Model-Agnostic Pricing of Exotic Derivatives Using Signatures

Andrew Alden
King's College London
London, United Kingdom
andrew.alden@kcl.ac.uk

Carmine Ventre
King's College London
London, United Kingdom
carmine.ventre@kcl.ac.uk

Blanka Horvath
University of Oxford
Oxford, United Kingdom
Blanka.Horvath@maths.ox.ac.uk

Gordon Lee
Independent Consultant
London, United Kingdom
gtylee@gmail.com

ABSTRACT

Neural networks hold out the promise of fast and reliable derivative pricing. Such an approach usually involves the supervised learning task of mapping contract and model parameters to derivative prices.

In this work, we introduce a model-agnostic path-wise approach to derivative pricing using higher-order distribution regression. Our methodology leverages the 2nd-order Maximum Mean Discrepancy (MMD), a notion of distance between stochastic processes based on path signatures. To overcome the high computational cost of its calculation, we pre-train a neural network that can quickly and accurately compute higher-order MMDs. This allows the combination of distribution regression with neural networks in a computationally feasible way. We test our model on down-and-in barrier options. We demonstrate that our path-wise approach extends well to the high-dimensional case by applying it to rainbow options and autocallables. Our approach has a significant speed-up over Monte Carlo pricing.

CCS CONCEPTS

• **Computing methodologies** → *Kernel methods*; *Neural networks*;
• **Mathematics of computing** → **Stochastic processes**.

KEYWORDS

Fast Derivative Pricing, Path Signature, Barrier Options, Rainbow Options, Autocallables

ACM Reference Format:
Andrew Alden, Carmine Ventre, Blanka Horvath, and Gordon Lee. 2022. Model-Agnostic Pricing of Exotic Derivatives Using Signatures. In *3rd ACM International Conference on AI in Finance (ICAIF '22), November 2–4, 2022, New York, NY, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3533271.3561740

1 INTRODUCTION

A common approach to derivative pricing using neural networks involves the supervised learning task of mapping contract and model parameters to derivative prices [28, Table 1]. This approach assumes a stochastic model, such as the Heston model [16, 24], that describes the asset dynamics. However, this standard supervised learning approach does not necessarily lend itself well to model-agnostic pricing where market simulators such as Generative Adversarial Networks [35], Variational Autoencoders (VAEs) [7], and normalising flows [36], are trained to generate realistic price paths from a statistical measure.

The path signature is a transform which translates the structure of a path into a vector [22]. It forms the basis of Distribution Regression on sequential data [22]. The authors of [22] use the 1st-order Maximum Mean Discrepancy (MMD) to perform distribution regression. Using higher-order kernel mean embeddings (KMEs), the authors of [30] formulate the 2nd-order MMD as the distance between the KMEs in a 'higher-order' Reproducing Kernel Hilbert Space (RKHS). This allows them to extend distribution regression to higher-order distribution regression by using the 2nd-order MMD instead of the 1st-order MMD.

Since neural networks have been used to accurately price derivatives [18], in this paper we present a novel framework which applies neural networks to the distribution regression problem on sequential data. Generating the vast amount of data using the 2nd-order MMD required to train a neural network is time-consuming and resource-expensive. To overcome this, we assume that a model describes the dynamics of the asset(s) and pre-train a neural network to map two stochastic models represented by two separate model parameters to the square of their 2nd-order MMD. This allows us to generate the required data in a short time. Since the 2nd-order MMD is computed using a collection of sample paths, this approach can be used to price in a model-agnostic environment. We apply a different approach to [26] for pricing exotic derivatives using signatures.

We test our pipeline on three exotic options. We train a model to price down-and-in barrier options under the Heston model, an exotic derivative which has proven challenging to price using signatures [2]. We compare our neural network with another neural network trained using the standard supervised learning approach. Our pricing model has a higher accuracy than the standard approach. We then test our approach on two basket options. To model multiple asset price processes, we use an extension of the Black-Scholes model to the multi-dimensional case. The first type of basket option tested is the best-call rainbow option. We show that our pricing model outperforms the standard neural network approach in terms of accuracy. Finally, we test our model on an autocallable with memory on a basket of three assets. Our pricing model manages to achieve a high accuracy on this highly path-dependent derivative.

Our pricing pipeline allows us to price derivatives much faster than with Monte Carlo. A large part of this speed-up is due to the pre-trained MMD approximator, which is significantly faster than the empirical estimator provided in [30].

2 PROBLEM FORMULATION

Consider M pairs $\left\{\left(\mathbb{X}^i, y^i\right)\right\}_{i=1}^M$ where each pair consists of a derivative price $y^i \in \mathbb{R}$ and a stochastic process $\mathbb{X}^i$ modelling the underlying asset(s). The objective is to regress the stochastic processes against the derivative price. This is achieved by solving the distribution regression [22] supervised learning problem

$$F\colon \delta^i \to \mathbb{R}, \tag{1}$$

where δ^i is the empirical probability measure of the collection of N_i paths

$$\left\{\boldsymbol{x}^{i,k} = \left\{\left(t_1, \boldsymbol{x}^{i,k}_{t_1}\right), \cdots, \left(t_{l_{i,k}}, \boldsymbol{x}^{i,k}_{t_{l_{i,k}}}\right)\right\}\right\}_{k=1}^{N_i}$$

with values $\boldsymbol{x}^{i,k}_{t_j} \in \mathbb{R}^d$ and length $l_{i,k}$ sampled from the stochastic process $\mathbb{X}^i$.

3 THEORETICAL BACKGROUND

3.1 Path Signature

Let $\mathbb{T} := [0,T]$ and $\mathbb{T} := \{0 = t_0 < t_1 < \cdots < t_N = T\}$ be a continuous and discrete time index respectively. Also, let $(\Omega, \mathcal{F}, \mathbb{F} := (\mathcal{F}_t)_{t\in\mathbb{T}}, \mathbb{P})$ be a filtered probability space and $\boldsymbol{X} := (\boldsymbol{X}_t)_{t\in\mathbb{T}}$ be an $\mathbb{R}^d$-valued $\mathbb{F}$-adapted process defined on Ω. If $\mathbb{T} = [0,T]$, assume that $\boldsymbol{X}$ has càdlàg sample paths. As defined in [4], $\mathbb{X} := (\Omega, \mathcal{F}, \mathbb{F}, \mathbb{P}, \boldsymbol{X})$ is a filtered process. Consider the Hilbert space $(\mathcal{H}, \langle\cdot,\cdot\rangle_{\mathcal{H}})$ and let $\mathcal{K} \subseteq \mathcal{H}$ be a compact subset. Let $\mathcal{X}(\mathcal{K}) := \{\boldsymbol{x}\colon \mathbb{T} \to \mathcal{K}\}$ be the compact set of continuous, piecewise linear paths with values in $\mathcal{K}$ obtained by linearly interpolating a discrete time path $\boldsymbol{x} = (\boldsymbol{x}_t)_{t\in\mathbb{T}}$ defined on $\mathbb{T} = \{0 = t_0 < t_1 < \cdots < t_N = T\}$. The path signature [32, 33] $\mathcal{S}\colon \mathcal{X}(\mathcal{K}) \to T(\mathcal{H})$ for any d-dimensional path $\boldsymbol{x} \in \mathcal{X}(\mathcal{K})$ is defined as the infinite collection of statistics

$$\mathcal{S}(\boldsymbol{x}) := \left(1, \left\{\mathcal{S}(\boldsymbol{x})^{(k_1)}\right\}_{k_1=1}^d, \left\{\mathcal{S}(\boldsymbol{x})^{(k_1,k_2)}\right\}_{k_1,k_2=1}^d, \cdots\right)$$

where each term is the iterated integral

$$\mathcal{S}(\boldsymbol{x})^{(k_1,\cdots,k_j)} := \int\cdots\int_{0<s_1<\cdots<s_j<T} d\boldsymbol{x}^{(k_1)}_{s_1}\ldots d\boldsymbol{x}^{(k_j)}_{s_j}$$

and $\boldsymbol{x}^{(k)}$ denotes the kth-coordinate. The signature feature space is defined as the direct sum of tensor powers of $\mathcal{H}$

$$T(\mathcal{H}) := \bigoplus_{i=0}^{\infty} \mathcal{H}^{\otimes i}$$

where $\otimes$ denotes the tensor product of vector spaces.

3.2 Signature Kernel

Let $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{X}(\mathcal{K})$. The signature kernel [30] $k_{\mathcal{S}}\colon \mathcal{X}(\mathcal{K}) \times \mathcal{X}(\mathcal{K}) \to \mathbb{R}$ is a characteristic kernel defined as the following inner product

$$k_{\mathcal{S}}(\boldsymbol{x}, \boldsymbol{y}) := \langle \mathcal{S}(\boldsymbol{x}), \mathcal{S}(\boldsymbol{y})\rangle_{T(\mathcal{H})}.$$

To compute the signature kernel between two paths we use the kernel trick presented in [29, Theorem 2.5].

3.3 Kernel Mean Embeddings

KMEs are used to represent the laws of stochastic processes as elements in a RKHS. Let $\mathcal{H}_{\mathcal{S}}$ denote the RKHS with $k_{\mathcal{S}}$ being the reproducing kernel. For any filtered stochastic process $\mathbb{X}$ with values in $\mathcal{K}$ satisfying the integrability condition $\mathbb{E}_{\mathbb{P}_{\boldsymbol{X}}}\left[k_{\mathcal{S}}(\boldsymbol{X}, \boldsymbol{X})\right] < \infty$, we embed the law $\mathbb{P}_{\mathbb{X}}$ in the RKHS $\mathcal{H}_{\mathcal{S}}$ using the signature KME [30] defined as

$$\mu_{\boldsymbol{X}} := \mathbb{E}_{\mathbb{P}_{\boldsymbol{X}}}\left[k_{\mathcal{S}}(\boldsymbol{X}, \cdot)\right] = \int_{\boldsymbol{x}\in\mathcal{X}(\mathcal{K})} k_{\mathcal{S}}(\boldsymbol{x}, \cdot)\, \mathbb{P}_{\boldsymbol{X}}(d\boldsymbol{x}).$$

Since stochastic processes contain a richer structure than standard random variables, the KME can be extended to embed the conditional law of the stochastic process (the law of the stochastic process conditioned on this structure). Mathematically, this added structure is described by the filtration $(\mathcal{F}_t)_{t\in\mathbb{T}}$ and we can condition the law of the stochastic process on the information available up to a particular time $(\mathcal{F}_t)$ [30]. The 1st-order KME of the conditional law $\mathbb{P}_{\boldsymbol{X}|\mathcal{F}_{\boldsymbol{X}t}}$ [30] is a $\mathcal{F}_{\boldsymbol{X}t}$-measurable, $\mathcal{H}_{\mathcal{S}}$-valued random variable defined as

$$\mu^1_{\boldsymbol{X}|\mathcal{F}_{\boldsymbol{X}t}} := \mathbb{E}\left[k_{\mathcal{S}}(\boldsymbol{X}, \cdot)\,|\mathbb{X}_{[0,t]}\right] = \int_{\boldsymbol{x}\in\mathcal{X}(\mathcal{K})} k_{\mathcal{S}}(\boldsymbol{x}, \cdot)\, \mathbb{P}_{\boldsymbol{X}|\mathcal{F}_{\boldsymbol{X}t}}(d\boldsymbol{x})$$

where $\mathbb{X}_{[0,t]}$ denotes the stochastic process restricted to the times $[0,t] \in \mathbb{T}$. By running the time index t over all possible time indexes, we obtain the following ordered collection of 1st-order conditional KMEs $\mu^1_{\boldsymbol{X}|\mathbb{F}} := \left(\mu^1_{\boldsymbol{X}|\mathcal{F}_{\boldsymbol{X}t}}\right)_{t\in\mathbb{T}}$. We will refer to this collection as the 1st-order predictive KME process of $\boldsymbol{X}$.

Since $\mu^1_{\boldsymbol{X}|\mathbb{F}}$ is a path taking values in the space of $\mathcal{H}_{\mathcal{S}}$-valued random variables, it is a stochastic process. Therefore, we can embed the law of $\mu^1_{\boldsymbol{X}|\mathbb{F}}$ via KMEs into a higher-order RKHS. The 2nd-order KME [30] is defined to be the point in $\mathcal{H}^2_{\mathcal{S}} = \mathcal{H}_{\mathcal{S}}(\mathcal{H}_{\mathcal{S}})$ which is the KME of the 1st-order predictive KME, i.e.,

$$\mu^2_{\boldsymbol{X}} := \mathbb{E}\left[k_{\mathcal{S}}\left(\mu^1_{\boldsymbol{X}|\mathbb{F}}, \cdot\right)\right] = \int_{\boldsymbol{x}\in\mathcal{X}(\mathcal{H}_{\mathcal{S}}(\mathcal{K}))} k_{\mathcal{S}}(\boldsymbol{x}, \cdot)\, \mathbb{P}_{\mu^1_{\boldsymbol{X}|\mathbb{F}}}(d\boldsymbol{x}).$$

3.4 Distance Between Stochastic Processes

By embedding the laws of stochastic processes in a RKHS using KMEs, we can define distances between stochastic processes using the norm in the RKHS. The 1st-order MMD [30] between two stochastic processes $\mathbb{X}$ and $\mathbb{Y}$ is the distance in $\mathcal{H}_{\mathcal{S}}$ between the two KMEs $\mu^1_{\boldsymbol{X}}, \mu^1_{\boldsymbol{Y}}$.

Although the 1st-order MMD distinguishes between laws of two stochastic processes [13], it does not incorporate the filtration of the stochastic process when calculating the distance. Since stochastic processes contain this rich structure, using distances which absorb this added information could prove a better representation of the true distance. In [30], they generalise the 1st-order MMD to a higher-order MMD, which makes better use of the information flow when computing distances. The 2nd-order MMD of two stochastic processes $\mathbb{X}$ and $\mathbb{Y}$ is defined as the norm of the difference in $\mathcal{H}^2_{\mathcal{S}}$ of their 2nd-order KMEs,

$$\mathcal{D}^2_{\mathcal{S}}(\mathbb{X},\mathbb{Y}) = \left\|\mu^2_{\mathbf{X}} - \mu^2_{\mathbf{Y}}\right\|_{\mathcal{H}^2_{\mathcal{S}}}.$$

It can be shown that the 2nd-order MMD is a stronger discrepancy measure than the 1st-order MMD [30, Theorem 2]. A consistent [30, Theorem 3] empirical estimate of the 2nd-order MMD is provided in [30, Section 3.4].

4 A DISTANCE-BASED FRAMEWORK FOR DERIVATIVE PRICING

Consider a collection of M stochastic processes $\left\{\mathbb{X}^i\right\}_{i=1}^M$ and M derivative prices $\left\{y^i := \mathbb{E}^{\mathbb{Q}}\left[\text{Payoff}\left(\boldsymbol{X},\Xi^i\right)\right]\right\}_{i=1}^M$ where Ξ^i are the contract parameters and $\mathbb{Q}$ is the risk-neutral probability measure. For example, if the derivative is a European Call Option with underlying S, strike K, and maturity T, then $\Xi = (K,T)$ and $\text{Payoff}(S,\Xi) = (S_T - K)^+$.

As outlined in Section 2, the distribution regression problem involves learning the function F provided in Eq. (1). In most cases, when pricing financial instruments, the optimal function F^* is discontinuous with respect to the 1st-order MMD but continuous with respect to the 2nd-order MMD [30]. Therefore, we use the 2nd-order MMD as features in the model.

In previous works [22, 30] the feature vectors consisted of computing M distances, each distance corresponding to one of the M stochastic processes in the dataset. In our work, we only use $N \ll M$ distances as features, significantly scaling down the dimensionality. This lends itself well to training neural networks. Training neural networks requires a vast amount of data and so computing distances to all processes in the training data is a very time consuming and resource expensive procedure.

The process of using $N \ll M$ distances is very similar to bagging [6, 8, 14]. In bagging, multiple 'weak' models are trained using a subset of the dataset (sampled with replacement) and then aggregated (usually averaged) to yield a better result. This results in reduced variance whilst maintaining a similar bias [6, 8]. In our case, one reference model is not enough to train an accurate pricing model. We need to price based on distances relative to $N\,(>1)$ randomly sampled models. Independently to our work and applied to a completely different set of examples, [20] use landmark points as part of their models when solving the distribution regression problem.

Let $\mathcal{P}(\mathcal{X}(\mathcal{K}))$ denote the set of stochastic processes with sample paths in $\mathcal{X}(\mathcal{K})$. Let $f\colon \mathbb{R}\to\mathbb{R}$ be defined as $f(x) := x^2$ and define the kernel $K_{\mathcal{S}}\colon \mathcal{P}(\mathcal{X}(\mathcal{K}))\times\mathcal{P}(\mathcal{X}(\mathcal{K}))\to\mathbb{R}$ as $K_{\mathcal{S}}(\mathbb{X},\mathbb{Y}) := f\left(\mathcal{D}^2_{\mathcal{S}}(\mathbb{X},\mathbb{Y})\right) = \mathcal{D}^2_{\mathcal{S}}(\mathbb{X},\mathbb{Y})^2$. The objective is to learn the mapping F by regressing N squared distances (along with contract parameters Ξ) against the y^i. Therefore, the feature vectors are of the form

$$\left(K_{\mathcal{S}}\left(\mathbb{Y},\mathbb{X}^1\right),\cdots,K_{\mathcal{S}}\left(\mathbb{Y},\mathbb{X}^N\right),\Xi\right).$$

5 EXPERIMENTS

5.1 Novel Model Pipeline and Experimental Setup

Computing the 2nd-order MMD is similar to a Monte-Carlo procedure on sample paths. Since its computation is model-agnostic, our approach to pricing is also model-agnostic. Therefore, we can use sample paths generated by a generative model to price derivatives. However, in our experiments we use sample paths obtained using a stochastic model.

We use the following novel pipeline:

(1) train a neural network to approximate $\mathcal{D}^2_{\mathcal{S}}(\cdot,\cdot)^2$ mapping stochastic model parameters to squared distances,
(2) train a regression model.

To obtain the price of a derivative with contract parameters Ξ^{new} under a new model $\mathbb{Y}$, we first compute the feature vector

$$\tilde{\boldsymbol{x}} = \left(\mathcal{D}^2_{\mathcal{S}}\left(\mathbb{X}^1,\mathbb{Y}\right)^2,\cdots,\mathcal{D}^2_{\mathcal{S}}\left(\mathbb{X}^N,\mathbb{Y}\right)^2,\Xi^{\text{new}}\right)$$

using the pre-trained neural network approximator of the squared 2nd-order MMD where $\left\{\mathbb{X}^1,\cdots,\mathbb{X}^N\right\}$ are the N base processes. We then pass this as input to the pricing model which computes the price of the new derivative. In all experiments, we use 80% of the data for training and the remaining 20% as a validation set [1]. We make use of mini-batches during training (as described in [12]). All neural networks are trained to minimise the Mean Squared Error (MSE) using the AdamW optimiser [25]. Moreover, to prevent overfitting and speed up convergence, we use the following regularization techniques:

- save the model with the best validation score and use that model when testing; and
- standardise features through centering and scaling.

We consider two stochastic models; one for derivatives based on a single underlying and the other for derivatives with multiple underlyings. To model the single underlying, we use the Heston model [16]. The Heston model is a stochastic volatility model such that under risk-neutral dynamics, the spot at time t follows the diffusion

$$dS_t = rS_t dt + \sqrt{v_t}S_t dW^S_t$$

where the variance follows a Cox, Ingersoll, and Ross [10] process

$$dv_t = \kappa\left(\theta - v_t\right)dt + \xi\sqrt{v_t}dW^v_t.$$

$\left(W^S_t\right)_t, \left(W^v_t\right)_t$ are correlated Brownian Motions with correlation ρ (i.e. $\langle dW^S_t, dW^v_t\rangle = \rho dt$) and v_t is a mean-reverting process such that

- θ is the long variance,
- κ is the speed of mean-reversion,
- ξ is the volatility of the volatility process.

To model multiple assets, an extension of the Black-Scholes model [5] to the multi-dimensional case is used [9, 34] (Multi-Geometric Brownian Motion (MGBM)). Suppose we have n-assets with corresponding price process $\left(\boldsymbol{S}^n_t\right)_t$. The risk-neutral dynamics

[1]Data was split as described in [1].

of the i^{th} price process is governed by the stochastic differential equation (SDE)

$$dS_t^i = rS_t^i dt + \sigma^i S_t^i dW_t^i$$

where $\left\{\left(W_t^j\right)_t\right\}_{j=1}^n$ are a collection of correlated Brownian Motions such that $\langle W_t^i, W_t^j\rangle = \rho_{i,j}dt$, $\rho_{i,i} = 1$, and $dS^i dS^j = \sigma^i\sigma^j S^i S^j \rho_{i,j}dt$.

5.2 Neural Network Approximation of the 2nd-order MMD

When working with the Heston model, a neural network

$$\mathcal{NN}_{\mathcal{D}^2}^{\text{Heston}} : \mathbb{R}^{10} \to \mathbb{R}$$

is trained to approximate the square of the 2nd-order MMD between two stochastic processes described by a Heston model. The input to the neural network is a pair of collections of model parameters (Θ_1, Θ_2), with $\Theta_k = \left(v_0^k, \theta_k, \kappa_k, \xi_k, \rho_k\right)$ for $k = 1, 2$. Therefore, the feature vector is of the form

$$\boldsymbol{x} = \left(v_0^1, v_0^2, \xi_1, \xi_2, \theta_1, \theta_2, \kappa_1, \kappa_2, \rho_1, \rho_2\right).$$

The model was trained to regress 45,000 pairs (Θ_i, Θ_j) against the squared distances $\mathcal{D}_{\mathcal{S}}^2(\Theta_i, \Theta_j)^2$. The neural network consisted of 4 hidden-layers of size 50 with Rectified Linear Unit (ReLU) activation function and an output layer with linear activation function.

To compute $\mathcal{D}_{\mathcal{S}}^2(\Theta_i, \Theta_j)^2$, the GitHub Repository [2] provided in [30] was used. Θ_i, Θ_j were generated using the randomised procedure: randomly sample a set of parameters $\left\{\Theta^i\right\}_i$ and for each new parameter $\tilde{\Theta}$ compute the distance to each one of the Θ^i. To compute the 2nd-order MMD, 300 sample paths of length 15 (including starting value) with a time-horizon of 1 were used. They were generated using the Heston Euler discretisation scheme

$$S_{t+1} = S_t + r\cdot S_t\cdot\Delta t + S_t\cdot\sqrt{v_t}\cdot\sqrt{\Delta t}Z^1$$
$$v_{t+1} = v_t + \kappa\cdot(\theta - v_t)\cdot\Delta t + \xi\cdot\sqrt{v_t}\cdot\sqrt{\Delta t}\cdot\left(\rho Z^1 + \left(1-\rho^2\right)Z^2\right)$$

where v_0, S_0 are provided and $Z^1, Z^2 \sim \mathcal{N}(0,1)$. To ensure that the square root can be computed, the square root of $\max(0, v_t)$ was computed instead. The code to simulate the Heston model was adapted from the GitHub Repository [3] corresponding to the work in [15]. In our simulations, S_t was updated before v_t. To calculate the 2nd-order MMD, we set $S_0 = 0.5$ for all simulations. Figure 1a shows the distribution of the squared 2nd-order MMD under both the null hypothesis and the alternative hypothesis [13, 30]

$$\mathcal{H}_0 : \mathbb{P}_{X|\mathcal{F}_X} = \mathbb{P}_{Y|\mathcal{F}_Y} \text{ and } \mathcal{H}_A : \mathbb{P}_{X|\mathcal{F}_X} \neq \mathbb{P}_{Y|\mathcal{F}_Y}.$$

Clearly, the 2nd-order MMD can distinguish between these two stochastic processes. When training the model, one value from the distribution of distances is used as the target distance. The value could be in the tails of the distribution. This along with the variance in the 2nd-order MMD could negatively affect the training of the neural network. If $\left\|\Theta_i - \Theta_j\right\| < \epsilon$, then it is likely that there is a substantial overlap in the empirical distributions of $\mathcal{D}_{\mathcal{S}}^2(\Theta_i, \Theta_k), \mathcal{D}_{\mathcal{S}}^2(\Theta_j, \Theta_k)$ for some $\Theta_k \in \mathbb{R}^5$. If the target variable is selected from the tails furthest from the overlapping section, it would not necessarily represent the true distance between the processes, making it difficult for a neural network to be trained. A possible remedy for this is generating the target variables by taking the mean of the distribution over the distances.

The neural network obtained a validation loss of 4.3×10^{-2}. To ensure that $\sqrt{\mathcal{NN}_{\mathcal{D}^2}^{\text{Heston}}}$ approximates a metric, we check whether

- $\sqrt{\mathcal{NN}_{\mathcal{D}^2}^{\text{Heston}}(\Theta,\Theta)} = 0$ for all Θ,
- $\sqrt{\mathcal{NN}_{\mathcal{D}^2}^{\text{Heston}}(\Theta_1,\Theta_2)} = \sqrt{\mathcal{NN}_{\mathcal{D}^2}^{\text{Heston}}(\Theta_2,\Theta_1)}$ for all Θ_1, Θ_2, and
- $\sqrt{\mathcal{NN}_{\mathcal{D}^2}^{\text{Heston}}(\Theta_1,\Theta_3)} \leq \sqrt{\mathcal{NN}_{\mathcal{D}^2}^{\text{Heston}}(\Theta_1,\Theta_2)}$ $+\sqrt{\mathcal{NN}_{\mathcal{D}^2}^{\text{Heston}}(\Theta_2,\Theta_3)}$ for all $\Theta_1, \Theta_2, \Theta_3$.

We used 50,000 parameter combinations for each test; 99.5% satisfied the condition $\mathcal{NN}_{\mathcal{D}^2}^{\text{Heston}}(\Theta,\Theta) < 0.05$, 100% satisfied the symmetry condition, and 100% satisfied the triangle inequality.

A similar procedure was applied to the multi-dimensional Black-Scholes model. Given n volatilities $\sigma_1, \cdots, \sigma_n$, and a constant correlation $\rho \in (0,1)$ (as in [21]), generate the covariance matrix $\boldsymbol{\Sigma} = \{\Sigma_{i,j} = \rho\sigma_i\sigma_j\}_{i,j=1}^n$. Let $\boldsymbol{\Sigma} = \mathbf{L}\mathbf{L}^T$ be the Cholesky decomposition of Σ. We simulate 200 sample paths of length 25 (including initial spot value) with time horizon 1 using the discretisation scheme [11]

$$\boldsymbol{S}_{t_{i+1}} = \boldsymbol{S}_{t_i}\exp\left(\left(\boldsymbol{r} - \frac{1}{2}\boldsymbol{\sigma}^2\right)\Delta t + \sqrt{\Delta t}\mathbf{L}\boldsymbol{Z}_i\right)$$

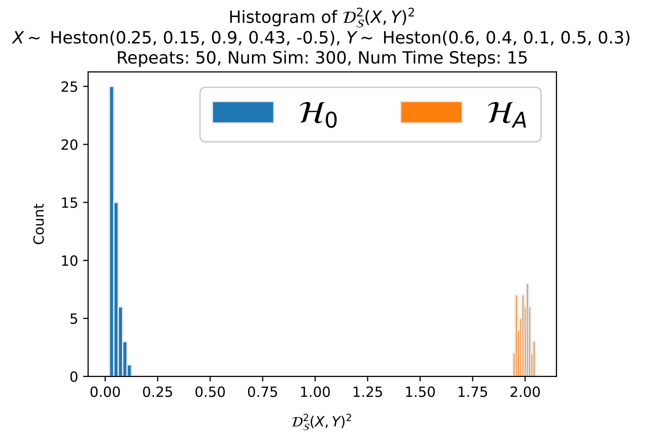
where $\boldsymbol{S}_{t_i}$ is the realisation of the multi-dimensional process at time t_i, Δt is the time-increment between time-steps, $\boldsymbol{Z}_i \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_n)$, $\mathbf{I}_n$ is the $n\times n$ identity matrix, $\boldsymbol{r} = r\mathbf{1}$, $\mathbf{1}$ is the all 1-vector, and $\boldsymbol{\sigma} = (\sigma_1, \cdots, \sigma_n)$. We use Sobol sequences [19, 21, 31] to simulate the Brownian Motion in the multi-dimensional case. The code to simulate the MGBM was adapted from the code used in [21]. We set $n = 3$ and $\mathbf{S}_0 = (0.5, 0.5, 0.5)$. The distribution of the squared 2nd-order MMD under both the null and the alternative hypotheses is depicted in Figure 1b. The same procedure was applied to train the neural network $\mathcal{NN}_{\mathcal{D}^2}^{\text{MGBM}} : \mathbb{R}^8 \to \mathbb{R}$ which learns to map stochastic model parameters to squared distances in the multi-dimensional Black-Scholes model (keeping dimension fixed at 3). The input to the neural network $\mathcal{NN}_{\mathcal{D}^2}^{\text{MGBM}}$ are vectors of the form $\boldsymbol{x} = \left(\sigma_1^1, \sigma_1^2, \sigma_2^1, \sigma_2^2, \sigma_3^1, \sigma_3^2, \rho^1, \rho^2\right)$. The neural network was trained using 20,000 samples. This neural network had a similar architecture as $\mathcal{NN}_{\mathcal{D}^2}^{\text{Heston}}$, the only difference being that the layers are of size 30. A validation loss of 1.8×10^{-3} was obtained. Similar tests as in the case of the Heston model show that the neural network $\sqrt{\mathcal{NN}_{\mathcal{D}^2}^{\text{MGBM}}}$ approximates a metric.

5.3 Pricing Down-and-In Barrier European Call Options in the Heston Model

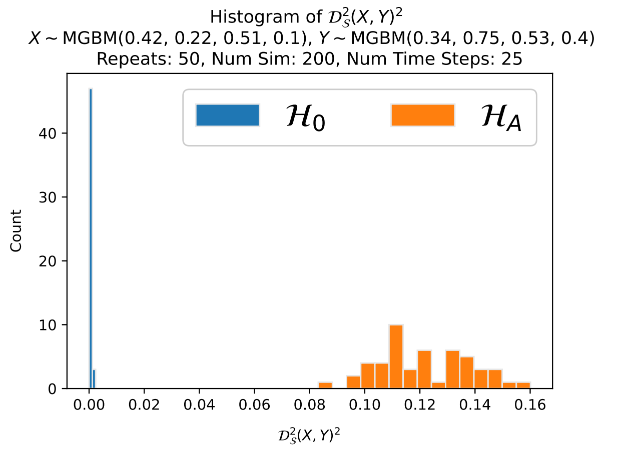
A barrier option is a path-dependent option where the payoff at maturity is dependent on the past performance of the underlying. We consider down-and-in barrier European call options. Define the running minimum of the asset price $m_T := \min_{0\leq u\leq T} S_u$ and let $\mathbb{1}$ denote the indicator function. The payoff of a down-and-in barrier option with strike K, barrier B, and maturity T is

$$(S_T - K)^+\,\mathbb{1}_{m_T\leq B}.$$

[2]https://github.com/maudl3116/higherOrderKME
[3]https://github.com/HeKrRuTe/OptStopRandNN



(a) Empirical distribution of the squared 2nd-order MMD in the Heston model.



(b) Empirical distribution of the squared 2nd-order MMD in the MGBM model.

Figure 1: Distribution of $\mathcal{D}^2(\cdot,\cdot)^2$ under the null hypothesis $\mathcal{H}_0$ and under the alternative hypothesis $\mathcal{H}_A$.

The stochastic model and contract parameters were sampled as follows:

- $S_0 \sim \mathcal{U}_{\text{disc}}(50, 80)$,
- $K \sim \mathcal{U}_{\text{disc}}(40, 90)$
- $r \sim \mathcal{U}(0.01, 0.1)$ (rounded to 5 decimal places)
- $365 \cdot T$ days where $T \sim \mathcal{U}_{\text{disc}}(\{0.5 + 0.25 \cdot i : 0 \le i \le 10\})$
- $B \sim \mathcal{U}_{\text{disc}}(S_0 - 25, S_0 - 5)$
- $v_0, \theta, \kappa, \xi \sim \mathcal{U}(0.2, 0.5)$ (rounded to 5 decimal places)
- $\rho \sim \mathcal{U}(-1, 1)$ (rounded to 5 decimal places)

where $\mathcal{U}$ denotes the uniform distribution and $\mathcal{U}_{\text{disc}}$ denotes the discrete uniform distribution on the integers.

The regression model is a neural network

$$\mathcal{NN}_{\text{Price}}^{\text{Dist-Barrier}} : \mathbb{R}^{N+5} \to \mathbb{R}$$

trained to minimise the MSE with L2-regularization where, as defined above, N is the number of base processes. The model was

Table 1: Train and validation MSE of the down-and-in barrier pricing model.

N	Train MSE	Valid. MSE
0	0.0108	0.0113
1	0.4014	0.4049
5	0.0369	0.0446
10	0.0139	0.0172
15	0.0106	0.0134
20	0.00889	0.0111

trained using 50,000 samples. To improve model generalisation, we follow [23] and start with a high learning rate which decays as training progresses. The neural network architecture consisted of 4 hidden layers of size 75 with ReLU activation function, and an output layer with a linear activation function. The training labels were generated using the QuantLib [4] library. The input to the model was a feature vector of the form

$$\boldsymbol{x}^i = \left(\mathcal{D}_{\mathcal{S}}^2 \left(\mathbb{X}^i, \mathbb{X}^{b_1} \right)^2, \cdots, \mathcal{D}_{\mathcal{S}}^2 \left(\mathbb{X}^i, \mathbb{X}^{b_N} \right)^2, S_0, K, r, B, T \right).$$

The elements of the feature vector were standardised (centred and scaled) before being passed as input to the neural network.

The model was trained using $N = 1, 5, 10, 15, 20$ base processes $\{\mathbb{X}^{b_i}\}_{i=1}^N$ chosen at random. We also trained the model in the standard supervised learning approach, i.e., mapping model and contract parameters to prices. This is not a path-wise approach and does not involve any distances. From now on we denote this approach by the model with $N = 0$. The model with $N = 0$ is a neural network with 4 hidden layers of size 30 with ReLU activation function and an output layer with linear activation function (architecture of this model was kept similar to [17]). Validation losses (without L2-regularization) of the models are provided in Table 1. As expected, the validation loss decreases as the number of base processes increases (excluding the case of $N = 0$) and then stabilises. This illustrates the bagging effect discussed in Section 4.

To test the model, the base processes need to be kept the same as the ones used to generate the feature vectors for training. The pricing model was tested by varying one model/contract parameter at a time and keeping all other parameters fixed. The tests were run using $N = 20$ base processes. A sample of the results is depicted in Figure 2. As is evident in Figure 2, the neural network prices approximate the QuantLib prices well.

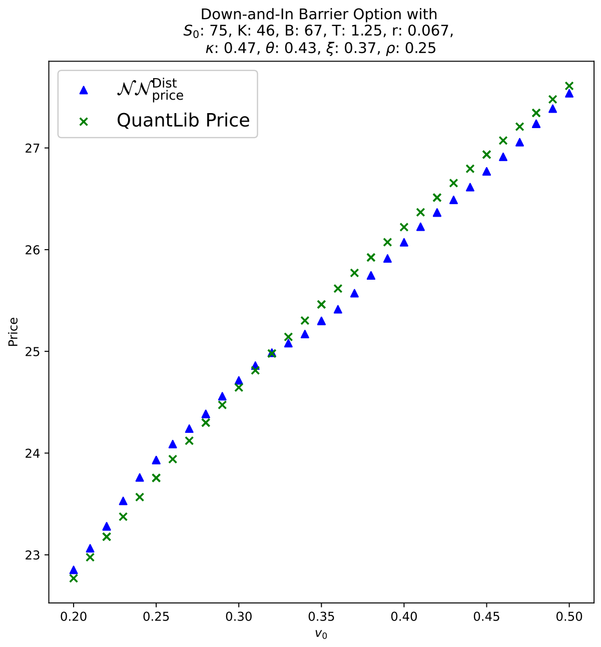
5.4 Pricing Rainbow Options in the Multi-Dimensional Black-Scholes Model

We focus on a best-call rainbow option [21]. A best-call rainbow option with maturity T has payoff

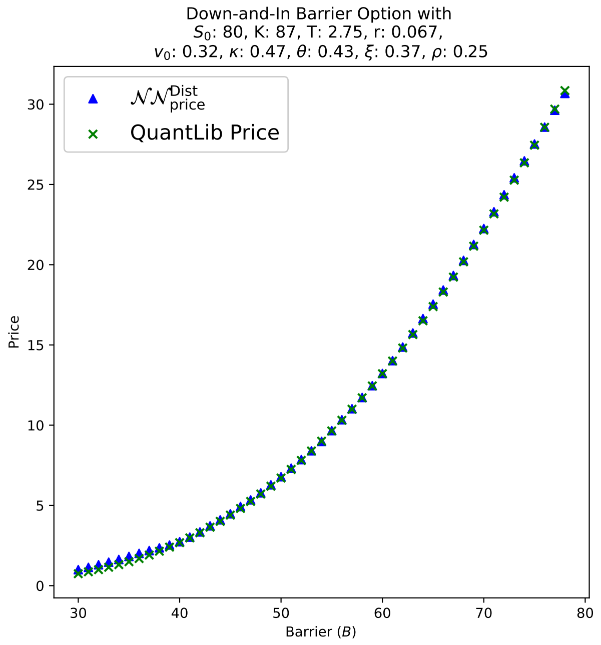
$$\left(\max \left(\frac{\boldsymbol{S}_T}{\boldsymbol{K}} \right) - K_B \right)^+$$

where $\boldsymbol{K}$ is a vector of normalising values and K_B is the basket strike. Vector operations in the payoff function are performed component-wise.

[4]QuantLib is an open-source quantitative finance library.



(a) Vary v_0 and keep other parameters fixed.



(b) Vary the barrier B and keep other parameters fixed.

Figure 2: Barrier option prices generated by our pricing model and QuantLib. $\mathcal{NN}^{\text{Dist}}_{\text{price}}$ denotes our pricing model.

Feature vectors are of the form

$$\boldsymbol{x}^i = \left(\mathcal{D}^2_{\mathcal{S}}\left(\mathbb{X}^i, \mathbb{X}^{b_1}\right)^2, \cdots, \mathcal{D}^2_{\mathcal{S}}\left(\mathbb{X}^i, \mathbb{X}^{b_N}\right)^2, \boldsymbol{S}_0, \boldsymbol{K}, K_B, r, T\right).$$

and the squared distances were computed using $\mathcal{NN}^{\text{MGBM}}_{\mathcal{D}^2}$. We used 3-underlyings and sampled model and contract parameters as follows:

- $\boldsymbol{S}_0 \sim \mathcal{U}^3_{\text{disc}}(50, 80)$
- $\boldsymbol{K} \sim \mathcal{U}^3_{\text{disc}}(30, 110)$
- $K_B \sim \mathcal{U}(0.25, 1)$ (rounded to 5 decimal places)
- $r \sim \mathcal{U}(0.01, 0.1)$ (rounded to 5 decimal places)
- $T \sim \mathcal{U}_{\text{disc}}(1, 5)$
- $\boldsymbol{\sigma} \sim \mathcal{U}^3(0.2, 0.8)$
- $\rho \sim \mathcal{U}(0.05, 0.95)$.

The target values (derivative prices) were computed using Monte Carlo. The number of simulations was set to 20,000 and the analytical solution of the SDE was used to generate $\boldsymbol{S}_T$. The neural network consisted of 4 hidden layers, each of dimension 75 and with ReLU activation function. An output layer with a linear activation function was used. The model was trained using a dataset of 50,000 samples. Features were centred and scaled before being used as input to the neural network. The neural network achieved a training loss of 3.43×10^{-4} and a validation loss of 3.98×10^{-4}. Figure 3 compares the price obtained by the model and by Monte Carlo. The model approximates the Monte Carlo prices accurately.

Another neural network was trained using the standard supervised approach: $\sigma_1, \sigma_2, \sigma_3, \rho$ and contract parameters were mapped to derivative prices (as in the barrier option case, this neural network is denoted by $N = 0$). This neural network had an identical architecture to the one in the barrier option experiments. The validation loss of the neural network with $N = 0$ was 5.07×10^{-4} as opposed to a validation loss of 3.98×10^{-4} obtained when $N = 20$.

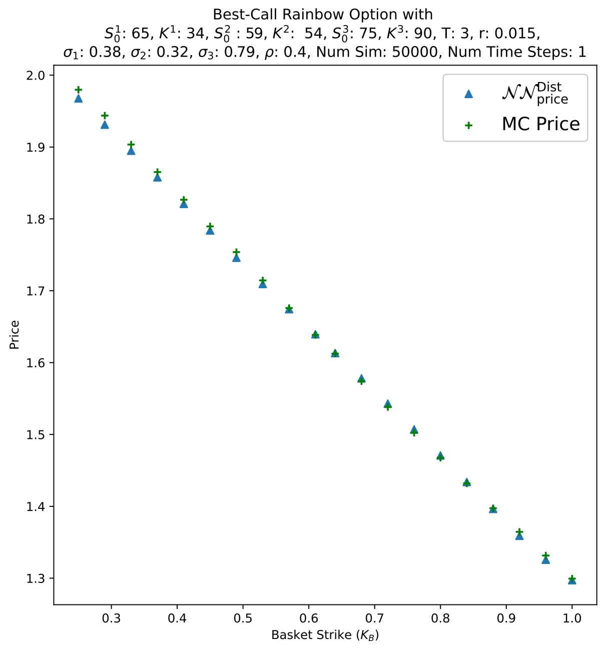
5.5 Pricing Autocallables in the Multi-Dimensional Black-Scholes Model

An autocallable with memory is a path-dependent basket derivative. The derivative pays a stream of coupons at fixing dates $\{T_i\}$ if the relative performance of the basket of assets is above the barrier level B_{coup}. If the basket performance is above the barrier B_{call} at some fixing date T_i, the derivative is terminated before maturity. Moreover, a capital guarantee barrier B_{gar} is present on the last fixing date T_N. At each fixing date T_i, the payoff [27] is

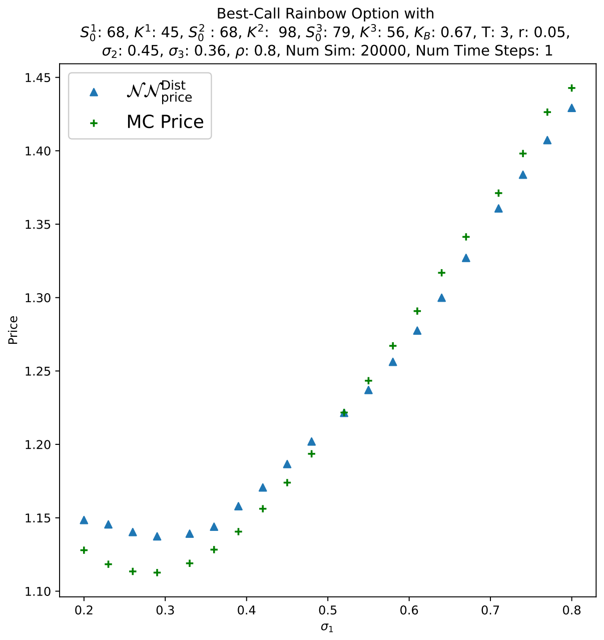
$$\Pi(T_i) := \mathbb{1}_{\{\tau > T_i\}}\left(\left[N_i + \sum_{j=1}^{i-1}\left(N_j - \Pi(T_j)\right)\right]\mathbb{1}_{\{P(T_i) \geq B_{\text{coup}}\}}\right.$$
$$\left. + \mathbb{1}_{\{i=N\}}\left(P(T_N) - 1\right)\mathbb{1}_{\{P(T_N) < B_{\text{gar}}\}}\right) + \mathbb{1}_{\{\tau = T_i\}}R$$

where N_i are the coupon notionals, R is the rebate, $P(t) := \min\left\{\frac{\boldsymbol{S}_t}{\boldsymbol{S}_{t_{\text{ref}}}}\right\}$ is the basket performance, and $\tau := \min\{T_i : P(T_i) \geq B_{\text{call}}\}$. We used the 3-dimensional Black-Scholes model to describe the underlying dynamics. Prices are computed using 5,000 Monte Carlo simulations. To compute the price using Monte Carlo, we compute the payoff at each fixing date T_i. These are then summed up and averaged, i.e.

$$\text{Price} = \frac{1}{O}\sum_{i=1}^{O}\sum_{j=1}^{N}\Pi\left(T_j, \boldsymbol{S}^i\right)$$



(a) Vary K_B and keep other parameters fixed.



(b) Vary the volatility of the first underlying (σ_1) and keep other parameters fixed.

Figure 3: Rainbow option prices generated by our pricing model and Monte Carlo. $\mathcal{NN}^{\text{Dist}}_{\text{price}}$ denotes our pricing model.

where $\boldsymbol{S}^i$ is the i^{th}-sample path and O is the number of sample paths simulated. The sample paths are generated using the solution of the SDE.

In our experiments, the fixing dates were kept constant at 1, 2, 3, 4, 5. We set each notional to 25, the rebate to 10, $\boldsymbol{S}_{\text{ref}} = (78, 60, 78)$, and $r = 0.015$. We sample the remaining model and contract parameters as follows:

- $\boldsymbol{S}_0 \sim \mathcal{U}^3_{\text{disc}}\,(80, 100)$
- $B_{\text{call}} \sim \mathcal{U}\,(0.7, 2)$ (rounded to 5 decimal places)
- $B_{\text{coup}} \sim \mathcal{U}\,(0.4, 0.8)$ (rounded to 5 decimal places)
- $B_{\text{gar}} \sim \mathcal{U}\,(0.2, 0.6)$ (rounded to 5 decimal places)
- $\boldsymbol{\sigma} \sim \mathcal{U}^3\,(0.2, 0.8)$ (rounded to 5 decimal places)
- $\rho \sim \mathcal{U}\,(0.05, 0.95)$ (rounded to 5 decimal places).

We train a neural network mapping features of the form

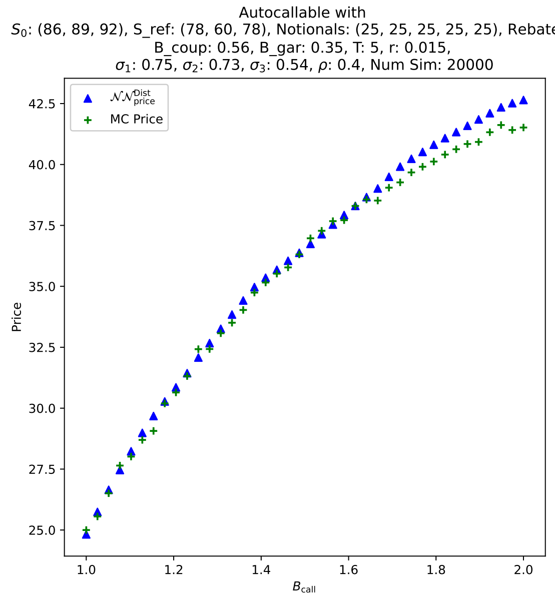
$$\boldsymbol{x}^i = \left(\mathcal{D}^2_{\mathcal{S}}\left(\mathbb{X}^i, \mathbb{X}^{b_1}\right)^2, \cdots, \mathcal{D}^2_{\mathcal{S}}\left(\mathbb{X}^i, \mathbb{X}^{b_N}\right)^2, \boldsymbol{S}_0, B_{\text{coup}}, B_{\text{call}}, B_{\text{gar}}\right)$$

to prices computed using Monte Carlo. The neural network contained 4 hidden layers of size 75 with ReLU activation function and an output layer with linear activation function. Layer normalisation [3] was used before the output layer. The features were centred and scaled before being used as input to the neural network. The pricing model was trained using 5,000 samples. It achieved a training MSE of 0.18 and a validation MSE of 0.773. The model was tested by varying one of the stochastic model parameters or contract parameters and keeping the others fixed. The average test MSE (one MSE per stochastic model/contract parameter varied) was 0.237. A subset of these results is depicted in Figure 4.

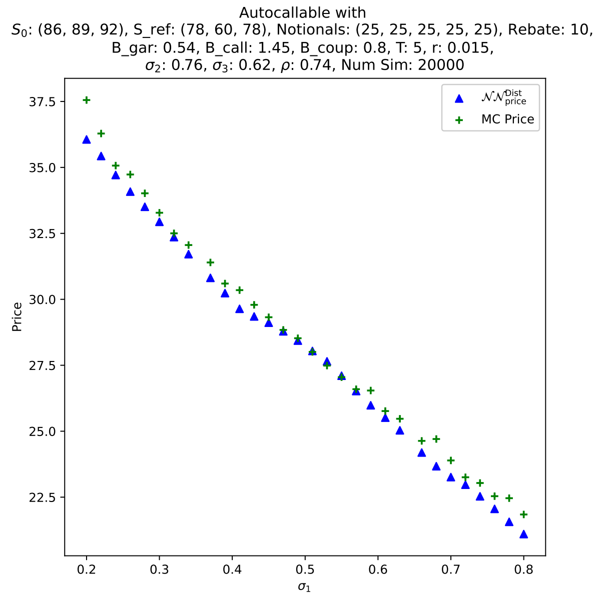
5.6 Run-Time of Our Pricing Model

A major benefit of our model compared with Monte Carlo simulations is the computation time. To price down-and-in barrier options using Monte Carlo with 50,000 realisations and $100 \cdot 3$ time-steps per realisation, it took 2.86s ±131ms. In comparison, our model took 16.5ms ±1.16ms. In the case of rainbow options, we use the solution to the SDE, and therefore it is not computationally expensive. However, there is a significant speed-up with our pricing model if we discretise the SDE and simulate the trajectories along multiple time-steps. Using 50 time-steps and 20,000 simulations, pricing using Monte Carlo is 24 times slower than using our neural network. There is a significant difference in running time when pricing autocallables with our pricing model as opposed to Monte Carlo. Our model has an average run-time of 32.9ms whist Monte Carlo's is 37.9s using 200,000 sample paths. Our pricing model is 1,152 times faster than Monte Carlo.

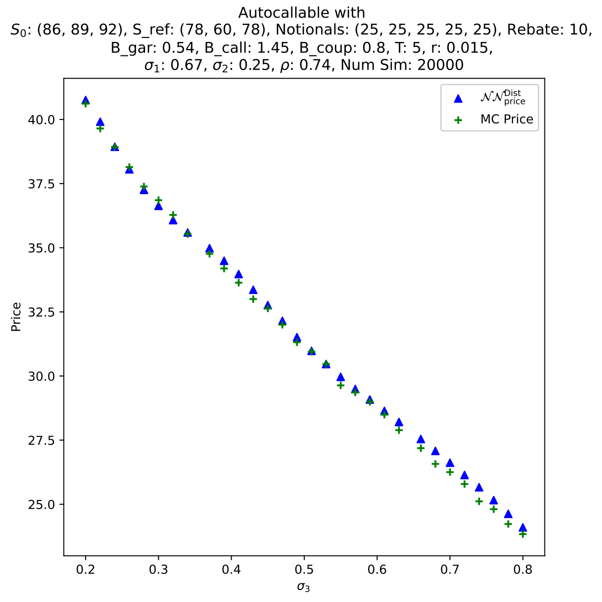
Speeding up the computation of the 2nd-order MMD using a pre-trained neural network is crucial to obtaining the improved run-time by our model as opposed to standard Monte Carlo. Computing the squared 2nd-order MMD between Heston models using the algorithm in [30] takes on average 6.45s, whilst using the pre-trained neural network results in an average run-time of 632μs. This is also the case for the MGBM model. The pre-trained squared 2nd-order MMD approximator has an average run-time of 789μs, whilst the algorithm in [30] has an average computation time of 5.77s.



(a) Vary B_{call} and keep other parameters fixed.



(b) Vary the volatility of the first underlying (σ_1) and keep other parameters fixed.



(c) Vary the volatility of the third underlying (σ_3) and keep other parameters fixed.

Figure 4: Autocallable prices generated by our pricing model and Monte Carlo. $\mathcal{NN}^{\text{Dist}}_{\text{price}}$ denotes our pricing model.

6 CONCLUSION

We have presented a method for reducing the run-time for computing the 2nd-order MMD for stochastic models. This approach uses a pre-trained neural network mapping stochastic model parameters to squared distances. The neural network approximates the square of the 2nd-order MMD to a high degree of accuracy whilst also significantly reducing the computation time needed to calculate these distances. Using the 2nd-order MMD we formulate a derivative pricing model. We significantly speed up the run-time of our pricing model by using the pre-trained MMD approximators. Our pricing model was tested on down-and-in European call barrier options, best-call rainbow options, and autocallables with memory. Our barrier and rainbow pricing models had a higher accuracy when compared with a neural network trained using standard supervised learning. We leave a comparison in the case of autocallables for future work.

We demonstrate that our model has a significantly faster runtime than Monte Carlo pricing. Since the empirical estimate of the 2$^{\text{nd}}$-order MMD is model-agnostic, future work could focus on speeding up the computation of the 2$^{\text{nd}}$-order MMD in a model-agnostic environment.

ACKNOWLEDGMENTS

This work was supported by the UK Engineering and Physical Sciences Research Council (EPSRC) under Grant [EP/T517963/1] and the UKRI Trusthworthy Autonomous Systems Hub under Grant [EP/V00784X/1]. The code and datasets supporting this article are not yet publicly available.

REFERENCES

[1] Afshine Amidi and Shervine Amidi. [n.d.]. A detailed example of how to generate your data in parallel with PyTorch. https://stanford.edu/~shervine/blog/pytorch-how-to-generate-data-parallel

[2] Imanol Perez Arribas. 2018. Derivative Pricing Using Signature Payoffs. (2018). arXiv:1809.09466

[3] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer Normalization. (2016). arXiv:1607.06450

[4] Daniel Bartl, Mathias Beiglböck, and Pammer Gudmund. 2021. The Wasserstein Space of Stochastic Processes. (2021). arXiv:2104.14245

[5] Fischer Black and Myron Scholes. 1973. The Pricing of Options and Corporate Liabilities. *Journal of Political Economy* 81, 3 (1973), 637–654. https://www.jstor.org/stable/1831029

[6] Leo Breiman. 1996. Bagging Predictors. *Machine Learning* 24 (1996), 123 – 140. https://doi.org/10.1007/BF00058655

[7] Hans Bühler, Blanka Horvath, Terry Lyons, Imanol Perez Arribas, and Ben Wood. 2020. A Data-Driven Market Simulator for Small Data Environments. (2020). arXiv:2006.14498

[8] Peter Bühlmann and Bin Yu. 2002. Analyzing bagging. *The Annals of Statistics* 30, 4 (2002), 927 – 961. https://doi.org/10.1214/aos/1031689014

[9] Mauricio Contreras, Alejandro Llanquihuén, and Marcelo Villena. 2016. On the Solution of the Multi-Asset Black-Scholes Model: Correlations, Eigenvalues and Geometry. *Journal of Mathematical Finance* 6, 4 (2016), 562–579. https://doi.org/10.4236/jmf.2016.64043

[10] John C. Cox, Jonathan E. Ingersoll Jr., and Stephen A. Ross. 1985. A Theory of the Term Structure of Interest Rates. *Econometrica* 53, 2 (1985), 385–407. https://doi.org/10.2307/1911242

[11] Paul Glasserman. 2004. *Monte Carlo Methods in Financial Engineering*. Springer, New York, NY. https://doi.org/10.1007/978-0-387-21617-1

[12] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org

[13] Arthur Gretton, Karsten M. Borgwardt, Malte J. Rasch, Bernhard Schölkopf, and Alexander Smola. 2012. A Kernel Two-Sample Test. *Journal Of Machine Learning Research* 13 (2012), 723–773.

[14] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. 2009. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition* (2nd ed.). Springer New York, NY. https://doi.org/10.1007/978-0-387-84858-7

[15] Calypso Herrera, Florian Krach, Pierre Ruyssen, and Josef Teichmann. 2021. Optimal Stopping via Randomized Neural Networks. (2021). arXiv:2104.13669

[16] Steven L. Heston. 1993. A Closed-Form Solution for Options with Stochastic Volatility with Applications to Bond and Currency Options. *The Review of Financial Studies* 6 (1993), 327–343. Issue 2. https://doi.org/10.1093/rfs/6.2.327

[17] Blanka Horvath, Aitor Muguruza, and Mehdi Tomas. 2019. Deep Learning Volatility. (2019). arXiv:1901.09647

[18] Brian Huge and Antoine Savine. 2020. Differential Machine Learning. (2020). arXiv:2005.02347

[19] Peter Jäckel. 2002. *Monte Carlo methods in finance*. John Wiley & Sons, Ltd., Chichester, England.

[20] Ho Chung Leon Law, Danica J. Sutherland, Dino Sejdinovic, and Seth Flaxman. 2018. Bayesian Approaches to Distribution Regression. In *International Conference on Artificial Intelligence and Statistics, AISTATS 2018, 9-11 April 2018, Playa Blanca, Lanzarote, Canary Islands, Spain (Proceedings of Machine Learning Research, Vol. 84)*. PMLR, 1167–1176. http://proceedings.mlr.press/v84/law18a.html

[21] Gordon Lee, Jöerg Kienitz, Nikolai Nowaczyk, and Qingxin Geng. 2021. Dynamically Controlled Kernel Estimation. (2021). https://doi.org/10.2139/ssrn.3829701

[22] Maud Lemercier, Cristopher Salvi, Theodoros Damoulas, Edwin V. Bonilla, and Terry Lyons. 2021. Distribution Regression for Sequential Data. In *Proceedings of the 24th International Conference on Artificial Intelligence and Statistics (AISTATS)* (San Diego, California, USA). PMLR, 3754–3762.

[23] Yuanzhi Li, Colin Wei, and Tengyu Ma. 2019. Towards Explaining the Regularization Effect of Initial Large Learning Rate in Training Neural Networks. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems* (Vancouver, Canada) *(NeurIPS 2019)*. Curran Associates Inc., Article 1047, 12 pages.

[24] Shuaiqiang Liu, Cornelis W. Oosterlee, and Sander M. Bohte. 2019. Pricing options and computing implied volatilities using neural networks. *Risks (Basel)* 7, 1 (2019), 16–.

[25] Ilya Loshchilov and Frank Hutter. 2019. Decoupled Weight Decay Regularization. In *7th International Conference on Learning Representations* (New Orleans, LA, USA) *(ICLR 2019)*.

[26] Terry Lyons, Sina Nejad, and Imanol Perez Arribas. 2019. Nonparametric pricing and hedging of exotic derivatives. (2019). arXiv:1905.00711

[27] Andrea Maran, Andrea Pallavicini, and Stefano Scoleri. 2021. Chebyshev Greeks: Smoothing Gamma without Bias. (2021). https://doi.org/10.2139/ssrn.3872744

[28] Johannes Ruf and Weiguan Wang. 2020. Neural networks for option pricing and hedging: a literature review. (2020). arXiv:1911.05620

[29] Cristopher Salvi, Thomas Cass, James Foster, Terry Lyons, and Weixin Yang. 2020. The Signature Kernel is the Solution of a Goursat PDE. (2020). arXiv:2006.14794

[30] Cristopher Salvi, Maud Lemercier, Chong Liu, Blanka Horvath, Theodoros Damoulas, and Terry Lyons. 2021. Higher Order Kernel Mean Embeddings to Capture Filtrations of Stochastic Processes. In *35th Conference on Neural Information Processing Systems (NeurIPS 2021)*.

[31] Antoine Savine. 2019. *Modern Computational Finance: AAD and Parallel Simulations*. John Wiley & Sons, Ltd.

[32] Terry J. Lyons. 1998. Differential equations driven by rough signals. *Revista Matemática Iberoamericana* 14 (1998), 215–310. Issue 2. http://eudml.org/doc/39555

[33] Terry J. Lyons. 2014. Rough paths, signatures and the modelling of functions on streams. In *Proceedings of the International Congress of Mathematicians* (Korea).

[34] Tomas Björk. 2004. *Arbitrage Theory in Continuous Time*. Oxford University Press. https://doi.org/DOI:10.1093/0199271267.001.0001

[35] Magnus Wiese, Lianjun Bai, Ben Wood, and Hans Buehler. 2019. Deep Hedging: Learning to Simulate Equity Option Markets. (2019). https://doi.org/10.2139/ssrn.3470756

[36] Magnus Wiese, Ben Wood, Alexandre Pachoud, Ralf Korn, Hans Buehler, Phillip Murray, and Lianjun Bai. 2021. Multi-Asset Spot and Option Market Simulation. (2021). arXiv:2112.06823