The Deep Parametric PDE Method: Application to Option Pricing*

Kathrin Glau† Linus Wunderlich†

December 14, 2020

Abstract

We propose the deep parametric PDE method to solve high-dimensional parametric partial differential equations. A single neural network approximates the solution of a whole family of PDEs after being trained without the need of sample solutions. As a practical application, we compute option prices in the multivariate Black-Scholes model. After a single training phase, the prices for different time, state and model parameters are available in milliseconds. We evaluate the accuracy in the price and a generalisation of the implied volatility with examples of up to 25 dimensions. A comparison with alternative machine learning approaches, confirms the effectiveness of the approach.

Keywords basket options, deep neural networks, high-dimensional problems, parametric option pricing, parametric partial differential equations, uncertainty quantification.

1 Introduction

Solving parametric partial differential equations (PDE) in high dimensions is a major challenge in many areas of science and engineering. In particular, it is of high importance for repetitive tasks in finance, such as real-time risk monitoring, uncertainty quantification and credit value adjustments. Frequent requests of derivative prices, require an efficient evaluation at different times, states and model parameters. Modern methods in option pricing based on Monte-Carlo are of limited efficiency. Other methods, such as based on PDEs or transforms, suffer from the curse of dimensionality.

Deep neural networks (DNN) offer efficient approximations with no curse of dimensionality [20, 28]. Thus, we apply them to numerically solve high-dimensional PDEs. While evaluating a neural network is fast, the training phase is computationally complex. To fully exploit the benefit of DNNs in the

*This work was funded by the Alan Turing Institute and EPSRC grant no. EP/T004738/1.

†School of Mathematical Sciences, Queen Mary University of London, Mile End Road, London E1 4NS, United Kingdom (k.glau@qmul.ac.uk, ł.wunderlich@qmul.ac.uk).

PDE context, we propose the *deep parametric PDE method* to learn the solution for all parameters simultaneously.

For an unsupervised training of the network, we first need a formulation of the parametric PDE as an appropriate optimisation problem. The least-squares formulation of a PDE yields a suitable loss function, as also used in the deep Galerkin method [48] for a fixed parameter set. The formulation naturally extends to the parametric setting and we propose a universal approach that allows for wide applications. As example we compute multi-asset option prices in the Black-Scholes model.[1] Further practical applications are pricing in stochastic volatility or jump-diffusion models, where Monte-Carlo methods are still state of the art for high dimensions, and examples beyond finance.

The deep parametric PDE method exhibits a natural offline-online decomposition. In the one-time offline phase, the neural network is trained. Then in the online phase, evaluating the solution for any time, state and parameter value is simply the matter of evaluating a single neural network.

1.1 Literature Review

There is a large research effort in medium- and high-dimensional option pricing. Classical methods include different variants of Monte-Carlo methods [16, 33], Fourier pricing [14], sparse grid integration [5, 19, 23] and low-rank approximations [17] (see also references therein). Also PDE-based solvers are considered, e.g. using an operator splitting [25], via expansion [46], wavelets [22] and radial basis function [43]. Monte-Carlo methods typically lack efficiency, while the other methods suffer from the curse of dimensionality in high-dimensional settings.

A possible remedy is the application of deep neural networks, which have drastically improved the field of artificial intelligence in the recent years [32]. The idea to use neural networks in finance was already researched in the 1990, e.g., in [27, 39] based on supervised learning. Also the use of shallow neural networks as a discretisation of PDEs using the Galerkin method was experimented with, e.g., in [4, 40]. A recent literature review [47] provides an overview over applications of neural networks in option pricing and hedging.

In general, the approaches can be classified as either unsupervised or supervised. An overview of unsupervised PDE-based approaches can be found in [50]. The deep BSDE method uses a reformulation of the problem as a backward stochastic differential equation (BSDE), which is then solved by a neural network, e.g,. [6, 10, 21, 26]. A method related to the proposed deep parametric PDE method is the deep Galerkin method introduced in [48]. Among the applications considered are the partial integro-differential equations and Hamilton-Jacobi-Bellman equations [2, 3] as well as general Stokes equations [34]. Physics-informed neural networks use a similar approach [11, 42, 45]. To the best of

[1] We provide an implementation of the deep parametric PDE method for two assets at `https://github.com/LWunderlich/DeepPDE/blob/main/TwoAssetsExample/DeepParametricPDEExample.ipynb`

our knowledge, no approach to solve general parametric PDEs directly using a neural network has been published yet.

An alternative approach to directly solving a PDE with neural networks is applying supervised learning to sample data, e.g., [38]. The deep Kolmogorov method recently introduced in [8] applies a hybrid approach using supervised learning and SDE-based techniques. There, samples of the underlying stochastic process of a Kolmogorov PDE are used to train a neural network.

There have been developed supervised learning approaches to parametric PDE problems, unrelated to finance. Examples are to approximate a low-dimensional quantity of interest [29] or the whole solution mapping [9, 35].

1.2 Main Contribution

We summarise the main contributions of this article.

- We introduce the deep parametric PDE method to solve a family of PDEs with a single network training. After a one-time offline phase to train the network, the solution for different parameter values can be evaluated in milliseconds.
- With an application to multi-asset option pricing, we show the practical use of the method.
- We provide a general proof of convergence which includes non-smooth data, as given for the option pricing problem.
- To evaluate the performance of the deep parametric PDE method, we study the error in the option price and in the implied volatility for problems of up to 25 dimensions.

The article is structured as follows. In Section 2, we introduce the deep parametric PDE method for parabolic problems. We specify the formulation for option pricing in the multivariate Black-Scholes model. Also, we show convergence of the deep parametric PDE method under general assumptions. Further details for the option pricing problem, i.e., the parameter dependency and reference solvers, are given in Section 3. Details regarding the implementation, including feature-scaling and hyper-parameter tuning, are described in Section 4. We investigate numerical examples with four to 25 dimensional problems in Section 5. Finally Section 6 summarises and concludes the article.

2 The Deep Parametric PDE Method

We consider a parametric parabolic PDE on the domain $(0,T)\times\Omega$, where $\Omega \subset \mathbb{R}^d$ with $d \in \mathbb{N}_{>0}$ is bounded with a smooth boundary. For each parameter μ in the

compact parameter domain $\mathcal{P}$, we solve for $u(\cdot;\mu)$, such that

$$\partial_t u(t,x;\mu) + \mathcal{L}_x^\mu u(t,x;\mu) = f(t,x;\mu), \qquad (t,x) \in \mathcal{Q} = (0,T)\times\Omega, \tag{1a}$$

$$u(0,x;\mu) = g(x;\mu), \qquad x \in \Omega, \tag{1b}$$

$$u(t,x;\mu) = u_\Sigma(t,x;\mu), \qquad (t,x) \in \Sigma = (0,T)\times\partial\Omega. \tag{1c}$$

Here $\mathcal{L}_x^\mu$ is a strongly elliptic differential operator of second order operating on the state variable x. It is parametrised by $\mu \in \mathcal{P}$ and we assume a continuous parameter dependency. Also the parameter dependency on $f(\mu) \in L^2(\mathcal{Q})$, $g(\mu) \in L^2(\Omega)$ and $u_\Sigma(\mu) \in L^2(0,T,H^{1/2}(\Sigma))$ is assumed to be continuous. Note that we frequently abbreviate $f(\cdot;\mu)$ as $f(\mu)$. Solvability and uniqueness of the solution for each parameter is given by, e.g., [36, Chapter 4, Equation 15.38].

We consider standard Lebesgue and Sobolev spaces, as introduced in [37]. L^2 denotes the space of square integrable functions, in the case of $L^2(\mathcal{Q})$ mapping from $\mathcal{Q}$ to $\mathbb{R}$ and for $L^2(0,T,X)$ mapping from $(0,T)$ to the Hilbert space X. $H^1(\Omega)$ is the space of square-integrable functions on Ω, whose first weak derivatives are also square-integrable. Its trace space is $H^{1/2}(\partial\Omega)$.

2.1 Overview of the Deep Parametric PDE Method

We propose the deep parametric PDE method as an approximation of the solution u by a single neural network. After training the network, the approximate solution is given for all times, states and parameter values. In order to truly exploit the given structure, we determine the loss function $\mathcal{J}(u)$ purely by the PDE. A suitable approach is based on a least-squares formulation of the PDE. To ease the learning process, we transform the solution to an auxiliary one, which is of a similar magnitude throughout the domain. This auxiliary solution is then approximated using a deep neural network with the time, state and parameter variables as the input.

The complete procedure is summarized as follows:

- In a one-time offline phase, which is the computationally expensive part of the method, we train the neural network by minimising the PDE's residuals.
- In the online phase, the solution to the PDE problem for any time, state and parameter value is obtained by evaluating the neural network, which is computationally fast.

Key advantage of the approach is that with a single training, an approximate solution is available for all considered PDEs simultaneously. This is in contrast to state-of-the-art deep learning approaches to solve PDEs, where one training phase yields the solution of a single PDE. Including the parameters of the problem into the neural network thus enables us to fully exploit the potential of deep learning to approximate high dimensional functions. The investment in a computationally expensive training phase thus pays off, since it yields a fast solution for the complete family of PDEs.

2.2 Choice of the Loss Function

To approximate (1) by a neural network, we first need to define an appropriate minimisation problem. In the deep parametric PDE method, we use a least-squares formulation of the PDE. To account for the parameter-dependency, another integral over the parameter domain $\mathcal{P}$ is added.

For a given function $u\colon \mathcal{Q} \times \mathcal{P} \to \mathbb{R}$ of sufficient smoothness, we define the loss based on the PDE's residuals:

$$\mathcal{J}(u) = \mathcal{J}_{\text{int}}(u) + \mathcal{J}_{\text{ic}}(u) + c_{\text{bc}}\mathcal{J}_{\text{bc}}(u). \tag{2}$$

The interior residual is defined as

$$\mathcal{J}_{\text{int}}(u) = |\mathcal{Q} \times \mathcal{P}|^{-1} \int_{\mathcal{P}} \int_{\mathcal{Q}} \left(\partial_t u(t,x;\mu) + \mathcal{L}_x^\mu u(t,x;\mu) - f(t,x;\mu)\right)^2 \, \mathrm{d}(t,x) \, \mathrm{d}\mu,$$

with $|\mathcal{Q} \times \mathcal{P}|$ the size of the domain, and the initial residual as

$$\mathcal{J}_{\text{ic}}(u) = |\Omega \times \mathcal{P}|^{-1} \int_{\mathcal{P}} \int_{\Omega} \left(u(0,x;\mu) - g(x;\mu)\right)^2 \, \mathrm{d}x \, \mathrm{d}\mu.$$

The boundary residual

$$\mathcal{J}_{\text{bc}}(u) = |\Sigma \times \mathcal{P}|^{-1} \int_{\mathcal{P}} \|u(\mu) - u_\Sigma(\mu)\|^2_{H^{1/2}(\Sigma)} \mathrm{d}\mu$$

is weighted by a factor $c_{\text{bc}} \geq 0$, which we will choose as zero in our experiments for simplicity. Unlike approaches based on supervised learning of expensively computed samples (e.g., [38]), no samples are required as this approach is *unsupervised*.

The integrals are numerically evaluated by Monte-Carlo quadrature, which yields a similarity to mean squared error-residuals often used in machine learning:

$$\begin{aligned}
\mathcal{J}_{\text{int}}(u) &\approx \sum_{i=1}^{N} \left(\partial_t u\big(t^{(i)}, x^{(i)}; \mu^{(i)}\big) + \mathcal{L}_x^\mu u\big(t^{(i)}, x^{(i)}; \mu^{(i)}\big) - f\big(t^{(i)}, x^{(i)}; \mu^{(i)}\big)\right)^2 / N, \\
\mathcal{J}_{\text{ic}}(u) &\approx \sum_{i=1}^{N} \left(u\big(0, \hat{x}^{(i)}; \hat{\mu}^{(i)}\big) - g\big(\hat{x}^{(i)}; \hat{\mu}^{(i)}\big)\right)^2 / N,
\end{aligned} \tag{3}$$

where $(t^{(i)}, x^{(i)}, \mu^{(i)}) \in \mathcal{Q} \times \mathcal{P}$ and $(\hat{x}^{(i)}, \hat{\mu}^{(i)}) \in \Omega \times \mathcal{P}$ for $i = 1, \ldots, N$ are chosen randomly with a uniform distribution. In our experiment, we choose $N = 10,000$ and observed less accurate approximations with smaller values of N. Approximating the boundary residual $\mathcal{J}_{\text{bc}}$ would be more involved. A practical approach could be to replace the $H^{1/2}(\Sigma)$ norm, by the $L^2(\Sigma)$-norm and to perform the same Monte-Carlo quadrature. As our experiments show good results without the term, we omit it for simplicity.

2.3 Multivariate Option Pricing in the Black-Scholes model

We apply the deep parametric PDE method to an option pricing problem in the Black-Scholes model. Expressing the option price in logarithmic asset variables, $u(t, x; \mu)$ denotes the fair price of an option at time to maturity t for the asset prices $s_i = e^{x_i}$:

$$\begin{aligned} u(t, x; \mu) &= \text{Price}(T - t, e^x; \mu), \\ \text{Price}(\tau, s; \mu) &= e^{-r(T-\tau)}\, \mathbb{E}(G(S_T(\mu)) \,|\, S_\tau(\mu) = s), \end{aligned} \tag{4}$$

with d underlyings $S_\tau(\mu) = (S^1_\tau(\mu), \ldots, S^d_\tau(\mu))$ and the physical time $\tau = T - t$. G denotes the payoff function at maturity and the assets S are modelled by a multivariate geometric Brownian motion.

The parameter μ can describe model parameters as well as option parameters. In our setting the parameter vector μ contains the risk-free rate of return, volatilities and correlations, each with a smooth parameter dependency.

In the Black-Scholes model, the differential equation (1) is homogeneous, i.e., $f(t, x; \mu) = 0$ and the operator reads

$$\mathcal{L}^\mu_x u(t, x; \mu) = r u(t, x; \mu) - \sum_{i=1}^{d} \left(r - \frac{\sigma_i^2}{2} \right) \partial_{x_i} u(t, x; \mu) - \sum_{i,j=1}^{d} \frac{\rho_{ij} \sigma_i \sigma_j}{2}\, \partial_{x_i x_j} u(t, x; \mu),$$

with $r = r(\mu)$, $\sigma_i = \sigma_i(\mu)$ and $\rho_{ij} = \rho_{ij}(\mu)$ with $\rho_{ii} = 1$. We choose the domain as a hypercube: $\Omega = (x_{\min}, x_{\max})^d$. We note that the boundary of the domain exhibits less regularity than assumed in our original setting, but we do not expect any issues arising from this.

For our main experiments, we consider European basket call options with equal weights and fixed strike price K:

$$g(x) = G(e^x) = \left(\frac{1}{d} \sum_{i=1}^{d} e^{x_i} - K \right)_+ = \max \left\{ 0,\ \frac{1}{d} \sum_{i=1}^{d} e^{x_i} - K \right\}.$$

As typical for option pricing, the initial condition is not smooth. However, in this case it is still in $H^1(\Omega)$. Thanks to the smoothing property of parabolic PDEs, this is sufficient regularity for the approximation results shown in Section 2.6.

For different parts of the boundary $\partial\Omega$, the boundary values u_Σ could be in principle chosen as the average asset price or the solution of a lower-dimensional option pricing problem. Since our numerical results are well even without the term, we omit the extra computational effort which would be required.

2.4 Localisation of the PDE

We can decompose the option price in two parts: the *trivial no-arbitrage bound*, and the remaining time value of the option. The no-arbitrage bound equals the maximum of zero and the expected payoff of an auxiliary derivative. This derivative shares all specifications of the basket option, despite that the owner is

forced to execute at maturity. In the case of a single asset, this bound captures the asymptotic behaviour of the option. As a consequence, the remaining time value is bounded and thus well suited for its approximation by a neural network.

However, the no-arbitrage bound in general is not smooth, making it unsuitable for a transformation of the PDE. Instead, we consider a smooth approximation $\widehat{u}_\lambda \in C^\infty$. In case of put and call options, the non-smoothness stems from the maximum function, which can be approximated excellently by the softplus function. For European basked call option, we thus have

$$\widehat{u}_\lambda(t,x;\mu) = \frac{1}{\lambda}\log\left(1 + e^{\lambda\left(\frac{1}{d}\sum_{i=1}^d e^{x_i}/d - Ke^{-rt}\right)}\right), \quad \text{for } \lambda > 0. \tag{5}$$

Note that for $\lambda \to \infty$, we approach the no-arbitrage bound:

$$\lim_{\lambda\to\infty} \widehat{u}_\lambda(t,x;\mu) = \left(\sum_{i=1}^d e^{x_i}/d - Ke^{-rt}\right)_+.$$

The drawback when using large values of λ is that the second derivative can become too large. Therefore in practice, we use a medium value, $\lambda = 0.1$.

In the deep parametric PDE method, we are left to approximate the *residual value* $u(t,x;\mu) - \widehat{u}_\lambda(t,x;\mu)$ by a neural network.

2.5 Neural Network

We use a variant of highway networks [49] that proved successful in the approximation of PDEs in [48]. After an initial dense layer, several gated layers are applied and finally combined to a scalar output. Denoting the input variables $(t,x;\mu)$ as $h^0 \in \mathbb{R}^n$ with $n = 1 + d + n_\mu$, we have the first dense layer as

$$h^1 = \psi(W^0 h^0 + b^0) \in \mathbb{R}^m,$$

with $W^0 \in \mathbb{R}^{m\times n}, b^0 \in \mathbb{R}^m$ for m nodes in each layer. The activation function ψ is the element-wise application of a smooth function, in our case the hyperbolic tangent.

Then for $l = 1,\dots,L$ with L the number of layers, we have

$$h^{l+1} = (1 - g^l) \odot h^{l+1/2} + z^l \odot h^l \in \mathbb{R}^m,$$

with gates g^l and z^l which can pass h^l and the intermediate layer computation $h^{l+1/2}$. The operator $\odot$ denotes element-wise multiplication of vectors. The intermediate layer computation $h^{l+1/2}$ includes an additional gate r^l which can drop the information present in the previous layer,

$$h^{l+1/2} = \psi(U^{h,l} h^0 + W^{h,l}(h^l \odot r^l) + b^{H,l}) \in \mathbb{R}^m.$$

Each of the three gates have the same standard structure:

$$\begin{aligned} g^l &= \psi(U^{g,l} h^0 + W^{g,l} h^l + b^{g,l}),\\ r^l &= \psi(U^{r,l} h^0 + W^{r,l} h^l + b^{r,l}),\\ z^l &= \psi(U^{z,l} h^0 + W^{z,l} h^l + b^{z,l}). \end{aligned}$$

In all cases the weights and biases are of the same size $U^{*,l} \in \mathbb{R}^{m\times n}$, $W^{*,l} \in \mathbb{R}^{m\times m}$ and $b^{*,l} \in \mathbb{R}^m$ and are trainable parameters.

A final dense layer and adding the localisation yields the trial functions for the option price:

$$u^\theta_{\mathrm{DNN}}(t,x;\mu) = \widehat{u}_\lambda(t,x;\mu) + W^{L+1}h^{L+1} + b^{L+1},$$

with $W^{L+1} \in \mathbb{R}^{1\times m}$, $b^{L+1} \in \mathbb{R}$ and the vector θ collecting all trainable *network parameters* W^*, U^*, b^*. We seek the network parameters $\hat{\theta}$ which minimise the loss:

$$\hat{\theta} = \arg\min_\theta \mathcal{J}(u^\theta_{\mathrm{DNN}}). \tag{6}$$

This defines the deep parametric PDE solution:

$$u(t,x;\mu) \approx u_{\mathrm{DPDE}}(t,x;\mu) = u^{\widehat{\theta}}_{\mathrm{DNN}}(t,x;\mu).$$

The resulting function u_{DPDE} is a single neural network that approximates the true solution of the PDE at all times, states and parameter values simultaneously. With one training, we have solved the whole family of PDEs.

2.6 Approximation Properties

In [48], convergence of the deep Galerkin method is shown for fixed parameters and with smooth solutions. In option pricing, we often deal with non-smooth initial conditions, so we adapt the proof to this case. We assume that the optimisation problem (6) is solved correctly. Research on convergence of numerical optimisers for neural networks is an emerging field, having shown first results, see e.g., [7]. We expect future results to allow us to push our analysis further in this direction.

We show convergence in two steps: First, we establish the existence of a neural network that minimises the loss function up to any prescribed accuracy. Then, we prove that any approximation with a loss function smaller than $\varepsilon > 0$ has an L^2-error less than $c\varepsilon$.

We denote the set of single layer-neural networks as

$$\mathcal{C} = \bigcup_{m=1}^{\infty} \{\zeta : \zeta(t,x;\mu) = \sum_{j=1}^{m} \beta_j \psi(w_j^t t + w_j^{x\top} x + w_j^{\mu\top} \mu + b_j)\} \subset C^\infty(\mathcal{Q}\times\mathcal{P}).$$

Our notation of function spaces follows [36, 37], where $H^{r,s}(\mathcal{Q}) = L^2(0,T,H^r(\mathcal{Q}))\cap H^s(0,T,L^2(\mathcal{Q}))$ and H^r is the Sobolev space of order $r \in \mathbb{R}$, both being defined in [37, Chapter 1]. We also consider by C continuous functions and by C^∞ smooth functions.

We consider the case with active boundary conditions, i.e., $c_{\mathrm{bc}} = 1$. The boundary conditions need to be sufficiently regular. An example would be a sequence of lower-dimensional Black-Scholes solutions. Regularity in this case can be shown by iterating this process up to the case of a single asset. Also the initial and boundary conditions are required to continuously intersect, which results in the compatibility condition $g(\mu)|_{\partial\Omega} = u_\Sigma(\mu)|_{\{0\}\times\partial\Omega}$.

Theorem 1. *Let $g \in C(\mathcal{P}, H^1(\Omega))$ and $u_\Sigma \in C(\mathcal{P}, H^{3/2,1}(\Sigma))$ fulfil the compatibility condition $g(\mu)|_{\partial\Omega} = u_\Sigma(\mu)|_{\{0\}\times\partial\Omega}$ for each parameter $\mu \in \mathcal{P}$ and let $f \in C(\mathcal{P}, L^2(\mathcal{Q}))$. Also let the parameter-dependency of the differential operator $\mathcal{L}_x^\mu$ be continuous in the operator norm from $H^2(\Omega)$ to $L^2(\Omega)$.*

Then, for any $\varepsilon > 0$, there exists a function $u_{\mathrm{DNN}} \in \boldsymbol{\mathcal{C}}$, such that $\mathcal{J}(u_{\mathrm{DNN}}) < \varepsilon$.

Proof. We use $H^{2,1}(\mathcal{Q})$-regularity of parabolic PDEs as shown in [36, Chapter 4, Equation 15.39]. While this result holds for each $\mu \in \mathcal{P}$, the analyticity of the inversion of a linear operator (see e.g., [15, Corollary A.2]) shows integrability on the compact domain $\mathcal{P}$, i.e., $u \in L^2(\mathcal{P}, H^{2,1}(\mathcal{Q}))$. Although this is sufficient regularity to show the desired approximation property, we could not find a direct results on the approximability of functions in anisotropic Sobolev spaces by neural networks. Therefore, we first approximate the solution by a smooth function, which we then approximate by a neural network.

Using a density argument, we can show that there exists $u_{\mathrm{sm}} \in C^\infty(\mathcal{Q}\times\mathcal{P})$, such that

$$\|u - u_{\mathrm{sm}}\|^2_{L^2(\mathcal{P},H^{2,1}(\mathcal{Q}))} \leq \varepsilon.$$

Now we can use [24, Theorem 3], which shows that there exists a neural network $u_{\mathrm{DNN}} \in \boldsymbol{\mathcal{C}}$ with

$$\|u_{\mathrm{sm}} - u_{\mathrm{DNN}}\|^2_{H^2(\mathcal{Q}\times\mathcal{P})} \leq \varepsilon.$$

This implies a close approximation also of u:

$$\begin{aligned}\|u - u_{\mathrm{DNN}}\|^2_{L^2(\mathcal{P},H^{2,1}(\mathcal{Q}))} &\leq c\|u - u_{\mathrm{sm}}\|^2_{L^2(\mathcal{P},H^{2,1}(\mathcal{Q}))} + c\|u_{\mathrm{DNN}} - u_{\mathrm{sm}}\|^2_{L^2(\mathcal{P},H^{2,1}(\mathcal{Q}))}\\ &\leq c\varepsilon.\end{aligned}$$

Then the interior residual can be rewritten and estimated as

$$\|\partial_t u_{\mathrm{DNN}} + \mathcal{L}_x^\mu u_{\mathrm{DNN}} - f\|^2_{L^2(\mathcal{Q}\times\mathcal{P})} = \|\partial_t(u_{\mathrm{DNN}} - u) + \mathcal{L}_x^\mu(u_{\mathrm{DNN}} - u)\|^2_{L^2(\mathcal{Q}\times\mathcal{P})} \leq c\varepsilon.$$

Trace estimates imply

$$\begin{aligned}\|u_\Sigma - u_{\mathrm{DNN}}\|^2_{L^2(\mathcal{P},H^{1/2}(\Sigma))} = \|u - u_{\mathrm{DNN}}\|^2_{L^2(\mathcal{P},H^{1/2}(\Sigma))} \leq c\varepsilon,\\ \|g - u_{\mathrm{DNN}}\|^2_{L^2(\{0\}\times\Omega\times\mathcal{P})} = \|u - u_{\mathrm{DNN}}\|^2_{L^2(\{0\}\times\Omega\times\mathcal{P})} \leq c\varepsilon,\end{aligned}$$

which concludes $\mathcal{J}(u_{\mathrm{DNN}}) \leq c\varepsilon$. □

In the second step we show that for any smooth function with a small loss, the L^2-error is bounded. The proof is fairly standard and relies on the stability of the PDE, which holds uniformly in $\mathcal{P}$.

Theorem 2. *Let the assumptions of Theorem 1 hold. Additionally let the parameter-dependency of the differential operator $\mathcal{L}_x^\mu$ be continuous in the operator norm from $H^1(\Omega)$ to $H^{-1}(\Omega)$.*

Then, there exists a constant $c < \infty$, such that for all $u_{\mathrm{DNN}} \in C^\infty(\mathcal{Q}\times\mathcal{P})$ it holds

$$\|u - u_{\mathrm{DNN}}\|^2_{L^2(\mathcal{Q}\times\mathcal{P})} \leq c\,\mathcal{J}(u_{\mathrm{DNN}}).$$

Proof. First, we rewrite the loss function as an average over the parameter space

$$\begin{aligned}\mathcal{J}(u_{\mathrm{DNN}}) &= \int_{\mathcal{P}} h(\mu)\,\mathrm{d}\mu, \text{ where}\\ h(\mu) &= \|u_{\mathrm{DNN}}(\mu) + \mathcal{L}_x^\mu u_{\mathrm{DNN}}(\mu) - f(\mu)\|^2_{L^2(\mathcal{Q})} + \|u_{\mathrm{DNN}}(\mu) - u(\mu)\|^2_{L^2(\{0\}\times\Omega)}\\ &\quad + \|u_{\mathrm{DNN}}(\mu) - u(\mu)\|^2_{H^{1/2}(\Sigma)}.\end{aligned}$$

Note that as $u(\mu) \in H^{2,1}(\mathcal{Q})$, we have sufficient regularity to replace $g(\mu)$ and $u_\Sigma(\mu)$ by $u(\mu)$ in these integrals.

We consider the residual equation for the error $e(\mu) = u(\mu) - u_{\mathrm{DNN}}(\mu)$:

$$\begin{aligned}\partial_t e(\mu) + \mathcal{L}_x^\mu e(\mu) &= r(\mu), \quad \text{on } \mathcal{Q},\\ e(\mu) &= u(\mu) - u_{\mathrm{DNN}}(\mu), \quad \text{on } \{0\}\times\Omega,\\ e(\mu) &= u(\mu) - u_{\mathrm{DNN}}(\mu), \quad \text{on } \Sigma,\end{aligned}$$

where $r(\mu) = f(\mu) - \partial_t u_{\mathrm{DNN}}(\mu) - \mathcal{L}_x^\mu u_{\mathrm{DNN}}(\mu)$. Stability of parabolic PDEs as shown in [36, Chapter 4, Equation 15.38] yields

$$\begin{aligned}\|e(\mu)\|^2_{L^2(\mathcal{Q})} &\le c\|r(\mu)\|^2_{L^2(\mathcal{Q})} + c\|u(\mu) - u_{\mathrm{DNN}}(\mu)\|^2_{H^{1/2}(\Gamma)}\\ &\quad + c\|u(\mu) - u_{\mathrm{DNN}}(\mu)\|^2_{L^2(\{0\}\times\Omega)} = ch(\mu),\end{aligned}$$

where we have used $\|e(\mu)\|^2_{L^2(\mathcal{Q})} \le \|e(\mu)\|^2_{H^{1,0}(\mathcal{Q})}$ and $\|r(\mu)\|^2_{H^{-1,0}(\mathcal{Q})} \le \|r(\mu)\|^2_{L^2(\mathcal{Q})}$. Again, as the inversion of a linear operator is analytic, the constant is bounded over $\mathcal{P}$ and the right hand side is thus integrable, which yields:

$$\|u - u_{\mathrm{DNN}}\|^2_{L^2(\mathcal{Q}\times\mathcal{P})} = \int_{\mathcal{P}} \|u(\mu) - u_{\mathrm{DNN}}(\mu)\|^2_{L^2(\mathcal{Q})}\,\mathrm{d}\mu \le c\int_{\mathcal{P}} h(\mu)\,\mathrm{d}\mu = c\,\mathcal{J}(u_{\mathrm{DNN}}).$$

□

3 Details Regarding Option Pricing

In this section, we provide the precise parametrisation of the option pricing problem and the evaluation of reference pricers.

3.1 Parametrisation of the Option Pricing Problem

In this section, we discuss the dependence of the Black-Scholes model on the vector of parameters μ. The potential parameters of basket options in the Black Scholes model are the strike price K, the risk-free rate of return r, the volatilities σ_i and the correlations ρ_{ij}. We note that by rescaling the asset prices, we can easily transform the problem into an option pricing problem with a fixed strike price K. For this reason, we do not consider the strike price as a parameter, but fix it at 100. The risk-free rate of return r and the volatilities σ_i are each an entry of the parameter vector.

For $d > 2$ parametrising the correlation matrix $(\rho_{ij})_{i,j}$ is a non-trivial task as the resulting covariance matrix has to be symmetric and positive semi-definite. A naive approach would be to parametrise the covariance matrix based on its Cholesky factors, leading $d(d-1)/2$ parameters, which are difficult to interpret. Instead, we consider a parametrised model based on the practical approach suggested in [13], where a valid correlation matrix is computed from pairwise correlations. The inputs are $(d-1)$ independent pairwise correlations $\hat{\rho}_i = \rho_{i,i+1}$ and the missing entries are calculated by successive products: $\rho_{ij} = \rho_{ji} = \prod_{k=i}^{j-1} \hat{\rho}_k$, for $j > i$. Positive definiteness for $\hat{\rho}_i \in (-1, 1)$ is shown in [13] by providing a Cholesky decomposition. Even for noisy or polluted estimates, this approach yields a valid correlation matrix.

In summary, the parameter vector is given as

$$\mu = (r, \sigma_1, \ldots, \sigma_d, \hat{\rho}_1, \ldots, \hat{\rho}_{d-1}),$$

which yields a smooth parameter-dependency of the pricing problem (1) in the sense of Theorems 1 and 2. We note that for $\sigma_i > 0$ and $\hat{\rho}_i \in (-1, 1)$, the differential operator $\mathcal{L}_x^\mu$ is strongly elliptic in x, thus we assume

$$\mathcal{P} \subset \mathbb{R} \times (0, \infty)^d \times (-1, 1)^{d-1}.$$

With these $n_\mu = 2d$ parameters, the overall dimension for a European basket call option with d underlyings is $n = 3d + 1$.

3.2 Error Evaluation with the Implied Volatility for Basket Options

For single-asset options, the implied volatility is the standard measure to compare prices and accuracies over a range of different products. There is no unique extension of the concept to a multivariate case. Here, we propose a natural formulation of the implied volatility for basket options. Each arbitrage-free option price is mapped to a unique volatility and computationally it boils down to the univariate case.

Given the price c (either the exact price $u(t, x; \mu)$ or the approximated one $u_{\mathrm{DPDE}}(t, x; \mu)$), we define the implied volatility as $\sigma_{\mathrm{iv}} > 0$, such that

$$\mathrm{BS}\left(t, \sum_{i=1}^{d} e^{x_i}/d, r, \sigma_{\mathrm{iv}}, K\right) = c.$$

A value for the implied volatility can be computed for any value in between the two trivial no-arbitrage bounds $c > c_{\mathrm{lb}} = \left(\sum_{i=1}^{d} e^{x_i}/d - Ke^{-rt}\right)_+$ and $c < c_{\mathrm{ub}} = \sum_{i=1}^{d} e^{x_i}/d$. The implied volatility is a good measure for the relative accuracy, as its values are of a similar magnitude over all sizes of the asset prices. The proposed extension inherits this property.

3.3 Reference Pricers

To evaluate the performance of the deep parametric PDE method, we need to compare it to alternative pricers. These reference pricers may be more expensive to evaluate for many parameters, as they only serve for comparison.

While for vanilla European basket options, the Black-Scholes formula

$$c(t, x; \mu) = \mathrm{BS}(t, e^x, r, \sigma, K) = \Phi(d_1)e^x - \Phi(d_2)Ke^{-rt}, \tag{7}$$

with $d_1 = \frac{1}{\sigma\sqrt{t}}\left(x - \ln(K) + rt + \frac{\sigma^2 t}{2}\right)$ and $d_2 = d_1 - \sigma\sqrt{t}$, provides an explicit option price, this is no longer available for basket options. We present a reference pricer for basket options as well as an academic example of a basket option with an explicit solution.

3.3.1 Smoothing the Payoff of European Basket Call Options

We can evaluate the option price by integrating a smoothened payoff, as developed in [5] and extended in [44]. After a variable transformation, the d-dimensional problem of computing the option price is split in a one-dimensional and a smooth $(d-1)$-dimensional problem. The first part can be solved precisely and the second part is solved using Gauß-Hermite quadrature. For convenience of the reader, we recapitulate the key steps.

Decomposing the covariance matrix as outlined in [5, Lemma 3.1] yields λ_i and $(v_{i,j})_{ij}$, such that for independent $Y_i \in \mathcal{N}(0, \lambda_i^2)$ the stochastic process of the logarithmic prices is $\log(S_T^i(\mu)) = x_i + (r - \sigma_i^2/2)t + Y_1 + \sum_{j=2}^d v_{i,j}Y_j$, with x_i the logarithmic asset price at time-to-maturity t. Solving a conditional expectation for Y_1 given $Y_2, \ldots, Y_d$ first, yields the option price as a $(d-1)$-dimensional problem with a smooth payoff function:

$$c(t, x; \mu) = \mathbb{E}\left(\mathrm{BS}(1, h(Y_2, \ldots, Y_d), 0, \lambda_1, e^{-rt}K)\right),$$

where $h(Y_2, \ldots, Y_d) = \frac{1}{d}\sum_{i=1}^d e^{x_i - \sigma^2 t/2} e^{\sum_{j=2}^d v_{i,j}Y_j}$. As the function h is smooth and the dimension is reduced by one, we have a simpler problem to solve than (4). Particularly a Gauß-Hermite quadrature as proposed in [44] suits well and is used here as a reference pricer. Because still a high-dimensional integral needs to be computed for each call, we only use it to validate our approach and do not propose it as an alternative parametric solver.

3.3.2 Special Cases With an Explicit Solution

Inspired by [48], we consider basket options with a payoff based on the geometric mean, which allows for an analytical solution of the parametric option price. Instead of the algebraic mean, the geometric mean $(\prod_{i=1}^d e^{x_i})^{1/d}$ enters the payoff:

$$g(x) = \left(e^{\sum_{i=1}^d x_i/d} - K\right)_+. \tag{8}$$

The geometric mean of a multivariate geometric Brownian motion is a univariate stochastic process of the same type. Thus the high-dimensional problem is reduced to a single-asset pricing problem with a dividend. For further simplicity, we consider underlyings of equal correlations and volatilities in which case the total dimension of the parametric problem is $n = d + 4$. The solution in this special case is given as

$$u(x, t; \mu) = N(d_1)e^{-qt}e^{\bar{x}} - N(d_2)Ke^{-rt}, \tag{9}$$

with $d_1 = \frac{1}{\bar{\sigma}\sqrt{t}}\left(\bar{x} - \ln(K) + rt - qt + \frac{\bar{\sigma}^2}{2}t\right)$, $d_2 = d_1 - \bar{\sigma}\sqrt{t}$ and $\bar{x} = \sum_{i=1}^{d} x_i/d$, $\bar{\sigma}^2 = \sigma^2/d(1 + (d-1)\rho)$ and $q = \sigma^2/2 - \bar{\sigma}^2/2$. We also adapt the extension of the implied volatility for basket options as described in Section 3.2 to this case.

4 Implementational Details

We implement the proposed deep parametric PDE method in `python` using the machine-learning package `keras` [12], based on `tensorflow` [1] (the versions used are `python 3.6.3`, `keras 2.4.0` and `tensorflow 2.3.0`). The models are trained on the GPU nodes of the high performance computing cluster at Queen Mary, Apocrita [30]. The used GPUs are Nvidia Tesla K80 and V100. They are accessed using `CUDA 10.1.243`. The trained neural networks are evaluated on an end-user device, a 2015 MacBook Pro with a 2.7GHz dual-core CPU and 8GB RAM. In the following subsections, we provide details regarding the algorithms to optimise trainable parameters as well as hyper-parameters.

4.1 Minimising the Loss

The minimisation of the loss function with respect to the weights of the neural network is a highly non-linear non-convex optimisation problem. As such we solve it iteratively by a variant of a gradient-descent method. We use an early stopping criterion to determine when the numerical optimiser has converged. Where the loss has not improved after 50 epochs (with 10 batches each), optimisation is stopped and the weights with the minimal observed loss are used. For a good generalisation of the solution, we re-sample the points used in the discrete loss (3) after each batch. This results in a good approximation on average of the loss function by the discrete loss.

A common problem when optimising neural networks are vanishing gradients, where a node saturates and the derivative with respect to its weights becomes numerically zero. To avoid this from happening, the initial value of the weights is of a high importance [18]. Most standard initialisations of the weights in neural networks assume input to be normalised. An input of the order of 100, as for the asset prices, would lead to vanishing gradients and thus extremely slow training. Thus, we linearly transform the computational domain to $[-1, 1]$ for time, asset values and each parameter. All weights are initialised using the Glorot normal initializer [18].

4.2 Hyper-parameter Optimisation

In addition to the weights and biases collected in the vector θ that will be optimised, neural networks have hyper-parameters, such as the number of layers and the number of nodes per layer. When these are chosen arbitrarily, results cannot be expected to be optimal. We use `keras-tuner` [41] for a hyper-parameter optimisation based on a random search. We set the limits for the neural network to be between 2 and 10 layers with 10 to 110 nodes each. Different optimisers are tested, and the learning rate (i.e., the step-size of the optimiser) is set to be between 0.01 and 0.0001.

While the precise values differed in different situations, we found that 9 layers with 90 nodes each provide good results in all of our cases. As the optimiser we choose Adam [31] with a learning rate of 0.001.

5 Numerical Examples

In this section, we consider different option pricing problems with one to eight underlyings. We look at the error in the price and the implied volatility in different situations, but also look into the convergence of the Adam optimiser.

If not stated otherwise, we consider volatility values between 10% and 30%, pairwise correlations between 0.2 and 0.8 and risk-free rates of return from 10% up to 30%. The maximal time to maturity considered is 4 in all cases, with the minimal time of interest being 0.5. The strike price is fixed to 100, which is no limitation thanks to re-scaling properties. The values of interest for the underlying assets range from 25 to 150. To limit the truncation error, we consider a larger computational domain with asset prices between 21 and 460. For the parameters, the computational domain and the domain of interest coincide. Unless stated otherwise, we use the default parameter values $r = 20\%$, $\sigma_i = 20\%$ and $\hat{\rho}_i = 0.5$.

In all examples, we use the same network architecture with 9 layers and 90 nodes per layer. This has the key advantage for applications, that no extra hyper-parameter optimisation is required to apply the method to a new situation. The numerical results confirm the required robustness.

We start with the univariate case in order to validate the method against the explicit solution. Then, we consider basket options in different scenarios and compare it to the reference pricer introduced in Section 3.3 and alternative machine learning methods. To further explore the versatility of our proposed method, we also consider the geometric payoff and briefly discuss the computation of Greeks.

5.1 Single-Assets Call Options

As an initial study, we consider the case of European call options with one underlying as we can compare the solution to the Black-Scholes formula (7).

Convergence of the residual and the validation error computed on a fixed set of $10,000$ random points are shown in Figure 1. We see a clear improvement of

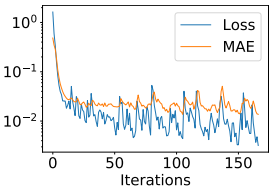


Figure 1: Convergence of loss and mean absolute error (MAE) over iteration steps of the optimisation. Only the convergence up to the optimal iteration is shown. 50 more epochs were computed but resulted in no improvement of the loss and are not shown.

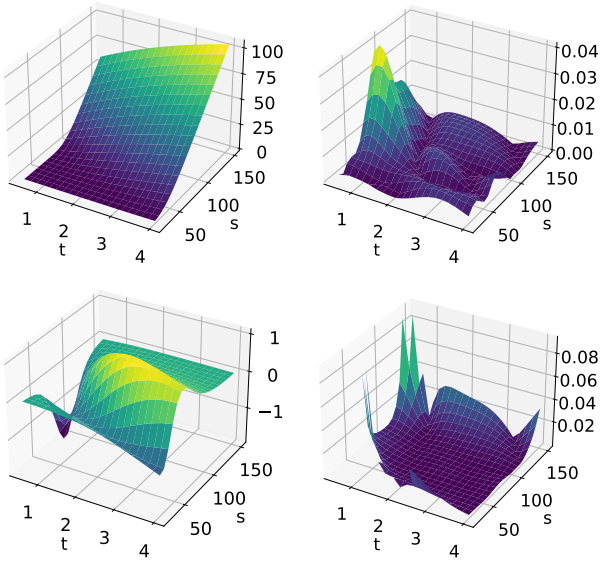


Figure 2: Top row: One-dimensional deep parametric PDE solution (left) and errors (right). Bottom row: Residual value of the solution (left) and relative error of the implied volatility (right). All values shown for fixed parameters.

the approximation over the iterations in both the loss function and the error, which shows the suitability of the considered loss functions and confirms the results of Theorem 2. In particular, we can see a clear relationship between the value of the loss function and the error on the validation set. This confirms that it is reasonable to use the loss function to detect convergence of the optimiser, as the validation error will not always be available in practice.

Figure 2 shows the approximation and the error over the domain of interest for fixed parameters. We see an overall well approximation of the option price with absolute errors considerably lower than 0.05. While the option price is between 0 and 100 in the domain of interest, the localisation captures most parts successfully. The residual value, which is approximated by the neural network, is betweens -1.5 to 1.

Also depicted in Figure 2 is the relative error of the implied volatility. As the implied volatility can be very sensitive to the option price, we only compute it where the difference between the exact option price and the lower trivial no-arbitrage bound c_{lb} (defined in Section 3.2) is larger than 0.005. In most parts of the domain, the implied volatility is accurate with a relative error less than 1%, overall less than 10%. We see that the relative error of the implied volatility peaks for small time to maturities and far in or out of the money, in which cases the implied volatility is too sensitive to be a reasonable error measure. After the successful results with one underlying, we next consider European basket options.

5.2 Basket Call Options

We consider basket options with between two and eight assets. First, we study the error at fixed parameter values and then focus on the parameter dependency. Finally, we compare the proposed deep parametric PDE method to alternative machine learning approaches.

5.2.1 Evaluation for Fixed Parameters

We first consider small baskets with two and three assets and then larger ones with five and eight assets. As the visualisation of high dimensional functions is challenging, we use different approaches in different dimensions. While in the examples of this section the parameters are fixed, we stress that the same solution can be used to evaluate any parameter in the domain of interest.

With two underlyings, we inspect the solution at the maximal time to maturity for fixed parameters, varying the price of the assets. Figure 3 shows the approximation and the error over the domain of interest. We see a good approximation with error values below 0.1 in the whole domain. Again, we note that the residual value of the option is of a small magnitude with values varying between -2 and 2.

For a more detailed evaluation of the solution quality, we also consider the implied volatility. As outlined in Section 3.2, we use a concept extending the standard implied volatility to the multi-asset case. When options are far in the

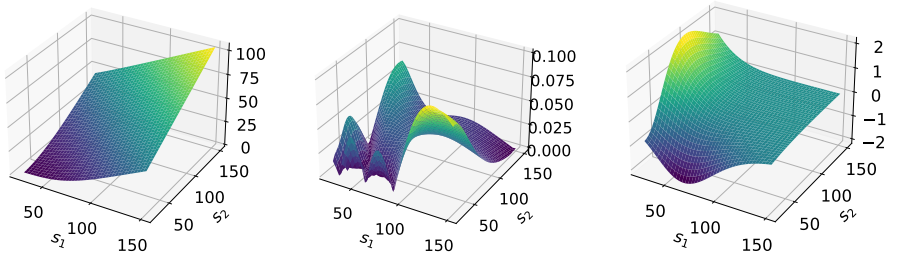
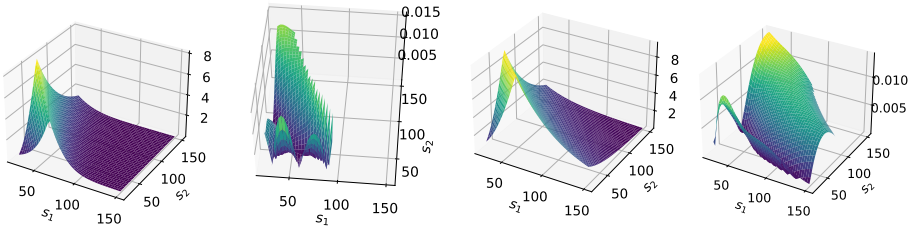


Figure 3: From left to right: Deep parametric PDE approximation, errors and approximated residual value for two underlyings. All pictures evaluated for $\sigma_1 = 0.1$ and $\sigma_2 = 0.3$ with $\rho_{12} = 0.5$ at maximal time to maturity $t = 4$.



(a) $\sigma_1 = 0.1$ and $\sigma_2 = 0.3$.

(b) $\sigma_1 = \sigma_2 = 0.3$

Figure 4: Left: Difference between the exact option price and the lower trivial no-arbitrage bound c_{lb}. Right: relative error of the implied volatility, computed where the difference is larger than 0.5.

money or out of the money, the implied volatility becomes extremely sensitive. Therefore, we consider implied volatilities on a smaller domain, excluding these extreme cases. In the left of Figure 4a, we see the difference between the exact option price and the trivial no-arbitrage bound c_{lb}. The difference is small far in- and far out of the money. We consider the implied volatility as an interesting measure, where the difference is larger than a threshold, which we choose as 0.5. The relative error of the implied volatility in this subdomain shown is on the right of Figure 4a. Figure 4b shows the same quantities for different parameter values. Thanks to the parametric approach, in both situations the solution is computed by evaluating the same neural network. With the second set of parameters, a smaller portion of the domain is far in the money. With both sets of parameters, the relative error in the implied volatility is smaller than 1.5% and significantly smaller in most of the domain of interest.

For three underlyings, we consider three-dimensional plots, varying each of the asset prices at final time and for fixed parameter values. In Figure 5, we show an excerpt of the error for $\sigma_1 = \sigma_3 = 0.1$, $\sigma_2 = 0.2$, $\rho_{1,2} = 0.2$ and $\rho_{2,3} = 0.5$. As seen in the left of the figure, in most parts of the domain of interest, the pricing error is less than 0.075. Only a small part of the domain exhibits errors of up to 0.22. In the right of the figure, we show the relative

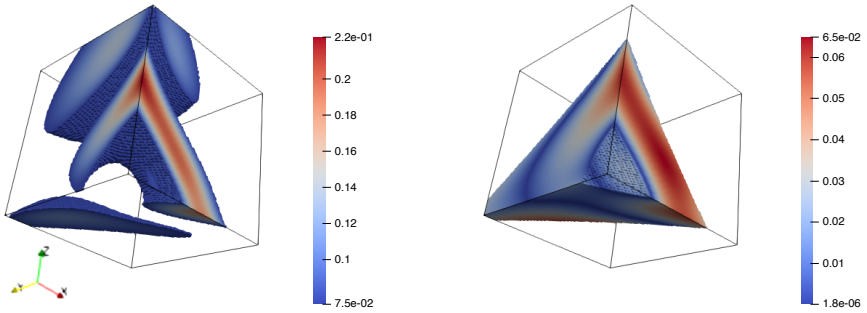


Figure 5: Different errors for three underlyings. Left: Pricing error, only shown where it is larger than 0.075. Right: Relative error in the volatility, evaluated where the difference to c_{lb} is larger than 0.5. Both images share the same perspective, facing the corner where $S_1 = S_2 = S_3 = 25$. The parameters are $\sigma_1 = \sigma_3 = 0.1$, $\sigma_2 = 0.2$, $\rho_{1,2} = 0.2$ and $\rho_{2,3} = 0.5$.

error of the implied volatility. We see an error well below 10% throughout the domain with parts below 1%.

In the cases of more than three underlyings, we cannot display the d asset prices any more. Instead, we sample over values with the same average asset price. In Figure 6, we display the average option price and the maximal error over a set of randomly selected asset prices with the same average for the case five and eight underlyings. We see a similar behaviour as in the lower dimensional cases. The maximal error remains well below 0.2. Note that with

eight underlyings, there are 16 parameters and the total dimensionality of the problem is 25.

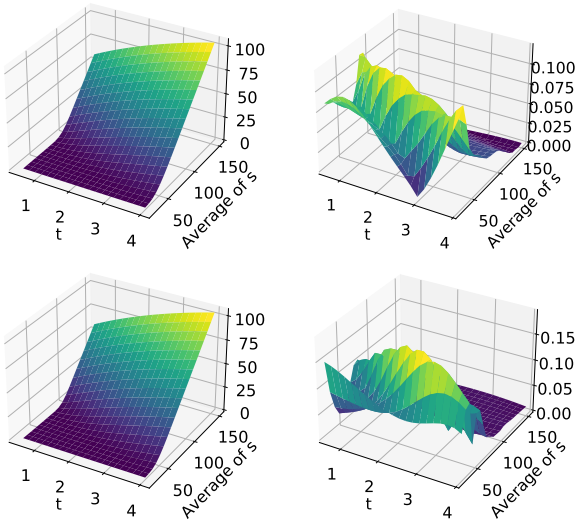


Figure 6: Averaged solution (left) and maximal errors (right) for five (top) and eight (bottom) underlyings.

5.2.2 Evaluation on the Whole Parameter Domain

So far, we have discussed the approximation error for arbitrary but fixed parameters. As the deep parametric PDE method is able to approximate the solution on the whole parameter domain, we now consider errors for varying parameters.

The parametric solution has up to 25 dimensions, posing challenges to visualising the error. As a first test, we consider $1,000$ random points from the whole domain of interest. These points have different time, state and parameter values, but with the deep parametric PDE method the option prices are quickly evaluated using a single neural network. In Figure 7, different scatter plots are shown, visualising the exact and approximated solution, the absolute error and the relative error in the implied volatility. Most absolute errors remain below 0.1. Isolated points can be seen with larger errors of up to 0.3. For the implied volatility a similar picture as before is seen, with most relative error values below 5% and all relative error values below 15%. As the overall approximation is good, but there are some outliers, we further investigate the parameter-dependency of the error for the different dimensionalities of the problem.

In Figure 8, the maximal error over different samples with the same mean asset price and parameter norm are shown at maximal time to maturity. We note that the parameter values are normalised, i.e., $\mathcal{P} = [-1, 1]^{n_\mu}$, which means that

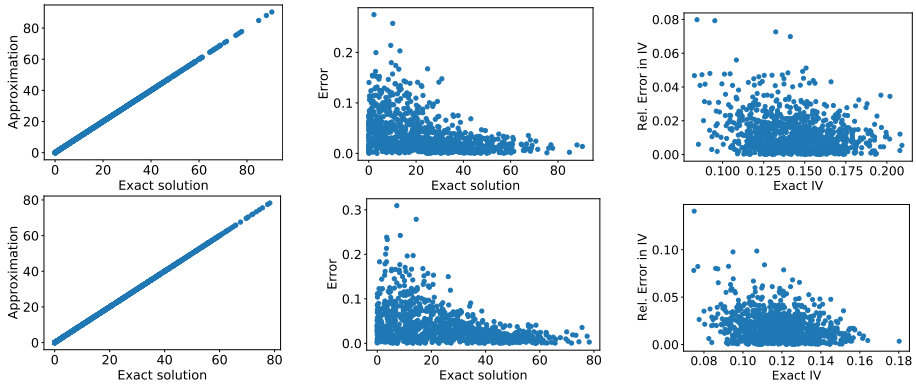


Figure 7: Scatter of prices evaluated for $1,000$ different tuples of time, asset prices and parameters chosen uniformly at random from the region of interest. Top: Five underlyings. Bottom: Eight underlyings. Left: Exact solution and approximation on the x- and the y-axis. Points close to the diagonal show a small error. Middle: Plot of the error against the exact solution. Right: Relative error of the implied volatility (IV), computed for option prices with a difference to c_{lb} of at least 0.5.

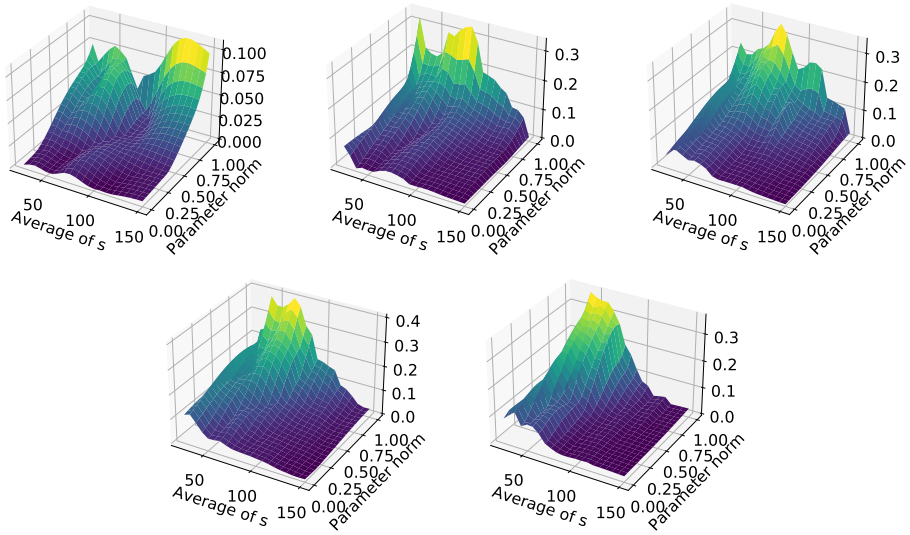


Figure 8: Maximal error depending on the parameter and the average asset price at maximal time to maturity. Top row from from left to right $d = 1, 2, 3$. Bottom row $d = 5, 8$.

the parameter norm measures the distance to the reference parameters $r = 0.2$, $\sigma_i = 0.2$ and $\hat{\rho}_i = 0.5$. We consider the maximal distance, i.e., the ℓ^∞-norm: $\|\mu\|_\infty = \max_{i\in\{1,\dots,n_\mu\}} |\mu_i|$. The mean asset price is given as $\bar{S} = \sum_{i=1}^d S_i/d$. The offline runtimes for the cases $d = 1, 2, 3, 5, 8$ are $6, 10, 15, 17$ and 58min. While runtimes grow with the dimension, they clearly do not exhibit a curse of dimensionality.

For each combination of $\bar{S}$ and $\|\mu\|_\infty$, $10,000$ samples are considered and the maximal error is shown in the figure. For eight underlyings, evaluating the reference pricer becomes very costly, so only $1,000$ samples are used. We see a clear increase of the error with the parameter norm, indicating that for parameters close to the center of the domain we have a higher accuracy. In the lower-dimensional cases, we see only little dependency on the average asset price, but the influence increases with more assets. In these cases, we see a peak of the error when the option is near the money. However, as seen in the previous section, this does not necessarily imply an increase in the relative error when considering the implied volatility. We note that the maximal error does not increase with the dimension of the problem, but remains stable for the considered basket options with a total dimension of up to 25.

5.2.3 Comparison to Alternative Machine Learning Methods

To fully understand the performance of the deep parametric PDE method, we comapare it to two other machine learning method: the deep BSDE solver [21] and the deep Galerkin method [48].

To compare with the deep BSDE solver, we use the code provided by the authors of [21] on GitHub. We adapt the example of the Black-Scholes equation with default risk to fit our setting. The only required changes are to change the payoff to a European basket call, to set the default risk to zero and to choose the number of underlying assets as 8. We keep the parameters as $T = 1$, $s_i = 100$, $r = 0.02$, $\sigma_i = 0.2$ and $\rho_{ij} = 0$ for $i, j = 1, \dots, 8$, $i \neq j$. Within $6,000$ iterations the method does not converge yet, so we choose $15,000$ iteration steps. For a fixed triple of time, state and parameters $(\hat{t}, \hat{x}; \hat{\mu})$, the deep BSDE method trains a neural network to compute the single option price $u(\hat{t}, \hat{x}; \hat{\mu})$.

As the specified parameters are not part of our previously used parameter set, we retrain the deep parametric PDE method on a slightly different domain with $r \in (0, 0.1)$ and $\hat{\rho} \in (-0.2, 0.2)$.

The deep Galerkin method is related to the deep parametric PDE method. It shares the least-squares formulation of the PDE, but trains a single solution $(t, x) \mapsto u(t, x; \hat{\mu})$ in the time-state-space for a fixed parameter $\hat{\mu}$. As an implementation of the deep Galerkin method, we therefore adapt the code of the deep parametric PDE method accordingly. The original deep Galerkin method as presented in [48], does not include a localisation which improves the accuracy. In order to examine the effect of solving for a whole parameter domain against solving for a fixed parameter set, we keep all other specifications equal. Particularly, we also use the localisation for the deep Galerkin method.

All values and the reference pricer are listed in Table 1 and compared to a

	Monte-Carlo	Reference pricer	Deep BSDE	Deep Galerkin	Deep Parametric PDE
value	3.9166	3.9217	3.8986	3.9335	3.9327
rel. error	–	0.130%	0.460%	0.431%	0.411%
offline runtime	–	–	–	28min for each $\hat{\mu}$	59min once $\forall \mu \in \mathcal{P}$
online runtime	6min 41s	1.34s	12min 7s	39.9ms	41.9ms

Table 1: Comparison of different methods to compute the value of an at the money European basket call option in the Black-Scholes model. Strike price 100 with eight underlying assets $s_i = 100$, $\sigma_i = 0.2$, $\rho_{ij} = 0$, $i, j = 1, \dots, 8, i \neq j$ and $r = 0.02$. Monte-Carlo solution with 10^9 samples as the exact solution (estimated standard deviation 0.000344).

Monte-Carlo solution with 10^9 samples. We see similar relative errors of less than 1% for all machine learning pricers, with the deep parametric PDE method slightly closer to the true solution. It is worth noting that also the reference pricer is only a bit more accurate, which confirms the efficiency of deep neural networks.

Table 1 also includes a comparison of runtimes. Reported runtimes for the offline-phases were measured on a GPU node on Queen Mary's cluster Apocrita, using an Nvidia Tesla V100. Online runtimes are measured on the end user device. Once the training is finished, the deep parametric PDE method provides the fastest solution to evaluate option prices with different parameters. The speed-up factor compared to the second fastest method, which is the reference pricer, is over 30. The evaluation time and accuracy of the deep Galerkin method and the deep parametric PDE method are both equivalent. For the cost of doubling the offline runtime, the deep parametric PDE method delivers the solution for the whole parameter set with no loss of accuracy. Already calling the solver for three different parameter sets, the deep parametric PDE method yields a total runtime gain of roughly half an hour.

While the offline run-time of the deep parametric PDE method is close to an hour, it only needs to be performed once. This can be done at idle times, i.e., at night. Afterwards, when the solution is required in real-time it is readily available.

5.3 Benchmark Case with Explicit Solution

To gain more insight into the performance in different settings, we the deep parametric PDE method on a second, more academic problem. The pricing problem with the geometric payoff, as described in Section 3.3.2, has an explicit solution, which makes it an interesting example.

Note that the trivial no-arbitrage bound is different than that of a standard

basket option. The expectation of the averaged asset prices is $e^{(r-\beta)t}e^{\sum_{i=1}^{d} x_i/d}$, with $\beta = \sigma^2/2(1-d^{-1})(1-\rho)$. Taking this into account in the localisation (5), it reads

$$\widehat{u}_\lambda(t,x;\mu) = \frac{1}{\lambda}\log\left(1+e^{\lambda\left(e^{-\beta t}e^{\sum_{i=1}^{d} x_i/d}-Ke^{-rt}\right)}\right).$$

The resulting approximation for two underlyings at maximal time to maturity is shown in Figure 9. We can see error levels below 0.05 in the domain of interest. Compared to the basket call options, we see a different profile of the approximative residual value.

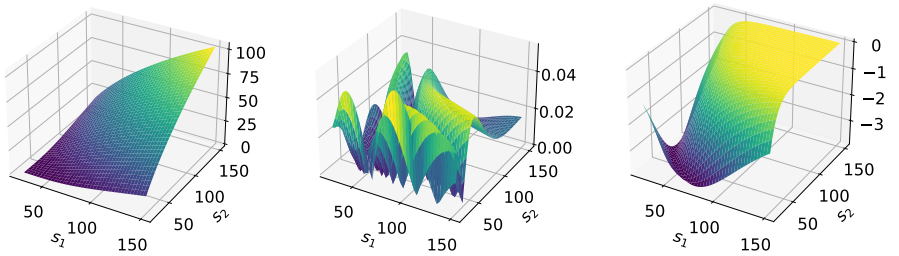


Figure 9: From left to right: Deep parametric PDE approximation, errors and approximated residual value for two underlyings. All pictures evaluated for $\sigma = 0.1$ at maximal time to maturity using the geometric payoff.

Figure 10 shows the maximal error depending on the parameter norm and the average asset price. In all cases, we see a peak of the maximal error near the money and for large parameter values. This means that for applications with a smaller parameter domain of interest, the method is significantly more accurate. In higher dimensions, the maximal error is only slightly higher.

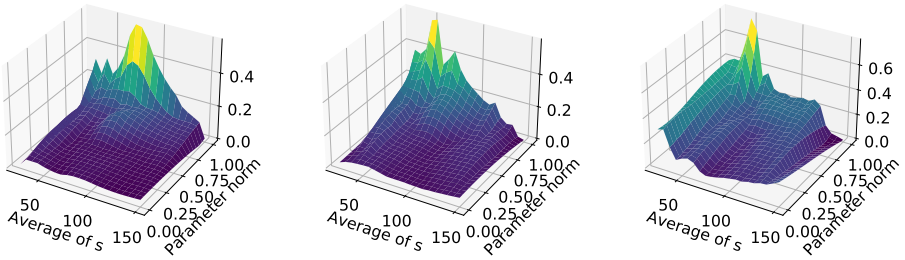


Figure 10: Maximal error using geometric payoff depending on the parameter and the average asset price at maximal time to maturity for different dimensions. From left to right $d = 3, 5, 8$.

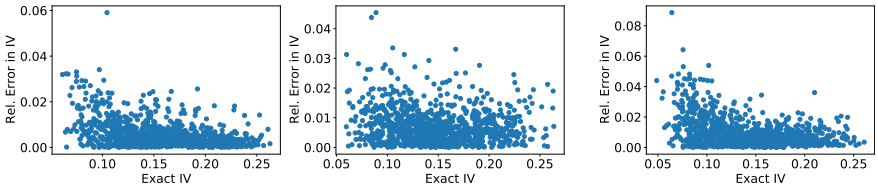


Figure 11: Relative implied volatility error for $d = 3, 5, 8$ (from left to right)

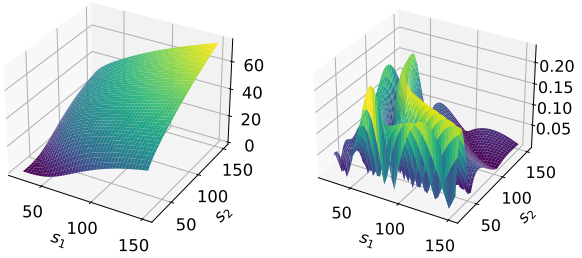


Figure 12: Derivative of the price with respect to the log-price of the first underlying (left) and its error (right) for $d = 2$.

To measure relative errors, we again consider the implied volatility as described in Section 3.2. The relative error for $1,000$ random values is shown in Figure 11. They show similar error magnitudes compared to the original setting of basket call options.

In summary, this shows flexibility of the deep parametric PDE method as we see robust results also in this setting.

5.4 Greeks and Sensitivities

Besides the approximated prices, the Greeks (i.e., derivatives of the price surface) are often of interest for applications, e.g., for hedging. Neural networks provide easy access to these derivatives. Exemplarily, Figure 12 shows the derivative of the price for two assets with respect to the log-price of the first underlying as well as its accuracy. We see a relative accuracy of below 1% throughout most of the domain.

Similarly, sensitivities with respect to the parameters can be computed. However, we observed significantly less accuracy. To improve the approximated sensitivities, it might be helpful to add the PDE of the sensitivity to the loss function. This approach is well-defined and will be subject of further research.

6 Conclusions

We have presented the deep parametric PDE method as an efficient solver for a whole family of parabolic problems. The use of deep neural networks allow us to solve high-dimensional problems accurately and fast. After a single training phase, the method quickly evaluates the solution at different time, state and parameter values.

Among the applications in science and engineering, we focussed on financial applications and presented results for pricing of basket options in the Black-Scholes model. We have seen a good approximation of the option price over a wide range of parameters. The runtimes in the online phase and the accuracy are relatively stable over different dimensions. In the offline phase, the runtimes grow, but do not exhibit a curse of dimensionality.

Compared to the reference pricer for the basket option with eight assets, the deep parametric PDE method has a speed-up factor of 30 in the evaluation phase with only a slight reduction of the accuracy. Compared to the deep Galerkin method, accuracy and evaluation time are equivalent with the benefit of having the solution readily available for all parameters. The additional cost in the offline phase pays off, for example in situations where the training can be done in idle times.

These good results motivate future research exploiting a key advantage of the proposed method: its structure allows for an easy adaptation to a multitude of other and more complex cases. For instance, the PDE can be replaced by partial integro differential equations or inequalities, thus opening up a large range of applications to different types of options and models, and examples beyond finance. The inherent offline-online decomposition shows its full potential in cases where the solution has to be evaluated for many parameters, an example application is the exposure calculation under uncertain parameters.

Acknowledgements

The authors thank Domagoj Demeterfi and Tobias Windisch for valuable discussions and Christian Pötz for valuable discussions and for providing the reference option pricer.

This research utilised Queen Mary's Apocrita HPC facility, supported by QMUL Research-IT. doi:10.5281/zenodo.438045

References

[1] M. Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from `https://tensorflow.org`.

[2] A. Al-Aradi, A. Correia, D. de Frietas Naiff, G. Jardim, and Y. Saporito. Applications of the deep Galerkin method to solving partial integro-differential and Hamilton-Jacobi-Bellman equations. *preprint*, 2019. arXiv:1912.01455.

[3] A. Al-Aradi, A. Correia, D. Naiff, G. Jardim, and Y. Saporito. Solving nonlinear and high-dimensional partial differential equations via deep learning. *preprint*, 2018. arXiv:1811.08782.

[4] E. Barucci, U. Cherubini, and L. Landi. Neural networks for contingent claim pricing via the Galerkin method. In H. Amman, B. Rustem, and A. Whinston, editors, *Computational Approaches to Economic Problems*, pages 127–141. Springer, 1997.

[5] C. Bayer, M. Siebenmorgen, and R. Tempone. Smoothing the payoff for efficient computation of basket option prices. *Quant. Finance*, 18(3):491–505, 2018.

[6] C. Beck, W. E, and A. Jentzen. Machine learning approximation algorithms for high-dimensional fully nonlinear partial differential equations and second-order backward stochastic differential equations. *J. Nonlinear Sci.*, pages 1563–1619, 2019.

[7] A. Bercher, L. Gonon, A. Jentzen, and D. Salimova. Weak error analysis for stochastic gradient descent optimization algorithms. *preprint*, 2020. https://arxiv.org/abs/2007.02723.

[8] J. Berner, M. Dablander, and P. Grohs. Numerically solving parametric families of high-dimensional Kolmogorov partial differentialequations via deep learning. In *Advances in Neural Information Processing Systems 33 pre-proceedings (NeurIPS 2020)*, 2020.

[9] K. Bhattacharya, B. Hosseini, N. B. Kovachki, and A. M. Stuart. Model reduction and neural networks for parametric PDEs. *preprint*, 2020. https://arxiv.org/abs/2005.03180.

[10] Q. Chan-Wai-Nam, J. Mikael, and X. Warin. Machine learning for semi linear PDEs. *J. Sci. Comput.*, 79(3):1667–1712, 2019.

[11] W. Chen, Q. Wang, J. Hesthaven, and C. Zhang. Physics-informed machine learning for reduced-order modeling of nonlinear problems. *Preprint submitted to J. Comput. Phys.*, 2020.

[12] F. Chollet et al. Keras. Software available from `https://keras.io`, 2015.

[13] P. Doust. Two useful techniques for financial modelling problems. *Appl. Math. Finance*, 17(3):201–210, 2010.

[14] E. Eberlein, K. Glau, and A. Papapantoleon. Analysis of Fourier transform valuation formulas and applications. *Appl. Math. Finance*, 17(3):211–240, 2010.

[15] J. Eldering. *Normally Hyperbolic Invariant Manifolds: The Noncompact Case*. Atlantis Press, 2013.

[16] M. B. Giles. Multilevel Monte Carlo methods. *Acta Numer.*, 24:259–328, 2015.

[17] K. Glau, D. Kressner, and F. Statti. Low-rank tensor approximation for Chebyshev interpolation in parametric option pricing. *SIAM J. Financial Math.*, 11(3):897–927, 2020.

[18] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In Y. W. Teh and M. Titterington, editors, *Proceedings of the13th International Conference on Artificial Intelligence and Statistics (AISTATS) 2010*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

[19] M. Griebel and M. Holtz. Dimension-wise integration of high-dimensional functions with applications to finance. *J. Complexity*, 26(5):455 – 489, 2010.

[20] P. Grohs, F. Hornung, A. Jentzen, and P. von Wurstemberger. A proof that artificial neural networks overcome the curse of dimensionality in the numerical approximation of Black-Scholes partial differential equations. *preprint*, 2018. https://arxiv.org/abs/1809.02362.

[21] J. Han, A. Jentzen, and W. E. Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences*, 115(34):8505–8510, 2018.

[22] N. Hilber, O. Reichmann, C. Schwab, and C. Winter. *Wavelet Methods*, pages 159–176. Springer, Berlin, 2013.

[23] M. Holtz. *Sparse Grid Quadrature in High Dimensions with Applications in Finance and Insurance*. Lecture Notes in Computational Science and Engineering. Springer, 2011.

[24] K. Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251 – 257, 1991.

[25] K. i. Hout and J. Toivanen. Application of operator splitting methods in finance. In R. Glowinski, S. J. Osher, and W. Yin, editors, *Splitting Methods in Communication, Imaging, Science, and Engineering*, pages 541–575. Springer International Publishing, Cham, 2016.

[26] C. Huré, H. Pham, and X. Warin. Deep backward schemes for high-dimensional nonlinear PDEs. *Math. Comp.*, 89:1547–1579, 2020.

[27] J. M. Hutchinson, A. W. Lo, and T. Poggio. A nonparametric approach to pricing and hedging derivative securities via learning networks. *The Journal of Finance*, 49(3):851–889, 1994.

[28] M. Hutzenthaler, A. Jentzen, T. Kruse, and T. A. Nguyen. A proof that rectified deep neural networks overcome the curse of dimensionality in the numerical approximation of semilinear heat equations. *SN Partial Differential Equations and Applications*, 1:10:1–34, 2020.

[29] Y. Khoo, J. Lu, and L. Ying. Solving parametric PDE problems with artificial neural networks. *European J. Appl. Math.*, page 1–15, 2020.

[30] T. King, S. Butcher, and L. Zalewski. Apocrita - high performance computing cluster for Queen Mary University of London. *Zenodo*, 2017.

[31] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *ICLR*, 2015.

[32] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[33] P. L'Ecuyer. Quasi-Monte Carlo methods with applications in finance. *Finance and Stochastics*, 13(3):307–349, 2009.

[34] J. Li, J. Yue, W. Zhang, and W. Duan. The deep learning Galerkin method for the general Stokes equations. *preprint*, 2020. https://arxiv.org/abs/2009.11701.

[35] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar. Fourier neural operator for parametric partial differential equations. *preprint*, 2020. https://arxiv.org/abs/2010.08895.

[36] J. L. Lions and E. Magenes. *Non-Homogeneous Boundary Value Problems and Applications*, volume II. Springer, 1972.

[37] J. L. Lions and E. Magenes. *Non-Homogeneous Boundary Value Problems and Applications*, volume I. Springer, 1972.

[38] S. Liu, C. Oosterlee, and S. Bohte. Pricing options and computing implied volatilities using neural networks. *Risks*, 7(1), 2019.

[39] M. Malliaris and L. Salchenberger. A neural network model for estimating option prices. *Applied Intelligence*, 3:193–206, 1993.

[40] A. Meade and A. Fernandez. The numerical solution of linear ordinary differential equations by feedforward neural networks. *Mathematical and Computer Modelling*, 19(12):1 – 25, 1994.

[41] T. O'Malley, E. Bursztein, J. Long, F. Chollet, H. Jin, L. Invernizzi, et al. Keras Tuner. Software available from `https://github.com/keras-team/keras-tuner`, 2019.

[42] G. Pang, L. Lu, and G. E. Karniadakis. fPINNs: Fractional physics-informed neural networks. *SIAM J. Sci. Comput.*, 41(4):A2603–A2626, 2019.

[43] U. Pettersson, E. Larsson, G. Marcusson, and J. Persson. Improved radial basis function methods for multi-dimensional option pricing. *J. Comput. Appl. Math.*, 222:82–93, 2008.

[44] C. Pötz. *Function approximation for option pricing and risk management.* PhD thesis, Queen Mary University of London, 2020.

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

[46] C. Reisinger and R. Wissmann. Finite difference methods for medium- and high-dimensional derivative pricing PDEs. In M. A. H. Dempster, J. Kanniainen, J. Keane, and E. Vynckier, editors, *High-Performance Computing in Finance Problems, Methods, and Solutions*, 2018.

[47] J. Ruf and W. Wang. Neural networks for option pricing and hedging: a literature review. *Journal of Computational Finance*, 24(1):1–46, 2020.

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

[49] R. K. Srivastava, K. Greff, and J. Schmidhuber. Training very deep networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, page 2377–2385, Cambridge, MA, USA, 2015. MIT Press.

[50] M. S. Vidales, D. Šiška, and L. Szpruch. Unbiased deep solvers for parametric PDEs. *preprint*, 2019. https://arxiv.org/pdf/1810.05094.pdf.