Applied Mathematics and Computation 432 (2022) 127355

Contents lists available at ScienceDirect

Applied Mathematics and Computation

journal homepage: www.elsevier.com/locate/amc

The deep parametric PDE method and applications to option pricing☆

Kathrin Glau, Linus Wunderlich*

School of Mathematical Sciences, Queen Mary University of London, Mile End Road, London E1 4NS, United Kingdom

ARTICLE INFO

Article history:
Received 7 July 2021
Revised 2 March 2022
Accepted 23 June 2022
Available online 2 July 2022

2020 MSC:
65M99
68T07
91G20

Keywords:
Basket options
Deep neural networks
High-dimensional problems
Greeks for multi-asset options
Parametric option pricing
Parametric partial differential equations

ABSTRACT

We propose, formalise and analyse the deep parametric PDE method to solve high-dimensional parametric partial differential equations with a focus on financial applications. 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 and Greeks in the multivariate Black–Scholes model as there is an urgent need for highly efficient methods. After a single training phase, the prices and sensitivities for different times, states and model parameters are available in milliseconds. Exploiting the PDE framework and incorporating a-priori knowledge of no-arbitrage bounds improves the performance significantly. We evaluate the accuracy in the price, the Greeks and the implied volatility with examples of up to 25 dimensions. A comparison with alternative machine learning methods confirms the effectiveness of the new approach and reveals advantages of the underlying PDE formulation.

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. A push for stricter financial regulations is currently driving a strong demand in highly efficient methods. There is a need for a high accuracy, stable evaluations of sensitivities as well as the possibility to easily evaluate different parameter values. Especially challenging are high-dimensional problems, which are currently often solved slowly and up a very low accuracy. Modern methods in option pricing based on Monte–Carlo are of limited efficiency and suffer from instability of the sensitivities. 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 [1–3]. The PDE offers a framework which allows for easy access to sensitivities. We combine the PDE framework and deep neural networks in such a way that the solution and sensitivities are accurate readily available on a whole range of parameters simultaneously. The solution along with its sensitivities inherit stability from the PDE framework. The ability to solve for all parameters

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

* Corresponding author.
E-mail addresses: K.Glau@qmul.ac.uk (K. Glau), L.Wunderlich@qmul.ac.uk (L. Wunderlich).

https://doi.org/10.1016/j.amc.2022.127355

at once is highly relevant, for example to handle uncertain parameters and risk analysis simulations involving stress tests. Comparisons to state-of-the-art machine learning methods demonstrate the competitiveness of the new approach in terms of accuracy, stability and online-runtimes. As such the proposed method offers an attractive and well-founded solution to urgent computational problems for instance in the financial industry.

The pricing problem in many cases takes the form of a parabolic PDE of the same structure as the heat equation, but it can be of a higher dimension. This connection between physics and finance is based on the use of the Brownian motion to model price processes and through the Feynman–Kac formula. The number of stocks involved in the option determines the dimension of the Brownian motion and consequently of the PDE. Each stock correspond to one space dimension of the PDE. Both the stock price model and the option are inherently parametric. For instance the volatility of the stock prices enters as model parameters in the Black–Scholes model and the strike price is a parameter for vanilla options. In the PDE the volatility enters the differential operator in a similar way as the heat conductivity, namely as a coefficient of the second derivative. The initial condition of the parabolic PDE is parametrised by the strike price. The precise value of the model parameters is uncertain and thus the solution needs to be evaluated for a whole model parameter range. Moreover, the solutions need also be often evaluated for different option parameters. Therefore, we consider the solution as a function from the time, state and parameter space. Instead of solving the PDE problem for fixed model and option parameters each time an evaluation is required, our approach delivers a solution for all time, state and parameter values at once. The solution is then readily available whenever an evaluation is required.

We propose the *deep parametric PDE method* as a solution to the outlined problems. We use a single neural network to approximate the solution for all time, state and parameters values simultaneously. Sensitivities are efficiently available through backpropagation. A standard approach to obtain the neural network solution is supervised learning. This would require a large amount of sample solutions, which would each require the solution of a high-dimensional problem themselves. Thus we would not solve the problem, but shift it to the creation of sample solutions. In contrast, an unsupervised approach overcomes the need for sample solutions. For this reason, we decide to train the neural network with an unsupervised approach. In order to do so, we reformulate the parametric parabolic PDE as an appropriate optimisation problem.

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 and sensitivities for any time, state and parameter value is simply the matter of evaluating a single neural network.

1.1. Novelty and main contribution

We provide the first algorithm solving high dimensional parametric PDEs on the whole parameter domain without sample solutions or simulations. We first formalise the solver of the parametric problem conveniently, provide a theoretical analysis and then investigate its performance numerically. The method extends the work of Sirignano and Spiliopoulos [4], whose main attention is on settings with fixed parameters. Our formalised extension covers the case of a parametric PDE with an univariate state space that is solved in their article without formalisation. The high-dimensional parametric problem presents an even higher challenge. Including prior knowledge on the solution allows us to achieve satisfying results even in this more complex setting. In contrast to the algorithm presented in Berner et al. [5], our algorithm solves the PDE directly and does so in an unsupervised manner. Particularly, we do not resort to Monte Carlo sampling via the related SDEs. Therefore our proposed approach is not restricted to Kolmogorov PDEs and does not suffer from simulation errors. The article of Berner et al. [5] has been developed independently of the first version of this article. In the present version of this article, we compare these two parametric approaches, which confirms the effectiveness of the deep parametric PDE method. In particular, we observe a higher stability for the deep parametric PDE method, yielding a higher accuracy of derivatives on the examples studied.

A major benefit of the method is the instant availability, accuracy and stability of the main sensitivities. The swift accessibility of sensitivities for high dimensional problems is one of the biggest challenges in risk management and hedging today. Current regulatory frameworks require banks to compute sensitivities on their portfolios on a regular basis, amounting to a high computational burden, whenever internal models are used. As outlined by the authors of Maran et al. [6] current industry standards suffer from instability and inaccuracy. In low dimensions, Chebyshev approaches solve this problem, but in higher dimensions such a solution is still missing. Once trained, the deep parametric PDE method provides immediate access to derivative prices and sensitivities for a whole range of parameters.

Especially, the first and second order space derivatives and the first order time derivative explicitly enter the optimisation during the training phase. This gives rise to an increased accuracy and stability of the Greeks delta, (cross) gamma and theta.

We summarise the main contributions of this article:

- We introduce the deep parametric PDE method to solve a family of high-dimensional PDEs with a single network training. After a one-time offline phase to train the network, the solution and its derivatives can be evaluated for different parameter values in milliseconds.
- To show the practical use in finance, we efficiently calculate multi-asset option prices and their Greeks.
- We significantly improve the accuracy by exploiting the PDE framework, incorporating prior knowledge of the solution.
- 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.
- Comparisons of the deep parametric PDE method with several alternative methods show its competitiveness.

1.2. Literature review

There is a large research effort in medium- and high-dimensional option pricing. Classical methods include different variants of Monte–Carlo methods [7,8], Fourier pricing [9], sparse grid integration [10–12] and low-rank approximations [13] (see also references therein). Also PDE-based solvers are considered, e.g., using an operator splitting [14], via expansion [15], wavelets [16] and radial basis function [17]. 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 [18]. The idea to use neural networks in finance was already researched in the 1990, e.g., in Hutchinson et al. [19], Malliaris and Salchenberger [20] 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 Barucci et al. [21], Meade and Fernandez [22]. A recent literature review [23] 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 Vidales et al. [24]. 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., [25–28]. A method related to the proposed deep parametric PDE method is the deep Galerkin method introduced in Sirignano and Spiliopoulos [4]. While their work includes an example with parameter-dependency, the considered dimensions are comparably low. In this article, we analyse the parametric case and investigate its performance in high dimensions. Among the applications considered are the partial integro-differential equations and Hamilton–Jacobi–Bellman equations [29,30] as well as general Stokes equations [31]. Physics-informed neural networks use a similar approach [32–34]. To the best of our knowledge, no demonstration of high-dimensional parametric PDEs solved 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., [35]. The deep Kolmogorov method recently (and independently of this article) introduced in Berner et al. [5] 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. Instead of as competing approaches, we see the deep parametric PDE method and the deep Kolmogorov method as continuations of the long-standing co-existence of PDE-based methods and SDE-based methods.

There have been developed supervised learning approaches to parametric PDE problems, unrelated to finance. Examples are to approximate a low-dimensional quantity of interest [36] or the whole solution mapping [37,38]. However, as supervised learning approaches, these approaches are dependent on the availability and the accuracy of a classical solver. Especially for high-dimensional problems, relying on a classical solver will negatively impact the performance. The deep parametric PDE method, as an unsupervised learning approach, performs unconstrained by any classical solvers.

Also beyond option pricing, the use of neural network in finance is an active research topic. Some examples are calibration [39,40], XVA [41,42] and exposure calculation [43].

1.3. Structure of the article

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. Incorporating prior knowledge of the solution in the PDE approach, we manage to boost the method's accuracy. We then show convergence of the deep parametric PDE method under general assumptions in Section 3. Further details for the option pricing problem, i.e., the parameter dependency and reference solvers, are given in Section 4. Details regarding the implementation, including feature-scaling and hyper-parameter tuning, are described in Section 5. We investigate numerical examples with four to 25 dimensional problems in Section 6. Finally Section 7 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., Lions and Magenes [44, Chapter 4, Equation 15.38].

We consider standard Lebesgue and Sobolev spaces, as introduced in Lions and Magenes [45]. 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. This allows for a natural extension to parametric PDEs, where the parameters are accounted for by an extra integral over the parameter domain $\mathcal{P}$. We emphasize a high advantage for the practical implementation as it allows us to use the same code structure as in the non-parametric case.

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

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

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., Liu et al. [35]), 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:

$$\mathcal{J}_{\text{int}}(u) \approx \sum_{i=1}^{N}\left(\partial_t u\left(t^{(i)},x^{(i)};\mu^{(i)}\right) + \mathcal{L}_x^\mu u\left(t^{(i)},x^{(i)};\mu^{(i)}\right) - f\left(t^{(i)},x^{(i)};\mu^{(i)}\right)\right)^2/N, \tag{2}$$

$$\mathcal{J}_{\text{ic}}(u) \approx \sum_{i=1}^{N}\left(u\left(0,\hat{x}^{(i)};\hat{\mu}^{(i)}\right) - g\left(\hat{x}^{(i)};\hat{\mu}^{(i)}\right)\right)^2/N,$$

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 $i = 1, \dots, N$ 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. In our experiments, we have chosen the computational domain larger than the domain of interest to lower the sensitivity to the boundary condition and we obtain good results without the term. Therefore, 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 $s_i = e^{x_i}$:

$$\begin{aligned} u(t, x; \mu) &= \text{Price}(T - t, e^x; \mu), \\ \text{Price}(\tau, s; \mu) &= e^{-r(T-\tau)} \mathbb{E}(G(S_T(\mu)) \,|\, S_\tau(\mu) = s), \end{aligned} \tag{3}$$

with d underlyings $S_\tau(\mu) = (S^1_\tau(\mu), \dots, S^d_\tau(\mu))$ 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 Eq. (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.

While we cover financial applications, we note that the Black–Scholes equation is a variation of the heat equation, which models the diffusive flow of heat through a heat-conductive material. It is connected to the physical models by the common use of the Wiener process, see e.g. [46]. The volatility corresponds to the heat-conductivity of the material, while the risk-free rate of return corresponds to a potential. The payoff function g corresponds to the initial heat distribution in the material and with a larger value of the volatility the heat diffuses more quickly.

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 3.

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. Neural network

We use a variant of highway networks [47] that proved successful in the approximation of PDEs in Sirignano and Spiliopoulos [4]. 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:

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

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 yields the trial functions for the option price:

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

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:

$$\hat\theta = \arg\min_\theta \mathcal{J}(u^\theta_{\mathrm{DNN}}). \tag{4}$$

This defines the deep parametric PDE solution:

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

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.

2.5. Initial validation and improvement of the method

With the basic approach outlined, we compare it to an alternative approach: the deep Kolmogorov method [5]. This allows us to validate the initial approach and to provide a benchmark for further enhancements of our method.

Like the method presented here, the deep Kolmogorov method uses a neural network to approximate the solution of a parametric PDE. It uses the Feynman–Kac formula to reformulate the PDE as a stochastic learning problem, which is then solved using a neural network. While this reformulation restricts the method to Kolmogorov PDEs (which includes the Black–Scholes PDE), the deep parametric PDE method can be applied to any parabolic PDE.

2.5.1. Initial comparison with the deep Kolmogorov method

We start the comparison with the single-asset example presented in Berner et al. [5, Chapter 1]. A single-asset European put option under the Black–Scholes model is considered with a fixed risk-free rate of return $r = 0.0$ and varying volatility $\sigma \in [0.1, 0.6]$, strike price $K \in [10, 12]$ and asset price $s \in [9, 10]$. The maximal time to maturity is $T = 1$. The relative error for an approximation $u_{\mathrm{approx.}}$ at a fixed time, asset price, strike price and volatility can be computed using the Black–Scholes formula u_{ex}:

$$\mathrm{err}_{\mathrm{rel}}(t,s,\sigma,K) = \frac{|u_{\mathrm{approx.}}(t,s,\sigma,K) - u_{\mathrm{ex}}(t,s,\sigma,K)|}{1 + |u_{\mathrm{ex}}(t,s,\sigma,K)|}. \tag{5}$$

The mean relative error is the average of the relative error for different values of the time, asset price, strike price and volatility. For the deep Kolmogorov method, it is given in Berner et al. [5, Table 1] as 0.11%. In contrast, our method yields a significantly larger error of 1.1% on a comparable setting. With a 10-times larger error, the method as introduced so far does not seem to be competitive yet.

As our approach is directly based on the PDE, we can use tried and tested methods from the numerics of PDEs. One particular advantage is the possibility to include prior knowledge by subtracting a trivial approximation. This concept has successfully been used in the context of reduced basis methods, e.g. in Cont et al. [48], Haasdonk et al. [49].

2.5.2. Incorporating prior knowledge about the solution

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 in the current setting. 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

$$\widehat{u}_\lambda(t,x;\mu) = \frac{1}{\lambda}\log\left(1 + e^{\lambda\left(\frac{1}{d}\sum_{i=1}^d e^{x_i}/d - Ke^{-rt}\right)}\right), \quad \text{for } \lambda > 0. \tag{6}$$

Table 1
Mean relative error for the price of European put options under the Black–Scholes model for one and three underlying assets. Comparison of the deep Kolmogorov method [5] and the deep parametric PDE method with and without the improvements from Section 2.5.2.

Method	Mean rel. error single asset	Mean rel. error 3 asset basket
Deep Kolmogorov	0.11%	0.39%
Deep Parametric PDE (unimproved)	1.1%	0.55%
Deep Parametric PDE (improved)	0.053%	0.11%

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, $\lambda = 0.1$.

We transform the PDE to solve for $v(t,x;\mu) = u - \widehat{u}_\lambda(t,x;\mu)$:

$$\begin{aligned} \partial_t v(t,x;\mu) + \mathcal{L}_x^\mu v(t,x;\mu) &= \tilde{f}(t,x;\mu), && (t,x) && \in \mathcal{Q} = (0,T)\times\Omega, \\ v(0,x;\mu) &= \tilde{g}(x;\mu), && x && \in \Omega, \\ v(t,x;\mu) &= v_\Sigma(t,x;\mu), && (t,x) && \in \Sigma = (0,T)\times\partial\Omega, \end{aligned}$$

with $\tilde{f} = f - \partial_t\widehat{u}_\lambda - \mathcal{L}_x^\mu\widehat{u}_\lambda$, $\tilde{g} = g - \widehat{u}_\lambda$ and $v_\Sigma = u_\Sigma - \widehat{u}_\lambda$. Using the adapted residual in (2), the parametric PDE method approximates for v. Recovering the solution of the original PDE is trivial:

$$u(t,x;\mu) = v(t,x;\mu) + \widehat{u}_\lambda(t,x;\mu)$$

2.5.3. Verification of the improvements

In this chapter we evaluate the gain in accuracy by the trivial no-arbitrage estimate and continue the comparison with the deep Kolmogorov method.

Evaluating the deep parametric PDE method with the presented improvement, the mean relative error drops to only 0.053%. This shows a clear improvement as previously the mean relative error was about 20-times larger. The results are summaried in Table 1. This confirms the practicability of the PDE approach, where prior knowledge (such as the trivial no-arbitrage bound) can easily be incorporated.

In particular, our method now yields a better result on this first example than the deep Kolmogorov method, halving the error. A confirmation of the improved accuracy of the deep parametric PDE method is the multi-asset result from Berner et al. [5, Table 2]. The mean relative error for a European basket put option with three underlyings is given as 0.39% for the deep Kolmogorov method. As in the univariate case, the incorporation of the no-arbitrage bound yields a lower error of 0.11%. Both comparisons are summarised in Table 1.

In summary, the use of a PDE approach allows us to easily incorporate prior knowledge. In the case of option pricing, we use a trivial estimate based on the no-arbitrage bound. This significantly improves the method and makes it more accurate than the deep Kolmogorov method on the examples presented in their article. In Section 6.5.1, we provide a more extensive comparison to the deep Kolmogorov method, which also includes the Greeks.

3. Approximation properties

In [4], 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 (4) is solved correctly. Research on convergence of numerical optimisers for neural networks is an emerging field, having shown first results, see, e.g., Bercher et al. [50]. 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 [44,45], where H^r is the Sobolev space of order $r \in \mathbb{R}$, and $H^{r,s}(\mathcal{Q}) = L^2(0,T,H^r(\mathcal{Q})) \cap H^s(0,T,L^2(\mathcal{Q}))$, both being defined in Lions and Magenes [45, Chapter 1]. We also consider by C continuous functions and by C^∞ smooth functions.

We consider the case with active boundary conditions, i.e., $c_{\text{bc}} = 1$. 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 $u_{\text{DNN}} \in \mathcal{C}$, such that $\mathcal{J}(u_{\text{DNN}}) < \varepsilon$.

Proof. We use $H^{2,1}(\mathcal{Q})$-regularity of parabolic PDEs as shown in Lions and Magenes [44, 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., Eldering [51, 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 $u_{\text{sm}} \in C^\infty(\mathcal{Q}\times\mathcal{P})$, such that

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

Now we can use Hornik [52, Theorem 3], which shows that there exists a neural network $u_{\text{DNN}} \in \mathcal{C}$ with

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

This implies a close approximation also of u:

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

Then the interior residual can be rewritten and estimated as

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

Trace estimates imply

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

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

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 $u_{\text{DNN}} \in C^\infty(\mathcal{Q}\times\mathcal{P})$ it holds

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

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

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

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 $e(\mu) = u(\mu) - u_{\text{DNN}}(\mu)$:

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

where $r(\mu) = f(\mu) - \partial_t u_{\text{DNN}}(\mu) - \mathcal{L}_x^\mu u_{\text{DNN}}(\mu)$. Stability of parabolic PDEs as shown in Lions and Magenes [44, Chapter 4, Equation 15.38] yields

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

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:

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

□

4. Details regarding option pricing

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

4.1. 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 Doust [53], 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 Doust [53] 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

$$\mu = (r, \sigma_1, \dots, \sigma_d, \hat\rho_1, \dots, \hat\rho_{d-1}),$$

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 $\mathcal{L}^\mu_x$ 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 $n = 3d + 1$.

4.2. 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 use a natural formulation of the implied volatility for basket options, as also used in Gulisashvili and Tankov [54]. 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

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

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. This extension inherits this property.

4.3. Reference pricers

In order 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

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

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.

4.3.1. Smoothing the payoff of European basket call options

We can evaluate the option price by integrating a smoothened payoff, as developed in Bayer et al. [10] and extended in Pötz [55]. 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 Bayer et al. [10, 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 $Y_2, \dots, Y_d$ first, yields the option price as a $(d-1)$-dimensional problem with a smooth payoff function:

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

where $h(Y_2, \dots, Y_d) = \frac{1}{d} \sum_{i=1}^d e^{x_i - \sigma^2 t/2} e^{\sum_{j=2}^d v_{i,j} Y_j}$. As the function h is smooth and the dimension is reduced by one, we have a simpler problem to solve than (3). Particularly a Gauß–Hermite quadrature as proposed in Pötz [55] 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.

4.3.2. Special cases with an explicit solution

Inspired by Sirignano and Spiliopoulos [4], 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:

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

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

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

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 4.2 to this case.

5. Implementational details

We implement the proposed deep parametric PDE method in `python` using the machine-learning package `keras` [56], based on `tensorflow` [57] (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 [58]. 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 2.7 GHz dual-core CPU and 8 GB RAM. In the following subsections, we provide details regarding the algorithms to optimise trainable parameters as well as hyper-parameters.

5.1. 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, the Adam optimiser [59]. 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 (2) 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 mitigate this undesired effect, the initial value of the weights is of a high importance [60]. 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 [60].

5.2. 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

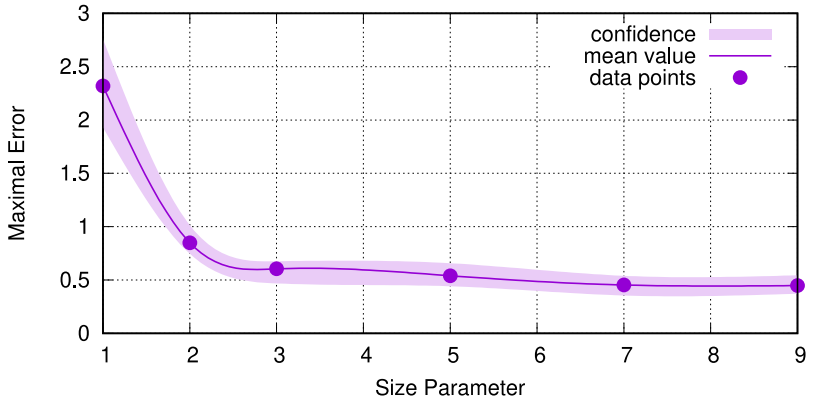


Fig. 1. Estimated maximal error on the domain of interest for neural networks of different size for $d = 5$ based on several independent trainings.

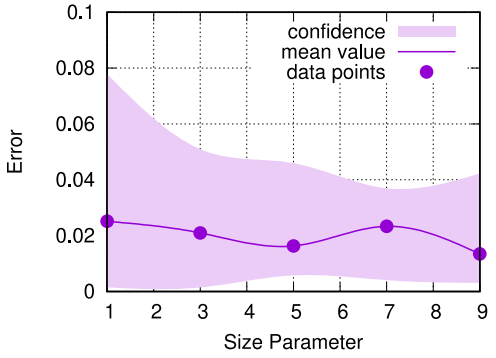


Fig. 2. Approximation error for a single price approximated by the DeepBSDE solver with neural networks of different sizes and $d = 5$, using the code provided by the authors of Han et al. [27]. For the network parameter L, a network with L hidden layers of $10L$ nodes each was considered.

cannot be expected to be optimal. We use `keras-tuner` [61] 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 [59] with a learning rate of 0.001.

We test the automatic hyper-parameter optimisation by a manual convergence test with regards to the size of the neural network. For $d = 5$ and a given value L, we consider a neural network with L hidden layers of $10L$ nodes each. For $L = 1, 2, 3, 5, 7, 9$ and ten independent trainings, we evaluate the maximal error. The average value of the maximal error as well as the largest and smallest values are shown in Fig. 1. We can see the error decreasing with a larger network, but mainly reaching a plateau soon. For $L \geq 3$, the error only decreases slightly. While the theoretical results would suggest a more significant decrease in the error, this is typical for approximation with neural networks. This can be seen in related studies in Shin et al. [62] (where the number of samples was varied) and in Fig. 2 using the DeepBSDE solver of Han et al. [27], where the error plateaus as well. We note that the error values of Figs. 1 and 2 cannot directly be compared, but show a similar trend. The observed plateau is due to limitations of the numerical optimiser and improvements of the optimisers will result in a higher accuracy. Nevertheless, the following numerical results demonstrate accurate and stable results, suitable for situations of practical interest.

6. 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. While the method is not restricted to eight underlyings, we note that the evaluation of the accuracy becomes challenging as the reference solver suffers from the curse of dimensionality. For this reason, the largest number of underlying presented in this article is eight.

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 1% up to 3%. 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

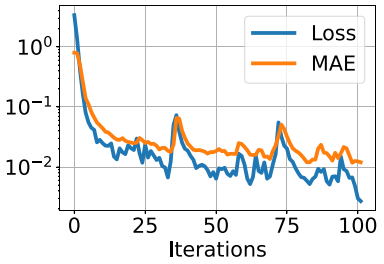


Fig. 3. Convergence of loss and mean absolute error (MAE) over iteration steps of the optimisation. Only the convergence up to the optimal iteration is shown. 50 more epochs were computed but resulted in no improvement of the loss and are not shown.

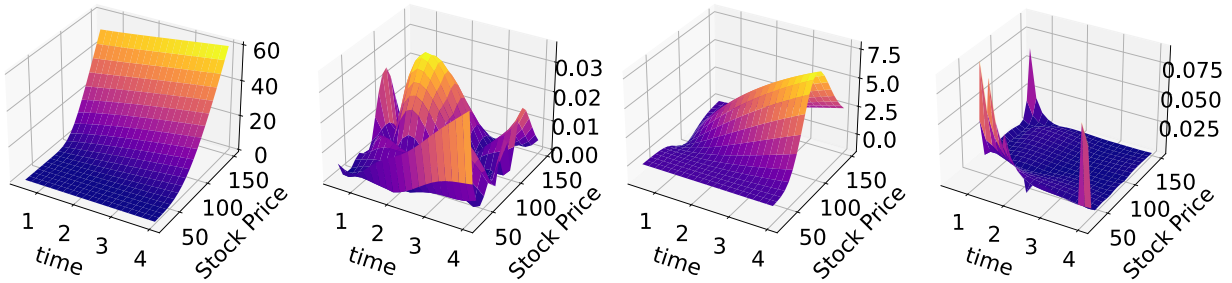


Fig. 4. One-dimensional deep parametric PDE solution (left) and errors (middle left), residual value of the solution (middle right) and relative error of the implied volatility (right). All values shown for fixed parameters.

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 $r = 2\%$, $\sigma_i = 20\%$ 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 4.3. To further explore the versatility of our proposed method, we also consider the geometric payoff and the computation of Greeks. Finally, we show the competitiveness of the deep parametric PDE method in a comparison with alternative methods.

6.1. 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 Fig. 3. We see a clear improvement of 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.

Fig. 4 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 60 in the domain of interest, the approximated no-arbitrage bound captures most parts successfully. The residual value, which is approximated by the neural network, is betweens -1.5 to 10. This highlights the importance of the improvement made in Section 2.5.2 as well as the accuracy of the neural network approximation.

For a more detailed investigation we consider the relative error of the implied volatility, also depicted in Fig. 4. As the implied volatility can be extremely 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 4.2) 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.

Table 2
CPU runtimes for the evaluation of the option prices during the online phase averaged over seven runs on the end-user laptop.

Nr evaluations	1 asset	2 assets	3 assets	5 assets	8 assets
1	47.2 ms	51.6 ms	46.4 ms	53.8 ms	50.2 ms
10	47.2 ms	54.3 ms	47.3 ms	51.7 ms	47.9 ms
100	60.5 ms	55.5 ms	55.1 ms	55.4 ms	54.5 ms
1000	125 ms	137 ms	134 ms	133 ms	129 ms

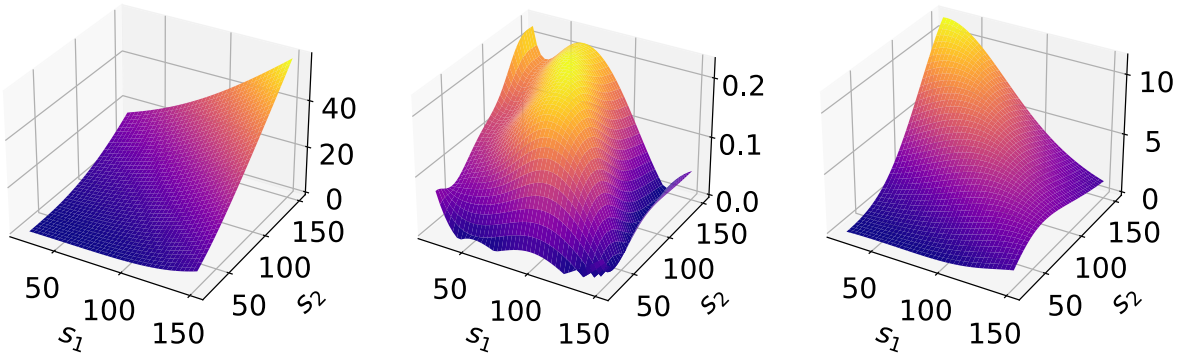


Fig. 5. From left to right: Deep parametric PDE approximation, errors and approximated residual value for two underlyings. All pictures evaluated for $\sigma_1 = 0.1$ and $\sigma_2 = 0.3$ with $\rho_{12} = 0.5$ at maximal time to maturity $t = 4$.

6.2. Basket call options

We consider basket options with between two and eight assets. After looking at runtimes during the training and evaluation phase, 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.

6.2.1. Runtime comparison

In Table 2, CPU runtimes for a growing number of underlying assets and evaluations are given. The runtimes were evaluated on the end-user laptop and were averaged over seven independent runs. We note that the runtimes stay stable with a growing dimension of the problem. This is a key advantage of neural network as unlike mesh-based numerical techniques for PDEs, the size of the problem only changes slightly with the dimension. Only the offline runtimes do grow with the dimension of the problem: for the cases $d = 1, 2, 3, 5, 8$ they are $6, 10, 15, 17$ and 58min, respectively (on the GPU-cluster). We note that they only grow mildly and do not exhibit a curse of dimensionality.

We also see that we can evaluate a large number of points simultaneously without a linear growth of the runtime. This is mainly due to the efficient implementation of tensorflow, which makes use of modern vector processors. This is of key importance for financial applications as to evaluate risk measures typically a large amount of option prices are required simultaneously.

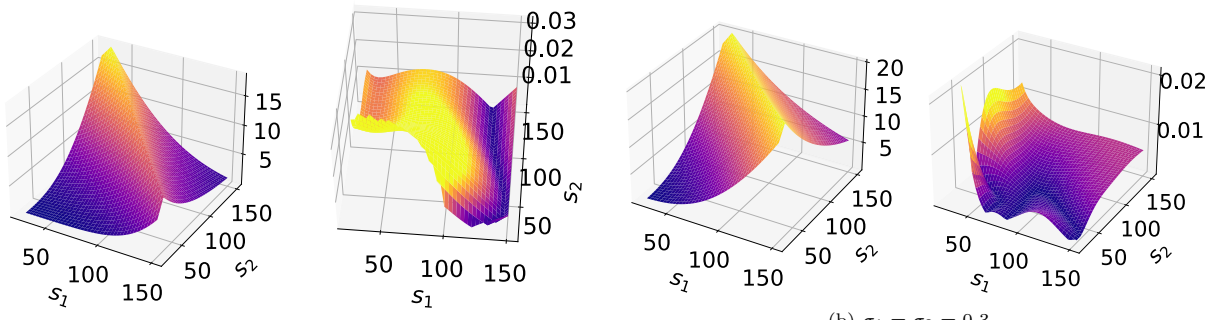
A comparison of runtimes with alternative methods is included in Section 6.5.2.

6.2.2. 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. Fig. 5 shows the approximation and the error over the domain of interest. We see a good approximation with error values below 0.2 in the whole domain. Again, we note that the residual value of the option is of a smaller magnitude with values varying between 0 and 10.

For a more detailed evaluation of the solution quality, we also consider the implied volatility. As outlined in Section 4.2, we use a concept extending the standard implied volatility to the multi-asset case. When options are far in the 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 Fig. 6a, 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 Fig. 6a. Fig. 6b shows the same quantities for different



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

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

Fig. 6. 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.

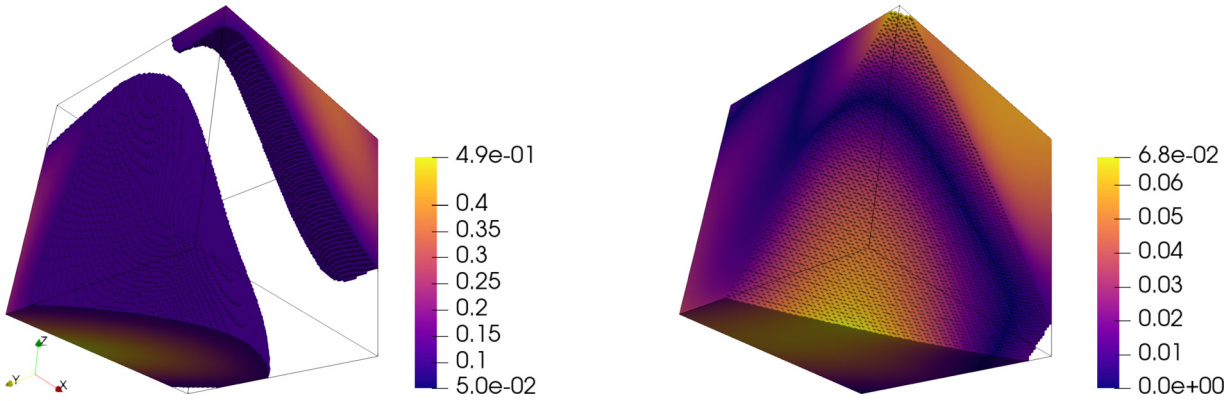


Fig. 7. Different errors for three underlyings. Left: Pricing error, only shown where it is larger than 0.1. Right: Relative error in the volatility, evaluated where the difference to c_{lb} is larger than 0.5. Both images share the same perspective, facing the corner where $S_1 = S_2 = S_3 = 25$. The parameters are $\sigma_1 = \sigma_3 = 0.1$, $\sigma_2 = 0.2$, $\rho_{1,2} = 0.2$ and $\rho_{2,3} = 0.5$.

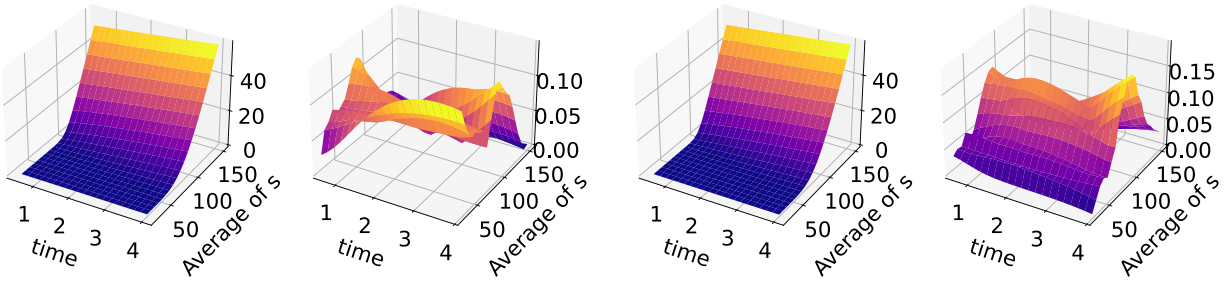
parameter values. Thanks to the parametric approach, in both situations the solution is computed by evaluating the same neural network. With the second set of parameters the difference between the exact option price and c_{lb} stays above 0.5, so we can compare the implied volatility on the whole domain. With both sets of parameters, the relative error in the implied volatility is smaller than 3% 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 the maximal time to maturity and for fixed parameter values. In Fig. 7, 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 0.1. Only a small part of the domain exhibits errors of up to 0.5. While the maximal error is larger than in the previous example of two assets, we will see in the following section that this is mainly due to the considered parameter value. We note that with $\sigma_1 = \sigma_3 = 0.1$, we have considered parameter values which are at the edge of the considered parameter domain $\sigma_i \in [0.1, 0.3]$, which is where we expect the largest errors In the right of the figure, we show the relative error of the implied volatility. As before, we see that the implied volatility is a reasonable error measure in most of the domain and see a relative error well below 10% with large 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 Fig. 8, 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. In the case of five underlying assets the error is the largest at the money as well as out of the money. For eight underlying assets, we see the largest error at the money, where the exact price is the furthest away from the trivial no-arbitrage bound. Note that with eight underlying assets, there are 16 parameters and the total dimensionality of the problem is 25.

6.2.3. 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. Here we see the advantages of the parametric approach of our method. This is especially true as we noted in Section 6.2.1 that the simultaneous evaluation at several points is more efficient than a single evaluation.



(a) Five underlyings (b) Eight underlyings

Fig. 8. Averaged solution (left) and maximal errors (right) for different numbers of underlyings.

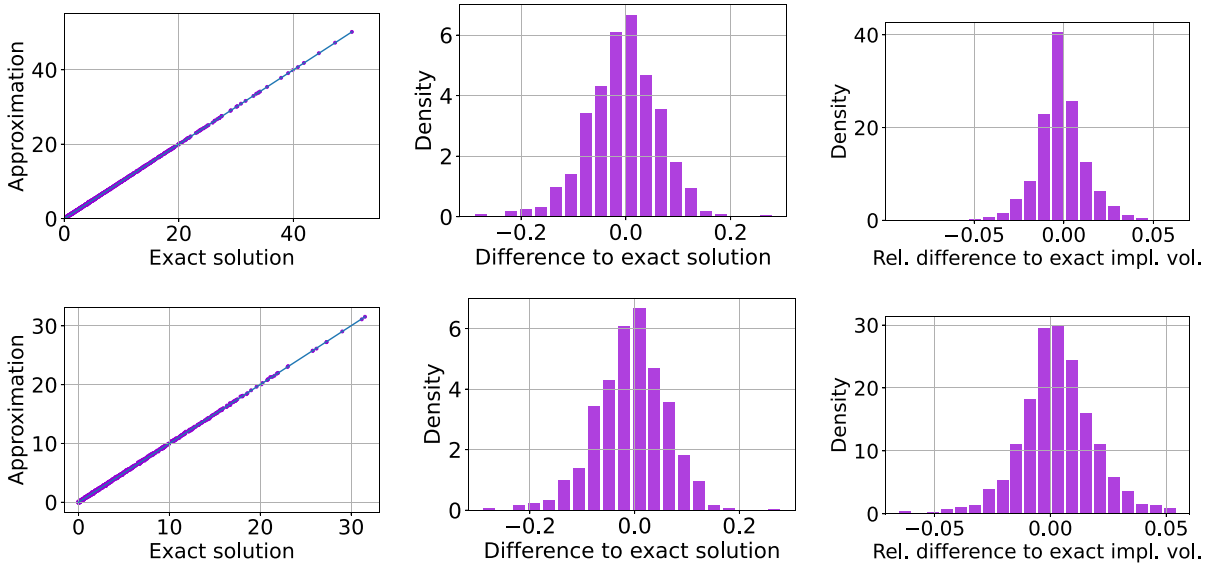


Fig. 9. Samples of prices evaluated for 1,000 different tuples of time, asset prices and parameters chosen uniformly at random from the region of interest. Top: Five underlyings. Bottom: Eight underlyings. Left: Exact solution and approximation on the x- and the y-axis. Points close to the diagonal show a small error. Middle: Histogram of the difference against the exact solution. Right: Histogram of the relative difference to the implied volatility (IV), computed for option prices with a difference to c_{lb} of at least 0.5.

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 Fig. 9, different scatter and histogram plots are shown, visualising the exact and approximated solution, the absolute error and the relative error in the implied volatility. Almost all absolute errors remain below 0.2, while isolated error values of close to 0.3 can be seen. For the implied volatility a similar picture as before is seen, with almost all relative error values below 5% and all relative error values below 10%. For both the absolute error and the relative error to the implied volatility, we see that the bias is significantly smaller than the variance. While this is common for supervised learning with neural networks, we note that the deep parametric PDE method does not use any trainings data. This confirms once more the suitability of our parametric least-squares formulation 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 Fig. 10, 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 the parameter norm measures the distance to the reference parameters $r = 0.02$, $\sigma_i = 0.2$ 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 $\bar{S} = \sum_{i=1}^{d} S_i/d$.

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 all cases, we see a peak of the error when the option is near the money with this peak becoming more pronounced for higher dimensions. However, as seen in the previous section, this does not necessarily imply an increase

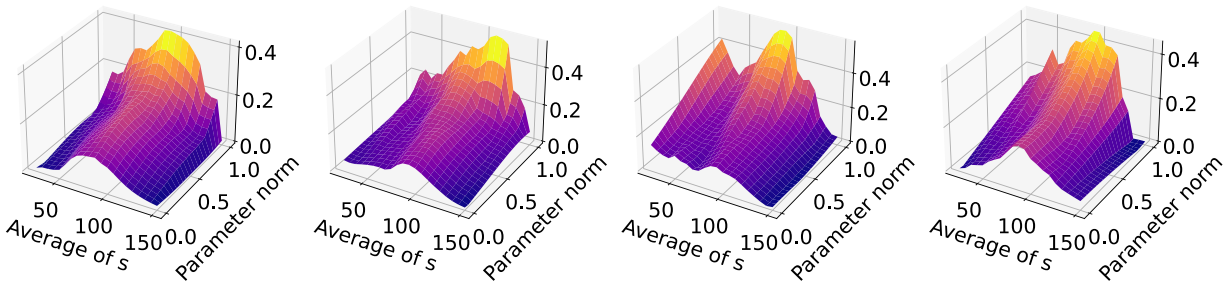


Fig. 10. Maximal error for the basket-options depending on the parameter and the average asset price at maximal time to maturity. From left to right $d = 2, 3, 5, 8$.

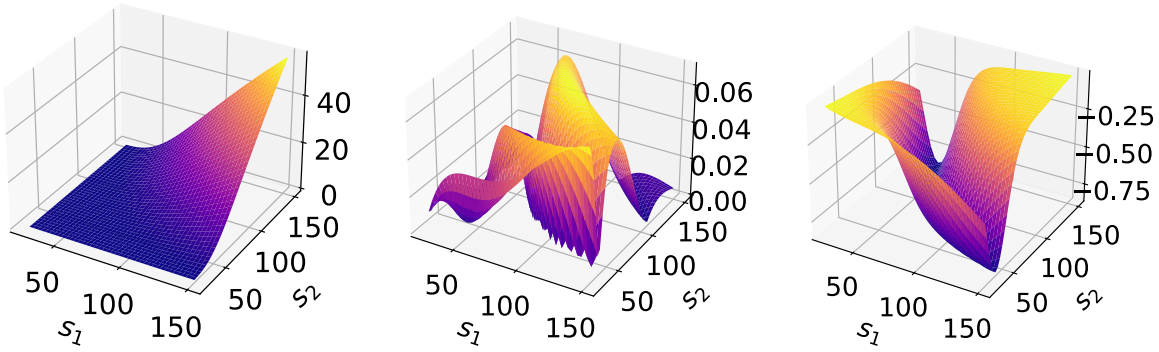


Fig. 11. From left to right: Deep parametric PDE approximation, errors and approximated residual value for two underlyings. All pictures evaluated for $\sigma = 0.1$ at maximal time to maturity using the geometric payoff.

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.

6.3. 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 4.3.2, 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 (6), the approximated no-arbitrage bound 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 Fig. 11. We can see error levels below 0.06 in the domain of interest. Compared to the basket call options, we see a different profile of the approximative residual value.

Fig. 12 shows the maximal error depending on the parameter norm and the average asset price. We still see a peak for large parameters, but only for $d = 3$ a clear peak at the money. For $d = 5, 8$ the error peaks far out of the money. 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 slightly higher.

To measure relative errors, we again consider the implied volatility as described in Section 4.2. Histograms of the relative difference to the exact implied volatility using 1,000 random values are shown in Fig. 13. They show slightly similar error magnitudes compared to the original setting of basket call options. We note that even in the case $d = 8$ the relative error in the implied volatility is below 0.05, which suggests that the slightly larger error observed previously is not practically relevant.

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

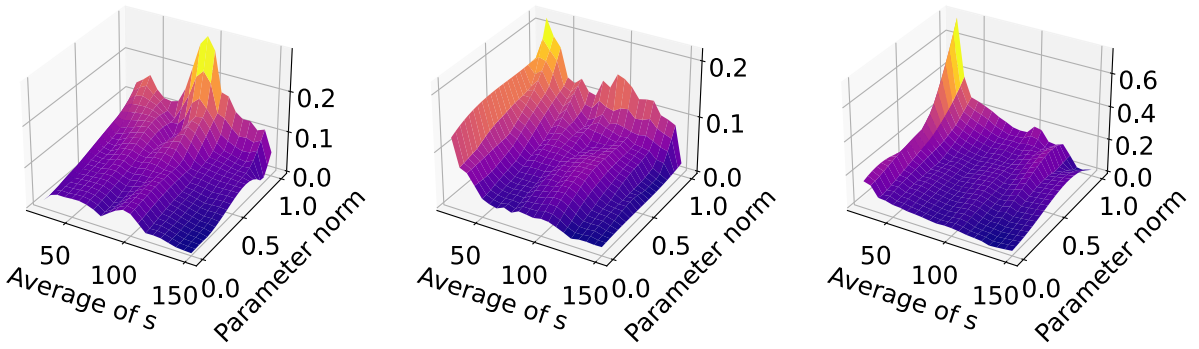


Fig. 12. Maximal error using geometric payoff depending on the parameter and the average asset price at maximal time to maturity for different dimensions. From left to right $d = 3, 5, 8$.

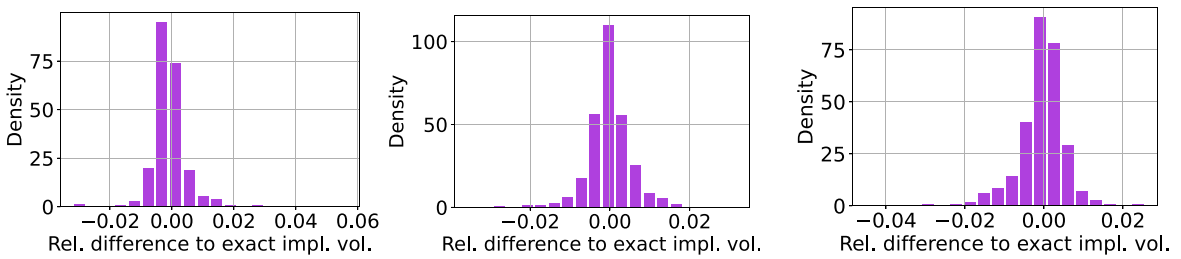
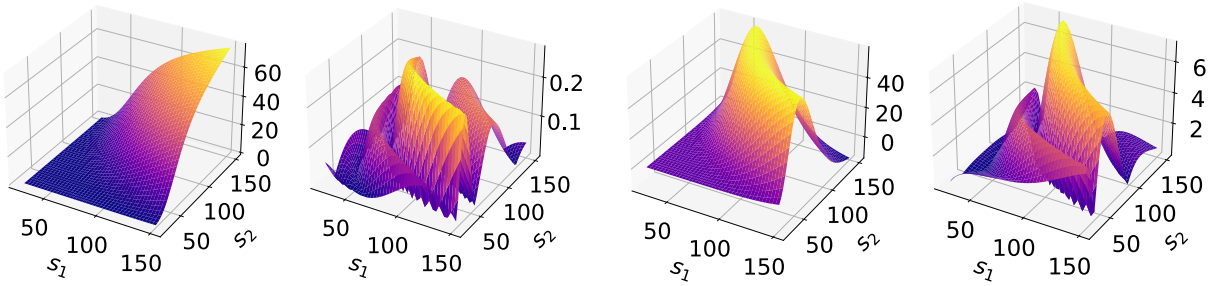


Fig. 13. Histogram of relative difference to exact implied volatility for $d = 3, 5, 8$ (from left to right).



(a) Derivative with respect to the log-price of the first underlying.

(b) Derivative with respect to the volatility.

Fig. 14. Accuracy of derivatives for $d = 2$. Left: Derivative; Right: Error.

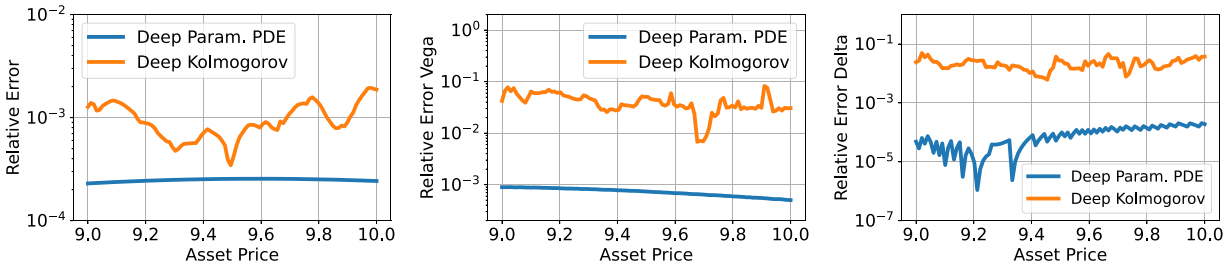
6.4. Greeks and sensitivities

Besides the approximated prices, the Greeks (i.e., derivatives of the price surface) are of utmost importance for applications, e.g., for hedging and risk management. As the deep parametric PDE method trains one network for all time, state and parameter values, we have easy access to these derivatives. Exemplary, Fig. 14a 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. Fig. 14b shows that the derivative with respect to the space variable is significantly more accurate than the one with respect to the volatility. We attribute this to the fact that the space derivatives are a part of the loss function. Therefore, to improve the accuracies of the sensitivities with respect to the parameters, it might be helpful to add the PDE of the sensitivity to the loss function. This approach is well-defined. A thorough study goes beyond the scope of this article and is part of ongoing research.

6.5. Comparison to alternative machine learning methods

To fully understand the performance of the deep parametric PDE method, we compare it to other machine learning method. We consider two separate cases: a comparison to a parametric method and non-parametric methods.



(a) Pricing error. (b) Error in vega. (c) Error in delta.

Fig. 15. Comparison of different relative errors for a single-asset European put option with $T = 1$, $K = 11, \sigma = 0.35$, $r = 0.0$ and a varying asset price. Comparison of the deep Kolmogorov method with the deep parametric PDE method.

6.5.1. Comparison to a parametric method

We extend the initial comparison from Section 2.5 with the deep Kolmogorov method. We consider the remaining results given in Berner et al. [5] and an additional example. Again, we consider the relative error norm (5) to be consistent with the available results.

As described in Section 2.5.1, we consider the setting of their article, i.e., European put options with a strike price of 11. Fig. 15 shows relative errors for a fixed parameter with varying asset prices. Both methods solve the PDE for all parameters in the same parametric domain, yielding a fair comparison. The first figure shows the pricing error. We see that the deep parametric PDE method shows a lower error for all asset prices considered. On average, our method is four times as accurate as the deep Kolmogorov method. In addition, the deep parametric PDE method proves clearly more stable over different asset prices. We attribute the oscillatory error of the deep Kolmogorov method to the inherent simulation steps during the training. In contrast, the PDE approach does not require simulations and hence shows a more stable error.

Also the Greeks are significantly improved thanks to the PDE approach. In Fig. 15b, we see the relative error of vega. The Greek vega is the derivative with respect to the volatility: $\partial_\sigma u(t, x; \mu)$. The deep parametric PDE method is by several magnitudes more accurate. We attribute this to the higher stability seen in Fig. 15a.

While the deep Kolmogorov approach is based on simulations during training, the deep parametric PDE method includes derivatives in the loss function. As we have seen in Section 6.4, these derivatives are approximated more accurately than the others. Therefore, we consider delta, the derivative with respect to the asset price: $\partial_s u(t, \log(s); \mu)$. While up to here the results of the deep Kolmogorov method directly stem from the paper, no values for delta are given. Instead, we used the trained neural network, which is publicly available. The results of this experiment are shown in Fig. 15c. As expected, the performance of the deep parametric PDE method is even stronger, with an accuracy in the order of 10^{-5} to 10^{-4}. The performance is so good, that we seem to already see the influence of the accuracy of the back-propagation which is used for the evaluation of derivatives.

6.5.2. Comparison to non-parametric methods

In addition, we consider non-parametric methods: the deep BSDE solver [27] and the deep Galerkin method [4]. To compare with the deep BSDE solver, we use the code provided by the authors of Han et al. [27] 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 Sirignano and Spiliopoulos [4], does not consider the trivial no-arbitrage bound 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 include the approximated no-arbitrage bound for the deep Galerkin method.

All values and the reference pricer are listed in Table 3 and compared to a 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 3 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.

Table 3
Comparison of different methods to compute the value of an at the money European basket call option with time to maturity T=1 in the Black–Scholes model. Strike price 100 with eight underlying assets $s_i = 100$, $\sigma_i = 0.2$, $\rho_{ij} = 0$, $i, j = 1, \ldots, 8$, $i \neq j$ and $r = 0.02$. Monte–Carlo solution with 10^9 samples as the exact solution (estimated standard deviation 0.000344).

	Monte–Carlo	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	–	–	–	28 min for each $\hat{\mu}$	59 min once $\forall \mu \in \mathcal{P}$
online runtime	6 min 41 s	1.34 s	12 min 7 s	39.9 ms	41.9 ms

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.

7. Conclusions

We have proposed, formalised and analysed the deep parametric PDE method as an efficient solver for a whole family of parabolic problems. We train a single neural network with a smooth activation funcition using the least-squares formulation of the parabolic parametric PDE. This formulation and the smoothness of the network allow us to prove approximation results for the proposed method, relating the size of the network to the accuracy. Exact rates on this relation are subject to a follow-up paper. Due to limitations of the numerical optimiser we only partially confirm the theoretical result in practice. More research is needed to the numerical optimisation procedure as any improvement in the optimiser will directly result in a more accurate approximation. On the other hand, we have seen that prior knowledge about the solution can be incorporated and improves the method's accuracy. The use of deep neural networks allows us to solve high-dimensional problems accurately and fast. After a single training phase, the method quickly evaluates the solution and its derivatives 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, Greeks and implied volatility 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, also in comparison to alternative methods, 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, instead of the Black–Scholes model a stochastic volatility model can be considered. Furthermore, 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 and its derivatives has to be evaluated for many parameters, an example application is the exposure calculation under uncertain parameters.

Funding sources

This work was funded by the Alan Turing Institute (KG) and EPSRC grant no. EP/T004738/1 (KG and LW). This research utilised Queen Mary's Apocrita HPC facility, supported by QMUL Research-IT. doi:10.5281/zenodo.438045.

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.

References

[1] L. Gonon, C. Schwab, Deep ReLU network expression rates for option prices in high-dimensional, exponential Lévy models, Finance Stoch. 25 (2021) 615–657.

[2] P. Grohs, F. Hornung, A. Jentzen, P. von Wurstemberger, A proof that artificial neural networks overcome the curse of dimensionality in the numerical approximation of Black–Scholes partial differential equations, arXiv:1809.02362 (2018).
[3] M. Hutzenthaler, A. Jentzen, T. Kruse, 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 Differ. Equ. Appl. 1 (10) (2020) 1–34.
[4] J. Sirignano, K. Spiliopoulos, DGM: a deep learning algorithm for solving partial differential equations, J. Comput. Phys. 375 (2018) 1339–1364.
[5] J. Berner, M. Dablander, P. Grohs, Numerically solving parametric families of high-dimensional Kolmogorov partial differential equations via deep learning, in: Advances in Neural Information Processing Systems 33 Pre-Proceedings (NeurIPS 2020), 2020, pp. 1–13.
[6] A. Maran, A. Pallavicini, S. Scoleri, Chebyshev Greeks: Smoothing Gamma without bias, preprint (2021). https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3872744.
[7] M.B. Giles, Multilevel Monte Carlo methods, Acta Numer. 24 (2015) 259–328.
[8] P. L'Ecuyer, Quasi-Monte Carlo methods with applications in finance, Finance Stoch. 13 (3) (2009) 307–349.
[9] E. Eberlein, K. Glau, A. Papapantoleon, Analysis of Fourier transform valuation formulas and applications, Appl. Math. Finance 17 (3) (2010) 211–240.
[10] C. Bayer, M. Siebenmorgen, R. Tempone, Smoothing the payoff for efficient computation of basket option prices, Quant. Finance 18 (3) (2018) 491–505.
[11] M. Griebel, M. Holtz, Dimension-wise integration of high-dimensional functions with applications to finance, J. Complex. 26 (5) (2010) 455–489.
[12] M. Holtz, Sparse Grid Quadrature in High Dimensions with Applications in Finance and Insurance, Lecture Notes in Computational Science and Engineering, Springer, 2011.
[13] K. Glau, D. Kressner, F. Statti, Low-rank tensor approximation for Chebyshev interpolation in parametric option pricing, SIAM J. Financ. Math. 11 (3) (2020) 897–927.
[14] K.i.t. Hout, J. Toivanen, Application of operator splitting methods in finance, in: R. Glowinski, S.J. Osher, W. Yin (Eds.), Splitting Methods in Communication, Imaging, Science, and Engineering, Springer International Publishing, Cham, 2016, pp. 541–575.
[15] C. Reisinger, R. Wissmann, Finite difference methods for medium- and high-dimensional derivative pricing PDEs, in: M.A.H. Dempster, J. Kanniainen, J. Keane, E. Vynckier (Eds.), High-Performance Computing in Finance Problems, Methods, and Solutions, 2018, pp. 175–195.
[16] N. Hilber, O. Reichmann, C. Schwab, C. Winter, Wavelet methods, in: Computational Methods for Quantitative Finance: Finite Element Methods for Derivative Pricing, Springer, Berlin, 2013, pp. 159–176.
[17] U. Pettersson, E. Larsson, G. Marcusson, J. Persson, Improved radial basis function methods for multi-dimensional option pricing, J. Comput. Appl. Math. 222 (2008) 82–93.
[18] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (7553) (2015) 436–444.
[19] J.M. Hutchinson, A.W. Lo, T. Poggio, A nonparametric approach to pricing and hedging derivative securities via learning networks, J. Finance 49 (3) (1994) 851–889.
[20] M. Malliaris, L. Salchenberger, A neural network model for estimating option prices, Appl. Intell. 3 (1993) 193–206.
[21] E. Barucci, U. Cherubini, L. Landi, Neural networks for contingent claim pricing via the Galerkin method, in: H. Amman, B. Rustem, A. Whinston (Eds.), Computational Approaches to Economic Problems, Springer, 1997, pp. 127–141.
[22] A. Meade, A. Fernandez, The numerical solution of linear ordinary differential equations by feedforward neural networks, Math. Comput. Model. 19 (12) (1994) 1–25.
[23] J. Ruf, W. Wang, Neural networks for option pricing and hedging: a literature review, J. Comput. Finance 24 (1) (2020) 1–46.
[24] M.S. Vidales, D. Šiška, L. Szpruch, Unbiased deep solvers for parametric PDEs, (2019). arXiv:1810.05094
[25] C. Beck, W. E, A. Jentzen, Machine learning approximation algorithms for high-dimensional fully nonlinear partial differential equations and second-order backward stochastic differential equations, J. Nonlinear Sci. (2019) 1563–1619.
[26] Q. Chan-Wai-Nam, J. Mikael, X. Warin, Machine learning for semi linear PDEs, J. Sci. Comput. 79 (3) (2019) 1667–1712.
[27] J. Han, A. Jentzen, W. E, Solving high-dimensional partial differential equations using deep learning, Proc. Natl. Acad. Sci. 115 (34) (2018) 8505–8510.
[28] C. Huré, H. Pham, X. Warin, Deep backward schemes for high-dimensional nonlinear PDEs, Math. Comput. 89 (2020) 1547–1579.
[29] A. Al-Aradi, A. Correia, D. de Frietas Naiff, G. Jardim, Y. Saporito, Applications of the deep Galerkin method to solving partial integro-differential and Hamilton–Jacobi–Bellman equations, (2019). arXiv:1912.01455
[30] A. Al-Aradi, A. Correia, D. Naiff, G. Jardim, Y. Saporito, Solving nonlinear and high-dimensional partial differential equations via deep learning, (2018). arXiv:1811.08782
[31] J. Li, J. Yue, W. Zhang, W. Duan, The deep learning Galerkin method for the general Stokes equations, (2020). arXiv:2009.11701.
[32] W. Chen, Q. Wang, J. Hesthaven, C. Zhang, Physics-informed machine learning for reduced-order modeling of nonlinear problems, J. Comput. Phys. 446 (2021) 110666.
[33] G. Pang, L. Lu, G.E. Karniadakis, fPINNs: fractional physics-informed neural networks, SIAM J. Sci. Comput. 41 (4) (2019) A2603–A2626.
[34] M. Raissi, P. Perdikaris, 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 (2019) 686–707.
[35] S. Liu, C. Oosterlee, S. Bohte, Pricing options and computing implied volatilities using neural networks, Risks 7 (1) (2019).
[36] Y. Khoo, J. Lu, L. Ying, Solving parametric PDE problems with artificial neural networks, Eur. J. Appl. Math. (2020) 1–15.
[37] K. Bhattacharya, B. Hosseini, N.B. Kovachki, A.M. Stuart, Model reduction and neural networks for parametric PDEs, SMAI J. Comput. Math. 7 (2021) 121–157.
[38] Z. Li, N.B. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, A. Anandkumar, Fourier neural operator for parametric partial differential equations, in: International Conference on Learning Representations, 2021, pp. 1–16.
[39] S. Liu, A. Borovykh, L.A. Grzelak, C.W. Oosterlee, A neural network-based framework for financial model calibration, J. Math. Ind. 9 (1) (2019) 9.
[40] B. Horvath, A. Muguruza, M. Tomas, Deep learning volatility: a deep neural network perspective on pricing and calibration in (rough) volatility models, Quant. Finance (2020) 1–17.
[41] K. Andersson, C.W. Oosterlee, Deep learning for CVA computations of large portfolios of financial derivatives, Appl. Math. Comput. 409 (2021) 126399.
[42] A. Gnoatto, A. Picarelli, C. Reisinger, Deep xVA solver–a neural network based counterparty credit risk management framework, (2020). arXiv:2005.02633
[43] K. Andersson, C.W. Oosterlee, A deep learning approach for computations of exposure profiles for high-dimensional Bermudan options, Appl. Math. Comput. 408 (2021) 126332.
[44] J.L. Lions, E. Magenes, Non-Homogeneous Boundary Value Problems and Applications, vol. II, Springer, 1972.
[45] J.L. Lions, E. Magenes, Non-Homogeneous Boundary Value Problems and Applications, vol. I, Springer, 1972.
[46] M. Kac, On distributions of certain Wiener functionals, Trans. Am. Math. Soc. 65 (1949) 1–13.
[47] R.K. Srivastava, K. Greff, J. Schmidhuber, Training very deep networks, in: Proceedings of the 28th International Conference on Neural Information Processing Systems, vol. 2, MIT Press, Cambridge, MA, USA, 2015, pp. 2377–2385.
[48] R. Cont, N. Lantos, O. Pironneau, A reduced basis for option pricing, SIAM J. Financ. Math. 2 (1) (2011) 287–316.
[49] B. Haasdonk, J. Salomon, B. Wohlmuth, A reduced basis method for the simulation of American options, in: A. Cangiani, R.L. Davidchack, E. Georgoulis, A.N. Gorban, J. Levesley, M.V. Tretyakov (Eds.), Numerical Mathematics and Advanced Applications 2011, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 821–829.
[50] A. Bercher, L. Gonon, A. Jentzen, D. Salimova, Weak error analysis for stochastic gradient descent optimization algorithms, (2020). arXiv:2007.02723
[51] J. Eldering, Normally Hyperbolic Invariant Manifolds: The Noncompact Case, Atlantis Press, 2013.
[52] K. Hornik, Approximation capabilities of multilayer feedforward networks, Neural Netw. 4 (2) (1991) 251–257.
[53] P. Doust, Two useful techniques for financial modelling problems, Appl. Math. Finance 17 (3) (2010) 201–210.

[54] A. Gulisashvili, P. Tankov, Implied volatility of basket options at extreme strikes, in: P.K. Friz, J. Gatheral, A. Gulisashvili, A. Jacquier, J. Teichmann (Eds.), Large Deviations and Asymptotic Methods in Finance, Springer International Publishing, Cham, 2015, pp. 175–212.
[55] C. Pötz, Function Approximation for Option Pricing and Risk Management, Queen Mary University of London, 2020 Ph.D. thesis.
[56] F. Chollet, et al., Keras, 2015, Software available from https://keras.io.
[57] M. Abadi, et al., TensorFlow: large-scale machine learning on heterogeneous systems, 2015, Software available from https://tensorflow.org.
[58] T. King, S. Butcher, L. Zalewski, Apocrita - High Performance Computing Cluster for Queen Mary University of London, Zenodo, 2017.
[59] D.P. Kingma, J. Ba, Adam: a method for stochastic optimization, 2015.
[60] X. Glorot, Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, in: Y.W. Teh, M. Titterington (Eds.), Proceedings of the13th International Conference on Artificial Intelligence and Statistics (AISTATS) 2010, Proceedings of Machine Learning Research, JMLR Workshop and Conference Proceedings, vol. 9, 2010, pp. 249–256.
[61] T. O'Malley, E. Bursztein, J. Long, F. Chollet, H. Jin, L. Invernizzi, et al., Keras Tuner, 2019, Software available from https://github.com/keras-team/keras-tuner.
[62] Y. Shin, J. Darbon, G.E. Karniadakis, On the convergence of physics informed neural networks for linear second-order elliptic and parabolic type PDEs, Commun. Comput. Phys 28 (2020) 2042–2074.