}{\partial \kappa}, \quad \frac{\partial P}{\partial \theta}, \quad \frac{\partial P}{\partial \sigma}, \quad \frac{\partial P}{\partial \rho}, \quad \frac{\partial P}{\partial v_0}$$

i.e., the price Greeks are discussed further in Section 2.8.1.

The implied volatility derivatives w.r.t the parameters $\phi \in \{\kappa, \theta, \sigma, \rho, v_0\}$,

$$\frac{\partial \sigma^{IV}}{\partial \kappa}, \quad \frac{\partial \sigma^{IV}}{\partial \theta}, \quad \frac{\partial \sigma^{IV}}{\partial \sigma}, \quad \frac{\partial \sigma^{IV}}{\partial \rho}, \quad \frac{\partial \sigma^{IV}}{\partial v_0}$$

i.e., the IV Greeks are derived in Section 2.5.4.

Since calibration is an iterative procedure in which the parameters are repeatedly updated until a suitable combination is found, it is valuable to understand how the option price changes with each individual parameter. Knowing the gradient therefore provides information about which parameters should be adjusted and in which direction. This results in an accelerated convergence for the methods using them (C. Zhang et al., 2025).

### 2.4.4 One-Step Deep Calibration and Two-Step Surrogate Pricing

In Heston calibration, neural networks may be used as surrogate models. According to Sridi and Bilokon (2023), such neural network-based calibration methods can be divided into two main approaches.

The first approach, often referred to as the "one-step approach" or Deep Calibration, was developed by Hernandez (2016), where an Artificial Neural Network (ANN) was developed to directly map observed market data to calibrated model parameters. It is for example possible to learn a direct mapping from an implied volatility surface to the corresponding model parameters. Since no optimization step is necessary, this would be much faster than using a neural network as a pricing surrogate. However, this approach is associated with several challenges and might generalize poorly for unseen market regimes. In particular, the corresponding mapping from parameters to market data is non-injective, meaning that several different parameter combinations may generate the same prices or implied volatility surface. This often induces instability when combined with measurement noise, as small changes in the inputs can lead to drastically different model parameters, making the calibration problem inherently ill-posed (Gambara & Teichmann, 2021).

The second approach, known as the "two-step approach", is more commonly adopted. In this case, deep learning is first applied as a surrogate model for approximating either implied volatilities or option prices. The second step is then to calibrate the model parameters. As described above, this is done using traditional optimization techniques to minimize the errors generated by the neural network (Gambara & Teichmann, 2021; Sridi & Bilokon, 2023). The "two-step approach" has, for example, been used by Liu, Oosterlee, and Bohte (2019) to calibrate stochastic volatility models, such as the Heston and Bates models.

## 2.5 Implied Volatility Computation

### 2.5.1 Numerical Inversion of the Black–Scholes Formula

Recovering implied volatility from option prices is one of the most common computational tasks in finance (Choi et al., 2024). As noted above, within the Black–Scholes framework, implied volatility is defined as the value of the volatility parameter for which the model price matches the observed market price (Hull, 2022). This can be expressed as

$$\sigma^{\mathrm{IV}}(m,\tau) = \left(V^{\mathrm{BS}}\right)^{-1}\left(V^{\mathrm{mkt}}; S, K, \tau, r\right) \tag{66}$$

where $\left(V^{\mathrm{BS}}\right)^{-1}$ denotes the inverse Black–Scholes function, $m = S_t/K$ denotes the moneyness, and $\tau$ is the time to maturity (Liu, Oosterlee, & Bohte, 2019).

In model-based settings, one may instead compute the corresponding implied volatility from a model-generated price. This is justified since the purpose of the model is to reproduce observed market prices as closely as possible (Cui et al., 2017). For example, Liu, Oosterlee, and Bohte (2019) do this for Heston-generated prices by first computing option prices under the Heston model and then extracting

the corresponding implied volatilities. If $\sigma_H^{IV}$ denotes this implied volatility, it is defined by

$$\sigma_H^{\mathrm{IV}}(m,\tau) = \left(V^{\mathrm{BS}}\right)^{-1}\left(V^{\mathrm{H}}; S, K, \tau, r\right) \tag{67}$$

Hence, in model-based settings, the following definitions and reasoning apply as above, with $V^{\mathrm{mkt}}$ replaced by $V^{\mathrm{H}}$.

In practice, this inversion problem does not admit a closed-form analytical solution. The implied volatility must therefore be determined numerically by means of an iterative approximation algorithm. This is achieved by reformulating the equation as a root-finding problem:

$$g\left(\sigma^{\mathrm{IV}}\right) = V^{\mathrm{BS}}\left(S, \tau, K, r, \sigma^{\mathrm{IV}}\right) - V^{\mathrm{mkt}}(K,T) = 0 \tag{68}$$

Several numerical approaches are available for solving this equation. Among them are the Newton-Raphson method, Brent's method, and Let's Be Rational (Liu, Oosterlee, & Bohte, 2019). Each of these is discussed in more detail below.

### 2.5.2 The Newton-Raphson and Brent Methods

Manaster and Koehler (1982) describe the Newton-Raphson method as a common numerical method for solving nonlinear equations. Applied to the implied volatility problem, they show that the iteration can be written as

$$\sigma_{k+1}^{\mathrm{IV}} = \sigma_k^{\mathrm{IV}} - \frac{V^{\mathrm{BS}}(\sigma_k^{\mathrm{IV}}) - V^{\mathrm{mkt}}}{g'(\sigma_k^{\mathrm{IV}})}, \qquad k = 0, \ldots \tag{69}$$

where $\sigma_k^{\mathrm{IV}}$ denotes the k-th iterate and $g'(\sigma_k^{\mathrm{IV}})$ the Greek called Vega. Since Vega is positive, the iteration above is well defined.

Newton-Raphson theory implies that if Equation 68 has a solution, then there exists an open interval $(a, b)$ around the root such that, if an iterate enters this interval, the sequence converges to the root.

For the present application, the initial guess is chosen as

$$\sigma_0^{\mathrm{IV}} = \sqrt{\frac{2}{\tau}|m + r\tau|} \tag{70}$$

Manaster and Koehler (1982) further show that this starting value lies within the interval $(a, b)$ containing the implied volatility root, which ensures convergence of the iteration. This is supported using that if $\sigma^{\mathrm{IV}} < \sigma_0^{\mathrm{IV}}$, Vega is increasing on the interval, whereas if $\sigma^{\mathrm{IV}} > \sigma_0^{\mathrm{IV}}$, Vega is decreasing. The iterations therefore approach $\sigma^{\mathrm{IV}}$ monotonically, downward in the former case and upward in the latter. Since the sequence is monotone and bounded, it converges to the unique implied volatility.

Although simple to implement, the naive Newton-Raphson method is not the most efficient one and can converge very slowly for out-of-the-money options with very low prices (Choi et al., 2024). The method may also fail when Vega becomes very small. In the Black–Scholes equation, the Vega can become very close to zero especially when the option is deep in-the-money (ITM) or deep out-of-the-money (OTM), making the division by Vega numerically unstable. As an alternative, hybrid methods combining Newton-Raphson and bisection have been suggested (Liu, Oosterlee, & Bohte, 2019). Choi et al. (2024) instead propose a hybrid method based on the log price.

Another derivative-free and reliable alternative for computing the implied volatility is Brent's method. This algorithm is a combination of bisection, inverse quadratic interpolation and the secant method (Liu, Oosterlee, & Bohte, 2019). Given three distinct iterates, $\sigma_k^{\mathrm{IV}}$, $\sigma_{k-1}^{\mathrm{IV}}$, and $\sigma_{k-2}^{\mathrm{IV}}$, the method first attempts

inverse quadratic interpolation, meaning that it approximates

$$\begin{aligned}\sigma_{k+1}^{\mathrm{IV}} = &\frac{\sigma_k^{\mathrm{IV}}\, g(\sigma_{k-1}^{\mathrm{IV}})\, g(\sigma_{k-2}^{\mathrm{IV}})}{\big(g(\sigma_k^{\mathrm{IV}}) - g(\sigma_{k-1}^{\mathrm{IV}})\big)\big(g(\sigma_k^{\mathrm{IV}}) - g(\sigma_{k-2}^{\mathrm{IV}})\big)} \\ &+ \frac{\sigma_{k-1}^{\mathrm{IV}}\, g(\sigma_{k-2}^{\mathrm{IV}})\, g(\sigma_k^{\mathrm{IV}})}{\big(g(\sigma_{k-1}^{\mathrm{IV}}) - g(\sigma_{k-2}^{\mathrm{IV}})\big)\big(g(\sigma_{k-1}^{\mathrm{IV}}) - g(\sigma_k^{\mathrm{IV}})\big)} \\ &+ \frac{\sigma_{k-2}^{\mathrm{IV}}\, g(\sigma_{k-1}^{\mathrm{IV}})\, g(\sigma_k^{\mathrm{IV}})}{\big(g(\sigma_{k-2}^{\mathrm{IV}}) - g(\sigma_{k-1}^{\mathrm{IV}})\big)\big(g(\sigma_{k-2}^{\mathrm{IV}}) - g(\sigma_k^{\mathrm{IV}})\big)}.\end{aligned} \tag{71}$$

If two consecutive approximations are identical, for example if $\sigma_k^{\mathrm{IV}} = \sigma_{k-1}^{\mathrm{IV}}$, Brent's method instead uses linear interpolation, i.e., the secant method. This estimates the next iterate from the straight line through the two previous points (Liu, Oosterlee, & Bohte, 2019):

$$\sigma_{k+1}^{\mathrm{IV}} = \sigma_{k-1}^{\mathrm{IV}} - g(\sigma_{k-1}^{\mathrm{IV}}) \frac{\sigma_{k-1}^{\mathrm{IV}} - \sigma_{k-2}^{\mathrm{IV}}}{g(\sigma_{k-1}^{\mathrm{IV}}) - g(\sigma_{k-2}^{\mathrm{IV}})}. \tag{72}$$

If the interpolation step produces a root candidate that is not considered suitable, the algorithm falls back on bisection, meaning it takes the midpoint of the current bracketing interval. This preserves the bracketing of the root and ensures convergence (Brent, 1973).

### 2.5.3 The "Let's Be Rational" Algorithm

Another prominent method for computing implied volatility is the *Let's Be Rational* algorithm proposed by Jäckel (2015). Liu, Oosterlee, and Bohte (2019) in short describe it as a way of selecting a more suitable initial guess in order to reduce the risk of divergence in the Newton-Raphson method, while Choi et al. (2024) identify it as one of the most effective methods for implied volatility computation.

In contrast to the methods described above, which are applied directly to the Black–Scholes formula, this algorithm first reformulates the inversion problem using the equivalent Black formula. The Black formula can be viewed as the forward–price version of Black–Scholes, where the spot price $S$ is replaced by the forward price $F$, and hence the log-moneyness $x = \ln(F/K)$. The algorithm further introduces the normalizations $\tilde{\sigma} = \sigma^{IV}\sqrt{\tau}$ and $V^b(x, \tilde{\sigma}, \varrho) = V^B(F, K, \sigma^{IV}, \tau, \varrho)/\sqrt{FK}$ where $B$ denotes the Black formula and $\varrho \in \{-1, +1\}$ denotes the option type (put or call). The market price, $V^{mkt}$, is then also expressed through a corresponding normalized forward price $\beta$. By doing these changes, both numerical stability and computational efficiency is improved, especially for options with very low or very high prices.

A key idea of the method is that, for a fixed $x = ln(F/K)$, the normalized Black function $b(x, \tilde{\sigma})$ is monotone increasing in the normalized implied volatility, $\tilde{\sigma}$, and has a single inflection point. This point is denoted by $\tilde{\sigma}_c = 2|x|$, and defines $b_c = b(x, \tilde{\sigma}_c)$. Using the tangent at the inflection point, two additional boundary values, $b_l$ and $b_u$, are introduced. This divides the normalized price domain into four regions, $[0, b_l]$, $[b_l, b_c]$, $(b_c, b_u]$, and $(b_u, b_{\max}]$, where $b_{\max}$ denotes the maximum attainable normalized option price. Based on this partition, a piecewise initial guess $\tilde{\sigma}_0(\beta)$ is constructed, reflecting the fact that the inverse function behaves differently across the price domain.

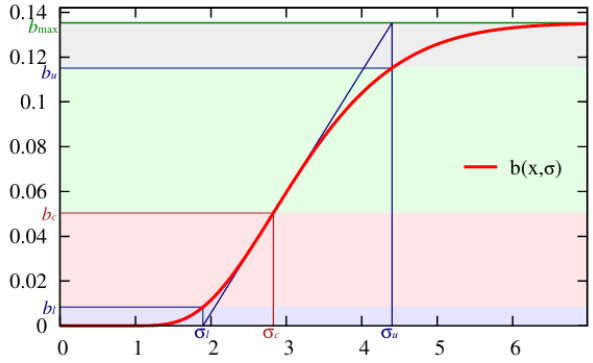



Figure 1: The construction of the four price intervals, using the tangent at the inflexion point, $(\sigma_c, \tilde{\sigma}_c)$ for $|x| = 4$ (Jäckel, 2015).

In the two middle intervals, $[b_l, b_c]$ and $(b_c, b_u]$, the inverse function is sufficiently smooth to be approximated accurately by two separate rational cubic interpolations, i.e., using a function with polynomial terms up to degree three. In the lower and upper intervals, the approximation is instead constructed for transformed functions motivated by the asymptotic behavior near the price boundaries.

Once the initial guess has been selected from the interval containing $\beta$, the root of the function $g(\tilde{\sigma})$ is once again determined iteratively. To improve convergence Jäckel also reformulates the function w.r.t. the different price intervals. The resulting objective function is therefore based on transformed residuals in the lower and upper region:

$$g(\tilde{\sigma}) = \begin{cases} \dfrac{1}{\ln\big(b(x,\tilde{\sigma})\big)} - \dfrac{1}{\ln(\beta)}, & \beta \in [0, b_l), \\ b(x,\tilde{\sigma}) - \beta, & \beta \in [b_l, \tilde{b}_u], \\ \ln\left(\dfrac{b_{\max} - \beta}{b_{\max} - b(x,\tilde{\sigma})}\right), & \beta \in (\tilde{b}_u, b_{\max}], \end{cases} \qquad \text{where} \qquad \tilde{b}_u := \max\left(b_u, \frac{b_{\max}}{2}\right). \tag{73}$$

The iterations are then carried out using the third-order Householder method:

$$\tilde{\sigma}_{k+1} = \tilde{\sigma}_k + \nu_k \frac{1 + \frac{1}{2}\gamma_k \nu_k}{1 + \nu_k\big(\gamma_k + \frac{1}{6}\delta_k \nu_k\big)}, \tag{74}$$

where

$$\nu_k := -\frac{g(\tilde{\sigma}_k)}{g'(\tilde{\sigma}_k)}, \qquad \gamma_k := \frac{g''(\tilde{\sigma}_k)}{g'(\tilde{\sigma}_k)}, \qquad \delta_k := \frac{g'''(\tilde{\sigma}_k)}{g'(\tilde{\sigma}_k)}. \tag{75}$$

The first-order version, i.e., the version using only $g'(\tilde{\sigma})$, is identical to the Newton-Raphson iteration.

The combination of both an interval-based initial guess and function $g(\tilde{\sigma})$, as well as using the third-order Householder method makes the algorithm both fast and robust. Jäckel shows that, with only two iterations, *Let's Be Rational* can reach machine precision for all admissible inputs. At the same time, the maximum attainable accuracy still depends strongly on having a Black function that near the solution is numerically stable down to the same relative accuracy. Although this issue is not unique to this method, a sufficiently stable Black implementation is required in order to realize the full advantage of the method in practice. With such an implementation, *Let's Be Rational* is well suited to compute implied volatilities both accurately and efficiently over a wide range of strikes and maturities.

### 2.5.4 Implied Volatility Greeks

In order to understand how the Heston parameters affect implied volatility, it is useful to also compute the implied volatility derivatives (Liu, Borovykh, et al., 2019). These IV Greeks can be derived from the definition of implied volatility within model-based settings, i.e., the volatility for which the Black–Scholes price equals the Heston price. Following Liu, Oosterlee, and Bohte (2019) it is defined by

$$V^{BS}(S, K, \tau, r, \sigma_H^{IV}) = V^H(S, K, \tau; \kappa, \theta, \sigma, \rho, v_0). \tag{76}$$

Since all other inputs in the Black–Scholes formula are held fixed, $\sigma_H^{IV}$ can be viewed as a function of the Heston parameters. Differentiating both sides with respect to a Heston parameter $\phi \in \{\kappa, \theta, \sigma, \rho, v_0\}$ and applying the chain rule yields

$$\frac{\partial V^{BS}}{\partial \sigma_H^{IV}} \frac{\partial \sigma_H^{IV}}{\partial \phi} = \frac{\partial V^H}{\partial \phi}. \tag{77}$$

Hence, each implied volatility Greek is given by:

$$\frac{\partial \sigma_H^{IV}}{\partial \phi} = \frac{\partial V^H / \partial \phi}{\partial V^{BS} / \partial \sigma_H^{IV}}, \tag{78}$$

where the numerator is the Heston price Greek and the denominator is the Black–Scholes Vega.

## 2.6 High-Dimensional Sampling Methods

Ordinary random (Monte Carlo) sampling draws each point independently from a uniform distribution. In low dimensions this works reasonably well, but in high dimensions the points tend to clump together in some areas while leaving big empty voids in others (Owen, 2023). This problem grows worse as the number of dimensions increases, since the volume of the dimensional space explodes and random points have a hard time spreading out evenly. As a result, parts of the input space are overcrowded, while others are ignored, leading to problems when sampled input labels in a high-dimensional space are to be used for training a model (Forrester et al., 2008).

### 2.6.1 Curse of Dimensionality

In the context of high-dimensional parameter sampling, the curse of dimensionality refers to the fundamental breakdown of traditional statistical and geometric intuition as the number of dimensions for data points increases (Peng et al., 2024). It is a collection of effects that can dampen the performance of machine learning models.

As the dimensionality $d$ increases, the volume of the parameter space expands exponentially. Figure 2 explains this phenomenon in a simple manner. Here, each parameter is divided into 5 intervals and the sample density for each dimension is calculated as $10/5^d$. We see that the density converges to 0 as the dimension goes to infinity and that the density already becomes close to 0 for dimensions bigger than 5. When the number of samples is much smaller than $5^d$ it means that most of the parameter intervals do not include samples. This shows the often inability to represent the entire parameter space for data samples in high-dimensional spaces, leaving empty space (Peng et al., 2024).

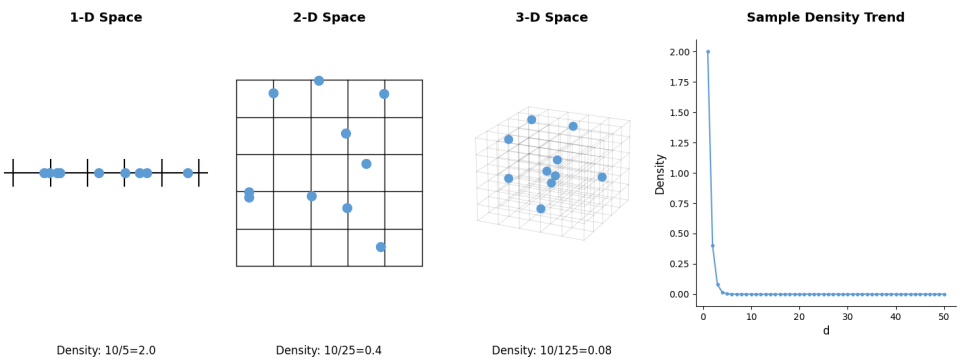



Figure 2: Visualization of how the density for 10 random uniformly distributed data points decreases as the number of dimensions increases. Each dimension is divided into 5 intervals.

The empty space that occurs for small datasets in high-dimensional spaces can complicate machine learning ability to generalize from training samples to unseen data. When the space is mostly empty, models trained on high-dimensional data is likely to overfit since the model finds "patterns" that actually do not exist in the huge gaps between sparse data points (Verleysen & François, 2005). This results in a model that performs perfectly on the training data, but fails to generalize well to previously unseen data points.

### 2.6.2 Latin Hypercube Sampling

Latin Hypercube Sampling (LHS) is a random sampling technique that ensures coverage of multidimensional parameter spaces. It was introduced by McKay et al. (1979) as a constrained sampling technique designed to analyze computer outputs more efficiently than simple random sampling. LHS ensures that each input variable is sampled throughout its distribution to obtain a good spread by first dividing the range of each of the $k$ input labels into $n$ intervals with equal probability $(1/n)$. One value is then selected at random within each of the $n$ intervals for every input label. These $n$ values for each input label are then randomly permuted to form $n$ sampled data-points in the space. This ensures that for each individual dimension, there is exactly one sample point per interval, effectively reducing the Curse of Dimensionality.

### 2.6.3 Sobol Sequences

Sobol sampling can be described as filling in the previously unsampled regions in a multidimensional space when a new label is sampled (Nicholson, n.d.). It generates a uniform distribution in the probability space. In a two dimensional sense, loosely speaking, imagine putting dots on a paper sheet, where each new dot is put as far away as possible from the earlier dots, that is, in the current most empty space.

For each dimension $j = 1, 2, \ldots, d$ a number of direction vectors $v_{j,i}$ (binary fractions ranging from 0 to 1) are pre-defined (Sobol' et al., 2011). To obtain the sequence corresponding to a certain dimension $j$, the algorithm utilizes a primitive polynomial from GF(2) which means that the field contains only two elements 0 and 1. The primitive polynomial of degree $s_j$ takes the following form:

$$P_j(x) = x^{s_j} + a_{1,j}x^{s_j-1} + \cdots + a_{s_j-1,j}x + 1. \tag{79}$$

where $a_{k,j}$ are the coefficients of the polynomial. Given these polynomials, a sequence of integer values

($m_{i,j}$, which determine the direction numbers for $i > s_j$) is obtained using the recurrence relation:

$$m_{i,j} = 2a_{1,j}m_{i-1,j} \oplus 2^2 a_{2,j}m_{i-2,j} \oplus \cdots \oplus 2^{s_j} m_{i-s_j,j} \oplus m_{i-s_j,j} \tag{80}$$

where the symbol $\oplus$ denotes the bit-wise XOR operation, which is equivalent to the addition modulo 2. That is, ($0 \oplus 0 = 0$, $1 \oplus 1 = 0$, and $0 \oplus 1 = 1$).

The first set of values to start this recurrence relation $(m_{1,j}, m_{2,j}, \ldots, m_{s_j,j})$ can be selected freely, as long as $m_{k,j} < 2^k$ and $m_{k,j}$ an odd number. This flexibility in choosing initial numbers is why many different Sobol sequences can be constructed for the exact same dimension. The direction numbers are then obtained as the binary fractions:

$$v_{j,i} = \frac{m_{i,j}}{2^i}. \tag{81}$$

The direction numbers can be seen as acting like a fixed set of "step sizes" unique to each dimension. By combining them appropriately, the generator can produce points that respect the natural power-of-two subdivisions of the unit interval (halves, quarters, eighths, etc.). To calculate the coordinates of the $n$-th point $X_n = (x_{1,n}, x_{2,n}, \ldots, x_{d,n})$ for the d-dimensional space, it is necessary to represent the number n as its binary representation n $= (b_b \ldots, b_2, b_1)$, where each $b_i$ can be only 0 or 1. The computation of the $j$-th coordinate is then done by computing the bitwise XOR operation among the direction numbers chosen by the binary digits of $n$:

$$x_n^j = b_1 v_1^j \oplus b_2 v_2^j \oplus \cdots \oplus b_b v_b^j, \tag{82}$$

This formula is applied independently to every dimension $j$ (from 1 to $d$) using the same binary digits of $n$ but different direction numbers $v_{j,i}$. Adding these coordinates for each dimension together in a coordinate vector then gives the next full sampled data point. This makes the Sobol sequence extremely fast at generating uniform distributions since it relies on straightforward binary operations rather than complicated mathematical equations.

### 2.6.4 Discrepancy Measures

Discrepancy is an objective measure of how far a set of points is from being perfectly uniformly distributed. If you scatter points in a specific space (like a square or a multidimensional hypercube), a perfectly uniform distribution means that the number of points falling into any given sub-region is exactly proportional to the volume of that sub-region. Discrepancy measures the "error" or difference between this ideal density and the actual point density (Niederreiter, 1992).

The $L_2$ discrepancy is a measure of how irregular the distribution of a finite set of points $\mathcal{P}$ within the unit square $[0,1)^2$ is (Faure et al., 2008). It evaluates how much the distribution of the points deviates from the uniform distribution by considering the squared deviation integrated over the entire space. The $L_2$ discrepancy of a point set ($\mathcal{P} = x_1, \ldots, x_N$) with $N \geq 1$ points is defined as:

$$L^2(\mathcal{P}) := \left( \int_0^1 \int_0^1 |E(x, y, \mathcal{P})|^2 \, dx \, dy \right)^{1/2} \tag{83}$$

where the discrepancy function $E(x, y, \mathcal{P})$ is given by:

$$E(x, y, \mathcal{P}) = A([0, x) \times [0, y), \mathcal{P}) - Nxy. \tag{84}$$

$A([0, x) \times [0, y)$ represents the number of points in $\mathcal{P}$ contained within the rectangle $[0, x) \times [0, y)$, while $Nxy$ refers to the area of this rectangle and represents the expected number of points in the rectangle for a uniform distribution. This definition uses the $L_2$-norm (root mean square) of the local discrepancy

function $E(x, y, \mathcal{P})$ over the unit square. It therefore provides an average-case measure of uniformity rather than focusing on the single worst-case deviation.

## 2.7 Neural Networks for Function Approximation

### 2.7.1 Feedforward Neural Networks

A neural network (NN) is a collection of connected artificial neurons used to approximate an algorithm or function. Each neuron represents an output function, while the connections between the neurons represent weights. The output of the neural network will therefore vary depending on the weights between neurons, the output functions of the neurons, as well as the structure of the network (Wu & Feng, 2018).

The simplest and most common form of a neural network is a feedforward neural network, in which information only traverses the network in one direction. The information will then propagate from the input layer to the output layer, usually through one or more hidden layers (Islam, 2019). In other words, the output of neurons in the $n^{\text{th}}$ layer is used as input to neurons in the $(n+1)^{\text{th}}$ layer. Figure 3 provides a visual representation of a feedforward neural network. If the network consists of at least one hidden layer, it is usually referred to as a deep feedforward neural network, or a multilayer perceptron (Goodfellow et al., 2016).

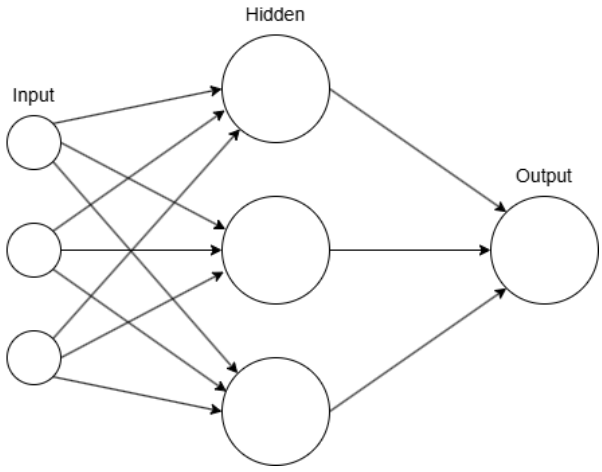



Figure 3: A visual representation of a feedforward neural network with three input neurons, one hidden layer containing three neurons, and one output neuron.

In addition to weights and input signals from the previous layer, a neuron may also have a bias, which is a constant applied to offset the function, and an activation function $\Phi$. The activation function is what gives the neural network the ability to learn and recognize complex mappings from the input data, by introducing non-linearity to the model. It can also be used to limit the range of the output signal (Sharma et al., 2020).

The output of a neuron can be written mathematically as:

$$y = \Phi\left(\sum_{i=1}^{N} \mathbf{w^{(i)}}\mathbf{x^{(i)}} + b\right), \tag{85}$$

where $\mathbf{x^{(i)}}$ are the input signals from the previous layer, $\mathbf{w^{(i)}}$ are the corresponding weights and b is the bias term (Z. Zhang, 2018). The expression inside the parentheses represents a linear transformation of

the inputs. The values $\mathbf{w}$ and $\mathbf{b}$ are commonly referred to as the trainable parameters of the network, denoted by $\omega$. These should not be confused with the parameters of the Heston model, $\phi$, which are used as inputs to the network in this study.

### 2.7.2 Universal Approximation Theorem

As mentioned earlier, neural networks are used for approximating functions. The Universal Approximation Theorem gives a theoretical motivation for the use of neural networks as function approximators. Cybenko (1989) showed that single-hidden-layer neural networks of the form

$$G(x) = \sum_{j=1}^{N} \alpha_j \Phi(\mathbf{w_j}^\top \mathbf{x} + b_j) \tag{86}$$

are dense in $C(I_n)$, where $\mathbf{x} \in \mathbb{R}^n$ is the input vector, $\mathbf{w_j}$ is the input weight vector to hidden unit $j$, $b_j$ is its bias term, $\Phi$ is a continuous activation function, and $a_j$ is the weight connecting the hidden unit $j$ to the output unit. Here, $I_n$ denotes the $n$-dimensional unit cube, $[0,1]^n$, and $C(I_n)$ denotes the space of continuous functions on $I_n$. In other words, the theorem states that neural networks of the above form can approximate any continuous function defined on $I_n$ with arbitrary precision, provided that the network contains sufficiently many neurons.

While the theorem is often stated for functions $f_\omega : \mathbb{R}^n \to \mathbb{R}$, the same idea can be extended componentwise for functions $f_\omega : \mathbb{R}^n \to \mathbb{R}^m$. The theorem also shows that a single hidden layer is sufficient for universal approximation, but newer research indicates that deeper architectures can be more effective for representing certain functions (Poggio et al., 2017).

Even though neural networks can approximate any arbitrary function, they largely favor those of lower complexity. This phenomenon is called *spectral bias* (Rahaman et al., 2019). Due to networks fitting lower complexity components first and therefore residing in the concept class of low complexity, spectral bias strongly promotes generalization. In particular, evaluating Fourier-based functions has proven that lower frequencies are learned first (Cao et al., 2019).

### 2.7.3 Activation Functions

The activation functions in neural networks are used to transform input signals into output signals, which are then supplied as inputs to the next layer. If the activation functions used in a neural network are either undefined or all linear, the network would not be able to learn non-linear mappings between the inputs and outputs, and would function like a linear regression model. Therefore, non-linear activation functions are commonly used for approximating complex and non-linear functions (Sharma et al., 2020).

One of the most widely used activation functions for neural networks is *ReLU*, which stands for Rectified Linear Unit. It can be defined mathematically as:

$$\text{ReLU}(x) = \max(0, x) \tag{87}$$

Figure 4 illustrates the shape of the ReLU activation function.

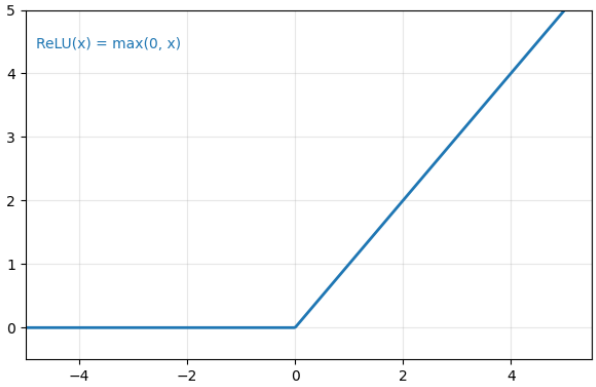



Figure 4: ReLU Activation Function

Although ReLU is popular due to its simplicity and efficiency, it is not continuously differentiable and can therefore be less suitable in settings where smooth gradients of the network outputs are required (Huge & Savine, 2020). Some of the alternative activation functions that resolve this issue are *SiLU* and *Softplus*.

SiLU, also called *Swish*, is expressed mathematically as:

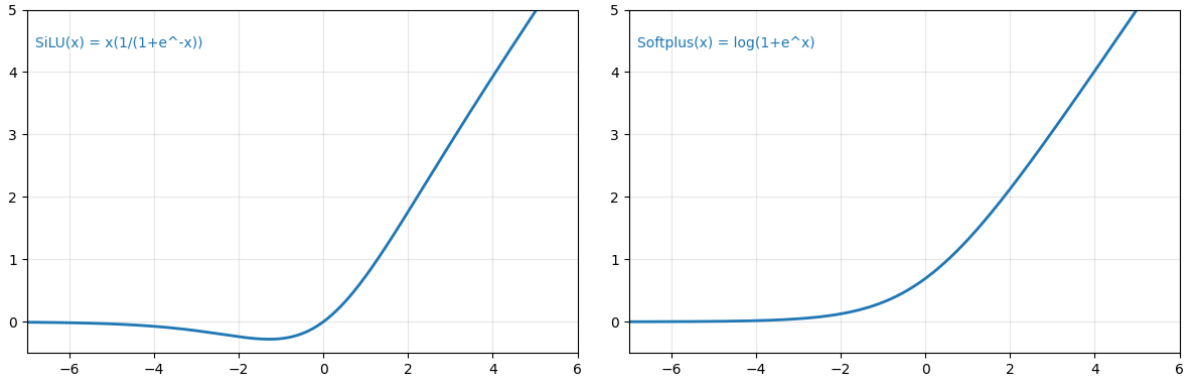
$$\text{SiLU}(x) = \frac{x}{1 + e^{-x}} \tag{88}$$

(Ramachandran et al., 2017), and is illustrated in Figure 5a.

The Softplus function, introduced by Dugas et al. (2000), can be described mathematically as:

$$\text{Softplus}(x) = \ln(1 + e^{x}). \tag{89}$$

Figure 5b shows the shape of the Softplus function. An important difference between SiLU and Softplus is that Softplus limits the range of the output of a neuron to non-negative values. This can be beneficial when approximating target functions that are non-negative by construction, such as option pricing functions.




(a) SiLU Activation Function

(b) Softplus Activation Function

Figure 5: Two common smooth alternatives to the ReLU activation function. Panel (a) shows the SiLU activation function, while panel (b) shows the Softplus activation function.

### 2.7.4 Loss Functions and Gradient-Based Optimization

The training of a neural network is a process consisting of adjusting the weight and bias parameters of the network in order to minimize some error function, also called a loss function. A loss function is a function that measures the error between the predicted output of the network and the desired target output (Bishop, 1994). Some of the most common forms of loss functions are *square loss functions*, which measure the square error between the predicted and target output (Q. Wang et al., 2022). One such function is the mean squared error (MSE) function, defined here as:

$$\mathcal{L}_{\text{MSE}}(\omega) = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}^{(i)} - y^{(i)})^2, \tag{90}$$

where $y^{(i)}$ is the target output, $\hat{y}^{(i)} = f_\omega(\mathbf{x}^{(i)})$ is the corresponding predicted output given an input vector $\mathbf{x^{(i)}}$ and the current network parameters $\omega$, and $N$ is the number of data points evaluated.

A common approach to minimizing the loss function is *gradient descent*. In gradient descent, the parameters are iteratively updated in the direction of the negative of the gradient of the loss function with respect to the parameters, $\nabla_\omega \mathcal{L}(\omega_t)$, where $t$ denotes the current step in the iterative process (Bishop, 1994). Mathematically, this process can be expressed as:

$$\omega_{t+1} = \omega_t - \eta \nabla_\omega \mathcal{L}(\omega_t), \tag{91}$$

where $\eta$ is called the *learning rate*, determining how aggressively the parameters are updated each iteration. $\eta$ can be adjusted according to some predefined schedule during training, which may improve convergence (Loshchilov & Hutter, 2017).

*Backpropagation*, introduced by Rumelhart et al. (1986), is used to efficiently compute the gradients needed for gradient descent by applying the chain rule recursively from the output layer backward through the network.

### 2.7.5 Optimization Algorithms

The algorithms used for minimizing the loss function are referred to as optimization algorithms. Gradient descent is a common example of such an algorithm. However, more advanced optimization methods are often used in practice.

One popular algorithm is *Adam*, introduced by Kingma and Ba (2015), which adaptively adjusts the update steps for each parameter based on the first and second moments of the gradients. Here, the first moment refers to the exponential moving average of the gradient, while the second moment refers to the exponential moving average of the squared gradient. We can describe the Adam optimization algorithm mathematically as:

$$\begin{aligned}
g_t &= \nabla_\omega \mathcal{L}(\omega_t) \\
m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \\
v_t &= \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \\
\hat{m}_t = \frac{m_t}{1-\beta_1^t}, &\qquad \hat{v}_t = \frac{v_t}{1-\beta_2^t} \\
\omega_{t+1} &= \omega_t - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}
\end{aligned} \tag{92}$$

where $m_t$ is the first moment estimate, $v_t$ is the second moment estimate, and $\beta_1, \beta_2 \in [0, 1)$ are the

corresponding exponential decay rates. However, since the moving averages are initialized as 0 in Adam, the early estimates will be systematically biased towards zero. To correct for this, the bias-corrected estimates $\hat{m}_t$ and $\hat{v}_t$ are calculated.

### 2.7.6 Weight Initialization

Since training neural networks is an iterative process, we need to specify some initial values for the parameters that are fitted during the process, primarily the weights. How to determine these initial weights is an area of active research (Kumar, 2017), but we know that the choice is critical for whether or not the algorithm converges, how fast it converges, as well as for its generalization capabilities (Goodfellow et al., 2016).

An important requirement for the initial parameters is that they must be chosen so as to break symmetry between different neurons in the beginning of the learning process. Otherwise, if several neurons are initialized with identical parameters, they will receive identical updates throughout training. In practice, this is achieved by initializing the weights to randomly sampled values from either a Gaussian distribution $\mathcal{N}(0, \sigma^2)$ or a uniform distribution $\mathcal{U}(a, b)$ (Goodfellow et al., 2016).

The question then becomes how $\sigma^2$ or $a$ and $b$ should be chosen, depending on the sample distribution. *Xavier initialization*, proposed by Glorot and Bengio (2010), has been shown to perform very well in many scenarios (Kumar, 2017). For ReLU activation functions, *Kaiming initialization*, introduced by He et al. (2015), is often preferred. Since ReLU suppresses negative inputs, this initialization uses a larger variance than Xavier to compensate for the loss of negative activations.

### 2.7.7 Parameter Normalization

From observation, the scale of labels carries over to the gradients of the cost functions and the size of gradient descent optimization steps. To avoid manual scaling of learning rate; gradient descent and variants are best implemented with normalized labels. Therefore, training deep learning models always starts with some sort of normalization step and ends with an "un-normalization step", where predictions are scaled back to original units (Huge & Savine, 2020).

An explicit mathematical justification for parameter scaling relates to the condition number of the Hessian matrix $H$, which can be calculated as:

$$\kappa(H) = \max_{i,j} \left| \frac{\lambda_i}{\lambda_j} \right| \tag{93}$$

where $\lambda_i$ and $\lambda_j$ represent the eigenvalues of the Hessian. This is the ratio of magnitude of the largest and smallest eigenvalue. A large condition number results in the matrix inversion being particularly sensitive to error in the input. Thus, by compressing parameter ranges to e.g. [-1, 1], the eigenvalues of the Hessian, which dictate the curvature of the loss landscape, converge closer to each other. This reduces the condition number, ensuring that first-order optimization methods like gradient descent navigate the loss surface more efficiently without oscillating (Goodfellow et al., 2016, Section 4.2.2).

Furthermore, when training neural networks, three specific criteria should be accounted for: 1. the average of each input variable over the training set should be close to zero, 2. input variables should be scaled such that their covariances are the same and 3. input variables should be uncorrelated if possible. One benefit of following these guidelines is that, geometrically, if inputs are shifted or scaled improperly, the weights are pushed into the flat regions of certain non-linear, self-gating activation functions (e.g.

sigmoid or SiLU), destroying the gradient (LeCun et al., 1998).

One common method for parameter scaling is the min-max scaling method. Min-max normalization performs a linear transformation on the original data, yielding the following formula:

$$x_{\text{scaled}} = \frac{x - x_{\min}}{x_{\max} - x_{\min}}, \tag{94}$$

where $x$ is the original input parameter and $x_{\min}$ and $x_{\max}$ denote the empirical minimum and maximum, respectively, of that feature across the training set. In the case where variables are scaled to a custom range, [a,b], Equation 94 becomes:

$$x_{\text{scaled}} = a + \frac{(x - x_{\min})(b - a)}{x_{\max} - x_{\min}}, \tag{95}$$

here, [a,b] represents the target bounding interval (e.g. [-1, 1]) (Han et al., 2012, Chapter 3).

An alternative to min-max scaling is feature standardization, often referred to as the z-score normalization. This transformation centers the input data around a mean of zero and scales it to have a unit variance. The standardized value, $z$, is computed as:

$$z = \frac{x - \mu}{\sigma} \tag{96}$$

where $x$ represents the raw input parameter, while $\mu$ and $\sigma$ denote the empirical mean and standard deviation of that specific feature across the training dataset, respectively. This method of normalization proves more useful when the exact minimum and maximum of $x$ is unknown or when there exist outliers that dominate the min-max normalization (Han et al., 2012, Chapter 3).

Both these normalization methods attempt to give all input parameters an equal weight, helping prevent input parameters with large ranges from outweighing those with smaller values (Han et al., 2012, Chapter 3).

### 2.7.8 Computational Efficiency of Neural Network Inference

As discussed earlier, neural networks can be interpreted as function approximators, mapping inputs to outputs through a sequence of linear transformations and nonlinear activations. This process can be expressed as a series of matrix multiplications followed by element-wise operations and is particularly well-suited for execution on graphics processing units (GPUs), which are designed for massively parallel computation (Raina et al., 2009).

Inference refers to applying an already trained neural network to compute outputs for new inputs by performing a single forward pass through the network (Goodfellow et al., 2016). By processing the inputs in batches, parallelization can be utilized to evaluate multiple samples at the same time, leading to highly efficient computation. Leveraging specialized hardware, such as the GPU, to perform specific tasks more efficiently than Central Processing Units (CPUs) is commonly referred to as hardware acceleration.

## 2.8 Differential Machine Learning

### 2.8.1 Differential Machine Learning Framework

As explained in section 2.7, a traditional feedforward neural network learns a mapping $f_\omega : \mathbb{R}^n \to \mathbb{R}^m$ from inputs $\mathbf{x} \in \mathbb{R}^n$ to a vector output $\mathbf{y} \in \mathbb{R}^m$ by minimizing the difference between the network predictions $\hat{\mathbf{y}} = f_\omega(\mathbf{x})$ and the training labels $\mathbf{y}$ (Prince, 2023). In the context of option pricing, the network could for example learn to map the Heston parameters and the market state to an option price or implied volatility. While this architecture often achieves good point-wise accuracy, the function it approximates is effectively a black box. We don't control the shape of the function and the partial derivatives of the output with respect to the inputs $\partial f_\omega / \partial x_i$ , i.e., the Greeks with respect to the Heston parameters in this case, may not correspond to the actual analytical or semi-analytical derivatives (Huge & Savine, 2020).

This limitation makes it impossible to trust the approximated function and use it to make calibration more efficient (Sridi & Bilokon, 2023). Gradient-based calibration, such as the Levenberg–Marquardt or L-BFGS algorithms, not only require correct outputs but also reliable derivatives, Greeks, of the function with respect to the parameters of the underlying model. Using such models becomes an impossibility if the core pricing engine cannot provide accurate derivatives. Using a traditional NN could make the optimizer converge to the wrong parameter values or fail to converge at all. Classically, using traditional pricers and not neural networks, the derivatives are computed by finite differences or by differentiating the semi-analytical pricing formula directly. This is much more time consuming or requires solving complex-valued integrals. The core challenge is therefore not only to approximate the function from parameters to prices accurately, but to also produce stable derivatives.

Differential Machine Learning (DML), that was introduced by Huge and Savine (2020), addresses this issue by including the derivatives in the loss function. Instead of training the network only on inputs and outputs $(\mathbf{x}, y)$, DML includes the true partial derivatives of the outputs with respect to the inputs $\partial y / \partial x_i$ and penalizes the network for errors in the output values as well as the derivatives, simultaneously. This results in a network that learns a surface that not only outputs the correct values but also has the correct slope at each point. This method is closely related to the concept of Sobolev training that was first proposed by Czarnecki et al. (2017), where the authors use Sobolev norms to penalize the function values as well as their derivatives at the same time. In this work, we bridge the methods related to the abstract mathematics of Sobolev spaces, but apply it using a traditional mean squared error based loss that penalizes outputs as well as their derivatives, but the underlying mathematical concepts are the same.

The practical value of this approach in the context of Heston model calibration can be understood in terms of two main aspects. Firstly, as the model has learned the shape of the derivatives it can output them explicitly and thereby provide the Jacobian matrix, or the gradient if the output is scalar. This enables the use of gradient-based optimizers without much additional cost, as only an additional backward pass is needed. Second, including the derivatives in the loss function works as a regularizer for the network, forcing it to learn a more meaningful and smoother mapping. This tends to improve generalization and reduces the tendency for overfitting.

### 2.8.2 Sobolev Training

The theoretical foundation for training neural networks with derivatives relies on theory from the realm of functional analysis, more specifically the theory of Sobolev spaces (Czarnecki et al., 2017). In traditional supervised training, the network is most commonly trained by minimizing a spatial loss, such as MSE, which only penalizes the difference between the predicted and target function values. However, when

incorporating derivatives, the network does not only learn the point-wise mapping, but also its underlying geometry.

To formalize this mathematically we can use the theory of Sobolev spaces, denoted as $W^{k,p}(\Omega)$ (Evans, 1998). To understand why this framework builds the foundation of differential deep learning, it is helpful to define some core concepts:

*Weak Derivatives*

In classical calculus, differentiating a function requires it to be point-wise differentiable in the relevant domain. A weak derivative is a mathematical generalization that allows us to define derivatives for a broader class of functions, including many that are not classically differentiable, over a domain (Evans, 1998, Sec. 5.2.1).

*P-integrability*

A function is p-integrable if the integral of its absolute value raised to the power of p is finite (Weber, 2018). In the context of training a neural network the power 2 is central, it leads to what is called the $L^2$ space. Here, we measure the size of a function by squaring its values and integrating them over the entire domain, analogous to how we would compute the Euclidean distance by summing the squared components of a vector. Hence, this could be viewed as the continuous version of MSE, i.e., across an entire function instead of a finite set of data points.

*The Sobolev Space $W^{k,p}(\Omega)$*

Using these two concepts, we can define the Sobolev space as a vector space of functions whose values and weak derivatives up to order k are all p-integrable over a specified domain $\Omega$ (Weber, 2018). This makes sure that the approximations of the function and its gradients are mathematically constrained through $L^p$-integrability (Evans, 1998).

Czarnecki et al. (2017) showed that the training of a neural network could be significantly improved by minimizing a loss function defined by a Sobolev norm. This concept makes it possible to not only look at the magnitude of the approximations but also the smoothness of the function. Using only first derivatives and the squared error in the training directly translates to utilizing the theory behind first-order Sobolev spaces, and more specifically the $H^1$ norm where $k = 1$, with $p = 2$, which generates the following objective function (Brezis, 2011; Son et al., 2021; Weber, 2018):

$$\|f - g\|^2_{W^{1,2}} = \int_\Omega |f(\mathbf{x}) - g(\mathbf{x})|^2 \, d\mathbf{x} + \int_\Omega |\nabla f(\mathbf{x}) - \nabla g(\mathbf{x})|^2 \, d\mathbf{x} \tag{97}$$

Where $\nabla$ denotes the gradient with respect to x. Note that as we square the norm, we cancel out the square root that is present in the regular definition of the Sobolev norm. The core insight from this function is that the first integral ensures that the network $f(x)$ approximates the target values $g(x)$, while the second integral handles the derivatives of the function.

In the practical implementation of this, one possible architecture is the Deep Differential Network (DDN). To make this feasible, we need to approximate these theoretical integrals over the domain $\Omega$ (Son et al., 2021). The solution lies in using the discrete empirical average over the finite training dataset, i.e., the MSE. We can combine the MSE of the predictions and their gradients, effectively creating a composite MSE loss which acts as the discrete approximation of the Sobolev norm. Huge and Savine (2020) used this framework and applied it to the quantitative finance setting through adapting what they called Differential Machine Learning (DML). This made it possible to use the theory of Sobolev norms with traditional ML tools.

### 2.8.3 The Differential Loss Function

To formalize this empirical implementation of the Sobolev norm, we will construct a mathematical formulation of the networks loss function (Huge & Savine, 2020). Consider a dataset used for training the neural network $(\mathbf{x}^{(i)}, y^{(i)}, \mathbf{z}^{(i)})_{i=1}^{N}$, where, for $N$ samples, $\mathbf{x}^{(i)} \in \mathbb{R}^n$ is the input vector, $y^{(i)} \in \mathbb{R}$ is the target output (e.g. the option price or implied volatility) and $\mathbf{z}^{(i)} = \nabla_{\mathbf{x}} y^{(i)} \in \mathbb{R}^n$ is the vector of partial derivatives of the output with respect to each input component (e.g. Heston parameter Greeks), i.e.,

$$\mathbf{z}^{(i)} = \left( \frac{\partial y}{\partial x_1}\bigg|_{\mathbf{x}^{(i)}}, \frac{\partial y}{\partial x_2}\bigg|_{\mathbf{x}^{(i)}}, \ldots, \frac{\partial y}{\partial x_n}\bigg|_{\mathbf{x}^{(i)}} \right)^{\top}. \tag{98}$$

In the context of option pricing under the Heston model, the input vector $\mathbf{x}$ consists of the Heston parameters $(\kappa, \theta, \sigma, \rho, v_0)$ as well as the market state variables $(\tau,\ r,\ m)$ so that the input vector is eight-dimensional.

Let $f_\omega(\mathbf{x})$ denote the function approximated by the neural network with the parameters $\omega$, and let $\hat{y}^{(i)} = f_\omega(\mathbf{x}^{(i)})$ be the predicted output for the sample with index $i$. The derivatives of the prediction are therefore obtained by differentiating the approximated function with respect to its inputs:

$$\hat{z}_j^{(i)} = \frac{\partial f_\omega}{\partial x_j}\bigg|_{\mathbf{x}^{(i)}}, \quad j = 1, \ldots, n. \tag{99}$$

where $n$ denotes the input dimension so that in the Heston case, $n = 8$ for all input derivatives, or $n = 5$ for the parameter Greeks.

The traditional loss function in deep learning is, as we discussed earlier, the mean squared error over only the function values:

$$\mathcal{L}_{\text{values}}(\omega) = \frac{1}{N} \sum_{i=1}^{N} \left( \hat{y}^{(i)} - y^{(i)} \right)^2. \tag{100}$$

Whereas the differential loss function instead penalizes the errors of each partial derivative:

$$\mathcal{L}_{\text{diff}}(\omega) = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{n} \left( \hat{z}_j^{(i)} - z_j^{(i)} \right)^2. \tag{101}$$

Hence, the final loss function is the composite of these two separate loss functions:

$$\mathcal{L}_{\text{DML}}(\omega) = \mathcal{L}_{\text{values}}(\omega) + \lambda\, \mathcal{L}_{\text{diff}}(\omega) = \frac{1}{N} \sum_{i=1}^{N} \left[ \left( \hat{y}^{(i)} - y^{(i)} \right)^2 + \lambda \sum_{j=1}^{n} \left( \hat{z}_j^{(i)} - z_j^{(i)} \right)^2 \right], \tag{102}$$

where $\lambda > 0$ is a tunable hyperparameter that controls the relative importance of the error of the derivatives in relation to the accuracy of the function output approximation. The choice of the parameter $\lambda$ effectively balances the loss of the output and its derivatives and therefore serves an important purpose, namely helping the network to converge and prioritize the optimal weight changes at any given epoch.

In practice, the ranges of the derivatives differ substantially and it can therefore be helpful to generalize the differential loss by introducing separate weights for each derivative:

$$\mathcal{L}_{\text{DML}}(\omega) = \frac{1}{N} \sum_{i=1}^{N} \left[ \left( \hat{y}^{(i)} - y^{(i)} \right)^2 + \sum_{j=1}^{n} \lambda_j \left( \hat{z}_j^{(i)} - z_j^{(i)} \right)^2 \right], \tag{103}$$

where $\lambda_j$ is the weight that is associated with the $j$-th derivative. Huge and Savine (2020) suggest setting the individual weights proportionally to the inverse variance of each derivative across the entire dataset.

More concretely, let $\text{Var}[z_j]$ denote the empirical variance of the $j$-th derivative and the weights can be chosen as:

$$\lambda_j = \frac{1}{\text{Var}[z_j] + \epsilon}, \tag{104}$$

where a small constant $\epsilon > 0$ is added to the denominator to ensure numerical stability.

### 2.8.4 Twin Networks and the Predicted Derivatives

Huge and Savine (2020) additionally introduces the concept of a twin network, which is not a separate architecture, but rather a new way of looking at the concept of deep differential learning. Here, they combine the forward pass, that represents prediction, and the backward pass, representing differentiation in one computational concept.

**Forward Pass**

To illustrate this, consider a traditional neural network with $L$ hidden layers. The forward pass computes the output through linear transformations followed by activation functions. Denote the pre-activation values at layer $\ell$ by $\mathbf{a}^{(\ell)}$ and the values post-activation by $\mathbf{h}^{(\ell)}$. The forward pass is then conducted as follows:

$$\begin{cases} \mathbf{h}^{(0)} = \mathbf{x} \\ \mathbf{a}^{(\ell)} = \mathbf{W}^{(\ell)}\mathbf{h}^{(\ell-1)} + \mathbf{b}^{(\ell)}, \quad \ell = 1, \dots, L \\ \mathbf{h}^{(\ell)} = \Phi(\mathbf{a}^{(\ell)}), \quad \ell = 1, \dots, L \\ \hat{y} = \mathbf{w}^{(L+1)\top}\mathbf{h}^{(L)} + b^{(L+1)} \end{cases} \tag{105}$$

where $\mathbf{W}^{(\ell)} \in \mathbb{R}^{d_\ell \times d_{\ell-1}}$ and $\mathbf{b}^{(\ell)} \in \mathbb{R}^{d_\ell}$ are the weight matrix and bias vector of layer $\ell$, $\Phi$ is the activation function, and the final layer consists of the weight vector $\mathbf{w}^{(L+1)} \in \mathbb{R}^{d_L}$ and scalar bias $b^{(L+1)}$.

**Backward Pass - Computing Input Derivatives**

Using the chain rule, the second part of the network computes the Jacobian of the output, with respect to the chosen inputs. This is done layer by layer starting from the final layer in the network and is mathematically analogous to normal backpropagation but with respect to the inputs rather than the network parameters $\omega$.

We seek the row vector $\frac{\partial \hat{y}}{\partial \mathbf{h}^{(\ell)}} \in \mathbb{R}^{1 \times d_\ell}$ at every layer $\ell$.

At the output layer the prediction is $\hat{y} = \mathbf{w}^{(L+1)\top}\mathbf{h}^{(L)} + b^{(L+1)}$, so:

$$\frac{\partial \hat{y}}{\partial \mathbf{h}^{(L)}} = \mathbf{w}^{(L+1)\top}. \tag{106}$$

We then propagate backward, recursively, through the network and for each layer $\ell = L, L-1, \dots, 1$, we apply the chain rule twice. First, since $\mathbf{h}^{(\ell)} = \Phi(\mathbf{a}^{(\ell)})$ element-wise:

$$\frac{\partial \hat{y}}{\partial \mathbf{a}^{(\ell)}} = \frac{\partial \hat{y}}{\partial \mathbf{h}^{(\ell)}} \cdot \frac{\partial \mathbf{h}^{(\ell)}}{\partial \mathbf{a}^{(\ell)}} = \frac{\partial \hat{y}}{\partial \mathbf{h}^{(\ell)}} \odot \Phi'(\mathbf{a}^{(\ell)}), \tag{107}$$

where $\odot$ denotes element-wise multiplication and $\Phi'(\mathbf{a}^{(\ell)})$ is the element-wise derivative of the activation function. Then, since $\mathbf{a}^{(\ell)} = \mathbf{W}^{(\ell)}\mathbf{h}^{(\ell-1)} + \mathbf{b}^{(\ell)}$:

$$\frac{\partial \hat{y}}{\partial \mathbf{h}^{(\ell-1)}} = \frac{\partial \hat{y}}{\partial \mathbf{a}^{(\ell)}} \cdot \frac{\partial \mathbf{a}^{(\ell)}}{\partial \mathbf{h}^{(\ell-1)}} = \frac{\partial \hat{y}}{\partial \mathbf{a}^{(\ell)}} \mathbf{W}^{(\ell)}. \tag{108}$$

At the input layer ($\ell = 0$, where $\mathbf{h}^{(0)} = \mathbf{x}$), we recover the desired gradient:

$$\frac{\partial \hat{y}}{\partial \mathbf{x}} = \frac{\partial \hat{y}}{\partial \mathbf{h}^{(0)}} = \hat{\mathbf{z}}. \tag{109}$$

This backward pass is what Huge and Savine (2020) refer to as the twin of the forward pass. They both share the same weights, and combined they produce both the function value $\hat{y}$ and the full gradient vector $\hat{\mathbf{z}} \in \mathbb{R}^n$. These two "parts" of the network are, therefore, effectively merely two views of the same function.

### 2.8.5 Training a Differential Neural Network

To train the network by minimizing the loss function, we require the gradient of the loss with respect to the network parameters $\omega$ (Huge & Savine, 2020). This gradient will have two separate components, as the loss function consists of two terms.

The first component, $\nabla_\omega \mathcal{L}_{\text{values}}$, is computed via standard backpropagation. For each training sample, the error $(\hat{y}^{(i)} - y^{(i)})$ is propagated backward through the network to obtain the gradient with respect to each weight and bias.

The second component, $\nabla_\omega \mathcal{L}_{\text{diff}}$, requires differentiating the predictions of the derivatives $\hat{z}_j^{(i)} = \partial f_\omega / \partial x_j$, hence, computing $\nabla_\omega \hat{z}_j^{(i)}$ involves a second-order differentiation. For a single training sample $i$ and derivative $j$, the contribution to the total gradient is:

$$\frac{\partial}{\partial \omega} \left( \hat{z}_j^{(i)} - z_j^{(i)} \right)^2 = 2 \left( \hat{z}_j^{(i)} - z_j^{(i)} \right) \frac{\partial \hat{z}_j^{(i)}}{\partial \omega}, \tag{110}$$

where $\hat{z}_j^{(i)} = \partial f_\omega(\mathbf{x}^{(i)}) / \partial x_j$, and hence the term $\partial \hat{z}_j^{(i)} / \partial \omega$ involves second-order mixed partial derivatives $\partial^2 f_\omega / \partial x_j \partial \omega$. The complete gradient of the total loss is:

$$\nabla_\omega \mathcal{L}_{\text{DML}} = \frac{2}{N} \sum_{i=1}^{N} \left[ \left( \hat{y}^{(i)} - y^{(i)} \right) \nabla_\omega f_\omega(\mathbf{x}^{(i)}) + \sum_{j=1}^{n} \lambda_j \left( \hat{z}_j^{(i)} - z_j^{(i)} \right) \nabla_\omega \frac{\partial f_\omega(\mathbf{x}^{(i)})}{\partial x_j} \right]. \tag{111}$$

Hence, training a differential neural network requires computing not only first-order gradients of the network output with respect to both inputs and parameters, but also mixed second-order derivatives. While this increases the computational cost for each epoch during training, the additional regularization provided by the derivatives often reduce the total number of required training samples and speeds up convergence (Huge & Savine, 2020).

As mentioned in Section 2.7.3, several common choices of activation functions, such as ReLU, are not continuously differentiable over their entire domain. A critical observation is that Equation 107 requires us to calculate $\Phi'$, and for functions that are not continuously differentiable, this yields discontinuities in the derivatives that may make them unreliable (Sridi & Bilokon, 2023). While the Sobolev training framework established earlier guarantees that learning with differential labels is well-posed even when the derivatives are weak, the practical implementation suffers from instability. Hence, at least $C^1$-smooth (continuously differentiable to the first degree) functions should be used in the practical implementation of this framework, such as the previously discussed Softplus or SiLU functions, which are infinitely differentiable ($C^\infty$) (Czarnecki et al., 2017). Furthermore, as established in this section, optimizing the network relies heavily on these mixed second-order derivatives. For ReLU, $\Phi'' = 0$ almost everywhere, so the network receives little useful gradient signal for smooth Greeks.

### 2.8.6 Application to the Two-Step Calibration Framework

Leaning on the logic of the two-step calibration approach, the differential neural network effectively works as a surrogate for the semi-analytical pricing models (Sridi & Bilokon, 2023). Hence, the network does not calibrate to an entire surface in one pass, but rather approximates the mapping

$$f_\omega : (\kappa, \theta, \sigma, \rho, v_0, \tau, r, m) \mapsto y, \tag{112}$$

where $y$ can be the option price or implied volatility. As the network is trained on the derivatives of the output with respect to the Heston parameters, practically simultaneous parameter Greeks can be provided:

$$\hat{\mathbf{z}} = \left( \frac{\partial f_\omega}{\partial \kappa}, \frac{\partial f_\omega}{\partial \theta}, \frac{\partial f_\omega}{\partial \sigma}, \frac{\partial f_\omega}{\partial \rho}, \frac{\partial f_\omega}{\partial v_0} \right). \tag{113}$$

During the calibration loop, the Heston parameters $\boldsymbol{\phi} = (\kappa, \theta, \sigma, \rho, v_0)$ are adjusted in order to minimize the error between the models predictions and the true market prices.

## 2.9 Evaluation and Stability Criteria

In standard machine learning, one might use simple evaluation metrics such as the Mean Squared Error. However, evaluating a neural network surrogate in quantitative finance requires moving beyond these standard metrics.

For this report's purpose, the evaluation of the surrogates must account for scale-dependent pricing errors. Because option prices vary drastically depending on e.g. log-moneyness, absolute error metrics can obscure a model's true performance across different market regimes by disproportionately penalizing errors on expensive options while ignoring relative errors on cheaper ones. Furthermore, the surrogate must be evaluated on the mathematical smoothness of its generated partial derivatives (the Greeks) (Huge & Savine, 2020). Because financial calibration and hedging rely on continuous and stable gradients, a model that predicts accurate prices but volatile Greeks is practically unusable in a production environment (Gouk et al., 2020).

In addition to standard regression metrics and Greek stability, a robust evaluation framework must verify that the surrogate model predictions comply with the static arbitrage constraints established in Section 2.1.6.

### 2.9.1 Error Metrics

To evaluate the performance of a neural network acting as a pricing surrogate, measuring the difference between the network's predictions and the true theoretical values is paramount. Choosing the right error metrics is an important step in assessing both the model's optimization convergence and its ability to generalize to unseen data (Goodfellow et al., 2016).

- **Mean Squared Error (MSE):** The MSE is the most common loss function for regression tasks and is defined as:

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^{N} (\hat{y}_i - y_i)^2 \tag{114}$$

  where $N$ is the total number of samples, $\hat{y}_i$ is the predicted output from the neural network, and $y_i$ is the true target value. Because MSE is continuously differentiable, it is naturally suited for

gradient-based optimization. However, the quadratic penalty means that MSE is highly sensitive to large outliers (Goodfellow et al., 2016). In the context of option pricing, this means that large absolute errors from deep in-the-money (ITM) options or extreme parameter combinations can easily dominate the metric, which might obscure how well the network performs under more standard market conditions.

- **Mean Absolute Error (MAE):** To measure the average magnitude of the errors without the quadratic penalty of MSE, the MAE is utilized, defined as:

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|\hat{y}_i - y_i| \tag{115}$$

By applying a linear penalty to deviations, MAE is much more robust to extreme outliers than MSE (Willmott & Matsuura, 2005). It also provides a more direct and interpretable representation of the expected absolute pricing error in standard currency units.

- **Mean Absolute Percentage Error (MAPE):** Because vanilla option prices vary drastically depending on their log-moneyness and time to maturity, absolute error metrics can sometimes misrepresent relative accuracy. A pricing error of \$0.10 is negligible for a deep ITM option but catastrophic for a deep out-of-the-money (OTM) "penny option". MAPE is used to measure accuracy independently of the price scale, defined as:

$$\text{MAPE} = \frac{100}{N}\sum_{i=1}^{N}\left|\frac{y_i - \hat{y}_i}{y_i}\right| \tag{116}$$

However, it comes with a limitation: the metric becomes undefined or highly asymmetric as $y_i \to 0$ (Hyndman & Koehler, 2006). Consequently, applying MAPE requires certain precautions, such as imposing a minimum price threshold, to prevent deep OTM options from causing numerical instability.

### 2.9.2 Stability of Greeks and Lipschitz Continuity

When evaluating deep neural networks for financial applications, point-wise accuracy is insufficient if the partial derivatives (Greeks) are highly volatile (Huge & Savine, 2020).

**Definition 2.1** (Lipschitz Condition)**.** A function $f$, defined on $[a, b]$, is said to satisfy a Lipschitz condition on $[a, b]$ if there exists a constant $L > 0$ such that

$$|f(x_1) - f(x_2)| \leq L|x_1 - x_2|, \tag{117}$$

for all $x_1, x_2 \in [a, b]$. $L$ is called the Lipschitz constant (Phillips & Taylor, 1996).

From Equation 117 we can see that, if $f$ fulfills a Lipschitz condition on $[a, b]$, then $f$ is uniformly continuous on $[a, b]$, meaning its rate of change is strictly bounded (Phillips & Taylor, 1996). For a deep neural network, maintaining a bounded Lipschitz constant is strongly correlated with improved generalization capabilities and robustness against small disturbances in the input data (Gouk et al., 2020).

Although Definition 2.1 describes a scalar function of a single variable, the concept naturally extends to multidimensional spaces. For a deep neural network mapping a vector of input parameters to an output space, the absolute value is simply replaced by an appropriate vector norm, $\|\cdot\|$.

However, analytically computing the exact global Lipschitz constant for a Deep Neural Network (DNN) requires solving a highly non-convex optimization problem. This has been shown to be computationally demanding and NP-hard (Virmaux & Scaman, 2018). To avoid this, one can instead evaluate the network over a finite dataset. By rearranging the inequality from Definition 2.1 for a multidimensional case, we obtain the difference quotient:

$$\frac{\|f(x_1) - f(x_2)\|}{\|x_1 - x_2\|} \leq L \tag{118}$$

Since the true global constant $L$ must be greater than or equal to any possible quotient, the best estimate we can obtain from a dataset is the maximum quotient we actually observe. Furthermore, because we are only evaluating a finite dataset of points rather than the infinite continuous space, this observed maximum acts as a guaranteed lower bound for the true global constant (Fazlyab et al., 2019; Weng et al., 2018). This leads to the following formulation of the empirical bound.

**Definition 2.2** (Empirical Lipschitz Constant)**.** Let $X = \{x_1, x_2, \ldots, x_N\}$ be a finite set of points in the input parameter space and let $g(x)$ represent a predicted Greek output. Evaluated over $X$, the empirical Lipschitz constant $\hat{L}$ is mathematically defined as the maximum pairwise difference quotient:

$$\hat{L} = \max_{\substack{x_i, x_j \in X \\ x_i \neq x_j}} \frac{\|g(x_i) - g(x_j)\|}{\|x_i - x_j\|}. \tag{119}$$

Finding this exact empirical lower bound through an exhaustive pairwise search across a dataset of size $N$ requires evaluating $\binom{N}{2}$ unique pairs. This yields a quadratic time complexity of $\mathcal{O}(N^2)$, which becomes computationally infeasible for large training or calibration datasets.

To overcome this secondary bottleneck, $\hat{L}$ can be approximated using Monte Carlo sampling. By randomly sampling a subset of input pairs from $X$, one computes the corresponding difference quotients to establish a lower bound for $\hat{L}$. While this stochastic approximation can underestimate the maximum pairwise difference present in the full dataset, sampling-based estimations like this are standard practice in the robustness literature for efficiently evaluating network stability (Weng et al., 2018).

# 3 Methodology

## 3.1 Sampling Input Parameters

The empirical foundation for this study relies on synthetic generated data rather than historical market data. While historical market data reflects the real market conditions it is hard to get a hold of, and is often inadequate, limited in volume and noisy, which can limit the generalization capabilities when it comes to training a neural network (Brigo et al., 2026). The parameters that were sampled were the Heston parameters $(\kappa, \theta, \sigma, \rho, v_0)$ and the market state variables, that is time to maturity $(\tau)$, risk-free rate $(r)$ and log-moneyness $(m)$.

Independent random sampling (Monte Carlo) was deliberately avoided to ensure that the model's parameter space was covered as uniformly as possible. In high-dimensional spaces, random sampling tends to create clusters and leave significant empty voids, a phenomenon directly associated with the curse of dimensionality, as described in Section 2.6. To theoretically justify the choice of a more advanced sampling technique, an initial discrepancy test was conducted. Two sampling methods, known for their superiority in sampling data-points in higher dimensions, were tested against each other. These were

Latin Hypercube sampling (LHS) and Sobol sampling. The test was conducted on multiple sample sizes of sampled data points in eight dimensions. As shown in Figure 6, both models exhibit exceptionally low discrepancy values even for lower numbers of samples. Figure 6 further shows how the discrepancy decreases as the number of samples increases. However, Sobol sampling showed both lower initial discrepancy and faster convergence in discrepancy for larger sample sizes, validating the superior uniformity distribution of this method.

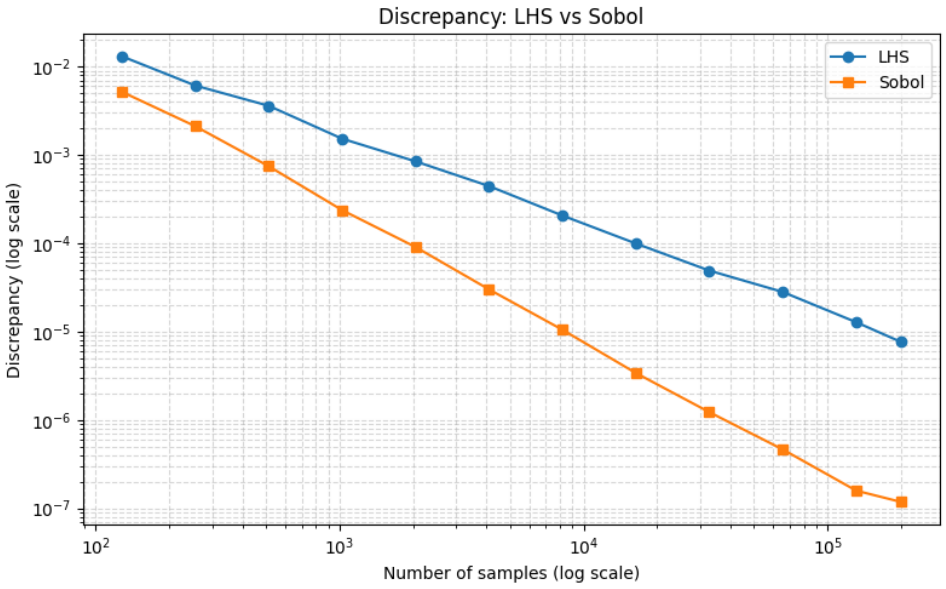



Figure 6: Comparison of discrepancy between Latin Hypercube sampling (LHS) and Sobol sampling in eight dimensions.

The parameters for the Heston model and the market variables were generated using Sobol sampling which generated each parameter within the interval $[0, 1]$. These parameter values were then scaled up within predefined bounds to reflect realistic and stressed market conditions. The intervals were defined as below:

Table 1: Parameter ranges used in the model.

| **Parameter** | **Range** |
|---|---|
| $\kappa$ | 0.05 to 5.0 |
| $\theta$ | 0.0 to 1.0 |
| $\sigma$ | 0.1 to 1.5 |
| $\rho$ | -0.95 to 0.0 |
| $v_0$ | 0.0 to 1.0 |
| $\tau$ | 1/52 to 3.0 (years) |
| $r$ | -0.01 to 0.1 |
| $m$ | -1.5 to 1.5 |

The parameter ranges were selected based on a literature review of previous Heston-based neural network studies, including Liu, Oosterlee, and Bohte (2019), Liu, Borovykh, et al. (2019), C. Zhang et al. (2025) and Sridi and Bilokon (2023). Since no exact market-standard bounds are generally provided, the review was used to obtain an indication of reasonable parameter values and to ensure consistency with previous work.

Although the intervals may appear relatively wide, with some boundary combinations not necessarily being representative of typical market conditions, this is intentional. C. Zhang et al. (2025) note that a wide parameter domain is retained in order for the network to learn the Heston pricing function more thoroughly. The wide ranges also reduce the potential need for extrapolation. This is relevant since neural networks have been shown to perform poorly beyond their training data and to deteriorate as new samples move far from the training set (Bonnasse-Gahot, 2022).

Additionally, very short maturities are excluded from the dataset as the standard Heston model generally fits IV surfaces poorly in these regimes (Rouah, 2013). As described in Section 2.2.3, this limitation arises because the CIR variance process is unable to generate steep short-term skew. For these maturities, models incorporating jump-diffusion, such as the Bates (1996) model, are generally preferred as they better capture the volatility differentials and steep skews characteristic of the short end of the maturity spectrum. Hence, calibrating the standard Heston model in this region is of limited relevance (Bakshi et al., 1997).

Once the data points consisting of the eight sampled parameters above were scaled to the bounds, the underlying asset price ($S$) was set to a constant value of 1 and then added to the dataset. This was done to reduce the dimensionality, which facilitates machine learning as well as dampens the curse of dimensionality when sampling. European option prices under the Heston model exhibit linear homogeneity of degree one with respect to the spot and strike prices, meaning that the absolute price levels are mathematically redundant (Merton, 1973). The option value is dependent on the ratio of $S$ to $K$ (moneyness), rather than on the actual absolute magnitudes of $S$ and $K$. By fixing $S = 1$, the neural network is spared from learning this trivial linear scaling relationship, effectively reducing the complexity of the input space by one dimension without any loss of generalization (Liu, Oosterlee, & Bohte, 2019). Given this, the strike price ($K$) at each data point was deterministically derived from the sampled log-moneyness ($m$) using the relationship $K = \exp(-m)$ and then added to the dataset.

## 3.2 Computing Output Labels

### 3.2.1 Computing Price and Price Greeks

After obtaining the sampled input variables, the corresponding European put option prices and price Greeks were calculated using the semi-analytical COS method introduced by Fang and Oosterlee (2008). The application of the COS method for generating the output labels was driven by the possibility to obtain accurate price Greeks needed for the Differential Machine Learning (DML) as well as the computational efficiency. DML requires not only a large number of option prices, but also their corresponding partial derivatives (parameter Greeks) to efficiently train the neural network (Huge & Savine, 2020). When assessing the alternative pricing approaches that can be adopted to the Heston model such as the Monte Carlo approach or the FFT technique, some fundamental deficiencies arise, either in regard to computational efficiency or gradient accuracy. Monte Carlo simulation was deemed incompatible with this study's methodology, while it is highly flexible, it suffers from a slow convergence rate (Glasserman, 2004). Moreover when estimating parameter Greeks via Monte Carlo, typically through finite-difference approximation, significant statistical noise and high variance occurs. According to Huge and Savine (2020), feeding noisy gradients into a DML algorithm results in a unstable neural network training process, making Monte Carlo an unsuitable choice for generating the training dataset. The Fast Fourier Transform (FFT) approach, pioneered by Carr and Madan (1999), was also excluded because of the COS method's superior convergence properties and overall robustness as presented by Fang and Oosterlee (2008).

The precision of the calculated prices and speed of the calculations using the COS method relies on the

number of expansion terms ($N$) and the truncation range parameter ($L$) (Fang & Oosterlee, 2008). For this data set, $N$ was set to 32 768 ($2^{15}$). This value dictates the resolution of the Fourier series, where a higher $N$ ensures exponential convergence to the true option price. The reason for setting $N = 32\,768$ was to achieve a high precision while keeping a feasible efficiency. The integration interval for the COS price is truncated to a finite interval $[a, b]$ that fits the majority of the density function. This interval is determined dynamically for each parameter combination using the cumulants of the Heston characteristic function, scaled by the parameter $L$ (as described in Section 2.3.2). By setting $L = 12$, the truncation domain was wide enough to ensure that the approximation error resulting from the truncated integration bounds was mathematically negligible (Fang & Oosterlee, 2008).

While the COS method provides rapid price generation, the Differential Machine Learning framework imposes strict requirements beyond prices. It is also necessary to have accurate and stable partial derivatives (Greeks) for the training process (Huge & Savine, 2020). Estimating these derivatives, using classical finite differences introduces an unavoidable bias-variance trade-off as explained in Section 2.3.3. To resolve this, the entire COS algorithm was implemented utilizing the JAX Python library, which is a high-performance numerical computing library optimized for automatic differentiation (Bradbury et al., 2018). The JAX automatic differentiation works by tracing the mathematical operations of the COS calculated price, mapping every step, so that it can later traverse them backwards using the chain rule to obtain the exact Greeks. This implementation ensured that the Greeks were analytically obtained in relation to the Heston parameters. In the JAX implementation of COS, certain mathematical adjustments had to be made to ensure numerical stability. The standard maximum function used to prevent $c_2$ from becoming too small was replaced with a continuously differentiable function based on the Softplus activation (as described in Section 2.7.3). Since $c_2$ affects the width of the truncation interval $[a, b]$, values close to zero make the interval very narrow, which can lead to unstable COS coefficients. The Softplus-based replacement keeps $c_2$ bounded away from zero while also preventing gradients from abruptly vanishing. Furthermore, clipping was applied to the arguments within the exponential functions during the Fourier inversion to mitigate the risk of numerical overflow associated with "The Little Heston Trap" (Albrecher et al., 2007).

Despite the high precision of the COS method, numerical artifacts and extreme parameter sampling combinations can occasionally yield invalid option prices. Because the fundamental payoff of a European put option is $max(K - S, 0)$, theoretical option prices must be strictly non-negative. However, due to truncation errors in the Fourier series, particularly for deep out-of-the-money options or when randomly sampled parameters breach the Feller condition the COS method could occasionally produce negative prices. Furthermore, a simulated price must strictly satisfy fundamental no-arbitrage bounds (e.g., $P \geq \max(Ke^{-r\tau} - S, 0)$) to represent an economically viable market state. All negative prices and prices that violated the no-arbitrage conditions were removed from the dataset.

### 3.2.2 Computing Implied Volatility and IV Greeks

In addition to training the network on the option price, the implied volatility is also a relevant output candidate. To obtain this value, the implied volatility corresponding to each combination of arbitrage-free price, time to maturity ($\tau$), risk-free rate ($r$) and log-moneyness ($m$) in the dataset was computed by inverting the Black–Scholes pricing formula.

As described earlier, several numerical methods can be used for this inversion. Newton–Raphson was excluded, as there is a risk for divergence and that the method may fail for small values of Vega (Liu, Oosterlee, & Bohte, 2019). According to Liu, Oosterlee, and Bohte (2019) Let's Be Rational can be viewed as an improved variant of Newton–Raphson. Jäckel (2015) further highlights both the superior accuracy and speed of the method, suggesting that it would be the most suitable choice. However, since

the convergence behavior of Brent's method can vary depending on the application (Brent, 1973), the final choice was also supported empirically on the dataset used in this study.

A comparison was therefore carried out between Let's Be Rational and the Brent method. Since the implied volatilities are used as target labels in the generated dataset, numerical accuracy was considered the most important aspect. First, the implied volatilities obtained from Let's Be Rational were compared with those obtained from Brent's method. As shown in Table 2 the median difference was within machine precision, indicating generally very similar results. The mean difference is slightly larger, but this is driven by a small number of observations close to the lower arbitrage bound, with short maturities or very small Vega. Since the inversion in those areas, regardless of method, is ill-conditioned this is not deemed alarming. Second, each computed implied volatility was inserted back into the Black formula and the resulting price was compared with the original forward price. This repricing test, reported in Table 3, shows that Let's Be Rational has a higher accuracy than Brent's method, reproducing the original prices to numerical precision. Additionally, across repeated runs, Let's Be Rational was consistently faster than Brent, with a speed-up of roughly 20–30 times. This further supported the choice of Let's Be Rational for computing the implied volatilities.

Table 2: Consistency of implied volatilities between methods.

| **Metric** | **Value** |
|---|---|
| Median $\lvert\sigma^{\mathrm{IV}}_{\mathrm{Brent}} - \sigma^{\mathrm{IV}}_{\mathrm{LBR}}\rvert$ | $6.66 \cdot 10^{-16}$ |
| Mean $\lvert\sigma^{\mathrm{IV}}_{\mathrm{Brent}} - \sigma^{\mathrm{IV}}_{\mathrm{LBR}}\rvert$ | $2.90 \cdot 10^{-6}$ |
| 99th percentile $\lvert\sigma^{\mathrm{IV}}_{\mathrm{Brent}} - \sigma^{\mathrm{IV}}_{\mathrm{LBR}}\rvert$ | $1.13 \cdot 10^{-9}$ |

Table 3: Repricing accuracy for implied volatility methods.

| **Metric** | **Let's Be Rational** | **Brent** |
|---|---|---|
| Mean absolute repricing error | $2.65 \cdot 10^{-17}$ | $1.07 \cdot 10^{-14}$ |
| Median absolute repricing error | $8.67 \cdot 10^{-19}$ | $9.02 \cdot 10^{-17}$ |
| Maximum absolute repricing error | $4.72 \cdot 10^{-16}$ | $3.83 \cdot 10^{-13}$ |

When implied volatility is used as an output variable for network training, it follows that the corresponding implied volatility derivatives are also of interest. As described earlier, these derivatives were obtained by dividing the price derivatives with the Black–Scholes Vega. The Vega was computed analytically by inserting the implied volatility $\sigma^{\mathrm{IV}}_H$, time to maturity ($\tau$), risk-free rate ($r$), and log-moneyness ($m$) into Equation 19.

The final dataset used to train the neural networks therefore consisted of the Heston parameters, the time to maturity, the risk-free rate, the log-moneyness, the put price, the price derivatives w.r.t to the Heston parameters, the implied volatility, and lastly the implied volatility derivatives w.r.t. the Heston parameters. This is specified in Table 4 below.

Table 4: Neural network input and output parameters for training, grouped by functional role.

| Parameter | Source | Training role |
|---|---|---|
| **Network Inputs** | | |
| Mean reversion speed, $\kappa$ | Sampled (Sobol) | Input |
| Long-term mean-variance, $\theta$ | Sampled (Sobol) | Input |
| Volatility of variance, $\sigma$ | Sampled (Sobol) | Input |
| Correlation, $\rho$ | Sampled (Sobol) | Input |
| Initial variance, $v_0$ | Sampled (Sobol) | Input |
| Time to maturity, $\tau$ | Sampled (Sobol) | Input |
| Risk-free rate, $r$ | Sampled (Sobol) | Input |
| Log-moneyness, $m$ | Sampled (Sobol) | Input |
| **Output Labels: Base Values** | | |
| Put price, $P$ | Computed (COS) | Output label |
| Implied volatility, $\sigma_H^{\text{IV}}$ | Computed (LBR) | Output label |
| **Output Labels: Price Sensitivities** | | |
| $\partial P/\partial\kappa$ | Computed (COS JAX) | Output label |
| $\partial P/\partial\theta$ | Computed (COS JAX) | Output label |
| $\partial P/\partial\sigma$ | Computed (COS JAX) | Output label |
| $\partial P/\partial\rho$ | Computed (COS JAX) | Output label |
| $\partial P/\partial v_0$ | Computed (COS JAX) | Output label |
| **Output Labels: IV Sensitivities** | | |
| $\partial\sigma_H^{\text{IV}}/\partial\kappa$ | Calculated | Output label |
| $\partial\sigma_H^{\text{IV}}/\partial\theta$ | Calculated | Output label |
| $\partial\sigma_H^{\text{IV}}/\partial\sigma$ | Calculated | Output label |
| $\partial\sigma_H^{\text{IV}}/\partial\rho$ | Calculated | Output label |
| $\partial\sigma_H^{\text{IV}}/\partial v_0$ | Calculated | Output label |
| **Intermediate Calculations** | | |
| BS Vega, $\mathcal{V}$ | Analytical | IV Greek calculation |

The dataset was generated from 200 000 Sobol-sampled data points. Due to some negative prices and arbitrage violations the final dataset used for training and evaluating was reduced to 199 820 data points.

## 3.3 Model Specification and Training of Neural Networks

### 3.3.1 Learning Targets

As discussed in the previous section, several different output targets were created for the dataset. There are multiple reasons for this, of which the two most important ones are (*i*) to improve training stability, extrapolation and accuracy, and (*ii*) to facilitate the integration with current calibration systems and

frameworks.

Neural networks are known to underperform on extreme-valued targets, and training stability as well as predictive accuracy generally improve when labels are well-scaled and bounded (Prince, 2023). Consequently, a key design choice is whether to use option prices as training targets or to instead rescale the problem by using implied volatility (IV) as labels. Using IV as the output target offers several advantages. First, the transformation from implied volatility to option price via a pricing formula is both faster and numerically more stable than the inverse operation. As a result, a surrogate model that predicts IVs can inherently be used to generate prices efficiently as a post-processing step. Hence, a model that can predict IVs can be applied to both calibration paradigms while the model trained on prices cannot in an efficient manner. Second, prior studies have shown that training neural networks on implied volatility surfaces leads to improved out-of-sample generalization compared to price-based targets (Mostafa & Dillon, 2008). Mostafa and Dillon (2008) argue that this improvement arises because IV-based models are better able to capture the underlying volatility level and structural features of the surface, rather than learning scale-dependent price effects. Furthermore, recent work has emphasized that practitioners typically prefer to calibrate models directly to implied volatility rather than prices, as it allows them to specify prices for all vanilla options consistently across the surface, while also ensuring mathematical stability by normalizing differences in raw option prices (Liu, Borovykh, et al., 2019). This further motivates the use of IVs as primary training targets, both from a numerical stability perspective and from the standpoint of practical relevance.

Another output label used during training was price normalized by strike, i.e., $P/K$, which serves as a natural transformation and scaling of the put option. As shown in Section 2.1.1 the payoff of a European put option is bounded above by $K$, since the maximum payoff $\max(K - S_T, 0)$ is attained at $S_T = 0$. Discounting this maximum payoff to the present gives the no-arbitrage bound

$$P(S, K, T, r, q, \sigma) \leq Ke^{-rT}, \tag{120}$$

and hence

$$\frac{P}{K} \leq e^{-rT}. \tag{121}$$

For non-negative interest rates this bounds the output to be less or equal to 1.

Furthermore, this choice follows the concept of a homogeneity hint as detailed in Garcia and Gençay (2000). While the authors apply this concept to call options ($C$), we apply this framework to put options ($P$). The theory is based on the fact that the option payoff is homogeneous of degree one, i.e., scaling S and K by a constant c scales the payoff by that same constant for both calls and puts. This property was used by Merton (1973) to establish the homogeneity of the option price. Garcia and Gençay (2000) used neural networks to approximate a generalized Black–Scholes-type option pricing function. The authors discuss the fact that traditional neural network architectures often suffer from poor extrapolative capacity, i.e., low accuracy in out-of-distribution (OOD) samples. This problem can be mitigated through the use of hints, implemented by giving the model more information about the properties of the underlying function than purely input-output pairs. They showed that training on $C/K$ and moneyness improved the performance of the non-differential networks by giving a hint of the homogeneity property. More recent implementations in deep learning architectures also followed this strategy, for instance Liu, Oosterlee, and Bohte (2019).

In total, three main versions of the same core differential neural network were trained; DDN_P ($P$), DDN_PdivK ($P/K$) and DDN_IV (IV). DDN_PdivK is only a minor modification of the training setup

used for the model trained on price $P$. The main differences between these training strategies lie in the scaling of the Greeks and the chain rule implications, detailed in Section 3.3.7. The setup required for the model trained on IV and IV Greeks additionally required more careful handling of the weights as the variance of the IV Greeks was substantially larger than that of price Greeks as a consequence of the division by small Vega.

A baseline model was also trained to give an indication of the improvements and advantages that differential learning brings. This model was trained as a traditional multilayer perceptron (MLP), i.e., not on the parameter Greeks and only on price labels, and is denoted as MLP_P.

### 3.3.2 Data Split and Batching

From the synthetically sampled dataset, consisting of 199 820 data points, 90% were used for training and 10% for validation. Given the relatively large dataset, this split was considered sufficient to provide a large training set while still retaining enough validation data for reliable evaluation.

The batch size was chosen based on training stability, available computational resources and a literature based review. As noted by Sridi and Bilokon (2023), the batch size induces a trade-off between training stability, which benefits from large batches, and the ability to escape local minima better, which is enhanced by using small batches. The authors use a batch size of 819, which is larger than other implementations and effectively prioritizes training stability. Other previous implementations of similar models have used a batch size of 256 for training (Liu, Oosterlee, & Bohte, 2019; C. Zhang et al., 2025). Batch sizes ranging from 256 to 1024 were tested to inform our implementation and methodology. These preliminary experiments showed that smaller batch sizes performed better in terms of validation MSE. While this is not a complete indicator of training stability and generalization capabilities, the low MSE and support from the convention established by previous studies motivated the choice of a final batch size of 256.

### 3.3.3 Activation Functions

For the networks that were trained on both option price and corresponding Greeks, smooth activation functions were used for all network layers. The mathematical foundation for this choice is detailed in Section 2.8.5. Specifically, SiLU was applied to all the hidden layers, as it has shown promising results in previous experiments (Dubey et al., 2022), while Softplus was used in the output layer. As mentioned earlier, Softplus returns only positive values, thereby enforcing non-negative option price predictions.

Although ReLU is a standard choice for conventional feedforward networks with loss functions that do not require smooth activation functions, we found that the SiLU-Softplus activation combination also gave slightly better results for the baseline model trained without Greek targets. Based on these preliminary experiments, this combination was therefore applied to all models.

### 3.3.4 Weight Initialization

Since Kaiming initialization was originally developed for ReLU-type activations, it is also a natural choice for SiLU-activated hidden layers. Ramachandran et al. (2017) used the same initialization in their experiments on SiLU networks, which empirically supports this choice.

For the output layer, we used Xavier initialization as a more moderate alternative to prevent unstable output scales, since excessively large initial weights can lead to large losses and badly scaled gradients, which can slow down convergence.

### 3.3.5 Loss Function

The loss function is highly influential in the training and convergence of differential neural networks (Huge & Savine, 2020). This is in part due to the fact that the optimizer now has to balance two different components that might not always agree in terms of the most relevant weight changes for improved loss. As introduced in Section 2.8.3 this can be managed through individual Greek weights or a global weight that only manages the relative importance of the Greek loss in relation to the price loss. As the Greeks differ in magnitude and hence, their relative importance, we decided to construct individual Greek weights. In alignment with Huge and Savine (2020), we defined the raw weights of the individual Greek loss components, in the loss of the DDN_P and DDN_PdivK models, as their inverse variance:

$$w_k^{\text{raw}} = \frac{1}{\text{Var}(y_k) + \epsilon} \tag{122}$$

where $y_k$ is the k:th Greek vector. However, at this point the ratios of the raw weights are informative but the overall scale is arbitrary. Hence, we divide each raw weight by the mean of all raw weights:

$$w_k = \frac{w_k^{\text{raw}}}{\frac{1}{K}\sum_j w_j^{\text{raw}}} \tag{123}$$

This only makes the relative scale matter, i.e.,

$$\frac{1}{K}\sum_k w_k = 1. \tag{124}$$

For the case of the DDN_IV model, as the IV Greeks were derived through division by Vega, some of the Greek values become large in relative terms. Hence, we chose to calculate the variance of an augmented Greek vector where the 99th percentile was removed. Subsequent calculations are analogous.

A global weight was considered for the loss components. Other studies have suggested to balance the loss, primarily increasing the relative importance of the Greek loss (Huge & Savine, 2020). We also considered using weights to prioritize the price loss, $\lambda_y$. Preliminary results indicated that the Greek loss component was naturally larger in magnitude and hence, implementing relative loss component balancing could therefore be relevant. Early tests of the training convergence showed that some models performed better when the loss was balanced, i.e., by amplifying the price component, while other models performed better as the global loss weights was excluded. In the final implementation DDN_P had no global loss component weights, $\lambda_y = 1$ and $\lambda_G = 1$. On the other hand, DDN_PdivK and DDN_IV was implemented using the aforementioned loss balancing strategy such that $\lambda_y = 10$ and $\lambda_G = 1$.

The choice of MSE as the loss objective follows naturally from the theoretical foundations established in Section 2.8.2, as it is the discrete empirical approximation of the Sobolev norm and minimizes the functional distance in the Sobolev space $W^{1,2}(\Omega)$ (Son et al., 2021).

Hence, the total loss was formulated as:

$$\mathcal{L}_{\text{DML}} = \frac{1}{N}\sum_{i=1}^{N}\left[\lambda_y\left(\hat{y}^{(i)} - y^{(i)}\right)^2 + \lambda_G\sum_{j=1}^{n} w_j\left(\hat{z}_j^{(i)} - z_j^{(i)}\right)^2\right]. \tag{125}$$

### 3.3.6 Optimizer and Learning Rate Schedule

The Adam optimization algorithm was chosen because the combined price and derivative loss may produce gradients of very different scales across the network parameters. Its adaptive parameter-wise update rule

can help make training more stable in such settings.

Specifically, we used *AdamW*, a variant of Adam with decoupled weight decay (Loshchilov & Hutter, 2019). Weight decay is a regularization mechanism that discourages large network weights by shrinking them during training. To avoid constraining the model too strongly, the weight decay was set to $10^{-5}$, which is a mild regularization strength.

For the learning rate, we used an initial value of $3 \times 10^{-4}$. Kingma and Ba (2015) recommended $10^{-3}$ as a good default learning rate for Adam, but the somewhat smaller learning rate yielded slightly better validation performance in our experiments. We also found that applying a learning rate scheduler, specifically `CosineAnnealingWarmRestarts` (Loshchilov & Hutter, 2017), further improved performance, which is consistent with the observations of Loshchilov and Hutter (2019). With cosine annealing, the learning rate decays according to a cosine schedule within each restart cycle. If we let $T_{cur}$ denote the number of epochs since the last restart and $T_i$ the length of the current cycle, i.e., the number of epochs before the next restart. Then the learning rate at epoch $t$ is given by:

$$\eta_t = \eta^i_{\min} + \frac{1}{2}(\eta^i_{\max} - \eta^i_{\min}) \left(1 + \cos\left(\frac{T_{\text{cur}}}{T_i}\pi\right)\right), \tag{126}$$

where $\eta^i_{max}$ is the initial, and hence maximum, learning rate and $\eta^i_{min}$ is the minimum rate reached at the end of a cycle. We used $T_i = 150$ and $\eta^i_{min} = 10^{-6}$, and kept all cycles at the same length.

### 3.3.7 Data Scaling

When choosing how to scale parameters, the first question to handle was what individual transformations were optimal for the Heston parameters. The first such transformation we applied to the input variables was taking the square root of the variance parameters, specifically the initial variance $v_0$ and the long-term variance $\theta$. The reason for this comes down to numerical stability and the relationship between variance and volatility.

In the Heston model, $v_0$ and $\theta$ are expressed in variance units, whereas option prices are affected by the corresponding instantaneous volatility, denoted by $\nu = \sqrt{v}$. Since volatility is the square root of variance, changes in variance and volatility are not on the same scale. This distinction becomes important when computing variance-based Greeks. Applying the chain rule gives the following relationship:

$$\frac{\partial P}{\partial v} = \frac{\partial P}{\partial \nu}\frac{1}{2\sqrt{v}}. \tag{127}$$

This reveals that, if the variance $v$ drops close to zero, the denominator $2\sqrt{v}$ becomes tiny. As a result, the calculated Greek explodes toward infinity. This mathematical singularity can easily cause division-by-zero errors or destabilize the neural network during training. By taking the square root of the variance parameters before feeding them into the network, we train the models on volatility rather than variance. When we unscale the Greeks, this transformation cancels out the denominator via the chain rule. This results in stable, bounded, and well-behaved Greeks that are much easier for the neural network to learn.

Before feeding data to the networks for training and validation, further generalized scaling was implemented. The input parameters and the price were all scaled according to the min-max scaling method with the input parameters being zero centered with bounds $[-1, 1]$ (calculated via Equation 95), using the pre-defined theoretical parameter boundaries outlined in Table 1. Since the option prices are non-negative, they were instead transformed to have bounds $[0, 1]$ (calculated via Equation 94) using the min-max values from the training data. Price bounds were not needed for DDN_PdivK as $P/K$ always ranges from 0 to 1, explained in Section 3.3.1. Since the implied volatility values in the dataset were

already well bounded, ranging from 0 to approximately 1.4, no scaling was applied to the IV targets.

An alternative scaling technique commonly used in neural network training is z-score standardization (defined in Section 2.7.7). However, this approach was not chosen for our model. Because the training data is generated synthetically using quasi-random Sobol sequences, the input parameters are uniformly distributed across pre-defined financial bounds rather than being Gaussian-distributed. Furthermore, the synthetic dataset lacks outliers, which is one of the primary use cases for z-score scaling (Han et al., 2012).

Min-max scaling was also specifically paired with the choice of the activation function SiLU (introduced in Section 2.7.3). By scaling inputs to [-1,1], initial pre-activations fall within the SiLU function's non-linear region near the origin. This choice of interval enables the model to learn complex data relationships more efficiently during the early stages of training. However, the last layer's activation function, Softplus, cuts off negative values to guarantee positive prices and IVs, thus matching the [0,1] scaling of price.

Since the neural network receives transformed inputs and predicts transformed outputs, the analytical derivatives must be mapped to the network's scaled domain using the chain rule. As all input parameters were min-max scaled to [-1, 1], the derivative of a raw input parameter $\phi_j \in \{\kappa, \theta, \sigma, \rho, v_0\}$ with respect to its scaled counterpart $\tilde{\phi}_j$ is a constant scaling factor:

$$\frac{\partial \phi_j}{\partial \tilde{\phi}_j} = \frac{\phi_j^{\max} - \phi_j^{\min}}{2} \tag{128}$$

Due to the square root transformations of the variance parameters $\theta$ and $v_0$, we must define a parameter-specific adjustment factor $\alpha_j$ to correctly scale the Greeks:

$$\alpha_j = \begin{cases} 2\sqrt{\phi_j}, & \text{if } \phi_j \in \{\phi, v_0\} \\ 1, & \text{otherwise} \end{cases} \tag{129}$$

The final differential labels used for training depend on the specific target output formulation of the network. We define three distinct model architectures, each requiring an individual scaling approach:

**DDN_P and MLP_P (Price Targets):** For these models, the option price $P$ is min-max scaled to a target variable $\tilde{P} \in [0, 1]$. The derivative of the scaled price with respect to the raw price is $\frac{\partial \tilde{P}}{\partial P} = \frac{1}{P_{\max} - P_{\min}}$. Applying the chain rule, the scaled Greek provided to the networks is:

$$\frac{\partial \tilde{P}}{\partial \tilde{\phi}_j} = \frac{\partial P}{\partial \phi_j} \left( \frac{\phi_j^{\max} - \phi_j^{\min}}{2(P_{\max} - P_{\min})} \right) \alpha_j \tag{130}$$

**DDN_PdivK (Normalized Price Targets):** In this formulation, the target variable is the price normalized by the strike, $y = P/K$, without further min-max scaling on the output. Thus, the corresponding differential label is:

$$\frac{\partial y}{\partial \tilde{\phi}_j} = \frac{1}{K} \frac{\partial P}{\partial \phi_j} \left( \frac{\phi_j^{\max} - \phi_j^{\min}}{2} \right) \alpha_j \tag{131}$$

**DDN_IV (Implied Volatility Targets):** The target variable is the unscaled implied volatility, $\sigma^{\text{IV}}$. The sensitivity of the implied volatility with respect to an input parameter $\phi_j$ is the ratio of the model

price Greek to the Black–Scholes Vega ($\mathcal{V}$) (Section 2.5.4):

$$\frac{\partial \sigma^{\mathrm{IV}}}{\partial \phi_j} = \frac{\frac{\partial P}{\partial \phi_j}}{\mathcal{V}} \tag{132}$$

Scaling this to the network's input domain yields the final training label:

$$\frac{\partial \sigma^{\mathrm{IV}}}{\partial \tilde{\phi}_j} = \left( \frac{\frac{\partial P}{\partial \phi_j}}{\mathcal{V}} \right) \left( \frac{\phi_j^{\max} - \phi_j^{\min}}{2} \right) \alpha_j \tag{133}$$

This presents a limitation, if $\mathcal{V} \to 0$, the IV Greek will approach infinity. To avoid this, we implemented a lower bound for Vega at $10^{-4}$. This bound was chosen because preliminary training results showed that the network struggled to converge when Vega was allowed below this threshold. However, the convergence was still somewhat unstable. To mitigate this, the 99th percentile IV Greeks were removed for each parameter Greek. The loss curves comparing the training convergence before and after these modifications are presented in Appendix C. In total 8.4% of the original training dataset was removed, but this is deemed necessary to make training convergence possible.

### 3.3.8 Summary of Design Choices

We followed the established architecture in the section on differential deep learning closely. The neural network architecture is not pre-determined and was chosen through a literature-based analysis to see what hyperparameters have worked for similar problems through hyperparameter tuning, where the end goal is to identify the hyperparameters with the lowest error and minimal complexity (Sridi & Bilokon, 2023). As the entire network has to be trained using a multitude of different permutations of parameters during hyperparameter tuning, it is computationally expensive. This made a literature based analysis the only feasible option for this study. The final hyperparameters and a summary of the training framework are detailed in Table 5. Note that the outputs $P/K$ and IV were not scaled.

Table 5: Hyperparameters used for training the differential deep network (DDN).

| Hyperparameter | Value | Hyperparameter | Value |
|---|---|---|---|
| **Network Architecture** | | **Optimization (AdamW)** | |
| Input dimension | 8 | Learning rate | $3 \times 10^{-4}$ |
| Hidden layers | 7 | Weight decay | $1 \times 10^{-5}$ |
| Neurons per hidden layer | 250 | Training batch size | 256 |
| Output dimension | 1 | Validation batch size | 2 048 |
| Hidden activation | SiLU | Loss criterion | MSE |
| Output activation | Softplus | Epochs | 450 |
| Hidden weight initialization | Kaiming normal | LR scheduler | CosineAnnealingWR. |
| Output weight initialization | Xavier normal | $T_0$ | 150 |
| Output bias initialization | 0.1 | $T_{\text{mult}}$ | 1 |
| | | $\eta_{\min}$ | $1 \times 10^{-6}$ |
| **Data and Preprocessing** | | **Loss Weighting** | |
| Total samples | 199 820 | DDN_P | $\lambda_P = 1, \lambda_G = 1$ |
| Train / validation split | 90 / 10 | DDN_IV | $\lambda_{\text{IV}} = 10, \lambda_G = 1$ |
| $\theta, v_0$ transform | $\sqrt{\cdot}$ before scaling | DDN_PdivK | $\lambda_{P/K} = 10, \lambda_G = 1$ |
| Input scaling | min–max to $[-1, 1]$ | Per-Greek weights | Inverse var. of $y_{\text{train}}$ |
| Price scaling | min–max to $[0, 1]$ | | |
| Greek scaling | chain-rule | | |

## 3.4 Model Evaluation

To thoroughly assess the performance of the neural network surrogates against the semi-analytical benchmark, a comprehensive evaluation framework was established. This section outlines the generation of the evaluation dataset, pre-processing methodologies, and the specific diagnostic metrics used to evaluate computational efficiency, pricing accuracy, Greek stability and arbitrage compliance. To ensure full reproducibility of the presented results, fixed random seeds were utilized and deterministic backend settings were enabled.

### 3.4.1 Evaluation Dataset and Ground Truth

The input parameters for the evaluation phase were sampled using the same methodology and bounds as when generating the training dataset. However, to establish a ground truth entirely independent of the COS method, the put price was computed via put-call parity (Equation 3) on the semi-closed pricing formula (Equation 36), whose solution is described in Sections 2.2.4 and 2.2.5 to obtain:

$$P(K) = Ke^{-r\tau}(1 - P_2) - S_t(1 - P_1) \tag{134}$$

Further, the price Greeks were calculated by derivation of this formula with respect to the heston param-

eters ($\phi$) as described by Sridi and Bilokon (2023) which gives the following formula:

$$\frac{\partial P(K)}{\partial \phi} = S_t \frac{\partial P_1}{\partial \phi} - Ke^{-r\tau} \frac{\partial P_2}{\partial \phi} \tag{135}$$

While computationally slower, this method provides a highly accurate benchmark for validating both our surrogate models and the COS method itself. The IV and the IV Greeks were then calculated in the same way as in Section 3.2.2.

The resulting evaluation dataset initially contained 5 000 samples, but was refined to 4 989 valid samples after filtering out instances of numerical instability.

### 3.4.2 Pre-processing and Unscaling

Because the networks were trained on a normalized domain, all outputs had to be unscaled by their inverse transformation prior to evaluation. To ensure fair and accurate metric comparisons, every model needed its unique unscaling formulation based on its specific learning target. The minimum and maximum prices derived from the training dataset, denoted as $p_{min}$ and $p_{max}$, served as the global scaling parameters.

To give further comparisons across all architectures, the predicted implied volatilities and IV Greeks from the DDN_IV model were transformed back to the price domain. Put prices were recovered by evaluating the Black–Scholes formula (Equation 13), taking the predicted implied volatilities as input. The price Greeks were reconstructed via the chain rule by multiplying the predicted, unscaled, IV Greeks by the Black–Scholes Vega (Equation 19), evaluated at the predicted IV. Furthermore, Greeks for the standard MLP_P baseline were extracted via automatic differentiation to serve as a non-differential benchmark.

Let $\hat{y}$ denote the normalized network prediction and $g_j$ the normalized partial derivative with respect to the $j$-th scaled input. For an unscaled Heston parameter $\phi_j \in \{\kappa, \theta, \sigma, \rho, v_0\}$, its predefined training boundaries are denoted by $[\phi_j^{min}, \phi_j^{max}]$. To correctly unscale the predicted Greeks, we must also account for the square-root transformation applied to the variance parameters $\theta$ and $v_0$ during the initial feature engineering. By applying the chain rule, we define a parameter-specific adjustment factor $\gamma_j$:

$$\gamma_j = \begin{cases} \frac{1}{2\sqrt{\phi_j}}, & \text{if } \phi_j \in \{\theta, v_0\} \\ 1, & \text{otherwise} \end{cases} \tag{136}$$

Using this framework, knowing that the inputs were min-max scaled to [-1,1] and the prices to [0,1] (specified in Section 3.3.7), the outputs for each distinct network architecture were unscaled as follows:

**DDN_P and MLP_P (Price Targets):** For models trained directly on min-max scaled option prices, the unscaled price $P$ and its corresponding parameter Greeks were recovered by reversing the global price scaling and applying the parameter bounds.

*(Note: Because the MLP_P baseline was trained exclusively on price targets, its scaled Greeks, $g_j$, were extracted during evaluation via automatic differentiation to serve as a non-differential benchmark.)*

The inverse transformations applied to the outputs of both models are defined as:

$$P = \hat{y}(p_{\max} - p_{\min}) + p_{\min} \tag{137}$$

$$\frac{\partial P}{\partial \phi_j} = g_j \left( \frac{2(p_{\max} - p_{\min})}{\phi_j^{\max} - \phi_j^{\min}} \right) \gamma_j \tag{138}$$

**DDN_PdivK (Normalized Price Targets):** For the model trained on strike-normalized prices ($P/K$), the outputs were unscaled by multiplying the prediction by the strike price $K$, where $K = \exp(-m)$ given the log-moneyness $m$:

$$P = \hat{y} \exp(-m) \tag{139}$$

$$\frac{\partial P}{\partial \phi_j} = g_j \left( \frac{2K}{\phi_j^{\max} - \phi_j^{\min}} \right) \gamma_j \tag{140}$$

**DDN_IV (Implied Volatility Targets):** Since implied volatility labels were not explicitly bounded using min-max scaling during training (remaining on their natural scale), the predicted implied volatility $IV$ required no inverse transformation. However, the predicted IV Greeks still required unscaling to account for the normalized input parameters:

$$\sigma^{\text{IV}} = \hat{y} \tag{141}$$

$$\frac{\partial \sigma^{\text{IV}}}{\partial \phi_j} = g_j \left( \frac{2}{\phi_j^{\max} - \phi_j^{\min}} \right) \gamma_j \tag{142}$$

### 3.4.3 Computational Efficiency and COS Benchmark Selection

To benchmark computational efficiency, the neural network models were compared against the COS method using throughput per second as the primary metric. A COS pipeline, similar to that used for creating the training dataset, was implemented. However, to fairly compare the models to COS, the arbitrage constraints for outputs used for data generation were removed, ensuring that the predictions from COS and Let's Be Rational (LBR) reflected only the core pricing formula.

Since the accuracy of the COS method depends on the choice of expansion terms $N$, as explained in Section 2.3.2, the computation time for COS was first evaluated over a wide range of $N$. To illustrate the tradeoff between computation time and accuracy, this evaluation also included the error metrics MAE and MaxAE, computed with respect to the ground truth prices, for each configuration. Based on the results of this test, a suitable $N$ was chosen to be used as the benchmark for subsequent evaluations. The test included both pricing and Greek computation.

The COS-pipeline then contained three distinct phases. These were: computing price (1), computing price and price Greeks (2) and computing price, price Greeks, IV and IV Greeks (3). This structure allowed each surrogate model to be compared against the exact semi-analytical workload it aimed to replace (i.e., DDN_IV to (3) etc.). Furthermore, we hypothesized that the DDN surrogates would exhibit significantly higher throughput per sample as batch sizes increased (due to the hardware acceleration mechanisms detailed in 2.7.8). To empirically test this, inference times were recorded across exponentially increasing

batch sizes, utilizing powers of two, up to the full dataset.

While neural network inferences were performed on GPU, the COS method consists of many heterogeneous and relatively costly numerical computations that are less naturally suited for efficient parallelization. Therefore, all COS benchmarks were executed on CPU, where the implementation ran faster than on the tested GPU setup.

### 3.4.4 Accuracy and Error Diagnostics

Accuracy was evaluated using three primary error metrics: MSE, MAE, and MAPE. These metrics capture the squared errors, average absolute deviations, and scale-independent relative performance, respectively, and were computed for direct prices, price Greeks, implied volatilities (IV) and IV Greeks.

For further analysis of these metrics, we examined the error distributions using heatmaps and dumbbell plots. The heatmaps visualize how pricing errors vary across log-moneyness and time to maturity, isolating regions where the models over- or underperform. Meanwhile, the dumbbell plots compare the median, mean, and 99th quantile to illustrate the overall error spread and the impact of extreme outliers. A pre-visualization sanitation algorithm removed non-finite values (e.g. NaN, $\pm\infty$), preventing divergent predictions from dominating the color scales of the heatmaps and the axes of the dumbbell plots, whilst reporting all removed predictions and the cause for removal.

### 3.4.5 Greek Stability and Arbitrage Compliance

When evaluating Greeks with respect to model parameters, stability is arguably more important than pointwise accuracy, because producing smooth gradients allows the surrogate model to be utilized as a highly efficient engine for gradient-based calibration (Huge & Savine, 2020; C. Zhang et al., 2025).

To evaluate the smoothness and numerical stability of the predicted Greeks, we estimated the empirical Lipschitz constants of the surrogate models (Equation 119). Because an exhaustive pairwise search across the full calibration set was infeasible based on our chosen hardware, we employed a Monte Carlo approach. We sampled 20 000 random pairs of input parameters to compute the networks' difference quotients, using the $L_2$ norm for the parameter space distance, yielding an approximation of the lower bound of the true empirical Lipschitz constant. To provide a comprehensive view of the models' stability, we extracted the maximum, median and 99th quantile slope from the sampled pairs. While the absolute maximum slope captures worst-case instability and the median reflects the baseline behavior, the upper quantile serves as the most robust metric. By filtering out isolated extreme outliers, it provides a clear picture of the network's general smoothness across the parameter space. The model constants were compared to those of COS, expressed in a ratio. This was used as the primary evaluation metric.

Visually, the models were benchmarked against the COS method following the framework established by Chakhoyan et al. (2025). We compared predicted prices, absolute pricing errors, and the Greeks $\Delta$ and $\Gamma$, computed from the network's m-derivatives via the chain rule and evaluated at $S = 1$. This evaluation was performed across the log-moneyness domain, ranging from -2 to 2, for three distinct state vectors, deliberately chosen to represent different market regimes: a calm market (low volatility), a normal market (median parameter values), and a stressed market (high volatility and strong negative correlation). This approach enabled a comprehensive visual inspection of Greek stability over different market states. By evaluating the models' capacity to extrapolate outside the spatial training domain (beyond $|m| > 1.5$) and their ability to generalize to non-explicit training targets (i.e., the state Greeks $\Delta$ and $\Gamma$), we could verify the gradient smoothness required for robust gradient-based calibration.

As the models differ in output targets and scaling, the calculated state Greeks must account for this

fact. Additionally, the chain rule implications from the scaling of the log-moneyness network input affect these sensitivities. Because we evaluate the Greeks at $S = 1$ to analyze the spatial derivatives across the log-moneyness domain, the base relationships are $\Delta = P + \frac{\partial P}{\partial m}$ and $\Gamma = \frac{\partial P}{\partial m} + \frac{\partial^2 P}{\partial m^2}$. Calculating Delta and Gamma for each model yields the following formulas:

**DDN_P and MLP_P (Price Targets):**

$$\Delta = P + \left(\beta_P \alpha_m \frac{\partial \tilde{P}}{\partial \tilde{m}}\right) \tag{143}$$

$$\Gamma = \left(\beta_P \alpha_m \frac{\partial \tilde{P}}{\partial \tilde{m}}\right) + \left(\beta_P \alpha_m^2 \frac{\partial^2 \tilde{P}}{\partial \tilde{m}^2}\right) \tag{144}$$

where $\alpha_m = 2/(m_{max} - m_{min})$ and $\beta_P = (P_{max} - P_{min})$. P is the unscaled price obtained by Equation 137. Additionally, $\frac{\partial \tilde{P}}{\partial \tilde{m}}$ and $\frac{\partial^2 \tilde{P}}{\partial \tilde{m}^2}$ are obtained through automatic differentiation.

**DDN_PdivK (Normalized Price Targets):**

$$\Delta = e^{-m} \cdot \alpha_m \cdot \frac{\partial u}{\partial \tilde{m}} \tag{145}$$

$$\Gamma = e^{-m}\left[\alpha_m^2 \frac{\partial^2 u}{\partial \tilde{m}^2} - \alpha_m \frac{\partial u}{\partial \tilde{m}}\right] \tag{146}$$

where $u = P/K$. We evaluate the Greeks at S=1 and, therefore, $K = e^{-m}$. Again, $\frac{\partial u}{\partial \tilde{m}}$ and $\frac{\partial^2 u}{\partial \tilde{m}^2}$ are obtained by automatic differentiation.

**DDN_IV (Implied Volatility Targets):**

$$\Delta = P_{BS} + \frac{\partial P_{BS}}{\partial m} + \mathcal{V}\left(\alpha_m \frac{\partial \sigma}{\partial \tilde{m}}\right) \tag{147}$$

$$\Gamma = \frac{\partial P_{BS}}{\partial m} + \frac{\partial^2 P_{BS}}{\partial m^2} + \mathcal{V}\left(\alpha_m \frac{\partial \sigma}{\partial \tilde{m}}\right) + \mathcal{V}\left(\alpha_m^2 \frac{\partial^2 \sigma}{\partial \tilde{m}^2}\right) + 2\mathcal{V}_m\left(\alpha_m \frac{\partial \sigma}{\partial \tilde{m}}\right) + \text{Volga}\left(\alpha_m \frac{\partial \sigma}{\partial \tilde{m}}\right)^2 \tag{148}$$

where $\frac{\partial P_{BS}}{\partial m} = \Delta_{BS} - P_{BS}$ and $\frac{\partial^2 P_{BS}}{\partial m^2} = \Gamma_{BS} - \Delta_{BS} + P_{BS}$, with $\Delta_{BS}$ and $\Gamma_{BS}$ representing the standard analytic Black–Scholes Delta and Gamma evaluated at unit spot ($S = 1$).The full derivation of these formulas is detailed in Appendix B.

To further evaluate the robustness of the predicted Greeks, metrics for directional accuracy (the proportion of correctly signed Greeks) were implemented.

Lastly, arbitrage diagnostics were divided into two main components: dataset-based bound checks on the observed predictions and dense-grid sweeps on synthetic surface grids. The synthetic grids were constructed by stratifying the log-moneyness and time to maturity domains to uncover violations across the price-space, IV-space, and IV-implied-price domains. Across these grids, we evaluated bound breaches, calendar and butterfly spread violations, monotonicity in strike, and calendar violations in total variance on the IV grid. The results were summarized by reporting the mean and standard deviation of the violation percentages. Finally, we performed a point-wise check on the put Delta bounds. This step measured the proportion of predicted put options that successfully stayed within the theoretical [-1, 0]

interval (as defined in Section 2.1.5), while also tracking the extreme high and low Delta values produced by each model.

## 3.5 Experimental Setup

### 3.5.1 Hardware Environment

For timing evaluations, the best hardware suited to the characteristics of each method was used. COS timing benchmarks were therefore executed on an Apple M2 CPU, where the implementation ran faster than on the tested GPU setup, while neural network inference was performed on an NVIDIA RTX 2070 GPU (8GB VRAM) on a separate system. The reported runtimes and throughputs therefore reflect both algorithmic differences and hardware-specific acceleration. This means that the results should be treated as an indicator of practical performance rather than as a strictly controlled comparison.

### 3.5.2 Frameworks

The neural network models were implemented using PyTorch, which enables efficient training and inference. The algorithm for the COS method was implemented using JAX, as mentioned in Section 3.2.1. For implied volatility computations, the *Let's Be Rational* framework, introduced in Section 2.5.3, was employed. Data preprocessing and analysis were performed using Pandas, and NumPy was used for general numerical computations and array-based operations.

# 4 Results

This section presents the evaluation results for all surrogate models relative to the COS model benchmark. It covers computational efficiency, pricing accuracy, Greek stability, and arbitrage compliance.

## 4.1 Computational Efficiency

Table 6 illustrates the tradeoff between accuracy and computational cost for the COS method, depending on the number of expansion terms $N$ used during the computation. Based on the convergence results, $N = 512$ was chosen as the benchmark for subsequent evaluations as it provides a reasonable balance between accuracy and computational cost. In this configuration, the pricing error is considered to be sufficiently small to act as a robust benchmark while avoiding unnecessarily high computational costs.

Table 6: COS convergence and computational cost for different numbers of expansion terms $N$ in the COS method. Time is reported for 4 989 test samples.

| **Metric** | $N = 128$ | $N = 256$ | $N = 512$ | $N = 1024$ | $N = 2048$ | $N = 4096$ | $N = 8192$ |
|---|---|---|---|---|---|---|---|
| MAE | 1.67e-3 | 4.36e-4 | 9.68e-5 | 2.02e-5 | 1.65e-6 | 2.98e-7 | 1.34e-8 |
| MaxAE | 6.95e-1 | 3.68e-1 | 8.45e-2 | 2.20e-2 | 1.38e-3 | 7.15e-4 | 2.62e-5 |
| Time (s) | 1.32 | 1.95 | 3.10 | 4.43 | 6.96 | 9.04 | 13.21 |

Table 7 reports the pricing time and the corresponding throughput for the COS benchmark and the

different neural network models. The results provide an indication of the speed advantage offered by neural network-based models compared to the semi-analytical benchmark.

Table 7: Inference time and throughput on 4 989 test samples. Speedup is relative to the corresponding COS benchmark. COS runtimes were measured on CPU and neural network runtimes on GPU; see Section 3.5.1

| **Task** | **Model** | **Runtime (s)** | **Throughput ($s^{-1}$)** | **Speedup** |
|---|---|---|---|---|
| Price | COS (stage 1) | $7.86 \times 10^{-1}$ | $6.34 \times 10^{3}$ | 1 |
| | MLP_P | $2.47 \times 10^{-3}$ | $2.05 \times 10^{6}$ | 323 |
| Price & Greeks | COS (stage 2) | 2.79 | $1.79 \times 10^{3}$ | 1 |
| | DDN_P | $4.12 \times 10^{-3}$ | $1.23 \times 10^{6}$ | 687 |
| | DDN_PdivK | $4.29 \times 10^{-3}$ | $1.19 \times 10^{6}$ | 664 |
| IV & IV Greeks | COS (stage 3) | 3.10 | $1.61 \times 10^{3}$ | 1 |
| | DDN_IV | $4.22 \times 10^{-3}$ | $1.23 \times 10^{6}$ | 764 |

Figure 7 illustrates how the throughput varies with batch size for each method. From the graph, it is evident that the neural network surrogates benefit substantially from increased batch sizes, which in turn allows for more efficient parallelization and hardware acceleration. All network inferences achieve their highest throughput at a batch size of 4 989, corresponding to the full test dataset. However, the continued increase in throughput at the largest batch size suggests that the models have not yet fully utilized the available hardware at the tested batch sizes. This indicates that higher throughput could potentially be achieved for a larger dataset, although this is ultimately constrained by hardware limitations.

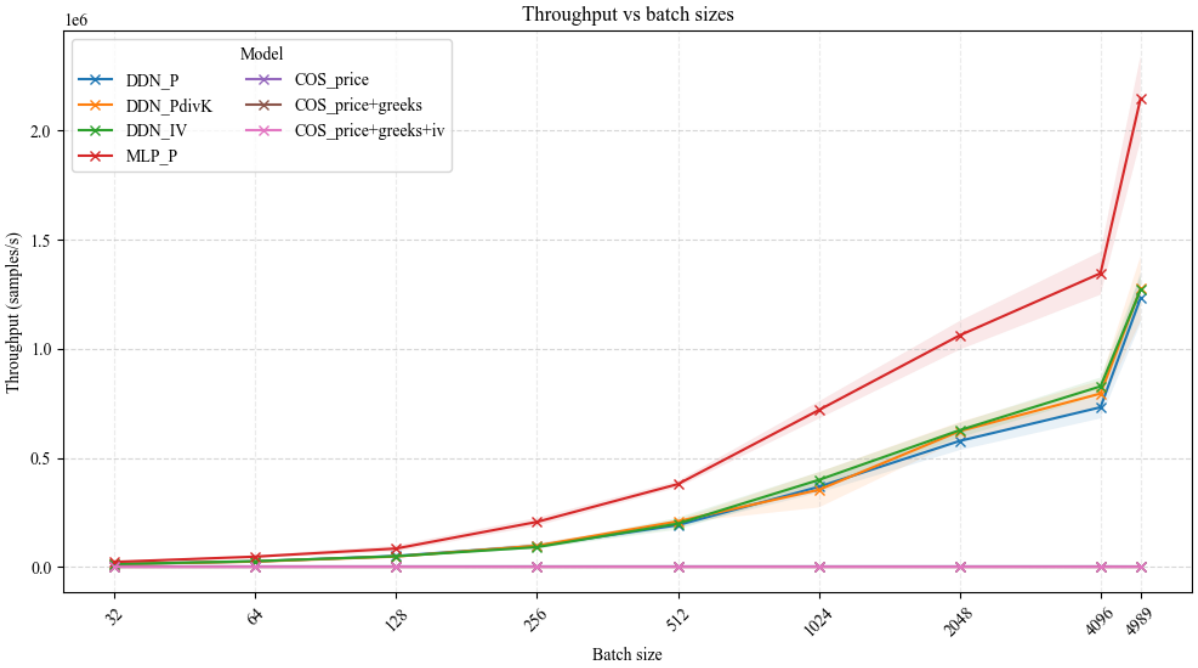



Figure 7: Throughput for different batch sizes (Neural Networks: Mean ± Std over 30 runs)

## 4.2 Accuracy

### 4.2.1 MSE and MAE

When predicting raw option prices, Greeks, IV and IV Greeks, the DDN_IV model demonstrates high precision, outperforming all the other models and the COS benchmark when it comes to MSE except on $\kappa$ where DDN_P, DDN_PdivK and MLP_P perform better (see Table 8). DDN_PdivK shows good performance on price and price Greeks with accuracy close to the DDN_IV model and better than the COS benchmark and the other models on all targets. When it comes to DDN_P it performs worse on MSE than DDN_PdivK but better than the both COS benchmark and the MLP_P model on all targets. The worst performing model for evaluating the raw price and Greeks is the MLP_P model, but it still performs better than the COS benchmark on all targets except $\theta$ and $\rho$.

Table 8: Mean squared error (MSE) by target and model.

| | | Model | | | | |
|---|---|---|---|---|---|---|
| **Group** | **Target** | **COS** | **DDN_P** | **DDN_PdivK** | **DDN_IV** | **MLP_P** |
| Price & Price Greeks | Price | $3.551 \times 10^{-6}$ | $1.942 \times 10^{-8}$ | $1.188 \times 10^{-8}$ | $\mathbf{8.038 \times 10^{-9}}$ | $1.760 \times 10^{-7}$ |
| | $\theta$ | $7.057 \times 10^{-5}$ | $1.338 \times 10^{-5}$ | $9.336 \times 10^{-6}$ | $\mathbf{5.434 \times 10^{-6}}$ | $2.495 \times 10^{-4}$ |
| | $\sigma$ | $3.273 \times 10^{-5}$ | $1.097 \times 10^{-7}$ | $9.571 \times 10^{-8}$ | $\mathbf{6.728 \times 10^{-8}}$ | $1.933 \times 10^{-6}$ |
| | $\rho$ | $4.130 \times 10^{-6}$ | $1.818 \times 10^{-7}$ | $1.481 \times 10^{-7}$ | $\mathbf{5.204 \times 10^{-8}}$ | $4.495 \times 10^{-6}$ |
| | $\kappa$ | $1.158 \times 10^{-2}$ | $9.353 \times 10^{-8}$ | $\mathbf{7.764 \times 10^{-8}}$ | $1.890 \times 10^{-6}$ | $1.836 \times 10^{-6}$ |
| | $v_0$ | $9.057 \times 10^{-5}$ | $3.673 \times 10^{-6}$ | $2.697 \times 10^{-6}$ | $\mathbf{2.422 \times 10^{-6}}$ | $4.973 \times 10^{-5}$ |
| IV & IV Greeks | IV | $9.65 \times 10^{-4\dagger}$ | – | – | $\mathbf{5.705 \times 10^{-4}}$ | – |
| | IV $\theta$ | $2.492 \times 10^{-1}$ | – | – | $\mathbf{4.456 \times 10^{-4}}$ | – |
| | IV $\sigma$ | $8.671 \times 10^{-1}$ | – | – | $\mathbf{1.693 \times 10^{-4}}$ | – |
| | IV $\rho$ | $3.617 \times 10^{-1}$ | – | – | $\mathbf{2.484 \times 10^{-3}}$ | – |
| | IV $\kappa$ | 6.207 | – | – | $\mathbf{7.034 \times 10^{-6}}$ | – |
| | IV $v_0$ | $1.534 \times 10^{-1}$ | – | – | $\mathbf{3.734 \times 10^{-3}}$ | – |

† Computed on the 4 927 of 4 989 test samples where the COS IV inversion converged. The full-sample MSE is unbounded.

When considering MAE (see Table 9), the errors are not squared, meaning that large errors dominate less. The results are similar, with the exception that the DDN_IV model performs best on all targets. The COS benchmark fails to converge to some IV values (explored later in Section 4.2.3), this breaks the method when calculating MAE and MSE for the full 4 989 test samples. This is not a problem for DDN_IV since it predicts the IV instead of inverting it. Further, the DDN_PdivK and DDN_P models also show a high accuracy and are overall better than the COS benchmark exhibiting lower MAE on price and all price Greeks except $\theta$ and $\rho$. Once again, the MLP_P model performs the worst out of all the models only performing better than the COS benchmark on $\kappa$.

Table 9: Mean absolute error (MAE) by target and model.

| | | Model | | | | |
|---|---|---|---|---|---|---|
| **Group** | **Target** | **COS** | **DDN_P** | **DDN_PdivK** | **DDN_IV** | **MLP_P** |
| Price & Price Greeks | Price | $9.680 \times 10^{-5}$ | $9.030 \times 10^{-5}$ | $5.793 \times 10^{-5}$ | $\mathbf{1.709 \times 10^{-5}}$ | $2.485 \times 10^{-4}$ |
| | $\theta$ | $4.460 \times 10^{-4}$ | $6.800 \times 10^{-4}$ | $4.891 \times 10^{-4}$ | $\mathbf{1.640 \times 10^{-4}}$ | $2.116 \times 10^{-3}$ |
| | $\sigma$ | $3.480 \times 10^{-4}$ | $1.790 \times 10^{-4}$ | $1.543 \times 10^{-4}$ | $\mathbf{5.267 \times 10^{-5}}$ | $7.767 \times 10^{-4}$ |
| | $\rho$ | $1.570 \times 10^{-4}$ | $2.340 \times 10^{-4}$ | $1.861 \times 10^{-4}$ | $\mathbf{6.859 \times 10^{-5}}$ | $1.126 \times 10^{-3}$ |
| | $\kappa$ | $4.400 \times 10^{-3}$ | $1.190 \times 10^{-4}$ | $9.839 \times 10^{-5}$ | $\mathbf{7.218 \times 10^{-5}}$ | $3.597 \times 10^{-4}$ |
| | $v_0$ | $5.460 \times 10^{-4}$ | $4.770 \times 10^{-4}$ | $3.551 \times 10^{-4}$ | $\mathbf{1.459 \times 10^{-4}}$ | $1.637 \times 10^{-3}$ |
| IV & IV Greeks | IV | $2.549 \times 10^{-3\dagger}$ | – | – | $\mathbf{2.090 \times 10^{-3}}$ | – |
| | IV $\theta$ | $2.440 \times 10^{-2}$ | – | – | $\mathbf{1.691 \times 10^{-3}}$ | – |
| | IV $\sigma$ | $3.500 \times 10^{-2}$ | – | – | $\mathbf{1.235 \times 10^{-3}}$ | – |
| | IV $\rho$ | $2.650 \times 10^{-2}$ | – | – | $\mathbf{4.082 \times 10^{-3}}$ | – |
| | IV $\kappa$ | $8.320 \times 10^{-2}$ | – | – | $\mathbf{2.713 \times 10^{-4}}$ | – |
| | IV $v_0$ | $2.390 \times 10^{-2}$ | – | – | $\mathbf{5.539 \times 10^{-3}}$ | – |

† Computed on the 4 927 of 4 989 test samples where the COS IV inversion converged. The full-sample MAE is unbounded.

The dumbbell plot (Figure 8) shows a representation of the MSE, the median squared error (q500SE) and the 99th quantile squared error (q990SE). The quantile errors provide a more detailed view of the error distribution, helping to illustrate how a small number of extreme outliers might affect the observed errors. The dumbbell plot is capped on the left side leaving out the real q500SE values for the semi-analytical COS benchmark (gray). The real q500SE for the COS benchmark is generally in the same order of magnitude for all outputs and spans between $10^{-27}$ and $10^{-30}$. Since these are squared errors, they correspond to residual magnitudes around $10^{-14}$ to $10^{-15}$, meaning that the median errors are close to machine precision. However, its performance severely degrades in the tails due to extreme outliers, which can be seen by the massive horizontal distance between its q500SE and q990SE. Further indications that the model is highly exposed to outliers can be inferred from the fact that the most extreme outliers have such a large magnitude that the MSE is greater than the q990SE.

In contrast, the deep differential networks exhibit significantly tighter error bounds, indicating greater robustness and consistency across diverse parameter combinations. The DDN_PdivK model (green) shows strong stability for direct price and price Greeks, keeping its 99th percentile and mean errors more tightly clustered and consistently outperforming the DDN_P model (orange) in these categories. Similarly, for price, price Greeks, IV and IV Greeks, the DDN_IV model (blue) demonstrates a much narrower spread than the COS baseline and has a q500SE that is constantly lower than the other networks, highlighting its overall relatively strong performance, but it still faces the same issue as the COS benchmark, where, for some targets, the most extreme outliers push the MSE higher than the q990SE. MLP_P (red) also shows narrow bounds but has overall worse accuracy than the other models. Notably, DDN_IV only performs worse than the other models on the kappa price Greek in terms of MSE. However, the low median and q990SE relative to the other models indicate that some large outliers are present.

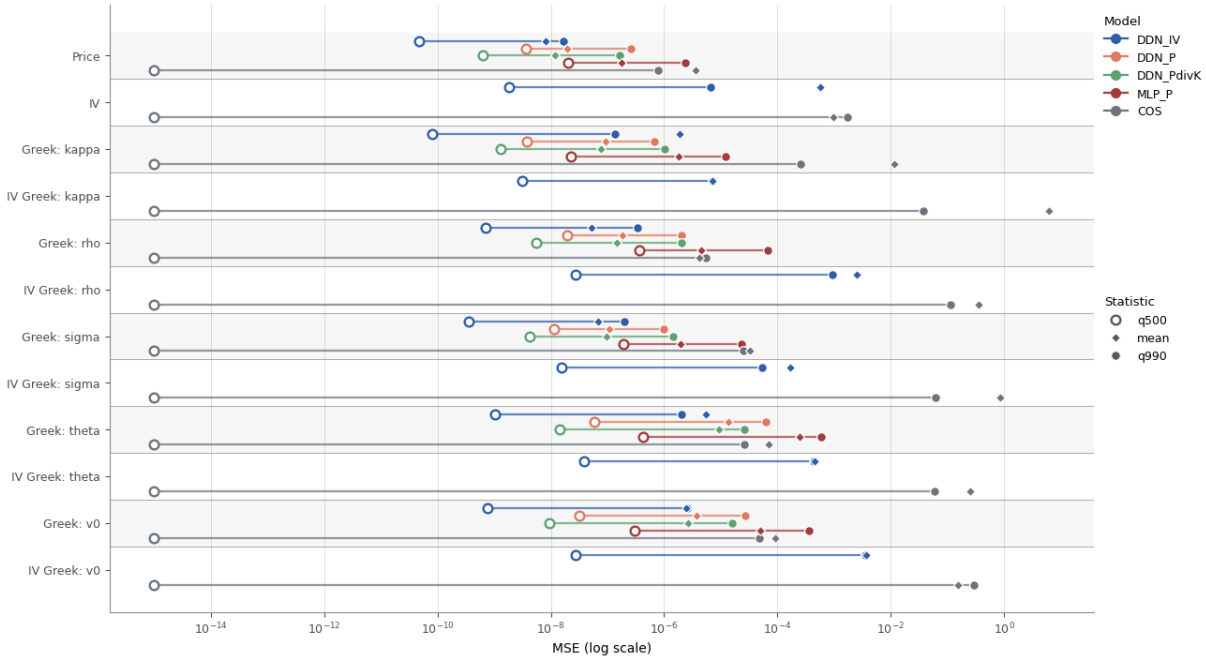
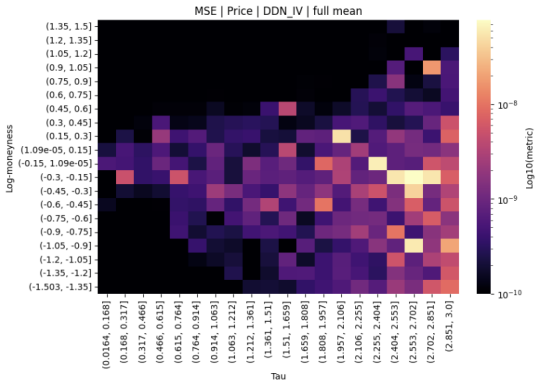


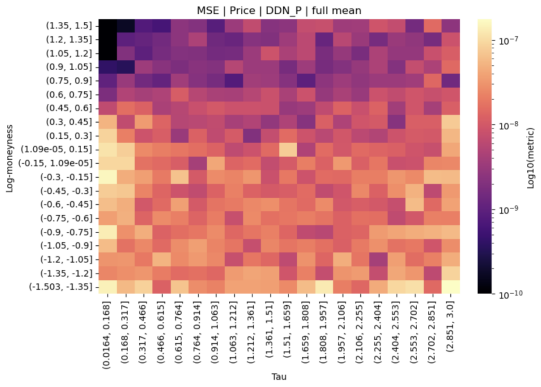

Figure 8: Model performance comparison across error quantiles, all output targets ($N = 4\,989$).

### 4.2.2 MSE Heatmaps

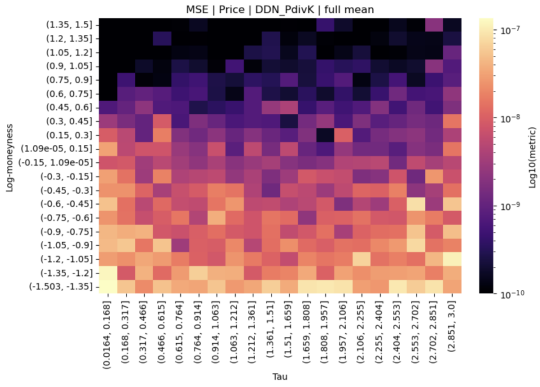
To understand how the models perform across different market regimes, the MSE was also evaluated across two-dimensional grids of time to maturity ($\tau$) and log-moneyness ($m$) as heatmaps (Figure 9). While MAE was also analyzed in this manner, it exhibited the exact same tendencies as MSE but in a dampened manner. Therefore, the MSE heatmaps provide a much clearer and more pronounced visualization of the models' structural error patterns.



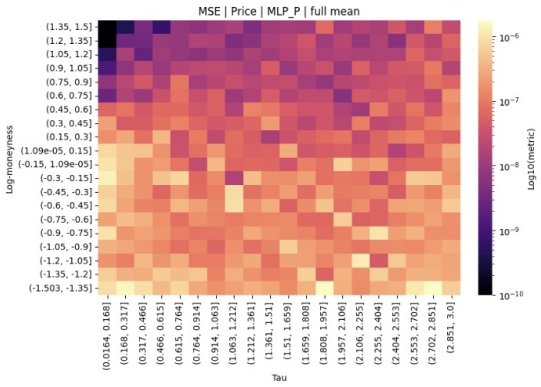

(a) DDN_IV (price).



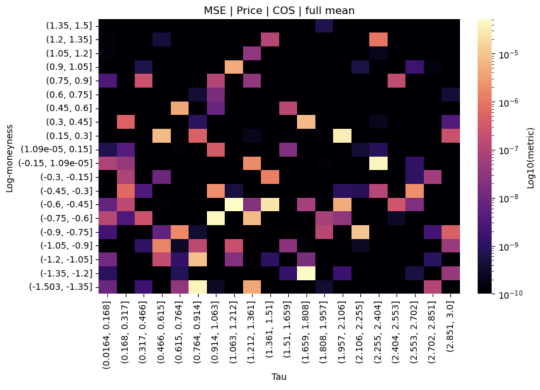

(b) DDN_P.



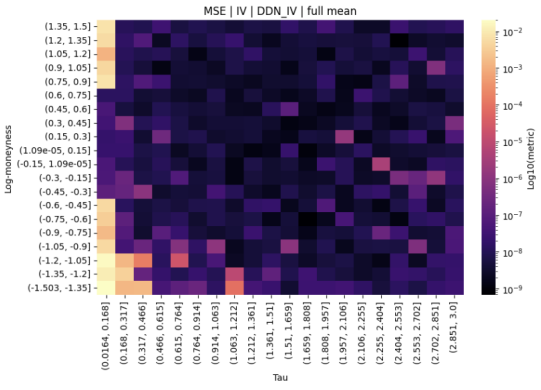

(c) DDN_PdivK.



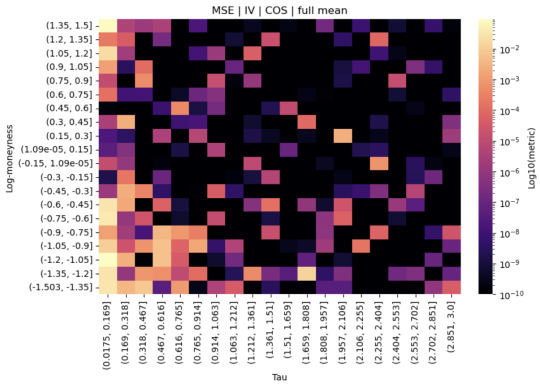

(d) MLP_P.




(e) COS (price).




(f) DDN_IV (IV).




(g) COS (IV).

Figure 9: MSE heatmaps across log-moneyness and time to maturity.

Looking at the COS benchmark's price errors (9e) and IV errors (9g), there is a distinct lack of structural patterns among most regimes. Instead of smooth transitions, the heatmaps resemble scattered noise. Andersen and Piterbarg (2007) states that models like Heston can exhibit fat tails with fairly typical values of parameters. As explained by C. Wang (2017) option prices can be unstable for too low truncation ranges where the underlying model exhibits fat tails. This explains why, for the COS benchmark, specific parameter combinations can result in randomly scattered deviations rather than regime-specific weaknesses. Despite scattered noise some trends can be seen when predicting IV (9g) for options with short maturities that are deep in-the-money (ITM) or deep out-of-the-money (OTM). The deep ITM areas are where the log-moneyness is low, while the deep OTM is where the log-moneyness is high. In these areas, especially for short maturities deep ITM, the COS benchmark performs significantly worse (which will further be presented in Section 4.2.3).

The DDN_IV model shows great accuracy for predicting IV in most market regimes (9f) but exhibits higher MSE in short maturity deep ITM or deep OTM regions. This breakdown and its connection to the evaluation dataset will be discussed in the next section. When looking at the DDN_IV model's price performance (9a) the reverse pattern is seen, while showing good accuracy across all market regimes, it performs best at predicting the price for options with short maturities that are deep ITM or deep OTM.

This reverse pattern for the price and the IV during short maturity and deep OTM/ITM occurs because the price from the DDN_IV is numerically calculated by applying the predicted IV in the Black–Scholes formula for a put derived by applying put-call parity (Equation 3 in Section 2.1.1) to (Equation 13 in Section 2.1.3) as shown below:

$$P(S,t) = Ke^{-r\tau}\mathcal{N}(-d_2) - S\mathcal{N}(-d_1) \tag{149}$$

where

$$d_1 = \frac{\ln(S/K) + \left(r + \frac{1}{2}\sigma^2\right)\tau}{\sigma\sqrt{\tau}}, \qquad d_2 = d_1 - \sigma\sqrt{\tau}, \tag{150}$$

When the put option is deep OTM ($S$ much larger than $K$) the log-moneyness ($m = \ln(S/K)$) becomes positive. When this occurs simultaneously as the time to maturity ($\tau$) $\to 0$, the terms $d_1$ and $d_2$ approach $\infty$, resulting in the normal cumulative distribution function $\mathcal{N}(-d_i)$ approaching 0. This in turn causes the put price (Equation 149) to move very close to 0, which is a good prediction for a put option in this market state. When the put option at the same $\tau$ is deep ITM ($K$ much larger than $S$) the log-moneyness is negative resulting in the terms $d_1$ and $d_2$ approaching $-\infty$, which yields that $\mathcal{N}(-d_i)$ approaches 1. From this, the put price (Equation 149) becomes $Ke^{-r(\tau)} - S$ which is the expected payoff of the put option ITM (i.e., a good prediction for this market state). These two results for put options deep OTM/ITM show that the price is barely affected by the IV in these market states, which explains why the predicted price for the DDN_IV model is good in the regions where it is inferior when predicting IV. However, it should also be noted that overall DDN_IV is quite uniform in its performance over the entire domain, especially for IVs, without showing distinct bias towards either OTM or ITM regimes for the vast majority of maturities.

Models trained directly on option prices exhibit errors that scale proportionally with the absolute magnitude of the option price. MLP_P (Figure 9d) shows a high MSE for most market regimes, except for short maturities OTM, where the price is essentially zero. Its error heatmap acts essentially as a proxy for the absolute option price itself, where it is highest in the deep ITM regions (where absolute prices are largest) and lowest in the deep OTM short-maturity regions (where prices are practically zero). The reason for this is that the magnitude of the true prices and predicted prices for short maturity deep OTM regions are very small, and hence produce small absolute errors. Even if these errors are large relative to the true price it will result in a smaller MSE in these regions. This can be seen in appendix A

which presents the corresponding MAPE heatmaps, showing that MAPE generally is lower in the areas where the MSE is high, proving that the elevated MSE in the ITM regions is essentially an artifact of the underlying magnitude of the targets. The Differential training improves upon the MLP_P baseline where the DDN_P model (Figure 9b) shows a similar pattern with superior accuracy across nearly all regimes, with strongest accuracy occurring in the OTM regimes, especially for short maturities. It also performs worse in the ITM regions but it is once again important to note that this does not necessarily represent a structural failure of the neural network to understand the pricing physics. Rather, like for the MLP_P model, it reflects that absolute errors scale with absolute prices. A smaller relative error on a highly valuable deep ITM option will yield larger MSE than a severe relative error on a nearly worthless OTM option would.

The DDN_PdivK model (Figure 9c), exhibits a similar but more accurate pattern compared to the models trained on prices. By dividing the target price by the strike, the vast absolute price scale of the Heston model is flattened, bounded strictly between 0 and 1 when predicted and then scaled back to normal price when evaluated. This transformation results in a more stable MSE across the x-axis. Overall, DDN_PdivK shows a higher MSE for ITM- compared to OTM-options. Since the predicted price for DDN_PdivK is bounded between $[0, K]$, the deep ITM options will approach the values of the strike prices, which makes the absolute errors here larger, yielding a higher MSE for these regimes.

### 4.2.3 IV Surfaces

The implied volatility is often analyzed from the perspective of the IV surface. As established in Section 2.1.4, the Heston model was introduced to address the constant volatility assumption of BS. Hence, IV surfaces modeled using the Heston model have the characteristic skewed IV surface that real markets demonstrate. Figure 10 compares the IV surfaces produced by DDN_IV and COS under a state vector that represents normal conditions. The state was modeled as $\kappa = 2.523$, $\theta = 0.7072$, $v_0 = 0.7079$, $\rho = -0.4743$, $\sigma = 0.8004$, $r = 4.496\%$. Surfaces for DDN_IV and COS is shown in Figures 10a and 10b, respectively, where log-moneyness spans from $-2$ to 2 and time to maturity from three years to two weeks (2/52 years) in a 150 by 150 point grid. It is evident that both models capture the skew of the IV surface and that DDN_IV is smooth. However, as time to maturity approaches two weeks, the COS model starts to break down, evident by the steep IV increases in that region. Therefore a second pair of surfaces was generated over the same parameter state, but in a shorter time to maturity domain. Figures 10c and 10d is evaluated under time to maturity ranging from one month (1/12 years) to 0.0 years, again in a 150 by 150 point grid. Here, the breakdown of the COS model is intensified and produces IVs approaching 25. This is more prevalent as log-moneyness moves into the tails (ITM/OTM). Meanwhile, DDN_IV continues to produce a smooth surface over the entire domain, even outside of training bounds, showing strong extrapolation to OOD domains ($T < 1/52$ and $|lm| > 1.5$).

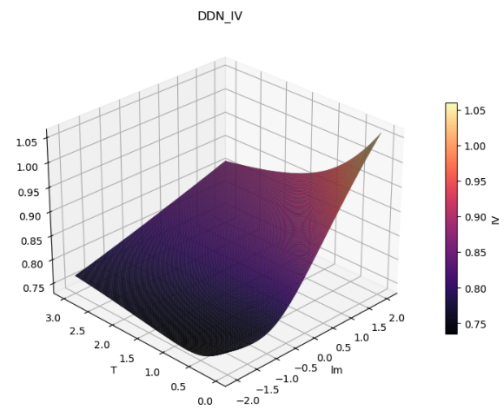
Note that the missing gaps in the COS plot are caused either because Let's Be Rational returned unrealistically large values or because it returned IVs that were zero. In total, this amounted to 6054 (27%) IVs for the short time to maturity surface and 105 (0.5%) for the longer time to maturity surface. As observed, the gaps mainly occur at short maturities and extreme log-moneyness, corresponding to deep ITM or deep OTM options. This is, however, not unexpected since these regions are associated with very small Vegas. From the Black–Scholes Vega expression, given by Equation 19, a short time to maturity directly reduces Vega through the $\sqrt{\tau}$ term. Moreover, a small $\tau$ and large absolute log-moneyness causes the $|d_1|$-term to become large. This in turn yields a very small value of $\phi(d_1)$ and therefore an even smaller Vega.

When Vega is close to zero, the Black–Scholes price function becomes nearly flat with respect to volatility. As a result, very different implied volatilities may correspond to almost the same option price. The inverse

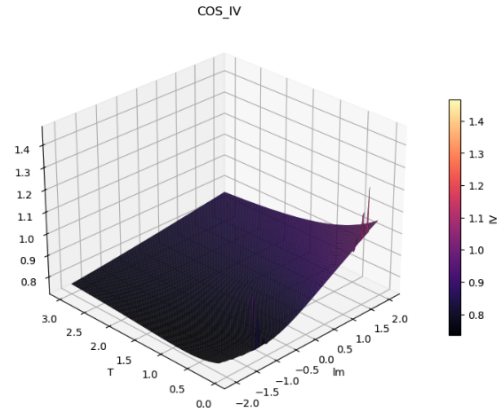
problem described in Section 2.5 therefore becomes ill-conditioned, meaning that even small pricing errors can be strongly amplified in the implied volatility domain. Consequently, the implied volatilities in these regions should not be interpreted as reliable.

It is also important to distinguish between convergence problems of a particular numerical algorithm and the conditioning of the inverse problem itself. This issue is therefore not specific to any particular inversion algorithm. As explained in Section 2.5, a small Vega may cause divergence of the Newton–Raphson iterations. Although the Let's Be Rational method is similar, also dividing by Vega, it is designed to handle the inversion more robustly (Jäckel, 2015). Either way, this design cannot remove the ill-conditioning of inverting price to implied volatility when having very small Vega. This is also why a bracketing method such as Brent's method, which does not divide by Vega, would still be solving for a root of an almost flat function. It may therefore converge, but the resulting implied volatility would remain highly sensitive to small errors in the input price.

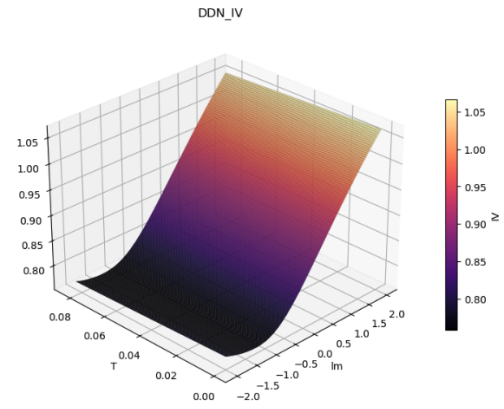
This distinction is important when interpreting the DDN_IV surface, which does not exhibit the same gaps or extreme spikes. This suggests that DDN_IV resolves the numerical instability of the COS model in this region. However, it does not guarantee that the IV behavior in short time to maturity options is necessarily correctly captured.



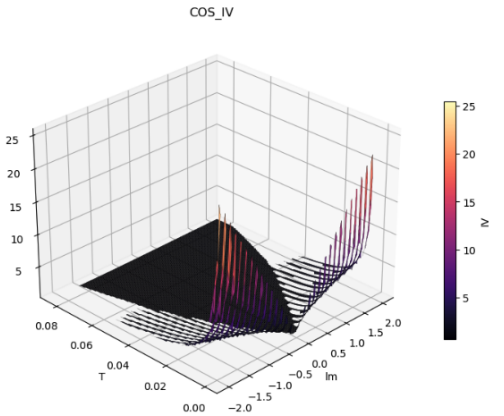

(a) DDN_IV, long maturities ($\tau \in [2/52, 3]$).




(b) COS, long maturities ($\tau \in [2/52, 3]$).




(c) DDN_IV, short maturities ($\tau \in [0, 1/12]$).




(d) COS, short maturities ($\tau \in [0, 1/12]$).

Figure 10: Implied volatility surfaces for DDN_IV and COS across long and short maturities.

As established in Section 4.2.2, DDN_IV showed poor MSE in the short time to maturity OTM/ITM

region. However, this result appears inconsistent with the smooth and realistic IV surface produced by the network in these domains. Hence, the large MSE values might indicate that the IV targets in the evaluation dataset suffer from the same type of numerical instability as the COS model's IVs. To investigate this, an IV surface was constructed to analyze the performance of the conversion in the evaluation dataset. This is shown in Figure 11. Note that since these individual IVs do not, as was the case of the prior surfaces, stem from the same parameter state vector, the surface will not be as smooth. Nevertheless, it provides an indication of the stability of the IV conversion and helps to identify possible outliers in the short time to maturity domain.

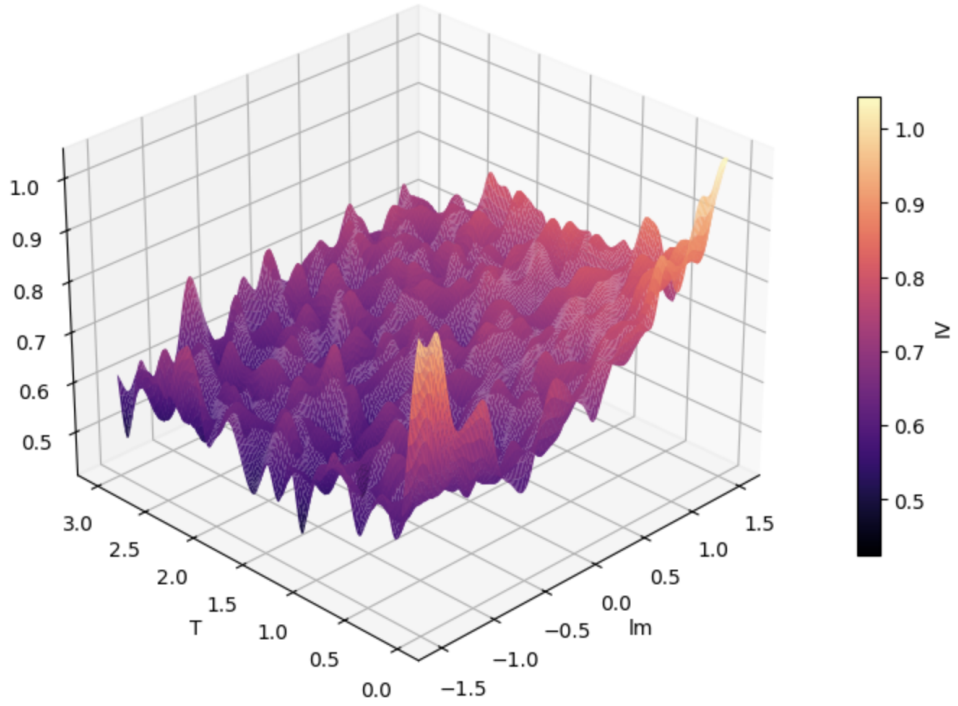



Figure 11: IV surface of the evaluation dataset.

## 4.3 Greek Evaluation

### 4.3.1 Directional Accuracy

Directional accuracy of the model's parameter sensitivities is one of the primary predictors of calibration performance. As the optimizer makes new parameter updates based on the value of the gradient, a sign error would force the optimizer to traverse the loss landscape in the opposite direction of the local minimum. This is therefore one of the main evaluations of Greek robustness and methods of inferring their calibration performance. In Table 10 the directional accuracy over the entire validation dataset is detailed for each model. For the price models, COS performs better compared to the other models on all Greeks, while DDN_PdivK marginally outperforms DDN_P. However, DDN_IV outperforms the COS baseline directional accuracy for all Greeks with respect to IV. MLP_P exhibits the worst directional accuracy across the board.

As shown in the prior section, DDN_IV also performs well on price sensitivities and hence, when converted, inherits the IV Greek directional accuracy, since Vega is strictly non-negative for vanilla options (Hull, 2022, pp. 435–437). This fact effectively makes this model the best predictor of both price and IV Greeks in terms of directional accuracy.

Table 10: Directional Accuracy of Greeks (%)

| **Price Greeks** | | | | | |
|---|---|---|---|---|---|
| Model | $\theta$ | $\sigma$ | $\rho$ | $\kappa$ | $v_0$ |
| DDN_P | 97.03 | 96.97 | 97.17 | 95.85 | 97.92 |
| DDN_PdivK | 97.21 | 96.93 | 97.61 | 96.51 | 97.84 |
| MLP_P | 95.77 | 94.47 | 95.49 | 94.57 | 97.18 |
| COS | **98.54** | **98.36** | **98.62** | **98.38** | **98.94** |
| **IV Greeks** | | | | | |
| Model | $\theta$ | $\sigma$ | $\rho$ | $\kappa$ | $v_0$ |
| DDN_IV | **99.64** | **99.58** | **99.54** | **99.42** | **99.7** |
| COS | 97.80 | 97.63 | 97.78 | 97.55 | 97.86 |

#### 4.3.2 Greek Stability

Figures 12 and 13 show price, absolute price error, Delta, and Gamma for three different market states where DDN_PdivK and DDN_P are compared to the COS benchmark on a small extended domain around the training bounds. They give a rough indication of the stability and extrapolation for the Greeks, which were not explicit training targets. The distinct difference between these results is their capacity for extrapolation. While both models interpolate well within the training domain, the DDN_P model diverges OOD and produces negative Gamma values, which violates the fundamental non-negative Gamma constraint established in Section 2.1.5. This is in contrast to DDN_PdivK which exhibits stronger extrapolative stability, its sensitivities diverging significantly less with respect to the baseline comparison and remaining non-negative. The price plots on the other hand show comparably accurate fits over the entire domain. This result therefore directly shows that while the models differ in extrapolative capacity with respect to the Greeks, both models extrapolate well over the entire price domain. However, looking at the absolute error in terms of price, the structural differences become clear. Both error functions of the two models first peak in the ATM region and then start to diminish. DDN_PdivK exhibits a smooth downturn of the error while the error of DDN_P starts to increase again as we move OOD.

Figure 14 also gives an indication of the performance of MLP_P, which was not trained on parameter Greeks, on the non-explicit training targets Delta and Gamma. These curves clearly follow the structural patterns of the DDN_P model and show poor extrapolation capabilities as we move further into the ITM domain. Note that the divergence of MLP_P in relation to DDN_P is larger, showing even worse performance in OOD regions.

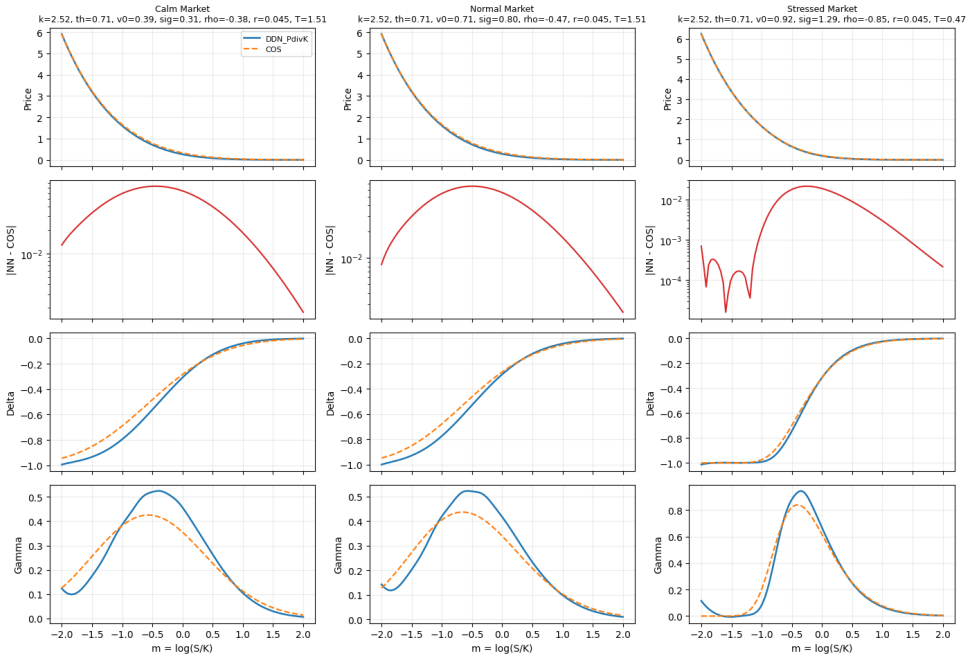



Figure 12: Price, absolute error, Delta, and Gamma across market regimes for DDN_PdivK vs. COS.

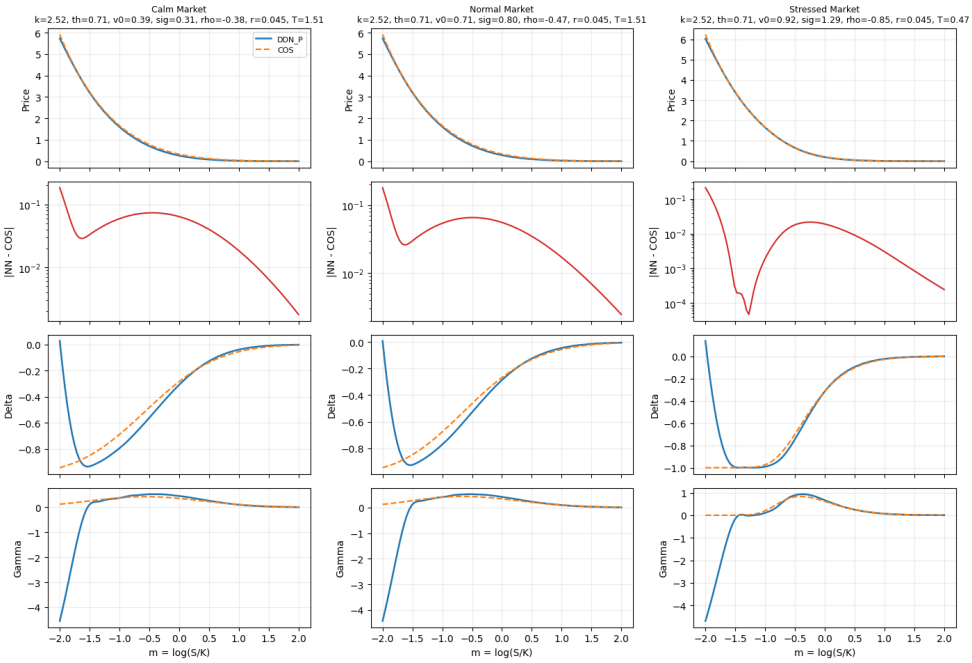



Figure 13: Price, absolute error, Delta, and Gamma across market regimes for DDN_P vs. COS.

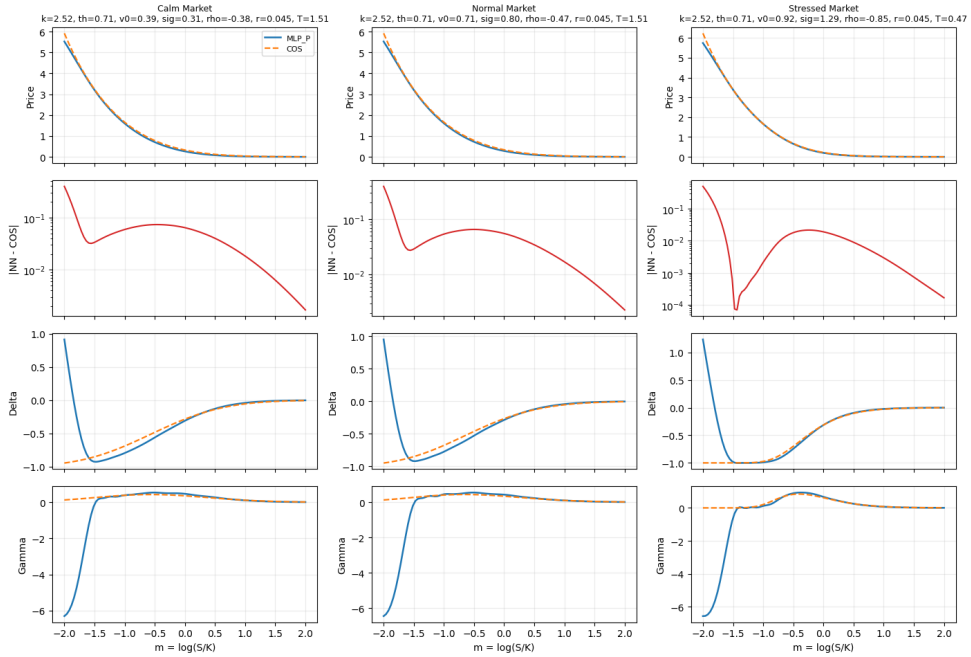



Figure 14: Price, absolute error, Delta, and Gamma across market regimes for MLP_P vs. COS.

From the bottom panels in Figure 12 one notable observation can be spotted. In the deep ITM regions, Gamma can be seen to slightly diverge from the benchmark comparison. In Figures 15 and 16, an even broader log-moneyness domain, spanning from -3 to 3, makes it possible to investigate if the extrapolative capacity of DDN_PdivK remains as values extend further from the training bounds whilst presenting the DDN_IV model's OOD and non-explicit training target performance.

Figure 15 shows that DDN_PdivK starts to diverge as $m < -2$. This makes it evident that, in the extended OOD evaluation, the extrapolative capabilities previously alluded to are not sustained. However, as shown in Figure 16, the performance of DDN_IV facing the same conditions exhibit great stability and generalization in OOD domains. Furthermore, the theoretical bounds of delta and gamma are maintained over the entire domain. Note that the accuracy of Gamma and Delta are best in the stressed market regime, the underlying reasons for this will be explored in Section 5.3.1.

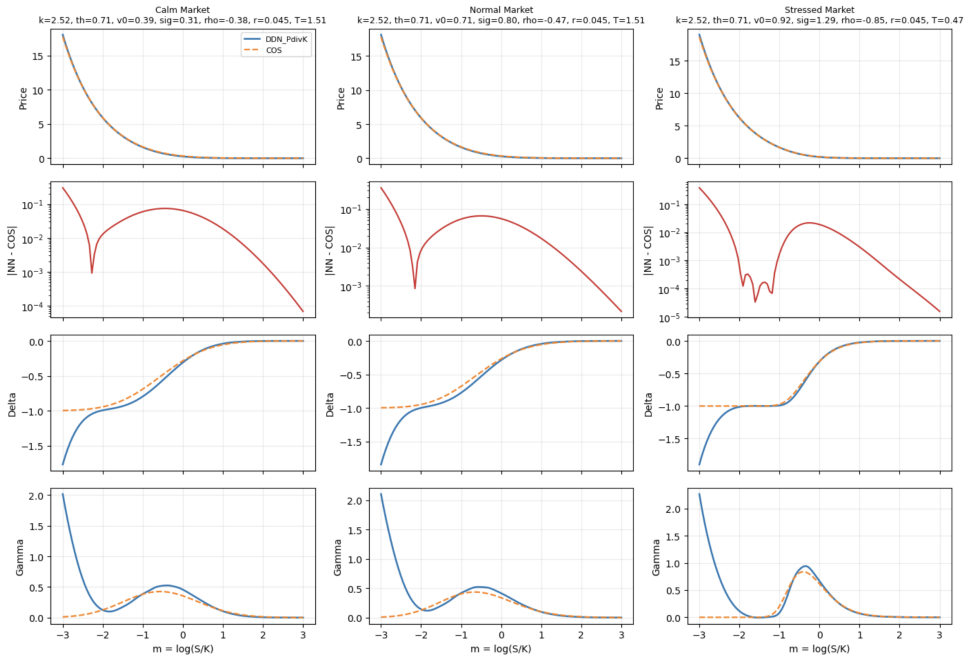



Figure 15: Extended extrapolative capacity evaluation for DDN_PdivK

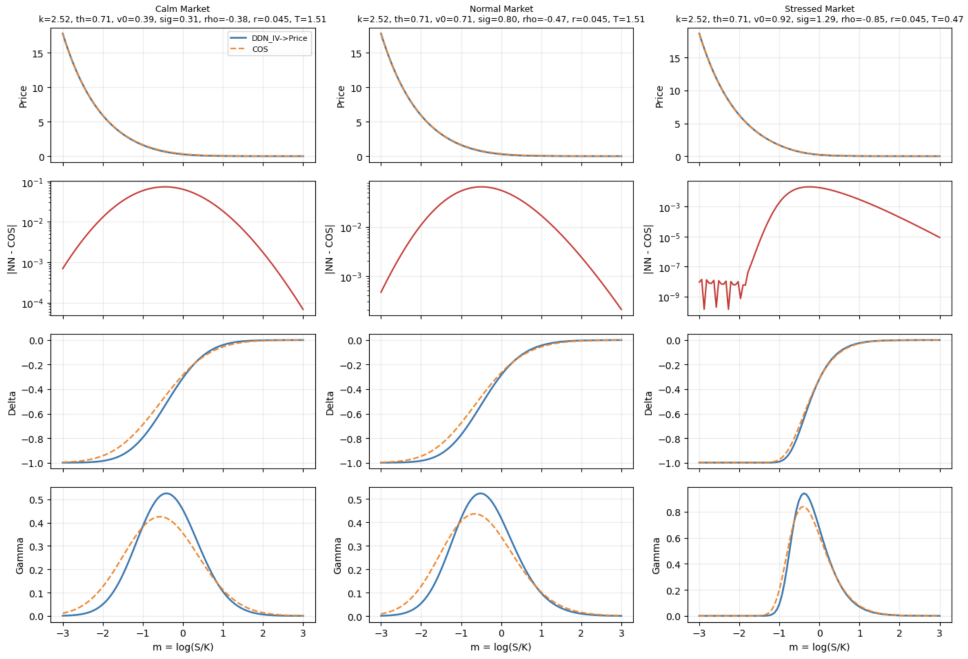



Figure 16: Extended extrapolative capacity evaluation for DDN_IV

### 4.3.3 Lipschitz Analytics

Following the methodology established in Section 3.4.5, Tables 11 and 12 present the ratio of the models' empirical Lipschitz constants to those of the COS benchmark. Because the absolute magnitude of a Lipschitz constant depends on the natural curvature and financial units of the specific parameter space, a raw value provides little intuition regarding overall stability. By evaluating the ratio of the neural networks' Lipschitz constants against the semi-analytical baseline, we normalize these domain-specific scales. This results in a dimensionless, relative metric that directly shows whether the surrogate models

produce smoother gradients. A value below 1.0 indicates that the surrogate's predicted Greeks exhibit flatter slopes than the semi-analytical baseline. This implies either a helpful reduction in numerical noise or an excessive smoothing of the true, sharp details of the pricing model.

For the price Greeks, the surrogates exhibit stability that is largely comparable to the COS benchmark, with both the median and 99th percentile ratios being near 1.0. Notably, the 99th percentile ratios are consistently lower than the median ratios, demonstrating a smoother learned shape of the pricing manifold in the regions where the COS method exhibits steeper parameter gradients. The most prominent exception is the mean-reversion speed, $\kappa$, where the neural networks produce significantly smoother gradients, primarily in the tail ends, than the baseline overall.

Table 11: Pairwise empirical Lipschitz ratios on price Greeks relative to COS ($L_2$ norm, 19 994 sampled pairs per row).

| **Greek** | **Model** | $\frac{L_{p99}}{L_{p99}^{\mathrm{COS}}}$ | $\frac{L_{\mathrm{med}}}{L_{\mathrm{med}}^{\mathrm{COS}}}$ |
|---|---|---|---|
| $\theta$ | DDN_P | 0.9897 | **0.9962** |
| | DDN_PdivK | 0.9890 | 0.9973 |
| | DDN_IV | **0.9888** | 0.9979 |
| | MLP_P | 1.0200 | 1.0040 |
| $\sigma$ | DDN_P | 0.9636 | 0.9855 |
| | DDN_PdivK | 0.9635 | **0.9819** |
| | DDN_IV | **0.9632** | 0.9831 |
| | MLP_P | 0.9644 | 0.9828 |
| $\rho$ | DDN_P | 1.0010 | 1.0010 |
| | DDN_PdivK | 1.0030 | 1.0010 |
| | DDN_IV | 1.0030 | **1.0000** |
| | MLP_P | **0.9900** | 1.0050 |
| $\kappa$ | DDN_P | 0.7619 | 0.9789 |
| | DDN_PdivK | 0.7607 | 0.9765 |
| | DDN_IV | **0.7600** | **0.9739** |
| | MLP_P | 0.7610 | 0.9804 |
| $v_0$ | DDN_P | 0.9863 | 0.9995 |
| | DDN_PdivK | 0.9877 | 0.9991 |
| | DDN_IV | **0.9830** | **0.9968** |
| | MLP_P | 0.9986 | 1.0050 |

When evaluating the IV Greeks, DDN_IV exhibits considerably more smooth sensitivities than COS in the 99th percentile, with ratios as low as 0.41. The median sensitivity ratios are highly consistent, all approximately equal to 1.0.

Table 12: Pairwise empirical Lipschitz ratios on IV Greeks relative to COS ($L_2$ norm, 19 994 sampled pairs per row).

| IV Greek | Model | $\frac{L_{p99}}{L_{p99}^{\mathrm{COS}}}$ | $\frac{L_{\mathrm{med}}}{L_{\mathrm{med}}^{\mathrm{COS}}}$ |
|---|---|---|---|
| $\theta$ | DDN_IV | 0.7586 | 0.9948 |
| $\sigma$ | DDN_IV | 0.5630 | 0.9927 |
| $\rho$ | DDN_IV | 0.8918 | 1.0280 |
| $\kappa$ | DDN_IV | 0.4136 | 0.9622 |
| $v_0$ | DDN_IV | 0.7034 | 0.9948 |

## 4.4 Arbitrage Diagnostics

As established in Section 2.1.5, theory requires $\Delta_{\mathrm{put}} \in [-1, 0]$. Table 13 reports the fraction of predictions from the test-set that violate this constraint, together with the respective observed extremes.

Table 13: Put Delta no-arbitrage check on the test set ($N = 4\,989$).

| Model | $\Delta_{\min}$ | $\Delta_{\max}$ | Violations | Violation % |
|---|---|---|---|---|
| DDN_P | $-1.0122$ | $-3.6590 \times 10^{-9}$ | 97 | 1.94 |
| DDN_PdivK | $-1.0059$ | $-1.0704 \times 10^{-9}$ | 108 | 2.16 |
| MLP_P | $-1.0243$ | $-1.2886 \times 10^{-3}$ | 114 | 2.29 |
| DDN_IV[†] | $-1.0000$ | $-2.2984 \times 10^{-27}$ | 4 | 0.08 |

[†] Although the bounds appear satisfied, the true $\Delta_{\min}$ for DDN_IV was $-1 - 4 \times 10^{-16}$. This precision-level difference clarifies why 4 violations occurs.

All three price models produce a small fraction of out-of-bound predictions. For DDN_P and DDN_PdivK, the magnitude of these violations is negligible, as the models show maximum deviations from the theoretical lower bound of just 0.0122 and 0.0059, respectively. MLP_P shows a slightly larger maximum deviation of 0.0243. DDN_IV produces next to no violations, occurring only at machine-precision level.

Table 14 shows lower- and upper-bound violations for all models, including the COS benchmark model. Lower bound checks enforce $P_{\mathrm{put}} \geq \max(Ke^{-rT} - S, 0)$ for absolute prices, while upper bound checks enforce $P_{\mathrm{put}} \leq Ke^{-rT}$, as established in Section 2.1.6.

Table 14: No-arbitrage price bound violations on the test set ($N = 4\,989$).

| Output | Model | Lower-bound viol. | Upper-bound viol. | Total breach % |
|---|---|---|---|---|
| Price (absolute) | COS | 59 | 0 | 1.18 |
| | DDN_P | 123 | 0 | 2.47 |
| | DDN_PdivK | 110 | 0 | 2.20 |
| | MLP_P | 138 | 0 | 2.77 |
| IV → price | COS | 0 | 0 | 0.00 |
| | DDN_IV | 0 | 0 | 0.00 |

The absolute price models show similar no-arbitrage violation rates. COS itself produces 59 lower-bound violations (1.18%), confirming that the semi-analytical benchmark is not entirely free from arbitrage. DDN_P is comparable at 2.47% breaches. DDN_PdivK is slightly lower at 2.20%. MLP_P shows the highest violation rate at 2.77%, suggesting that, without differential training, the model learns the general price shape less reliably.

For the IV → price models, both COS and DDN_IV produce zero violations across both bounds. This is expected, since both models derive prices by passing predicted IVs through the Black–Scholes formula, which is already arbitrage-free by construction.

Table 15 shows the percentage of valid implied volatility predictions for the two models with IV output. A valid IV is defined as finite and we investigated large IVs by identifying IVs outside the range $(0, 5]$.

Table 15: IV surface validity on the test set ($N = 4\,989$).

| Model | Valid IV | Invalid ($\leq 0$ or NaN) | $\lvert IV\rvert > 5$ |
|---|---|---|---|
| DDN_IV | 4 989/4 989 (100%) | 0 | 0 |
| COS | 4 928/4 989 (98.78%) | 61 | 0 |

DDN_IV produces valid IVs for every sample. COS fails to converge on 61 samples (1.22%), yielding non-finite or out-of-range IVs, likely coming from numerical inversion failures for extreme parameter combinations.

Table 16 displays static arbitrage violation rates evaluated on full strike-maturity surfaces. This enables shape checks (calendar, monotonicity, and butterfly) to be evaluated simultaneously within each surface.

Static no-arbitrage diagnostics for the models and COS were evaluated on 30 repeats × 400 surfaces ($\approx 240\,000$ quotes). The IV total-variance check applies only to DDN_IV and COS.

Table 16: Static no-arbitrage diagnostics on grouped Heston surfaces (mean violation rate ± one standard deviation).

| **Model** | **Bound (%)** | **Calendar (%)** | **Monotonic. (%)** | **Butterfly (%)** | **IV-TV (%)** |
|---|---|---|---|---|---|
| DDN_IV | **0.000** | $\mathbf{0.31 \pm 0.084}$ | **0.000** | $\mathbf{0.00035 \pm 0.0013}$ | $\mathbf{0.067 \pm 0.034}$ |
| DDN_P | $3.62 \pm 0.10$ | $1.92 \pm 0.15$ | $0.0001 \pm 0.0004$ | $2.71 \pm 0.08$ | — |
| DDN_PdivK | $2.55 \pm 0.09$ | $2.27 \pm 0.17$ | $0.002 \pm 0.002$ | $2.28 \pm 0.08$ | — |
| COS | $2.18 \pm 0.11$ | $1.21 \pm 0.12$ | $0.63 \pm 0.03$ | $1.98 \pm 0.13$ | $0.74 \pm 0.05$ |
| MLP_P | $3.69 \pm 0.14$ | $2.83 \pm 0.16$ | 0.000 | $3.12 \pm 0.09$ | — |

From Table 16, DDN_IV performs the best across all checks, with close to zero violations across any of the checks. DDN_P and DDN_PdivK perform similarly, both showing moderate violation rates of around 2-4% for bounds, calendar, and butterfly. COS is not entirely free from arbitrage either, its monotonicity violations (0.63%) stand out compared to the DDN models, which essentially produce zero monotonicity violations.

MLP_P, on the other hand, shows the highest violation rates among all models for bounds, calendar, and butterfly (3.69%, 2.83%, and 3.12% respectively). However, it shows zero monotonicity violations. Because the check only verifies that higher strikes produce lower prices, which is a simple directional relationship, it implies that MLP_P is able to capture this relationship consistently, even though it violates the other arbitrage conditions at higher rates than any other model.

Looking at the IV total-variance check, DDN_IV performs significantly better than COS, with a much lower violation rate (0.067% compared to 0.74%). This indicates that the network learns a smoother implied volatility surface. This is particularly evident in the IV surfaces in Figure 10, where COS shows major instabilities at short maturities.

# 5 Discussion

## 5.1 Computational Efficiency

The results in Table 7 show that the neural network models have a substantial time advantage compared to the COS benchmark for pricing. This is expected since the COS method requires numerical evaluation of the semi-analytical pricing formula for every new combination of Heston parameters and market states. Neural networks, however, only require a forward pass through the network (after training). As discussed in Section 2.7.8, this can be made very efficient through parallelization, especially on GPUs. Part of this advantage is therefore also hardware-dependent.

At the same time, it is important to distinguish between training time and inference time. For neural networks to function as reliable approximators, they first need to be trained. This is often a relatively time consuming task, and the time advantage compared to COS is therefore, in practice, only realized when the trained model is used repeatedly. In that case, the computational cost of training can be quickly amortized over a large number of pricing operations. In other words, network-based option pricing is only really relevant in cases where many evaluations need to be performed within the same trained parameter range. This is exactly the case when it comes to calibrating the Heston model, which can justify the use of surrogate models for this purpose.

However, since we have not explicitly conducted a full end-to-end calibration in this study, the results

cannot directly demonstrate how much faster an actual calibration process might be when using these surrogates. What the results do show is that the central component in the calibration process, being the pricing operation, can be evaluated orders of magnitude faster than the COS method.

## 5.2 Accuracy

### 5.2.1 MSE and MAE

The MLP_P model, which was exclusively trained on price targets without derivative information, performed overall better than the COS benchmark when it comes to MSE but worse when looking at MAE. This is explained by the dumbbell plot (Figure 8), which illustrates the big spread between the COS benchmark's accuracy, showing how much the worst cases (the extreme outliers) affected its performance. Even if the extreme outliers were few for the COS benchmark and it showed an overall lower MAE than the MLP_P model, the absolute magnitude of these errors being squared results in the overall MSE being worse than for the MLP_P model. The COS method would be expected to perform better and experience fewer outliers if the truncation domain was wider (i.e., using a higher truncation constant (L)) and the number of expansion terms (N) were higher than the 512 used for evaluation. However, this would decrease the efficiency of the COS method, which, as shown in Section 4.1, is already substantially slower than the neural networks in the tested setup.

Although MLP_P performed better than the COS benchmark when looking at MSE on most targets, it consistently experienced the worst overall accuracy of the neural network-based models. Its performance validates that standard supervised training performs worse when mapping the complex, high dimensional dynamics of stochastic volatility models. Incorporating the parameter Greeks into the loss function, like in the DDN models, effectively utilizes Sobolev training to make sure that the approximation of the function and its gradients are mathematically constrained. This differential data prevents overfitting and decreases the MSE and MAE by forcing the network to learn the true shape of the pricing function rather than just point-wise estimates. The models trained on price exhibited errors that scaled proportionally with the absolute magnitude of the option price.

One of the most significant findings from the error metrics is the overall relatively strong performance of the DDN_IV model. Despite being trained to predict IV it outperformed the models trained directly on price and price Greeks both on MSE and MAE. This aligns with the theory discussed in Section 3.3.1, stating that IV-based models are better at capturing the underlying volatility level and structure of the surface compared to models trained on raw prices. Because the IV target avoids the extreme variance in magnitude between near-worthless OTM options and highly valuable deep ITM options, the neural network can specify prices for all options across the entire surface by passing the predicted IVs through the Black–Scholes formula, which resulted in prices that exhibited exceptional accuracy.

During calibration, gradient-based optimizers (like Levenberg-Marquardt or L-BFGS) use derivatives to understand exactly how adjusting each model parameter will affect the predicted price or implied volatility. Conversely, highly accurate Greeks provide a reliable gradient path which makes the models with low MSE Greeks favorable for calibration, since this ensures the optimizer makes precise parameter adjustments accelerating convergence and reducing the number of iterations required to successfully calibrate the model.

### 5.2.2 MSE Heatmaps

The IV conversion issue was raised in the section on IV surfaces (Section 4.2.3). Looking at the results of Figure 11 it is evident that the IV conversion showed instability in the ITM and OTM region as maturity approaches one week. In turn, this means that the MSE presented in the DDN_IV heatmap could be misleading as the ground truth dataset is unreliable in these regions. It can therefore be argued that the true MSE of DDN_IV is lower and more stable at short maturities. Extending this argument yields that the MSE of this model is even more homogeneous than what the heatmap indicated and hence significantly outperformed the other models in terms of the uniformity of the MSE distribution over the entire domain.

As discussed in Section 2.4.3, most gradient-based optimization algorithms are implemented as a minimization of squared residuals. This means that each option contributes to the parameter update in proportion to its squared error. Hence, the larger residuals dominate the gradient even if the smaller residuals are equally as important to fit. By extension, this means that for price-based surrogates, whose MSE scales with the magnitude of the underlying price, the optimizer's loss function may become dominated by the larger residuals of ITM options. If the MSE in the OTM region is low compared to ITM, the model will prioritize updates that produce better fits of ITM options while ignoring the OTM options during calibration. It can therefore be argued that uniformity of the MSE distribution over the pricing domain is of high importance for the model to capture the embedded parameter information of all market options.

As DDN_IV is a much less biased predictor in this respect compared to the other neural network surrogates, by extension, it also makes for a less biased optimization during calibration. The COS benchmark, on the other hand, was often highly unreliable in the short maturity region and could therefore mislead the optimization algorithm. In terms of price, DDN_IV was also more balanced than the other networks showing low MSE for both OTM and ITM regions. While it had a larger MSE in the ATM region, this error difference generally only spans around one to two orders of magnitude, whereas the bias of the other networks span over three orders of magnitude. While complex weighting schemes can be implemented to offset these effects, DDN_IV intrinsically solves this issue and may therefore be more robust. Combining these results we argue that the bias of the price networks will impact the optimizer more than that of DDN_IV. Additionally, as shown in Section 4.2.2 the inversion from IVs to price effectively forces the predictions closer to the true prices as we move further ITM or OTM.

Furthermore, as presented in Section 4.2.2, some parameter combinations can yield large tails and therefore large errors in the COS model. These regions would also dominate the optimizer's loss, especially as these errors span upwards of six orders of magnitude. Hence, the parameter updates might only focus on these regions, effectively breaking down the optimization performance. Note that this might be less of an issue with increased expansion terms.

### 5.2.3 IV Surfaces

As we have noted before, very short maturities are not necessarily the domain where the Heston model is used in practice as it does not have the capacity to model jumps. However, the IV surfaces showed that the COS model and more specifically the IV inversion had already started to break down as we approached two weeks, while DDN_IV showed great stability even as time to maturity approached 0. This shows that the domain where this model could be applied in calibration surpasses traditional models relying on inversion. Additionally, the extrapolative capacity of DDN_IV showed that the network has actually learned the shape of the underlying volatility structure of the Heston model.

The convexity of the DDN_IV surfaces also suggests no-arbitrage predictions, while the COS benchmark, especially at very short maturities exhibited a highly non-convex structure implying that butterfly conditions could have been breached. This is supported by the arbitrage results, where DDN_IV showed almost no butterfly arbitrage, while COS had a significantly higher ratio.

Regarding the implications for calibration, these results further support the usage of a DDN surrogate as it would accurately predict the Heston IVs for arbitrarily small maturities. When adherence to the underlying model is not an issue, the main problem rather lies in the structural limitations of Heston in modeling the sharp nature of short-maturity IVs. Furthermore, the divergence of the COS price inversion in these domains is not noise, but rather structured, meaning that the optimizer would get a clear signal to move in the wrong direction. In other words the COS IV results are not just inaccurate in these domains but are actively misleading to the optimizer. The stability of DDN_IV in the extrapolated domains, i.e., OOD in terms of log-moneyness, makes for more reliable predictions during extreme events. This effectively circumvents the need for retraining the network or applying other models in these regimes. Neural networks are often thought of as black-boxed, but these indications of a learned structure, consistent with the underlying model, and extrapolative capacities provide concrete grounds for trusting the model's predictions.

## 5.3 Greeks

### 5.3.1 Greek Stability

The results presented in Figures 12, 13 and 14 exposed a vulnerability in the capacity of DDN_P and MLP_P to extrapolate for out-of-distribution (OOD) values. While all three evaluated models displayed strong interpolation capabilities and accurate price fits within the training bounds, we observed severe divergence in the Gamma predictions for DDN_P and MLP_P. This observation emphasizes that achieving a low absolute pricing error does not guarantee that the network has learned a smooth, well-behaved pricing manifold.

Conversely, the extrapolative stability from DDN_PdivK on the smaller extended domain is likely caused by the target normalization. As established in Section 3.3.1, by scaling the training labels by the strike price (P/K), the network was provided with a structural hint that naturally bounds the absolute price scale. As a result, this transformation appears to restrain the Gamma to an extent, preventing the severe local divergence observed in the other models. This can be seen in the stressed market scenario of Figure 12, where Gamma for DDN_PdivK flattened safely at zero instead of plunging into negative territory, although it started to diverge in the tail. The same argument could explain the smooth decrease that DDN_PdivK exhibited in the OOD ITM absolute pricing error, compared to a slight increase observed in the other two models.

The extraploative capabilities on the larger extended log-moneyness domain of DDN_PdivK and DDN_IV on non-target Greeks were also investigated. Not only did DDN_PdivK start to diverge as $m < -2$, it also produced breaches of the true Delta bounds and similarly, it heavily violated the true shape of Gamma. The combination of the two Figures 12 and 15 suggest a greater capacity for extrapolation compared to the non-scaled price-based models, albeit limited to small extensions OOD. DDN_IV performed very well and did not violate the true bounds of the theoretical Delta and Gamma sensitivities. Notably it performed the best in terms of accuracy in the stressed market regime. One possible explanation for this is that these high volatility regimes offer stronger signal to the network as the volatility surface is steeper. In calmer markets, the surface is flatter and provides less structural information for the network to learn from, which may hinder the model's ability to capture the underlying dynamics in these regimes.

With the effects of target normalization established, evaluating the two raw-price models (DDN_P and MLP_P) side-by-side visually demonstrated a clear advantage of differential training. The MLP_P model, which was trained solely on price targets without explicit parameter Greek information, exhibited the most severe divergence from the baseline standard in OOD ITM regions. Although DDN_P also struggled with OOD extrapolation, its divergence was visibly less extreme. Because MLP_P was not trained on parameter or state Greeks, it relies entirely on the raw price surface to infer sensitivities. This improvement can be attributed to the fact that parameter derivative labels are incorporated into its loss function. As established in Section 2.8.1, this acts as a powerful shape regularizer that enforces a smoother mapping. While this geometric constraint is applied with respect to the Heston parameters, it translates into a more stable state space as well, dampening OOD divergence.

For gradient-based calibration to efficiently converge, the optimizer requires a stable gradient with respect to the Heston parameters ($\kappa$, $\theta$, $\sigma$, $\rho$, $v_0$) (Section 2.4.3). Although our visual diagnostics focused on state Greeks ($\Delta$ and $\Gamma$), the stability of these non-explicit targets is a powerful indicator of the network's overall ability to generalize. A model that maintains structural integrity (e.g. non-negative Gamma) in unseen regimes demonstrates a smooth learned manifold, implying its parameter Greeks will be equally reliable. The substantial OOD divergence seen in DDN_P, MLP_P, and DDN_PdivK as we expand the domain indicates that their parameter gradients would likely mislead an optimizer in extreme market states. DDN_IV on the other hand showed stable extrapolation. The performance on Delta and Gamma suggest smoothness of the learned IV surface and stability of the network's approximations. Additionally, the fact that the model produced bounded Delta and non-negative Gamma over the OOD domain is consistent with the findings of the arbitrage analysis, where DDN_IV exhibited the lowest number of arbitrage breaches. These are desirable properties for calibration surrogates that support the suitability of DDN_IV for calibration purposes.

### 5.3.2 Lipschitz Analytics

The empirical Lipschitz ratios presented in Tables 11 and 12 highlight two distinct benefits of the differential neural network architecture. Across the majority of the Heston parameters ($\theta$, $\sigma$, $\rho$, $v_0$), the median Lipschitz ratios lie close to one 1.0. This indicates that the networks match the benchmark's gradient magnitudes, suggesting consistent local smoothness across the pricing manifold. However, the 99th percentile ratios consistently fall below those of the median, in almost all cases being below 1.0. This shows that while the surrogates match the baseline in standard market regimes, they dampen the large gradient spikes produced by the COS benchmark in the tail of the distribution. This behavior is consistent with the well-known spectral bias (Section 2.7.2) of neural networks. While this indicates that the surrogates likely over-smooth true, sharp localized features of the Heston surface, it can also prevent the networks from replicating high-frequency numerical artifacts that occasionally arise from discrete Fourier inversion, preserving bulk accuracy.

The most prominent deviation in the price domain was observed in the mean-reversion speed, $\kappa$, where all neural network models produced significantly smoother gradients than COS in the 99th percentile (ratios $\approx$0.76). This may be due to the fact that, as established in Section 2.2.3, $\kappa$ directly interacts with the volatility of variance ($\sigma$) to determine whether the variance process hits zero. Near this boundary, Fourier-based pricing methods like COS can be observed to suffer from highly oscillatory integrands and severe truncation errors, leading to volatile partial derivatives. The continuous activation functions of the neural networks interpolate smoothly across this boundary, trading exact local replication for numerical stability, resulting in bounded sensitivities where the COS method's gradients become volatile.

The better performance of the DDN_IV model in the tail ends highlights a significant advantage of mapping directly to the implied volatility domain. As shown in Table 12, the COS benchmark's tail instability

was magnified in the IV Greeks, whereas the DDN_IV model achieves ratios as low as 0.41. As explained in Section 2.5.4, computing IV sensitivities requires dividing the price Greek by the Black–Scholes Vega. For options where Vega $\to 0$, this division creates a mathematical singularity, causing the resulting Greek to explode. Because the DDN_IV network learns the IV surface directly, it can approximate smoother gradients, avoiding the singularity that may present itself in semi-analytical models. Because the COS benchmark's Lipschitz constant is artificially inflated by these low-Vega singularities, the much lower ratios achieved by DDN_IV (e.g., 0.41 for $\kappa$) do not purely indicate over–smoothing, but rather a successful bypass of a numerical explosion.

Ultimately, the increased stability of these differential surrogates carries noteworthy implications for the two-step calibration approach introduced in Section 2.4.4. Gradient-based calibration relies on parameter Greeks to provide information on both the magnitude and direction of parameter adjustments, which is essential for accelerating convergence. While the COS method overall provides accurate directional signals, it empirically produces volatile or exploding gradients near the Feller boundary or in low-Vega regimes. Because standard optimization algorithms heavily penalize exploding gradients, these numerical artifacts render the calibration process highly unstable. By supplying smoothed, bounded sensitivities, even if they slightly under-represent the true steepness in extreme regimes, the DDN architectures ensure reliable, stable gradient updates. For the DDN_IV model in particular, these stable gradients allow the optimizer to fully leverage the numerical advantages of calibrating in the IV domain (Section 2.4.2), further reducing the ratio between Hessian entries.

Beyond improved stability, the DDNs also delivered superior gradient directionality. As seen in Table 10, the inclusion of differential training allowed these models to consistently outperform the standard MLP baseline in directional accuracy. In addition, the DDN_IV's directional accuracy exceeded 99% across all Heston Greeks, surpassing even the COS benchmark. Therefore, utilizing these differential surrogates effectively replaces the mathematically thorough but numerically unstable tail sensitivities of the COS method with smooth, reliable gradients, enabling fast, robust, and accurate gradient-based calibration.

As a final point of discussion, it is important to keep in mind that the approximated Lipschitz constants are guaranteed lower bounds of the true empirical Lipschitz constants, which in turn are guaranteed lower bounds of the true global Lipschitz constant for each model and its respective Greek. This implies that, if calculating the global Lipschitz constant was possible for our study, the ratios presented in the result may have been different.

## 5.4 Arbitrage Diagnostics

DDN_IV achieved near-zero violation rates across all no-arbitrage checks, which is based on two sources. First, prices are recovered by passing predicted IVs through the Black–Scholes formula, which is free from arbitrage by construction. This structural property is consistent with the observed zero bound and monotonicity violation rates. Second, the calendar and IV total-variance checks are not covered by this guarantee. These check whether prices across different maturities are mutually consistent, something the Black–Scholes formula does not automatically guarantee since it is applied independently at each point. DDN_IV exhibited lower violation rates of 0.31% and 0.067% respectively, compared to 1.21% and 0.74% for COS. These results are further supported by the smooth IV surfaces in Section 4.2. A network, in this case DDN_IV, that has learned a coherent volatility structure across different maturities tends to produce a total implied variance that increases with $\tau$, which is consistent with the calendar and IV total-variance checks.

In contrast to DDN_IV's near-perfect compliance with the arbitrage checks, COS produces a 0.63% monotonicity violation rate. However, this is not a property of the Heston model, which is arbitrage-free

by construction. Since the DDN models approximate the same underlying Heston prices, yet produce essentially zero monotonicity violations, the failures are specific to the numerical method rather than the model itself. The Fourier inversion that builds up the COS method can produce small local pricing errors for certain parameter combinations. These instabilities may break the expected decreasing relationship between price and strike. In contrast to the COS method, the DDN learns a smooth global approximation of the pricing surface. This tends to reduce local oscillations that can arise in numerical methods, and promotes consistency across strikes. This is also consistent with the near-zero monotonicity violations observed for the surrogate models.

Moreover, the differences between the DDN models and MLP_P are moderate in absolute terms. MLP_P performed worse by roughly one percentage point across almost all no-arbitrage checks, but the pattern itself is informative. All models, including MLP_P, reached near-zero monotonicity violations, which is expected since this is a simple directional relationship that only requires higher strikes to produce lower prices. The butterfly and calendar conditions are more demanding and require local convexity across strikes and consistency across maturities respectively. MLP_P's higher violation rates on these conditions suggest that differential training improves the local precision and global coherence of the learned surface.

As seen in the Greek stability results (Section 4.3.2), DDN_P, DDN_PdivK and MLP_P produced negative Gamma values in OOD regions. As explained in Section 2.1.5, Gamma measures the second-order sensitivity to the underlying price, representing the convexity of the option, and should be strictly non-negative. A negative Gamma implies local concavity, which directly violates the butterfly spread no-arbitrage constraint (Equation 26). This connection further reinforces the slightly higher butterfly violations observed in Table 16 for DDN_P and MLP_P. This suggests that the models fail to fully learn a globally convex pricing surface.

Overall, DDN_IV performed best across the arbitrage checks. As discussed above, this is partly a structural feature of the model. Recovering prices through the Black–Scholes formula enforces certain properties by construction. But the low calendar and IV total-variance violation rates show that the model has also learned a smooth IV-surface that is coherent across maturities, which was not guaranteed by the architecture alone. Looking from the perspective of eventually calibrating, arbitrage violations are undesirable. They mean the model assigns inconsistent prices to options. When an optimizer or similar uses such a model to search for the best prices, these inconsistencies may lead to biased or unstable parameter estimates. DDN_IV's near-zero violation rates therefore make it the most reliable among the tested models for eventual gradient-based calibration.

# 6 Conclusions

This study investigated neural network surrogates as alternatives to the COS method for option pricing, which is the main element of the calibration pipeline. Four architectures were compared: a baseline MLP trained on prices (MLP_P), two deep differential networks trained on prices and prices normalized by strikes (DDN_P and DDN_PdivK), respectively, and a deep differential network trained on implied volatilities (DDN_IV). These were compared against a COS benchmark with $N = 512$ expansion terms. The results support three main conclusions: (i) the differential neural network surrogates can match or exceed the accuracy of the COS benchmark at some error metrics while computing outputs orders of magnitude faster, (ii) the choice of learning targets matters substantially for the performance and stability of the networks, and (iii) that some architectures can outperform the COS benchmark in terms of gradient stability and arbitrage compliance.

Regarding computational efficiency, all networks evaluated Heston prices substantially faster than the

COS benchmark, and this gap widens as hardware capabilities are fully utilized. This advantage is realized once training costs are amortized over many evaluations, which makes these surrogate models particularly well-suited for repeated pricing operations, such as the inner loop of the calibration pipeline.

The accuracy results identify that the learning target of the networks is a highly influential design choice. The price-only baseline (MLP_P) is outperformed by all differential neural networks, confirming that traditional neural network architectures struggle with the high-dimensional structure of stochastic volatility models. Incorporating parameter Greeks into the loss through Sobolev training substantially improved both MSE and MAE. This is because the network is constrained to additionally learn the shape of the pricing function rather than point-wise predictions only. Among the differential models, DDN_IV produced the lowest errors on both metrics despite only being trained to predict implied volatilities. Additionally, the network exhibited prediction improvements by orders of magnitude over the COS benchmark on IV MSE.

We have argued that the calibration performance is not only determined by the model's aggregate errors but also the uniformity of the error distribution across the pricing surface. Because gradient-based optimizers update parameters in proportion to squared residuals, surrogates whose errors scale with the magnitude of the option price will systematically prioritize ITM regions during calibration, effectively discounting the information content of OTM contracts. The COS benchmark exhibits a related weakness at short maturities, where structured failures of the IV conversion can actively mislead the optimizer. DDN_IV, by contrast, displayed a more homogeneous error distribution across the surface and remained stable in the regions where COS-based inversion broke down. This further indicates that DDN_IV is the most suitable of the evaluated models for use as a calibration surrogate, not only because its aggregate errors are lower, but also because the structure of the error is less biased.

The IV surface analysis offered further evidence for this view. DDN_IV produced convex surfaces consistent with no-arbitrage conditions and remained stable as time to maturity approached zero, while the COS-derived surfaces became non-convex. The fact that DDN_IV continued to produce well-behaved surfaces in regions of the input domain where its training data was itself unreliable, combined with stable OOD performance, suggests that the network has captured the underlying volatility structure of the Heston model rather than memorizing point-wise outputs. This extrapolative behavior is a partial response to the common critique of neural surrogates as black-boxed, as it provides empirical grounds for trusting the model's predictions in regimes outside the training distribution.

The Greek stability and Lipschitz analyzes provide the strongest evidence that DDN_IV has learned a structurally consistent pricing manifold rather than a high-accuracy point-wise fit. In the minor extension of domain to out-of-distribution market states, MLP_P and DDN_P both produced divergent and at times negative Gamma values, while DDN_PdivK remained well-behaved. However, when the OOD domain was extended further, DDN_PdivK also diverged while DDN_IV remained highly stable and converged to the COS benchmark Greek values.

The empirical Lipschitz ratios reinforce this picture. Across the Heston parameters, the differential networks tracked the COS benchmark's curvature closely at the median while also reducing its gradient spikes in the tails. DDN_IV extended this advantage further by avoiding the low-Vega division that destabilizes COS-derived IV Greeks, producing tail Lipschitz ratios as low as 0.41 and directional accuracy exceeding 99% across all parameter Greeks. For gradient-based calibration, where the optimizer depends on both the magnitude and the direction of parameter sensitivities, this combination of bounded curvature and high directional accuracy translates into reliable updates in precisely the regions where COS becomes unreliable.

The arbitrage diagnostics show that DDN_IV does not only exhibit stable gradients, but that it also has structural benefits in terms of pricing. Because prices are recovered by passing predicted IVs through

the Black–Scholes formula, the model satisfies bound and monotonicity conditions by construction. Additionally, DDN_IV's low calendar and IV total-variance violation rates, substantially below those of the COS benchmark, indicate that the network has learned a volatility structure that is consistent across maturities. This is not guaranteed by the architecture alone. The connection between the negative-Gamma behavior observed in MLP_P and DDN_P, and their elevated butterfly violation rates further illustrates that smoothness of the learned surface and arbitrage compliance are closely related. In this sense, they can be seen as two views of the same underlying property. Since arbitrage violations can cause an optimizer to chase inconsistent prices, DDN_IV's low violation rates also make it the most trustworthy model. These conclusions point to DDN_IV as the most suitable candidate for calibration to both IV and prices under the Heston model, offering several benefits compared to the semi-analytical alternatives, not only in terms of speed, but also stability.

# 7 Limitations and Further Research

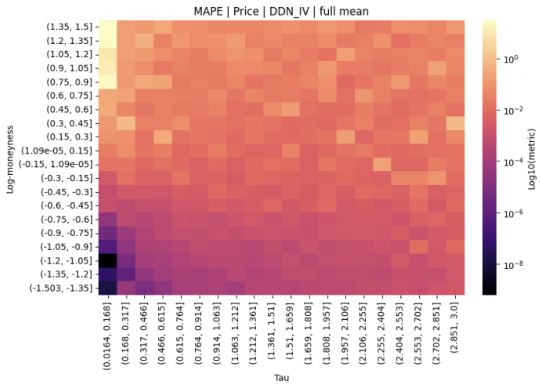
The most notable limitation of this study is that no end-to-end calibration was performed. The developed surrogate models were only evaluated by their ability to perform in relevant performance dimensions, but were never deployed within an actual calibration routine. The conclusions regarding calibration speed and accuracy are therefore implied from the surrogate evaluation metrics rather than being demonstrated empirically.

Future work could therefore implement a full calibration loop. This would be done to validate whether the accuracy and computational advantages of the DDN models found in this study translate to the calibration setting.

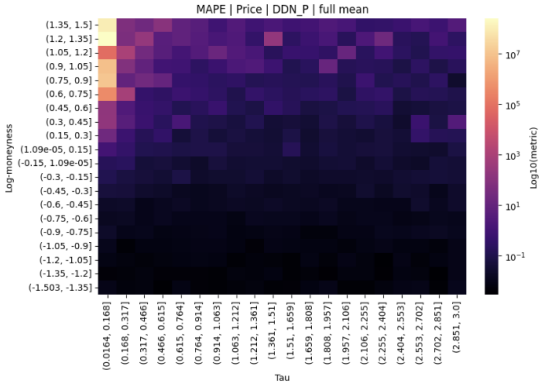
The final hyperparameters of the Differential Neural Networks were, as previously discussed, not implemented following a hyperparameter tuning protocol due to computational constraints. Further studies could implement a more thorough exploration of potential hyperparameter combinations to further optimize the results presented in this study.

Furthermore, it may appear as if the use of synthetic data rather than real market data creates a limitation. However, real market data would first need to be calibrated to the Heston model to extract the underlying parameters, which would likely serve as an additional source of error. Under the assumption that the underlying follows the dynamics of the Heston model, synthetic data generated directly from the model represent the truth more accurately.

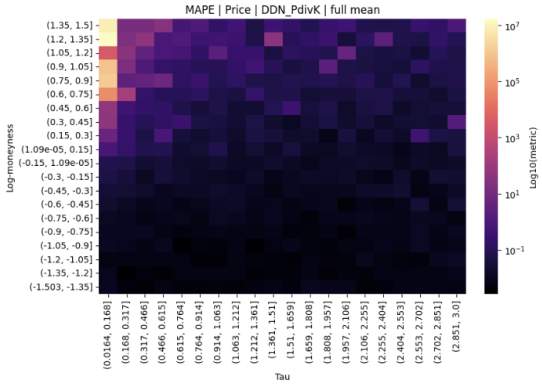
# A MAPE Heatmaps



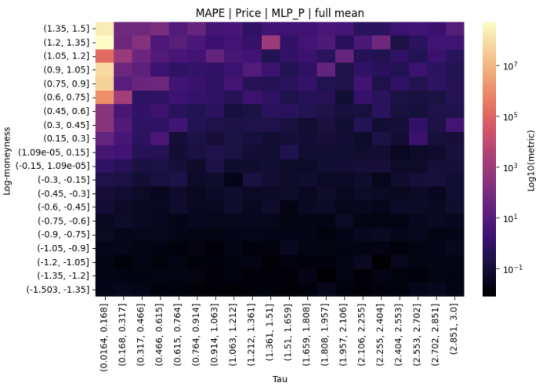

(a) DDN_IV (price).



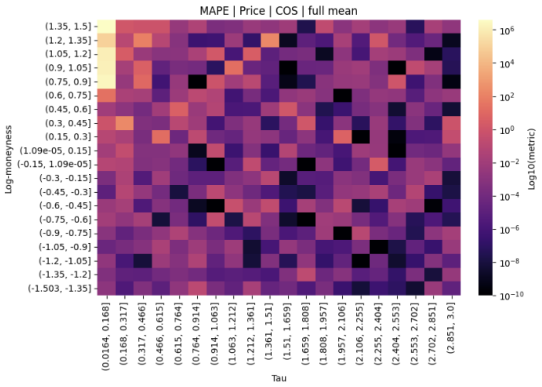

(b) DDN_P.



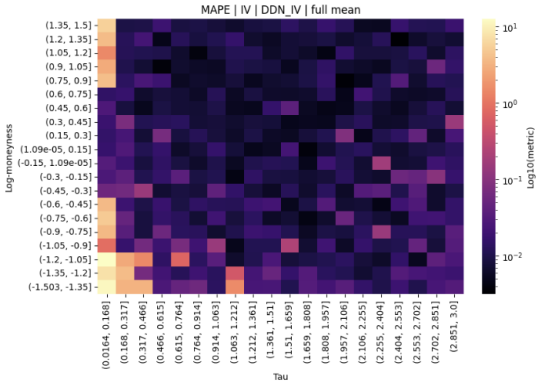

(c) DDN_PdivK.



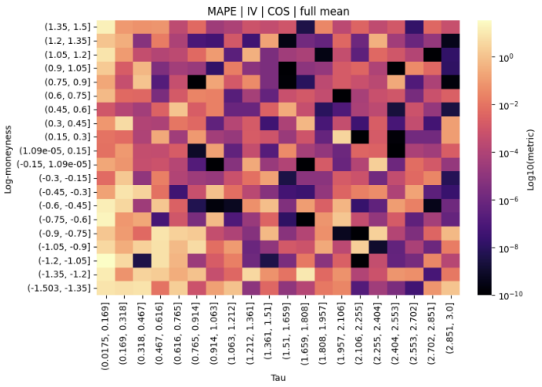

(d) MLP_P.




(e) COS (price).




(f) DDN_IV (IV).




(g) COS (IV).



Figure 17: MAPE heatmaps across log-moneyness and time to maturity.

# B Derivations of Network Delta and Gamma Formulas

As our neural network takes scaled log-moneyness $\tilde{m} \in [-1, 1]$ as its input, the gradients produced by automatic differentiation are taken with respect to $\tilde{m}$, not $m$. Even if the outputs for DDN_PdivK ($P/K$) and DDN_IV (IV) are not explicitly min-max scaled like the raw prices in DDN_P, the input is still scaled. Therefore, the chain rule must extract the input scaling factor $\alpha_m$, Equation 157, across all three architectures to transform the network's scaled gradients back to the true financial domain.

## B.1 The Mathematical Framework: Dimensionality Reduction and Scaling

Because European option prices exhibit linear homogeneity of degree one with respect to the spot price $S$ and strike price $K$, we can reduce the dimensionality of the pricing problem by setting $S = 1$. Hence, the network's state derivatives can be described across the log-moneyness domain $m$. Applying these formulas to unnormalized market data would require keeping S throughout the entire derivation of the state Greeks. Recall that the network learns over the log-moneyness domain $m = \ln(S/K)$. As described in Section 3.3.1, using the homogeneity of degree one of European option prices in (S,K), an unnormalized option price $P(S, K)$ can be expressed via a function $g(m)$:

$$P(S, K) = S \cdot g(m), \tag{151}$$

where $g(m) = P(1, e^{-m})$. To derive the standard Spot Greeks ($\Delta = \partial P/\partial S$ and $\Gamma = \partial^2 P/\partial S^2$), we apply the chain rule, recognizing that $\partial m/\partial S = 1/S$.

**1. Spot Delta.**

$$\Delta = \frac{\partial}{\partial S}[S \cdot g(m)] = g(m) + S \cdot g'(m) \cdot \frac{1}{S} = g(m) + g'(m). \tag{152}$$

Evaluated at $S = 1$, this yields the fundamental relation

$$\Delta = P + \frac{\partial P}{\partial m}. \tag{153}$$

**2. Spot Gamma.**

$$\Gamma = \frac{\partial \Delta}{\partial S} = \frac{\partial}{\partial S}[g(m) + g'(m)] = g'(m) \cdot \frac{1}{S} + g''(m) \cdot \frac{1}{S}. \tag{154}$$

Evaluated at $S = 1$, this yields

$$\Gamma = \frac{\partial P}{\partial m} + \frac{\partial^2 P}{\partial m^2}. \tag{155}$$

**The Input Scaling Factor $\alpha_m$.** Before feeding data to the networks, the inputs are min-max scaled to $[-1, 1]$ (Section 3.3.7). Let $\tilde{m}$ denote the scaled log-moneyness. The relationship between the raw and scaled coordinate is

$$\tilde{m} = 2 \cdot \frac{m - m_{\min}}{m_{\max} - m_{\min}} - 1. \tag{156}$$

Taking the derivative of the scaled input with respect to the raw input yields the constant scaling factor $\alpha_m$:

$$\alpha_m = \frac{\partial \tilde{m}}{\partial m} = \frac{2}{m_{\max} - m_{\min}}. \tag{157}$$

Therefore, any derivative extracted via Autograd with respect to $\tilde{m}$ must be multiplied by $\alpha_m$ to recover the derivative with respect to $m$:

$$\frac{\partial}{\partial m} = \alpha_m \frac{\partial}{\partial \tilde{m}}, \qquad \frac{\partial^2}{\partial m^2} = \alpha_m^2 \frac{\partial^2}{\partial \tilde{m}^2}. \tag{158}$$

We now apply this framework to transform the outputs of the three neural network architectures back to the financial domain.

## B.2 Model 1: DDN_P and MLP_P (Direct Price)

In these architectures, the network predicts a min-max scaled price $\tilde{P} \in [0, 1]$ (Section 3.3.7). To obtain the raw price from the scaled price the inverse of the scaling function is applied $P = \tilde{P}(P_{\max} - P_{\min}) + P_{\min}$. Let $\beta_P = \frac{\partial P}{\partial \tilde{P}} = P_{\max} - P_{\min}$ denote the price scaling factor.

**1. Unscaling the Autograd Derivatives.** Using automatic differentiation, we extract the derivatives of the scaled price with respect to the scaled log-moneyness, $\partial \tilde{P}/\partial \tilde{m}$ and $\partial^2 \tilde{P}/\partial \tilde{m}^2$. Applying the chain rule to unscale both the input and the output yields

$$\frac{\partial P}{\partial m} = \beta_P \cdot \alpha_m \cdot \frac{\partial \tilde{P}}{\partial \tilde{m}}, \tag{159}$$

$$\frac{\partial^2 P}{\partial m^2} = \beta_P \cdot \alpha_m^2 \cdot \frac{\partial^2 \tilde{P}}{\partial \tilde{m}^2}. \tag{160}$$

**2. Final Spot Greeks.** Substituting these into the established framework in B.1 at $S = 1$:

$$\Delta = P + \left( \beta_P \, \alpha_m \, \frac{\partial \tilde{P}}{\partial \tilde{m}} \right), \tag{161}$$

$$\Gamma = \left( \beta_P \, \alpha_m \, \frac{\partial \tilde{P}}{\partial \tilde{m}} \right) + \left( \beta_P \, \alpha_m^2 \, \frac{\partial^2 \tilde{P}}{\partial \tilde{m}^2} \right). \tag{162}$$

## B.3 Model 2: DDN_PdivK (Strike-Normalized Price)

Here, the network directly predicts the unscaled ratio $u = P/K$ using the scaled input $\tilde{m}$. Because we evaluate the Greeks at $S = 1$ and $m = \ln(S/K)$, it follows that $K = e^{-m}$. Therefore, the unnormalized price is $P(S, K) = K \cdot u(\tilde{m})$. Note that as we use $u$ instead of $g$, the mathematical framework established in Section B.1 does not apply.

**1. Spot Delta.** We take the partial derivative with respect to $S$, holding $K$ constant:

$$\Delta = \frac{\partial P}{\partial S} = \frac{\partial}{\partial S}[K \cdot u] = K \cdot \frac{\partial u}{\partial m} \cdot \frac{\partial m}{\partial S} = K \cdot \left( \alpha_m \frac{\partial u}{\partial \tilde{m}} \right) \cdot \frac{1}{S}. \tag{163}$$

Evaluating at $S = 1$ (where $K = e^{-m}$) yields

$$\Delta = e^{-m} \cdot \alpha_m \cdot \frac{\partial u}{\partial \tilde{m}}. \tag{164}$$

**2. Spot Gamma.** Differentiating Delta with respect to $S$:

$$\Gamma = \frac{\partial}{\partial S}\left[KS^{-1}\frac{\partial u}{\partial m}\right] = K\left[-S^{-2}\frac{\partial u}{\partial m} + S^{-1}\frac{\partial^2 u}{\partial m^2}\cdot\frac{1}{S}\right] = \frac{K}{S^2}\left[\frac{\partial^2 u}{\partial m^2} - \frac{\partial u}{\partial m}\right]. \tag{165}$$

Substituting the $\alpha_m$ chain rule expansions of Equation 158 and evaluating at $S = 1$ and $K = e^{-m}$:

$$\Gamma = e^{-m}\left[\alpha_m^2\frac{\partial^2 u}{\partial \tilde{m}^2} - \alpha_m\frac{\partial u}{\partial \tilde{m}}\right]. \tag{166}$$

## B.4 Model 3: DDN_IV (Implied Volatility)

This model predicts the unscaled implied volatility $\sigma$ using the scaled input $\tilde{m}$ and hence it provides $\partial\sigma/\partial\tilde{m}$ and $\partial^2\sigma/\partial\tilde{m}^2$ via Autograd. To obtain the Spot Greeks, we utilize the Black–Scholes pricing function.

By positive homogeneity of degree one in $(S, K)$, the BS price can be described as

$$V^{BS}(S, K, \sigma) = S \cdot P_{BS}(m, \sigma), \qquad m = \log(S/K), \tag{167}$$

where $P_{BS}(m, \sigma) = V^{BS}(1, e^{-m}, \sigma)$ is the BS price at unit spot. $V^{BS}$ and $P_{BS}$ denote the same analytic pricing function, where $P_{BS}$ has reduced dimensionality. Differentiating in $S$ at fixed $K$ and $\sigma$ yields the standard BS Delta and Gamma in terms of $m$:

$$\Delta_{BS} = P_{BS} + \frac{\partial P_{BS}}{\partial m}, \qquad \Gamma_{BS} = \frac{\partial P_{BS}}{\partial m} + \frac{\partial^2 P_{BS}}{\partial m^2}. \tag{168}$$

The model's predicted price can be obtained using the BS pricing formula which takes the predicted IV and log-moneyness as inputs,

$$\widehat{P}(m) \coloneqq P_{BS}(m,\, \sigma(m))\,, \tag{169}$$

with $\sigma(m)$ understood as the network's output (in terms of the unscaled $m$ by inverting Equation 156). Spot Delta and Gamma are derivatives of $\widehat{P}$ with respect to $S$. Let $\mathcal{V} = \partial P_{BS}/\partial\sigma$ denote Vega, $\mathcal{V}_m = \partial\mathcal{V}/\partial m$ its cross derivative with respect to log-moneyness, and Volga $= \partial\mathcal{V}/\partial\sigma$.

**1. Spot Delta.** Applying the total derivative with respect to $S$:

$$\Delta = \frac{d\widehat{P}}{dS} = \frac{\partial V^{BS}}{\partial S} + \frac{\partial V^{BS}}{\partial\sigma}\cdot\frac{\partial\sigma}{\partial S}. \tag{170}$$

Since $\partial\sigma/\partial S = (\partial\sigma/\partial m)(\partial m/\partial S) = (\alpha_m\,\partial\sigma/\partial\tilde{m})\,/S$, evaluating at $S = 1$ and substituting $\Delta_{BS} = P_{BS} + \partial P_{BS}/\partial m$ yields

$$\Delta = P_{BS} + \frac{\partial P_{BS}}{\partial m} + \mathcal{V}\left(\alpha_m\frac{\partial\sigma}{\partial\tilde{m}}\right). \tag{171}$$

**2. Spot Gamma.** From the Delta expression, $\Delta = \Delta_{BS}(m, \sigma) + \mathcal{V}(m, \sigma)\,\sigma_m$ with $\sigma_m = \partial\sigma/\partial m$. Differentiating with respect to $S$ via the chain rule, since $m$ and $\sigma(m)$ both depend on $S$:

$$\Gamma = \frac{d\Delta}{dS} = \frac{d\Delta_{BS}}{dS} + \frac{d\mathcal{V}}{dS}\,\sigma_m + \mathcal{V}\,\frac{d\sigma_m}{dS}. \tag{172}$$

Each total derivative expands via the chain rule, using $\partial m/\partial S = 1/S$ and $d\sigma/dS = \sigma_m/S$:

$$\frac{d\Delta_{BS}}{dS} = \frac{1}{S}\left[\frac{\partial \Delta_{BS}}{\partial m} + \frac{\partial \Delta_{BS}}{\partial \sigma}\,\sigma_m\right], \tag{173}$$

$$\frac{d\mathcal{V}}{dS} = \frac{1}{S}\left[\mathcal{V}_m + \text{Volga}\cdot\sigma_m\right], \tag{174}$$

$$\frac{d\sigma_m}{dS} = \frac{\sigma_{mm}}{S}. \tag{175}$$

Combining these:

$$\Gamma = \frac{1}{S}\left[\frac{\partial \Delta_{BS}}{\partial m} + \frac{\partial \Delta_{BS}}{\partial \sigma}\,\sigma_m\right] + \frac{1}{S}\left[\mathcal{V}_m + \text{Volga}\cdot\sigma_m\right]\sigma_m + \mathcal{V}\,\frac{\sigma_{mm}}{S}. \tag{176}$$

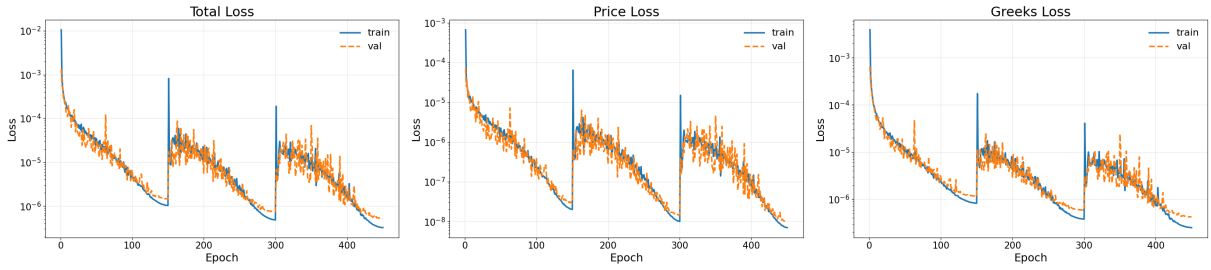
Using $\partial\Delta_{BS}/\partial m = \partial P_{BS}/\partial m + \partial^2 P_{BS}/\partial m^2$ and $\partial\Delta_{BS}/\partial\sigma = \mathcal{V} + \mathcal{V}_m$, evaluating at $S = 1$, and substituting $\sigma_m = \alpha_m\,\partial\sigma/\partial\tilde{m}$ and $\sigma_{mm} = \alpha_m^2\,\partial^2\sigma/\partial\tilde{m}^2$:

$$\begin{aligned}\Gamma = \frac{\partial P_{BS}}{\partial m} + \frac{\partial^2 P_{BS}}{\partial m^2} + \mathcal{V}\left(\alpha_m \frac{\partial\sigma}{\partial\tilde{m}}\right) + \mathcal{V}\left(\alpha_m^2 \frac{\partial^2\sigma}{\partial\tilde{m}^2}\right) \\ + 2\mathcal{V}_m\left(\alpha_m \frac{\partial\sigma}{\partial\tilde{m}}\right) + \text{Volga}\left(\alpha_m \frac{\partial\sigma}{\partial\tilde{m}}\right)^2,\end{aligned} \tag{177}$$

where $\partial P_{BS}/\partial m = \Delta_{BS} - P_{BS}$ and $\partial^2 P_{BS}/\partial m^2 = \Gamma_{BS} - \Delta_{BS} + P_{BS}$, with $\Delta_{BS}$ and $\Gamma_{BS}$ representing the standard analytic Black–Scholes Delta and Gamma evaluated at unit spot ($S = 1$).

# C Preliminary Training Convergence


(a) Before removing large IV Greeks.




(b) After removing large IV Greeks.

Figure 18: Convergence before and after removal of 99th percentile IV Greeks.

# References


Albrecher, H., Mayer, P. A., Schoutens, W., & Tistaert, J. (2007). The little heston trap. *Wilmott*, (1), 83–92. https://perswww.kuleuven.be/~u0009713/HestonTrap.pdf

Andersen, L., & Piterbarg, V. (2007). Moment explosions in stochastic volatility models. *Finance Stoch.*, *11*, 29–50.

Bakshi, G., Cao, C., & Chen, Z. (1997). Empirical performance of alternative option pricing models. *The Journal of Finance*, *52*(5), 2003–2049. https://doi.org/10.1111/j.1540-6261.1997.tb02749.x

Bates, D. S. (1996). Jumps and stochastic volatility: Exchange rate processes implicit in deutsche mark options. *Review of Financial Studies*, *9*(1), 69–107. https://doi.org/10.1093/rfs/9.1.69

Bishop, C. M. (1994). Neural networks and their applications. *Review of Scientific Instruments*, *65*(6), 1803–1832. https://doi.org/10.1063/1.1144830

Black, F., & Scholes, M. (1973). The pricing of options and corporate liabilities. *The Journal of Political Economy*, *81*(3), 637–654. http://www.jstor.org/stable/1831029

Bonnasse-Gahot, L. (2022). Interpolation, extrapolation, and local generalization in common neural networks. https://arxiv.org/abs/2207.08648

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., & Zhang, Q. (2018). *JAX: Composable transformations of Python+NumPy programs* (Version 0.3.13). http://github.com/jax-ml/jax

Brent, R. P. (1973). *Algorithms for minimization without derivatives*. Prentice-Hall.

Brezis, H. (2011). *Functional analysis, Sobolev spaces and partial differential equations*. Springer. https://doi.org/10.1007/978-0-387-70914-7

Brigo, D., Huang, X., Pallavicini, A., & Sáez de Ocáriz Borde, H. (2026). Interpretability in deep learning for finance: A case study for the heston model. *Risk Sciences*, *2*, 100030. https://doi.org/10.1016/j.risk.2025.100030

Cao, Y., Fang, Z., Wu, Y., Zhou, D.-X., & Gu, Q. (2019). Towards understanding the spectral bias of deep learning. *arXiv preprint arXiv:1912.01198*. https://doi.org/10.48550/arXiv.1912.01198

Carr, P., & Madan, D. B. (1999). Option valuation using the fast fourier transform. *Journal of Computational Finance*, *2*(4), 61–73. https://engineering.nyu.edu/sites/default/files/2018-08/CarrMadan2_0.pdf

Carr, P., & Madan, D. B. (2005). A note on sufficient conditions for no arbitrage. *Finance Research Letters*, *2*(3), 125–130. https://doi.org/10.1016/j.frl.2005.04.005

Chakhoyan, H., Kalici, S., & Malandain, K. A. (2025). Deepsvm: Learning stochastic volatility models with physics-informed deep operator networks. https://doi.org/10.48550/arXiv.2512.07162

Choi, J., Huh, J., & Su, N. (2024). Tighter 'uniform bounds for black-scholes implied volatility' and the applications to root-finding. https://doi.org/10.1016/j.orl.2024.107189

Cox, J. C., Ingersoll, J. E., & Ross, S. A. (1985). A theory of the term structure of interest rates. *Econometrica*, *53*(2), 385–408.

Cui, Y., del Baño Rollin, S., & Germano, G. (2017). Full and fast calibration of the heston stochastic volatility model. *European Journal of Operational Research*, *263*(2), 625–638. https://doi.org/10.1016/j.ejor.2017.05.018

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, *2*, 303–314. https://doi.org/10.1007/BF02551274

Czarnecki, W. M., Osindero, S., Jaderberg, M., Świrszcz, G., & Pascanu, R. (2017). Sobolev training for neural networks. https://doi.org/10.48550/arXiv.1706.04859

Dubey, S. R., Singh, S. K., & Chaudhuri, B. B. (2022). Activation functions in deep learning: A comprehensive survey and benchmark. https://doi.org/10.48550/arXiv.2109.14545

Dugas, C., Bengio, Y., Bélisle, F., Nadeau, C., & Garcia, R. (2000). Incorporating second-order functional knowledge for better option pricing. *Advances in Neural Information Processing Systems 13*, 472–478. https://proceedings.neurips.cc/paper/2000/hash/44968aece94f667e4095002d140b5896-Abstract.html

Evans, L. C. (1998). *Partial differential equations* (Vol. 19). American Mathematical Society.

Fang, F., & Oosterlee, C. W. (2008). A novel pricing method for european options based on fourier-cosine series expansions. *SIAM Journal on Scientific Computing*, *31*(2), 826–848. https://doi.org/10.1137/08071806

Faure, H., Pillichshammer, F., Pirsic, G., & Schmid, W. C. (2008). $L_2$ discrepancy of generalized two-dimensional hammersley point sets scrambled with arbitrary permutations. *Monatshefte für Mathematik*, *153*(1), 59–82. https://doi.org/10.1007/s00605-007-0493-5

Fazlyab, M., Robey, A., Hassani, H., Morari, M., & Pappas, G. J. (2019). Efficient and accurate estimation of Lipschitz constants for deep neural networks. *Advances in Neural Information Processing Systems*, *32*, 11423–11434.

Forrester, A. I. J., Sóbester, A., & Keane, A. J. (2008). Sampling plans. In *Engineering design via surrogate modelling: A practical guide*. John Wiley & Sons. https://ndl.ethernet.edu.et/bitstream/123456789/21891/1/25.pdf

Gambara, M., & Teichmann, J. (2021). Consistent recalibration models and deep calibration. https://doi.org/10.48550/arXiv.2006.09455

Garcia, R., & Gençay, R. (2000). Pricing and hedging derivative securities with neural networks and a homogeneity hint. *Journal of Econometrics*, *94*(1–2), 93–115. https://doi.org/10.1016/S0304-4076(99)00018-4

Gatheral, J. (2006). *The volatility surface: A practitioner's guide*. Wiley Finance.

Glasserman, P. (2004). *Monte carlo methods in financial engineering* (Vol. 53). Springer. https://doi.org/10.1007/978-0-387-21617-1

Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, *9*, 249–256. https://proceedings.mlr.press/v9/glorot10a.html

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning* [http://www.deeplearningbook.org]. MIT Press.

Gouk, H., Frank, E., Pfahringer, B., & Cree, M. J. (2020). Regularisation of neural networks by enforcing lipschitz continuity. https://doi.org/10.48550/arXiv.1804.04368

Han, J., Kamber, M., & Pei, J. (2012). *Data mining: Concepts and techniques* (3rd). Morgan Kaufmann. https://doi.org/10.1016/C2009-0-61819-5

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 1026–1034. https://doi.org/10.1109/ICCV.2015.123

Hernandez, A. (2016). Model calibration with neural networks. https://doi.org/10.2139/ssrn.2812140

Heston, S. L. (1993). 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. https://doi.org/10.1093/rfs/6.2.327

Huge, B., & Savine, A. (2020). Differential machine learning. https://doi.org/10.48550/arXiv.2005.02347

Hull, J. C. (2012). *Options, futures, and other derivatives* (8th ed.). Pearson.

Hull, J. C. (2022). *Options, futures, and other derivatives* (11th) [Global Edition, eBook ISBN: 978-1-292-41062-3]. Pearson Education Limited.

Hyndman, R. J., & Koehler, A. B. (2006). Another look at measures of forecast accuracy. *International Journal of Forecasting*, *22*(4), 679–688. https://doi.org/10.1016/j.ijforecast.2006.03.001

Islam, M. (2019). An overview of neural network. *American Journal of Neural Networks and Applications*. https://doi.org/10.11648/J.AJNNA.20190501.12

Jäckel, P. (2015). Let's be rational. *Wilmott*, *2015*(75), 40–53. https://doi.org/10.1002/wilm.10395

Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. *International Conference on Learning Representations (ICLR)*. https://openreview.net/forum?id=ryQu7f-RZ

Kumar, S. K. (2017). On weight initialization in deep neural networks. *arXiv preprint arXiv:1704.08863*. https://doi.org/10.48550/arXiv.1704.08863

LeCun, Y., Bottou, L., Orr, G. B., & Müller, K.-R. (1998). Efficient backprop. In G. B. Orr & K.-R. Müller (Eds.), *Neural networks: Tricks of the trade* (pp. 9–50, Vol. 1524). Springer Berlin Heidelberg. https://doi.org/10.1007/3-540-49430-8_2

Lin, W., Li, S., Luo, X., & Chern, S. (2017). Consistent pricing of VIX and equity derivatives with the 4/2 stochastic volatility plus jumps model. *Journal of Mathematical Analysis and Applications*, *447*(2), 778–797. https://doi.org/10.1016/j.jmaa.2016.10.039

Liu, S., Borovykh, A., Grzelak, L. A., & Oosterlee, C. W. (2019). A neural network-based framework for financial model calibration.

Liu, S., Oosterlee, C. W., & Bohte, S. M. (2019). Pricing options and computing implied volatilities using neural networks. *Risks*, *7*(1), 16. https://doi.org/10.3390/risks7010016

Loshchilov, I., & Hutter, F. (2017). Sgdr: Stochastic gradient descent with warm restarts. *International Conference on Learning Representations (ICLR)*. https://openreview.net/forum?id=Skq89Scxx

Loshchilov, I., & Hutter, F. (2019). Decoupled weight decay regularization. https://doi.org/10.48550/arXiv.1711.05101

Manaster, S., & Koehler, G. (1982). The calculation of implied variances from the black–scholes model: A note. *The Journal of Finance*, *37*(1), 227–230. https://doi.org/10.2307/2327127

McKay, M. D., Beckman, R. J., & Conover, W. J. (1979). A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, *21*(2), 239–245. https://doi.org/10.2307/1268522

Merton, R. C. (1973). Theory of rational option pricing. *The Bell Journal of Economics and Management Science*, *4*(1), 141–183. https://doi.org/10.2307/3003143

Mostafa, F., & Dillon, T. (2008). A neural network approach to option pricing. *Computational Finance and Its Applications III*, *41*, 71–80. https://doi.org/10.2495/CF080081

Nicholson, M. (n.d.). *Understanding sobol sampling* [Zemax OpticStudio documentation]. Ansys Optics. Retrieved April 28, 2026, from https://optics.ansys.com/hc/en-us/articles/43071055058323-Understanding-Sobol-sampling

Niederreiter, H. (1992). *Random number generation and quasi-monte carlo methods* (Vol. 63). Society for Industrial; Applied Mathematics (SIAM). https://www.ricam.oeaw.ac.at/files/people/siambook_nied.pdf

NIST/SEMATECH. (2012). E-handbook of statistical methods [Accessed: March 26th, 2026]. https://doi.org/10.18434/M32189

Oosterlee, C. W., & Grzelak, L. A. (2019). *Mathematical modeling and computation in finance: With exercises and python and matlab computer codes*. World Scientific Publishing Europe Ltd. https://doi.org/10.1142/Q0236

Owen, A. B. (2023). Quasi-monte carlo. In *Practical quasi-monte carlo integration*. Stanford University. https://artowen.su.domains/mc/practicalqmc.pdf

Peng, D., Gui, Z., & Wu, H. (2024). Interpreting the curse of dimensionality from distance concentration and manifold effect. https://doi.org/10.48550/arXiv.2401.00422

Phillips, G. M., & Taylor, P. J. (1996). Basic analysis. In *Theory and applications of numerical analysis* (Second, pp. 11–38). Academic Press. https://doi.org/10.1016/B978-012553560-1/50003-3

Poggio, T., Mhaskar, H., Rosasco, L., Miranda, B., & Liao, Q. (2017). Why and when can deep-but not shallow-networks avoid the curse of dimensionality: A review. *International Journal of Automation and Computing*, *14*(5), 503–519. https://doi.org/10.1007/s11633-017-1054-2

Prince, S. J. D. (2023). *Understanding deep learning*. The MIT Press. http://udlbook.com

Rahaman, N., Arpit, D., Baratin, A., Draxler, F., Lin, M., Hamprecht, F., Bengio, Y., & Courville, A. (2019). On the spectral bias of neural networks. *Proceedings of the 36th International Conference on Machine Learning*, *97*, 5301–5310. https://doi.org/10.48550/arXiv.1806.08734

Raina, R., Madhavan, A., & Ng, A. Y. (2009). Large-scale deep unsupervised learning using graphics processors. *Proceedings of the 26th International Conference on Machine Learning (ICML)*, 873–880. https://doi.org/10.1145/1553374.1553486

Ramachandran, P., Zoph, B., & Le, Q. V. (2017). Searching for activation functions. https://doi.org/10.48550/arXiv.1710.05941

Roper, M. (2010, March 26). *Arbitrage free implied volatility surfaces* (Working paper). School of Mathematics and Statistics, The University of Sydney.

Rouah, F. D. (2013). *The heston model and its extensions in matlab and c#*. John Wiley & Sons, Inc.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*, 533–536. https://doi.org/10.1038/323533a0

Sharma, S., Sharma, S., & Athaiya, A. (2020). Activation functions in neural networks. *International Journal of Engineering Applied Sciences and Technology (IJEAST)*, *4*(12), 310–316. http://www.ijeast.com

Shreve, S. E. (2004). *Stochastic calculus for finance ii: Continuous-time models*. Springer.

Sobol', I. M., Asotsky, D., Kreinin, A., & Kucherenko, S. (2011). Construction and comparison of high-dimensional sobol' generators. *Wilmott*, *2011*(56), 64–79. https://doi.org/10.1002/wilm.10056

Son, H., Jang, J. W., Han, W. J., & Hwang, H. J. (2021). Sobolev training for physics informed neural networks. https://arxiv.org/abs/2101.08932

Sridi, A., & Bilokon, P. (2023). Applying deep learning to calibrate stochastic volatility models. https://doi.org/10.2139/ssrn.4572108

Verleysen, M., & François, D. (2005). The curse of dimensionality in data mining and time series prediction. *Computational Intelligence and Bioinspired Systems*, *3512*, 758–770. https://doi.org/10.1007/11494669_93

Virmaux, A., & Scaman, K. (2018). Lipschitz regularity of deep neural networks: Analysis and efficient estimation. *Advances in Neural Information Processing Systems*, *31*, 3835–3844.

Wang, C. (2017). Pricing european options by stable fourier-cosine series expansions [[v2] 8 Jan 2017]. *arXiv preprint arXiv:1701.00886*. https://arxiv.org/abs/1701.00886

Wang, Q., Ma, Y., Zhao, K., & Tian, Y. (2022). A comprehensive survey of loss functions in machine learning. *Annals of Data Science*, *9*(2), 187–212. https://doi.org/10.1007/s40745-020-00253-5

Weber, J. (2018, December). Introduction to Sobolev spaces [Last revised November 17, 2025]. https://www.math.stonybrook.edu/~joa/PUBLICATIONS/SOBOLEV.pdf

Weng, T.-W., Zhang, H., Chen, P.-Y., Yi, J., Su, D., Gao, Y., Hsieh, C.-J., & Daniel, L. (2018). Evaluating the robustness of neural networks: An extreme value theory approach. *arXiv*. https://doi.org/10.48550/arxiv.1801.10578

Willmott, C. J., & Matsuura, K. (2005). Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance. *Climate Research*, *30*(1), 79–82. https://doi.org/10.3354/cr030079

Wu, Y.-c., & Feng, J.-w. (2018). Development and application of artificial neural network. *Wireless Personal Communications*, *102*(2), 1645–1656. https://doi.org/10.1007/s11277-017-5224-x

Zhang, C., Amici, G., & Morandotti, M. (2025). Calibrating the heston model with deep differential networks. https://doi.org/10.48550/arXiv.2407.15536

Zhang, Z. (2018). Artificial neural network. In *Multivariate time series analysis in climate and environmental research* (pp. 1–35). Springer International Publishing. https://doi.org/10.1007/978-3-319-67340-0_1

**DEPARTMENT OF TECHNOLOGY MANAGEMENT AND ECONOMICS**
**CHALMERS UNIVERSITY OF TECHNOLOGY**
Gothenburg, Sweden 2026
www.chalmers.se


CHALMERS