Machine Learning in Finance

The Deep Parametric PDE Method

Université Paris Cité
Master M2MO

This project focuses on a summary and critical discussion of the research paper [GW22], "The deep parametric PDE method and applications to option pricing". We first provide an overview of the paper and a state of the art of Deep Learning techniques used to solve pricing problems in finance, whith an emphasis on partial differential equation (PDE) approaches. We then implement the Deep Parametric PDE method for pricing a European basket put option, test its limits, and benchmark its performance. Finally, we provide some possible future research on this topic.

Students:
MICHAL Tangui, PÉCHEUL Ronan, SANGLIER Nathan
Email: michaltangui@gmail.com, ronan.pecheul@ensae.fr, nathan.sanglier@etu.u-paris.fr

Course Professors:
FERMANIAN Jean-David, PHAM Huyên

Date: April 27, 2025
Academic Year: 2024-2025

Université Paris Cité

Contents

1 The Deep Parametric PDE Method

In this project, we analyze the work of [GW22], who develop a new Deep Learning method to solve potentially high dimensional parametric PDEs: the Deep Parametric PDE method. For any set of parameters values $\mu \in P$, assume we want to find the solution on a bounded domain $(0,T) \times \Omega$ of the following PDE:

$$\begin{cases} \partial_t u(t,x;\mu) + \mathcal{L}_x^\mu u(t,x;\mu) &= f(t,x;\mu) \quad (t,x) \in (0,T] \times \Omega, \\ u(0,x;\mu) &= g(x;\mu) \quad x \in \Omega, \\ u(t,x;\mu) &= u_\Sigma(t,x;\mu) \quad (t,x) \in (0,T] \times \partial\Omega, \end{cases} \tag{1}$$

where $\mathcal{L}_x^\mu$ is a second order differential operator on x and parametrized by μ. Moreover, u_Σ represents the boundary conditions due to the truncated domain $\Omega \subset \mathbb{R}^d$. Throughout this section and as in [GW22], we focus on the specific framework of parabolic PDEs, meaning that $\mathcal{L}_x^\mu$ is elliptical. Indeed, it refers to the type of PDEs we find in European option pricing. The goal is to learn the solution of (1) for any values of parameters μ *simultaneously*. Once the model has been trained, one can evaluate u at (t,x) for different values of μ, without additional computational burden.

1.1 An extension of the Deep Galerkin Method

The Deep Parametric (DP) PDE method is nothing but an extension of the Deep Galerkin (DG) method presented in [SS18]. In DG, the parameter μ is fixed and they reformulate the PDE as a stochastic optimization problem on the set of all the Borel measurable functions v associated with the following loss:

$$\begin{aligned} \mathbb{L}(v) =& \mathbb{E}\left[|\partial_t v(\hat{t},\hat{x};\mu) + \mathcal{L}_x^\mu v(\hat{t},\hat{x};\mu) - f(\hat{t},\hat{x};\mu)|^2 \right. \\ & \left. + |v(0,\tilde{x};\mu) - g(\tilde{x};\mu)|^2 + |v(\bar{t},\bar{x};\mu) - v_\Sigma(\bar{t},\bar{x};\mu)|^2\right], \end{aligned} \tag{2}$$

where $(\hat{t},\hat{x}) \sim \nu_1$ with support on $(0,T] \times \Omega$, $\tilde{x} \sim \nu_2$ with support on Ω, and $(\bar{t},\bar{x}) \sim \nu_3$ with support on $(0,T] \times \partial\Omega$. In practice, uniform distributions are used.

We see that u is solution of the PDE (1) if and only if u minimizes $\mathbb{L}$, ie $\mathbb{L}(u) = \inf_v \mathbb{L}(v) = 0$. As this problem has infinite dimension, they focus on the specific class of Neural Network functions v_θ parametrized by θ. Thus, the problem is now to find θ^* such that $\mathbb{L}(v_{\theta^*}) = \inf_\theta \mathbb{L}(v_\theta)$. Notice there is no guarantee to find $\mathbb{L}(v_{\theta^*})$, meaning that in practice, v_{θ^*} does not exactly solve the PDE (1). As it is not possible to directly calculate the expectation in (2), they use the trick of stochastic gradient descent (SGD).

1. Generate sample points $s_n = \left\{(\hat{t}_n,\hat{x}_n), \tilde{x}_n, (\bar{t}_n,\bar{x}_n)\right\}$ where $(\hat{t}_n,\hat{x}_n) \sim \nu_1$, $(\tilde{x}_n) \sim \nu_2$, and $(\bar{t}_n,\bar{x}_n) \sim \nu_3$.
2. Compute the empirical error at the sampled points:

$$\begin{aligned} l(\theta_n, s_n) =& |\partial_t v_{\theta_n}(\hat{t}_n,\hat{x}_n;\mu) + \mathcal{L}_x^\mu v_{\theta_n}(\hat{t}_n,\hat{x}_n;\mu) - f(\hat{t}_n,\hat{x}_n;\mu)|^2 \\ & + |v_{\theta_n}(0,\tilde{x}_n;\mu) - g(\tilde{x}_n;\mu)|^2 + |v_{\theta_n}(\bar{t}_n,\bar{x}_n;\mu) - v_\Sigma(\bar{t}_n,\bar{x}_n;\mu)|^2. \end{aligned} \tag{3}$$

3. Take a gradient descent step $\theta_{n+1} = \theta_n - \alpha_n \nabla_\theta l(\theta_n, s_n)$, where α_n is the learning rate.
4. Repeat until a convergence criterion is satisfied.

In this training procedure, the trick is that $\nabla_\theta l(\theta_n, s_n)$ is an unbiased estimate of $\nabla_\theta \mathbb{L}(v_{\theta_n})$. As neural network functions are generally non-convex in θ, there is no guarantee to converge to a global minimum. However, this approach is effective in practice and has been extended to have better control over the bias-variance tradeoff. Both [GW22] and [SS18] use the Adam optimizer (see [KB17]) for their implementation.

The authors do not introduce the Deep Parametric PDE method as a stochastic optimization problem, but directly as an empirical minimization of square residuals with an error similar to (3), but randomizing μ in the same way as for (t, x):

$$\begin{aligned} l(\theta_n, s_n) =& |\partial_t v_{\theta_n}(\hat{t}_n, \hat{x}_n; \hat{\mu}_n) + \mathcal{L}_x^{\hat{\mu}_n} v_{\theta_n}(\hat{t}_n, \hat{x}_n; \hat{\mu}_n) - f(\hat{t}_n, \hat{x}_n; \hat{\mu}_n)|^2 \\ &+ |v_{\theta_n}(0, \tilde{x}_n; \tilde{\mu}_n) - g(\tilde{x}_n; \tilde{\mu}_n)|^2 + c_{bc} \cdot |v_{\theta_n}(\bar{t}_n, \bar{x}_n; \bar{\mu}_n) - v_\Sigma(\bar{t}_n, \bar{x}_n; \bar{\mu}_n)|^2, \end{aligned} \tag{4}$$

where $s_n = \big\{(\hat{t}_n, \hat{x}_n, \hat{\mu}_n), (\tilde{x}_n, \tilde{\mu}_n), (\bar{t}_n, \bar{x}_n, \bar{\mu}_n)\big\}$. In practice, they also take uniform distributions for training measures. Notice that c_{bc} is a binary constant, which is set to 0 if we want to discard the impact of boundary conditions on the loss function (see Section 3). Thus, with a single training procedure, an approximate solution is available for all considered PDEs simultaneously (ie for different values of μ) and for any point (t, x), but at the cost of a higher dimensional minimization problem.

1.2 Key Properties of the Deep Parametric PDE Method

It is important to note that both the Deep Galerkin and the Deep Parametric PDE methods are *unsupervised* learning problems and *meshfree*. Indeed, no sample values of the function satisfying (1) are needed and the neural network is trained on randomly sampled time and space points, unlike other approaches detailed in Section 2. Moreover, these two methods use the same variant of highway networks (see [SGS15]) for their neural network architecture, which accommodates well to sharp initial conditions like $(f(x) - K)_+$ we find in option pricing. This kind of architecture enables to ease gradient-based training of very deep neural networks through the use of gates (tanh activation functions). Inspired by LSTM recurrent networks [HS97], these gates control information flow across several layers without attenuation.

The authors also provide the existence of a neural network minimizing their loss, which approximates the true solution of the parametric PDE up to any prescribed accuracy (assuming global convergence of the Adam optimizer). Their proof can deal with non-smooth initial conditions as in finance, but requires enough regularity on boundary conditions (not the case in finance).

Solving parametric PDEs is of particular interest in finance as it enables to directly compute the partial derivatives of the solution with respect to the parameters in μ (the "Greeks"), using automatic differentiation. Moreover, it enables the calibration of the parameters using real-world data and uncertainty risk analyses.

2 Literature Review

2.1 Deterministic Approaches

The first way to solve the PDE (1) is to use a natural deterministic approach, ie solve the PDE as it is given. Historically, the main methods used are finite difference schemes. It consists in

the discretization of the domain $(0,T) \times \Omega$ into a grid and approximating the derivatives of u at each node by finite differences. The pros of these finite difference methods is that they can be extended to more complicated PDEs as in American option pricing [BD20], stochastic control and mean-field games [ACD10]. Moreover, the approximation errors are well quantified. However, they suffer from the curse of dimensionality because of the mesh. Thus, it is not appropriate here, where we want the solution for a parametric family of PDEs.

Among other techniques, the use of Deep Learning to solve high dimensional PDEs has gained increasing interest in the recent years, thanks to progress made in computational efficiency and neural networks architectures. The idea of leveraging the universal approximation property of neural networks is not new, and can be traced back to the early 1990's, for instance with the work of [LK90] which uses a Hopfield network to solve the finite difference equations related to the PDE. More modern approaches, such as [LLF98], approximate the solution of the PDE based on a feedforward neural network which is trained on a specific grid. Unfortunately, these kind of approaches are not meshfree and cannot be employed for high dimensional PDEs.

Contrary to classic (large) feedforward neural networks, deep neural networks are made of multiple hidden layers enabling more complex representations. Most of the approaches are unsupervised learning tasks, like the Deep Galerkin [SS18] detailed in Section 1. Moreover, Physics-Informed Neural Networks (PINNs) in [RPK19] allow to incorporate (potentially noisy) prior information on the PDE and its solution by an additional supervised training. PINNs have been extended in [BdBS+24] to solve parametric PDEs, but to the best of our knowledge, only the Deep Parametric PDE method of [GW22] uses a fully deterministic and unsupervised approach to solve a parametric PDE. Although these methods are simple to implement and can be easily extended to other classes of PDEs, their performance highly relies on the choice of the training measures.

2.2 Probabilistic Approaches

On the other hand, probabilistic approaches rely on the presentation of the PDE (1) as an expectation of a stochastic process. In financial applications, such equivalence is usually given by the Feynman-Kac theorem for classic stochastic differential equations or by the Dynamic Programming Principle for stochastic control problems. First, an intuitive approach is to compute this expectation by classic Monte Carlo methods. However, these methods only provide the value of the function at a specific point.

Hence, one can use neural networks regression, ie approximate the conditional expectation as a neural network function. Samples of the stochastic process are generated with an appropriate SDE discretization scheme, and regression is performed by minimizing a MSE loss function. Thus, it is referred as a supervised learning problem. For instance, the Deep BSDE method of [HJE18] enables to solve backward stochastic differential equations using forward simulations of the stochastic processes at stake. [HPW20] refines this idea with a backward approach and local minimization problems for each discretization date.

There have been intensive research on the resolution of parametric PDEs with these Deep Learning probabilistic approaches. For instance, [SVSS21] propose different deep learning algorithms for solving linear parametric PDEs and [BDG20] solve high-dimensional parametric Kolmogorov PDEs. Specifically, their Deep Kolmogorov method is the main benchmark used to evaluate the performance of the Deep Parametric PDE method in [GW22]. However, it is only applicable to Kolmogorov PDEs, which are specific cases of (1) but frequently appear in finance, like the Black-Scholes equation. They construct a supervised learning problem via

a nontrivial application of the Feynman-Kac formula. Then, they simulate i.i.d. samples of $(t, x; \mu)$ according to uniform distributions, and i.i.d. samples of the target random variable with an Euler-Maruyama scheme. Finally, they employ a Multilevel neural network architecture to approximate the regression function.

3 Implementation

3.1 European Option Pricing in the Multivariate Black-Scholes Model

In order to benchmark the Deep Parametric PDE method, we will focus on European option pricing in the multivariate Black-Scholes model. Assume we have a risk-neutral probability $\mathbb{Q}$ and d assets $S_t(\mu) = (S_t^1(\mu), \cdots, S_t^d(\mu))^T$ such that:

$$dS_t = \text{diag}(S_t)\left(r\mathbf{1}dt + \sigma dY_t\right),$$

where $\sigma = \text{diag}(\sigma_1, \cdots, \sigma_d)$ and Y is a vector of d correlated Brownian motions under $\mathbb{Q}$, with correlation matrix ρ. Using the Cholesky decomposition $\rho = LL^T$, it can be reformulated as:

$$dS_t = \text{diag}(S_t)\left(r\mathbf{1}dt + \sigma L dW_t\right),$$

where W is a d-dimensional $\mathbb{Q}$-Brownian motion. The $\mathbb{Q}$-price of a European option with payoff $g(S_T)$ at time T is given by:

$$P(\tau, s; \mu) = \mathbb{E}^{\mathbb{Q}}\left[e^{-r(T-\tau)} g\left(S_T(\mu)\right) \mid S_\tau(\mu) = s\right].$$

In order to get the standard form of parabolic PDEs, we reason backward in time and with the assets log-prices, meaning that $u(t, x; \mu) = P(T-t, e^x; \mu)$, where $e^x = (e^{x_1}, \cdots, e^{x_d})^T$. Applying the Feynmac-Kac theorem yields the PDE (1) with:

$$\begin{cases} \mathcal{L}_x^\mu u(t, x; \mu) & = ru(t, x; \mu) - \sum_{i=1}^d \left(r - \frac{\sigma_i^2}{2}\right) \partial_{x_i} u(t, x; \mu) - \sum_{i,j=1}^d \frac{\rho_{ij}\sigma_i\sigma_j}{2} \partial_{x_i x_j} u(t, x; \mu), \\ f(t, x; \mu) & = 0. \end{cases}$$

In their research paper, [GW22] do not provide the boundary condition u_Σ. They completely discard this term and its associated expression in the loss function (4), setting $c_{bc} = 0$. They justify this trick by taking a larger domain of computation $\Omega = (x_{\min}, x_{\max})^d$ than their domain of interest, to lower the sensitivity to the boundary condition. We will study the influence of this choice in this Section.

Notice that the parameters are $\mu = \left(r, (\sigma_i)_{1 \le i \le d}, (\rho_{ij})_{1 \le i \ne j \le d}\right)$. However, the correlation matrix ρ must stay symmetric positive semi-definite when drawing random samples of μ. The solution chosen in [GW22] is to parametrize ρ by its $(d-1)$ pairwise correlations $\hat{\rho}_i = \rho_{i,i+1}$ and calculate $\rho_{ij} = \rho_{ji} = \prod_{k=i}^{j-1} \hat{\rho}_k$, $\forall j > i$. It yields a vector of $2d$ parameters:

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

3.2 Case Study: Pricing a European Basket Put Option

[GW22] provide the procedure and numerical results for a basket call option, but we detail here the pricing of a basket *put* option. This option payoff for log-prices is given for a fixed strike

price K by:

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

For the call option, the authors employ a small trick for increasing the quality of the estimated solution by the highway neural network, which can also be done in the context of our put option. Indeed, we can include prior knowledge by breaking down the option price into a "no-arbitrage bound" and the remaining time value (residual). For a put option, we know that:

$$u(t,x;\mu) \geq \hat{u}_\infty(t,x;\mu) = \left(Ke^{-rt} - \frac{1}{d}\sum_{i=1}^{d} e^{x_i}\right)_+ .$$

Thus, solving the PDE associated to the residual $u - \hat{u}_\infty$ is expected to yield a better approximation for u. However, the no-arbitrage bound $\hat{u}_\infty$ is not smooth and can't be used as such for the neural network. A solution is to consider the following smooth approximation instead of $\hat{u}_\infty$:

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

where λ is set to 0.1 in [GW22] (notice that $\lim_{\lambda\to+\infty} \hat{u}_\lambda = \hat{u}_\infty$).

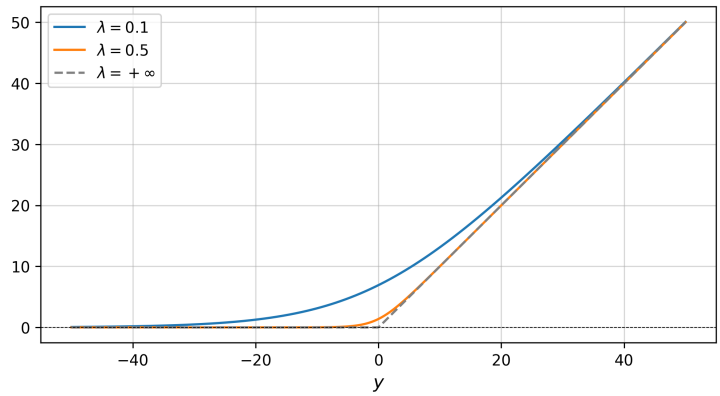

Figure 1: Function $h_\lambda : y \mapsto \frac{1}{\lambda}\ln\left(1 + e^{\lambda y}\right)$ for different values of λ.

If λ is too high, the second derivative of h_λ in Figure 1 would become too large near 0. The transformed PDE for $v = u - \hat{u}_\lambda$ (residual PDE) is now:

$$\begin{cases} \partial_t v(t,x;\mu) + \mathcal{L}_x^\mu v(t,x;\mu) &= \tilde{f}(t,x;\mu) \quad (t,x) \in (0,T] \times \Omega, \\ v(0,x;\mu) &= \tilde{g}(x;\mu) \quad x \in \Omega, \\ v(t,x;\mu) &= v_\Sigma(t,x;\mu) \quad (t,x) \in (0,T] \times \partial\Omega, \end{cases} \tag{5}$$

where $\tilde{f} = f - \partial_t \hat{u}_\lambda - \mathcal{L}_x^\mu \hat{u}_\lambda$, $\tilde{g} = g - \hat{u}_\lambda$, and $v_\Sigma = u_\Sigma - \hat{u}_\lambda$. Estimating v by our deep neural network v_{DPDE} yields the approximation $u_{\text{DPDE}} = v_{\text{DPDE}} + \hat{u}_\lambda$ of the true solution u.

3.3 Numerical Results

There is no analytical formula available for a European basket put option price in the Black-Scholes framework. Thus, in order to evaluate the performance of the Deep Parametric PDE

method, we compare the results with a reference pricer. As in [GW22], we use the work of [BST17] and [Pö20] as the reference pricer (which is not adapted to be an alternative parametric solver). Notice the authors in [GW22] provide an toy implementation of their method[1], but their code does not compile and it is not possible to replicate the results as such. Thus, we have decided to re-implement from scratch the Deep Parametric PDE method for the case of $d = 2$ assets with a code entirely replicable[2] and a frozen `pip` environment. We trained a smaller instance of the highway neural network due to computational limitations. The training was done in the computational domain $[0, T] \times \Omega \times P$. However, the results are presented in this Section for a domain of interest in a subspace of the computational domain, in order to reduce sensitivity to boundaries (see notebook for more information on domains bounds), as in [GW22]. In the same way, we discarded the loss term related to boundary condition ($c_{bc} = 0$) in (4).

In Figure 2, we show (left) a scatter plot of reference prices against our predicted prices for randomly sampled values of (t, x, μ) in the domain of interest. We remark that the predicted prices are close to the reference ones, but often under-valued. Moreover, we show (right) a scatter plot of the predicted prices absolute errors against these predicted prices. We see that the absolute errors are of the same order as the predicted prices, meaning that the out-of-sample errors are amplified for high prices of the put option.

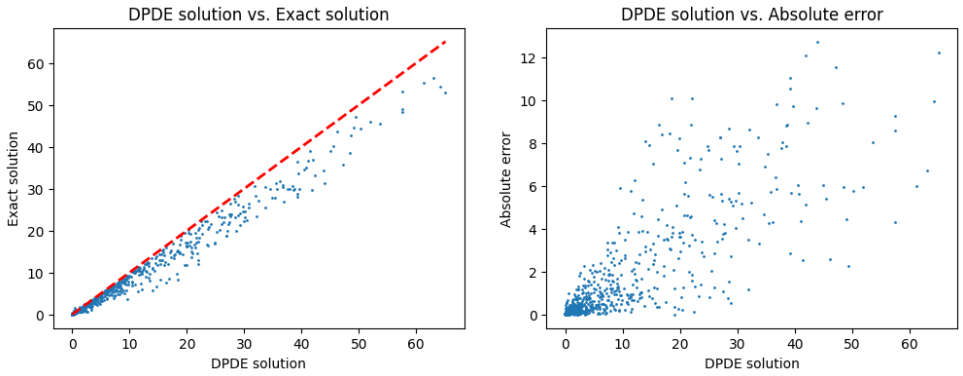

Figure 2: Put option. Predicted and reference prices for randomly sampled points.

Aggregating across these randomly sampled points, we obtain the following metrics:

$$\begin{cases} \text{RMSE} & = 2.60 \\ \text{Norm. RMSE} & = 23.47\% \\ \text{Max. abs. error} & = 13.36, \end{cases}$$

where Norm. RMSE is the RMSE normalized by the L^2 norm of reference prices, and Max. abs. error is the maximal absolute error. We see that the normalized RMSE is high, maybe because the put option price is well-known to be more sensitive (than a call option) to boundary conditions that are discarded here.

To provide further visual insight, we present in Figure 3 the surfaces (with respect to the assets spot prices) of the predicted put prices (DPDE solution), the reference put prices (Exact solution), the absolute errors[3], the arbitrage bound $\hat{u}_\lambda$, and the estimated residual[4] v solution of the residual PDE (5). Notice that $t \in [0, T]$ and $\mu \in P$ are set to fixed values (see notebook for more details), and only assets spot prices are varying. The plots demonstrate the structural accuracy of the DPDE model, with relatively small absolute error.

[1] accessible at github.com/LWunderlich/DeepPDE.
[2] accessible at Nathan-Sanglier/M2MO-ML-Finance.
[3] Absolute error = |DPDE solution − Exact solution|.
[4] Estimated residual = DPDE solution − Arbitrage bound.

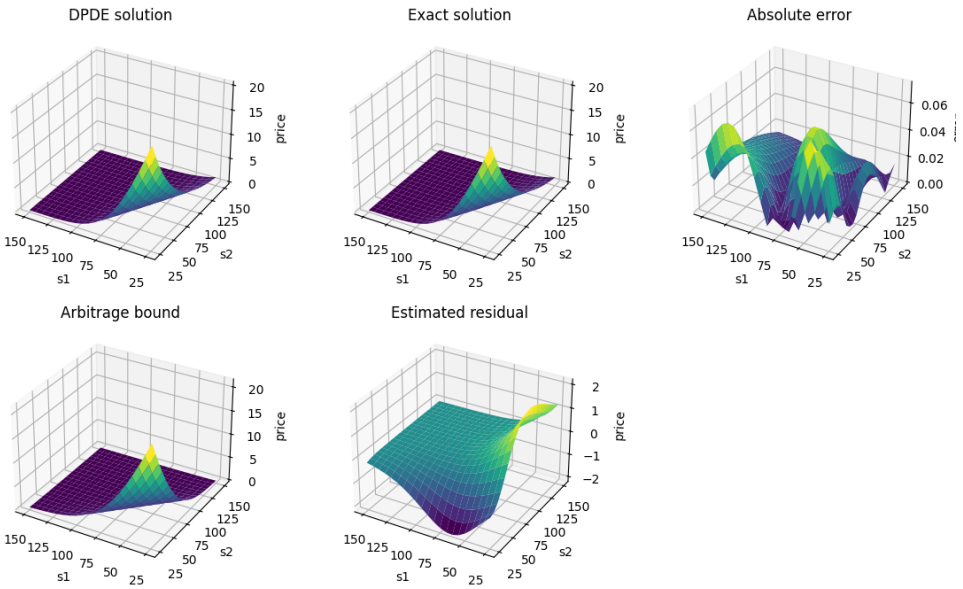

Figure 3: Put option. Surfaces evaluated at fixed time and parameters. Domain of interest smaller than computational domain.

In Appendix A, we provide similar numerical results for the call option, and check that we recover the put option price using the DPDE call price and the call-parity.

4 Limitations of the Method and Extensions

4.1 Possible Limitations

One of the main limitations of the Deep Parametric PDE method as presented in [GW22] lies in the absence of explicit boundary conditions in the loss function, as the authors set $c_{bc} = 0$ in practice. In order to reduce the sensitivity to boundaries in their results they have taken a much smaller domain of interest compared to the computational domain. If one sets a computational domain identical to the domain of interest (eg. large domain of interest, where we cannot afford to take a larger computational domain), then the performance would be degraded near boundaries. Such experiment is presented in Figure 4, where we see that the absolute error is very high on the edges of the assets spot prices domain compared to Figure 3. This limitation remains to investigate further with extensive robustness checks.

Moreover, the choice of network architecture (number of layers, neurons, or highway blocks) has a significant impact on performance. However, there is no theoretical guidance for this selection. As a result, the training process relies on costly manual tuning of hyperparameters, which affects reproducibility and robustness.

Finally, it has been shown in [GW22] that not including the arbitrage bound information in the PDE significantly affects performance of the method, and it would be preferable to use the Deep Kolmogorov method of [BDG20]. This is an important limitation as we may think of options other than bakset ones for which arbitrage bound is not easily available.

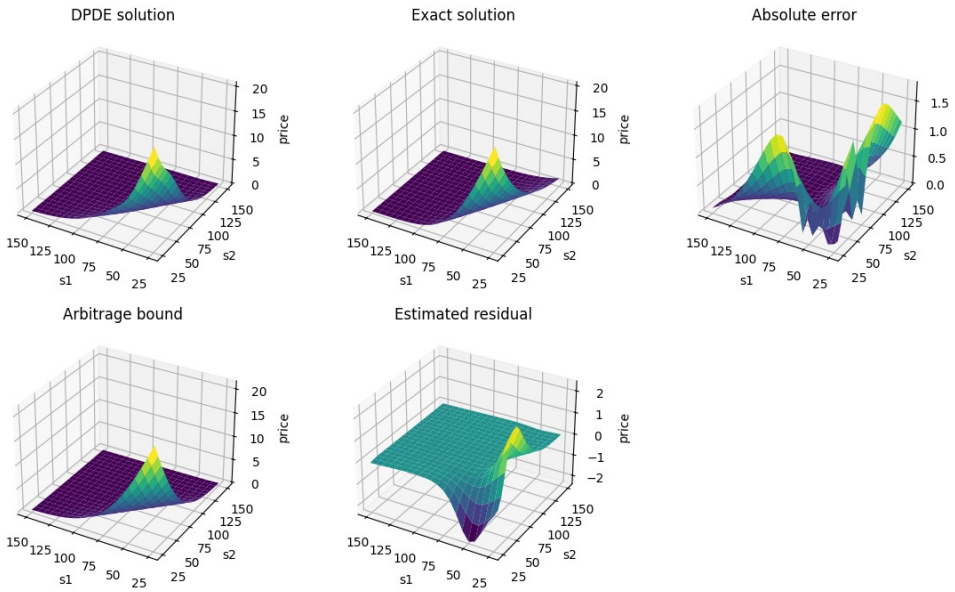

Figure 4: Put option. Surfaces evaluated at fixed time and parameters. Domain of interest identical to computational domain.

4.2 Possible Extensions

Based on the previous remarks, a natural extension would be to explicitly reintegrate boundary terms into the loss function (by setting $c_{bc} = 1$), possibly with a carefully chosen regularization to avoid numerical instabilities.

One major advantage of the method is also its differentiability with respect to both inputs and parameters. This property allows for the direct computation of Greeks (sensitivities) via automatic differentiation. The sensitivities that appear in the PDE (1) are usually well calibrated, but it is not the case for the other sensitivities. These quantities could be added to the loss function as regularization terms to improve reliability of the solution partial derivatives.

Finally, the DPDE framework could be generalized to other financial products such as American options and more exotic options. However, the PDE structure must be adapted accordingly to incorporate features such as early exercise, path-dependency, or jump processes.

5 Conclusion

In this project, we provided a critical review and practical implementation of the Deep Parametric PDE (DPDE) method introduced in [GW22], within the context of option pricing. After presenting the theoretical background and comparing the method to other existing PDE-solving approaches, we implemented the DPDE framework for the pricing of European basket options under the multivariate Black-Scholes model.

Our numerical experiments confirm that the method provides reasonably accurate results for basket put options, even if we see the limitation of the method in regions where boundary conditions play a critical role, with the normalized RMSE exceeding 20%. These findings highlight a trade-off between generality and precision in the DPDE method. While it offers an elegant meshfree solution for high-dimensional parametric PDEs, its effectiveness is constrained by architectural choices and training stability, and the lack of boundary supervision.

Overall, this work demonstrates both the promise and current challenges of deep learning-based PDE solvers in quantitative finance, and opens paths for further research on neural networks methods tailored to financial applications.

References

[ACD10] Yves Achdou and Italo Capuzzo-Dolcetta. Mean field games: Numerical methods. SIAM Journal on Numerical Analysis, 48(3), 2010.

[BD20] Olivier Bokanowski and Kristian Debrabant. Backward differentiation formula finite difference schemes for diffusion equations with an obstacle term. IMA Journal of Numerical Analysis, 2020.

[BdBS+24] Lise Le Boudec, Emmanuel de Bezenac, Louis Serrano, Ramon Daniel Regueiro-Espino, Yuan Yin, and Patrick Gallinari. Learning a neural solver for parametric pde to enhance physics-informed methods, 2024.

[BDG20] Julius Berner, Markus Dablander, and Philipp Grohs. Numerically solving parametric families of high-dimensional kolmogorov partial differential equations via deep learning, 2020.

[BST17] Christian Bayer, Markus Siebenmorgen, and Raul Tempone. Smoothing the payoff for efficient computation of basket option prices, 2017.

[GW22] Kathrin Glau and Linus Wunderlich. The deep parametric pde method and applications to option pricing. Applied Mathematics and Computation, 432, 2022.

[HJE18] Jiequn Han, Arnulf Jentzen, and Weinan E. Solving high-dimensional partial differential equations using deep learning. Proceedings of the National Academy of Sciences, 115, 2018.

[HPW20] Côme Huré, Huyên Pham, and Xavier Warin. Deep backward schemes for high-dimensional nonlinear pdes, 2020.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. Neural Computation, 9, 1997.

[KB17] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[LK90] Hyuk Lee and In Seok Kang. Neural algorithm for solving differential equations. Journal of Computational Physics, 91(1), 1990.

[LLF98] Isaac Lagaris, Aristidis Likas, and Dimitrios Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. IEEE Transactions on Neural Networks, 9, 1998.

[Pö20] Christian Pötz. Function approximation for option pricing and risk management Methods, theory and applications. PhD thesis, School of Mathematical Sciences Queen Mary, University of London, 2020.

[RPK19] M. Raissi, P. Perdikaris, and G.E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. Journal of Computational Physics, 378, 2019.

[SGS15] Rupesh Kumar Srivastava, Klaus Greff, and Jurgen Schmidhuber. Training very deep networks. CoRR, abs/1507.06228, 2015.

[SS18] Justin Sirignano and Konstantinos Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. Journal of Computational Physics, 375, 2018.

[SVSS21] Marc Sabate Vidales, David Siska, and Lukasz Szpruch. Unbiased deep solvers for linear parametric pdes. Applied Mathematical Finance, 28(4), 2021.

Appendices

A Appendix - Call Option

In Figure 5, we show (left) a scatter plot of reference prices against our predicted prices for randomly sampled values of (t, x, μ) in a subspace of $[0, T] \times \Omega \times P$. We remark that the predicted prices are close to the reference ones, but often over-valued. Moreover, we show (right) a scatter plot of the predicted prices absolute errors against these predicted prices. We see that the absolute errors are highly dispersed, meaning that the neural network may be overfitting.

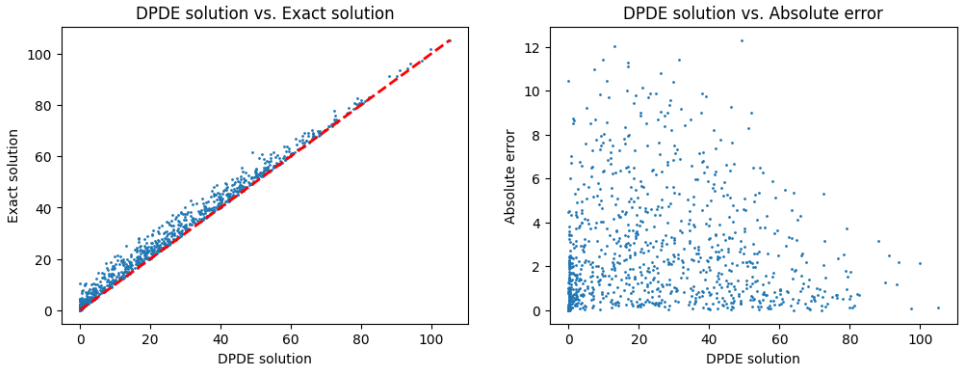

Figure 5: Call option. Predicted and reference prices for randomly sampled points.

Aggregating across these randomly sampled points, we obtain the following metrics:

$$\begin{cases} \text{RMSE} & = 3.49 \\ \text{Norm. RMSE} & = 9.67\% \\ \text{Max. abs. error} & = 13.85. \end{cases}$$

We see that the normalized RMSE is smaller than for the put studied in Section 3.2. In Figure 6, we present the surfaces of the predicted call prices, the reference call prices, the absolute errors, the arbitrage bound, and the estimated solution of the residual PDE. It demonstrates the accuracy of the DPDE model, with small absolute error.

In Figure 7, we compare the predicted DPDE put price (DPDE solution), and the put price obtained by predicted DPDE call price and call-put parity (DPDE Parity solution). We see (Models variation) that the two solutions are close to each other, highlighting that the method is adapting well to each payoff. We also provide the error of the DPDE solution and the DPDE Parity solution against reference prices.

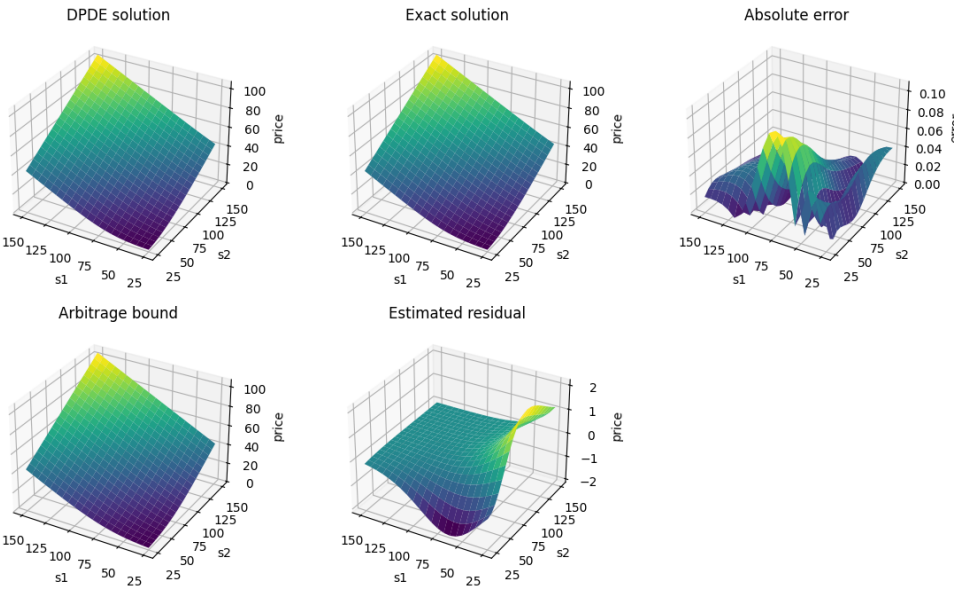

Figure 6: Call option. Surfaces evaluated at fixed time and parameters. Domain of interest smaller than computational domain.

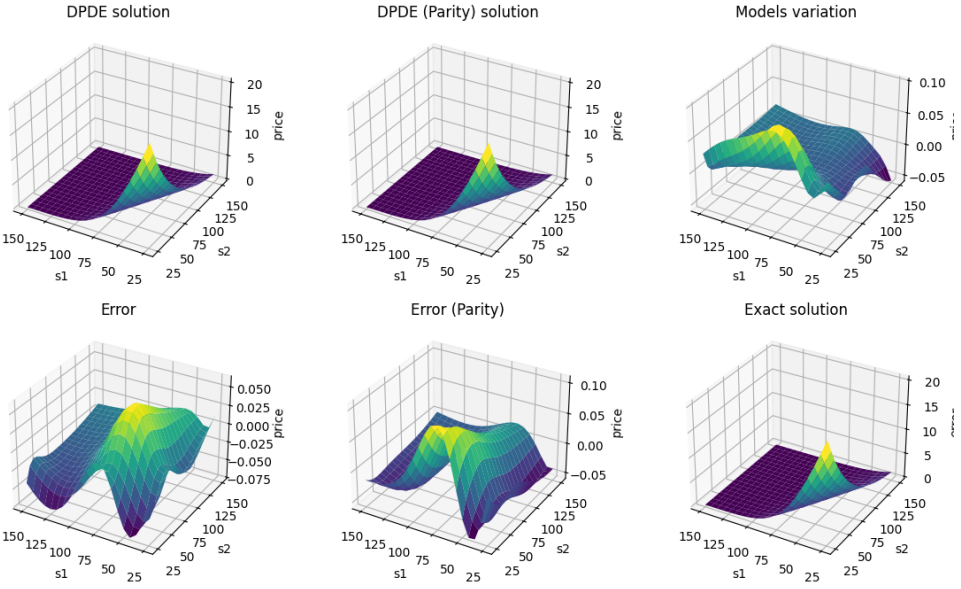

Figure 7: Put option obtained by call DPDE and call-put parity. Surfaces evaluated at fixed time and parameters. Domain of interest smaller than computational domain.