Numerical Methods for High-dimensional Option Pricing Problems

Zewen Shen

Dec, 2019

Abstract

Due to the curse of dimensionality, the pricing of multi-asset options becomes a longstanding computational challenge. Extensive studies have focused on the use of Monte Carlo methods with variation reduction techniques and the low-discrepancy method. However, due to the randomness of Monte Carlo methods, the standard deviation of approximations can be large when the number of simulations is insufficient. More importantly, the computation of hedge parameters is expensive with Monte Carlo methods. This article presents a Fourier transform method that is more efficient than Monte Carlo methods when the number of risk factors is less than 8, along with the deep Galerkin method that outperforms Monte Carlo methods in American option pricing. Additionally, we proved an error bound of the deep Galerkin method's approximation of the solution to the Black-Scholes equation through the maximum principle. Detailed analysis, numerical experiments[1] and comparisons are carried out.

***K*eywords** Multi-asset Option Pricing · Monte Carlo Methods · Fast Fourier Transform · Deep Galerkin Method

[1]The source code can be found at https://github.com/ZewenShen/hdp

Acknowledgements

Firstly, I am deeply thankful to my supervisor, Professor Ken Jackson, who strengthened my determination to continue studying numerical analysis and scientific computing. The thesis would have not been possible without his continuous support, endless patience and enlightening suggestions. I would like to thank Professor Christina Christara, who generously gave me her valuable insight into my confusion. Finally, I would like to thank my family for their support and love.

Contents

1 Introduction

Multi-asset options, e.g., basket options, rainbow options, spread options, are popular trading instruments in equity markets nowadays. Various numerical methods can be applied to price multi-asset options. In general, they can be classified into four categories: numerical partial differential equation (PDE) methods, Monte Carlo methods, Fourier transform methods and deep learning methods. While the first three traditional methods are the most well-studied in both academia and industry, deep learning methods have also received lots of attention in academia since 2017 and the two most influential ones are the deep Galerkin method (DGM) [1] and the deep BSDE method [2].

Although we regarded numerical PDE methods and deep learning methods as two different methods in the last paragraph, in fact, the deep learning method can also be classified as a meshfree numerical PDE method since it aims to solve the Black-Scholes equation using neural networks. One of the biggest differences between the DGM and the deep BSDE method is that the deep BSDE method requires the PDE to satisfy the condition of Feynman–Kac formula, while the DGM doesn't have such a restriction. In contrast, the deep BSDE method requires less amount of computation in every iteration of training since only the gradients are needed in the calculation of the loss function, whereas the DGM also requires the Hessian. In this paper, we decide to mainly focus on the deep Galerkin method.

We aimed to analyze the advantage and the disadvantage of Monte Carlo methods, the FFT-based convolution method and the deep Galerkin method when they are applied to high-dimensional option pricing problems, i.e., the number of underlying assets is between 4 and 7. The reason why high-dimensional problems are particularly hard is due to the *curse of dimensionality*: A five-dimensional problem with 32 grid points per coordinate will result in over 33 million grid points in total. In this case, a regular numerical PDE method will introduce a 33 million $\times$ 33 million sparse linear system, which typically cannot be efficiently solved. There are certain types of techniques that can alleviate the curse of dimensionality for numerical PDE methods, e.g., the sparse grid technique [3], but we are not going to investigate it. Although the Fourier transform methods also suffer from the *curse of dimensionality*, the computation using 33 million grid points is still a reasonable amount of work thanks to the invention of the fast Fourier transform (FFT).

The paper is organized as follows: In section 2, we introduce the background of the multi-asset option pricing problems, including both the PDE framework and the probabilistic framework. During the discussion of the probabilistic framework, we will also present Monte Carlo methods for simulating correlated assets, including the low-discrepancy method and two variance reduction techniques. In section 3, we derive the formulation of the N-dimensional FFT convolution method (Conv method) and the approximation of hedge parameters following R. Lord, C. C. W. Leentvaar's work in [4] and [5], respectively. Introduction to the deep Galerkin method and analysis of its error bound of the approximation of option prices is carried on in section 4. Additionally, we compare the advantages and disadvantages of the deep Galerkin method and the deep BSDE method. Finally, numerical experiments, including the pricing of basket European options with different terminal payoff functions and American options, are conducted to compare the three methods introduced in this paper.

2 Multi-Asset Option Pricing Frameworks and Monte Carlo Methods

2.1 The PDE Framework

The Black-Scholes equation was introduced by F. Black and M. Scholes in [6], which first established the connection between PDEs and option pricing. However, in the original paper, only single-asset options without dividends are considered. Therefore, we derive the multi-asset Black-Scholes equation with dividend-paying by following the similar delta hedging idea presented in [6].

Consider a portfolio consisting of one option and N underlying assets. Assuming that S_i follows a geometric Brownian motion for $i = 1 \dots N$. Therefore,

$$dS_i = (\mu_i - q_i)S_i d\tau + \sigma_i S_i dW_i, \tag{2.1}$$

where μ_i is the continuous drift rate, q_i the continuous dividend rate and σ_i the instantaneous volatility. The N Wiener processes are correlated according to

$$dW_i dW_j = \rho_{ij} d\tau \tag{2.2}$$

where $\rho_{i,j}$ is the i, j-th entry of the correlation matrix ρ. We can immediately derive that

$$dS_i dS_j = \sigma_i \sigma_j S_i S_j \rho_{ij} d\tau + \mathcal{O}(d\tau^{1.5}). \tag{2.3}$$

Let $V(\vec{S}, \tau)$ represent the value of the option. Then, the total value of the portfolio Π can be written as

$$\Pi = V - \sum_{i=1}^{N} \Delta_i S_i \tag{2.4}$$

where $-\Delta_i$ represents the shares of each asset in the portfolio, which will be specified later. Assuming that Δ_i is fixed during the period dt, the jump in Π during this time is

$$d\Pi = dV - \sum_{i=1}^{N} \Delta_i dS_i - \sum_{i=1}^{N} \Delta_i q_i S_i d\tau. \tag{2.5}$$

Note that the last term in (2.5) is resulted from the dividend.

Then, by applying Itô's Lemma to V, (2.5) becomes

$$d\Pi = \Big(\frac{\partial V}{\partial \tau} d\tau + \sum_{i=1}^{N} \frac{\partial V}{\partial S_i} dS_i + \frac{1}{2} \sum_{i,j=1}^{N} \frac{\partial^2 V}{\partial S_i \partial S_j} dS_i dS_j\Big) - \sum_{i=1}^{N} \Delta_i dS_i - \sum_{i=1}^{N} \Delta_i q_i S_i d\tau. \tag{2.6}$$

Notice that we can eliminate all the stochastic terms that contain dW by setting $\Delta_i = \frac{\partial V}{\partial S_i}$. Then by the no-arbitrage principle, the expected return of this portfolio should be

$$d\Pi = r\Pi d\tau, \tag{2.7}$$

where r is the continuous risk-free interest rate. By combining (2.6), (2.7) and (2.3), we have

$$r(V - \sum_{i=1}^{N} \frac{\partial V}{\partial S_i} S_i) d\tau = \frac{\partial V}{\partial \tau} d\tau + \frac{1}{2} \sum_{i,j=1}^{N} \frac{\partial^2 V}{\partial S_i \partial S_j} \sigma_i \sigma_j S_i S_j \rho_{ij} d\tau - \sum_{i=1}^{N} \frac{\partial V}{\partial S_i} q_i S_i d\tau. \tag{2.8}$$

By rearranging the terms, we finally derive the N-dimensional Black-Scholes equation:

$$\frac{\partial V}{\partial \tau} + \frac{1}{2} \sum_{i,j=1}^{N} \sigma_i \sigma_j S_i S_j \rho_{ij} \frac{\partial^2 V}{\partial S_i \partial S_j} + \sum_{i=1}^{N} (r - q_i) S_i \frac{\partial V}{\partial S_i} - rV = 0 \tag{2.9}$$

with the terminal condition

$$V(\vec{S},T) = \Phi(\vec{S}), \tag{2.10}$$

where $V : [0,+\infty]^N \to R$ and $\Phi(\vec{S})$ is the payoff function.

Remark. In our derivation, the correlated asset prices are assumed to follow geometric Brownian motions, which doesn't perfectly capture the behaviour of asset prices in real world and causes the well-known flaw *volatility smile*. Mathematicians proposed lots of other stochastic processes that better depict the asset price movement, e.g., Lévy processes, but in the mean while, more difficult PDEs are introduced. For example, an option price whose underlying asset follows the Lévy process will be a solution to an integro-differential equation. See [7] for details of a numerical method that aims to solve the integro-differential equation.

In the American-style option setting, the owner of the option has the right to exercise the option at any time, thus (2.9) becomes a free boundary PDE:

$$\begin{aligned}
0 &= \frac{\partial V}{\partial \tau} + \frac{1}{2}\sum_{i,j=1}^{N} \sigma_i\sigma_j S_i S_j \rho_{ij} \frac{\partial^2 V}{\partial S_i \partial S_j} + \sum_{i=1}^{N}(r-q_i)S_i\frac{\partial V}{\partial S_i} - rV, \quad \forall\, \{(\vec{S},t) : V(\vec{S},t) > \Phi(\vec{S})\};\\
V(\vec{S},t) &\geq \Phi(\vec{S}), \quad \forall\, (\vec{S},t);\\
V(\vec{S},t) &\in C^1(R^N \times R^+), \quad \forall\, \{(\vec{S},t) : V(\vec{S},t) = \Phi(\vec{S})\};\\
V(\vec{S},T) &= \Phi(\vec{S}).
\end{aligned} \tag{2.11}$$

When the dimension is low ($1 \sim 3$), one can efficiently solve it using the penalty method or the operator splitting method. However, due to the curse of dimensionality, these techniques are too inefficient when applied to higher-dimensional problems.

2.2 The Probabilistic Framework

By the Feynman-Kac formula, the solution to the Black-Scholes equation at time t can be expressed as the discounted payoff of an option whose strike date is T given the stock price at time t under the risk-neutral measure. In the other words,

$$V(t,\boldsymbol{\vec{S}}(t)) = e^{-r(T-t)}\mathbb{E}^Q\Big[V(T,\boldsymbol{\vec{S}}(T)) \mid \boldsymbol{\vec{S}}(t)\Big] = e^{-r(T-t)}\int_{\mathbb{R}^d} V\big(T,\boldsymbol{\vec{S}}(T)\big) f\big(\boldsymbol{\vec{S}}(T) \mid \boldsymbol{\vec{S}}(t)\big) d\boldsymbol{\vec{S}}(t). \tag{2.12}$$

Therefore, we can simulate the price movements of the basket of d underlying assets using a system of stochastic differential equations (SDEs) and compute its discounted payoff to get the option price. Under the Black-Scholes framework, the system of SDEs follows a multivariate geometric Brownian motion whose drift rate equals $r - \vec{q}$ (due to the risk-neutral measure), which can be written as

$$\begin{aligned}
\frac{dS_i(t)}{S_i(t)} &= (r-q_i)dt + \sigma_i dW_i, \quad i = 1,\dots,d,\\
\boldsymbol{\vec{S}}(t_0) &= \vec{S}_0 \in R^d,
\end{aligned} \tag{2.13}$$

where r, q_i and σ_i are a continuous risk-free interest rate, a continuous dividend rate and the instantaneous volatility respectively. Let L be the Cholesky factorization of the correlation matrix, i.e., $\rho = LL^T$, then,

$$dW_i(t) \sim \sum_{j=1}^{d} L_{ij}\phi_j\sqrt{dt}, \tag{2.14}$$

where $\phi_i \sim \mathcal{N}(0,1)$, i.e., each W_i is a standard one-dimensional Wiener process and $W_i(t)$ and $W_j(t)$ have correlation ρ_{ij}.

By means of Ito's lemma, the solution to the SDE is given by

$$S_i(t) = S_i(t_0)e^{(r-q_i-\frac{\sigma_i^2}{2})(t-t_0)+\sigma_i\sqrt{t-t_0}\sum_{j=1}^{d} L_{ij}\phi_j} \tag{2.15}$$

or equivalently

$$s_i(t) = \ln S_i(t) = s_i(t_0) + (r - q_i - \frac{\sigma_i^2}{2})(t-t_0) + \sigma_i\sqrt{t-t_0}\sum_{j=1}^{d} L_{ij}\phi_j, \tag{2.16}$$

where $s_i(t_0) = \ln S_i(t_0)$. Define $d_i(t) = s_i(t) - s_i(t_0)$, we get

$$d_i(t) = s_i(t) - s_i(t_0) \sim \mathcal{N}\big((r - q_i - \frac{\sigma_i^2}{2})(t-t_0), \sigma_i^2(t-t_0)\sum_{j=1}^{d} L_{ij}^2\big). \tag{2.17}$$

We can also write the solution in a more compact way:

$$\vec{\boldsymbol{d}} \sim \mathcal{N}(\vec{\boldsymbol{\mu}}, \boldsymbol{\Sigma}), \tag{2.18}$$

where $\vec{\boldsymbol{\mu}} = (r - \vec{\boldsymbol{q}} - \frac{1}{2}\vec{\boldsymbol{\sigma}}^2)(t-t_0), \boldsymbol{\Sigma}$ is the covariance matrix with $\boldsymbol{\Sigma}_{jk} = \rho_{jk}\sigma_j\sigma_k(t-t_0)$.

Remark. By (2.15), given a vector of initial stock prices, we can directly simulate the price of the stock at time $t = T$, which saves large amount of computations during the European option pricing.

In the American-style option pricing, the most traditional approach is the least square Monte Carlo method (LSMC) proposed in [8], which requires one to use (2.15) recursively to get the stock prices at every time step. After that, one first computes the payoff at the terminal time t_N and then discount it to the second last time step t_{N-1}. By doing a least square regression, one can compute the continuation value at t_{N-1}. Finally, one compares the continuation value and the strike value to get the American option price at t_{N-1}. Applying this algorithm back and forth to all the time steps, finally we will get the American option price at t_0. To sum up, one uses the price of Bermudan options with a large number of exercise times to approximate the price of American options.

Remark. LSMC uses the degree-χ monomial basis during the least squares regression:

$$\phi_\chi(\boldsymbol{x}) = \{x_1^{a_1}x_2^{a_2}\dots x_d^{a_d} | a_1 + a_2 + \cdots + a_d \le \chi\}. \tag{2.19}$$

If we pick χ to be close to d, the regression will be both overfitting and time-consuming. Instead, we pick $\chi << d$ in our experiments, then the number of the monomial basis becomes $\begin{pmatrix} d+\chi \\ d \end{pmatrix} \approx \frac{1}{\chi!}d^\chi$ and the total computational complexity is $\mathcal{O}(NM(\frac{1}{\chi!}d^\chi)^2) = \mathcal{O}(NMd^{2\chi})$, where N, M are the number of time steps and the number of paths simulated, respectively.

2.3 Improving Monte Carlo Methods in Multi-Asset Option Pricing Problems

Although we mentioned in the last subsection that we can efficiently simulate the price of stock at time $t = T$ by the analytical solution to the SDE, some improvements are still necessary since the convergence rate of the vanilla Monte Carlo method is $\mathcal{O}(\frac{1}{\sqrt{N}})$, where N is the number of simulations. Here we are first going to introduce the low-discrepancy method, then, we derive the N-dimensional method of antithetic variates and the control variates method.

2.3.1 Low-Discrepancy Method

To present the low-discrepancy method, we need to figure out how to precisely describe the uniformity of a point set. The following is the definition of discrepancy:

Definition 2.1. Given a collection $\mathcal{A}$ of (Lebesgue measurable) subsets of $[0,1)^d$, the discrepancy of a set $P = \{x_1, \dots, x_N\}$ relative to $\mathcal{A}$ is defined as

$$D(P;\mathcal{A}) = \sup_{B\in\mathcal{A}} \left| \frac{\#\{x_i \in B\}}{N} - vol(B) \right|, \tag{2.20}$$

where vol represents the volume of B.

Remark. In other words, the discrepancy can be seen as the maximum point density deviation of a uniform set. Note that the normal distribution instead of the uniform distribution is needed in the simulation for asset prices, one can use the Box-Muller method to transform the uniform low-discrepancy sequence into a normal low-discrepancy sequence. The most well-known low-discrepency sequences are the van der Corput sequence, the Halton sequence, the Sobol sequence, etc. Before we illustrate the mechanics of them, we are first going to introduce the fundamental operation of constructing these sequence.

Definition 2.2 (Radical Inversion)**.** Let i be a positive integer, then, we can convert i to base N by

$$i = \sum_{l=1}^{M} a_l(i) N^{l-1}. \tag{2.21}$$

Given a $M \times M$ generator matrix C, the radical inversion of the integer i given base N and the generator matrix C is written as

$$\Phi_{N,C}(i) = \begin{pmatrix} N^{-1} & \dots & N^{-M} \end{pmatrix} \left[C \begin{pmatrix} a_1(i) \\ \vdots \\ a_M(i) \end{pmatrix} \right]. \tag{2.22}$$

Remark. The generator matrix is a matrix that guarantees the generation of robust quasi-random numbers and we skip the discussion of criterion of a good generator matrix in this paper.

To gain a better insight of this definition, consider the van der Corput sequence whose generator matrix is an identity matrix, for simplicity, let the base $N = 2$. Accordingly, (2.22) becomes

$$\Phi_{2,\mathcal{I}}(i) = \begin{pmatrix} 2^{-1} & \dots & 2^{-M} \end{pmatrix} \begin{pmatrix} a_1(i) & \dots & a_M(i) \end{pmatrix}^T. \tag{2.23}$$

Then,

$$\Phi_{2,\mathcal{I}}(1) = b'0.1 = 1/2,\ \Phi_{2,\mathcal{I}}(2) = b'0.01 = 1/4,\ \Phi_{2,\mathcal{I}}(3) = b'0.11 = 3/4,\ \Phi_{2,\mathcal{I}}(4) = b'0.001 = 1/8,\ \dots \tag{2.24}$$

We can see that this sequence fills $[0,1]$ quite evenly. However, we can't simply use the van der Corput sequence in our Monte Carlo simulations, since each of the sample requires paths of N correlated asset prices, while the van der Corput sequence only generates 1-D samples. This motivates the construction of the Halton sequence and the Sobol sequence.

Definition 2.3 (Halton Sequence)**.** Let $p_1, p_2, \dots, p_N$ be coprime numbers. Then, the Halton sequence is defined by

$$X(i) = (\Phi_{p_1,\mathcal{I}}(i), \Phi_{p_2,\mathcal{I}}(i), \dots, \Phi_{p_N,\mathcal{I}}(i)) \tag{2.25}$$

Definition 2.4 (Sobol Sequence)**.** Let $C_1, C_2, \dots, C_N$ be different generator matrices. Then, the Sobol sequence is defined by

$$X(i) = (\Phi_{2,C_1}(i), \Phi_{2,C_2}(i), \dots, \Phi_{2,C_N}(i)) \tag{2.26}$$

Remark. The Halton sequence is just a trivial generalization of the van der Corput sequence, consequently, it's also of low discrepancy. One of the disadvantages of it is that, there exists some correlation between the points in the sequence when p_N is large. Such deficiency can be alleviated by the scrambling techniques, see [9] for details. On the contrary, the Sobol sequence is free of correlations between the points when good generator matrices are used. More importantly, all the radical inversion use 2 as a base, which allows us to use bit operations to get a fast implementation of the Sobol sequence generator. Given these arguments, we are going to use the Sobol sequence in our numerical experiments.

2.3.2 Antithetic Variates

Another variation reduction technique that is easy to implement is the method of antithetic variates. Its application in option pricing has been discussed thoroughly in [10]. In this section, we plan to illustrate the core idea of it.

In the usual Monte Carlo method, a sequence of independent observations of discounted payoff values $Y_1, Y_2, \dots, Y_n$ are generated. Instead, the method of antithetic variates requires one to generate a sequence of pairs of observations $(Y_1, \tilde{Y_1}), (Y_2, \tilde{Y_2}), \dots, (Y_n, \tilde{Y_n})$, where the pairs are independent of each other and Y_i and $\tilde{Y_i}$ have a negative correlation. Before proving that the variance gets reduced under this setting, we first write down the antithetic variates estimator:

$$\hat{Y}_{AV} = \frac{1}{2n}\Big(\sum_{i=1}^{n} Y_i + \sum_{i=1}^{n} \tilde{Y_i}\Big) = \frac{1}{n}\sum_{i=1}^{n}\Big(\frac{Y_i + \tilde{Y_i}}{2}\Big). \tag{2.27}$$

Without the method of antithetic variates, the estimator is

$$Y = \frac{1}{2n}\Big(\sum_{i=1}^{2n} Y_i\Big) \tag{2.28}$$

Thus, the variance gets reduced if

$$\text{Var}\Big[\frac{1}{n}\sum_{i=1}^{n}(\frac{Y_i + \tilde{Y_i}}{2})\Big] < \text{Var}\Big[\frac{1}{2n}(\sum_{i=1}^{2n} Y_i)\Big], \tag{2.29}$$

We assume that $Y_i, \hat{Y_i}$ have the same distribution. Therefore, (2.29) holds if the following equation holds:

$$\text{Var}\big[Y_i + \hat{Y_i}\big] < 2\text{Var}\big[Y_i\big]. \tag{2.30}$$

Since the variance on the left can be written as

$$\text{Var}\big[Y_i + \hat{Y_i}\big] = \text{Var}\big[Y_i\big] + \text{Var}\big[\hat{Y_i}\big] + 2\text{Cov}\big[Y_i, \hat{Y_i}\big] = 2\text{Var}\big[Y_i\big] + 2\text{Cov}\big[Y_i, \hat{Y_i}\big] < 2\text{Var}\big[Y_i\big], \tag{2.31}$$

both (2.30) and (2.29) are satisfied, i.e., the method of antithetic reduces the variance.

After showing that the negative correlation is a necessary condition for the variance reduction, our next task is to generate the negatively correlated pairs $(Y_i, \hat{Y_i})$. We hope that the negative dependence in the inputs (in our case, inputs are $\vec{\phi}$ and $-\vec{\phi}$ in (2.33)) will produce negative correlation between the outputs. In simulating geometric Brownian motions, we are going to generate the Y_i by vanilla Monte Carlo methods, then construct the $\hat{Y_i}$ based on Y_i. In general, the Y_i is given by

$$Y_i = e^{-rT}\Phi(S_1(T), \dots, S_d(T)), \tag{2.32}$$

where

$$\{S_i(T)\}(\phi_1, \phi_2, \ldots, \phi_d) = S_i(0) \exp\Big\{(r - q_i - \frac{\sigma_i^2}{2})(T - t_0) + \sigma_i \sqrt{T - t_0} \sum_{j=1}^{d} L_{ij}\phi_j\Big\}, \tag{2.33}$$

and ϕ_i, Φ denote the standard normal distribution and the terminal payoff function, respectively.

Glasserman's book [10] states that, given negatively correlated input pairs, the monotonicity of the mapping from inputs to outputs will ensure the negative correlation between outputs. For the common terminal payoff functions, for examples, $\Phi(\vec{S}(T)) = \max((\sum_{i=1}^{d} S_i(T))/d - K, 0)$ and $\Phi(\vec{S}(T)) = \max((\prod_{i=1}^{d} S_i(T))^{1/d} - K, 0)$, Φ is a non-decreasing function with respect to $S_i(T)$. However, $S_i(T)$ is not necessarily a monotonic function with respect to ϕ_i since the matrix L (outputted by the Cholesky decomposition of the correlation matrix) may have negative entries. Therefore, the method of antithetic variates may give a higher variance than vanilla Monte Carlo simulations. Nevertheless, in our numerical experiments. we will demonstrate that the method is usually effective. A. Hirsa also pointed out that typically the payoff function preserves the effects of the anti-correlation and reduces the variance in [11].

2.3.3 Control Variates

Although the method of antithetic variates is easy to implement and saves half of the sampling time, it usually will not provide too much variance reduction. Furthermore, its compatibility with the Sobol sequence is weak since the antithetic pairs can be sampled only after generating the Sobol sequence, which impairs the discrepancy of the sequence. Therefore, we are going to introduce the control variates methods, which not only provide more variance reduction but also is compatible with the Sobol sequence. Again, P. Glasserman has discussed the application of the control variates method in [10]. We will extract the main idea from it and make more comments on the method.

The control variates method aims to improve the estimation of an unknown quantity using the errors in estimates of known quantities. Here is an intuitive interpretation of this method: say we are simulating the paths of d asset prices under the geometric Brownian motion with spot prices $\vec{S}(0)$. In this scenario, $\vec{S}(T)$ is regarded as the known quantities since given the analytical solution to the SDEs (2.15), we can derive that

$$\mathbb{E}\big[\vec{S}(T)\big] = e^{(r-\vec{q})T}\vec{S}(0). \tag{2.34}$$

Therefore, by comparing the simulated $\vec{S}(T)$ with its expectation, we can estimate how biased our samples are and adjust the estimation of the option value correspondingly. Next, we formalize the arguments above mathematically.

Let $X_i = (X_i^{(1)}, \ldots, X_i^{(d)})^T$ be the simulated terminal price vector in the i-th simulation, Y_i be the discounted payoff given X_i. For $b \in \mathbb{R}^d$, the control variates estimator for Y is

$$\bar{Y}(b) = \bar{Y} - b^T(\bar{X} - \mathbb{E}[X]). \tag{2.35}$$

Obviously, (2.35) is an unbiased estimator since

$$\mathbb{E}[\bar{Y}(b)] = \mathbb{E}[\bar{Y}] - b^T(\mathbb{E}[\bar{X}] - \mathbb{E}[X]) = \mathbb{E}[\bar{Y}] - b^T\vec{0} = \mathbb{E}[\bar{Y}]. \tag{2.36}$$

Assuming that the pairs (X_i, Y_i) for $i = 1, \ldots, n$ are i.i.d. with the covariance matrix

$$\Sigma = \begin{pmatrix} \Sigma_{XX} & \Sigma_{XY} \\ \Sigma_{XY}^T & \sigma_Y^2 \end{pmatrix}, \tag{2.37}$$

where Σ_{XX}, Σ_{XY} are $d \times d$, $d \times 1$ matrices, respectively. σ_Y is the variance of the Y_i. Then the variance of the estimator becomes

$$\text{Var}[Y_i - b^T(X_i - \mathbb{E}[X])] = \sigma_Y^2 - 2b^T\Sigma_{XY} + b^T\Sigma_{XX}b. \tag{2.38}$$

By multivariate calculus, we can show that the variance gets minimized at $b^* = \Sigma_{XX}^{-1}\Sigma_{XY}$. Although in most of the cases, we don't know the exact value of b^*, we can estimate Σ_{XX}, Σ_{XY} using the sample covariance matrix S_{XX}, S_{XY}. Finally, the adjusted option price is given by

$$Y = \bar{Y} - (S_{XX}^{-1}S_{XY})^T(\bar{X} - \mathbb{E}[X]). \tag{2.39}$$

Remark. Since the control variates method does not rely on the sampling algorithm, we can sample the normal distribution using the Sobol sequence instead of the regular random number generator. The numerical experiment in section 5 shows that the combination of the Sobol sequence and the control variates method gives the most accurate approximation of option prices.

2.4 Disadvantages of Monte Carlo Methods

American-Style Options

In the European-style option pricing, we can easily parallelize the simulation to accelerate the computation of option price since the paths simulated by Monte Carlo methods are independent of each other and only the evaluation of the terminal payoff function is needed. However, in the American-style option pricing described at the end of 2.2, not only do we need to simulate the stock prices at each time step, but also compute least square regressions at these steps, which plays the dominant role in the computation. To sum up, it takes much longer time to price multi-asset American options than multi-asset European options using Monte Carlo methods, and we will show this by numerical experiments.

Sensitivity Estimation

The most straightforward method in approximating the sensitivity is to use the finite difference method. For example, to get the delta with a second-order accuracy, we need to compute both $V(S_1, \ldots, S_i + \epsilon, \ldots, S_d, t_0)$ and $V(S_1, \ldots, S_i - \epsilon, \ldots, S_d, t_0)$ by Monte Carlo methods based on a same random seed, such that the delta is given by

$$\frac{V(S_1, \ldots, S_i + \epsilon, \ldots, S_d, t_0) - V(S_1, \ldots, S_i - \epsilon, \ldots, S_d, t_0)}{2\epsilon}. \tag{2.40}$$

Therefore, to compute the delta with respect to the d assets, another $2d$ simulations have to be done aside from the simulation which calculates the option price, which means that the total computation time will increased by $2d$ times.

3 FFT-Based Convolution Method

3.1 Background

The Fourier transform method is first proposed by Carr and Madan in [12]. In the paper, the authors derived the closed-form of the damped European option price on the frequency domain such that one can apply the inverse Fourier transform to get the option prices with respect to N different strike prices in $\mathcal{O}(N \log N)$, which is useful for building the volatility surface. However, there is one particular drawback in Carr and Madan's method: the expression of the option price needs to be multiplied by a damping factor $e^{\alpha}k$ (where k and α are the strike variable and a constant, respectively) such that the damped option price is L^1 integrable and consequently the Fourier transform exists. Although the effect of the damping factor will be cancelled by multiplying the inverse of it, the selection of α will affect the accuracy of numerical integration since the larger the damping factor is, the more oscillatory the integrand becomes. Therefore, we need to optimize our selection of α such that the damped price function is both L^1 integrable and smooth, which introduces an unwanted degree of freedom in our algorithm. Most importantly, the damping factor technique cannot be easily generalized to an N-dimensional case.

In this chapter, we will introduce a transform-based method called the Conv method that is free of all the symptoms stated above. The 1-dimensional case of the Conv method is first proposed by Lord et al. in [4]. Leentvaar et al. extended this method to N-dimensions and parallelized the algorithm to reduce the computation time and memory requirements in [5]. However, there are several errors in their derivations which makes it impossible to implement the N-dimensional Conv method by following the algorithm stated in [5]. Therefore, we will derive the correct algorithm more clearly. Before that, we first introduce some concepts that will be used during the derivation.

Definition 3.1. Multidimensional Fourier Transform
Let $f : R^n \to C$ be an L^1 function. The Fourier transform of $f(\vec{\boldsymbol{x}})$ is the function $\mathcal{F}f(\vec{\boldsymbol{u}})$, defined by

$$\mathcal{F}\{f(\vec{\boldsymbol{x}})\}(\vec{\boldsymbol{u}}) = \int_{\mathbb{R}^n} e^{-i\vec{\boldsymbol{u}}\cdot\vec{\boldsymbol{x}}} f(\vec{\boldsymbol{x}}) d\vec{\boldsymbol{x}}. \tag{3.1}$$

Let $H : R^n \to C$ be an L^1 function. The inverse Fourier transform of a function $H(\vec{\boldsymbol{u}})$ is

$$\mathcal{F}^{-1}\{H(\vec{\boldsymbol{u}})\}(\vec{\boldsymbol{x}}) = (\frac{1}{2\pi})^n \int_{\mathbb{R}^n} e^{i\vec{\boldsymbol{u}}\cdot\vec{\boldsymbol{x}}} H(\vec{\boldsymbol{u}}) d\vec{\boldsymbol{u}}. \tag{3.2}$$

Remark. We are able to show that $\mathcal{F}\mathcal{F}^{-1} = \mathcal{F}^{-1}\mathcal{F} = \mathcal{I}$. And the cross-correlation theorem tells us that

$$\mathcal{F}\{Corr(f,g)\} = \mathcal{F}(f) \circ \mathcal{F}(g)^*,\ \mathcal{F}^{-1}\{Corr(f,g)\} = \mathcal{F}^{-1}(f) \circ \mathcal{F}^{-1}(g)^*, \tag{3.3}$$

where $\circ$ represents element-wise multiplication. In other words, the convolution in the space domain can be seen as the multiplication in the frequency domain.

Although usually we don't know the closed form of probability density functions of the stochastic processes appeared in mathematical finance, we can derive an analytical solution to the characteristic functions of some classes of stochastic processes, e.g., Lévy processes. Here we introduce the definition of the characteristic function.

Definition 3.2. Characteristic Function of a Multivariate Probability Density Function
If $f(\vec{\boldsymbol{X}})$ is the probability density function (PDF) of a d-dimensional random vector $\vec{\boldsymbol{X}}$, then, for $\vec{\boldsymbol{v}} \in \mathbb{R}^d$, $f(\vec{\boldsymbol{X}})$'s inverse Fourier transform is called the characteristic function

$$\Phi(\vec{\boldsymbol{v}}) = \mathcal{F}^{-1}\{f(\vec{\boldsymbol{X}})\} = \int_{-\infty}^{\infty} e^{i\vec{\boldsymbol{v}}\cdot\vec{\boldsymbol{X}}} f(\vec{\boldsymbol{X}}) dX = \mathbb{E}[e^{i\vec{\boldsymbol{v}}^T\vec{\boldsymbol{X}}}] \tag{3.4}$$

Remark. The characteristic function contains all the information given by a PDF, since by performing numerical integration only once, the characteristic function can be recovered to the underlying probability density function and cumulative distribution function using Fourier transform and Fourier transform with a damping factor respectively (for details see [11]). Additionally, an arbitrary number of moments of that distribution is given by

$$\mathbb{E}[X^n] = i^{-n}\Phi^{(n)}(\vec{\mathbf{0}}). \tag{3.5}$$

Note that the characteristic function of the multivariate normal distribution (2.18) is given by

$$\phi_{\vec{\boldsymbol{d}}(t)}(\vec{\boldsymbol{u}}) = exp\Big(i\vec{\boldsymbol{\mu}} \cdot \vec{\boldsymbol{u}} - \frac{1}{2}\vec{\boldsymbol{u}}^T \boldsymbol{\Sigma} \vec{\boldsymbol{u}}\Big). \tag{3.6}$$

Remark. Here we calculate the characteristic function of $\vec{\boldsymbol{d}}(t)$ instead of $\vec{\boldsymbol{s}}(t)$, since in the FFT Conv method, it allows us to put the spot price on the grid point directly, which skips the numerical error caused by the interpolation.

3.2 The Multidimensional Convolution Method

Like all transform-based methods, the convolution method is based on the risk-neutral valuation formula, which is given as follows:

$$V(t, \vec{\boldsymbol{s}}) = e^{-r(T-t)}\mathbb{E}\Big[V(T, \vec{\boldsymbol{y}}) \mid \vec{\boldsymbol{s}}\Big] = e^{-r(T-t)} \int_{\mathbb{R}^d} V\big(T, \vec{\boldsymbol{y}}\big) f\big(\vec{\boldsymbol{y}} \mid \vec{\boldsymbol{s}}\big) d\vec{\boldsymbol{y}}, \tag{3.7}$$

where $\vec{\boldsymbol{s}} = \ln \vec{\boldsymbol{S}}(t_0) - \ln \vec{\boldsymbol{S}}_0, \vec{\boldsymbol{y}} = \ln \vec{\boldsymbol{S}}(T) - \ln \vec{\boldsymbol{S}}_0$ and $V(T, \vec{\boldsymbol{y}})$ represents the value of the option at the maturity date. Note that $\ln \vec{\boldsymbol{S}}_0$ is a constant that represents the log spot price and during the computation of the payoff function, we will need to recover $\vec{\boldsymbol{S}}(T)$ by $\vec{\boldsymbol{S}}_0 exp(\vec{\boldsymbol{y}})$. We take the log price since Fourier transform requires the input function V defined everywhere in $\mathbb{R}^d$. Also, it makes the characteristic function (3.6) of the PDF f more readable.

The key point of the convolution method is that, for stochastic processes which have independent increments (e.g., Lévy processes), the transition density function $f(\vec{\boldsymbol{y}} \mid \vec{\boldsymbol{s}})$ equal to the density of the difference of $\vec{\boldsymbol{y}}$ and $\vec{\boldsymbol{s}}$:

$$f(\vec{\boldsymbol{y}} \mid \vec{\boldsymbol{s}}) = f(\vec{\boldsymbol{y}} - \vec{\boldsymbol{s}}). \tag{3.8}$$

So the option price at time t can be expressed as

$$V(t, \vec{\boldsymbol{s}}) = e^{-r(T-t)} \int_{\mathbb{R}^d} V\big(T, \vec{\boldsymbol{y}}\big) f\big(\vec{\boldsymbol{y}} - \vec{\boldsymbol{s}}\big) d\vec{\boldsymbol{y}}. \tag{3.9}$$

With the following change of variable

$$\vec{\boldsymbol{z}} = \vec{\boldsymbol{y}} - \vec{\boldsymbol{s}}, \tag{3.10}$$

we can write (3.9) as

$$V(t, \vec{\boldsymbol{s}}) = e^{-r(T-t)} \int_{\mathbb{R}^d} V\big(T, \vec{\boldsymbol{s}} + \vec{\boldsymbol{z}}\big) f\big(\vec{\boldsymbol{z}}\big) d\vec{\boldsymbol{z}}, \tag{3.11}$$

which can be seen as a cross-correlation.

Since $V(t, \vec{\boldsymbol{s}})$ is usually not an L^1 function, we are unable to use cross-correlation theorem. Instead, we have to truncate its domain such that the L^1 integrability on $(-\infty, \infty)$ is replaced by L^1 summability on a truncated domain. Note that in [12] and [4], the damping factor technique is applied, s.t., V is L^1 integrable on R, but this only works for the one dimensional case, see [5] for details.

Based on our numerical experiments and the discussions in [13] and [5], truncating $\mathbb{R}^d$ by $\Omega^d = [-5\sqrt{T}\vec{\boldsymbol{\sigma}}, 5\sqrt{T}\vec{\boldsymbol{\sigma}}]$ will still give us accurate results, since the most of the mass of the PDF (3.8) is inside Ω^d. Then, the inverse Fourier transform of V becomes

$$\begin{aligned}
\mathcal{F}^{-1}\Big\{V(t,\vec{\boldsymbol{s}})\Big\}(\vec{\boldsymbol{u}}) &= e^{-r(T-t)}\mathcal{F}^{-1}\Big\{\int_{\Omega^d} V\big(T,\vec{\boldsymbol{s}}+\vec{\boldsymbol{z}}\big)f\big(\vec{\boldsymbol{z}}\big)d\vec{\boldsymbol{z}}\Big\}(\vec{\boldsymbol{u}}) \\
&= e^{-r(T-t)}\int_{\Omega^d} e^{i\vec{\boldsymbol{s}}\cdot\vec{\boldsymbol{u}}}\Big(\int_{\Omega^d} V\big(T,\vec{\boldsymbol{s}}+\vec{\boldsymbol{z}}\big)f\big(\vec{\boldsymbol{z}}\big)d\vec{\boldsymbol{z}}\Big)d\vec{\boldsymbol{s}} \\
&= e^{-r(T-t)}\int_{\Omega^d}\int_{\Omega^d} e^{i(\vec{\boldsymbol{s}}+\vec{\boldsymbol{z}})\cdot\vec{\boldsymbol{u}}}V\big(T,\vec{\boldsymbol{s}}+\vec{\boldsymbol{z}}\big)f\big(\vec{\boldsymbol{z}}\big)e^{-i\vec{\boldsymbol{z}}\cdot\vec{\boldsymbol{u}}}d\vec{\boldsymbol{z}}d\vec{\boldsymbol{s}} \qquad (3.12) \\
&= e^{-r(T-t)}\int_{\Omega^d}\int_{\Omega^d} e^{i\vec{\boldsymbol{y}}\cdot\vec{\boldsymbol{u}}}V\big(T,\vec{\boldsymbol{y}}\big)f\big(\vec{\boldsymbol{z}}\big)e^{-i\vec{\boldsymbol{z}}\cdot\vec{\boldsymbol{u}}}d\vec{\boldsymbol{z}}d\vec{\boldsymbol{y}} \\
&= e^{-r(T-t)}\Big(\int_{\Omega^d} e^{i\vec{\boldsymbol{y}}\cdot\vec{\boldsymbol{u}}}V\big(T,\vec{\boldsymbol{y}}\big)d\vec{\boldsymbol{y}}\Big)\Big(\int_{\Omega^d} f\big(\vec{\boldsymbol{z}}\big)e^{-i\vec{\boldsymbol{z}}\cdot\vec{\boldsymbol{u}}}d\vec{\boldsymbol{z}}\Big) \quad \text{(by Fubini theorem)}
\end{aligned}$$

Therefore, the option price is computed using

$$V(t,\vec{\boldsymbol{s}}) = \mathcal{F}\{\mathcal{F}^{-1}\{V(t,\vec{\boldsymbol{s}})\}\} = e^{-r(T-t)}\mathcal{F}\Big\{\mathcal{F}^{-1}\Big\{V\big(T,\vec{\boldsymbol{y}}\big)\Big\}(\vec{\boldsymbol{u}})\cdot\phi_{\vec{\boldsymbol{d}}(T)}(-\vec{\boldsymbol{u}})\Big\}, \qquad (3.13)$$

where $\mathcal{F}$ and $\mathcal{F}^{-1}$ are defined on two truncated domains respectively.

3.3 Approximations of the Hedge Parameters

The approximated closed form of hedge parameters usually can be obtained easily using the FFT-based method by using the derivative property of the Fourier transform, i.e.,

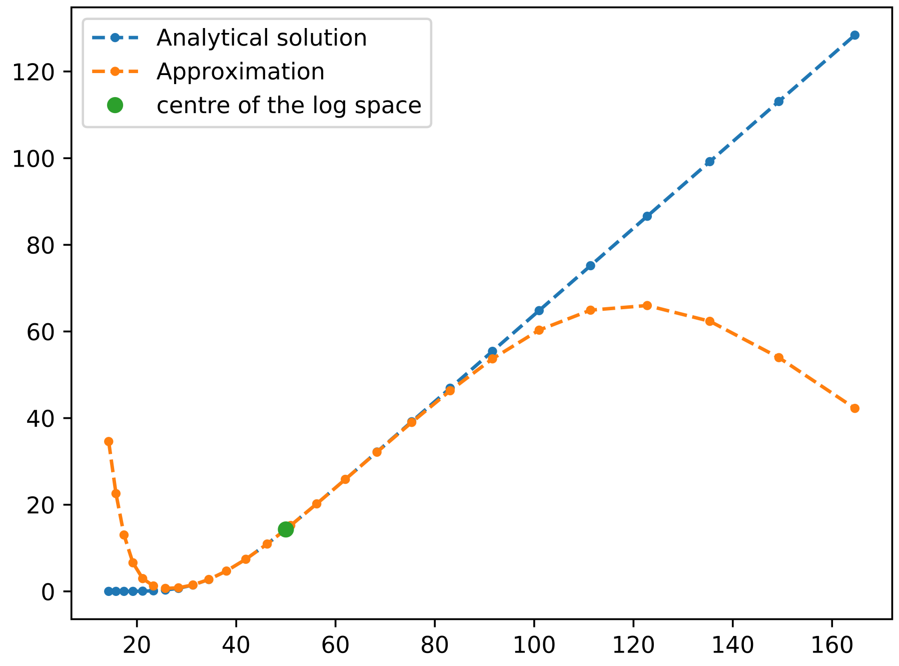
$$\mathcal{F}\Big(\frac{\partial f}{\partial x_j}\Big) = iu_j\mathcal{F}(f), \quad \mathcal{F}^{-1}\Big(\frac{\partial f}{\partial x_j}\Big) = -iu_j\mathcal{F}(f). \qquad (3.14)$$

For example, the delta can be computed by

$$\begin{aligned}
\frac{\partial V(t,\vec{\boldsymbol{s}})}{\partial \boldsymbol{S}_j}\Big|_{\boldsymbol{S}_j(t_0)} &= \frac{1}{\boldsymbol{S}_j(t_0)}\frac{\partial V(t,\vec{\boldsymbol{s}})}{\partial \boldsymbol{s}_j} \quad \text{(by Chain rule)} \\
&= \frac{1}{\boldsymbol{S}_j(t_0)}\mathcal{F}\Big\{\mathcal{F}^{-1}\Big\{\frac{\partial V(t,\vec{\boldsymbol{s}})}{\partial \boldsymbol{s}_j}\Big\}\Big\} \\
&= \frac{1}{\boldsymbol{S}_j(t_0)}\mathcal{F}\Big\{-iu_j\mathcal{F}^{-1}\Big\{V(t,\vec{\boldsymbol{s}})\Big\}\Big\} \quad \text{(by the derivative property of the Fourier transform)} \qquad (3.15) \\
&= -\frac{1}{\boldsymbol{S}_j(t_0)}\mathcal{F}\Big\{iu_j\mathcal{F}^{-1}\Big\{e^{-r(T-t)}\mathcal{F}\Big\{\mathcal{F}^{-1}\Big\{V\big(T,\vec{\boldsymbol{y}}\big)\Big\}(\vec{\boldsymbol{u}})\cdot\phi_{\vec{\boldsymbol{d}}(T)}(-\vec{\boldsymbol{u}})\Big\}\Big\}\Big\} \\
&= -\frac{e^{-r(T-t)}}{\boldsymbol{S}_j(t_0)}\mathcal{F}\Big\{iu_j\mathcal{F}^{-1}\Big\{V\big(T,\vec{\boldsymbol{y}}\big)\Big\}(\vec{\boldsymbol{u}})\cdot\phi_{\vec{\boldsymbol{d}}(T)}(-\vec{\boldsymbol{u}})\Big\}.
\end{aligned}$$

Remark. The closed-form of other hedge parameters, e.g., gamma, theta, can be computed in a similar way. However, it's time consuming to compute the hedge parameters with respect to every underlying stock price as both Fourier and inverse Fourier transform are needed to perform for calculation of each Hedge parameter. Instead, we will use the nonuniform finite difference method to compute the Greeks since the Conv method gives a range of option prices which correspond to different spot prices. From Figure 3.1, the option prices are very accurate around the centre of the log spot price domain, which allows the finite difference method to output good approximations of hedge parameters. Although the effect of the circular convolution causes the option prices near the boundary of the log spot price domain to be inaccurate, we found that approximations on [1/4, 3/4] of the log domain are accurate in general.

Figure 3.1: Approximation of 1D European call option by FFT Conv method



3.4 Discretization of the Algorithm

To compute (3.13), let's first compute $\mathcal{F}^{-1}\Big\{V\big(T,\vec{\boldsymbol{y}}\big)\Big\}$, which gives us

$$\mathcal{F}^{-1}\Big\{V\big(T,\vec{\boldsymbol{y}}\big)\Big\}(\vec{\boldsymbol{u}}) = (\frac{1}{2\pi})^n \int_{\Omega^d} e^{i\vec{\boldsymbol{y}}\cdot\vec{\boldsymbol{u}}} V\big(T,\vec{\boldsymbol{y}}\big)d\vec{\boldsymbol{y}}. \tag{3.16}$$

We will give the detailed parameter settings in the discussion below.

Suppose we have $2^{n_1}, 2^{n_2}, \ldots, 2^{n_d}$ grid points on each dimension. To simplify the notation, define $\vec{\boldsymbol{N}} = (2^{n_1}, \ldots, 2^{n_d})^T$. Since the total integral length vector $\vec{\boldsymbol{L}}$ equals $10\sqrt{T}\vec{\boldsymbol{\sigma}}$, the grid spacing becomes

$$\Delta\vec{\boldsymbol{s}} = \Delta\vec{\boldsymbol{y}} = \vec{\boldsymbol{L}}/\vec{\boldsymbol{N}} = \vec{\boldsymbol{L}}/2^{\vec{\boldsymbol{n}}} = \Big(\frac{10\sigma_1\sqrt{T}}{2^{n_1}}, \ldots, \frac{10\sigma_d\sqrt{T}}{2^{n_d}}\Big)^T. \tag{3.17}$$

Additionally, to use FFT, the Nyquist relation has to be satisfied:

$$\begin{aligned} \Delta\vec{\boldsymbol{y}} \circ \Delta\vec{\boldsymbol{u}} &= 2\pi/\vec{\boldsymbol{N}} = \Big(2\pi/2^{n_1}, 2\pi/2^{n_2}, \ldots, 2\pi/2^{n_d}\Big)^T \\ \implies \Delta\vec{\boldsymbol{u}} &= 2\pi/\vec{\boldsymbol{L}} = \Big(\frac{2\pi}{10\sigma_1\sqrt{T}}, \ldots, \frac{2\pi}{10\sigma_d\sqrt{T}}\Big)^T. \end{aligned} \tag{3.18}$$

By our definition of Ω^d, let $s_{j,0} = y_{j,0} = -\boldsymbol{L}_j/2, u_{j,0} = -\boldsymbol{N}_j\Delta\boldsymbol{u}_j/2$. Therefore, we can write the point $s_{j,k}$ as $s_{j,0} + k\Delta\boldsymbol{s}_j$, $y_{j,k}$ as $y_{j,0} + k\Delta\boldsymbol{y}_j$, $u_{j,k}$ as $u_{j,0} + k\Delta\boldsymbol{u}_j$ for $k \in \{0, 1, \ldots, \boldsymbol{N}_j - 1\}$, where $s_{j,k}, y_{j,k}$ represent the k-th point in the j-th dimension of the spot price domain, $u_{j,k}$ represents the k-th point in the j-th dimension of the frequency domain. Note that by (3.17), the points can be further simplified as

$$s_{j,k} = y_{j,k} = (k - \boldsymbol{N}_j/2)\Delta\boldsymbol{y}_j, \quad u_{j,k} = (k - \boldsymbol{N}_j/2)\Delta\boldsymbol{u}_j, \quad k = 0, \ldots, \boldsymbol{N}_j - 1. \tag{3.19}$$

After the truncation, we are able to approximate (3.16) by the trapezoidal rule:

$$\begin{aligned}\mathcal{F}^{-1}\Big\{V\big(T,\vec{\boldsymbol{y}}\big)\Big\}(\vec{\boldsymbol{u}}_{\vec{\boldsymbol{j}}}) &= (\frac{1}{2\pi})^d \int_{\Omega^d} e^{iy_{\vec{\boldsymbol{k}}}\cdot\vec{\boldsymbol{u}}_{\vec{\boldsymbol{j}}}} V\big(T,\vec{\boldsymbol{y}}\big)d\vec{\boldsymbol{y}} \\ &= (\frac{1}{2\pi})^d d\boldsymbol{Y} \sum_{\boldsymbol{k}_1=0}^{\boldsymbol{N}_1-1}\cdots\sum_{\boldsymbol{k}_d=0}^{\boldsymbol{N}_d-1} Q_{\vec{\boldsymbol{k}}}\; exp(i\boldsymbol{y}_{1,k_1}\boldsymbol{u}_{1,j_1}+\cdots+i\boldsymbol{y}_{d,k_d}\boldsymbol{u}_{d,j_d})V(T,\vec{\boldsymbol{y}}_{\vec{\boldsymbol{k}}}) \\ &\quad + \mathcal{O}(\sum_{j=1}^{d}\Delta\boldsymbol{y}_j^2),\end{aligned} \tag{3.20}$$

where $\vec{\boldsymbol{k}} = (\boldsymbol{k}_1,\dots,\boldsymbol{k}_d)^T,\ \vec{\boldsymbol{j}} = (\boldsymbol{j}_1,\dots,\boldsymbol{j}_d)^T,\ \vec{\boldsymbol{u}}_{\vec{\boldsymbol{j}}} = (\boldsymbol{u}_{1,j_1},\dots,\boldsymbol{u}_{d,j_d})^T,\ \vec{\boldsymbol{y}}_{\vec{\boldsymbol{k}}} = (\boldsymbol{y}_{1,\boldsymbol{k}_1},\dots,\boldsymbol{y}_{d,\boldsymbol{k}_d})^T,\ d\boldsymbol{Y} = \prod_{j=1}^d \Delta\boldsymbol{y}_j,\ Q_{\vec{\boldsymbol{k}}} = \prod_{j=1}^d R(\boldsymbol{k}_j)$ and the trapezoidal weights:

$$R(\boldsymbol{k}_j) = \begin{cases}\frac{1}{2}, & \boldsymbol{k}_j = 0 \text{ or } \boldsymbol{N}_j - 1\\ 1, & \text{otherwise}\end{cases} \tag{3.21}$$

Note that the terms $exp(i\boldsymbol{y}_{m,k_m}\boldsymbol{u}_{m,j_m})$ can be written as

$$\begin{aligned}exp(i\boldsymbol{y}_{m,\boldsymbol{k}_m}\boldsymbol{u}_{m,\boldsymbol{j}_m}) &= exp(i(\boldsymbol{k}_m - \boldsymbol{N}_m/2)\Delta\boldsymbol{y}_j(\boldsymbol{j}_m - \boldsymbol{N}_m/2)\Delta\boldsymbol{u}_j) \\ &= exp(-i\boldsymbol{N}_m(\boldsymbol{k}_m+\boldsymbol{j}_m)\Delta\boldsymbol{y}_j\Delta\boldsymbol{u}_j/2)exp(i\boldsymbol{N}_m^2\Delta\boldsymbol{y}_j\Delta\boldsymbol{u}_j/4)exp(i\boldsymbol{k}_m\boldsymbol{j}_m\Delta\boldsymbol{y}_j\Delta\boldsymbol{u}_j) \\ &= exp(-\pi i(\boldsymbol{k}_m+\boldsymbol{j}_m))exp(\pi i\boldsymbol{N}_m/2)exp(\frac{2\pi i\boldsymbol{k}_m\boldsymbol{j}_m}{\boldsymbol{N}_j}) \quad \text{(by (3.18))} \\ &= (-1)^{\boldsymbol{k}_m+\boldsymbol{j}_m}(-1)^{\boldsymbol{N}_m/2}W_{\boldsymbol{N}_m}^{\boldsymbol{k}_m\boldsymbol{j}_m} \quad \text{(by Euler's formula)} \\ &= (-1)^{\boldsymbol{k}_m}(-1)^{\boldsymbol{j}_m}W_{\boldsymbol{N}_m}^{\boldsymbol{k}_m\boldsymbol{j}_m} \quad (\text{since } \boldsymbol{N}_m = 2^{\boldsymbol{n}_m}\text{, assume } n_m > 1),\end{aligned} \tag{3.22}$$

where $W_{\boldsymbol{N}_j} = exp(2\pi i/\boldsymbol{N}_j)$, for $m = 1,\dots,d$.

Thus, (3.20) becomes

$$\begin{aligned}\mathcal{F}^{-1}\Big\{V\big(T,\vec{\boldsymbol{y}}\big)\Big\}(\vec{\boldsymbol{u}}_{\vec{\boldsymbol{j}}}) &\approx (\frac{1}{2\pi})^d d\boldsymbol{Y} \sum_{\boldsymbol{k}_1=0}^{\boldsymbol{N}_1-1}\cdots\sum_{\boldsymbol{k}_d=0}^{\boldsymbol{N}_d-1} Q_{\vec{\boldsymbol{k}}}\; exp(i\boldsymbol{y}_{1,k_1}\boldsymbol{u}_{1,j_1}+\cdots+i\boldsymbol{y}_{d,k_d}\boldsymbol{u}_{d,j_d})V(T,\vec{\boldsymbol{y}}_{\vec{\boldsymbol{k}}}) \\ &= (\frac{1}{2\pi})^d d\boldsymbol{Y} \sum_{\boldsymbol{k}_1=0}^{\boldsymbol{N}_1-1}\cdots\sum_{\boldsymbol{k}_d=0}^{\boldsymbol{N}_d-1} Q_{\vec{\boldsymbol{k}}}\; V(T,\vec{\boldsymbol{y}}_{\vec{\boldsymbol{k}}})\prod_{m=1}^{d}(-1)^{\boldsymbol{k}_m}(-1)^{\boldsymbol{j}_m}W_{\boldsymbol{N}_m}^{\boldsymbol{k}_m\boldsymbol{j}_m} \\ &= (\frac{1}{2\pi})^d d\boldsymbol{Y}\big(\prod_{m=1}^{d}(-1)^{\boldsymbol{j}_m}\big)\sum_{\boldsymbol{k}_1=0}^{\boldsymbol{N}_1-1}\cdots\sum_{\boldsymbol{k}_d=0}^{\boldsymbol{N}_d-1}\Big(Q_{\vec{\boldsymbol{k}}}\; V(T,\vec{\boldsymbol{y}}_{\vec{\boldsymbol{k}}})\prod_{m=1}^{d}(-1)^{\boldsymbol{k}_m}\Big)W_{\boldsymbol{N}_m}^{\boldsymbol{k}_m\boldsymbol{j}_m} \\ &= (\frac{1}{2\pi})^d d\boldsymbol{Y}\big(\prod_{m=1}^{d}(-1)^{\boldsymbol{j}_m}\big)\big(\prod_{m=1}^{d}N_m\big)\mathcal{D}^{-1}\Big\{Q_{\vec{\boldsymbol{k}}}\; V(T,\vec{\boldsymbol{y}}_{\vec{\boldsymbol{k}}})\prod_{m=1}^{d}(-1)^{\boldsymbol{k}_m}\Big\}(\vec{\boldsymbol{u}}_{\vec{\boldsymbol{j}}}) \\ &= d\boldsymbol{H}\cdot\Big(\prod_{m=1}^{d}(-1)^{\boldsymbol{j}_m}\Big)\cdot\mathcal{D}^{-1}\Big\{V(T,\vec{\boldsymbol{y}}_{\vec{\boldsymbol{k}}})\; G(\vec{\boldsymbol{k}})\Big\}(\vec{\boldsymbol{u}}_{\vec{\boldsymbol{j}}}),\end{aligned} \tag{3.23}$$

where $\mathcal{D}^{-1}$ represents d-dimensional inverse discrete Fourier transform, $G(\vec{\boldsymbol{k}}) = Q_{\vec{\boldsymbol{k}}}\cdot\prod_{m=1}^d(-1)^{\boldsymbol{k}_m}$ and $d\boldsymbol{H} = (\frac{1}{2\pi})^d\cdot d\boldsymbol{Y}\cdot\big(\prod_{m=1}^d N_m\big)$.

Similar to what we have done at the beginning of the subsection, we need to truncate $\mathbb{R}^d$ by $\Upsilon^d = [-\vec{\boldsymbol{N}} \circ \Delta\vec{\boldsymbol{u}}/2, \vec{\boldsymbol{N}} \circ \Delta\vec{\boldsymbol{u}}/2]$. Thus, by the left-hand rectangle rule, the option price (3.13) becomes

$$\begin{aligned} V(t,\vec{\boldsymbol{s}}) &= e^{-r(T-t)}\mathcal{F}\Big\{\mathcal{F}^{-1}\Big\{V\big(T,\vec{\boldsymbol{y}}\big)\Big\}(\vec{\boldsymbol{u}}) \cdot \phi_{\vec{\boldsymbol{d}}(T)}(-\vec{\boldsymbol{u}})\Big\} \\ &= e^{-r(T-t)}\mathcal{F}\Big\{d\boldsymbol{H} \cdot \Big(\prod_{m=1}^{d}(-1)^{\boldsymbol{j}_m}\Big) \cdot \mathcal{D}^{-1}\Big\{V(T,\vec{\boldsymbol{y}}_{\vec{\boldsymbol{k}}})\,G(\vec{\boldsymbol{k}})\Big\}(\vec{\boldsymbol{u}}_{\vec{\boldsymbol{j}}}) \cdot \phi_{\vec{\boldsymbol{d}}(T)}(-\vec{\boldsymbol{u}}_{\vec{\boldsymbol{j}}})\Big\} \\ &\approx e^{-r(T-t)}\int_{\Upsilon^d} e^{-i\vec{\boldsymbol{u}}_{\vec{\boldsymbol{j}}}\cdot\vec{\boldsymbol{s}}_{\vec{\boldsymbol{x}}}} \cdot d\boldsymbol{H} \cdot \Big(\prod_{m=1}^{d}(-1)^{\boldsymbol{j}_m}\Big) \cdot \mathcal{D}^{-1}\Big\{V(T,\vec{\boldsymbol{y}}_{\vec{\boldsymbol{k}}})\,G(\vec{\boldsymbol{k}})\Big\}(\vec{\boldsymbol{u}}_{\vec{\boldsymbol{j}}}) \cdot \phi_{\vec{\boldsymbol{d}}(T)}(-\vec{\boldsymbol{u}}_{\vec{\boldsymbol{j}}})d\vec{\boldsymbol{u}} \qquad (3.24) \\ &= e^{-r(T-t)} \cdot d\boldsymbol{U} \cdot d\boldsymbol{H} \sum_{\boldsymbol{j}_1=0}^{\boldsymbol{N}_1-1}\cdots\sum_{\boldsymbol{j}_d=0}^{\boldsymbol{N}_d-1} exp(-i\boldsymbol{u}_{1,j_1}\boldsymbol{s}_{1,x_1} - \cdots - i\boldsymbol{u}_{d,j_d}\boldsymbol{s}_{d,x_d}) \cdot \Big(\prod_{m=1}^{d}(-1)^{\boldsymbol{j}_m}\Big) \\ &\quad \cdot \mathcal{D}^{-1}\Big\{V(T,\vec{\boldsymbol{y}}_{\vec{\boldsymbol{k}}})\,G(\vec{\boldsymbol{k}})\Big\}(\vec{\boldsymbol{u}}_{\vec{\boldsymbol{j}}}) \cdot \phi_{\vec{\boldsymbol{d}}(T)}(-\vec{\boldsymbol{u}}_{\vec{\boldsymbol{j}}}), \end{aligned}$$

where $\vec{\boldsymbol{s}}_{\vec{\boldsymbol{x}}} = (\boldsymbol{s}_{1,x_1}, \ldots, \boldsymbol{s}_{d,x_d})^T$, $\vec{\boldsymbol{u}}_{\vec{\boldsymbol{j}}} = (\boldsymbol{u}_{1,j_1}, \ldots, \boldsymbol{u}_{d,j_d})^T$, $d\boldsymbol{U} = \prod_{j=1}^{d}\Delta u_j = \prod_{j=1}^{d}\frac{2\pi}{\boldsymbol{N}_j\Delta\boldsymbol{y}_j} = \frac{(2\pi)^d}{d\boldsymbol{Y}\cdot\prod_{j=1}^{d}\boldsymbol{N}_j} = \frac{1}{d\boldsymbol{H}}$.

By an argument similar to (3.22), we claim that the term $exp(-i\boldsymbol{u}_{m,j_m}\boldsymbol{s}_{m,x_m}) = (-1)^{\boldsymbol{j}_m}(-1)^{\boldsymbol{x}_m}W_{\boldsymbol{N}_m}^{-\boldsymbol{j}_m\boldsymbol{x}_m}$. Therefore,

$$\begin{aligned} V(t,\vec{\boldsymbol{s}}_{\vec{\boldsymbol{x}}}) &= e^{-r(T-t)}\sum_{\boldsymbol{j}_1=0}^{\boldsymbol{N}_1-1}\cdots\sum_{\boldsymbol{j}_d=0}^{\boldsymbol{N}_d-1}\Big(\prod_{m=1}^{d}(-1)^{2\boldsymbol{j}_m}(-1)^{\boldsymbol{x}_m}W_{\boldsymbol{N}_m}^{-\boldsymbol{j}_m\boldsymbol{x}_m}\Big) \cdot \mathcal{D}^{-1}\Big\{V(T,\vec{\boldsymbol{y}}_{\vec{\boldsymbol{k}}})\,G(\vec{\boldsymbol{k}})\Big\}(\vec{\boldsymbol{u}}_{\vec{\boldsymbol{j}}}) \cdot \phi_{\vec{\boldsymbol{d}}(T)}(-\vec{\boldsymbol{u}}_{\vec{\boldsymbol{j}}}) \\ &= e^{-r(T-t)}\Big(\prod_{m=1}^{d}(-1)^{\boldsymbol{x}_m}\Big)\sum_{\boldsymbol{j}_1=0}^{\boldsymbol{N}_1-1}\cdots\sum_{\boldsymbol{j}_d=0}^{\boldsymbol{N}_d-1}W_{\boldsymbol{N}_m}^{-\boldsymbol{j}_m\boldsymbol{x}_m} \cdot \mathcal{D}^{-1}\Big\{V(T,\vec{\boldsymbol{y}}_{\vec{\boldsymbol{k}}})\,G(\vec{\boldsymbol{k}})\Big\}(\vec{\boldsymbol{u}}_{\vec{\boldsymbol{j}}}) \cdot \phi_{\vec{\boldsymbol{d}}(T)}(-\vec{\boldsymbol{u}}_{\vec{\boldsymbol{j}}}) \\ &= e^{-r(T-t)}\Big(\prod_{m=1}^{d}(-1)^{\boldsymbol{x}_m}\Big) \cdot \mathcal{D}\Big\{\mathcal{D}^{-1}\Big\{V(T,\vec{\boldsymbol{y}}_{\vec{\boldsymbol{k}}})\,G(\vec{\boldsymbol{k}})\Big\}(\vec{\boldsymbol{u}}_{\vec{\boldsymbol{j}}}) \cdot \phi_{\vec{\boldsymbol{d}}(T)}(-\vec{\boldsymbol{u}}_{\vec{\boldsymbol{j}}})\Big\}(\vec{\boldsymbol{s}}_{\vec{\boldsymbol{x}}}), \end{aligned} \tag{3.25}$$

which gives us the discretization of the algorithm.

4 Deep Galerkin Method

4.1 Background

The deep Galerkin method aims to alleviate the *curse of dimensionality* that arises in numerical PDE problems because it is a mesh-free method. It has been shown that not only can DGM give accurate results for the Black-Scholes equation, but also partial integro-differential equations and Hamilton-Jacobi-Bellman equations (see [14] for details), which regularly appear in mathematical finance. To begin with, consider a backward parabolic PDE with N spatial dimensions:

$$\begin{aligned} &\frac{\partial u(x,t)}{\partial t} + \mathcal{L}u(x,t) = 0, \quad (x,t) \in \Omega \times [0,T], \\ &u(x,t=T) = u_T(x), \\ &u(x,t) = g(x,t), (x,t) \in \partial\Omega \times [0,T]. \end{aligned} \tag{4.1}$$

The most important property of the deep Galerkin method is the differentiability of the deep neural network $f(x,t;\theta)$ where θ are the parameters to train, which allows us to optimize the following loss function:

$$J(f) = \left\|\frac{\partial f(x,t;\theta)}{\partial t} + \mathcal{L}f(x,t;\theta)\right\|^2_{\Omega\times[0,T]} + \left\|f(x,t;\theta) - g(x,t)\right\|^2_{\partial\Omega\times[0,T]} + \left\|f(x,T;\theta) - u_T(x)\right\|^2_{\Omega}. \tag{4.2}$$

Since the N-dimensional Black-Scholes equation contains second-order and first-order derivatives of V, both the Hessian and the gradient of f have to be computed. However, the time complexity of computing Hessian grows exponentially as the dimension increases, and as the stochastic gradient descent will be used, $\nabla\frac{\partial f(x,t;\theta)}{\partial x^2}$ will further increase the cost. As a result, J. Sirignano et al. proposed a Monte Carlo method for the fast computation of second derivatives in [1]. But we found that the cost of computing the Hessian is reasonable when the dimension is around $4 \sim 7$. Thus, we are going to compute the exact Hessian in our implementations, which will give us a more accurate gradient descent direction.

The vanilla European option can be priced by substituting the Black-Scholes operator for the operator $\mathcal{L}$ and training with the loss function 4.2. But for the American option, we need to modify the loss function due to the nature of free boundary problems. A. Al-Aradi et al. proposed the following loss function:

$$\begin{aligned} J(f) = &\left\|\Big(\frac{\partial f(x,t;\theta)}{\partial t} + \mathcal{L}f(x,t;\theta)\Big)\cdot\big(f(x,t;\theta) - u_T(x)\big)\right\|^2_{\Omega\times[0,T]} + \left\|ReLU\Big(\frac{\partial f(x,t;\theta)}{\partial t} + \mathcal{L}f(x,t;\theta)\Big)\right\|^2_{\Omega\times[0,T]} \\ &+ \left\|ReLU\big(u_T(x) - f(x,t;\theta)\big)\right\|^2_{\Omega\times[0,T]} + \left\|f(x,T;\theta) - u_T(x)\right\|^2_{\Omega}. \end{aligned} \tag{4.3}$$

The first three terms of 4.3 comes from the linear complementarity form of the American Black-Scholes equation 2.11, and the last term comes from the terminal payoff function. Although this loss function is different from the one used in [1], we will show its effectiveness in the numerical experiments.

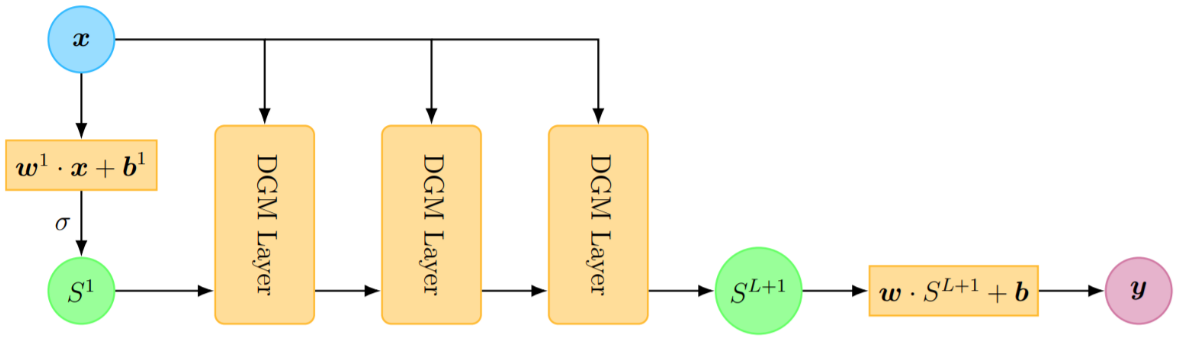
4.2 The Deep Neural Network Architecture

J. Sirignano and K. Spiliopoulos found the following deep neural network to be effective in the N-dimensional Black-Scholes equation:

$$\begin{aligned} S^1 &= \sigma(W^1\vec{x} + b^1), \\ Z^l &= \sigma(U^{z,l}\vec{x} + W^{z,l}S^l + b^{z,l}), \quad l = 1,\dots,L, \\ G^l &= \sigma(U^{g,l}\vec{x} + W^{g,l}S^l + b^{g,l}), \quad l = 1,\dots,L, \\ R^l &= \sigma(U^{r,l}\vec{x} + W^{r,l}S^l + b^{r,l}), \quad l = 1,\dots,L, \\ H^l &= \sigma(U^{h,l}\vec{x} + W^{h,l}(S^l \circ R^l) + b^{h,l}), \quad l = 1,\dots,L, \\ S^{l+1} &= (1 - G^l)\circ H^l + Z^l \circ S^l, l = 1,\dots,L, \\ f(x,t;\theta) &= WS^{L+1} + b, \end{aligned} \tag{4.4}$$

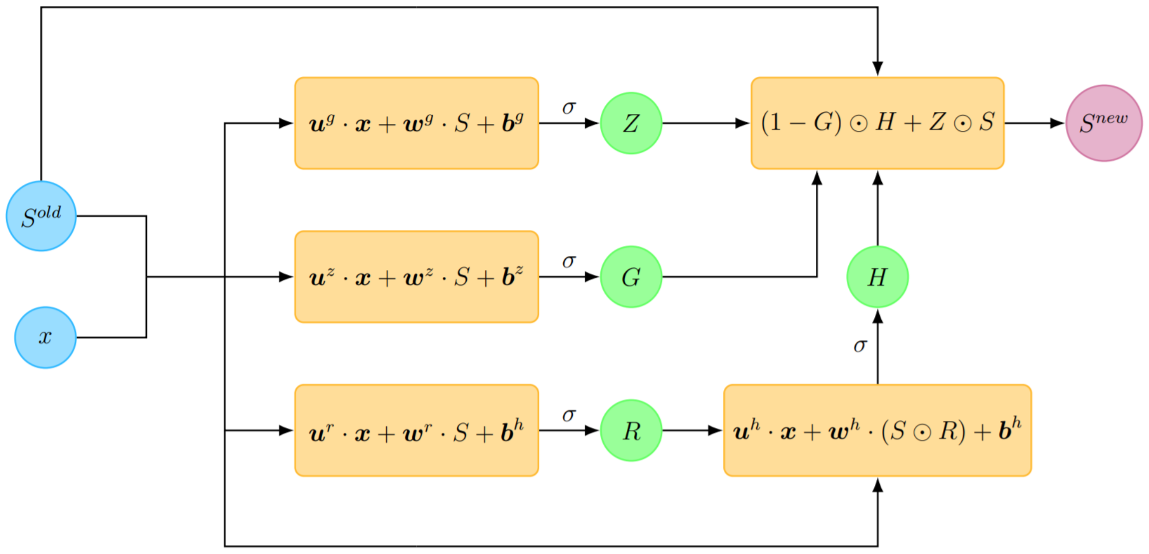
where $\vec{x} = (x,t)$, the number of the hidden layers is $L+1$, $\circ$ denotes element-wise multiplication, and σ represents the activation function. The author found $L = 3$ and $\sigma = \tanh$ to be effective in solving PDEs. Additionally, they set the width of each layer to be 50. For details, please refer to the chapter 4.2 in [1]. A. Al-Aradi et al. visualizes the architecture in [14], see Figures 4.1, 4.2.

Figure 4.1: Bird's-eye perspective of overall DGM architecture [14]



$\boldsymbol{x}$
$\boldsymbol{w}^1 \cdot \boldsymbol{x} + \boldsymbol{b}^1$
σ
S^1
DGM Layer
DGM Layer
DGM Layer
S^{L+1}
$\boldsymbol{w} \cdot S^{L+1} + \boldsymbol{b}$
$\boldsymbol{y}$

Figure 4.2: Operations within a single DGM layer [14]



S^{old}
x
$\boldsymbol{u}^g \cdot \boldsymbol{x} + \boldsymbol{w}^g \cdot S + \boldsymbol{b}^g$
$\boldsymbol{u}^z \cdot \boldsymbol{x} + \boldsymbol{w}^z \cdot S + \boldsymbol{b}^z$
$\boldsymbol{u}^r \cdot \boldsymbol{x} + \boldsymbol{w}^r \cdot S + \boldsymbol{b}^h$
σ
Z
G
R
H
$(1-G) \odot H + Z \odot S$
S^{new}
$\boldsymbol{u}^h \cdot \boldsymbol{x} + \boldsymbol{w}^h \cdot (S \odot R) + \boldsymbol{b}^h$

4.3 Error Bound of the Deep Galerkin Method's Approximation of European Option Prices

J. Sirignano and K. Spiliopoulos have shown that their neural network architecture gives accurate results to various types of problems. In this paper, we discovered another advantage of the deep Galerkin method: given the loss of the trained model, we are able to compute an error bound of the approximation. As we derived in section 2.1, the N-dimensional Black-Scholes equation is written as follows:

$$\frac{\partial V}{\partial \tau} + \frac{1}{2}\sum_{i,j=1}^{N} \sigma_i \sigma_j S_i S_j \rho_{ij} \frac{\partial^2 V}{\partial S_i \partial S_j} + \sum_{j=1}^{N} (r - q_j) S_j \frac{\partial V}{\partial S_j} - rV = 0, \tag{4.5}$$

where

$$V(\vec{S}_{min}, \tau) = \Omega_1(\tau), \quad V(\vec{S}_{max}, \tau) = \Omega_2(\tau), \quad V(\vec{S}, T) = \Phi(\vec{S}). \tag{4.6}$$

By deep Galerkin method, we solved U that satisfies

$$\frac{\partial U}{\partial \tau} + \frac{1}{2}\sum_{i,j=1}^{N} \sigma_i\sigma_j S_i S_j \rho_{ij}\frac{\partial^2 U}{\partial S_i \partial S_j} + \sum_{j=1}^{N}(r-q_j)S_j\frac{\partial U}{\partial S_j} - rU = f(\vec{S},\tau), \tag{4.7}$$

where

$$V(\vec{S}_{min},\tau) = \Omega_1(\tau) + h_1(\tau), \quad V(\vec{S}_{max},\tau) = \Omega_2(\tau) + h_2(\tau), \quad U(\vec{S},T) = \Phi(\vec{S}) + g(\vec{S}). \tag{4.8}$$

In other words, $f(\vec{S},\tau), h_1(\tau), h_2(\tau), g(\vec{S})$ represent the loss of the trained model and when the model is trained perfectly, $f(\vec{S},\tau) = h_1(\tau) = h_2(\tau) = g(\vec{S}) = 0$. Before we proceed with the proof, let's first introduce few definitions and lemmas.

Definition 4.1 (uniformly parabolic)**.** We say that the partial differential operator $\frac{\partial}{\partial t} + \mathcal{L}$ where

$$\mathcal{L} = \sum_{i,j}^{N} a_{ij}(x,t)\frac{\partial^2}{\partial x_i \partial x_j} + \sum_{i}^{N} b_i(x,t)\frac{\partial}{\partial x_i} \tag{4.9}$$

is uniformly parabolic if there exists a constant $\theta > 0$ such that

$$\sum_{i,j}^{N} a_{ij}(x,t)\xi_i\xi_j \geq \theta|\xi|^2 \tag{4.10}$$

for all $(x,t) \in U_T, \xi \in \mathbb{R}^n$.

Lemma 4.1 (Weak maximum principle)**.** *Assume the operator $\mathcal{L}$ has the nondivergence form*

$$\mathcal{L} = \sum_{i,j}^{N} a_{ij}(x,t)\frac{\partial^2}{\partial x_i \partial x_j} \tag{4.11}$$

with the coefficients $a_{ij}(x,t)$ are continuous, non-negative and $a_{ij}(x,t) = a_{ji}(x,t)$ for $i,j = 1,\ldots,n$. If the operator $\frac{\partial}{\partial t} + \mathcal{L}$ is uniformly parabolic, then

1.

$$u_t + \mathcal{L}u \leq 0 \ \textit{ in } \ U_T \implies \min_{\bar{U}_T} u = \min_{\Gamma_T} u. \tag{4.12}$$

2.

$$u_t + \mathcal{L}u \geq 0 \ \textit{ in } \ U_T \implies \max_{\bar{U}_T} u = \max_{\Gamma_T} u. \tag{4.13}$$

Remark. Although this is an incomplete version of the weak maximum principle, it's enough for our error analysis.

Now we can start the proof. Define $R = U - V$. By linearity of Black-Scholes equation, we have

$$\frac{\partial R}{\partial \tau} + \frac{1}{2}\sum_{i,j=1}^{N} \sigma_i\sigma_j S_i S_j \rho_{ij}\frac{\partial^2 R}{\partial S_i \partial S_j} + \sum_{j=1}^{N}(r-q_j)S_j\frac{\partial R}{\partial S_j} - rR = f_0(\vec{S},\tau), \tag{4.14}$$

where

$$R(\vec{S}_{min},\tau) = h_0^l(\tau), \quad R(\vec{S}_{max},\tau) = h_0^r(\tau), \quad R(\vec{S},T) = g_0(\vec{S}). \tag{4.15}$$

We assume that $|f_0(\vec{S},\tau)| < \epsilon_0$, $|h_0^l(\tau)| < \epsilon_1$, $|h_0^r(\tau)| < \epsilon_2$, $|g_0(\vec{S})| < \epsilon_3$.

We first transform the equation (4.5) into a constant variable PDE and eliminate the first derivative terms. Define $x_i = \ln S_i - (r - q_i - \frac{1}{2}\sigma_i^2)\tau$, then,

$$Q(\vec{x},\tau) = R(exp[\vec{x} + (r - \vec{q} - \frac{1}{2}\vec{\sigma}^2)\tau],\tau), \quad f_1(\vec{x},\tau) = f_0(exp[\vec{x} + (r - \vec{q} - \frac{1}{2}\vec{\sigma}^2)\tau],\tau), \tag{4.16}$$

$$h_1^l(\tau) = h_0^l(\tau), \quad h_1^r(\tau) = h_0^r(\tau), \quad g_1(\vec{x}) = g_0(exp[\vec{x} + (r - \vec{q} - \frac{1}{2}\vec{\sigma}^2)T]). \tag{4.17}$$

This gives us

$$\frac{\partial Q}{\partial \tau} + \frac{1}{2}\sum_{i,j=1}^{N} \sigma_i\sigma_j\rho_{ij}\frac{\partial^2 Q}{\partial x_i \partial x_j} - rQ = f_1(\vec{x},\tau) \tag{4.18}$$

with boundary and terminal conditions

$$Q(\overrightarrow{x_{min}}(\tau),\tau) = h_1^l(\tau), \quad Q(\overrightarrow{x_{max}}(\tau),\tau) = h_1^r(\tau), \quad Q(\vec{x},T) = g_1(\vec{x}), \tag{4.19}$$

where

$$\overrightarrow{x_{min}}(\tau) = \ln \overrightarrow{S_{min}} - (r - \frac{1}{2}\vec{\sigma}^2)\tau, \quad \overrightarrow{x_{max}}(\tau) = \ln \overrightarrow{S_{max}} - (r - \frac{1}{2}\vec{\sigma}^2)\tau. \tag{4.20}$$

We next eliminate the source term $-rR$. Define

$$Q(\vec{x},\tau) = e^{-r(T-\tau)}\Psi(\vec{x},\tau), \tag{4.21}$$

then, Ψ satisfies

$$\frac{\partial \Psi}{\partial \tau} + \frac{1}{2}\sum_{i,j=1}^{N} \sigma_i\sigma_j\rho_{ij}\frac{\partial^2 \Psi}{\partial x_i \partial x_j} = f_1(\vec{x},\tau) \tag{4.22}$$

with boundary and terminal conditions

$$\Psi(\overrightarrow{x_{min}}(\tau),\tau) = e^{r(T-\tau)}h_1^l(\tau), \quad \Psi(\overrightarrow{x_{max}}(\tau),\tau) = e^{r(T-\tau)}h_1^r(\tau), \quad \Psi(\vec{x},T) = e^{r(T-\tau)}g_1(\vec{x},T). \tag{4.23}$$

Theorem 4.1. *Define $\mathcal{L} = \frac{1}{2}\sum_{i,j=1}^{N}\sigma_i\sigma_j\rho_{ij}\frac{\partial^2}{\partial x_i\partial x_j}$. Then, $\frac{\partial}{\partial\tau} + \mathcal{L}$ is uniformly parabolic.*

Proof. The covariance matrix C of the N correlated assets can be represented by $(\vec{\boldsymbol{\sigma}}^T\vec{\boldsymbol{\sigma}})\circ\rho$, where $\circ$ is the element-wise multiplication of two matrices. Since the covariance matrix C is a real symmetric values, there is an orthogonal basis $v_1,\ldots,v_N$ such that $Cv_i = \lambda_i v_i$ for $i = 1,\ldots,N$. Without the loss of generality, we can let $\lambda_1 \le \lambda_2 \le \ldots \lambda_N$ by reindexing the eigenvectors. Therefore, $C = QDQ^T$ where Q is a matrix whose column vectors are $v_1, v_2, \ldots, v_N$ and D is a diagonal matrix whose diagonals are $\lambda_1, \lambda_2, \ldots, \lambda_N$.

Let $\xi \in \mathbb{R}^n$. Let $\theta = \frac{\lambda_1}{2}$. Then,

$$\frac{1}{2}\sum_{i,j}^{N}\sigma_i\sigma_j\rho_{ij}\xi_i\xi_j = \frac{1}{2}\xi^T C\xi = \frac{1}{2}\xi^T(QDQ^T)\xi = \frac{1}{2}(Q^T\xi)^T D(Q^T\xi) = \frac{1}{2}\sum_{i}^{N}\lambda_i(Q^T\xi)_i^2. \tag{4.24}$$

Since λ_1 is the smallest eigenvalue,

$$\frac{1}{2}\sum_{i,j}^{N}\sigma_i\sigma_j\rho_{ij}\xi_i\xi_j \ge \frac{\lambda_1}{2}\sum_{i}^{N}(Q^T\xi)_i^2 = \frac{\lambda_1}{2}\|\xi\|^2 = \theta\|\xi\|^2. \tag{4.25}$$

The second last equality holds since Q^T is an orthogonal matrix. □

Since all the condition of the weak maximum principle holds, we are now able to calculate the error bound using that theorem. Define two vectors of size N

$$l^{inf} = \ln(\overrightarrow{S_{min}}) - |r - \frac{1}{2}\vec{\sigma}^2|T, \quad l^{sup} = \ln(\overrightarrow{S_{max}}) + |r - \frac{1}{2}\vec{\sigma}^2|T \tag{4.26}$$

with l_i^{inf}, l_i^{sup} denoting the i-th entry of l^{inf} and l^{sup}, respectively. We are going to construct a function U, s.t., $\mathcal{L}U \geq 0$ and $U > \psi$.

Let $c \in R$. Define $U = \Psi + c\sum_{i,j=1}^{N}\Big((l_i^{sup} - l_i^{inf})(l_j^{sup} - l_j^{inf}) - (x_i - l_i^{inf})(x_j - l_j^{inf})\Big)$. We have

$$\begin{aligned}\frac{\partial U}{\partial \tau} = \frac{\partial \Psi}{\partial \tau} &= -\frac{1}{2}\sum_{i,j=1}^{N}\sigma_i\sigma_j\rho_{ij}\frac{\partial^2 \Psi}{\partial x_i \partial x_j} + f_1(\vec{x},\tau) \\ &= -\frac{1}{2}\sum_{i,j=1}^{N}\sigma_i\sigma_j\rho_{ij}(\frac{\partial^2 U}{\partial x_i \partial x_j} + c) + f_1(\vec{x},\tau),\end{aligned} \tag{4.27}$$

which gives us

$$\frac{\partial U}{\partial \tau} + \mathcal{L}U + \frac{1}{2}c\sum_{i,j=1}^{N}\sigma_i\sigma_j\rho_{ij} - f_1(\vec{x},\tau) = 0. \tag{4.28}$$

Let $c = \frac{max|f_1(\vec{x},\tau)|}{\sum_{i,j=1}^{N}\sigma_i\sigma_j\rho_{ij}/2}$, the equation (4.28) becomes $\mathcal{L}U \geq 0$. Since the coefficient of the second derivative terms are larger than 0, by the weak maximum principle, we know that U attains its maximum on the parabolic boundary Γ. Thus,

$$\begin{aligned}U < &\, max|\Psi(\overrightarrow{x_{min}}(\tau),\tau)| + max|\Psi(\overrightarrow{x_{max}}(\tau),\tau)| + max|\Psi(\vec{x},T)| \\ &+ 3c\sum_{i,j=1}^{N}\Big((l_i^{sup} - l_i^{inf})(l_j^{sup} - l_j^{inf}) - (x_i - l_i^{inf})(x_j - l_j^{inf})\Big)\end{aligned} \tag{4.29}$$

To simplify our notation, let $K = 4\sum_{i,j=1}^{N}\Big(l_i^{sup} - l_i^{inf}\Big)\Big(l_j^{sup} - l_j^{inf}\Big)/\sum_{i,j=1}^{N}(\sigma_i\sigma_j\rho_{ij})$. By the definition of U, we have

$$\begin{aligned}\Psi &= U - c\sum_{i,j=1}^{N}\Big((l_i^{sup} - l_i^{inf})(l_j^{sup} - l_j^{inf}) - (x_i - l_i^{inf})(x_j - l_j^{inf})\Big) \\ &< max|\Psi(\overrightarrow{x_{min}}(\tau),\tau)| + max|\Psi(\overrightarrow{x_{max}}(\tau),\tau)| + max|\Psi(\vec{x},T)| \\ &\quad + 2c\sum_{i,j=1}^{N}\Big((l_i^{sup} - l_i^{inf})(l_j^{sup} - l_j^{inf}) - (x_i - l_i^{inf})(x_j - l_j^{inf})\Big) \\ &< max|\Psi(\overrightarrow{x_{min}}(\tau),\tau)| + max|\Psi(\overrightarrow{x_{max}}(\tau),\tau)| + max|\Psi(\vec{x},T)| + Kmax|f_1(\vec{x},\tau)|.\end{aligned} \tag{4.30}$$

We can obtain a corresponding lower bound for Ψ in a similar way by taking $c = -\frac{max|f_1(\vec{x},\tau)|}{\sum_{i,j=1}^{N}\sigma_i\sigma_j\rho_{ij}/2}$, which gives us

$$|\Psi| < max|\Psi(\overrightarrow{x_{min}}(\tau),\tau)| + max|\Psi(\overrightarrow{x_{max}}(\tau),\tau)| + max|\Psi(\vec{x},T)| + Kmax|f_1(\vec{x},\tau)|. \tag{4.31}$$

By the definition of Q (4.21), we have

$$\begin{aligned}|Q| &< max|h_1^l(\tau)| + max|h_1^r(\tau)| + max|g_1(\vec{x})| + Ke^{-r(T-\tau)}max|f_1(\vec{x},\tau)| \\ &\leq max|h_1^l(\tau)| + max|h_1^r(\tau)| + max|g_1(\vec{x})| + Kmax|f_1(\vec{x},\tau)|\end{aligned} \tag{4.32}$$

Finally, by the definition of R (4.16), we have

$$\begin{aligned}|R| &< max|h_0^l(\tau)| + max|h_0^r(\tau)| + max|g_0(\vec{x})| + Kmax|f_0(\vec{S},\tau)| \\ &< K\epsilon_0 + \epsilon_1 + \epsilon_2 + \epsilon_3.\end{aligned} \tag{4.33}$$

4.4 Comparisons with the Deep BSDE Method

J. Han et al. proposed the deep BSDE method in [2] and used the European N-dimensional Black-Scholes equation as an example. Y. Chen et al. extended their Han's work to the American-style options by using a new network architecture and a new loss function. In our paper, we analyze Chen's method for European-style options. To start, we first transform the PDE into a BSDE.

Lemma 4.2. *The N-dimensional second-order linear PDE*

$$\frac{\partial V}{\partial t}(\boldsymbol{x},t) + \frac{1}{2}\sum_{i,j=1}^{d} \sigma_i\sigma_j x_i x_j \rho_{ij} \frac{\partial^2 V}{\partial x_i \partial x_j}(\boldsymbol{x},t) + \sum_{j=1}^{d}(r-q_j)x_j \frac{\partial V}{\partial x_j}(\boldsymbol{x},t) - rV(\boldsymbol{x},t) = 0 \tag{4.34}$$

is equivalent to the following BSDE:

$$dV(\boldsymbol{X},t) = rV(\boldsymbol{X},t)dt + \sum_{i=1}^{d} \sigma_i \boldsymbol{X}_i(t) \frac{\partial V}{\partial x_i}(\boldsymbol{X},t)dW_i(t), \tag{4.35}$$

where $\boldsymbol{X}$ satisfies the SDE (2.13).

As pointed out by Y. Chen in [15], (4.35) can be discretized as

$$V(\boldsymbol{X}_m^{n+1}, t^{n+1}) = (1 + r\Delta t)V(\boldsymbol{X}_m^n, t^n) + \sum_{i=1}^{d} \sigma_i (X_i)_m^n \frac{\partial V}{\partial x_i}(\boldsymbol{X}_m^n, t^n)(\Delta W_i)_m^n \tag{4.36}$$

for $m = 1, \ldots, M$, $n = N-1, \ldots, 0$. Note that m represents the index of the simulation, while n represents the time step index. $\boldsymbol{X}_m^n$ are known for all eligible m and n since we can get the analytical solution to them by (2.15). Therefore, by designing a neural network $f(\boldsymbol{X};\theta)$ that takes $V(\boldsymbol{X}_m^{n+1})$ as an input and returns $V(\boldsymbol{X}_m^n)$ as the output, we are able to write the cost function as

$$J(f) = \sum_{m=1}^{M} \mathcal{R}[f(V^{n+1})]_m^2, \tag{4.37}$$

where

$$\mathcal{R}[f(V^{n+1})]_m = \mathcal{R}[V^n]_m = (1 + r\Delta t)V^n(\boldsymbol{X}_m^n, t^n) + \sum_{i=1}^{d} \sigma_i (X_i)_m^n \frac{\partial V}{\partial x_i}(\boldsymbol{X}_m^n, t^n)(\Delta W_i)_m^n - V^{n+1}(\boldsymbol{X}_m^{n+1}, t^{n+1}). \tag{4.38}$$

Computation of the Loss Function

In this regard, the deep BSDE method outperforms the deep Galerkin method since the BSDE method only requires the gradients to calculate the loss function. Because the gradient operation has been efficiently implemented in Tensorflow while the Hessian operation is much more expensive, the deep BSDE method saves lots of time by avoiding the computation of the loss function. Even though the deep Galerkin method can use Monte Carlo methods to efficiently compute the Hessian, some degree of accuracy is lost due to the nature of Monte Carlo methods, while the deep BSDE method only has numerical errors to worry about.

Chen's deep neural network architecture is formally defined as

$$\begin{aligned} y^N(\boldsymbol{x}) &= f(\boldsymbol{x}), \quad n = N; \\ y^n(\boldsymbol{x};\theta^n) &= y^{n+1}(\boldsymbol{x};\theta^{n+1}) + \Delta t \cdot \mathcal{F}(\boldsymbol{x};\theta^n), \quad n = N-1, \ldots, 0. \end{aligned} \tag{4.39}$$

Regarding each remainder function $\mathcal{F}(\boldsymbol{x};\Sigma^n)$, Chen constructs it using an L-layer feedforward network with batch normalizations and ReLU activation. Chen emphasized that the recursive architecture is critical to the accuracy of the model, since option prices between two "neighboring" time steps should be close to each other, leading to the fact that $y^{n+1}(\boldsymbol{x};\theta^{n+1})$ has already become a good approximation of the option price at the n-th time step.

Therefore, in Chen's model, there is one neural network at each time step which aims to compute the difference of the option prices between "neighboring" two time steps. By the nature of the neural network, we are able to evaluate the delta ($\frac{\partial V}{\partial x}$) and gamma ($\frac{\partial^2 V}{\partial x^2}$) of the option price at the n-th time step by computing the gradient and the Hessian of $y^n(x;\theta^n)$ with respect to the input $\boldsymbol{x}$. The computation of theta ($\frac{\partial V}{\partial t}$) is a little different since the time is neither an input nor a model parameter. Instead, theta at the n-th time step is given by

$$\frac{\partial V}{\partial t}(\boldsymbol{x};\theta^n) = \mathcal{F}(\boldsymbol{x};\theta^n) = \frac{y^n(\boldsymbol{x};\theta^n) - y^{n+1}(\boldsymbol{x};\theta^{n+1})}{\Delta t}. \tag{4.40}$$

Computation of the Hedge parameters

The deep Galerkin method is known for its ability to compute the Greeks on the entire domain. Chen claimed that his method is also able to do that, while we think this is not completely true. The training data of the deep BSDE model is simulated by (4.35), which is composed of mostly "reasonable" paths and only a few "extreme" paths. This tells us that the training data is biased and we can expect both the option values and the Greeks are inaccurate on less sampled paths. In spite of this fact, Chen showed that the option values and the Greeks are accurate on most of the usual paths numerically. Overall, this cannot be seen as a drawback of the deep BSDE method, since less sampled paths rarely happened in the real-life, although the deep Galerkin method should be preferred when a large range of option prices and the Greeks are needed to be calculated.

Generality of the Method

It has been shown that a large class of PDEs, including linear/nonlinear/high-dimensional/free-boundary/integro-differential equations, can be solved by the deep Galerkin method. However, the deep BSDE method aims to solve backward stochastic differential equations and parabolic PDEs (due to the Feynman–Kac formula).

5 Numerical Experiments and Comparisons

5.1 Benchmark

Assume the stock price follows the geometric Brownian motion, we don't have an analytical solution to the price of most of the multi-asset options except the vanilla option with a geometric average terminal payoff function f, which is given by

$$f(\vec{\boldsymbol{S}}) = \max\Big(\big(\prod_{i=1}^{N}\boldsymbol{S}_i(T)\big)^{1/N} - K, 0\Big), \tag{5.1}$$

where K represents the strike price. Assume that the dividend rates q are all same for each asset, the analytical solution to the option price is written as

$$V(\vec{\boldsymbol{S}}(0), 0) = e^{-rT}[F\Phi(d_1) - K\Phi(d_2)], \tag{5.2}$$

where

$$\begin{gathered} F = \Big(\Pi_{i=1}^{N} S_i(0)\Big)^{\frac{1}{N}} \exp\Big(\Big(r - q + \frac{1}{2n}\sum_{i=1}^{N}\sigma_i^2 + \frac{1}{2}\sigma^2\Big)T\Big), \quad \sigma = \frac{1}{N}\sqrt{\sum_{i,j=1}^{N}\rho_{i,j}\sigma_i\sigma_j}, \\ d_1 = \frac{\ln\frac{F}{K} + \frac{1}{2}\sigma^2 T}{\sigma\sqrt{T}}, \quad d_2 = d_1 - \sigma\sqrt{T}, \end{gathered} \tag{5.3}$$

and Φ is the cumulative distribution function of a standard normal random variable.

5.2 Experiments: Geometric Average Payoff European Call Option

In this experiment, we will use the parameter settings shown in Table 5.1, where the volatility, dividend rates and correlations are same for all assets.

Table 5.1: Parameter Settings for 4-Dimensional Geometric Average Payoff Call Option

Parameters	Value
Asset num	4
Spot price	40
Strike price K	40
Time to maturity T	1
Volatility σ	0.2
Interest rate r	0.06
Dividend rate q	0.04
Correlation ρ	0.25
Analytical sol	2.16524

FFT-based Convolution Method

From table 5.2, we can see that the FFT-based convolution method can give accurate approximations to both the prices and the deltas in a short amount of time: within 1 second, an approximation to both the price and the delta with a relative error within 0.2% can be computed. As we mentioned in section 3.3, since the FFT Conv method outputs a range of prices, the calculation of deltas becomes cost-free by using finite difference methods. However, the CPU time of executing the algorithm, as well as the memory consumption, grows exponentially as N increases. Nevertheless, we will see in the next experiment that, even in 6D case, the FFT Conv method is still robust and efficient.

Table 5.2: Rel error and execution time of the Conv method (4D geometric average call option)

4D geometric average call option (price ≈ 2.16524, delta ≈ 0.13125)							
FFT-based Convolution Method							
N	Approx	Rel err	Order	Delta	Rel err	Order	CPU Time
8^4	2.116	2.3e-02	NA	0.122	7.0e-02	NA	0.01
16^4	2.155	4.6e-03	2.3	0.130	7.8e-03	3.2	0.06
32^4	2.163	1.1e-03	2.0	0.131	1.8e-03	2.1	0.88
64^4	2.165	2.9e-04	2.0	0.131	4.7e-04	1.9	15.18
128^4	2.165	8.3e-05	1.8	0.131	1.5e-04	1.7	284.66

Monte Carlo Methods

We first look at the non-quasi-Monte Carlo methods. From table 5.3, we made the following observations:

- Given a roughly same amount of time, the method of antithetic variates can sample twice as many as the vanilla MC does since it gets $N/2$ samples for free. Although one can't see a significant improvement of accuracy by using the method of antithetic variates, we showed that it indeed reduces the variance in the next experiment.

- The control variates method outputs much more accurate results compared with the other two methods. Once N is larger or equal to 2^{16}, the relative errors are always within 0.01%. From this, we can deduce that its variance reduction effect is better than the other methods. Generally speaking, it takes around 1 second to give an approximation with a relative error less than 0.01%, and this result is slightly better than the one given by the FFT Conv method. However, it requires another two Monte Carlo simulations to compute the deltas, while minimum delta computations are needed for the FFT Conv method.

As for the quasi-Monte Carlo methods, we observed the following phenomena:

- The accuracy of the approximations is much better than the one outputted by non-quasi-Monte Carlo methods. For example, the relative errors of both the Sobol sequence and the Sobol sequence with control variates are less oscillating than the relative errors of other three methods. Thus, we claim that the low-discrepancy methods are more robust in general.

- It takes roughly 3 times the length of time to sample one point from Sobol sequence compared with the regular random number generator. Nevertheless, even if we consider the accuracy versus the time, the low-discrepancy methods still outperform the regular Monte Carlo methods. Even more, our implementation of the Sobol sequence is not as optimized as the regular random number generator.

In conclusion, the methods using the Sobol sequence are significantly more accurate than the ones without that. The control variates method is the best technique introduced in this paper since the use of it doesn't increase much computation time and as we will show in the next experiment, it has the best variance reduction effect.

Table 5.3: Rel error and execution time of vanilla MC, antithetic variates and control variates (4D geom. avg call option)

4D geometric average call option (price ≈ 2.16524)												
	Vanilla				Antithetic Variates				Control Variates			
N	Approx	Rel err	Order	Time	Approx	Rel err	Order	Time	Approx	Rel err	Order	Time
2^{14}	2.158	3.5e-03	NA	0.29	2.147	8.6e-03	NA	0.18	2.144	9.9e-03	NA	0.34
2^{15}	2.187	9.9e-03	-1.5	0.50	2.142	1.1e-02	-0.3	0.30	2.161	2.2e-03	2.2	0.58
2^{16}	2.153	5.6e-03	0.8	0.99	2.172	2.9e-03	1.9	0.60	2.165	2.6e-04	3.0	1.06
2^{17}	2.164	5.4e-04	3.4	2.01	2.160	2.4e-03	0.3	1.21	2.164	4.8e-04	-0.9	2.13
2^{18}	2.156	4.4e-03	-3.0	4.02	2.164	7.0e-04	1.8	2.44	2.161	2.0e-03	-2.0	4.27
2^{19}	2.169	1.8e-03	1.3	8.12	2.161	1.8e-03	-1.3	4.90	2.166	5.6e-04	1.8	8.56

Table 5.4: Rel error and execution time of Sobol seq and Sobol seq with control variates (4D geom. avg call option)

4D geometric average call option (price ≈ 2.16524)								
	Sobol Sequence				Control Variates + Sobol Sequence			
N	Approx	Rel err	Order	Time	Approx	Rel err	Order	Time
2^{14}	2.163	1.1e-03	NA	0.81	2.164	6.8e-04	NA	0.80
2^{15}	2.164	6.0e-04	0.9	1.59	2.164	3.5e-04	1.0	1.58
2^{16}	2.165	1.7e-04	1.8	3.19	2.165	3.5e-05	3.3	3.17
2^{17}	2.165	1.1e-04	0.6	6.46	2.165	3.8e-05	-0.1	6.36
2^{18}	2.165	3.3e-05	1.7	13.05	2.165	4.2e-06	3.2	12.99
2^{19}	2.165	3.8e-05	-0.2	26.53	2.165	1.9e-05	-2.2	26.31

Comparisons

Admittedly, the Conv method and Monte Carlo methods can achieve the same degree of relative error (0.02%) in roughly same amount of time (1 sec), but the FFT-based Conv method should be preferred in the 4 dimensional case due to its determinism and ability of computing the Greeks efficiently and accurately. As for the memory complexity, the Monte Carlo method outperforms the Conv method, for example, in this experiment it requires around 134 megabytes memory to execute the Conv method when 32^4 grid points are used, while the Monte Carlo method will only consume 4 megabyte when $4 \cdot 2^{17}$ paths are simulated. In the other words, the memory consumption increases exponentially for Conv method and linearly for Monte Carlo methods. Nevertheless, 134 megabytes are negligible nowadays, so the memory consumption is not a big issue in a 4 dimensional problem.

5.3 Experiments: Basket European Put Option

In this subsection, we will demonstrate the effectiveness of improvements introduced in section 2.3, along with the disadvantage of Monte Carlo methods. Since the basket option is more common in the market, we also like to test the efficiency of the FFT-based convolution method in basket options.

The parameter settings are shown in Table 5.5, and same as the last experiment, the volatility, dividend rates and correlations are same for all assets. Note that our selection of the correlation matrix ρ guarantees that all entries in ρ's Cholesky decomposition L are positive, thus, as we pointed out at the end of 2.3.2, the method of antithetic variates will always reduce the variance in this setting.

Table 5.5: Parameter Settings for 6-Dimensional Basket Put Option

Parameters	Value
Asset num	4
Spot price	40
Strike price K	40
Time to maturity T	1
Volatility σ	0.2
Interest rate r	0.06
Dividend rate q	0.04
Correlation ρ	0.25
MC approx	1.50600

Monte Carlo Methods

We first numerically prove that the variance reduction techniques introduced in section 2.3 are indeed effective. In this experiment, we use 500 different random seeds to conduct the vanilla MC, the method of antithetic variates and the control variates methods 500 times, and compute the standard deviation of the output option price. The standard deviations of the three methods are shown in Table 5.6 and visualized in Figure 5.1. Since the low-discrepancy sequence is deterministic, we are not able to calculate the standard deviation of the methods that involve the Sobol sequence. From Table 5.6 and Figure 5.1, we see that the control variates method $>$ the method of antithetic variates $>$ vanilla Monte Carlo methods in variation reduction.

Table 5.6: Std dev of vanilla MC, antithetic variates and control variates (6D basket put option)

6D basket put option			
Standard Deviation of Monte Carlo Approx			
N	Vanilla MC	Antithetic Var	Control Var
2^{10}	7.31e-02	5.45e-02	4.41e-02
2^{11}	5.17e-02	3.88e-02	3.10e-02
2^{12}	3.66e-02	2.76e-02	2.17e-02
2^{13}	2.45e-02	1.95e-02	1.52e-02
2^{14}	1.78e-02	1.35e-02	1.07e-02
2^{15}	1.12e-02	9.44e-03	7.64e-03
2^{16}	8.69e-03	6.82e-03	5.28e-03
2^{17}	6.42e-03	4.64e-03	3.71e-03
2^{18}	4.46e-03	3.28e-03	2.59e-03

Figure 5.1: Visualization of the Std dev of vanilla MC, antithetic variates and control variates (6D basket put option)

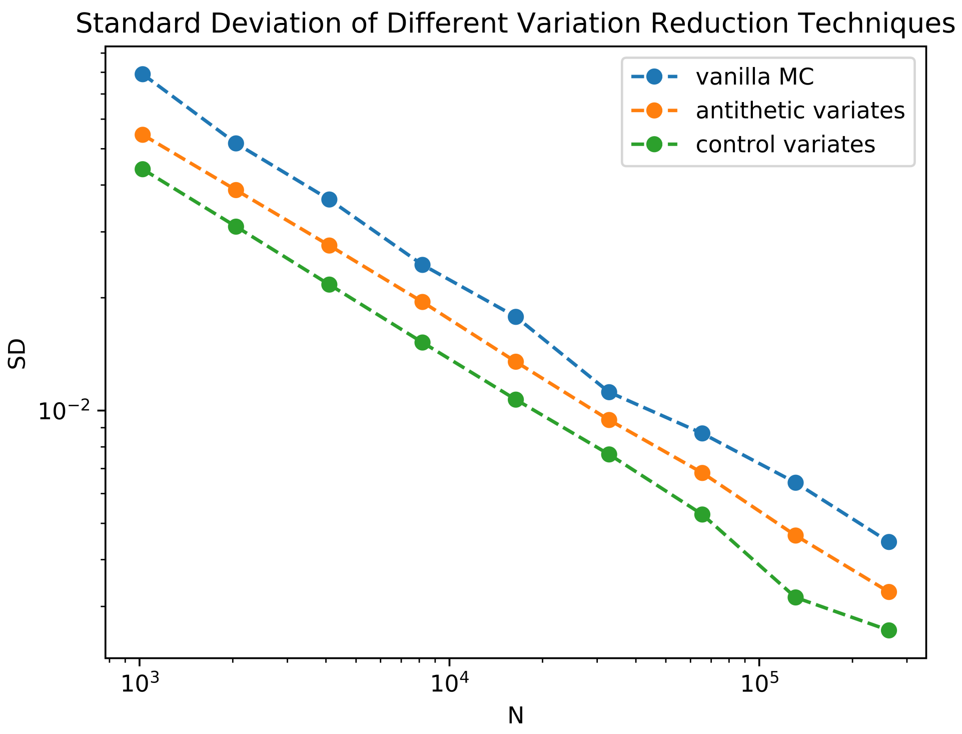


Table 5.7: Rel error and execution time of vanilla MC, antithetic variates and control variates (6D basket put option)

6D basket put option (price ≈ 1.50600)												
	Vanilla MC				Antithetic Variates				Control Variates			
N	Approx	Rel err	Order	Time	Approx	Rel err	Order	Time	Approx	Rel err	Order	Time
2^{14}	1.495	7.2e-03	NA	0.33	1.520	9.6e-03	NA	0.21	1.505	8.3e-04	NA	0.27
2^{15}	1.517	7.0e-03	0.0	0.67	1.499	4.5e-03	1.1	0.37	1.517	7.6e-03	-3.2	0.57
2^{16}	1.490	1.1e-02	-0.6	1.33	1.513	4.7e-03	-0.1	0.65	1.492	9.4e-03	-0.3	1.00
2^{17}	1.498	5.4e-03	1.0	2.72	1.495	7.6e-03	-0.7	1.31	1.496	6.3e-03	0.6	2.01
2^{18}	1.510	2.4e-03	1.2	5.52	1.498	5.1e-03	0.6	2.63	1.507	4.1e-04	3.9	4.01
2^{19}	1.501	3.1e-03	-0.4	11.11	1.506	1.6e-04	5.0	5.31	1.502	2.9e-03	-2.8	8.18
2^{20}	1.505	7.7e-04	2.0	22.17	1.502	2.9e-03	-4.2	10.65	1.505	4.6e-04	2.6	16.25

Table 5.8: Rel error and execution time of Sobol seq and Sobol seq with control variates (6D basket put option)

6D basket put option (price ≈ 1.50600)								
	Sobol Sequence				Control Variates + Sobol Sequence			
N	Approx	Rel err	Order	Time	Approx	Rel err	Order	Time
2^{14}	1.506	2.7e-04	NA	0.76	1.506	1.8e-04	NA	0.86
2^{15}	1.505	5.9e-04	-1.2	1.49	1.505	8.5e-04	-2.2	1.52
2^{16}	1.505	8.3e-04	-0.5	2.96	1.505	9.6e-04	-0.2	3.06
2^{17}	1.506	2.2e-04	1.9	5.97	1.506	2.9e-04	1.7	6.09
2^{18}	1.506	5.9e-05	1.9	11.94	1.506	1.9e-05	3.9	12.31
2^{19}	1.506	4.6e-06	3.7	24.08	1.506	2.4e-05	-0.3	24.73
2^{20}	1.506	3.2e-05	-2.8	48.29	1.506	4.3e-05	-0.8	49.45

FFT-based Convolution Method
From Table 5.9, we can see that the Conv method still works very well when the number of risk factors is 6. An approximation of the option price whose relative error is within 0.4% is outputted within 0.2 second. A more accurate approximation can be retrieved within 17 seconds by discretizing each axis into 16 intervals. Although we don't have analytical solutions to the hedge parameters, it's reasonable to believe that the approximation of hedge parameters will also be accurate, as we have shown in the last experiment.

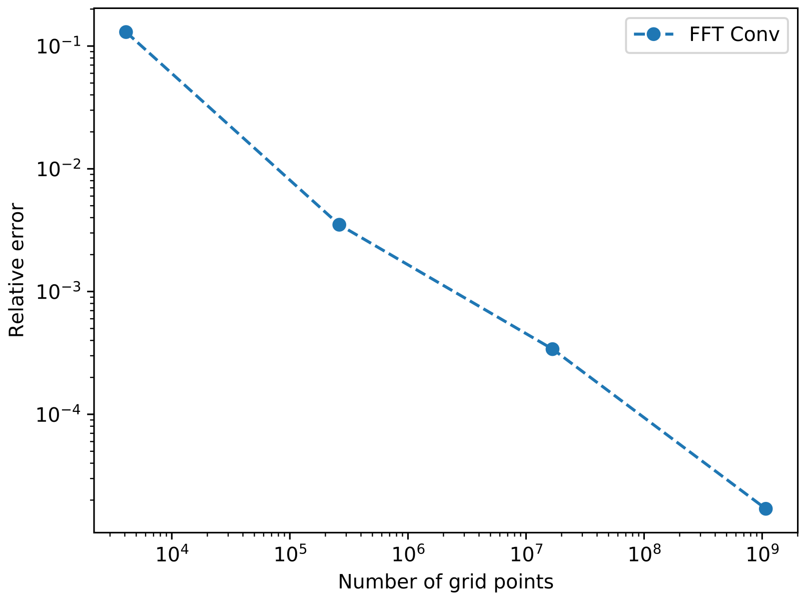
Notice that when one discretize each axis into 32 intervals, both the execution time and memory consumption are no longer affordable. By our observation, for a N-dimensional option pricing problem, the algorithm can be completed within 20 seconds by discretizing each axis into 2^{10-N} intervals. Based on our experience, one can get a reasonable estimation when N is smaller than 8 by using this formula to determine the grid size.

We plotted the relative error of Conv method versus number of grid points used in Figure 5.2. One can observe that the order of convergence is larger than 2. This may be caused by some properties of the basket option since we observed the same behaviour when we compute the 4-d basket option price. Thus, we surmised that the basket option pricing is highly compatible with the Conv method.

Table 5.9: Rel error in prices and deltas and execution time of the Conv method (6D basket put option)

6D basket put option (price ≈ 1.50600)				
FFT-based Convolution Method				
N	Approx	Rel err	Order	CPU Time
4^6	1.704	1.3e-01	NA	0.03
8^6	1.511	3.5e-03	5.2	0.19
16^6	1.507	3.4e-04	3.4	16.40
32^6	1.506	1.7e-05	4.3	1159.92

Figure 5.2: FFT Conv method's accuracy versus number of grid points (6D basket put option)



5.4 Experiments: American Put Option

In this experiment, we will show the advantage of the deep Galerkin method in American option pricing and compare it with Monte Carlo methods. Recall that as the number of underlying assets increase, the execution time of LSMC grows exponentially, while the training time of the deep Galerkin method grows linearly if the Monte Carlo method for computing second derivatives is used. Therefore, we can claim that the deep Galerkin method will always outperform the least square Monte Carlo method in computational efficiency if the deep Galerkin method performs better even when there is only one risk factor.

The parameter settings are shown in Table 5.10. Since we don't have an analytical solution to the American option price, we will use the finite difference method to approximate the exact solution. We picked the domain of the stock price to be [0, 300] and discretized it into 1500 uniform intervals. The time domain was discretized into 600 uniform intervals. Although the finite difference method outputs the option prices on the whole domain, in the Monte Carlo method's experiment, we will only examine the case when the spot price is 50 and the time to maturity is 1. On the contrary, since the deep Galerkin method also outputs all the option prices, we will contrast its approximation with the finite difference method's approximation.

Table 5.10: Parameter Settings for 1-Dimensional American Put Option

Parameters	Value
Asset num	1
Spot price	50
Strike price K	50
Time to maturity T	1
Volatility σ	0.5
Interest rate r	0.05
Dividend rate q	0
FD approx	8.72334

Monte Carlo Methods

We approximate the American option using a Bermudan option that allows exercise every day, i.e., the Monte Carlo simulation has 365 equidistant time steps. By contrasting Table 5.4 with Table 5.3 and 5.7, we see that it takes much longer time to price an American option than a European option. More importantly, by comparing Table 5.12 with Table 5.6, one can observe that although more time has been put in, the approximations of the American option price have larger standard deviations, i.e., larger errors. Therefore, the pricing of American options is much more difficult than the pricing of European options by Monte Carlo methods.

Table 5.11: Rel error and execution time of vanilla MC (1D American put option)

1D American put option (price ≈ 8.72334)				
	Vanilla LSMC			
N	Approx	Rel err	Order	Time
2^{10}	8.598	1.4e-02	NA	62.07
2^{11}	8.645	9.0e-03	0.7	119.89
2^{12}	8.808	9.7e-03	-0.1	232.97
2^{13}	8.765	4.8e-03	1.0	468.11
2^{14}	8.701	2.6e-03	0.9	896.49
2^{15}	8.699	2.8e-03	-0.1	1631.96

Table 5.12: Std dev of vanilla LSMC (1D American put option) (for each N, 50 experiments are conducted)

1D American put option	
Standard Deviation of Approx	
N	LSMC
2^{10}	2.91e-01
2^{11}	2.20e-01
2^{12}	1.63e-01
2^{13}	1.22e-01
2^{14}	9.48e-02
2^{15}	6.05e-02

Deep Galerkin Method

In this experiment, for each batch we sample 2000 interior points and 100 terminal points. Since our batch size is not too small, we train the model 10 times using a same batch during one iteration to save the sampling time. We plot the loss versus iterations graph in Figure 5.3 where $L1$ represents the first loss term in 4.3, and so on. One can observe that $L1$ and $L4$ contribute the most to the total loss. Recall that $L4$ loss represent the approximation of the terminal payoff function, which is not smooth at the at-the-money point. Due to our design of the neural network architecture, it's difficult to "learn" the non-smoothness. However, we can see that the error at the terminal time does not propagate too far from Figure 5.4 and 5.5. More importantly, the average absolute error at time 0 is around 0.043, which is regarded as a good approximation.

Figure 5.3: Visualization of the loss versus training iterations (1D American put option)

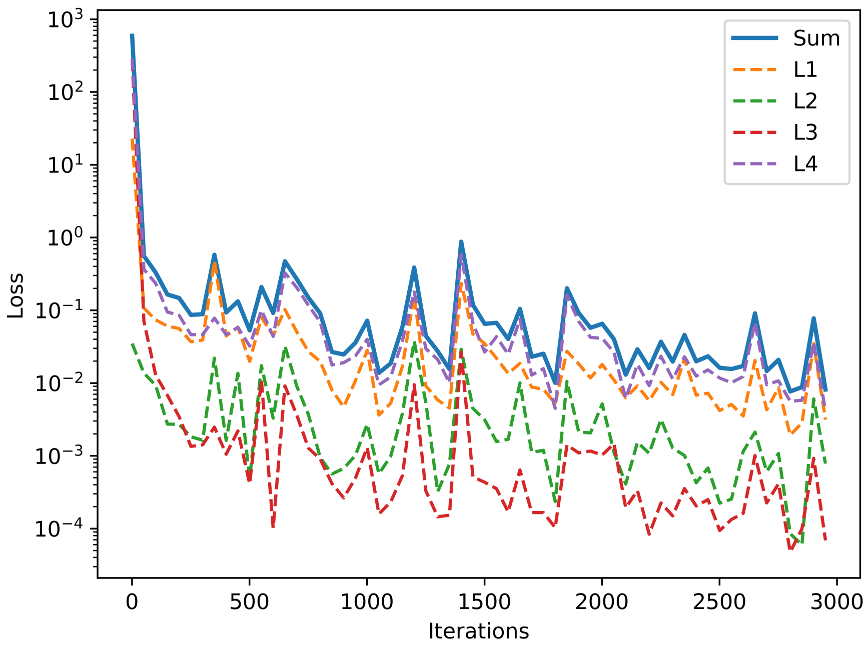


Figure 5.4: DGM's approximation versus FDM's approximation at different times (1D American put option)

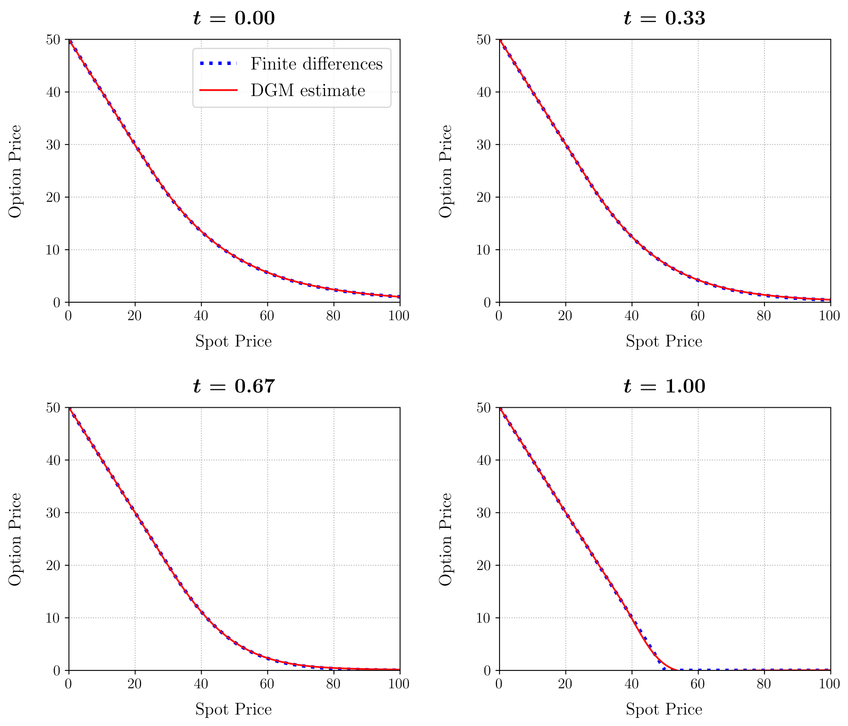
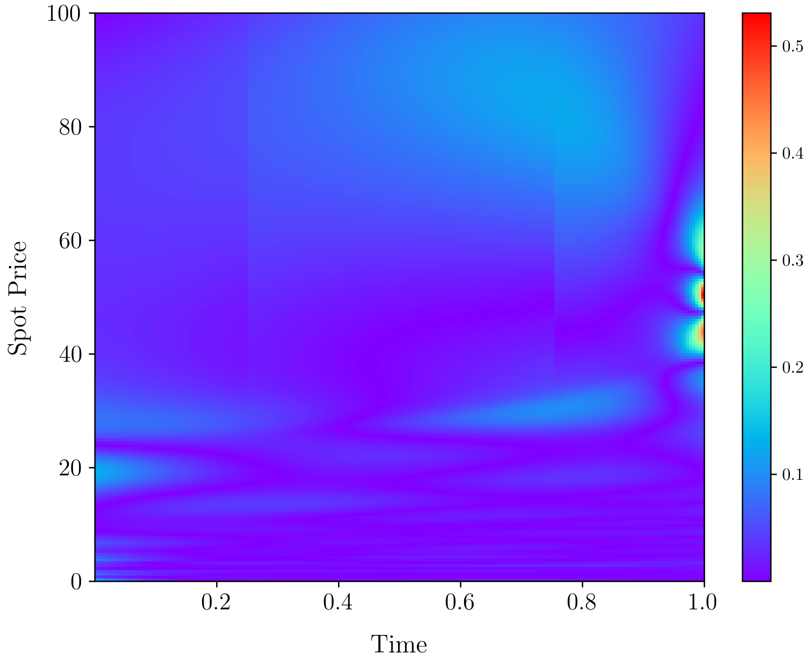


Figure 5.5: DGM's approximation's absolute error on the whole domain (1D American put option)



Comparisons

Computational Efficiency in One Dimension
For the LSMC method, it takes around 500 seconds for one to get an approximation with the absolute error within 0.04. For the deep Galerkin method, it takes 1742 seconds to train the neural network, and the neural network's average absolute error at time 0 is around 0.043. If one also needs to calculate the hedge parameters, 1500 seconds of computation will be needed in total by the LSMC method. However, the neural network has already given an accurate range of approximations, which allows us to compute the hedge parameters and option prices with respect to other spot prices both efficiently and accurately. Hence, the DGM outperforms the LSMC method in computational efficiency when a range of option prices are needed.

Computational Complexity
For the LSMC method, as the number of risk factors increases, the number of the monomial basis used in the least square regression increases exponentially. Thus, it is unrealistic to apply LSMC to high-dimensional option pricing problems. On the contrary, the training time of the deep Galerkin method grows linearly if the Monte Carlo method for computing second derivatives is used. According to [1], the DGM can output accurate American option prices even when the number of risk factors is equal to 200. Therefore, we claim that the deep Galerkin method is better than Monte Carlo methods in high-dimensional option pricing problems.

6 Conclusions and Future Work

Admittedly, vanilla/quasi-Monte Carlo methods with variation reduction techniques are good choices for pricing multi-asset European options since they do not suffer from the curse of dimensionality and are easy to implement. But in certain cases, the Monte Carlo method is not the optimal choice. Therefore, in this paper, we present two different algorithms that are better than Monte Carlo methods in different aspects:

- For the FFT-based convolution method, although it suffers from the dimensionality issues, we observe that it gives more accurate approximations in a shorter amount of time compared with the Monte Carlo methods when the number of risk factors is less than 8. Notably, it outputs a range of option prices, which facilitate the calculation of hedge parameters and the curve building. However, it only allows for pricing under models with a characteristic function while Monte Carlo methods do not have such a restriction. Therefore, the Conv method outperforms Monte Carlo methods for multi-asset European option pricing under Lévy processes when the number of risk factors is less than 8.
- For the deep Galerkin method, the computational cost always grows linearly as the number of risk factors increases as long as one uses the Monte Carlo method to approximate the Hessian. On the contrary, we have shown in 2.2 that the computational complexity of LSMC is $\mathcal{O}(NMd^{2\chi})$, which is worse than quadratic in the dimensionality. Thus, the deep Galerkin method is asymptotically more efficient than the least square Monte Carlo method in American option pricing. The deep Galerkin method also outputs the approximation on the whole stock price and time domains as the regular PDE method does, which is useful in building curves and calculating hedge parameters. More importantly, DGM produces considerably accurate approximations in the

numerical experiments. Thus, the deep Galerkin method outperforms the least square Monte Carlo method for multi-asset American option pricing.

We also proved an error bound of the deep Galerkin method's approximation of the solution to the Black-Scholes equation by means of the maximum principle. This alleviates the uncertainty of the deep Galerkin method compared with Monte Carlo methods since we can only use a confidence interval to approximate the error of Monte Carlo methods while the deep Galerkin method's absolute error bound can be calculated based on our work. There are several possible extension to this bound:

1. The error bound we derived is not too tight. One can attempt to derive a better bound by some more advanced techniques.
2. We only considered the Dirichlet boundary condition, while the Neumann and Robin boundary conditions also frequently appear in the high dimensional problems. Thus, it will be valuable to consider error bound where the PDE has a different boundary condition.
3. For exotic derivatives, e.g., American options, path-dependent options, the resulting PDEs are usually nonlinear and difficult to tackle. An error bound for these problems will also be very useful.

Apart from these, other numerical methods for pricing American options that will not suffer from dimensionality issues are worth looking into.

References

[1] J. Sirignano and K. Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. *The Journal of Computational Physics, 375*, pages 1339–1364, 2018.

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

[3] C. C. W. Leentvaar and C. W. Oosterlee. On coordinate transformation and grid stretching for sparse grid pricing of basket options. *Journal of Computational and Applied Mathematics, 222 (1)*, pages 193–209, 2008.

[4] R. Lord, F. Fang, F. Bervoets, and C. W. Oosterlee. A fast and accurate fft-based method for pricing early-exercise options under lévy processes. *SIAM J. Sci. Comput., 30(4)*, page 1678–1705, 2008.

[5] C. C. W. Leentvaar and C. W. Oosterlee. Multi-asset option pricing using a parallel fourier-based technique. *The Journal of Computational Finance, 12(1)*, 2008.

[6] F. Black and M. Scholes. The pricing of options and corporate liabilities. pages 637–654, 1973.

[7] K. Jackson, S. Jaimungal, and V. Surkov. Fourier space time-stepping for option pricing with lévy models. *The Journal of Computational Finance, 12 (2)*, pages 1–29, 2008.

[8] F. A. Longstaff and E. S. Schwartz. Valuing american options by simulation: A simple least-squares approach. *The Review of Financial Studies, 14(1)*, 2001.

[9] B. Vandewoestyne and R. Cools. Good permutations for deterministic scrambled halton sequences in terms of l_2-discrepancy. *Journal of Computational and Applied Mathematics, 189 (1-2)*, pages 341–361, 2006.

[10] P. Glasserman. *Monte Carlo Methods in Financial Engineering*. Springer, 2003.

[11] A. Hirsa. *Computational Methods in Finance*. CRC Press, 2012.

[12] P. Carr and D. B. Madan. Option valuation using the fast fourier transform. *The Journal of Computational Finance, 2*, 1999.

[13] C. O'Sullivan. Path dependent option pricing under lévy processes. *EFA 2005 Moscow Meetings Paper*, 2005.

[14] A. Al-Aradi, A. Correia, D. F. Naiff, G. Jardim, and Y. Saporito. Applications of the deep galerkin method to solving partial integro-differential and hamilton-jacobi-bellman equations. *arXiv:1912.01455*, 2019.

[15] Y. Chen and J. W. L. Wan. Deep neural network framework based on backward stochastic differential equations for pricing and hedging american options in high dimensions. *arXiv:1909.11532*, 2019.