MSc thesis in Applied Mathematics

Pricing options under Merton jump–diffusion dynamics with deep-learning based numerical methods

Yufan Zhou

2026

TU Delft

MSc thesis in Applied Mathematics

Pricing options under Merton jump–diffusion dynamics with deep-learning based numerical methods

Yufan Zhou

September 2026

A thesis submitted to the Delft University of Technology in partial fulfillment of the requirements for the degree of Master of Applied Mathematics

Yufan Zhou: *Pricing options under Merton jump–diffusion dynamics with deep-learning based numerical methods* (2026)

Thesis committee: Prof.dr. Cornelis Vuik
Dr. Shuaiqiang Liu (daily supervisor)
Dr. Fenghui Yu

Abstract

Pricing Bermudan options under high-dimensional jump-diffusion dynamics is challenging because the valuation problem combines high dimensionality, early-exercise decisions, and discontinuous jumps. This thesis addresses this problem under multidimensional Merton jump-diffusion (MJD) dynamics by extending the deep backward dynamic programming (DBDP) scheme [1], a deep-learning based numerical method that trains local neural networks sequentially backward over the discretized time steps, to forward-backward stochastic differential equations (FBSDEs) with jumps. To incorporate jumps, the proposed scheme, DBDP-Jump, separates the jump integral into realized-jump and compensator contributions. Both are evaluated using the learned next-step value function, with auxiliary Monte Carlo sampling used for the compensator. This differs from the existing DBDP extension in [2], which introduces an additional neural network to approximate the jump integrand. Bermudan exercise is incorporated through a maximum projection between the continuation estimate and the immediate exercise payoff at each admissible exercise date. The convergence analysis extends the error framework of DBDP to the jump setting by accounting for the additional approximation errors introduced by the realized-jump and compensator terms. The resulting analysis shows that convergence is retained under time-grid refinement when the local neural-network approximation errors are sufficiently controlled. A grid refinement study provides numerical results consistent with this analysis. Numerical experiments evaluate DBDP-Jump from 1 to 200 dimensions under different MJD parameters, payoff structures, and exercise settings. The scheme achieves reasonable accuracy where independent benchmarks are available and remains stable in high dimensions.

Acknowledgements

First, I would like to express my gratitude to my daily supervisor, Shuaiqiang Liu. He is very organized and has provided me with all the important information I needed throughout the thesis process. He is patient and experienced, and his guidance has helped keep me on track, especially when my mind sometimes wandered. I would also like to thank my responsible supervisor, Kees Vuik, for overseeing my thesis process. He always responded very quickly at every essential point, carefully read my results, and provided valuable suggestions and support. I am also grateful to Fenghui Yu for taking the time to be a member of my thesis committee.

I would like to thank my parents for providing financial support and giving me a comfortable and stress-free environment in which to complete my master's degree. I would also like to thank all the friends who have cared about my progress, asked how my thesis was going, offered encouragement, or shared their own experiences with me. Finally, I am satisfied with the effort I put into this thesis over the past few months and glad to have overcome the struggles along the way.

Contents

1 Introduction **1**

2 Problem formulation **5**
2.1 Bermudan options . . . 5
2.2 Multidimensional Merton jump-diffusion process . . . 5
2.3 Pricing Bermudan options under MJD . . . 6

3 DBDP-Jump scheme **11**
3.1 From the FBSDE formulation to discrete time . . . 11
3.2 Neural-network approximation . . . 13
3.3 Approximation of jump-related terms . . . 15
3.4 Baseline DBDP-Jump algorithm . . . 15

4 Convergence analysis **17**
4.1 Model assumptions . . . 17
4.2 Time discretization error . . . 18
4.3 Neural-network approximation error . . . 22
4.4 Consistency estimate for DBDP-Jump scheme . . . 23

5 Improved Monte Carlo simulations for DBDP-Jump **25**
5.1 Offline MC . . . 25
5.2 Enhanced MC paths for estimating the early-exercise boundary . . . 26
5.2.1 Sample Enrichment . . . 26
5.2.2 Existing MC approaches . . . 30

6 Numerical results **33**
6.1 Problem setup . . . 33
6.2 DBDP-Jump implementation . . . 34
6.3 Sanity check . . . 35
6.4 Grid refinement study . . . 36
6.5 Efficiency of sample enrichment . . . 37
6.6 High-dimensional problems . . . 41
6.6.1 20D and 50D problems . . . 41
6.6.2 100D and 200D problems . . . 45

7 Discussion **47**
7.1 Main findings . . . 47
7.2 Practical considerations for time discretization and training hyperparameters . . . 47
7.3 Use and limitations of sample enrichment . . . 48

A DBDP-Jump algorithm **49**

B Loss explosions and the need for adaptive learning-rate control in DBDP-Jump **51**

C DBDP-Jump under high jump intensity **55**

1 Introduction

A Bermudan option can be exercised on any date in a predetermined finite set, unlike a European option, which can be exercised only at maturity. Under risk-neutral valuation, a rational holder selects an admissible stopping time (i.e., an exercise date) to maximize the expected discounted payoff. Bermudan option pricing is therefore naturally formulated as a optimal stopping problem in discrete time. By the dynamic programming principle, at each exercise date, the holder compares the immediate exercise payoff with the value of retaining the option and exercising optimally thereafter. The latter is represented by the so-called continuation value, defined as the risk-neutral conditional expectation of the discounted option value at the next exercise date, given the currently available information. Therefore, the option price and the optimal exercise policy can be obtained by starting from the terminal payoff and applying this comparison recursively backward through the exercise dates [3].

Numerical methods for Bermudan option pricing generally share a backward structure but differ in how they approximate the continuation value. They can be classified as partial differential equation (PDE)-based, tree-based, transform-based, or simulation-based methods. PDE-based methods discretize the pricing equation using, for example, finite-difference or finite-element schemes [4, 5]. Tree-based methods approximate the state dynamics with recombining binomial [6] or multinomial trees. Transform-based methods instead evaluate the conditional expectation through Fourier representations. Representative examples include fast Fourier transform (FFT) convolution [7] and the Fourier-cosine (COS) method [8]. These three classes are most practical in one or two dimensions. As the dimension increases, the required number of grid points, tree nodes, or expansion coefficients can grow exponentially, reflecting the curse of dimensionality [9, 10]. Simulation-based methods avoid full tensor-product discretizations and therefore alleviate this difficulty considerably. Monte Carlo methods generate sample paths and approximate continuation values using mesh-based or regression estimators. Reported applications of the stochastic mesh method [11] and the least-squares Monte Carlo (LSMC) method [12] reach seven and twenty state variables, respectively. However, as the dimension increases, the complexity of approximating continuation values through regression or mesh-based estimators can still increase substantially [11].

To solve problems in even higher dimensions, neural networks (NNs) are considered as a promising tool. The work in [13] can be viewed as a direct continuation of LSMC [12], in which the classical regression basis is replaced by a neural networks structure to approximate the continuation value. A different perspective is adopted in [14], where deep learning is used to approximate the optimal stopping rule itself rather than explicitly constructing a backward recursion for the continuation value. Another line of research is based on the backward stochastic differential equation (BSDE) representation, through the nonlinear Feynman-Kac formula. In this direction, the deep BSDE method proposed in [15] uses a forward discretization of the BSDE and defines the loss through the terminal condition, thereby training NNs to approximate the initial value. The method has been shown to price European options up to 100 dimensions. Building on this idea, [16] reformulates the loss function as to reduce the variance among the simulated estimates of the initial option value based on the fact that the true

initial price should be path-independent, so the method can be implemented in backward and applied to Bermudan swaption pricing and hedging in the Libor market model. Their results demonstrate that backward neural-network schemes can effectively address high-dimensional Bermudan option pricing problems. This observation connects Bermudan pricing to a broader class of backward deep learning methods that were not originally developed for Bermudan options, but whose structure is highly relevant. A representative example is the deep backward dynamic programming (DBDP) scheme proposed in [1]. This method also relies on the BSDE representation of semilinear PDEs, but uses NNs to approximate the value function and its gradient at each time step by minimizing a local loss in a backward manner. Deep splitting [17] shares the same spirit, relying on local recursive approximations, although it originates from a splitting-based Feynman-Kac construction rather than from a BSDE representation.

Pricing Bermudan options under jump-diffusion dynamics requires additional treatment of the discontinuous jump component, which increases the complexity of the pricing problem. This extension is nevertheless financially relevant because jump-diffusion models can represent sudden asset-price movements more realistically than pure diffusion models. In particular, they can reproduce empirical features such as heavier tails, higher peaks, asymmetry, and volatility smiles [18]. Numerically, the jump dynamics must be incorporated into the approximation of state transitions or continuation values. In low to moderate dimensions, the binomial model [6] is extended in [19] by superimposing multivariate jumps. Under a differential formulation, the pricing equation between exercise dates becomes a partial integro-differential equation (PIDE); a representative finite-difference treatment under the Kou model is presented in [20]. For simulation-based pricing, the Stochastic Grid Bundling Method (SGBM) introduced in [21] is applied to Merton jump-diffusion dynamics in [22]. SGBM bundles simulated paths and applies local regressions to approximate continuation values, reducing regression bias and improving numerical tractability. The reported tests cover up to five dimensions and show lower computational time than the standard least-squares method at comparable accuracy. For higher-dimensional problems, the literature using NNs specifically for options with early-exercise opportunities under jump-diffusion dynamics remains limited. Most existing studies instead consider European options. On the PIDE side, a temporal-difference-learning approach [23] uses NNs to represent both the solution and the nonlocal terms of the equation. On the stochastic representation side, work in [24] reformulates forward-backward stochastic differential equations (FBSDEs) with jumps by removing the random jump term while retaining the compensator correction, so that the modified system still yields the same PIDE value at the initial point. This simplifies the construction of a forward numerical scheme without nested conditional expectations across time steps. Another development is the extension of the deep BSDE method [15] to FBSDEs with jumps [25], which introduces additional neural networks to approximate nonlocal jump terms and has been implemented for European option pricing in both low and high dimensions, including dimensions up to 100. Finally, work in [2] directly extends DBDP [1] by introducing an additional NN to approximate the nonlocal jump term. However, the work focuses on the construction and consistency analysis of the numerical scheme, and no numerical experiments are provided.

This thesis addresses the gap in pricing Bermudan options in high dimensions under jump-diffusion dynamics and focuses on the multidimensional Merton jump-diffusion model [26]. Because backward deep-learning schemes align naturally with the Bellman recursion of Bermudan valuation, the DBDP framework [1] is adopted. The continuation problems between successive exercise dates are formulated as a sequence of PIDEs and represented by corresponding BSDEs with jumps through the Feynman-Kac formula for jump diffusions. In contrast to the DBDP extension reviewed above, which introduces an additional NN for the jump integrand, the proposed scheme evaluates the realized-jump and compensator terms from

the learned next-step value function. Both terms are precomputed outside local NN optimization, using pathwise realized jumps and fixed auxiliary Monte Carlo samples for the compensator; the resulting scheme is termed DBDP-Jump. An error analysis is provided, and a data-driven sample-enrichment procedure is investigated to improve exercise-boundary estimation at early times. Numerical experiments evaluate DBDP-Jump across dimensions, model parameter settings, and payoff functions. Where independent benchmarks are available, the method achieves reasonable accuracy; tests on Bermudan and American-style options up to 200 dimensions further demonstrate stable behavior and favorable empirical scaling.

The remainder of the thesis first formulates the pricing problem and presents DBDP-Jump together with its error analysis. It then describes the Monte Carlo components used in the implementation and the sample-enrichment procedure, reports the numerical evaluation, and concludes with a discussion of practical considerations and limitations.

2 Problem formulation

Since this thesis focuses on high-dimensional Bermudan option pricing under the MJD model from a backward stochastic differential equation with jumps (BSDEj) perspective, this chapter will first give definitions of Bermudan options and the multidimensional MJD model and then formulate the Bermudan option pricing problem to a backward iteration over a sequence of forward-backward stochastic differential equations (FBSDEs) with jumps via the extended Feynman-Kac theorem [27].

All processes are defined on a filtered probability space $(\Omega, \mathcal{F}, \{\mathcal{F}_t\}_{0\le t\le T}, \mathbb{Q})$ satisfying the usual conditions, where $\mathbb{Q}$ is a risk-neutral measure. The market is assumed to be arbitrage-free, and the asset price dynamics as well as the option price are specified under $\mathbb{Q}$.

The filtration $\{\mathcal{F}_t\}_{0\le t\le T}$ is generated by the following two independent sources of randomness:

- a d-dimensional Brownian motion $W_t = (W_t^1, \dots, W_t^d)^\top$, where $d \in \mathbb{N} \setminus \{0\}$, with correlation matrix $\boldsymbol{\rho}_W := (\rho_{ij})_{1\le i,j\le d}$;
- a Poisson random measure $N(dt, dz)$ on $[0, T] \times \mathbb{R}^d$ with compensator $\nu(dz)\, dt$, so that all d assets jump simultaneously.

The compensated Poisson random measure is $\tilde{N}(dt, dz) = N(dt, dz) - \nu(dz)\, dt$.

2.1 Bermudan options

A Bermudan option is an early-exercise derivative whose holder may exercise the option only at a contractually specified finite set of dates before or at maturity, rather than continuously over time as in the American case or only at maturity as in the European case. Let

$$\mathcal{T} = \{t_0, \dots, t_M\}, \qquad 0 = t_0 < \cdots < t_M = T,$$

denote the set of admissible exercise dates. The exercise decision of a Bermudan option is restricted to the finite time set $\mathcal{T}$. For a payoff function $g(\cdot) : \mathbb{R}_+^d \to \mathbb{R}$, exercising the option at $t_m \in \mathcal{T}$ yields the payoff $g(X_{t_m})$.

2.2 Multidimensional Merton jump-diffusion process

In the multidimensional MJD model the log-jump vector $Z = (Z^1, \dots, Z^d)^\top \in \mathbb{R}^d$ is assumed to be multivariate normal,

$$Z \sim \mathcal{N}_d(\mu_J, \Sigma_J), \tag{2.1}$$

with

- $\mu_J = (\mu_J^1, \dots, \mu_J^d)^\top \in \mathbb{R}^d$: the vector of per-asset log-jump means;
- $\Sigma_J \in \mathbb{R}^{d\times d}$: the jump covariance matrix, with entries $(\Sigma_J)_{ij} = \rho_J^{ij}\,\sigma_J^i\,\sigma_J^j$, where $\sigma_J^i > 0$ is standard deviation of the logarithmic jump size of asset i and $\rho_J^{ij} \in [-1, 1]$ is the jump correlation between assets i and j.

Since Z admits a density f_Z, the compensator takes the form $\nu(dz) = \lambda\, f_Z(z)\, dz$, where $\lambda \geq 0$ is the common jump intensity.

Let $X_t = (X_t^1, \dots, X_t^d)^\top \in \mathbb{R}_+^d$ denote the price vector of d underlying assets at time t. Under the MJD model, for each $i = 1, \dots, d$, the price process X_t^i follows the SDE:

$$\begin{cases} \dfrac{dX_t^i}{X_{t-}^i} = (r - \lambda\kappa^i)\,dt + \sigma_i\,dW_t^i + \displaystyle\int_{\mathbb{R}^d} (e^{z^i} - 1)\,N(dt, dz), \quad i = 1, \dots, d, \\ X_0^i = x_0^i, \end{cases} \tag{2.2}$$

or equivalently, in compensated form,

$$\frac{dX_t^i}{X_{t-}^i} = r\,dt + \sigma_i\,dW_t^i + \int_{\mathbb{R}^d} (e^{z^i} - 1)\,\tilde{N}(dt, dz), \quad i = 1, \dots, d, \tag{2.3}$$

where $r > 0$ is the constant risk-free interest rate, $\sigma_i > 0$ is the volatility of asset i, and

$$\kappa^i = \int_{\mathbb{R}^d} (e^{z^i} - 1) f_Z(z)\,dz = \mathbb{E}^{\mathbb{Q}}[e^{Z^i} - 1] = e^{\mu_J^i + \frac{1}{2}(\sigma_J^i)^2} - 1 \tag{2.4}$$

The explicit solution is

$$X_t^i = X_0^i \exp\Big(\big(r - \tfrac{1}{2}\sigma_i^2 - \lambda\kappa^i\big)t + \sigma_i W_t^i\Big) \prod_{k=1}^{N_t} e^{Z_k^i}, \quad i = 1, \dots, d, \tag{2.5}$$

where N_t is the Poisson counting process with intensity λ, so that N_t gives the number of jumps up to time t, and the log-jump vectors $Z_k = (Z_k^1, \dots, Z_k^d)^\top$ are i.i.d. draws from $\mathcal{N}_d(\mu_J, \Sigma_J)$.

Applying Itô's formula for jump-diffusion processes to a spatial test function $\varphi \in C^2(\mathbb{R}_+^d)$, the infinitesimal generator $\mathcal{L}$ of the multidimensional Merton jump-diffusion process X is given by

$$(\mathcal{L}\varphi)(x) = \sum_{i=1}^d (r - \lambda\kappa^i) x_i \partial_{x_i}\varphi(x) + \frac{1}{2}\sum_{i,j=1}^d (\sigma_i\sigma_j\rho_{ij}) x_i x_j \partial_{x_i x_j}\varphi(x) + \lambda \int_{\mathbb{R}^d} \Big(\varphi(x \odot e^z) - \varphi(x)\Big) f_Z(z)\,dz. \tag{2.6}$$

where $\odot$ denotes componentwise multiplication, and this notation is used throughout the thesis.

2.3 Pricing Bermudan options under MJD

Assume the option starts at time 0 and matures at time T, and that the underlying asset process follows the MJD dynamics in (2.2). The money-market account is chosen as numeraire

and is given by $B_t = e^{rt}$, so the discounted asset price process $\frac{X_t}{B_t}$ is a martingale under $\mathbb{Q}$. The holder chooses an exercise time $\tau \in \mathcal{T}$. If exercise occurs at τ, the contract pays $g(X_\tau)$, whose no-arbitrage value at t_0 is $\mathbb{E}^{\mathbb{Q}}\left[\frac{g(X_\tau)}{B_\tau} \mid \mathcal{F}_0\right]$. Optimizing over admissible exercise times gives the Bermudan price. Thus, valuing a Bermudan option is an optimal stopping problem:

$$V_0 = \sup_{\tau \in \mathcal{T}} \mathbb{E}^{\mathbb{Q}}\left[\frac{g(X_\tau)}{B_\tau} \mid \mathcal{F}_0\right]. \tag{2.7}$$

This is a stochastic optimal control with:

- state: X the underlying asset price process,
- control: $\tau \in \mathcal{T}$, the stopping decision,
- objective function: $\sup_{\tau \in \mathcal{T}} \mathbb{E}^{\mathbb{Q}}\left[\frac{g(X_\tau)}{B_\tau} \mid \mathcal{F}_0\right]$.

As X is Markov, the option value depends only on the current state, hence can be simplified to

$$V_0(x_0) = \sup_{\tau \in \mathcal{T}} \mathbb{E}^{\mathbb{Q}}\left[\frac{g(X_\tau)}{B_\tau} \mid X_0 = x_0\right]. \tag{2.8}$$

By the dynamic programming principle, at each possible exercise date $t_m \in \mathcal{T}$, the option holder compares the immediate exercise payoff $g(X_{t_m})$ with the continuation value, which is the conditional expectation of the discounted option value until the next exercisable date t_{m+1}, and chooses the one that gives a higher payoff. Therefore, at each exercisable date $t_m \in \mathcal{T}$ the option value satisfies the Bellman equation:

$$V_{t_m}(x) = \max\left\{g(x), C_{t_m}(x)\right\}. \tag{2.9}$$

where the continuation value is defined as

$$C_{t_m}(x) = \mathbb{E}^{\mathbb{Q}}\left[\frac{B_{t_m}}{B_{t_{m+1}}} V_{t_{m+1}}(X_{t_{m+1}}) \mid X_{t_m} = x\right]. \tag{2.10}$$

Hence, the initial time value function $V_0(x)$ can be solved by following backward recursion: At maturity T, the option value equals its payoff,

$$V_T(X_T) = g(X_T).$$

For $m = M-1, \ldots, 0$,

$$V_{t_m}(X_{t_m}) = \max\left\{g(X_{t_m}), C_{t_m}(X_{t_m})\right\}.$$

where $C_{t_m}(X_{t_m})$ is the continuation value defined in (2.10).

In each period $[t_m, t_{m+1}]$ between the recursion steps, the option can be treated as a European-style claim with payoff $V_{t_{m+1}}(X_{t_{m+1}})$ at time t_{m+1} as $V_{t_{m+1}}$ has been determined in the previous recursion step. Thus, the continuation value $C_{t_m}(X_{t_m})$ can be viewed as the price of this European claim at time t_m and the discounted continuation value $\frac{C_t(X_t)}{B_t}$ is a Martingale under $\mathbb{Q}$.

Define, for $t \in [t_m, t_{m+1}]$,

$$u(t,x) = \mathbb{E}^{\mathbb{Q}}\left[\frac{B_t}{B_{t_{m+1}}} V_{t_{m+1}}(X_{t_{m+1}}) \mid X_t = x\right].$$

Then $C_t(X_t) = u(t, X_t)$.

Applying Itô's formula for jump-diffusions to the process $u(t, X_t)$ yields

$$\begin{aligned} du(t,X_t) = \Big(\partial_t u + \mathcal{L}u\Big)(t,X_{t-})\,dt + \sum_{i=1}^{d} \sigma_i X^i_{t-}\,\partial_{x_i} u(t,X_{t-})\,dW^i_t \\ + \int_{\mathbb{R}^d} \Big(u(t,X_{t-}\odot e^z) - u(t,X_{t-})\Big)\,\tilde{N}(dt,dz), \end{aligned} \tag{2.11}$$

where the infinitesimal generator $\mathcal{L}$ is given by (2.6). Given $B_t = e^{rt}$, then

$$d\left(\frac{u(t,X_t)}{B_t}\right) = \frac{1}{B_t}\Big(du(t,X_t) - r\,u(t,X_t)\,dt\Big).$$

Since the discounted continuation value process $\frac{u(t,X_t)}{B_t}$ is a $\mathbb{Q}$-martingale on $[t_m, t_{m+1}]$, its drift must be zero, and this gives

$$\partial_t u(t,x) + \mathcal{L}u(t,x) - r\,u(t,x) = 0, \qquad (t,x) \in [t_m, t_{m+1}) \times \mathbb{R}^d_+,$$

with terminal condition $u(t_{m+1}, x) = V_{t_{m+1}}(x)$.

Therefore, the function u solves the PIDE on $[t_m, t_{m+1}] \times \mathbb{R}^d_+$:

$$\begin{cases} \partial_t u(t,x) + \mathcal{L}u(t,x) - ru(t,x) = 0, & (t,x) \in [t_m, t_{m+1}) \times \mathbb{R}^d_+, \\ u(t_{m+1},x) = V_{t_{m+1}}(x), & x \in \mathbb{R}^d_+. \end{cases} \tag{2.12}$$

By the nonlinear Feynman–Kac formula, follow the derivation:
Set option value at $t \in [t_m, t_{m+1}]$ as $Y_t := u(t, X_t)$, then $dY_t = du(t, X_t)$ given by (2.11). Since u solves (2.12), it follows

$$dY_t = r\,u(t,X_{t-})\,dt + \big(\sigma_X(X_{t-})\nabla_x u(t,X_{t-})\big)^\top dW_t + \int_{\mathbb{R}^d} \big(u(t,X_{t-}\odot e^z) - u(t,X_{t-})\big)\,\tilde{N}(dt,dz),$$

where $\sigma_X(x) := \operatorname{diag}(\sigma_1 x_1, \ldots, \sigma_d x_d)$, and $\nabla_x u = (\partial_{x_1} u, \ldots, \partial_{x_d} u)^\top$ is the gradient of u with respect to x. Define the Brownian and jump integrands by

$$Z_t := \sigma_X(X_{t-})\nabla_x u(t,X_{t-}), \tag{2.13}$$

$$U_t(z) := u(t, X_{t-}\odot e^z) - u(t,X_{t-}). \tag{2.14}$$

Then dY_t can be rewritten as

$$dY_t = rY_t\,dt + Z_t^\top dW_t + \int_{\mathbb{R}^d} U_t(z)\,\tilde{N}(dt,dz).$$

Hence, u admits the following BSDEj representation:

$$\begin{cases} dY_t = rY_t\,dt + Z_t^\top dW_t + \displaystyle\int_{\mathbb{R}^d} U_t(z)\,\tilde{N}(dt,dz), & t \in [t_m, t_{m+1}), \\ Y_{t_{m+1}} = V_{t_{m+1}}(X_{t_{m+1}}), \end{cases} \tag{2.15}$$

or equivalently,

$$Y_t = V_{t_{m+1}}(X_{t_{m+1}}) - \int_t^{t_{m+1}} rY_s\, ds - \int_t^{t_{m+1}} Z_s^\top dW_s - \int_t^{t_{m+1}} \int_{\mathbb{R}^d} U_s(z)\, \tilde{N}(ds, dz), \qquad t \in [t_m, t_{m+1}). \tag{2.16}$$

Together with (2.3), this forms a decoupled FBSDE, at each period $[t_m, t_{m+1})$. Therefore, the Bermudan option pricing problem can be formulated as a backward recursion over a sequence of FBSDEs with max-update $V_t = \max\{g(X_t), Y_t\}$ at each exercisable dates in $\mathcal{T}$. Specifically, in each interval $[t_m, t_{m+1})$, solve the local FBSDE with terminal condition $Y_{t_{m+1}} = V_{t_{m+1}}(X_{t_{m+1}})$ to obtain the continuation value $C_{t_m}(X_{t_m})$, and then compare it with the immediate exercise payoff $g(X_{t_m})$ to determine $V_{t_m}(X_{t_m})$.

3 DBDP-Jump scheme

This chapter develops the numerical method used to solve the Bermudan pricing problem formulated in Chapter 2. The main method, the DBDP-Jump scheme, is adapted to the MJD setting from the original DBDP1 scheme in [1]. DBDP1 solves the time-discretized BSDE recursively backward in time by approximating, at each Euler step, the option value and its gradient with neural networks (NNs) through the minimization of a local mean-squared one-step residual. To make this method capable of pricing Bermudan options under MJD, two main adaptations are needed. On the one hand, DBDP1 is formulated for a single time-discretized BSDE, whereas the Bermudan pricing problem has been reformulated as a backward recursion over a sequence of local BSDEj problems. This does not prevent the use of the DBDP idea, because the method is already built on backward time-stepping. Once the admissible exercise dates are included in the chosen time discretization grid, the Bermudan feature can be incorporated in a simple and consistent way: on each interval, DBDP1 is used to approximate the continuation value backward in time exactly as in the European case, and at every admissible exercise date, the option value is updated by taking the maximum of the resulting continuation estimate and the immediate exercise payoff. Hence, the DBDP idea can be used to solve the Bermudan option pricing problem by simply adding a max-projection step at admissible exercise dates. On the other hand, the DBDP1 scheme is designed to solve a specific class of nonlinear parabolic PDEs, whereas the MJD model introduces jumps and leads to PIDEs. Therefore, the additional jump terms in the associated BSDEj representation must be handled accordingly.

The DBDP-Jump scheme handles the jump-related terms in the BSDEj representation by approximating them with auxiliary offline Monte Carlo sampling and the right-endpoint value on each time interval, respectively. This means that neither of these two jump-related terms carries trainable neural-network parameters; only the value and gradient components are learned during training.

In this chapter, the first three sections build the main components of the method: the time-discrete local BSDEj recursion, the artificial neural-network notation used for the value and gradient approximations, and the treatment of the jump-related terms. These components are then assembled into the complete baseline DBDP-Jump scheme.

3.1 From the FBSDE formulation to discrete time

As derived in Section 2.3, in each period $[t_m, t_{m+1})$, the local FBSDE with jump terms is given by (2.3) and (2.15). The DBDP-Jump scheme handles the FBSDE with jumps. For readability, the explicit solution of the forward SDE (2.5) in vector form and the local BSDEj (2.16) are presented again.

Vector form of the explicit forward solution. For the numerical scheme, it is more convenient to rewrite the componentwise explicit solution (2.5) in vector form. Recall that $W_t = (W_t^1, \dots, W_t^d)^\top$ is a d-dimensional Brownian motion with correlation matrix $\boldsymbol{\rho}_W$. Then the explicit solution of the forward SDE (2.3) in vector form is

$$X_t = X_0 \odot \exp\Big(\big(r\mathbf{1} - \tfrac{1}{2}\boldsymbol{\sigma}^{\odot 2} - \lambda\boldsymbol{\kappa}\big)\, t + \boldsymbol{\sigma} \odot W_t\Big) \odot \prod_{k=1}^{N_t} e^{Z_k}, \tag{3.1}$$

where $\boldsymbol{\sigma} = (\sigma_1, \dots, \sigma_d)^\top$, $\boldsymbol{\sigma}^{\odot 2} = (\sigma_1^2, \dots, \sigma_d^2)^\top$, and $\boldsymbol{\kappa} = (\kappa^1, \dots, \kappa^d)^\top$. Note that the correlation structure enters through W_t itself, i.e. $\mathbb{E}[W_t^i W_t^j] = \rho_{ij}\, t$, and the Itô correction $\frac{1}{2}\boldsymbol{\sigma}^{\odot 2}$ remains valid since $\rho_{ii} = 1$.

Local BSDEj for the continuation value. For each time interval $[t_m, t_{m+1}]$ determined by $\mathcal{T}$, assume that the value function at the right endpoint, $V_{t_{m+1}}$, has already been determined by backward induction. Then the continuation value on $[t_m, t_{m+1}]$ is characterized by the local BSDEj

$$\begin{cases} Y_t = V^\pi_{t_{m+1}}(X_{t_{m+1}}) - \int_t^{t_{m+1}} rY_s\, ds - \int_t^{t_{m+1}} Z_s^\top\, dW_s - \int_t^{t_{m+1}} \int_{\mathbb{R}^d} U_s(z)\, \tilde{N}(ds, dz), & t \in [t_m, t_{m+1}], \\ Z_t = \sigma_X(X_{t-})\nabla_x u(t, X_{t-}), \qquad U_t(z) = u(t, X_{t-} \odot e^z) - u(t, X_{t-}). \end{cases} \tag{3.2}$$

The relation between the continuation value process Y_t and function $u(t,x)$ is given by $Y_t = u(t, X_t)$.

Time discretization. Let $\pi = \{0 = t_0 < t_1 < \cdots < t_N = T\}$, be a time grid such that all Bermudan exercise dates are included in it. For simplicity, write $\Delta t_n := t_{n+1} - t_n$, $\Delta W_n := W_{t_{n+1}} - W_{t_n}$, $X_n := X_{t_n}$, and $(Y_n, Z_n, U_n(\cdot)) := (Y_{t_n}, Z_{t_n}, U_{t_n}(\cdot))$.

Since the multidimensional MJD admits an explicit solution (3.1), the forward process is simulated on each interval $[t_n, t_{n+1}]$ by its exact one-step transition rather than by a generic Euler approximation:

$$X_{n+1} = X_n \odot \exp\left(\left(r\mathbf{1} - \frac{1}{2}\,\boldsymbol{\sigma}^{\odot 2} - \lambda\boldsymbol{\kappa}\right)\Delta t_n + \boldsymbol{\sigma} \odot \Delta W_n\right) \odot \prod_{k=1}^{\Delta N_n} e^{Z_k}, \tag{3.3}$$

where $\Delta N_n := N_{t_{n+1}} - N_{t_n}$ denotes the number of jumps on $(t_n, t_{n+1}]$, i.e. $\Delta N_n \sim \mathrm{Poisson}(\lambda \Delta t_n)$. If no jump occurs, then $\Delta N_n = 0$ and the product is interpreted as 1.

Since all admissible exercise dates are included in the grid, each interval $[t_n, t_{n+1}]$ can be treated as one local BSDEj step. The value at t_{n+1} is fixed before solving backward to t_n. Applying Euler discretization to (3.2) on this interval gives the one-step relation

$$Y^\pi_{n+1}(X_{n+1}) \approx Y^\pi_n(X_n) + rY^\pi_n(X_n)\,\Delta t_n + Z^\pi_n(X_n)^\top \Delta W_n + J^\pi_n - C^\pi_n\,\Delta t_n. \tag{3.4}$$

The superscript π indicates the Euler-discretized approximation of the corresponding continuous-time term. The last two jump terms are obtained by decomposing the compensated jump

integral via $\tilde{N}(dt, dz)$:

$$\int_{t_n}^{t_{n+1}} \int_{\mathbb{R}^d} U_s(z)\, \tilde{N}(ds, dz) = \underbrace{\int_{t_n}^{t_{n+1}} \int_{\mathbb{R}^d} U_s(z)\, N(ds, dz)}_{J_n} - \underbrace{\lambda \int_{t_n}^{t_{n+1}} \int_{\mathbb{R}^d} U_s(z) f_Z(z)\, dz\, ds}_{C_n \Delta t_n}. \quad (3.5)$$

J_n is the realized-jump contribution, whereas $C_n \Delta t_n$ is the compensator contribution over the interval. When discretized on the grid π, these quantities are approximated by

$$J_n^\pi := U_{n+1}^\pi \left(\sum_{k=1}^{\Delta N_n} Z_k \right), \quad (3.6)$$

$$C_n^\pi := \lambda \int_{\mathbb{R}^d} U_{n+1}^\pi(z) f_Z(z)\, dz. \quad (3.7)$$

Here, $U_{n+1}^\pi(z)$ denotes a grid-based right-endpoint approximation of $U_t(z)$. As the forward paths (3.3) are simulated only at grid points, the intrainterval jump times and their left-limit states are not resolved. Therefore, the realized jump marks are aggregated as $\sum_{k=1}^{\Delta N_n} Z_k$, and U_{n+1}^π is evaluated at this cumulative jump mark using X_{n+1}^{pre} as its base state. X_{n+1}^{pre} is a proxy for the pre-jump state, which is obtained by evolving X_n from t_n to t_{n+1} using only the drift and Brownian terms:

$$X_{n+1}^{\mathrm{pre}} = X_n \odot \exp\left(\left(r1 - \frac{1}{2} \boldsymbol{\sigma}^{\odot 2} - \lambda \boldsymbol{\kappa} \right) \Delta t_n + \boldsymbol{\sigma} \odot \Delta W_n \right). \quad (3.8)$$

The right endpoint is used because U_{n+1}^π is available when the scheme proceeds backward to t_n. Similarly, C_n^π approximates the average compensator contribution per unit time C_n.

At each time grid t_n, the option value function is updated by

$$V_{t_n}(X_{t_n}) = \begin{cases} \max\{g(X_{t_n}),\, Y_{t_n}\}, & \text{if } t_n \in \mathcal{T}, \\ Y_{t_n}, & \text{otherwise}, \end{cases}$$

starting at maturity T with $V_T(X_T) = g(X_T)$.

After the Bermudan projection, the next-step term in (3.4) is represented by $V_{n+1}^\pi(X_{n+1})$. The Euler-discretized projected relation is therefore

$$V_{n+1}^\pi(X_{n+1}) \approx Y_n^\pi(X_n) + r Y_n^\pi(X_n)\, \Delta t_n + Z_n^\pi(X_n)^\top \Delta W_n + J_n^\pi - C_n^\pi\, \Delta t_n. \quad (3.9)$$

3.2 Neural-network approximation

Following the idea of DBDP1 [1], DBDP-Jump trains neural networks (NNs) at each discretized time step to approximate the unknown value function and its spatial gradient by minimizing a local backward loss on $[t_n, t_{n+1}]$. Before specifying this local optimization problem, some basic definitions and notation for feedforward artificial neural networks are recalled.

The idea of NNs can be traced back to early work on artificial neurons and the perceptron [28, 29]. Modern feedforward NNs extend this idea into a parametrized nonlinear approximation class built from repeated compositions of affine transformations and nonlinear

activation functions. This compositional form provides greater flexibility than fixed basis expansions. It is particularly useful in high dimensions, where classical bases such as polynomials, splines, or other hand-chosen families can be difficult to design and computationally expensive.

To be clear and precise, a feedforward neural network can be defined as the following composition:

$$f(x;\theta,\phi) = (A_L \circ \phi \circ A_{L-1} \circ \cdots \circ \phi \circ A_1)(x), \qquad f(\cdot;\theta,\phi) : \mathbb{R}^{m_0} \to \mathbb{R}^{m_L}. \tag{3.10}$$

Here $L+1 \in \mathbb{N} \setminus \{1,2\}$ denotes the total number of layers. The first layer is the input layer, the last layer is the output layer, and the $L-1$ intermediate layers are the hidden layers. The number of neurons in layer ℓ is denoted by m_ℓ, for $\ell = 0,\dots,L$. Hence m_0 is the input dimension and m_L is the output dimension. For the problem considered in this thesis, the input dimension is $m_0 = d$, where d is the number of underlying assets in the option. The output dimension m_L depends on the function to be approximated. Therefore, $m_L = 1$ for the value function, and $m_L = d$ for the gradient function. For simplicity, only fully connected NNs are considered. The hidden width is also set to be constant across layers, i.e. $m_1 = \cdots = m_{L-1} = m$.

For $\ell = 1,\dots,L$, A_ℓ is an affine transformation:

$$A_\ell(x) = W_\ell x + \beta_\ell, \text{ where } W_\ell \in \mathbb{R}^{m_\ell \times m_{\ell-1}}, \beta_\ell \in \mathbb{R}^{m_\ell}.$$

W_ℓ is the weight matrix, and β_ℓ is the bias vector. $\phi : \mathbb{R} \to \mathbb{R}$ $\phi : \mathbb{R} \to \mathbb{R}$ is an activation function applied componentwise. Namely, for $x = (x_1,\dots,x_m) \in \mathbb{R}^m$, $\phi(x) = (\phi(x_1),\dots,\phi(x_m))$. Typical activation functions include the sigmoid function, the hyperbolic tangent function, the rectified linear unit (ReLU), and the exponential linear unit (ELU).

$\theta = \{(W_\ell,\beta_\ell)\}_{\ell=1}^L$. is the collection of all trainable parameters in the network. $\theta \in \mathbb{R}^{N_m}$, where the total number of trainable parameters $N_m = \sum_{\ell=1}^L (m_\ell m_{\ell-1} + m_\ell)$. With the constant hidden width and d-dimensional input, $N_m = m(d+1) + (L-2)m(m+1) + m_L(m+1)$.

Now, define the (d, m_L) NN class by

$$\mathcal{NN}_{\phi,L,d,m_L} := \bigcup_{N_m \in \mathbb{N}} \mathcal{NN}_{\phi,L,d,m_L,N_m},$$

where $\mathcal{NN}_{\phi,L,d,m_L,N_m}$ denotes the set of networks of the form (3.10) with input dimension d, output dimension m_L, activation function ϕ, depth L, and $N_m = \sum_{\ell=1}^L (m_\ell m_{\ell-1} + m_\ell)$ trainable parameters. The capability of NNs as function approximators is justified by the following universal approximation theorem.

Theorem 3.1 (Universal approximation theorem (UAT) [30, Theorem 1])**.** *Let μ be a finite measure on $\mathbb{R}^d$, and let the activation function ϕ be bounded and nonconstant. For any $q \in \mathbb{N}$, the class $\mathcal{NN}_{\phi,L,d,m_L}$ is dense in $L^2(\mu;\mathbb{R}^q)$. Equivalently, for every $F \in L^2(\mu;\mathbb{R}^q)$ and every $\varepsilon > 0$, there exists $f(\cdot;\theta,\phi) \in \mathcal{NN}_{\phi,L,d,m_L}$ such that*

$$\|F - f(\cdot;\theta,\phi)\|_{L^2(\mu)} < \varepsilon.$$

Thus, under the $L^2(\mu)$ error criterion, the NN approximation error can be made arbitrarily small by increasing the network size. This supports the use of NNs to approximate the local value function and the Brownian-integrand component in DBDP-Jump.

Neural-network approximation and local loss. At each time step n, the DBDP-Jump scheme approximates the current continuation value and its gradient in terms of assets by NNs:

$$\widehat{Y}_n(x) \approx \mathcal{Y}_n(x;\theta_n^{\mathcal{Y}}), \qquad \widehat{\nabla_x u_n}(x) \approx \mathcal{Z}_n(x;\theta_n^{\mathcal{Z}}).$$

Using the one-step relation (3.9), define following map:

$$F_n(\theta_n) = \mathcal{Y}_n(X_n;\theta_n^{\mathcal{Y}}) + r\,\mathcal{Y}_n(X_n;\theta_n^{\mathcal{Y}})\Delta t_n + (\sigma_X(X_n)\mathcal{Z}_n(X_n;\theta_n^{\mathcal{Z}}))^\top \Delta W_n + \widehat{J}_n - \widehat{C}_n\Delta t_n, \quad (3.11)$$

where $\widehat{J}_n$ and $\widehat{C}_n$ are defined in (3.13) and (3.15), respectively. The map $F_n(\theta_n)$ is a function of the trainable parameters $\theta_n = (\theta_n^{\mathcal{Y}}, \theta_n^{\mathcal{Z}})$ in the NNs. The goal is to find the optimal parameters $\theta_n^\star$ such that $F_n(\theta_n^\star)$ approximates the learned next-step $\widehat{V}_{n+1}(X_{n+1})$ as closely as possible. The local loss is then defined by the mean-squared one-step residual

$$Loss_n(\theta_n) = \mathbb{E}\left[\left|\widehat{V}_{n+1}(X_{n+1}) - F_n(\theta_n)\right|^2\right]. \quad (3.12)$$

In practice, the expectation is approximated by Monte Carlo samples, and θ_n is updated with a stochastic optimizer (e.g., Adam [31]). Denote the fitted parameters by $\theta_n^\star = \arg\min_{\theta_n} Loss_n(\theta_n)$.

3.3 Approximation of jump-related terms

This section describes how the jump-related terms $\widehat{J}_n$ and $\widehat{C}_n$ in the one-step map (3.11) are computed. In contrast to [2], which uses an additional NN to approximate the jump integrand directly, DBDP-Jump separates the jump correction into the realized-jump contribution and the compensator contribution. These two terms are evaluated offline as follows.

Recall the discretized realized-jump term in (3.6), the compensator in (3.7), and the jump-integrand definition in (2.14). When solving the local problem at time step n, the approximation of value function at t_{n+1} is already trained via NNs. Hence, the realized-jump approximation is given by

$$\widehat{J}_n := \widehat{V}_{n+1}\left(X_{n+1}\right) - \widehat{V}_{n+1}\left(X_{n+1}^{\mathrm{pre}}\right), \quad (3.13)$$

where X_{n+1}^{pre} is the pre-jump endpoint state defined in (3.8). For the compensator,

$$C_n^\pi = \lambda \int_{\mathbb{R}^d} \left[V_{n+1}^\pi\left(X_{n+1}^{\mathrm{pre}} \odot e^z\right) - V_{n+1}^\pi\left(X_{n+1}^{\mathrm{pre}}\right)\right] f_Z(z)\,dz, \quad (3.14)$$

the integration over the jump distribution is approximated by auxiliary Monte Carlo sampling: sample M_J auxiliary jump sizes $Z^{(j)} \sim \mathcal{N}_d(\mu_J, \Sigma_J)$, $j = 1, \ldots, M_J$. Its approximation is then given by

$$\widehat{C}_n := \lambda \cdot \frac{1}{M_J} \sum_{j=1}^{M_J} \left[\widehat{V}_{n+1}\left(X_{n+1}^{\mathrm{pre}} \odot e^{Z^{(j)}}\right) - \widehat{V}_{n+1}\left(X_{n+1}^{\mathrm{pre}}\right)\right]. \quad (3.15)$$

3.4 Baseline DBDP-Jump algorithm

The preceding sections have introduced the main components of the DBDP-Jump scheme: the time-discretized local BSDEj, the neural-network approximation, and the treatment of jump-related terms. For readability, these components are now assembled into a structured workflow for the baseline algorithm. A detailed pseudocode is given in Appendix A.

Structured backward workflow. Starting from the terminal condition

$$\widehat{V}_N(x) = g(x),$$

DBDP-Jump proceeds backward for $n = N-1, \dots, 0$:

1. On the interval $[t_n, t_{n+1}]$, use the simulated samples X_n, X_{n+1}, X_{n+1}^{pre}, ΔW_n, and the realized jump information.
2. Represent the local value and gradient components by NNs:

$$\widehat{Y}_n(x) \approx \mathcal{Y}_n(x; \theta_n^{\mathcal{Y}}), \qquad \widehat{\nabla_x u_n}(x) \approx \mathcal{Z}_n(x; \theta_n^{\mathcal{Z}}).$$

3. Compute the realized-jump and compensator approximations $\widehat{J}_n$ and $\widehat{C}_n$ from (3.13) and (3.15). Since both terms depend only on $\widehat{V}_{n+1}$, they are fixed during the optimization at time step n.
4. Train the local networks by solving

$$\theta_n^\star = \left(\theta_n^{\mathcal{Y},\star}, \theta_n^{\mathcal{Z},\star}\right) \in \arg\min_{\theta_n} Loss_n(\theta_n),$$

   where the loss is defined in (3.12). Then set $\widehat{Y}_n(x) = \mathcal{Y}_n(x; \theta_n^{\mathcal{Y},\star})$.
5. Update the value approximation by the Bermudan projection

$$\widehat{V}_n(x) = \begin{cases} \max\{g(x), \widehat{Y}_n(x)\}, & \text{if } t_n \in \mathcal{T}, \\ \widehat{Y}_n(x), & \text{otherwise.} \end{cases}$$

This defines the recursion $\widehat{V}_{n+1} \mapsto \widehat{V}_n$ and is repeated until the initial estimator $\widehat{V}_0(x_0)$ is obtained.

4 Convergence analysis

This chapter studies the consistency of the proposed DBDP-Jump scheme for Bermudan option pricing under the multidimensional MJD model. The analysis is organized by error source. First, time discretization error of the BSDEj (2.16) is considered. Then, the neural-network approximation layer is analyzed by adapting the original DBDP argument to the present MJD setting. Finally, these components are combined into a consistency estimate.

4.1 Model assumptions

This section fixes the assumptions used throughout the convergence discussion. It establishes that finite-moment MJD dynamics, an L^2 payoff, and a Lipschitz linear driver together yield a well-posed backward recursion for the Bermudan value process.

First, the MJD model has constant drift, volatility, finite jump intensity, and jump sizes with finite moments. Throughout the convergence analysis, assume that the Brownian correlation matrix $\boldsymbol{\rho}_W$ is positive definite. Then, assume that the payoff $g(\cdot)$ is square-integrable along the forward process at the exercise dates, i.e., $g(X_{t_m}) \in L^2$, for all $t_m \in \mathcal{T}$. Here, L^2 means that the second moment is finite.

The Bermudan problem is solved backward in time. At maturity, the terminal value is $V_T(X_T) = g(X_T)$, which is square-integrable by assumption. Suppose that $V_{t_{m+1}}(X_{t_{m+1}}) \in L^2$ on the interval $[t_m, t_{m+1}]$, this value is used as the terminal condition of the local continuation BSDEj. In the target pricing problem, the driver $f(t, x, y, z, u) = -ry$ is linear in y and independent of t, x, z, and u. Hence, it satisfies the Lipschitz condition

$$|f(t, x, y, z, u) - f(t, x, y', z', u')| = r|y - y'|.$$

Moreover, $f(t, 0, 0, 0, 0) = 0$, so the square-integrability condition on the driver at the origin is trivially satisfied.

By the standard well-posedness result for a BSDEj with a Lipschitz generator and square-integrable terminal data [32], the local BSDEj admits a unique adapted solution

$$(Y, Z, U) \in \mathcal{S}^2 \times \mathcal{H}^2 \times \mathcal{H}^2_\nu.$$

Here, the spaces are defined as follows:

- $\mathcal{S}^2$ denotes the space of adapted càdlàg processes Y satisfying

$$\mathbb{E}\left[\sup_{0 \le t \le T} |Y_t|^2\right] < \infty.$$

- $\mathcal{H}^2$ denotes the space of predictable Brownian integrands Z satisfying

$$\mathbb{E}\int_0^T \|Z_t\|^2 dt < \infty.$$

- $\mathcal{H}_\nu^2$ denotes the space of predictable jump integrands U satisfying

$$\mathbb{E}\int_0^T \int_{\mathbb{R}^d} |U_t(z)|^2 \nu(dz)dt < \infty.$$

These integrability properties ensure that the mean-square error quantities used in the discretization analysis are well-defined.

In particular, $Y_{t_m} \in L^2$. The continuation value at the exercise date is $C_{t_m}(X_{t_m}) = Y_{t_m}$. Therefore, the Bermudan projection

$$V_{t_m}(X_{t_m}) = \max\{g(X_{t_m}), C_{t_m}(X_{t_m})\}$$

also remains square-integrable. Indeed,$|\max(a,b)|^2 \le 2|a|^2 + 2|b|^2$.

Since both $g(X_{t_m})$ and $C_{t_m}(X_{t_m})$ belong to L^2, it follows that $V_{t_m}(X_{t_m}) \in L^2$. This closes the backward induction and shows that every local terminal condition used in the Bermudan recursion is square-integrable.

Remark. For the arithmetic basket put payoff, $g(x) = (K - \frac{1}{d}\sum_{j=1}^d x_j)^+$, $0 \le g(x) \le K$ *for non-negative stock prices. Hence* $g(X_{t_m}) \in L^2$ *at all exercise dates. Moreover, arithmetic basket put payoffs are Lipschitz continuous in the state variable because for any* $x, x' \in \mathbb{R}^d$,

$$|g(x) - g(x')| \le \frac{1}{d}\sum_{j=1}^d |x_j - x_j'|.$$

4.2 Time discretization error

For the target problem considered in this thesis, the MJD process (2.5) admits an explicit transition representation at the prescribed time-grid points. The forward process is therefore simulated exactly on the grid, so no Euler discretization error is introduced.

Recall from Chapter 3 that the time grid $\pi = \{0 = t_0 < t_1 < \cdots < t_N = T\}$ contains all Bermudan exercise dates and that $\Delta t_n = t_{n+1} - t_n$. Denote the mesh size by $|\pi| = \max_{0\le n\le N-1} \Delta t_n$. The local BSDEj and its Euler one-step relation have been given in (3.2) and (3.4). The discrete jump terms J_n^π and C_n^π are defined in (3.6) and (3.7). At this level, these terms are interpreted using the exact next-step value function $V_{t_{n+1}}$, before any neural-network approximation error is introduced. The exact continuous-time counterparts, J_n and $C_n\Delta t_n$, are defined in (3.5).

The discretization error analysis of the local BSDEj follows the one in [33]. Their assumptions include a finite jump measure, Lipschitz forward coefficients, a Lipschitz terminal condition, a Lipschitz driver, and a jump coefficient $\beta(x,e)$ satisfying the boundedness and uniform Lipschitz condition in [33, Eq. (2.1)]. Their BSDEj is given in [33, Eq. (2.3)].

For MJD, the jump coefficient is $\gamma^X(x,z) = x \odot (e^z - 1)$. Since the log-jump size z is normally distributed, $e^z - 1$ is not uniformly bounded on the full jump space.

To use the regularity result of [33] directly, the discretization discussion is stated under a truncated-jump MJD setting. Let $E_R = \{z \in \mathbb{R}^d : \|z\| \le R\}$ be a bounded jump-mark space, and restrict the jump measure to E_R. On this truncated space, $\sup_{z\in E_R} |e^z - 1| < \infty$, forcing the jump coefficient to be uniformly Lipschitz in x. The additional condition H in [33] is also satisfied under this truncation, since $I + D_x\gamma^X(x,z) = \mathrm{diag}(e^z)$ is uniformly invertible on E_R.

Under this truncated-jump MJD setting, the following theorem applies.

Theorem 4.1 ([33, Theorem 2.1] in the notation of this thesis)**.** *There exists a constant $C_R > 0$, possibly depending on the truncation level R but independent of the time grid π, such that:*

(i) The value process Y satisfies

$$\epsilon^Y(\pi) := \max_{0\le n\le N-1} \mathbb{E}\left[\sup_{t\in[t_n,t_{n+1}]} |Y_t - Y_{t_n}|^2\right] \le C_R|\pi|. \tag{4.1}$$

(ii) The L^2-regularity of Z satisfies

$$\epsilon^Z(\pi) = \mathbb{E}\left[\sum_{n=0}^{N-1} \int_{t_n}^{t_{n+1}} \|Z_t - \bar{Z}_{t_n}\|^2\, dt\right] \le C_R|\pi|, \tag{4.2}$$

where $\bar{Z}_{t_n} := \frac{1}{\Delta t_n}\mathbb{E}_n\left[\int_{t_n}^{t_{n+1}} Z_t\, dt\right]$ is the conditional expectation.

Theorem 4.1(i) gives the L^2-time regularity of the value process along the forward path. For the jump terms, $U_s(z) = u(s, X_{s-}\odot e^z) - u(s, X_{s-}), s \in (t_n, t_{n+1}]$ is approximated by the right-endpoint discretization $U^\pi_{n+1}(z) = V^\pi_{n+1}(X^{\mathrm{pre}}_{n+1}\odot e^z) - V^\pi_{n+1}(X^{\mathrm{pre}}_{n+1})$.

Write the restricted jump measure as $\nu_R(dz) := 1_{E_R}(z)\lambda f_Z(z)\, dz < \infty$. Hereafter, C_R denotes a generic positive constant that may depend on R but is independent of the time grid π.

Under the truncated-jump MJD setting, the forward process is locally L^2-continuous, and the Markovian value function is Lipschitz in the state variable, uniformly over the grid times, and $1/2$-Hölder in time along the relevant states. Hence,

$$\mathbb{E}\int_{t_n}^{t_{n+1}}\int_{E_R} |U_s(z) - U^\pi_{n+1}(z)|^2 \nu_R(dz)\, ds \le C_R(\Delta t_n)^2.$$

Summing over n gives

$$\eta^U(\pi) := \mathbb{E}\left[\sum_{n=0}^{N-1}\int_{t_n}^{t_{n+1}}\int_{E_R} |U_s(z) - U^\pi_{n+1}(z)|^2 \nu_R(dz)\, ds\right] \le C_R|\pi|. \tag{4.3}$$

Therefore, $\eta^U(\pi) \to 0$ as $|\pi| \to 0$.

The corresponding discretization errors of the realized-jump and compensator contributions are

$$\epsilon^J(\pi) := \sum_{n=0}^{N-1} \mathbb{E}\left[\left|J_n - J^\pi_n\right|^2\right] \tag{4.4}$$

and

$$\epsilon^C(\pi) := \sum_{n=0}^{N-1} \mathbb{E}\left[\left|C_n \Delta t_n - C_n^\pi \Delta t_n\right|^2\right]. \tag{4.5}$$

For the realized-jump contribution, the approximation in (3.6) aggregates all realized jump marks on $(t_n, t_{n+1}]$ and evaluates U_{n+1}^π once from X_{n+1}^{pre}. This differs from the exact realized-jump contribution, which evaluates each jump increment at its actual jump time and corresponding left-limit state. To account for this difference, define the aggregation residual

$$A_n^\pi := \int_{t_n}^{t_{n+1}} \int_{E_R} U_{n+1}^\pi(z)\, N(ds, dz) - U_{n+1}^\pi\left(\sum_{k=1}^{\Delta N_n} Z_k\right). \tag{4.6}$$

Note that $A_n^\pi = 0$ when $\Delta N_n \le 1$. The realized-jump error admits the decomposition:

$$J_n - J_n^\pi = \int_{t_n}^{t_{n+1}} \int_{E_R} \left(U_s(z) - U_{n+1}^\pi(z)\right) N(ds, dz) + A_n^\pi. \tag{4.7}$$

The first term measures the error caused by replacing the continuous-time jump integrand with its grid-based right-endpoint approximation. The second term measures the additional error caused by aggregating all jump marks on the interval.

First consider the aggregation residual. By the definition of the Poisson integral,

$$A_n^\pi = \sum_{k=1}^{\Delta N_n} U_{n+1}^\pi(Z_k) - U_{n+1}^\pi\left(\sum_{k=1}^{\Delta N_n} Z_k\right).$$

The spatial Lipschitz continuity of the Markovian value function implies

$$\left|U_{n+1}^\pi(z)\right| \le C_R \left\|X_{n+1}^{\mathrm{pre}} \odot (e^z - 1)\right\|.$$

Fix $m \ge 0$ and consider the event $\{\Delta N_n = m\}$. Since $Z_k \in E_R$, the quantity $\|e^{Z_k} - 1\|$ is uniformly bounded. Therefore,

$$\left|U_{n+1}^\pi(Z_k)\right| \le C_R \left\|X_{n+1}^{\mathrm{pre}}\right\|, \qquad k = 1, \dots, m.$$

Moreover,

$$\left\|\sum_{k=1}^{m} Z_k\right\| \le mR, \qquad \left\|e^{\sum_{k=1}^m Z_k} - 1\right\| \le C_R e^{mR}.$$

Hence,

$$\left|U_{n+1}^\pi\left(\sum_{k=1}^{m} Z_k\right)\right| \le C_R e^{mR} \left\|X_{n+1}^{\mathrm{pre}}\right\|.$$

The triangle inequality now gives

$$\left|A_n^\pi\right| \le \sum_{k=1}^{m} \left|U_{n+1}^\pi(Z_k)\right| + \left|U_{n+1}^\pi\left(\sum_{k=1}^{m} Z_k\right)\right| \le C_R \left\|X_{n+1}^{\mathrm{pre}}\right\| \left(m + e^{mR}\right) \le C_R \left\|X_{n+1}^{\mathrm{pre}}\right\| (m+1) e^{mR}.$$

Since $A_n^\pi = 0$ when $m \le 1$, squaring this residual on $\{\Delta N_n \ge 2\}$ yields

$$\left|A_n^\pi\right|^2 \le C_R \left\|X_{n+1}^{\mathrm{pre}}\right\|^2 (\Delta N_n + 1)^2 \, e^{2R\Delta N_n} 1_{\{\Delta N_n \ge 2\}}. \tag{4.8}$$

Under the truncated model, ΔN_n is Poisson distributed with mean $q_n := \nu_R(E_R)\Delta t_n$. Hence,

$$\begin{aligned}
\mathbb{E}\left[(\Delta N_n + 1)^2 \, e^{2R\Delta N_n} 1_{\{\Delta N_n \ge 2\}}\right] &= e^{-q_n} \sum_{m=2}^{\infty} (m+1)^2 e^{2Rm} \frac{q_n^m}{m!} \\
&= q_n^2 e^{-q_n} \sum_{m=2}^{\infty} (m+1)^2 e^{2Rm} \frac{q_n^{m-2}}{m!} \qquad (4.9) \\
&\le C_R q_n^2 \le C_R (\Delta t_n)^2. \qquad (4.10)
\end{aligned}$$

The factor multiplying q_n^2 in (4.9) is uniformly bounded because $0 \le q_n \le \nu_R(E_R)T$. The finite-moment property of the MJD process gives

$$\sup_{n,\pi} \mathbb{E}\left[\left\|X_{n+1}^{\mathrm{pre}}\right\|^2\right] < \infty.$$

Combining (4.8) and (4.10), and using the independence of X_{n+1}^{pre} and ΔN_n, yields

$$\mathbb{E}\left[\left|A_n^\pi\right|^2\right] \le C_R \mathbb{E}\left[\left\|X_{n+1}^{\mathrm{pre}}\right\|^2\right] \mathbb{E}\left[(\Delta N_n + 1)^2 \, e^{2R\Delta N_n} 1_{\{\Delta N_n \ge 2\}}\right] \le C_R (\Delta t_n)^2. \tag{4.11}$$

The first decomposition term in (4.7) is controlled by the L^2-consistency of the jump integrand approximation, measured by $\eta^U(\pi)$. Since the jump measure has finite intensity on E_R, the L^2 estimate for Poisson integrals gives,

$$\mathbb{E}\left[\left|\int_{t_n}^{t_{n+1}} \int_{E_R} \left(U_s(z) - U_{n+1}^\pi(z)\right) N(ds, dz)\right|^2\right] \le C_R \mathbb{E}\left[\int_{t_n}^{t_{n+1}} \int_{E_R} \left|U_s(z) - U_{n+1}^\pi(z)\right|^2 \nu_R(dz)\, ds\right]. \tag{4.12}$$

Applying $|a+b|^2 \le 2|a|^2 + 2|b|^2$ to (4.7) and using (4.11) and (4.12), gives

$$\mathbb{E}\left[\left|J_n - J_n^\pi\right|^2\right] \le C_R \mathbb{E}\left[\int_{t_n}^{t_{n+1}} \int_{E_R} \left|U_s(z) - U_{n+1}^\pi(z)\right|^2 \nu_R(dz)\, ds\right] + C_R (\Delta t_n)^2.$$

Summing over n and utilizing (4.3) yields

$$\epsilon^J(\pi) \le C_R \eta^U(\pi) + C_R \sum_{n=0}^{N-1} (\Delta t_n)^2 \le C_R |\pi|. \tag{4.13}$$

Therefore, $\epsilon^J(\pi)$ converges to zero as the time grid is refined.

The compensator error is governed by $\eta^U(\pi)$. On the truncated jump space,

$$C_n = \frac{1}{\Delta t_n} \int_{t_n}^{t_{n+1}} \int_{E_R} U_s(z) \nu_R(dz)\, ds, \qquad C_n^\pi = \int_{E_R} U_{n+1}^\pi(z) \nu_R(dz).$$

Hence,

$$C_n \Delta t_n - C_n^\pi \Delta t_n = \int_{t_n}^{t_{n+1}} \int_{E_R} \left(U_s(z) - U_{n+1}^\pi(z)\right) \nu_R(dz)\, ds.$$

By the Cauchy-Schwarz inequality and the finiteness of $\nu_R(E_R)$,

$$\left| \int_{t_n}^{t_{n+1}} \int_{E_R} \left(U_s(z) - U_{n+1}^{\pi}(z) \right) \nu_R(dz)\, ds \right|^2 \leq \Delta t_n\, \nu_R(E_R) \int_{t_n}^{t_{n+1}} \int_{E_R} \left| U_s(z) - U_{n+1}^{\pi}(z) \right|^2 \nu_R(dz)\, ds.$$

Taking expectations and summing over n gives $\epsilon^C(\pi) \leq C_R |\pi| \eta^U(\pi)$, so $\epsilon^C(\pi)$ converges to zero when $|\pi|$ goes to zero.

4.3 Neural-network approximation error

This section adapts the original DBDP neural-network approximation argument [1]. Before introducing the NN approximation error, the one-step regression target is first rewritten from the discretized BSDEj relation:

$$V_{n+1}^{\pi}(X_{n+1}) \approx Y_n + r Y_n \Delta t_n + Z_n^{\top} \Delta W_n + J_n^{\pi} - C_n^{\pi} \Delta t_n.$$

Here J_n^{π} denotes the discrete realized-jump term, and C_n^{π} denotes the discrete compensator term. They are approximated using the next-step learned value function. Therefore, group $V_{n+1}^{\pi}(X_{n+1})$ and the jump-related terms on the same side and define the jump-adjusted one-step target:

$$H_{n+1}^{\pi} := V_{n+1}^{\pi}(X_{n+1}) - J_n^{\pi} + C_n^{\pi} \Delta t_n.$$

Then

$$H_{n+1}^{\pi} \approx Y_n + r Y_n \Delta t_n + Z_n^{\top} \Delta W_n. \tag{4.14}$$

Taking conditional expectation with respect to $\mathcal{F}_{t_n}$ gives

$$\mathbb{E}_n\left[H_{n+1}^{\pi} \right] \approx Y_n + r Y_n \Delta t_n. \tag{4.15}$$

This motivates the ideal discrete continuation target: $Y_n^{\pi}(X_n) = \frac{\mathbb{E}_n\left[H_{n+1}^{\pi} \right]}{1 + r \Delta t_n}$.

For the Brownian-integrand target, multiplying (4.14) by ΔW_n, taking conditional expectation, and using the conditional covariance of the Brownian increment yields

$$\mathbb{E}_n\left[H_{n+1}^{\pi} \Delta W_n \right] \approx \mathbb{E}_n\left[(Z_n^{\top} \Delta W_n) \Delta W_n \right] = \boldsymbol{\rho}_W Z_n \Delta t_n.$$

Then, the ideal Brownian-integrand target is $Z_n^{\pi}(X_n) = \frac{1}{\Delta t_n} \boldsymbol{\rho}_W^{-1} \mathbb{E}_n\left[H_{n+1}^{\pi} \Delta W_n \right]$.

By the Markov property of the discretized Merton jump-diffusion process, the conditional expectations above depend only on the current state X_n. Hence Y_n^{π} and Z_n^{π} are deterministic target functions at time t_n. Recall $\mathcal{Y}_n(x; \theta_n^{\mathcal{Y}})$ denotes the local value network, and $\mathcal{Z}_n(x; \theta_n^{\mathcal{Z}})$ denotes the local gradient network. The corresponding Brownian-integrand approximation is $\sigma_X(x) \mathcal{Z}_n(x; \theta_n^{\mathcal{Z}})$. Then the local NN approximation errors are defined as

$$\varepsilon_n^{NN,Y} := \inf_{\theta_n^{\mathcal{Y}}} \mathbb{E}\left[\left| Y_n^{\pi}(X_n) - \mathcal{Y}_n(X_n; \theta_n^{\mathcal{Y}}) \right|^2 \right], \tag{4.16}$$

$$\varepsilon_n^{NN,Z} := \inf_{\theta_n^{\mathcal{Z}}} \mathbb{E}\left[\left\| Z_n^{\pi}(X_n) - \sigma_X(X_n) \mathcal{Z}_n(X_n; \theta_n^{\mathcal{Z}}) \right\|^2 \right]. \tag{4.17}$$

The universal approximation theorem [34] ensures that, given sufficient hidden neurons, $\varepsilon_n^{NN,Y}$ and $\varepsilon_n^{NN,Z}$ can be made arbitrarily small in L^2, provided the target functions Y_n^π and Z_n^π are square-integrable under the law of X_n.

Following the DBDP-Jump update, the jump-related terms J_n^π and C_n^π are evaluated with the next-step learned value function, and therefore inherit NN approximation error. When solving time step n, the continuation network $\mathcal{Y}_{n+1}(\cdot;\theta_{n+1}^{\mathcal{Y}})$ is already available. After the Bermudan projection, the learned value approximation is $\widehat{V}_{n+1}$. Define the next-step value error by

$$e_{n+1}(x) := V_{n+1}^\pi(x) - \widehat{V}_{n+1}(x).$$

Recall (3.13), the realized-jump term propagates the next-step value error through

$$J_n^\pi[V_{n+1}^\pi] - J_n^\pi[\widehat{V}_{n+1}] = e_{n+1}(X_{n+1}) - e_{n+1}(X_{n+1}^{\mathrm{pre}}).$$

The corresponding realized-jump NN propagation error is

$$\varepsilon_n^{J,NN} := \mathbb{E}\left[\left|J_n^\pi[V_{n+1}^\pi] - J_n^\pi[\widehat{V}_{n+1}]\right|^2\right]. \tag{4.18}$$

For the compensator, the next-step value error propagates through the integral over the jump space:

$$C_n^\pi[V_{n+1}^\pi] - C_n^\pi[\widehat{V}_{n+1}] = \int_{E_R} \left[e_{n+1}(X_{n+1}^{\mathrm{pre}} \odot e^z) - e_{n+1}(X_{n+1}^{\mathrm{pre}})\right] \nu_R(dz).$$

Since the one-step target contains $C_n^\pi \Delta t_n$, the compensator-related NN propagation error is defined as

$$\varepsilon_n^{C,NN} := (\Delta t_n)^2 \mathbb{E}\left[\left|C_n^\pi[V_{n+1}^\pi] - C_n^\pi[\widehat{V}_{n+1}]\right|^2\right]. \tag{4.19}$$

4.4 Consistency estimate for DBDP-Jump scheme

Sections 4.1 – 4.3 prepare the consistency estimate. They first establish well-posedness of the local BSDEj, then identify the time discretization errors and define the neural network and jump propagation errors. The resulting error sources can be summarized as follows. The forward MJD process is simulated exactly at the grid points and therefore contributes no Euler discretization error. The remaining time-discretization error comes from the backward BSDEj and is described by the regularity of the value process, the Brownian integrand, and the two jump contributions. The neural-network approximation error enters through the local value and gradient networks. Compared with the original DBDP setting, DBDP-Jump needs to evaluate jump-related terms with the learned next-step value function, which creates two additional propagation errors.

Before stating the theorem, define the squared error of the learned (Y, Z)-components by

$$\begin{aligned}\mathcal{E}_{MJD}\left[(\mathcal{Y}^N, \mathcal{Z}^N), (Y, Z)\right] := &\max_{0\le n\le N-1} \mathbb{E}\left[\left|Y_{t_n} - \mathcal{Y}_n(X_n;\theta_n^{\mathcal{Y}})\right|^2\right] \\ &+ \mathbb{E}\left[\sum_{n=0}^{N-1} \int_{t_n}^{t_{n+1}} \left\|Z_t - \sigma_X(X_n)\mathcal{Z}_n(X_n;\theta_n^{\mathcal{Z}})\right\|^2 dt\right].\end{aligned} \tag{4.20}$$

The next result bounds this metric in terms of the time-discretization errors, the neural-network approximation errors, and the jump-propagation errors introduced above.

Theorem 4.2 (Consistency of DBDP-Jump)**.** *Under the model assumptions and specifications in Section 4.1, there exists a constant $C > 0$, independent of the time grid π, such that*

$$\mathcal{E}_{MJD}\left[(\mathcal{Y}^N, \mathcal{Z}^N), (Y, Z)\right] \leq C\Bigg(\epsilon^Y(\pi) + \epsilon^Z(\pi) + \epsilon^J(\pi) + \epsilon^C(\pi) + \sum_{n=0}^{N-1}\left[N(\varepsilon_n^{NN,Y} + \varepsilon_n^{J,NN} + \varepsilon_n^{C,NN}) + \varepsilon_n^{NN,Z}\right]\Bigg). \tag{4.21}$$

Theorem 4.2 follows the consistency estimate of the original DBDP scheme [1], but with the additional MJD-specific jump terms. The terms $\epsilon^Y(\pi)$, $\epsilon^Z(\pi)$, $\epsilon^J(\pi)$, and $\epsilon^C(\pi)$ are the time-discretization errors of the local BSDEj defined in (4.1), (4.2), (4.4), and (4.5). The terms $\varepsilon_n^{NN,Y}$ and $\varepsilon_n^{NN,Z}$ are the local neural-network approximation errors for the value and Brownian-integrand components defined in (4.16) and (4.17). The additional terms $\varepsilon_n^{J,NN}$ and $\varepsilon_n^{C,NN}$ are the jump-related propagation errors defined in (4.18) and (4.19).

Remark. Under the truncated-jump MJD, Theorem 4.1 applies, and the estimates discussed in Section 4.2 show that the full time-discretization contribution is of order $|\pi|$. Therefore, Theorem 4.2 can reduce to

$$\mathcal{E}_{MJD}\left[(\mathcal{Y}^N, \mathcal{Z}^N), (Y, Z)\right] \leq C_R\Bigg(|\pi| + \sum_{n=0}^{N-1}\left[N(\varepsilon_n^{NN,Y} + \varepsilon_n^{J,NN} + \varepsilon_n^{C,NN}) + \varepsilon_n^{NN,Z}\right]\Bigg). \tag{4.22}$$

5 Improved Monte Carlo simulations for DBDP-Jump

5.1 Offline MC

For clarity, this section summarizes the two essential offline MC components used in the DBDP-Jump implementation. Here, offline is defined in an operational sense, relative to the NN optimization loop. It refers to MC samples and associated target quantities generated or evaluated before the relevant optimization and then held fixed throughout the corresponding training stage. The two components are (i) a large pool of simulated state trajectories and (ii) a fixed collection of auxiliary jump samples.

(i) **Offline MC path pool.**

A large pool of asset paths is simulated before the global backward NN training procedure to provide the state inputs and pathwise random quantities required in the local one-step BSDE loss (3.12). For every discretized interval, the pool stores, the asset states at the interval endpoints, the corresponding pre-jump states, Brownian increments, jump counts, and realized jump sizes. Since a single set of full trajectories covers all time intervals, these pathwise quantities can be reused throughout the backward recursion when training the sequence of local NNs.

In the actual training procedure, the pool is divided into disjoint training and validation subsets, and random mini-batches are drawn from the training subset for each gradient update. Compared with independently simulating new samples at every update, this construction avoids repeated path generation and is therefore more computationally efficient. The benefit can be particularly significant for models without an exact transition, where obtaining states at a given time level may require resimulating trajectories from the initial condition up to that level, even though only the quantities associated with the current local interval are used in the loss. In addition, the validation subset is disjoint with the training subset and remains fixed. So it can provide an independent measure of performance on trajectories not used for optimization and make the validation losses directly comparable across iterations of the same local training problem, without introducing additional MC fluctuations caused by repeated resampling.

(ii) **Offline auxiliary MC for the compensator approximation** (3.15)**.**

The compensator in (3.14) contains an expectation with respect to the jump-size distribution that must be evaluated at each state. This expectation is approximated by MC sampling. As with the offline MC path pool, a fixed collection of M_J auxiliary jump

sizes is sampled from $\mathcal{N}_d(\mu_J, \Sigma_J)$. These samples are generated before the global backward training procedure and retained across all time steps. Reusing them is appropriate for the MJD model considered here because the jump-size distribution is time-homogeneous and independent of the current state. The fixed sample collection therefore provides a common empirical approximation of the jump-size distribution throughout the sequence of local training problems.

The compensator approximation at time step n (3.15) also depends on the learned next-step value approximation $\widehat{V}_{n+1}$ and must therefore be evaluated sequentially within the backward recursion. Once $\widehat{V}_{n+1}$ is available, and before current local optimization begins, the compensator approximation is evaluated for all relevant states in the pool using the fixed auxiliary jump samples. The resulting quantities are stored, and each mini-batch retrieves the corresponding precomputed values during the subsequent local optimization.

This stepwise precomputation also relies on the offline MC path pool, which fixes the relevant states in advance and allows their compensator approximations to be evaluated and stored before local optimization. Reusing the fixed auxiliary jump samples further avoids repeated jump-size sampling and prevents additional resampling noise. Also note that the auxiliary MC samples help directly approximate the compensator term in the BSDE, in contrast to [2], which introduces an additional NN to approximate the jump integrand.

In summary, the offline path pool supplies the forward path samples, whereas the offline auxiliary MC set supplies the jump-distribution samples. Together, they separate MC generation and target construction from local NN optimization, which to some extent improves training efficiency.

5.2 Enhanced MC paths for estimating the early-exercise boundary

A potential limitation of the baseline DBDP-Jump training procedure arises from the distribution of the simulated states at early time steps. Since the forward paths start from a fixed initial state, the early-time samples may remain concentrated in a small region around that state. For an option with exercise opportunities at early times, the exercise-relevant region may lie outside this concentrated sample support. In that case, the NNs receive limited or even no training information in the region where the continuation value is critical, which may influence the accuracy of the estimated option value. An observation of poor exercise boundary estimation at an early stage is presented in Section 6.5.

5.2.1 Sample Enrichment

To address this potential issue, an additional sample-enrichment procedure is introduced. The key idea is to generate additional states for training at early time steps by rescaling the existing states in pool from a target interval, so that these rescaled samples can cover the possible exercise boundary region. This sample enrichment is only necessary at time steps for which the original sample pool does not sufficiently cover that region. The procedure consists of

three components: construction of the target interval for rescaling, selection of the enrichment steps, and integration of sample enrichment into NN training.

To make the rescaling compatible with high dimensions, a state variable $A(x) \in \mathbb{R}$ is introduced to map the asset vector x to a scalar representative value. A natural choice is the aggregation appearing in the payoff function. For a d-dimensional arithmetic basket option, $A(x) = \frac{1}{d}\sum_{i=1}^{d} x_i$. For each reference state X_n^{ref} from the original sample pool, let $A_n^{\mathrm{ref}} = A(X_n^{\mathrm{ref}})$ and let A_n^{tar} denote the corresponding target value in the target interval at time step n. The enriched state is obtained by rescaling the reference state to match the target value:

$$X_n^{\mathrm{enrich}} = \frac{A_n^{\mathrm{tar}}}{A_n^{\mathrm{ref}}} \cdot X_n^{\mathrm{ref}}.$$

Target interval construction. This construction is developed for put options whose payoff is determined by a positive scalar aggregation $A(x)$ through a monotone threshold. For Bermudan and American options, the exercise decision at time t_n depends on both the immediate payoff and the continuation value. Current basket levels with nontrivial future in-the-money (ITM) probabilities are more likely to be relevant to this comparison. The future ITM probability is therefore used as a proxy for locating the exercise-relevant region.

In principle, the next exercise date could serve as the future reference horizon for a Bermudan option. However, when it lies close to t_n, the resulting short-horizon basket distribution may be too narrow to yield an informative target interval. Maturity T is therefore used as a more stable reference horizon. For the put option with strike K, the terminal basket level A_T defines the terminal ITM event $A_T < K$.

To relate this terminal event to the current basket level at time t_n, define the remaining basket log-return as $R_n = \log\left(\frac{A_T}{A_n}\right)$. Since

$$\log\left(\frac{A_T}{K}\right) = R_n + \log\left(\frac{A_n}{K}\right),$$

the terminal ITM event can be written as

$$A_T < K \quad \Longleftrightarrow \quad R_n < -\log\left(\frac{A_n}{K}\right).$$

Thus, for any candidate current basket level $a > 0$, the conditional terminal ITM probability is

$$p_n(a) := \mathbb{P}(A_T < K \mid A_n = a) = \mathbb{P}\Big(R_n < -\log\left(\frac{a}{K}\right) \Big| A_n = a\Big).$$

Let $[p_{\mathrm{low}}, p_{\mathrm{high}}]$ denote the desired terminal ITM probability band. Conceptually, the target region contains current basket levels a satisfying $p_n(a) \in [p_{\mathrm{low}}, p_{\mathrm{high}}]$. The lower probability level limits the extension toward out-of-the-money states, while the upper level excludes states with extremely high terminal ITM probabilities.

In practice, the conditional distribution of R_n given $A_n = a$ is approximated by the empirical distribution constructed from the original simulated path pool. The corresponding remaining

log-return samples are

$$R_n^{(m)} = \log\left(\frac{A_T^{(m)}}{A_n^{(m)}}\right), \qquad m = 1, \dots, M,$$

where $A_n^{(m)} = A\left(X_n^{(m)}\right)$ and $A_T^{(m)} = A\left(X_T^{(m)}\right)$. Here, $X_n^{(m)}, X_T^{(m)} \in \mathbb{R}^d$ denote the states on the mth simulated path at t_n and T, respectively. Let $Q_{R,n}(p)$ denote the empirical p-quantile of $\{R_n^{(m)}\}_{m=1}^M$. Inverting the pooled approximation at p_{low} and p_{high} gives the target basket levels

$$A_{n,\text{high}} = K\exp\left(-Q_{R,n}(p_{\text{low}})\right), \qquad A_{n,\text{low}} = K\exp\left(-Q_{R,n}(p_{\text{high}})\right).$$

The resulting target interval at time t_n is given by

$$I_n^{\text{tar}} = \left[A_{n,\text{low}}, A_{n,\text{high}}\right] = \left[K\exp\left(-Q_{R,n}(p_{\text{high}})\right), K\exp\left(-Q_{R,n}(p_{\text{low}})\right)\right]. \tag{5.1}$$

In the implementation, the interval endpoints are further clipped to a prescribed safe range to avoid degenerate or excessively wide target regions.

Remarks.
(i) This target interval construction applies to put options whose payoff depends monotonically on a positive, positively homogeneous aggregation $A(x)$. Extending it to calls only requires reversing the ITM event, whereas path-dependent, nonmonotone, or nonpositive payoff drivers require a different construction.

(ii) In higher dimensions, the arithmetic basket level may become more concentrated because of cross-sectional averaging. In that case, the upper probability level p_{high} can be reduced, or the probability band can be tightened, if the default interval places too much emphasis on deep ITM states.

Selection of enrichment steps. For greater flexibility, the time steps at which sample enrichment is applied are again selected in a data-driven manner. The idea is to check whether a specified quantile range of the current samples, such as the 2nd to 98th percentile range, covers the target interval. If the empirical quantile range fully contains the target interval, the original sample pool is regarded as having sufficient support over the target region, and enrichment is not activated. Otherwise, enrichment is applied. Since the empirical quantile range of the representative values $A(X_n)$ generally widens over time, this check is performed forward in time and is determined before the NN training starts. The detailed selection logic is described as follows:

1. The original sample pool is stored on the time grid π. The selection check starts from t_1 and proceeds through $t_2, t_3, \dots$.
2. **Compute the empirical quantile range of the original basket samples.** For a prescribed quantile level $\varepsilon \in (0, 0.5)$, define the empirical quantile range of the original basket samples at time step n by $Q_n^\varepsilon = \left[q_n^\varepsilon, q_n^{1-\varepsilon}\right]$, where

$$q_n^\varepsilon = \text{Quantile}_\varepsilon\left(A_n^{(1)}, \dots, A_n^{(M)}\right), \qquad q_n^{1-\varepsilon} = \text{Quantile}_{1-\varepsilon}\left(A_n^{(1)}, \dots, A_n^{(M)}\right).$$

3. **The quantile-containment indicator.** For the target interval $I_n^{\mathrm{tar}} = \left[A_{n,\mathrm{low}}, A_{n,\mathrm{high}}\right]$ given in (5.1), the quantile-containment indicator is defined as

$$\mathcal{I}_n = 1\left\{q_n^{\varepsilon} \le A_{n,\mathrm{low}} \quad \text{and} \quad A_{n,\mathrm{high}} \le q_n^{1-\varepsilon}\right\}.$$

   Hence, $\mathcal{I}_n = 1 \iff I_n^{\mathrm{tar}} \subseteq Q_n^{\varepsilon}$, and $\mathcal{I}_n = 0$ means that at least part of the target interval lies outside the empirical quantile range of the original samples.

4. **Stop the forward scan once the target interval is covered.** Once the first time step n^* satisfying $\mathcal{I}_{n^*} = 1$ is found, the forward scan is stopped. Sample enrichment is applied only at the earlier checked steps where $\mathcal{I}_n = 0$. All later time steps $n \ge n^*$ are treated as non-enriched during the DBDP training.

Integration of sample enrichment into NN training. At each active enrichment time step n, enriched samples are generated as follows:

1. Draw a reference state X_n^{ref} from the original sample pool $\{X_n^{(m)}\}_{m=1}^{M}$ and compute its representative value $A_n^{\mathrm{ref}} = A(X_n^{\mathrm{ref}})$.

2. Sample a target representative value uniformly from the interval computed via (5.1):

$$A_n^{\mathrm{tar}} \sim U\left(A_{n,\mathrm{low}}, A_{n,\mathrm{high}}\right).$$

3. Rescale the reference state to match the sampled target value:

$$X_n^{\mathrm{enrich}} = \frac{A_n^{\mathrm{tar}}}{A_n^{\mathrm{ref}}} \cdot X_n^{\mathrm{ref}}.$$

   The common scaling factor preserves the relative composition of the reference state.

4. Starting from X_n^{enrich}, simulate the one-step transition under the MJD dynamics and compute the quantities required by the local NN loss at time step n.

For computational efficiency, the enriched sample pool is generated once at each selected time step and stored in memory. Its size is set as a specified fraction of the original pool size. The original and enriched samples are retained in separate pools and are not merged. At a selected step, the local loss is evaluated separately on the two pools and combined only at the objective level:

$$Loss_n(\theta_n) = (1-\alpha)Loss_n^{\mathrm{original}}(\theta_n) + \alpha Loss_n^{\mathrm{enriched}}(\theta_n), \qquad \alpha \in [0,1],$$

where $Loss_n^{\mathrm{original}}(\theta_n)$ and $Loss_n^{\mathrm{enriched}}(\theta_n)$ are sample-based approximations of (3.12) computed from the original and enriched pools, respectively. The weight α controls the contribution of the enriched samples.

The training stopping criterion is still evaluated under the validation data which follows the original model distribution. This means the loss used to determine training stopping is only based on the true samples. The enriched samples are only used to improve function learning near the possible exercise region.

5.2.2 Existing MC approaches

This subsection reviews existing sampling methods that may improve early-exercise boundary estimation and compares them conceptually with the proposed sample-enrichment procedure.

Sequential Monte Carlo (SMC) sampling. The lack of early-stage training samples in the exercise-relevant region can be viewed as a mismatch between the distribution of the simulated states and the region in which the continuation–exercise comparison must be learned. Importance sampling (IS) [35] addresses such a mismatch by sampling from a proposal distribution that assigns more probability mass to the region of interest. Likelihood-ratio weights then correct for the change of distribution. When the proposal and target distributions differ substantially, a single IS step may suffer weight degeneracy, with only a few samples carrying most of the total weight. SMC samplers [36] mitigate this problem by replacing the single distributional shift with a sequence of intermediate targets represented by weighted particles. Starting from an easily sampled distribution, the particles are moved progressively toward the final target, commonly through tempering. At each stage, the importance weights are updated for the next target. When weight degeneracy occurs, resampling replicates high-weight particles and removes those with negligible weights, while a subsequent Markov chain Monte Carlo (MCMC) mutation step restores diversity without changing the current target distribution. Repeating this cycle progressively concentrates the particle population in the target region. This framework has been applied in option pricing in [37].

However, applying SMC to the early exercise-boundary problem considered here is not direct. First, an SMC sampler requires a specified sequence of intermediate target distributions. Concentrating particles near the exercise boundary would require a measure of boundary relevance, such as the proximity between the immediate exercise payoff and the continuation value. Yet this region is itself unknown and is precisely what the additional samples are intended to estimate, making informative intermediate targets difficult to construct. Second, resampling only redistributes states already present in the particle population. If the early-time sample pool contains no states in the exercise-relevant region, resampling cannot fill the gap. New states could be generated through a proposal or MCMC mutation kernel, but this would require a boundary-informed target density, suitable mutation directions and scales, evaluation of density or acceptance ratios, and additional tuning. These requirements would add substantial complexity to the local DBDP-Jump training procedure.

Compared with the proposed procedure, which is a simpler rescaling-based method tailored to the local DBDP recursion, SMC offers limited additional benefit relative to its computational and implementation cost. It is therefore not pursued here.

Sequential design for optimal stopping. A more closely related approach is the sequential design proposed in [38]. The method defines the timing value at an exercise date as the difference between the continuation value and the immediate payoff: $T_n(x) = C_n(x) - g(x)$, so that the exercise boundary is characterized by the zero contour $T_n(x) = 0$. At each exercise step in the backward recursion, an initial set of states is used to simulate conditional future trajectories under the previously estimated future stopping policy. A surrogate for the timing value is fitted to the resulting observations, and candidate states are selected according to their proximity to the estimated boundary and the predictive uncertainty of the surrogate. New

conditional simulations are then started from the selected states, the surrogate is updated, and the fit–select–simulate–refit cycle is repeated until the local simulation budget is exhausted.

A key advantage of sequential design is that it targets the exercise boundary directly. However, adapting it to DBDP-Jump would turn each exercise step at which it is applied into an iterative local optimization procedure. The local NN would first need to be trained on the original sample pool to provide a preliminary continuation-value estimate. This estimate, together with a measure of predictive uncertainty, would then identify additional states near the estimated boundary. After the corresponding one-step MJD transitions and local targets are generated, the same NN would need to be updated and the selection procedure repeated. Several refinement rounds may therefore be required at the same time step.

A further difficulty is determining at which steps this sequential refinement should be activated. The insufficient coverage considered here occurs only at early time steps, and the affected range depends on the dimension, payoff, MJD parameters, and pool size. Applying sequential design at every exercise step would introduce redundant refinement where the original MC pool already provides adequate coverage. Restricting it to the affected steps would instead require either a preliminary baseline run or a separate coverage diagnostic, followed by retraining of the selected local networks.

Sequential design may locate the exercise boundary more precisely than the proposed sample-enrichment procedure. However, such precision is unnecessary when the objective is only to provide sufficient training coverage over an exercise-relevant region. The proposed procedure determines the activation range and target interval directly from the simulated path pool. It therefore avoids a separate preliminary continuation-value fit, repeated surrogate updates, candidate-state searches, and retraining rounds.

6 Numerical results

This chapter presents the numerical evaluation of DBDP-Jump. It first specifies the problem setup, implementation choices, and benchmark construction. Sanity checks are performed in 1D and 5D, followed by a grid-refinement study and a discussion of sample-enrichment efficiency. DBDP-Jump is then evaluated from 20D to 200D, focusing on pricing accuracy, computational scalability, sensitivity to MJD parameters, robustness across payoff functions, and learning-rate control in demanding cases. All computational experiments were performed in Python using the PyTorch framework on a computer equipped with an Apple M2 chip and 16 GB of RAM.

6.1 Problem setup

This section specifies the main problem setting used in the numerical experiments, including the stock dynamics, payoff, option types, and corresponding admissible exercise sets.

Unless stated otherwise, all stock paths in this chapter are simulated under the MJD dynamics with the following parameter settings:

- **Maturity:** $T = 1$,
- **Parameters:** $X_0 = 100, r = 0.1, \sigma = 0.2, \rho_{ij} = 0.5$,
- **Jump Parameters:** $\lambda = 0.5, \mu_J = -0.1, \sigma_J = 0.3, \rho_J^{ij} = 0$.

The main payoff considered is an arithmetic basket put with strike price $K = 100$:

$$g(x) = \max(K - \frac{1}{d}\sum_{i=1}^{d} x_i, 0).$$

The time to maturity is discretized into $N = 100$ steps, , and for different option types, the early exercise rights are defined by the parameter M as follows:

Table 6.1: Option types and corresponding admissible exercise steps used in the numerical experiments.

Option Type	M	Admissible Exercise Steps
European	0	100
Bermudan	3	25, 50, 75, 100
American	99	Every step (1 to 100)

Note that the American case in this table is a discretized approximation, not a continuously exercisable American option; it mimics the American limit by allowing exercise at every time step.

6.2 DBDP-Jump implementation

This section lists the implementation choices used in the numerical experiments. It specifies the neural-network architecture for the local value and gradient approximations, the offline sampling pool, the training and validation protocol, the auxiliary Monte Carlo approximation of the compensator, and the averaging convention for reported prices. These settings define the default implementation, with experiment-specific changes stated where they occur. The benchmark reference values are also specified.

Neural network architecture. A feedforward network with two hidden layers is used throughout the experiments. This follows [25, 1]. It is also supported by the empirical results in [13]. The architecture provides sufficient flexibility for the local one-step regression problems considered here while keeping the computational cost manageable. Unless otherwise stated, for a d-dimensional problem, the hidden width is set to $30 + d$ neurons per layer, as in [1]. The activation function is $\tanh$ for all hidden layers:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \qquad \tanh'(x) = 1 - \tanh^2(x).$$

$\tanh$ is smooth and bounded. For large $|x|$, however, it saturates: $\tanh(x)$ approaches ± 1, and its derivative becomes close to zero. Input normalization is therefore applied to keep the inputs in a stable range and improve training stability.

Sampling and training configuration. All experiments use an offline sample pool of 100,000 asset trajectories, generated on the time grid before the backward recursion.

After sampling, the pool is split into 80% training data and 20% validation data. Training is performed with mini-batches drawn from the training set, with the mini-batch size fixed at 1000 if not otherwise specified. The Adam optimizer is chosen to train the NNs. The learning rate and maximum number of iterations vary with the problem dimension and are specified in the corresponding numerical experiments. For a time discretization with 100 grid steps, the stopping threshold is fixed at 0.01. That is, local training stops if the loss value on the validation set falls below 0.01, which is checked every 100 iterations. For the compensator approximation (3.15), the default number of auxiliary jump sizes Z sampled from $\mathcal{N}_d(\mu_J, \Sigma_J)$ is 256. A larger auxiliary sample size reduces the Monte Carlo noise in the compensator approximation, but increases the cost of evaluating the local targets. Therefore, 256 is chosen to balance approximation accuracy and computational cost.

All low-dimensional and most moderate-dimensional problems use a prescribed three-stage learning-rate schedule at the terminal step and the learning rate is fixed at 10^{-4} at all remaining time steps. Cases with greater loss instability due to higher dimensionality or a larger payoff scale instead use an adaptive design based on validation progress. The motivation and behavior of the adaptive design are detailed in Subsection 6.6.2.

To reduce the effect of training and sampling randomness, the reported DBDP-Jump price is averaged over 5 trials.

Reference values. For high-dimensional European options, the reference value is computed by MC simulation using 100,000 paths if not otherwise specified. For 1D test cases, the reference value of European put is computed using the Merton semi-closed-form formula [26]. For the Bermudan and discretely monitored American-style puts, reference values are obtained using a jump-diffusion tree [19]. To ensure a like-for-like comparison, early exercise in the tree is restricted to the exercise dates used by DBDP.

6.3 Sanity check

DBDP-Jump is first evaluated on 1D and 5D problems. These tests aim to assess whether DBDP-Jump produces reasonable price estimates before it is applied to higher dimensions.

Because these low-dimensional BSDEs are comparatively simple, both sanity checks use 512 auxiliary jump samples to obtain a more precise compensator approximation and cleaner local training targets, thereby supporting more stable training at modest additional computational cost.

The 1D cases use the main setting in Section 6.1. As its loss space is comparatively simple, to avoid overfitting, the maximum number of training iterations is 5000, and the learning rate schedule at the terminal step is 10^{-2} for the first 1500 iterations, 10^{-3} for the next 1500 iterations, and 10^{-4} for the remaining. Table 6.2 reports the results for the 1D cases. The standard error (s.e.) is computed over the five independent trials. The relative error (rel. err.) is defined as the percentage difference between the trial mean and the reference value. The training time is averaged over trials and reported in seconds.

Table 6.2: DBDP-Jump sanity check results for 1D problems.

case (type, d)	ref. value	mean value	s.e.	rel. err.	time (s)
E, 1	6.5313	6.5105	0.0415	-0.32%	571
B, 1	7.3336	7.2936	0.0330	-0.55%	580
A, 1	7.5787	7.5900	0.0301	+0.15%	578

Available benchmark values on 1D problems give a way to quantify the accuracy of the DBDP-Jump estimates. As shown in Table 6.2, the DBDP-Jump estimates agree closely with these benchmarks, with absolute relative errors of at most 0.55%, demonstrating the method's effectiveness in 1D.

For the 5D Bermudan arithmetic put, both the problem setting and the reference value are taken from [22]. The problem parameters are:

$$
\begin{aligned}
&X_0^i = 100, \quad r = 0.05, \quad \sigma_i = 0.15, \quad \rho_{ij} = 0.3,\\
&\lambda = 0.5, \quad \boldsymbol{\mu}_J = (-0.3, -0.2, -0.1, 0.1, 0.2)^\top, \quad \sigma_J = 0.1, \quad \rho_J^{ij} = -0.2,\\
&K = 100, \quad M = 8.
\end{aligned}
$$

Here, $i, j = 1, \ldots, 5$, and the correlation parameters apply when $i \neq j$. Together with maturity, there are nine admissible exercise dates, so $N = 99$ is set to place all nine dates uniformly on the time grid. The maximum number of training iterations is 10,000, and the learning rate

schedule at the terminal step is 10^{-2} for the first 3000 iterations, 10^{-3} for the next 3000 iterations, and 10^{-4} for the remaining. Moreover, it is worth noting that the volatility and jump variation in this 5D problem are lower than those in the main setting. This produces less variable continuation targets and, consequently, smaller one-step residuals. Since the loss (3.12) is the mean squared one-step residual, a threshold of 0.01 frequently leads to premature stopping. Hence, it is lowered to 0.002 to force training and limit error propagation. The result is summarized in Table 6.3.

Table 6.3: DBDP-Jump sanity check result for the 5D Bermudan problem.

case (type, d)	ref. value	mean value	s.e.	rel. err.	time (s)
B, 5	2.576	2.5874	0.0057	+0.44%	842

Despite the different problem setting, the 5D Bermudan estimate remains close to the benchmark, with a relative error of 0.44%. Together with the 1D results, this shows that DBDP-Jump works well in low dimensions.

6.4 Grid refinement study

This section investigates the numerical behavior of DBDP-Jump under grid refinement for a 1D European put. The number of time steps is varied over $N \in \{10, 20, 40, 80, 160, 200, 320\}$. Besides the default jump intensity $\lambda = 0.5$, a higher intensity $\lambda = 2.0$ is considered because it increases the probability of multiple jumps within an interval, making the additional aggregation error in the realized-jump approximation more relevant. To isolate the influence of the time-step size, all remaining settings follow the 1D sanity-check configuration, except for the validation-loss stopping threshold. The threshold is fixed at 0.01 at the terminal regression step, where the NN parameters are randomly initialized, and set to $\frac{1}{N}$ at subsequent steps, where the NN are warm-started from the parameters learned at the next time step, to maintain sufficient training under grid refinement. For each (λ, N) pair, five independent trials are performed. Each trial is assigned one sample-pool seed and one NN training seed. The same five seed pairs are reused for all values of N and both values of λ, reducing the influence of arbitrary seed changes on comparisons across the different settings.

The results are summarized in Table 6.4. The abbreviations abs. err. and 95% CI h.w. denote the absolute error and the half-width of the 95% confidence interval, respectively. First, the average training time increases approximately in proportion to the number of time steps. This is expected because DBDP-Jump solves one local NN regression problem at each backward time step. More importantly, for both jump intensities, the pricing error decreases as N increases from 10 to 80, where it reaches its minimum, but does not continue to decrease under further grid refinement. Thus, the practical error does not exhibit a monotonic convergence pattern. As shown in Theorem 4.2, the error bound contains the time-discretization error, the local NN approximation errors, and the jump-propagation errors. While the time-discretization error decreases under grid refinement, the latter errors accumulate over the backward time steps. The UAT implies that the local approximation errors can, in principle, be made arbitrarily small with sufficiently expressive and well-trained NNs, allowing the overall scheme to converge as the time grid is refined. In practice, however, these errors cannot be eliminated

Table 6.4: Grid-refinement results for the 1D European put.

N	mean value	abs. err.	rel. err.	s.e.	95% CI h.w.	time (s)
		$\lambda = 0.5$	(ref. value = 6.5313)			
10	6.6070	0.0758	+1.16%	0.0373	0.1036	86
20	6.5950	0.0637	+0.98%	0.0441	0.1223	135
40	6.4833	0.0479	-0.73%	0.0393	0.1090	243
80	6.5179	**0.0133**	**-0.20%**	0.0516	0.1432	464
160	6.5487	0.0174	+0.27%	0.0385	0.1068	906
200	6.5780	0.0468	+0.72%	0.0365	0.1013	1128
320	6.4905	0.0408	-0.63%	0.0310	0.0862	1831
		$\lambda = 2.0$	(ref. value = 12.8337)			
10	13.0987	0.2650	+2.07%	0.1194	0.3314	80
20	13.0467	0.2131	+1.66%	0.1191	0.3307	130
40	12.9529	0.1192	+0.93%	0.1126	0.3127	242
80	12.9271	**0.0934**	**+0.73%**	0.1193	0.3311	479
160	12.9613	0.1277	+1.00%	0.1126	0.3127	940
200	12.9887	0.1550	+1.21%	0.1087	0.3019	1164
320	13.0140	0.1803	+1.41%	0.1027	0.2850	1832

completely. As the number of time steps increases, their accumulation can eventually offset the reduction in the time-discretization error, causing the overall error to cease decreasing.

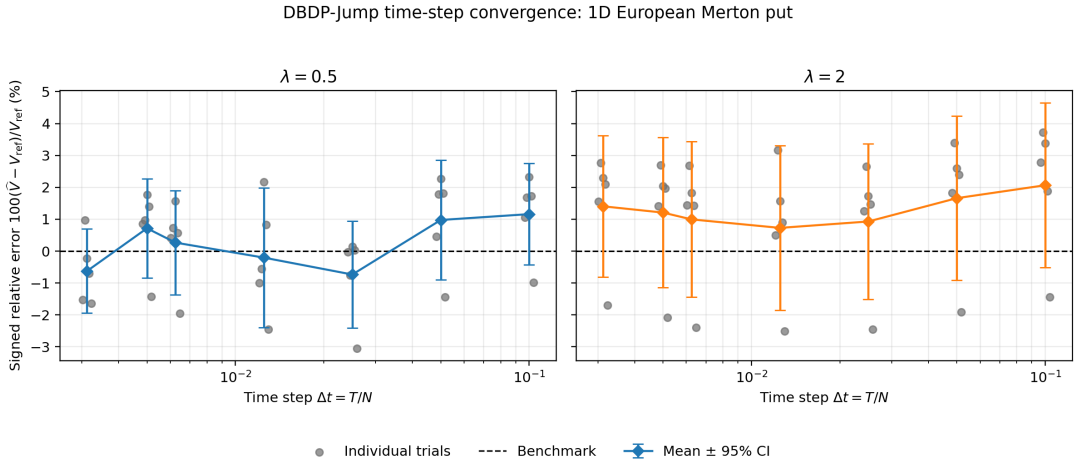
To make the practical grid-refinement pattern more visible, the time-step size is shown on a logarithmic horizontal axis in both panels. Figure 6.1a presents the signed relative errors. The grey points represent the five individual trials, the diamonds represent their mean signed relative error, and the error bars show the 95% confidence intervals. The dashed horizontal line indicates zero relative error, corresponding to the reference price. All confidence intervals intersect this line, indicating that no statistically significant pricing bias is identified from the five trials. Figure 6.1b shows the absolute relative bias using logarithmic scales on both axes. For each value of λ, the estimation error initially decreases as the grid is refined, indicating improved pricing accuracy, but does not continue to decrease once the grid becomes finer than a certain level. A comparison between the two jump-intensity settings shows that the errors for $\lambda = 2.0$ are generally larger than those for $\lambda = 0.5$. This difference is plausibly related to the increased probability of multiple jumps within one time interval, which introduces an additional aggregation residual into the realized-jump approximation.

6.5 Efficiency of sample enrichment

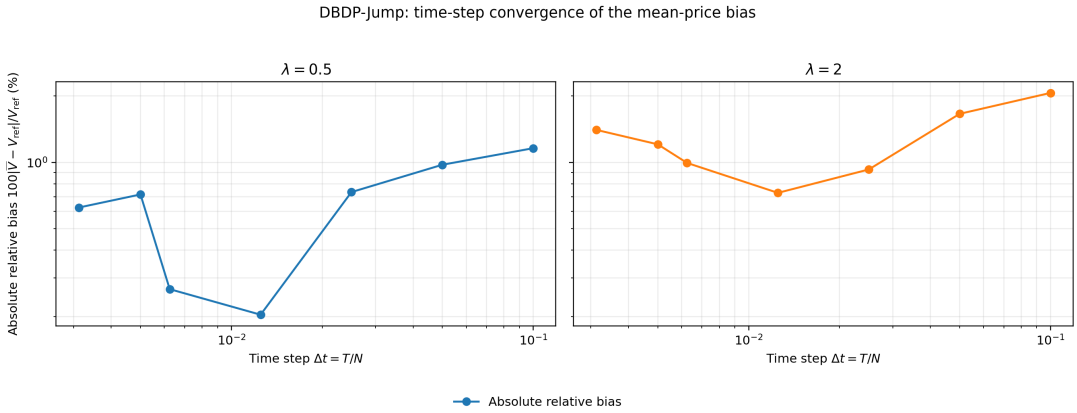
This section reports the results of the sample-enrichment modification proposed in Subsection 5.2.1. It examines whether adding training samples in the exercise-relevant region reduces the pronounced bias in the estimated early-time exercise boundary. It further assesses whether improved boundary estimation leads to more accurate option price.

To enable direct comparison of the estimated exercise boundaries, the modification is first tested on the 1D American-style option under the main setting, for which the tree benchmark provides an exercise boundary at every exercise step. It is then tested on a 5D geometric

basket American-style put under the main MJD parameters. Because the geometric basket state can be reduced to an effective 1D MJD process, its exercise boundary can likewise be computed. Given their low dimensionality and comparable benchmark price scales, both cases use the same training configuration as the preceding 1D sanity check. They also share the same sample-enrichment parameters. Specifically, $p_{\text{low}} = 0.50$ and $p_{\text{high}} = 0.85$, so the target interval contains current basket levels with moderate to high empirical probabilities of being in the money at maturity. For the geometric basket, the representative aggregation is $A(x) = \left(\prod_{i=1}^{d} x_i\right)^{1/d}$. The quantile level $\varepsilon = 0.02$ defines the empirical 2nd–98th percentile range used by the containment rule to select the enrichment steps. The enriched pool contains 10,000 samples, equal to 10% of the original pool, and its loss contribution weight $\alpha = 0.3$.



(a) Relative error with 95% confidence intervals.



(b) Absolute relative bias.

Figure 6.1: Grid-refinement results for the 1D European put.

Table 6.5 focuses on the accuracy of the early-time exercise boundary comparison in both cases. The active step range gives the earliest and latest time-step indices at which the automatic containment rule activated enrichment across the trials. At each exercise date t_n,

the DBDP boundary is first averaged over the $R = 5$ independent trials:

$$\overline{B}^*_{\mathrm{DBDP}}(t_n) = \frac{1}{R}\sum_{r=1}^{R} \widehat{B}^{*,(r)}_{\mathrm{DBDP}}(t_n).$$

This trial-averaged boundary is then compared with the tree reference:

$$e_n = \overline{B}^*_{\mathrm{DBDP}}(t_n) - B^*_{\mathrm{tree}}(t_n).$$

Here, B^* is averaged exercise boundary. Let $\mathcal{D}$ be the set of valid matched exercise dates and $m = |\mathcal{D}|$. The three error measures are

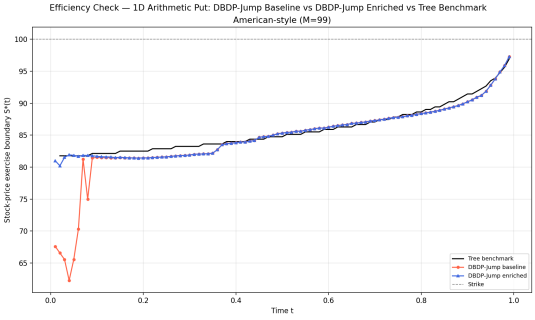
$$\mathrm{MAE} = \frac{1}{m}\sum_{n\in\mathcal{D}} |e_n|, \qquad \mathrm{RMSE} = \sqrt{\frac{1}{m}\sum_{n\in\mathcal{D}} e_n^2}, \qquad \text{max. error} = \max_{n\in\mathcal{D}} |e_n|.$$

Thus, the MAE is the mean absolute discrepancy between the trial-averaged DBDP boundary and the tree reference. The RMSE places greater weight on large boundary errors, while the maximum error records the largest absolute discrepancy among the matched dates. In both cases, sample enrichment substantially reduces the MAE, RMSE, and maximum boundary error. The consistent improvement across all three measures shows that sample enrichment improves exercise-boundary estimation and addresses the observed early-time boundary bias.

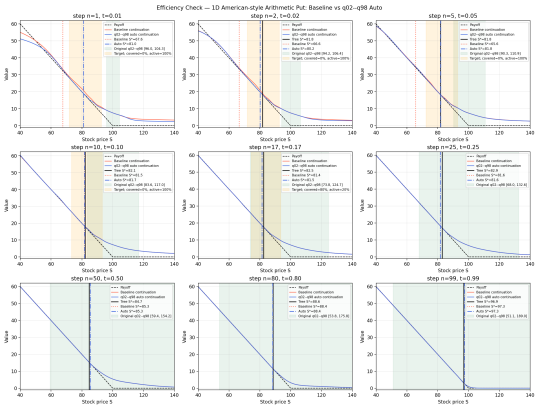
Table 6.5: Exercise-boundary errors for the baseline and sample-enriched DBDP-Jump estimates in the 1D and 5D American-style put tests.

case (d, method)	active step range	MAE	RMSE	max. error
1D, baseline	–	1.4588	3.7347	19.4934
1D, enriched	1–17	0.5954	0.7463	1.5636
5D geometric put, baseline	–	1.3706	3.2907	16.9258
5D geometric put, enriched	1–21	0.5242	0.6805	1.7826

Figures 6.2 and 6.3 provide the corresponding diagnostics for the 1D and 5D cases. Their left panels compare the exercise boundaries obtained by the baseline DBDP-Jump, the sample-enriched DBDP-Jump, and the tree reference. In both cases, the baseline exhibits a pronounced bias at early time steps, whereas the enriched estimate follows the tree boundary much more closely. For the 5D geometric put, the exercise boundary is expressed in terms of the geometric basket level $A(x)$. The right panels show selected time steps in greater detail. The curves show the payoff and the continuation values estimated by the baseline and enriched DBDP-Jump. The vertical lines in matching colors mark the corresponding exercise boundaries. The green region represents the empirical quantile range of the original path pool, and the yellow region represents the target interval from which the enriched basket levels are sampled. At the early steps shown, the original range does not contain the tree boundary, and the baseline boundary is strongly biased. By contrast, the target interval contains the tree boundary, and the enriched estimate is substantially more accurate. As time advances, the original pool covers a wider range of basket levels. Once it adequately covers the target interval and the boundary region, the baseline estimate improves and enrichment is no longer activated. These observations support both the constructed target interval and the automatically selected active ranges. They also show that enrichment improves boundary estimation when the original pool provides insufficient coverage.

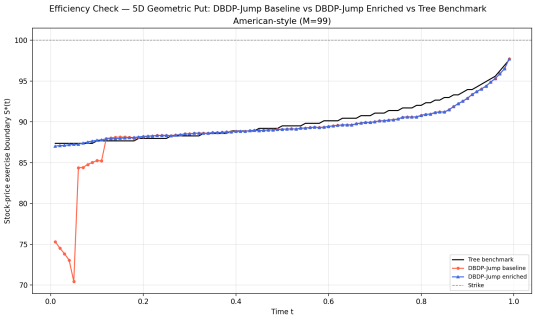


(a) Exercise-boundary comparison.

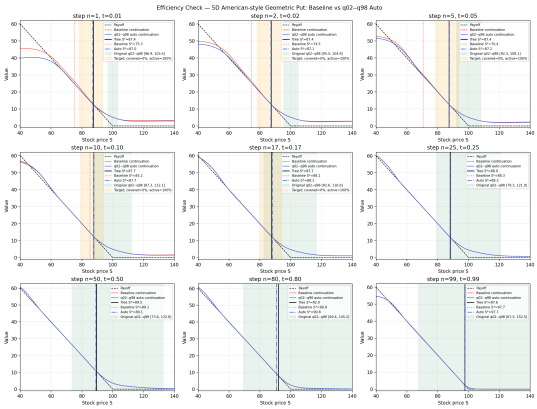


(b) Continuation-value and coverage diagnostics.

Figure 6.2: Sample-enrichment diagnostics for the 1D American-style put.



(a) Exercise-boundary comparison.



(b) Continuation-value and coverage diagnostics.

Figure 6.3: Sample-enrichment diagnostics for the 5D American-style geometric basket put.

However, the improvement in option-value accuracy is much smaller than the improvement in boundary accuracy. In the 1D case, the baseline mean is 7.5900, with a relative error of +0.15%, whereas the enriched mean is 7.5759, with a relative error of -0.04%. In the 5D case, the corresponding means are 5.1716 and 5.1488, with relative errors of +1.38% and +0.93%, respectively. In both cases, the baseline and sample-enriched approaches produce reasonable price estimates, with the enriched estimates closer to their respective reference values. This limited improvement may be due to the exercise boundaries lying in low-probability regions, as suggested by the limited coverage of the original pool. Therefore, correcting the policy in these regions may have little effect on the initial option value. Since only a modest price benefit is observed, subsequent experiments use only the baseline DBDP-Jump.

6.6 High-dimensional problems

With its basic performance confirmed by the preceding sanity checks, DBDP-Jump is next evaluated on high dimensions. The experiments are organized into moderate dimensions (20D and 50D) and very high dimensions (100D and 200D). In the moderate-dimensional cases, the method is first tested under the main problem setting given in Section 6.1. Robustness is then examined by varying the MJD parameters in 20D and considering three other payoff functions in 50D. On 100D and 200D problems, the focus is on scalability therefore only the main setting is considered.

6.6.1 20D and 50D problems

DBDP-Jump is further evaluated on 20D and 50D problems following the main setting. Both use a maximum of 10,000 training iterations. At the terminal step, the 20D problem uses learning rates of 10^{-2} for the first 500 iterations, 10^{-3} for the next 2000 iterations, and 10^{-4} thereafter. For the 50D problem, the corresponding schedule is 10^{-2} for the first 1000 iterations, 10^{-3} for the next 3000 iterations, and 10^{-4} thereafter. The learning rate at all other time steps is fixed at 10^{-4}. The results are reported in Table 6.6.

Table 6.6: DBDP-Jump results for 20D and 50D put options under the main setting.

case (type, d)	ref. value	mean value	s.e.	rel. err.	time (s)
E, 20	2.43	2.4004	0.0297	-1.21%	630
B, 20	–	3.2236	0.0200	–	633
A, 20	–	3.5342	0.0215	–	632
E, 50	2.22	2.2253	0.0341	+0.23%	1069
B, 50	–	3.0956	0.0344	–	1120
A, 50	–	3.2894	0.0171	–	1111

Both European estimates are close to their direct MC benchmarks, with relative errors of -1.21% and 0.23% in 20D and 50D, respectively. The standard errors remain small across all six cases. For each option type, the estimated value decreases as the dimension increases from 20 to 50. Within each dimension, the estimates follow the ordering European $<$ Bermudan $<$ American, consistent with increasing exercise opportunities.

The sensitivity study is conducted for a 20D European put option under different MJD settings (λ, μ_J, σ_J, and ρ_J). The variations are made on the main set problem defined in Section 6.1. European option is considered because, unlike high-dimensional Bermudan or American options under jump-diffusion dynamics, it admits a reliable benchmark price via direct MC simulation. Such a benchmark is needed to assess the accuracy of the DBDP-Jump estimates as the MJD parameters vary. The dimension is fixed at $d = 20$ so that the test problem remains genuinely high-dimensional while the computational cost of the parameter sweeps stays manageable.

The following training settings are shared for all 20D sensitivity tests. To increase model flexibility, the hidden width is set to $d + 100$. The maximum number of training iterations is set to 10,000. At the terminal step, the learning rate is set to 10^{-2} for the first 500 iterations, reduced to 10^{-3} until iteration 2500, and then fixed at 10^{-4} for the remaining iterations. For all other time steps, the learning rate is fixed at 10^{-4}.

Sensitivity to jump intensity λ. The specific values considered in this sensitivity check are $\lambda \in \{0.0, 0.1, 0.3, 0.5, 1.0, 2.0\}$, while all other model parameters are fixed as in the main setting. For $\lambda = 2.0$, the auxiliary jump-sample size and mini-batch size are increased to 512 and 2000, respectively. All other cases use the default values of 256 and 1000. The results are summarized in Table 6.7.

Table 6.7: 20D European put option value estimation under different λ.

λ	ref. value	mean value	s.e.	rel. err.	time (s)
0.0	2.00	2.0267	0.0333	+1.30%	1148
0.1	2.08	2.1243	0.0429	+1.93%	1363
0.3	2.26	2.3265	0.0522	+2.97%	1186
0.5*	2.43	2.5242	0.0413	+3.99%	1136
1.0	2.84	2.8881	0.0282	+1.53%	1152
2.0	3.62	3.6318	0.1335	+0.35%	2253

* marks the main set problem.

Overall, DBDP-Jump provides estimates with acceptable errors across the tested jump intensities. A preliminary run at $\lambda = 2.0$ using the same configuration as the other cases showed noticeable bias; the corresponding result is reported in Appendix C. A higher jump intensity increases both the expected number of realized jumps and the contribution of the jump-related terms to the local loss, making their numerical approximation more demanding. Therefore, the $\lambda = 2.0$ result in Table 6.7 uses more auxiliary jump samples and a larger mini-batch. With this adjustment, the relative error is reduced to 0.35%, while the average runtime rises to 2253 seconds, approximately twice that of the other cases, illustrating a trade-off between estimation accuracy and computational cost.

Sensitivity to jump size distribution $\mathcal{N}_{20}(\mu_J, \Sigma_J)$. The following experiments examine the sensitivity of DBDP-Jump to the jump mean μ_J and jump standard deviation σ_J. Motivated by the λ-sensitivity results, the auxiliary jump-sample size is fixed at 512 to improve the approximation across these jump-size settings.

The jump mean is varied over $\mu_J \in \{-0.4, -0.3, -0.2, -0.1\}$. The results are summarized in Table 6.8.

Table 6.8: 20D European put option value estimation under different μ_J.

μ_J	ref. value	mean value	s.e.	rel. err.	time (s)
-0.40	6.82	6.7396	0.0388	-1.25%	2164
-0.30	4.99	4.9723	0.0826	-0.27%	2136
-0.20	3.43	3.5883	0.0294	+4.72%	2210
-0.10*	2.43	2.4354	0.0618	+0.33%	2277

* marks the main set problem.

The jump standard deviation is varied over $\sigma_J \in \{0.1, 0.2, 0.3, 0.4\}$. The results are summarized in Table 6.9.

Table 6.9: 20D European put option value estimation under different σ_J.

σ_J	ref. value	mean value	s.e.	rel. err.	time (s)
0.10	2.51	2.5355	0.0299	+1.19%	2193
0.20	2.46	2.5136	0.0549	+2.26%	2272
0.30*	2.43	2.4354	0.0618	+0.33%	2277
0.40	2.47	2.5355	0.0854	+2.49%	2324

* marks the main set problem.

Taken together, Tables 6.8 and 6.9 indicate that DBDP-Jump remains reasonably robust to variations in the jump-size distribution. Across the tested values of the jump mean μ_J and jump standard deviation σ_J, the estimated option values remain close to their corresponding reference values, with absolute relative errors no larger than 4.72% and most cases within approximately 2%.

Sensitivity to jump correlation ρ_J. The tested jump correlations are $\rho_J \in \{0.0, 0.3, 0.6, 0.9\}$. The case $\rho_J = 0.0$ corresponds to independent jump sizes across assets. The value $\rho_J = 0.3$ represents a moderate positive jump-size correlation, which is closer to a realistic setting where assets tend to jump in related directions. The case $\rho_J = 0.9$ represents an almost common jump-size movement across assets. The number of auxiliary jump samples is set to 256. The results are summarized in Table 6.10.

Table 6.10: 20D European put option value estimation under different ρ_J.

ρ_J	ref. value	mean value	s.e.	rel. err.	time (s)
0.00*	2.43	2.3762	0.0568	-2.11%	1161
0.30	3.49	3.4845	0.0803	-0.09%	1166
0.60	4.30	4.3396	0.1148	0.89%	1128
0.90	4.98	4.9645	0.1302	-0.40%	1110

* marks the main set problem.

Table 6.10 shows that the DBDP-Jump gives relatively accurate price estimates for all tested jump correlation levels.

*Remark. Note that the main set problem is included in each sensitivity sweep and marked with *. The slight differences among its mean estimates in Tables 6.6, 6.7, and 6.10, which share the same training configuration, are due to the stochastic nature of NN training. Averaging over 5 trials reduces but does not eliminate the effect of this training randomness. In the case reported in Tables 6.8 and 6.9, where the number of auxiliary jump samples increases from 256 to 512, a lower relative error is observed, while the average runtime approximately doubles. However, across all tests for the same main set problem, the absolute relative errors remain below 5%, further supporting the robustness of the DBDP-Jump.*

Thus far, this subsection has considered only the arithmetic basket put payoff. The remaining experiments evaluate DBDP-Jump with three alternative payoffs in 50D to assess its payoff

flexibility in high dimensions. Except for the payoff function, the model and contract parameters remain as specified in Section 6.1. The alternative payoffs, the corresponding option types, and any training modifications are listed below.

- Geometric basket put: $g(x) = \max(K - (\prod_{i=1}^{d} x_i)^{\frac{1}{d}}, 0)$.

The geometric put has a similar price scale to the arithmetic put, so the same training hyperparameters of 50D arithmetic basket put are retained. Because the logarithm of the geometric mean is a linear combination of the asset log-prices, the multidimensional MJD dynamics can reduce to an effective one-dimensional MJD process. The European benchmark can therefore be computed using the Merton semi-closed-form formula [26], while the Bermudan and discretely monitored American-style benchmarks are obtained using the jump-diffusion tree [19]. All three option types are therefore tested, as the DBDP-Jump estimates can be compared directly with their respective benchmarks.

- Put on the minimum: $g(x) = \max(K - \min_{i=1,\dots,d} x_i, 0)$.
- Call on the maximum: $g(x) = \max(\max_{i=1,\dots,d} x_i - K, 0)$.

Unlike the arithmetic and geometric payoffs, which aggregate all basket components, the put on the minimum and call on the maximum depend only on the extreme values of the basket. They therefore provide a structurally different test that is more sensitive to the tails and dependence of the joint asset distribution. Under the equicorrelated MJD setting used here, the common jump count permit semi-analytical European reference values to be computed following the framework of [39]. These deterministic benchmarks avoid the sampling noise present in direct MC estimates. In addition to the European case, the put on the minimum is further tested as a Bermudan option to assess the exercise projection for an extremum payoff. For the call on the maximum, only the European case is tested, since with non-dividend-paying assets and $r > 0$, early exercise does not add value and the European, Bermudan, and American-style prices coincide. The European benchmarks show that, under the present 50D parameter setting, both extreme-value payoffs produce option values on a substantially larger scale than the arithmetic and geometric basket puts. Because the DBDP-Jump local objective (3.12) is a mean-squared residual, the larger target scale leads to larger loss magnitudes and greater sensitivity to the learning rate. Accordingly, the maximum iteration budget is increased to 20,000, and the adaptive learning-rate design introduced in Subsection 6.6.2 is applied to all extreme-value cases. The common settings are a plateau patience of 800 iterations, a relative tolerance of 0.1%, a learning-rate reduction factor of 0.5, and a minimum learning rate of 10^{-6}.

Table 6.11: 50D problems with different payoff functions and option types.

case (payoff, option type)	ref. value	mean value	s.e.	rel. err.	time (s)
Geometric put, E	3.3427	3.3379	0.0121	-0.14%	1177
Geometric put, B	3.9826	4.0276	0.0264	+1.13%	1215
Geometric put, A	4.1793	4.3249	0.0263	+3.49%	2635
Put on min , E	29.8017	29.8155	0.0202	+0.05%	3036
Put on min , B	–	31.4630	0.0217	–	2972
Call on max , E	75.4985	75.0936	0.1745	-0.54%	3054

Table 6.11 shows that DBDP-Jump remains reasonably accurate across the three alternative payoff functions. Among all cases with independent benchmarks, the absolute relative errors do not exceed 3.49%, while those of the three European cases do not exceed 0.54%. The

extreme-value cases have average runtimes around 3000 seconds. This higher computational cost is consistent with the larger loss magnitudes and the greater iteration budget allocated to these cases. Overall, the results provide evidence that DBDP-Jump extends beyond the arithmetic basket payoff.

6.6.2 100D and 200D problems

Learning-rate control becomes more critical in high dimensions. As the dimension increases, the local DBDP-Jump loss becomes more sensitive to the learned parameters, and the loss landscape becomes more complicated. An aggressive gradient step can therefore produce a sudden loss explosion. Once this happens, the optimizer may not recover within the remaining local training budget. A detailed analysis of the causes of loss explosions, together with a representative example, is provided in Appendix B. In very high dimensions, a pre-specified learning-rate schedule may require repeated fine-tuning based on the loss convergence observed at the first training step. A simple three-stage schedule may therefore be insufficiently adaptive for such cases. To reduce manual tuning and improve training efficiency, the learning rate should be adjusted dynamically in response to the observed optimization behaviour. Hence, the 100D and 200D experiments use an adaptive learning-rate design.

The adaptive learning-rate design is applied consistently across all time steps. It automatically reduces the current learning rate when the validation progress has plateaued, and it restores the model with the lowest recorded validation loss at the end of the local training rather than simply using the latest checkpoint. A concise flowchart is shown in Figure 6.4.

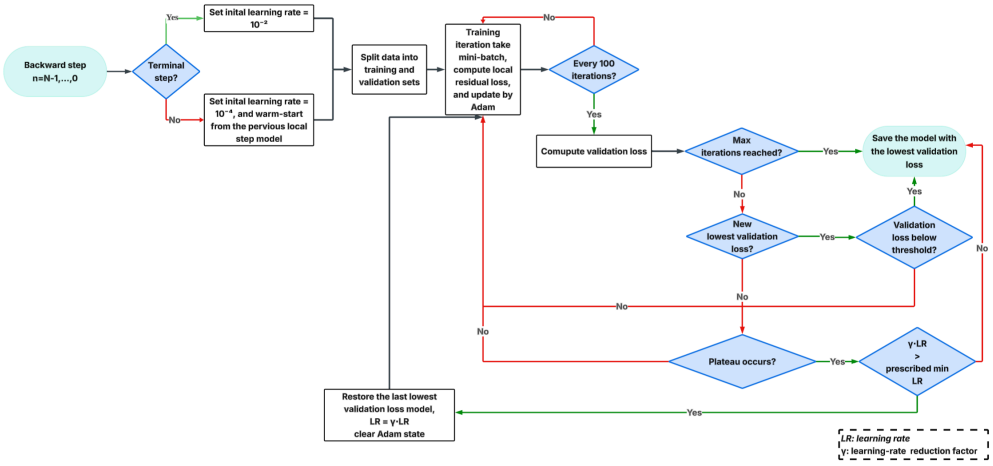


Figure 6.4: Adaptive learning-rate design used in every training step.

In the experiments, the validation loss is evaluated every 100 iterations. The initial learning rate is 10^{-2} for the terminal step and 10^{-4} for all subsequent backward steps. The model is saved whenever a new lowest validation loss is observed. A validation loss plateau is defined as the absence of significant improvement over a prescribed number of iterations, which is 500 iterations for 100D and 1000 iterations for 200D. An improvement is regarded as significant only if the new validation loss decreases by at least a prescribed relative tolerance compared

with the previous best value. In the reported estimation experiments, this tolerance is set to a decrease of at least 10% for 100D and 0.1% for 200D. When a plateau is detected, the best saved model is reloaded, and the learning rate is multiplied by 0.1 for 100D and by 0.5 for 200D. A minimum learning rate is imposed to avoid ineffective updates with excessively small step sizes. It is 10^{-4} for 100D and 10^{-6} for 200D.

The local training process terminates when the validation loss falls below the stopping threshold 0.01, when the maximum iteration budget is reached, or when a plateau occurs but the reduced learning rate would be smaller than the dimension-specific minimum learning rate. The maximum iteration budget is 10,000 for 100D and 20,000 for 200D. At the end of each local training step, the model with the lowest recorded validation loss is restored and saved as the final model for that step.

Table 6.12: DBDP-Jump pricing results for 100D and 200D arithmetic basket put options.

case(type, d)	ref. value	mean value	s.e.	rel. err.	time (s)
E, 100	2.14	2.1141	0.0433	-1.21%	2280
B, 100	-	2.9525	0.0546	–	2182
A, 100	-	3.1978	0.0393	–	2109
E, 200	2.10	2.0809	0.0108	-0.91%	4179
B, 200	-	2.8561	0.0143	–	4469
A, 200	-	3.1519	0.0144	–	4566

The results in Table 6.12 show stable behavior of the proposed DBDP-Jump method in both high-dimensional cases. For the European options, where reference values are available, the estimated prices remain close to the corresponding reference values. The relative differences are below 2% for both $d = 100$ and $d = 200$, which provides a useful consistency check for the learned continuation scheme in the absence of early exercise. Under the arithmetic basket put payoff considered here, the expected ordering across option types is also preserved: for each fixed dimension, the Bermudan value is larger than the European value, and the American value is larger than the Bermudan value. This is consistent with the increasing flexibility of the exercise opportunity set. For the same payoff and option type, the estimated value decreases slightly when the dimension increases from 100 to 200, matching the trend observed in the reference European prices. The average training time roughly doubles from the 100D to the 200D case, which is reasonable given that the problem dimension is doubled and already large. In addition, a larger iteration budget, longer plateau patience, and a lower minimum learning rate are used for the 200D setting, all of which further extend the training time.

7 Discussion

7.1 Main findings

This thesis developed DBDP-Jump for pricing high-dimensional Bermudan options under multidimensional MJD dynamics. On each interval between consecutive exercise dates, the continuation value satisfies a PIDE and, through the Feynman-Kac representation for jump diffusions, admits an equivalent BSDE with jumps (BSDEj) representation. The optimal stopping problem can therefore be formulated as a backward recursion over a sequence of local BSDEs with jumps. This construction is extended from the DBDP1 scheme of [1], which was developed for high-dimensional parabolic PDEs. DBDP-Jump adapts this local backward scheme to the PIDE induced by MJD dynamics. At each time step, two NNs approximate the continuation value and its spatial gradient. Two further adaptations incorporate the Bermudan and jump features: a pointwise maximum with the exercise payoff is applied at each admissible exercise date, and the jump integral is separated into realized-jump and compensator terms. Both terms are evaluated from the learned next-step value function, with auxiliary MC used to approximate the compensator. Unlike the approach in [2], DBDP-Jump therefore requires no additional NN for the jump integrand. The same construction can extend to other finite-activity jump-diffusion models when the jump-size distribution can be sampled, e.g. the Kou model [18]. An error analysis is done by combining the backward time-discretization, local NN approximation, and jump-propagation errors.

The numerical experiments first validate DBDP-Jump on benchmarked 1D and 5D problems, then extend the evaluation to 20D, 50D, 100D, and 200D to assess scalability. The main tests from 20D to 200D cover European, Bermudan, and American-style options. The agreement between the European estimates and their reference values provides evidence that pricing accuracy is maintained as the dimension increases. For the early-exercise cases, the expected ordering of option values is preserved, supporting the numerical consistency of the estimates. The benchmarked estimates also remain reasonably accurate when the MJD parameters are varied in 20D and the payoff function is varied in 50D. Computational cost depends on the dimension, the number of auxiliary jump samples, the permitted training iterations, and the problem-dependent difficulty of the local regression. Across the main dimensional tests, the observed runtime grows approximately in proportion to dimension, indicating favorable empirical scaling over the tested range.

7.2 Practical considerations for time discretization and training hyperparameters

The consistency inequality (4.21) shows that time-grid refinement and local training accuracy must be considered jointly. Refining the grid reduces the backward discretization error, but

it also creates more local learning problems. The local value-network and jump-propagation errors accumulate over the backward recursion and are weighted by the number of time steps N. A finer grid need not improve the final estimate under fixed network capacity, sample sizes, and optimization budgets unless these local errors are reduced at the same time. The grids with $N = 100$ used in the experiments should therefore be viewed as a practical compromise: they resolve the prescribed exercise dates without making the sequence of local regressions unnecessarily long. The practical grid-refinement results in Section 6.4 show a pattern consistent with this discussion.

Within the tested problem family, the experiments identify a useful baseline training configuration. To avoid repeated MC generation during local optimization, each trial reuses an offline pool of simulated state trajectories and a fixed collection of auxiliary jump samples throughout the backward recursion. Two hidden $\tanh$ layers of width $d + 30$, input normalization, an offline pool of 100,000 paths, a mini-batch size of 1000, and 256 auxiliary jump samples were sufficient for most cases. A maximum of 10,000 iterations, a three-stage terminal learning-rate schedule from 10^{-2} to 10^{-4}, and a fixed rate of 10^{-4} at the remaining steps gave stable results in most low- and moderate-dimensional problems. Note that these settings are only empirical defaults. More auxiliary samples and a larger mini-batch are beneficial when the jump terms become more variable, as observed for high jump intensity. For 100D, 200D, and the larger-scale extreme-value payoffs, learning-rate reduction with restoration of the best checkpoint is more reliable than a fixed schedule, and the most demanding cases require up to 20,000 iterations. The stopping threshold must likewise reflect the loss scale. The default 0.01 was adequate for most $N = 100$ tests. The 5D sanity check used a different MJD setting with lower volatility and jump variation, producing smaller one-step residuals. Its threshold was therefore reduced to 0.002 to avoid premature stopping. Thus, dimension, jump variability, and payoff scale are the main practical signals for increasing the sampling budget or strengthening learning-rate control.

7.3 Use and limitations of sample enrichment

The sample-enrichment modification addresses a specific weakness of training on forward paths. At early times, states generated from a fixed initial condition may not cover the exercise-relevant region, forcing the learned continuation function to extrapolate when locating the exercise boundary. The proposed procedure uses empirical remaining-return quantiles to construct a target interval, rescales existing states into that interval, and activates enrichment only until the original sample pool provides adequate coverage. In both the 1D test and the reducible 5D geometric-basket test, enrichment substantially reduces the early-time exercise-boundary error. It is therefore useful when the stopping policy or boundary is itself of interest and the original path pool provides insufficient coverage of the relevant region.

However, the same tests show only a modest improvement in the option price estimate. Two factors may explain why the improvement is small. First, the option value is defined as the supremum of expected discounted payoffs over admissible stopping times under the risk-neutral measure, whereas the enriched states are generated by rescaling. Although this rescaling preserves relative asset composition, it does not preserve the original state distribution. Second, the failure of the original sample pool to reach the early boundary suggests that this boundary lies in a low-probability region. Correcting stopping decisions in this region may have little effect on the initial option value.

A DBDP-Jump algorithm

Algorithm A.1: DBDP-Jump (baseline)

Require: Dimension d; number of time intervals N; maturity T; model parameters μ_J, Σ_J, λ; initial asset price x_0; exercise time set $\mathcal{T}$.

Require: Network depth L_{NN}; hidden width m; activation function; optimizer; learning-rate schedule; maximum number of training iterations $\mathrm{MaxIter}$; pool sample size P; batch size B; auxiliary jump-sample size M_J.

Output: Trained value approximation $\widehat{V}_0(x_0)$.

Initialize $\widehat{V}_N(\cdot) = g(\cdot)$.

Sample P paths of the underlying asset process. Record the pre-jump states, jump counts ΔN_n, realized jump sizes $Z_{n,k}$, and Brownian increments ΔW_n.

Fix the training and validation index sets using the prescribed split of the path pool.

Sample one auxiliary set $\{Z^{(j)}\}_{j=1}^{M_J}$, where $Z^{(j)} \sim \mathcal{N}_d(\mu_J, \Sigma_J)$, and reuse it throughout the trial.

for $n = N-1, \dots, 0$ **do**

  Initialize the time-n network randomly if $n = N-1$; otherwise warm-start it from the trained time-$(n+1)$ network.

  Using the fixed $\widehat{V}_{n+1}$, precompute and store for every path $p = 1, \dots, P$:

$$\widehat{V}_{n+1}^{(p)} = \widehat{V}_{n+1}\left(X_{n+1}^{(p)}\right), \qquad \widehat{J}_n^{(p)} \text{ from (3.13)}, \qquad \widehat{C}_n^{(p)} \text{ from (3.15)},$$

  where $\widehat{C}_n^{(p)}$ is evaluated using the fixed auxiliary set $\{Z^{(j)}\}_{j=1}^{M_J}$.

  for $i = 1, \dots, \mathrm{MaxIter}$ **do**

    Select a mini-batch index set $\mathcal{B}_i$ of size B from the training pool and retrieve the corresponding precomputed quantities.

    Evaluate the current network outputs for $p \in \mathcal{B}_i$:

$$\widehat{Y}_n^{(p,i)} = \mathcal{Y}_n(X_n^{(p)}; \theta_n^{\mathcal{Y}}), \qquad \widehat{\nabla_x u}_n^{(p,i)} = \mathcal{Z}_n(X_n^{(p)}; \theta_n^{\mathcal{Z}}).$$

    Form the one-step map:

$$F_n^{(p,i)} = \widehat{Y}_n^{(p,i)} + r\widehat{Y}_n^{(p,i)}\,\Delta t_n + \left(\sigma_X(X_n^{(p)})\widehat{\nabla_x u}_n^{(p,i)}\right)^{\top} \Delta W_n^{(p)} + \widehat{J}_n^{(p)} - \widehat{C}_n^{(p)}\,\Delta t_n, \qquad p \in \mathcal{B}_i.$$

    Compute the empirical mini-batch loss:

$$Loss_n^i(\theta_n) = \frac{1}{|\mathcal{B}_i|} \sum_{p \in \mathcal{B}_i} \left| \widehat{V}_{n+1}^{(p)} - F_n^{(p,i)} \right|^2.$$

    Update $\theta_n^{\mathcal{Y}}$ and $\theta_n^{\mathcal{Z}}$ using the optimizer and learning-rate schedule.

  if $t_n \in \mathcal{T}$ **then**

    Set

$$\widehat{V}_n(\cdot) \leftarrow \max\left\{g(\cdot), \mathcal{Y}_n(\cdot; \theta_n^{\mathcal{Y},\star})\right\}.$$

  else

    Set

$$\widehat{V}_n(\cdot) \leftarrow \mathcal{Y}_n(\cdot; \theta_n^{\mathcal{Y},\star}).$$

B Loss explosions and the need for adaptive learning-rate control in DBDP-Jump

At time step $N-1$, the loss function is

$$Loss_{N-1}(\theta_{N-1}) = \mathbb{E}\left[|g(X_N) - F_{N-1}(\theta_{N-1})|^2\right],$$

where the one-step map $F_{N-1}(\theta_{N-1})$ is given by

$$\begin{aligned} F_{N-1}(\theta_{N-1}) = {} & \mathcal{Y}_{N-1}(X_{N-1};\theta^{\mathcal{Y}}_{N-1}) + r\,\mathcal{Y}_{N-1}(X_{N-1};\theta^{\mathcal{Y}}_{N-1})\Delta t_{N-1} \\ & + \left(\sigma_X(X_{N-1})\mathcal{Z}_{N-1}(X_{N-1};\theta^{\mathcal{Z}}_{N-1})\right)^\top \Delta W_{N-1} + \widehat{J}_{N-1} - \widehat{C}_{N-1}\Delta t_{N-1}. \end{aligned}$$

Since $\widehat{V}_N = g$ at the terminal time step N, both the jump term $\widehat{J}_{N-1}$ and the compensator $\widehat{C}_{N-1}$ are evaluated using the known terminal payoff g rather than a learned approximation. Moreover, these two terms are treated as constants during training. Consequently, at time step $N-1$ the jump and compensator terms introduce only MC noise and do not contribute trainable approximation error. Among the remaining terms, the drift term $r\,\mathcal{Y}_{N-1}(X_{N-1};\theta^{\mathcal{Y}}_{N-1})\Delta t_{N-1}$ is scaled by Δt_{N-1} and is therefore small. The dominant source of trainable error is the diffusion term

$$\left(\sigma_X(X_{N-1})\mathcal{Z}_{N-1}(X_{N-1};\theta^{\mathcal{Z}}_{N-1})\right)^\top \Delta W_{N-1} = \sum_{i=1}^{d} \sigma_i X^i_{N-1}\mathcal{Z}^i_{N-1}(X_{N-1};\theta^{\mathcal{Z}}_{N-1})\Delta W^i_{N-1}.$$

As $\mathcal{Z}_{N-1}(X_{N-1};\theta^{\mathcal{Z}}_{N-1}) \in \mathbb{R}^d$ is a NN approximation of the gradient of the value function and $\Delta W_{N-1} \in \mathbb{R}^d$ is a d-dimensional Brownian increment, the error contribution of this inner product grows with the dimension d. Hence, the diffusion term is increasingly sensitive to inaccuracies in $\mathcal{Z}_{N-1}(X_{N-1};\theta^{\mathcal{Z}}_{N-1})$ as the dimension rises.

Even when the Adam optimizer is used, the base learning rate still plays a critical role in the training process. With a fixed and large learning rate such as 10^{-2}, after a sufficient number of iterations, the parameter updates can overshoot the neighborhood of a local minimizer, causing oscillations in the loss value. Furthermore, in the Adam update rule $\theta_{i+1} = \theta_t - \eta\frac{\hat{m}_i}{\sqrt{\hat{v}_i}+\epsilon}$, the first-moment estimate $\hat{m}_t$ accumulates the history of these oscillating gradients. If the second-moment estimate $\hat{v}_t$ happens to be small at some iteration. For instance, after passing through a relatively flat region of the loss landscape. The effective step size $\frac{\eta}{\sqrt{\hat{v}_t}+\epsilon}$ can become temporarily very large, pushing the parameters into a region where the gradient is steep. Once in such a region, the NN approximated gradient $\mathcal{Z}_{N-1}(X_{N-1};\theta^{\mathcal{Z}}_{N-1})$ deviates substantially from the true gradient of the value function. In high dimensions, since the parameter space of the network is larger and the loss landscape is more complex, such destabilizing jumps are more likely to occur and the accumulation effect of the inner product can lead to an extremely large

loss value at some iterations. This phenomenon is observed at the terminal backward step $N-1$ in the 50D problem with a fixed learning rate of 10^{-2}. As shown by the blue curves in Figure B.1, the loss can suddenly exceed 1000 after several iterations. This results in poor convergence, and the training process struggles to recover from such a spike.

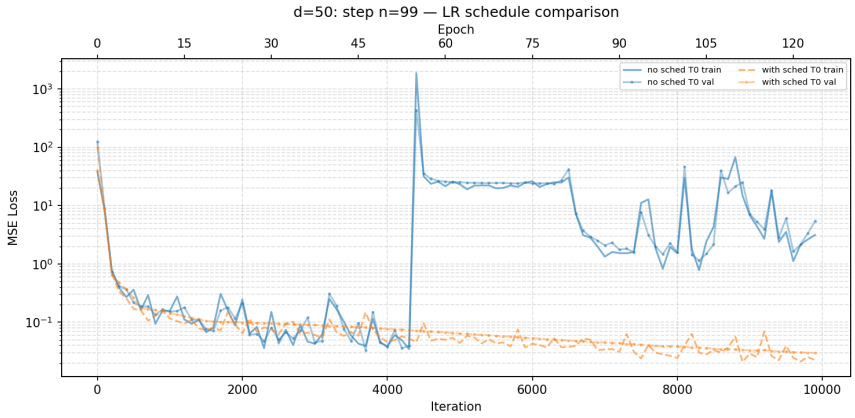
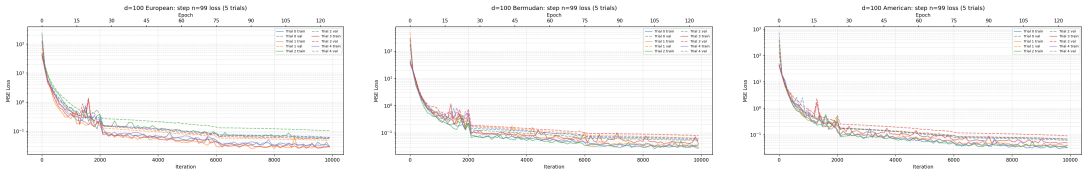


Figure B.1: Loss value explosion at time step N-1 (an example of 50D problem).

Therefore, a learning-rate schedule is adopted at time step $N-1$ to mitigate the risk of loss explosions. A learning-rate schedule consisting of 10^{-2} for the first 500 iterations, 10^{-3} for the next 1000 iterations, and 10^{-4} thereafter yields consistent convergence, as illustrated by the orange curves in Figure B.1.

However, in even higher dimensions, a pre-specified three-stage learning-rate schedule may require further tuning based on the loss convergence observed at the terminal backward step $n = N-1$. For example, the 100D problems under the main setting use a maximum of 10,000 iterations. The learning rate is set to 10^{-2} for the first 2000 iterations, 10^{-3} for the next 4000 iterations, and 10^{-4} thereafter. Figure B.2 shows the corresponding training and validation losses at time step $N-1$. Oscillations between iterations 1000 and 2000 are consistently observed across the European, Bermudan, and American cases. This pattern suggests that reducing the learning rate to 10^{-3} earlier, for example at iteration 1500, may improve convergence stability.



(a) European. (b) Bermudan. (c) American.

Figure B.2: Training and validation losses at the terminal backward step $N-1$ for the 100D cases under a three-stage learning-rate schedule.

Thus, manually selecting the learning-rate levels and transition points becomes more demanding as the dimension increases. A schedule calibrated for one setting may require repeated

adjustment based on the observed terminal-step convergence in another. This increasing calibration burden motivates the adaptive learning-rate design introduced in Subsection 6.6.2, which responds directly to validation progress.

C DBDP-Jump under high jump intensity

This appendix supplements the jump-intensity sensitivity study reported in Table 6.7 by examining the preliminary $\lambda = 2.0$ result obtained with the default sampling configuration.

For $\lambda \leq 1.0$, the sensitivity experiments use a mini-batch size of 1000 and $M_J = 256$ auxiliary jump samples. As a direct test of whether the same configuration remains adequate at a higher jump intensity, it was initially retained for $\lambda = 2.0$. Table C.1 reports this preliminary test gives a relative error of -8.23% and a standard error of 0.1896 which is apparently higher than those obtained for $\lambda \leq 1.0$.

Table C.1: Preliminary 20D European put option value estimate for $\lambda = 2.0$

λ	ref. value	mean value	s.e.	rel. err.
2.0	3.62	3.3215	0.1896	-8.23%

Figure C.1 shows the training and validation losses at the terminal step $N - 1$ under the same default sampling configuration. It can be seen that the curves for the case $\lambda = 2.0$, exhibits more pronounced and recurrent fluctuations across the five trials. These fluctuations remain visible after the scheduled learning-rate reductions.

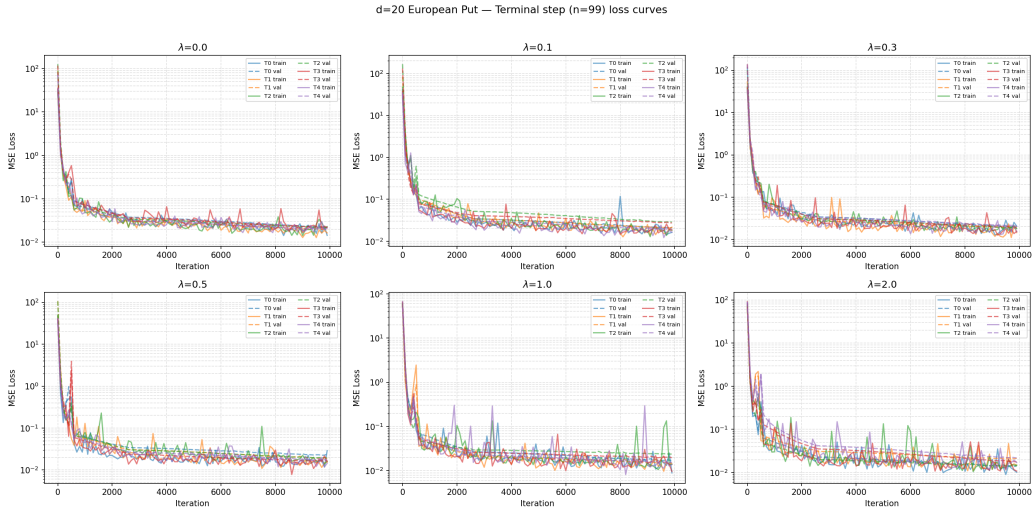


Figure C.1: Training and validation losses at the terminal step $N - 1 = 99$ under different jump intensities, using a mini-batch size of 1000 and 256 auxiliary jump samples.

For a Poisson jump process, a larger λ means that realized-jump contributions occur more frequently in the pathwise training targets, potentially increasing the variability of stochastic

gradient updates. Increasing the mini-batch size averages the local loss over more trajectories and may therefore provide a practical way to improve training stability. The auxiliary sample size addresses a different source of error. In (3.15), the sample average used to approximate the compensator is multiplied by λ. Consequently, for a fixed M_J, the effect of its finite-sample MC error can become more pronounced as λ increases. Together, these considerations motivate increasing the mini-batch size from 1000 to 2000 and doubling the auxiliary jump-sample size from 256 to 512 for the $\lambda = 2.0$ test.

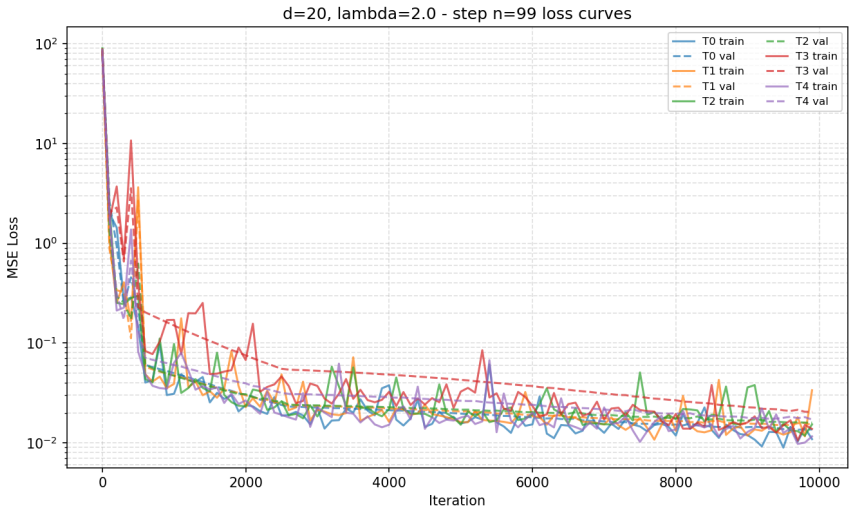


Figure C.2: Training and validation losses at the terminal step $N - 1 = 99$ for $\lambda = 2.0$, using the adjusted mini-batch size of 2000 and 512 auxiliary jump samples.

With the adjusted sampling configuration, the loss curves in Figure C.2 settle into a more stable range after the initial training phase. The corresponding estimate reported in Table 6.7 is 3.6318, with a standard error of 0.1335 and a relative error of +0.35%.

List of Figures

6.1 Grid-refinement results for the 1D European put. 38
6.2 Sample-enrichment diagnostics for the 1D American-style put. 40
6.3 Sample-enrichment diagnostics for the 5D American-style geometric basket put. 40
6.4 Adaptive learning-rate design used in every training step. 45

B.1 Loss value explosion at time step N-1 (an example of 50D problem). 52
B.2 Training and validation losses at the terminal backward step $N-1$ for the 100D cases under a three-stage learning-rate schedule. 52

C.1 Training and validation losses at the terminal step $N-1=99$ under different jump intensities, using a mini-batch size of 1000 and 256 auxiliary jump samples. 55
C.2 Training and validation losses at the terminal step $N-1=99$ for $\lambda=2.0$, using the adjusted mini-batch size of 2000 and 512 auxiliary jump samples. 56

List of Tables

6.1 Option types and corresponding admissible exercise steps used in the numerical experiments. 33
6.2 DBDP-Jump sanity check results for 1D problems. 35
6.3 DBDP-Jump sanity check result for the 5D Bermudan problem. 36
6.4 Grid-refinement results for the 1D European put. 37
6.5 Exercise-boundary errors for the baseline and sample-enriched DBDP-Jump estimates in the 1D and 5D American-style put tests. 39
6.6 DBDP-Jump results for 20D and 50D put options under the main setting. . . . 41
6.7 20D European put option value estimation under different λ. 42
6.8 20D European put option value estimation under different μ_J. 42
6.9 20D European put option value estimation under different σ_J. 43
6.10 20D European put option value estimation under different ρ_J. 43
6.11 50D problems with different payoff functions and option types. 44
6.12 DBDP-Jump pricing results for 100D and 200D arithmetic basket put options. . 46

C.1 Preliminary 20D European put option value estimate for $\lambda = 2.0$ 55

List of Algorithms

A.1 DBDP-Jump (baseline) 50

Acronyms

PDE partial differential equation

FFT fast Fourier transform

COS Fourier-cosine method

LSMC least-squares Monte Carlo

NN neural network

BSDE backward stochastic differential equation

DBDP deep backward dynamic programming

PIDE partial integro-differential equation

SGBM stochastic grid bundling method

FBSDE forward-backward stochastic differential equation

MJD Merton jump-diffusion

BSDEj backward stochastic differential equation with jumps

SDE stochastic differential equation

DBDP1 deep backward dynamic programming scheme 1

ReLU rectified linear unit

ELU exponential linear unit

UAT universal approximation theorem

MC Monte Carlo

ITM in-the-money

SMC sequential Monte Carlo

IS importance sampling

MCMC Markov chain Monte Carlo

MAE mean absolute error

RMSE root mean squared error

Bibliography

[1] Côme Huré, Huyên Pham, and Xavier Warin. Deep backward schemes for high-dimensional nonlinear PDEs. *Mathematics of Computation*, 89(324):1547–1579, 2020. doi: 10.1090/mcom/3514.

[2] Javier Castro. Deep learning schemes for parabolic nonlocal integro-differential equations. *Partial Differential Equations and Applications*, 3(6):77, October 2022. doi: 10.1007/s42985-022-00213-z.

[3] Martin Schweizer. On Bermudan Options. In Klaus Sandmann and Philipp J. Schönbucher, editors, *Advances in Finance and Stochastics: Essays in Honour of Dieter Sondermann*, pages 257–270. Springer, Berlin, Heidelberg, 2002. ISBN 978-3-662-04790-3. doi: 10.1007/978-3-662-04790-3_13.

[4] Michael J. Brennan and Eduardo S. Schwartz. The Valuation of American Put Options. *The Journal of Finance*, 32(2):449–462, 1977. doi: 10.2307/2326779.

[5] Yves Achdou and Olivier Pironneau. *Computational Methods for Option Pricing*. Society for Industrial and Applied Mathematics, 2005. ISBN 978-0-89871-573-6 978-0-89871-749-5. doi: 10.1137/1.9780898717495.

[6] John C. Cox, Stephen A. Ross, and Mark Rubinstein. Option pricing: A simplified approach. *Journal of Financial Economics*, 7(3):229–263, 1979. doi: 10.1016/0304-405X(79)90015-1.

[7] 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 Journal on Scientific Computing*, 30(4):1678–1705, 2008. doi: 10.1137/070683878.

[8] F. Fang and C. W. Oosterlee. Pricing early-exercise and discrete barrier options by fourier-cosine series expansions. *Numerische Mathematik*, 114(1):27–62, November 2009. doi: 10.1007/s00211-009-0252-4.

[9] Richard Bellman. Dynamic Programming. *Science*, 153(3731):34–37, July 1966. doi: 10.1126/science.153.3731.34.

[10] Jérôme Barraquand and Didier Martineau. Numerical Valuation of High Dimensional Multivariate American Securities. *Journal of Financial and Quantitative Analysis*, 30(3):383–405, September 1995. doi: 10.2307/2331347.

[11] Mark Broadie and Paul Glasserman. A stochastic mesh method for pricing high-dimensional American options. *The Journal of Computational Finance*, 7(4):35–72, 2004. doi: 10.21314/JCF.2004.117.

[12] Francis A. Longstaff and Eduardo S. Schwartz. Valuing American Options by Simulation: A Simple Least-Squares Approach. *Review of Financial Studies*, 14(1):113–147, 2001. doi: 10.1093/rfs/14.1.113.

[13] Bernard Lapeyre and Jérôme Lelong. Neural network regression for Bermudan option pricing. *Monte Carlo Methods and Applications*, 27(3):227–247, 2021. doi: 10.1515/mcma-2021-2091.

[14] Sebastian Becker, Patrick Cheridito, and Arnulf Jentzen. Deep optimal stopping. arXiv preprint arXiv:1804.05394, 2020. doi: 10.48550/arXiv.1804.05394.

[15] Weinan E, Jiequn Han, and Arnulf Jentzen. Deep Learning-Based Numerical Methods for High-Dimensional Parabolic Partial Differential Equations and Backward Stochastic Differential Equations. *Communications in Mathematics and Statistics*, 5(4):349–380, 2017. doi: 10.1007/s40304-017-0117-6.

[16] Haojie Wang, Han Chen, Agus Sudjianto, Richard Liu, and Qi Shen. Deep Learning-Based BSDE Solver for Libor Market Model with Application to Bermudan Swaption Pricing and Hedging. arXiv preprint arXiv:1807.06622, 2018. doi: 10.48550/arXiv.1807.06622.

[17] Christian Beck, Sebastian Becker, Patrick Cheridito, Arnulf Jentzen, and Ariel Neufeld. Deep Splitting Method for Parabolic PDEs. *SIAM Journal on Scientific Computing*, 43(5): A3135–A3154, 2021. doi: 10.1137/19M1297919.

[18] S. G. Kou. A jump-diffusion model for option pricing. *Management Science*, 48(8):1086–1101, August 2002. doi: 10.1287/mnsc.48.8.1086.166.

[19] Kaushik I. Amin. Jump Diffusion Option Valuation in Discrete Time. *The Journal of Finance*, 48(5):1833–1863, 1993. doi: 10.2307/2329069.

[20] Jari Toivanen. Numerical valuation of european and american options under kou's jump-diffusion model. *SIAM Journal on Scientific Computing*, 30(4):1949–1970, 2008. doi: 10.1137/060674697.

[21] Shashi Jain and Cornelis W. Oosterlee. The Stochastic Grid Bundling Method: Efficient pricing of Bermudan options and their Greeks. *Applied Mathematics and Computation*, 269:412–431, 2015. doi: 10.1016/j.amc.2015.07.085.

[22] F. Cong and C.W. Oosterlee. Pricing Bermudan options under Merton jump-diffusion asset dynamics. *International Journal of Computer Mathematics*, 92(12):2406–2432, 2015. doi: 10.1080/00207160.2015.1070838.

[23] Liwei Lu, Hailong Guo, Xu Yang, and Yi Zhu. Temporal Difference Learning for High-Dimensional PIDEs with Jumps. *SIAM Journal on Scientific Computing*, 46(4):C349–C368, 2024. doi: 10.1137/23M1584538.

[24] Reiichiro Kawai, Riu Naito, and Toshihiro Yamada. A deep learning-based forward scheme for forward-backward SDEs with jumps. *Journal of Scientific Computing*, 106 (3):76, 2026. doi: 10.1007/s10915-025-03176-6.

[25] Kristoffer Andersson, Alessandro Gnoatto, Marco Patacca, and Athena Picarelli. A Deep Solver for BSDEs with Jumps. *SIAM Journal on Financial Mathematics*, 16(3):875–911, 2025. doi: 10.1137/23M1615048.

[26] Robert C. Merton. Option pricing when underlying stock returns are discontinuous. *Journal of Financial Economics*, 3(1):125–144, 1976. doi: 10.1016/0304-405X(76)90022-2.

[27] Łukasz Delong. *Backward Stochastic Differential Equations with Jumps and Their Actuarial and Financial Applications: BSDEs with Jumps*. EAA Series. Springer London, 2013. ISBN 978-1-4471-5330-6 978-1-4471-5331-3. doi: 10.1007/978-1-4471-5331-3.

[28] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, December 1943. doi: 10.1007/BF02478259.

[29] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958. doi: 10.1037/h0042519.

[30] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, January 1991. doi: 10.1016/0893-6080(91)90009-T.

[31] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization, January 2017. doi: 10.48550/arXiv.1412.6980.

[32] Shanjian Tang and Xunjing Li. Necessary conditions for optimal control of stochastic systems with random jumps. *SIAM Journal on Control and Optimization*, 32(5):1447–1475, 1994. doi: 10.1137/S0363012992233858.

[33] Bruno Bouchard and Romuald Elie. Discrete-time approximation of decoupled Forward–Backward SDE with jumps. *Stochastic Processes and their Applications*, 118(1):53–75, January 2008. doi: 10.1016/j.spa.2007.03.010.

[34] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, January 1989. doi: 10.1016/0893-6080(89)90020-8.

[35] Peter W. Glynn and Donald L. Iglehart. Importance Sampling for Stochastic Simulations. *Management Science*, 35(11):1367–1392, November 1989. doi: 10.1287/mnsc.35.11.1367.

[36] Pierre Del Moral, Arnaud Doucet, and Ajay Jasra. Sequential monte carlo samplers. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 68(3):411–436, June 2006. doi: 10.1111/j.1467-9868.2006.00553.x.

[37] Ajay Jasra and Pierre Del Moral. Sequential Monte Carlo Methods for Option Pricing. *Stochastic Analysis and Applications*, 29(2):292–316, February 2011. doi: 10.1080/07362994.2011.548993.

[38] Robert B. Gramacy and Michael Ludkovski. Sequential Design for Optimal Stopping Problems. *SIAM Journal on Financial Mathematics*, 6(1):748–775, January 2015. doi: 10.1137/140980089.

[39] Xingchun Wang. Pricing options on the maximum or minimum of multi-assets under jump-diffusion processes. *International Review of Economics & Finance*, 70:16–26, November 2020. doi: 10.1016/j.iref.2020.05.014.