

Mean-Field Neural Networks-Based Algorithms for McKean-Vlasov Control Problems

Huy en Pham *¹ and Xavier Warin †²

¹Laboratoire de Probabilit es, Statistique et Mod elisation, Universit  Paris Cit , Paris 75205, France

²Electricit  de France, Research and Development, Palaiseau 91120, France

Abstract. This paper is devoted to the numerical resolution of McKean-Vlasov control problems via the class of mean-field neural networks introduced in our companion paper [Pham and Warin, Neural Netw., 168, 2023] in order to learn the solution on the Wasserstein space. We propose several algorithms either based on dynamic programming with control learning by policy or value iteration, or backward stochastic differential equation SDE from stochastic maximum principle with global or local loss functions. Extensive numerical results on different examples are presented to illustrate the accuracy of each of our eight algorithms. We discuss and compare the pros and cons of all the tested methods.

Keywords:

McKean-Vlasov control,
Mean-field neural networks,
Learning on Wasserstein space,
Dynamic programming,
Backward SDE.

Article Info.:

Volume: 3
Number: 2
Pages: 176 - 214
Date: /2024
doi.org/10.4208/jml.230106

Article History:

Received: 06/01/2023
Accepted: 17/03/2024

Communicated by:

Arnulf Jentzen

1 Introduction

This paper is concerned with the numerical resolution of McKean-Vlasov (MKV) control, also called mean-field control (MFC) problems over finite horizon. The dynamics of the controlled state process $X = (X_t)_t$ valued in \mathbb{R}^d is driven by the mean-field SDE (stochastic differential equation)

$$dX_t = b(X_t, \mathbb{P}_{X_t}, \alpha_t)dt + \sigma(X_t, \mathbb{P}_{X_t}, \alpha_t)dW_t, \quad 0 \leq t \leq T, \quad X_0 \sim \mu_0,$$

where W is a d -dimensional Brownian motion on a filtered probability space $(\Omega, \mathcal{F}, \mathbb{F} = (\mathcal{F}_t)_t, \mathbb{P})$, the initial distribution μ_0 of X_0 lies in $\mathcal{P}_2(\mathbb{R}^d)$, the Wasserstein space of square-integrable probability measures, $\alpha \in \mathcal{A}$ is a control process, i.e an \mathbb{F} -progressively measurable process valued in $A \subset \mathbb{R}^m$, and \mathbb{P}_{X_t} denotes the law of X_t , valued on $\mathcal{P}_2(\mathbb{R}^d)$, under standard assumptions on the coefficients b, σ defined on $\mathbb{R}^d \times \mathcal{P}_2(\mathbb{R}^d) \times A$, and valued respectively in \mathbb{R}^d and $\mathbb{R}^{d \times d}$. The objective is to minimize over controls $\alpha \in \mathcal{A}$, a cost functional of the form

$$J(\alpha) = \mathbb{E} \left[\int_0^T f(X_t, \mathbb{P}_{X_t}, \alpha_t)dt + g(X_T, \mathbb{P}_{X_T}) \right] \rightarrow v(\mu_0) = \inf_{\alpha \in \mathcal{A}} J(\alpha), \quad (1.1)$$

*Corresponding author. pham@lpsm.paris

†xavier.warin@edf.fr

where f is a running cost function on $\mathbb{R}^d \times \mathcal{P}_2(\mathbb{R}^d) \times A$, and g is a terminal cost function on $\mathbb{R}^d \times \mathcal{P}_2(\mathbb{R}^d)$.

The theory and applications of mean-field control problems that study models of large population of interacting agents controlled by a social planner, have generated a vast literature in the last decade, and we refer to the monographs [4, 6, 7] for a comprehensive treatment of this topic. As analytical solutions to MFC are rarely available, it is crucial to design efficient numerical schemes for solving such problem, and the main challenging issue is the infinite dimensional feature of MFC coming from the distribution law state variable.

Following the tremendous impact of machine learning methods for solving high-dimensional partial differential equations (PDEs) and control problems, see e.g. the survey papers [3, 16], and the link to the website deeppde.org, some recent works have proposed deep learning schemes for MFC, based on neural network approximations of the feedback control and/or the value function solution to the Hamilton-Jacobi-Bellman equation or backward stochastic differential equations (BSDEs). In these articles, the authors consider either approximate feedback controls by standard feedforward neural networks with input the time and the state variable X_t in \mathbb{R}^d by viewing the law of X_t as a deterministic function of time (see [9, 12, 14, 24, 26, 27]), or consider a particle approximation of the MFC for reducing the problem to a finite-dimensional problem that is numerically solved by means of symmetric neural networks, see [13]. However, the outputs obtained by these deep learning schemes only provide an approximation of the solution for a given initial distribution of the state process. Hence, for another distribution μ_0 of the initial state, these algorithms have to be run again.

In this paper, we aim to compute the minimal cost function $v(\mu_0)$ for any $\mu_0 \in \mathcal{P}_2(\mathbb{R}^d)$, and to find the optimal control, which can be searched w.l.o.g. in the class of feedback controls, i.e. of the form $\alpha_t = \alpha(t, X_t, \mathbb{P}_{X_t}), 0 \leq t \leq T$, for some measurable function α on $[0, T] \times \mathbb{R}^d \times \mathcal{P}_2(\mathbb{R}^d)$. In other words, our goal is to learn the value function and the optimal feedback control on the Wasserstein space. We shall rely on a new class of neural networks, introduced in our companion paper [25], called mean-field neural networks with input a probability measure in order to approximate mappings on the Wasserstein space. We then develop several numerical schemes based either on dynamic programming (DP) or stochastic maximum principle (SMP). We first propose, in the spirit of [17, 18] a global learning of the feedback control approximated by a mean-field neural network. In the DP approach, we then propose two algorithms inspired by [20]: The first one learns the control by policy iteration while the second one learns sequentially the control and the value function by value iteration. In the SMP approach, we exploit the backward SDE characterization of the solution, and propose five different algorithms in line with recent methods developed in the context of standard BSDE (see [11, 15, 21]) that we extend to MKV BSDE with various choices of global or local loss functions to be minimized in the training of mean-field neural networks. We then provide extensive numerical experiments on three examples: a mean-field systemic risk model, a min/max linear quadratic model, and the classical mean-variance problem. We compare and discuss the advantages and drawbacks of all our algorithms.

The rest of the paper is organized as follows. We recall in Section 2 some key results about the characterization of MKV control problems by DP or SMP approach, and introduce the class of mean-field neural networks. Section 3 presents three algorithms based on DP, while Section 4 develops five algorithms based on the BSDE representation of the solution to MKV. The performances of all our algorithms are illustrated via three examples in Section 5. Finally, we give in Section 6 some concluding remarks about the pros and cons of the different schemes.

2 Preliminaries

2.1 Characterization of McKean-Vlasov control

Solution to the MKV control problem (1.1) can be characterized by dynamic programming (DP) or maximum principle methods (see [6] for a detailed treatment of this topic). We recall the main results that will be used for designing our algorithms. In the DP approach, one considers the dynamic version of problem (1.1) by defining the decoupled value function V defined on $[0, T] \times \mathbb{R}^d \times \mathcal{P}_2(\mathbb{R}^d)$, which satisfies the backward recursion

$$V(t, X_t, \mathbb{P}_{X_t}) = \inf_{\alpha \in \mathcal{A}} \mathbb{E} \left[\int_t^{t+h} f(X_s, \mathbb{P}_{X_s}, \alpha_s) ds + V(t+h, X_{t+h}, \mathbb{P}_{X_{t+h}}) \middle| \mathcal{F}_t \right]$$

for any $t \in [0, T], h \in (0, T-t]$, and starting from the terminal condition $V(T, x, \mu) = g(x, \mu)$ for $(x, \mu) \in [0, T] \times \mathcal{P}_2(\mathbb{R}^d)$, so that $v(\mu_0) = \mathbb{E}[V(0, X_0, \mu_0)]$. By sending h to zero, we derive the master Bellman equation for the value function (see [6, Section 6.5.2])

$$\begin{aligned} & \partial_t V(t, x, \mu) + b(x, \mu, \hat{a}(x, \mu, \mathcal{U}(t, x, \mu), \partial_x \mathcal{U}(t, x, \mu))) \cdot \partial_x V(t, x, \mu) \\ & + \frac{1}{2} \sigma \sigma^\top(x, \mu, \hat{a}(x, \mu, \mathcal{U}(t, x, \mu), \partial_x \mathcal{U}(t, x, \mu))) \cdot \partial_{xx}^2 V(t, x, \mu) \\ & + \mathbb{E}_{\xi \sim \mu} \left[b(\xi, \mu, \hat{a}(\xi, \mu, \mathcal{U}(t, \xi, \mu), \partial_x \mathcal{U}(t, \xi, \mu))) \cdot \partial_\mu V(t, x, \mu)(\xi) \right. \\ & \quad \left. + \frac{1}{2} \sigma \sigma^\top(\xi, \mu, \hat{a}(\xi, \mu, \mathcal{U}(t, \xi, \mu), \partial_x \mathcal{U}(t, \xi, \mu))) \cdot \partial_{x'} \partial_\mu V(t, x, \mu)(\xi) \right] \\ & + f(x, \mu, \hat{a}(x, \mu, \mathcal{U}(t, x, \mu), \partial_x \mathcal{U}(t, x, \mu))) = 0 \end{aligned}$$

for $(t, x, \mu) \in [0, T] \times \mathbb{R}^d \times \mathcal{P}_2(\mathbb{R}^d)$. Here \cdot is the inner product in Euclidian spaces, $^\top$ is the transpose operator for a matrix, $x' \in \mathbb{R}^d \mapsto \partial_\mu V(t, x, \mu)(x') \in \mathbb{R}^d$ is the Lions derivative on $\mathcal{P}_2(\mathbb{R}^d)$ (see [6]), the notation $\mathbb{E}_{\xi \sim \mu}[\cdot]$ means that the expectation is taken with respect to the random variable ξ distributed according to the law μ ,

$$\begin{aligned} \mathcal{U}(t, x, \mu) &= \partial_x V(t, x, \mu) + \mathbb{E}_{\xi \sim \mu} [\partial_\mu V(t, \xi, \mu)(x)] = \partial_\mu v(t, \mu)(x), \\ v(t, \mu) &:= \mathbb{E}_{\xi \sim \mu} [V(t, \xi, \mu)], \end{aligned} \tag{2.1}$$

and it is assumed that for any $(x, \mu, p, M) \in \mathbb{R}^d \times \mathcal{P}_2(\mathbb{R}^d) \times \mathbb{R}^d \times \mathbb{R}^{d \times d}$, there exists a minimizer

$$\hat{a}(x, \mu, p, M) \in \operatorname{argmin}_{a \in A} H(x, \mu, p, M, a),$$

$$H(x, \mu, p, M, a) := b(x, \mu, a) \cdot p + \frac{1}{2} \sigma \sigma^\top(x, \mu, a) \cdot M + f(x, \mu, a),$$

which is Lipschitz in all its variables, so that we get an optimal feedback control given by

$$\alpha^*(t, x, \mu) = \hat{a}(x, \mu, \mathcal{U}(t, x, \mu), \partial_x \mathcal{U}(t, x, \mu)), \quad (t, x, \mu) \in [0, T] \times \mathbb{R}^d \times \mathcal{P}_2(\mathbb{R}^d). \quad (2.2)$$

In the case where the diffusion coefficient $\sigma(x, \mu)$ does not depend on the control variable a , and so $\hat{a}(x, \mu, p)$ does not depend on the variable M , we have a probabilistic characterization of the solution in terms of forward-backward SDE of MKV type: By setting

$$Y_t = V(t, X_t, \mathbb{P}_{X_t}), \quad Z_t = \sigma(X_t, \mathbb{P}_{X_t})^\top \partial_x V(t, X_t, \mathbb{P}_{X_t}), \quad 0 \leq t \leq T,$$

it follows from Itô's formula and master Bellman equation that (X, Y, Z) satisfies the forward-backward SDE

$$\begin{cases} dX_t = b(X_t, \mathbb{P}_{X_t}, \hat{a}(X_t, \mathbb{P}_{X_t}, P_t)) dt \\ \quad + \sigma(X_t, \mathbb{P}_{X_t}) dW_t, & 0 \leq t \leq T, \quad X_0 \sim \mu_0, \\ dY_t = -f(X_t, \mathbb{P}_{X_t}, \hat{a}(X_t, \mathbb{P}_{X_t}, P_t)) dt \\ \quad + Z_t \cdot dW_t, & 0 \leq t \leq T, \quad Y_T = g(X_T, \mathbb{P}_{X_T}), \end{cases} \quad (2.3)$$

where the pair

$$(P_t, M_t)_t = (\mathcal{U}(t, X_t, \mathbb{P}_{X_t}), \partial_x \mathcal{U}(t, X_t, \mathbb{P}_{X_t}) \sigma(X_t, \mathbb{P}_{X_t}))_t$$

of adjoint processes, valued in $\mathbb{R}^d \times \mathbb{R}^{d \times d}$, satisfies from the Pontryagin maximum principle the backward SDE

$$\begin{cases} dP_t = -\partial_x H(X_t, \mathbb{P}_{X_t}, P_t, M_t, \hat{a}(X_t, \mathbb{P}_{X_t}, P_t)) dt \\ \quad - \tilde{\mathbb{E}}[\partial_\mu H(\tilde{X}_t, \mathbb{P}_{\tilde{X}_t}, \tilde{P}_t, \tilde{M}_t, \hat{a}(\tilde{X}_t, \mathbb{P}_{\tilde{X}_t}, \tilde{P}_t))(X_t)] dt + M_t dW_t, & 0 \leq t \leq T, \\ P_T = \partial_x g(X_T, \mathbb{P}_{X_T}) + \tilde{\mathbb{E}}[\partial_\mu g(\tilde{X}_T, \mathbb{P}_{\tilde{X}_T})(X_T)], \end{cases} \quad (2.4)$$

where $(\tilde{X}, \tilde{P}, \tilde{M})$ are independent copies of (X, P, M) on $(\tilde{\Omega}, \tilde{\mathcal{F}}, \tilde{\mathbb{P}})$. Under the assumption that $(x, \mu) \in \mathbb{R}^d \times \mathcal{P}_2(\mathbb{R}^d) \mapsto g(x, \mu)$ is convex, $(x, \mu, a) \in \mathbb{R}^d \times \mathcal{P}_2(\mathbb{R}^d) \times A$ (with A convex set) $\mapsto H(x, \mu, p, M, a)$ is convex for any (p, M) , together with additional regularity conditions on the coefficients b, σ, f, g , it is known from [5] that the solution to the adjoint BSDE (2.4) yields an optimal control given by

$$\alpha_t^* = \alpha^*(t, X_t, \mathbb{P}_{X_t}) = \hat{a}(X_t, \mathbb{P}_{X_t}, P_t), \quad 0 \leq t \leq T.$$

We are then led to consider the generic form of MKV forward-backward $(X, \mathcal{Y}, \mathcal{Z})$

$$\begin{cases} dX_t = B(X_t, \mathbb{P}_{X_t}, \mathcal{Y}_t) dt + \sigma(X_t, \mathbb{P}_{X_t}) dW_t, & 0 \leq t \leq T, \quad X_0 \sim \mu_0, \\ d\mathcal{Y}_t = \tilde{\mathbb{E}}[\mathcal{H}(X_t, \mathbb{P}_{X_t}, \mathcal{Y}_t, \mathcal{Z}_t, \tilde{X}_t, \tilde{\mathcal{Y}}_t, \tilde{\mathcal{Z}}_t)] dt \\ \quad + \mathcal{Z}_t dW_t, & 0 \leq t \leq T, \quad \mathcal{Y}_T = G(X_T, \mathbb{P}_{X_T}). \end{cases} \quad (2.5)$$

2.2 Mean-field neural networks

The solution to MKV control problem, i.e. the value function and optimal feedback control, are mappings of the state process and its probability distribution. In order to approximate such mappings, we shall rely on mean-field neural networks introduced in our companion paper [25]. Those are mappings

$$\mathcal{N} : \mu \in \mathcal{P}_2(\mathbb{R}^d) \mapsto \mathcal{N}(\mu)(\cdot) : \mathbb{R}^d \rightarrow \mathbb{R}^p \quad \text{with quadratic growth condition,}$$

in one of the following forms:

- (i) **Bin density:** $\mathcal{N}(\mu)(x) = \Phi(x, \mathbf{p}^\mu)$, for $x \in \mathbb{R}^d, \mu \in \mathcal{D}_2(\mathbb{R}^d)$ the subset of probability measures μ in $\mathcal{P}_2(\mathbb{R}^d)$, which admit density functions \mathbf{p}^μ with respect to the Lebesgue measure λ_d on \mathbb{R}^d . Here, Φ is a standard feedforward neural network from $\mathbb{R}^d \times \mathbb{R}^K$ into \mathbb{R}^p , and $\mathbf{p}^\mu = (p_k^\mu)_{k \in \llbracket 1, K \rrbracket}$ is the bin weight of the discrete density approximation of \mathbf{p}^μ on a fixed bounded rectangular domain \mathcal{K} of \mathbb{R}^d divided into K bins: $\cup_{k=1}^K \text{Bin}(k) = \mathcal{K}$, of center x_k , with same area size $h = \lambda_d(\mathcal{K})/K$, hence given by (see Fig. 2.1 in the case of one dimensional Gaussian distribution for μ)

$$p_k^\mu = \frac{\mathbf{p}^\mu(x_k)}{\sum_{k=1}^K \mathbf{p}^\mu(x_k)h}, \quad k = 1, \dots, K.$$

- (ii) **Cylindrical:** $\mathcal{N}(\mu)(\cdot) = \Psi(\cdot, \langle \varphi, \mu \rangle)$, where Ψ is a feedforward network function (outer neural network) from $\mathbb{R}^d \times \mathbb{R}^q$ into \mathbb{R}^p , and φ is another feedforward network function (inner neural network) from \mathbb{R}^d into \mathbb{R}^q (called latent space). Here we denote $\langle \varphi, \mu \rangle := \int \varphi(x)\mu(dx)$.

The relevance of mean-field neural networks is theoretically justified in [25] by universal approximation theorems, and it has been also shown how they can be trained accurately from samples of probability measures $\mu = \mathcal{L}_D(\mathbf{p})$ with discrete density of bin weight $\mathbf{p} = (p^k)_{k \in \llbracket 1, K \rrbracket}$ drawn randomly on

$$\mathcal{D}_K = \left\{ \mathbf{p} = (p_k)_{k \in \llbracket 1, K \rrbracket} \in \mathbb{R}_+^K : \sum_{k=1}^K p_k h = 1 \right\},$$

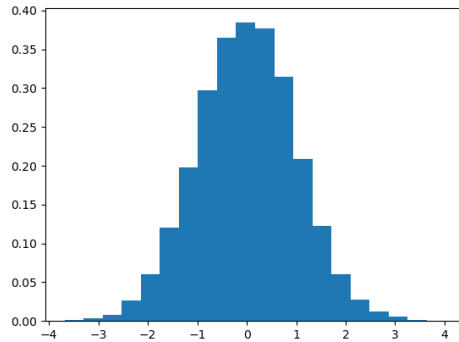


Figure 2.1: Bin approximation of a Gaussian distribution.

and simulations of random variables $X \sim \mu$ by inverse transform. Notice that for $\mu = \mathcal{L}_D(\mathbf{p})$, we have $\mathbf{p}^\mu = \mathbf{p}$, and so the bin density network at such μ is equal to $\mathcal{N}(\mu)(\cdot) = \Phi(\cdot, \mathbf{p})$. On the other hand, for any cylindrical function F of the measure of the form $F(\mu) = \Psi(\langle \varphi, \mu \rangle)$, we can compute it approximately from samples $X^{(n)}, n = 1, \dots, N$, of μ by

$$F(\mu) \simeq \Psi\left(\frac{1}{N} \sum_{n=1}^N \varphi(X^{(n)})\right).$$

This is the case in particular for cylindrical neural network.

As described in [25], exploring the space of probability measures is crucial for both neural networks. In both cases, we employ the bins method to generate samples of probability measures for training mean-field neural networks. The algorithm used to generate these measures is outlined in [25] and is currently limited to dimension one. Consequently, all numerical tests conducted in the article are confined to dimension one. However, it is possible to handle cases in dimension two by employing a different algorithm proposed in [30]. In all subsequent algorithms, the proper selection of the domain \mathcal{K} is crucial, particularly for the bins method. When the support of the distribution is unknown, an iterative procedure becomes necessary. Two algorithms can be implemented as follows:

1. First algorithm: (i) Initially, make an initial guess of the support. (ii) Once the resolution is obtained, verify that the generated distribution's support is primarily contained within \mathcal{K} , sufficiently far from its boundary. (iii) If the support is not mainly within \mathcal{K} , adapt the size and center of \mathcal{K} accordingly.
2. Second algorithm: (i) Use a very large \mathcal{K} during the first iteration to locate the domain of importance, employing a coarse resolution. (ii) In the subsequent calculation, reduce the size of \mathcal{K} to achieve an accurate resolution.

3 Dynamic programming-based algorithms

We consider a time discretization of the MKV control problem by fixing a time grid $\mathcal{T} = \{t_i = i\Delta t : i = 0, \dots, N_T\}$ with $\Delta t = T/N_T$, and introducing the corresponding mean-field Markov decision process: Minimize over feedback controls \mathbf{a} on $\mathcal{T} \times \mathbb{R}^d \times \mathcal{P}_2(\mathbb{R}^d)$ the cost functional

$$J_{N_T}(\mathbf{a}) = \mathbb{E} \left[\sum_{i=0}^{N_T-1} f(X_i, \mu_i, \mathbf{a}(t_i, X_i, \mu_i)) \Delta t + g(X_{N_T}, \mu_{N_T}) \right],$$

where

$$\begin{aligned} X_{i+1} &= X_i + b(X_i, \mu_i, \mathbf{a}(t_i, X_i, \mu_i)) \Delta t + \sigma(X_i, \mu_i, \mathbf{a}(t_i, X_i, \mu_i)) \Delta W_i, \\ &=: F_{\Delta t}(X_i, \mu_i, \mathbf{a}(t_i, X_i, \mu_i), \Delta W_i), \quad i = 0, \dots, N_T - 1, \quad X_0 \sim \mu_0 \end{aligned}$$

with $\Delta W_i := W_{t_{i+1}} - W_{t_i}$, and $\mu_i = \mathbb{P}_{X_i}$ denotes the law of X_i .

We present two classes of algorithms. The first one is learning the control by a single optimization but allows us to compute the solution of the problem (1.1) and therefore

the solution of the corresponding master Bellman equation only at time $t = 0$ for all distributions μ_0 . The second class with two other algorithms solves N_T local optimization problems, and allows us to compute the master equation at all dates for all distributions.

3.1 Global learning on control

In the spirit of the method introduced in [17,18], which does not actually rely on dynamic programming, we replace feedback controls by time-dependent mean-field neural networks $\mathcal{N}(t, \mu)(x)$ valued in $A \subset \mathbb{R}^m$ with input $t \in [0, T], \mu \in \mathcal{P}_2(\mathbb{R}^d)$, and $x \in \mathbb{R}^d$, and minimize over the parameters θ of this mean-field neural network $\mathcal{N} = \mathcal{N}_\theta$ the global cost function

$$J(\theta) = \mathbb{E} \left[\sum_{i=0}^{N_T-1} f(X_i, \mu_i, \mathcal{N}_\theta(t_i, \mu_i)(X_i)) \Delta t + g(X_{N_T}, \mu_{N_T}) \right]$$

with

$$X_{i+1} = F_{\Delta t}(X_i, \mu_i, \mathcal{N}_\theta(t_i, \mu_i)(X_i), \Delta W_i), \quad i = 0, \dots, N_T - 1, \quad X_0 \sim \mu_0.$$

In practice, for $i = 1, \dots, N_T, \mu_i$ has to be estimated/approximated from samples of X_i , and this is done as follows. We use a training batch of M probability measures $\mu_0^{(m)} = \mathcal{L}_D(\mathbf{p}^{(m)})$ in $\mathcal{D}_2(\mathbb{R}^d)$ from samples $\mathbf{p}^{(m)} = (p_k^{(m)})_{k \in \llbracket 1, K \rrbracket}, m = 1, \dots, M$, in \mathcal{D}_K . Then, for each m , we sample $X_0^{(m),(n)}, n = 1, \dots, N$, from $\mu_0^{(m)}$, and for $i = 0, \dots, N_T - 1, X_{i+1}^{(m),(n)}, n = 1, \dots, N$ are sampled as

$$X_{i+1}^{(m),(n)} = F_{\Delta t}(X_i^{(m),(n)}, \hat{\mu}_i^{(m)}, \mathcal{N}_\theta(t_i, \hat{\mu}_i^{(m)})(X_i^{(m),(n)}), \Delta W_i^{(m),(n)})$$

with $\hat{\mu}_i^{(m)} = \mathcal{L}_D(\hat{\mathbf{p}}_i^{(m)}), \hat{\mathbf{p}}_0^{(m)} = \mathbf{p}^{(m)}$, and $\hat{\mathbf{p}}_i^{(m)} = (\hat{p}_{i,k}^{(m)})_{k \in \llbracket 1, K \rrbracket}$ are the estimated density weights in \mathcal{D}_K of $X_i^{(m),(n)}, i = 1, \dots, N_T$ (truncated on \mathcal{K}), namely

$$\hat{p}_{i,k}^{(m)} = \frac{\#\{n \in \llbracket 1, N \rrbracket : \text{Proj}_{\mathcal{K}}(X_i^{(m),(n)}) \in \text{Bin}(k)\}}{Nh}, \quad k = 1, \dots, K,$$

where $\text{Proj}_{\mathcal{K}}(\cdot)$ is the projection on \mathcal{K} . The cost function is then approximated by

$$J_{M,N}(\theta) = \frac{1}{MN} \sum_{m=1}^M \left[\sum_{n=1}^N \sum_{i=0}^{N_T-1} f(X_i^{(m),(n)}, \hat{\mu}_i^{(m)}, \mathcal{N}_\theta(t_i, \hat{\mu}_i^{(m)})(X_i^{(m),(n)})) \Delta t + g(X_{N_T}^{(m),(n)}, \hat{\mu}_{N_T}^{(m)}) \right].$$

The pseudo-code using a gradient descent method is described in Algorithm 1. The global algorithms that directly minimize the objective function have demonstrated effectiveness in practice, even without having a theoretical convergence proof. The output of this global algorithm is an approximation of the optimal feedback control at initial

Algorithm 1 Global Learning on the Control.

Input data: A time-dependent mean-field neural network $\mathcal{N}_\theta(t, \mu)(x)$.

Initialization: Learning rate γ and parameters θ .

for each epoch do

Generate a batch of M initial distributions $\mu_0^{(m)}, m = 1, \dots, M$.

for $m = 1, \dots, M$ **do**

Generate Brownian increments $\Delta W_i^{(m),(n)}, i = 0, \dots, N_T - 1, n = 1, \dots, N$.

Compute sample trajectories $X_0^{(m),(n)}, X_i^{(m),(n)}, n = 1, \dots, N$, and estimate $\hat{\mu}_i^{(m)}, i = 1, \dots, N_T$.

Compute the batch cost $J_{M,N}(\theta)$ and its gradient $\nabla_\theta J_{M,N}(\theta)$.

Update $\theta \leftarrow \theta - \gamma \nabla_\theta J_{M,N}(\theta)$.

Return: The set of optimized parameters θ^* .

time $t_0 = 0$ by a mean-field neural network $\mathcal{N}_{\theta^*}(t_0, \cdot)$, and yields an approximation of the optimal control at other times $t_i, i = 1, \dots, N_T - 1$, by mean-field neural networks $\mathcal{N}_{\theta^*}(t_i, \mu_i)(X_i)$ along the law μ_i , and the state X_i explored during the learning algorithm. The value function can then be estimated at initial time t_0 by regression as follows: We approximate the initial value function by a mean-field neural network $\vartheta_\eta(\mu)(x)$ valued in \mathbb{R} , and minimize over the parameters η of this neural network the quadratic loss function

$$\mathbb{E} \left| \sum_{i=0}^{N_T-1} f(X_i, \mu_i, \mathcal{N}_{\theta^*}(t_i, \mu_i)(X_i)) \Delta t + g(X_{N_T}, \mu_{N_T}) - \vartheta_\eta(\mu_0)(X_0) \right|^2,$$

where

$$X_{i+1} = F_{\Delta t}(X_i, \mu_i, \mathcal{N}_{\theta^*}(t_i, \mu_i)(X_i), \Delta W_i), \quad i = 0, \dots, N_T - 1, \quad X_0 \sim \mu_0.$$

When using the global algorithm and the cylinder network, there is no need to estimate the support of the distribution. The parameter \mathcal{K} is solely used to generate probability distributions at time 0, and its selection is based on ensuring that the initial distribution of X_0 primarily concentrates its mass within \mathcal{K} . On the other hand, when employing the bin method, it is necessary to monitor the generated distribution and verify that its support is predominantly contained within \mathcal{K} . If this is not the case, the size of \mathcal{K} should be adjusted using the procedure suggested in Section 2.2.

3.2 Control learning by policy iteration

Our next algorithm is inspired by the method in [20], which is a combination of the global algorithm on control and dynamic programming. We replace at any time $t_i, i = 0, \dots, N_T - 1$, feedback controls by mean-field neural networks \mathcal{N}_{θ_i} with parameter θ_i , and proceed by backward induction for computing approximate optimal controls: For $i = N_T - 1, \dots, 0$, keep track of the approximate optimal feedback controls $\mathcal{N}_{\theta_j^*}, j = i + 1, \dots, N_T - 1$, and minimize over θ_i the cost function

$$J^i(\theta_i) = \mathbb{E} \left[f(X_i, \mu_i, \mathcal{N}_{\theta_i}(\mu_i)(X_i)) \Delta t + \sum_{j=i+1}^{N_T-1} f(X_j, \mu_j, \mathcal{N}_{\theta_j^*}(\mu_j)(X_j)) \Delta t + g(X_{N_T}, \mu_{N_T}) \right]$$

(with the convention that the above sum over j is empty when $i = N_T - 1$), where

$$\begin{cases} X_{i+1} = F_{\Delta t}(X_i, \mu_i, \mathcal{N}_{\theta_i}(\mu_i)(X_i), \Delta W_i), & X_i \sim \mu_i, \\ X_{j+1} = F_{\Delta t}(X_j, \mu_j, \mathcal{N}_{\theta_j^*}(\mu_j)(X_j), \Delta W_j), & j = i + 1, \dots, N_T - 1. \end{cases} \quad (3.1)$$

In the practical implementation, the cost function $J^i(\cdot)$ is approximately computed from a training of M probability measures $\mu_i^{(m)} = \mathcal{L}_D(\mathbf{p}_i^{(m)})$ in $\mathcal{D}_2(\mathbb{R}^d)$ with samples $\mathbf{p}_i^{(m)} = (p_{i,k}^{(m)})_{k \in \llbracket 1, K \rrbracket}$, $m = 1, \dots, M$, in \mathcal{D}_K . For each batch m , one then computes N samples $X_i^{(m),(n)} \sim \mu_i^{(m)}$, $X_j^{(m),(n)}$, $j = i + 1, \dots, N_T - 1$, $n = 1, \dots, N$, according to (3.1) with estimated probability measures $\hat{\mu}_j^{(m)} = \mathcal{L}_D(\hat{\mathbf{p}}_j^{(m)})$, as in Section 3.1, and thus approximate the local cost function by

$$J_{M,N}^i(\theta_i) = \frac{1}{MN} \sum_{m=1}^M \sum_{n=1}^N \left[f(X_i^{(m),(n)}, \mu_i^{(m)}, \mathcal{N}_{\theta_i}(\mu_i^{(m)})(X_i^{(m),(n)})) \Delta t + \sum_{j=i+1}^{N_T-1} f(X_j^{(m),(n)}, \hat{\mu}_j^{(m)}, \mathcal{N}_{\theta_j^*}(\hat{\mu}_j^{(m)})(X_j^{(m),(n)})) \Delta t + g(X_{N_T}^{(m),(n)}, \hat{\mu}_{N_T}^{(m)}) \right].$$

The pseudo-code is described in Algorithm 2. The output of this algorithm is an approximation of the optimal feedback control at any time t_i by a mean-field neural network $\mathcal{N}_{\theta_i^*}$, $i = 0, \dots, N_T - 1$. The value function can then be estimated at any time t_i by regression as follows: We approximate the value function at time t_i by a mean-field neural network $\vartheta_{\eta_i}(\mu)(x)$ valued in \mathbb{R} , and minimize over the parameters η_i of this neural network the quadratic loss function

$$\mathbb{E} \left| \sum_{j=i}^{N_T-1} f(X_j, \mu_j, \mathcal{N}_{\theta_j^*}(\mu_j)(X_j)) \Delta t + g(X_{N_T}, \mu_{N_T}) - \vartheta_{\eta_i}(\mu_i)(X_i) \right|^2, \quad (3.2)$$

where

$$X_{j+1} = F_{\Delta t}(X_j, \mu_j, \mathcal{N}_{\theta_j^*}(\mu_j)(X_j), \Delta W_j), \quad j = i, \dots, N_T - 1, \quad X_i \sim \mu_i.$$

In a backward algorithm, having a good estimate of the support of the distribution being tested is crucial at each time step i . This estimate helps in efficiently sampling the distribution in areas of interest. If the support is unknown, an iterative procedure, such as the one proposed in Section 2.2, needs to be implemented to gradually refine the estimation of the support.

Algorithm 2 Learning by Policy Iteration.**Input data:** Mean-field neural networks \mathcal{N}_{θ_i} .**for** $i = N_T - 1, \dots, 0$ **do** **Initialization:** Learning rate γ and parameters θ_i . **for** each epoch **do** Generate a batch of M distributions $\mu_i^{(m)}, m = 1, \dots, M$. **for** $m = 1, \dots, M$ **do** Generate Brownian increments $\Delta W_k^{(m),(n)}, k = i, \dots, N_T - 1, n = 1, \dots, N$. Compute sample trajectories $X_i^{(m),(n)}, X_j^{(m),(n)}, n = 1, \dots, N$, and estimate $\hat{\mu}_j^{(m)}$, $j = i + 1, \dots, N_T$. Compute the batch cost $J_{M,N}^i(\theta_i)$ and its gradient $\nabla_{\theta} J_{M,N}^i(\theta_i)$. Update $\theta_i \leftarrow \theta_i - \gamma \nabla_{\theta} J_{M,N}^i(\theta_i)$. $\theta_i^* = \theta_i$.**Return:** Optimized parameters $\theta_i^*, i = 0, \dots, N_T - 1$.**3.3 Control learning by value iteration**

The two previous algorithms provide low bias estimates of the learnt controls, but in general high-variance estimate due to this cumulated sum over the cost functions. Moreover, these algorithms are very memory demanding as, at each epoch, all the N trajectories for the M distributions have to be generated for the $\mathcal{O}(N_T)$ time values and stored. To circumvent this possible variance issue, we propose an alternate algorithm of actor-critic type, similarly as in [20] (called there hybrid algorithm), where the feedback control and value function are learnt sequentially. We are given a family of mean-field neural networks \mathcal{N}_{θ_i} and $\vartheta_{\eta_i}, i = 0, \dots, N_T - 1$, for the approximation of the feedback control (actor) and value function (critic). We proceed by backward induction as follows: Starting from $\vartheta_{N_T}^*(\mu)(x) = g(x, \mu)$, we minimize over θ_i , for $i = N_T - 1, \dots, 0$, the cost function

$$J^i(\theta_i) = \mathbb{E}[f(X_i, \mu_i, \mathcal{N}_{\theta_i}(\mu_i)(X_i))\Delta t + \vartheta_{i+1}^*(\mu_{i+1})(X_{i+1})],$$

where

$$X_{i+1} = F_{\Delta t}(X_i, \mu_i, \mathcal{N}_{\theta}(t_i, \mu_i)(X_i), \Delta W_i), \quad X_i \sim \mu_i, \quad (3.3)$$

update θ_i^* as the resulting optimal parameter, then minimize over η_i the quadratic loss function

$$L^i(\eta_i) = \mathbb{E}|f(X_i, \mu_i, \mathcal{N}_{\theta_i^*}(\mu_i)(X_i))\Delta t + \vartheta_{i+1}^*(\mu_{i+1})(X_{i+1}) - \vartheta_{\eta_i}(\mu_i)(X_i)|^2,$$

update η_i^* as the resulting optimal parameter, and set $\vartheta_i^* = \vartheta_{\eta_i^*}$. Again, in the practical implementation, we use a training of M probability measures $\mu_i^{(m)} = \mathcal{L}_D(\mathbf{p}_i^{(m)})$ in $\mathcal{D}_2(\mathbb{R}^d)$ with samples $\mathbf{p}_i^{(m)} = (p_{i,k}^{(m)})_{k \in \llbracket 1, K \rrbracket}, m = 1, \dots, M$, in \mathcal{D}_K . For each batch m , one then computes N samples $X_i^{(m),(n)} \sim \mu_i^{(m)}, X_{i+1}^{(m),(n)}$ according to (3.3) with estimated probability

measure $\hat{\mu}_{i+1}^{(m)} = \mathcal{L}_D(\hat{p}_{i+1}^{(m)})$, as in Section 3.1, and approximate the function J^i by

$$J_{M,N}^i(\theta_i) = \frac{1}{MN} \sum_{m=1}^M \sum_{n=1}^N \left[f\left(X_i^{(m),(n)}, \mu_i^{(m)}, \mathcal{N}_{\theta_i}(\mu_i^{(m)})(X_i^{(m),(n)})\right) \Delta t + \vartheta_{i+1}^*(\mu_{i+1}^{(m)})(X_{i+1}^{(m),(n)}) \right],$$

while similarly the second loss function L^i is approximated by

$$L_{M,N}^i(\eta_i) = \frac{1}{MN} \sum_{m=1}^M \sum_{n=1}^N \left| f\left(X_i^{(m),(n)}, \mu_i^{(m)}, \mathcal{N}_{\theta_i^*}(\mu_i^{(m)})(X_i^{(m),(n)})\right) \Delta t + \vartheta_{i+1}^*(\mu_{i+1}^{(m)})(X_{i+1}^{(m),(n)}) - \vartheta_{\eta_i}(\mu_i^{(m)})(X_i^{(m),(n)}) \right|^2.$$

The pseudo-code is described in Algorithm 3.

Algorithm 3 Actor/Critic Algorithm: Learning by Value Iteration.

Input data: Mean-field neural networks $\mathcal{N}_{\theta_i}, \vartheta_{\eta_i}, i = 0, \dots, N_T - 1$.

Initialization: $\vartheta_{N_T}^*(\mu)(x) = g(x, \mu)$.

for $i = N_T - 1, \dots, 0$ **do**

Initialization: Learning rates γ_C, γ_V and parameters θ_i, η_i .

for each epoch **do**

 Generate a batch of M distributions $\mu_i^{(m)}, m = 1, \dots, M$.

for each batch m **do**

 Generate Brownian increments $\Delta W_i^{(m),(n)}, n = 1, \dots, N$.

 Compute samples $X_i^{(m),(n)}, X_{i+1}^{(m),(n)}, n = 1, \dots, N$, and estimate $\hat{\mu}_{i+1}^{(m)}$.

 Compute the batch cost $J_{M,N}^i(\theta_i)$ and its gradient $\nabla_{\theta} J_{M,N}^i(\theta_i)$.

 Update $\theta_i \leftarrow \theta_i - \gamma \nabla_{\theta} J_{M,N}^i(\theta_i)$.

 Store optimized parameter θ_i^* .

for each epoch **do**

 Generate a batch of M distributions $\mu_i^{(m)}, m = 1, \dots, M$.

for each batch m **do**

 Generate Brownian increments $\Delta W_i^{(m),(n)}, n = 1, \dots, N$.

 Compute samples $X_i^{(m),(n)}, X_{i+1}^{(m),(n)}, n = 1, \dots, N$, and estimate $\hat{\mu}_{i+1}^{(m)}$.

 Compute the batch cost $L_{M,N}^i(\eta_i)$ and its gradient $\nabla_{\eta} L_{M,N}^i(\eta_i)$.

 Update $\eta_i \leftarrow \eta_i - \gamma \nabla_{\eta} L_{M,N}^i(\eta_i)$.

$\vartheta_i^* = \vartheta_{\eta_i^*}$.

Return: The optimized parameters $\theta_i^*, \eta_i^*, i = 0, \dots, N_T - 1$.

The output of this algorithm is an approximation of the optimal feedback control and value function at any time t_i by mean-field neural networks $\mathcal{N}_{\theta_i^*}$, and $\vartheta_{\eta_i^*}, i = 0, \dots, N_T - 1$.

Since the resolution is performed in a backward manner, when the support of the distribution is unknown, it becomes necessary to employ an iterative algorithm, as described in Section 2.2, to explore and estimate the distributions of interest.

4 Backward SDE-based algorithms

We start from the time discretization of the MKV forward-backward SDE (2.5) that characterizes the solution to the MKV control problem

$$\begin{cases} X_{i+1} = X_i + B(X_i, \mu_i, \mathcal{Y}_i)\Delta t + \sigma(X_i, \mu_i)\Delta W_i, & i = 0, \dots, N_T - 1, \quad X_0 \sim \mu_0, \\ \mathcal{Y}_{i+1} = \mathcal{Y}_i + \tilde{\mathbb{E}}[\mathcal{H}(X_i, \mu_i, \mathcal{Y}_i, \mathcal{Z}_i, \tilde{X}_i, \tilde{\mathcal{Y}}_i, \tilde{\mathcal{Z}}_i)]\Delta t \\ \quad + \mathcal{Z}_i\Delta W_i, & i = 0, \dots, N_T - 1, \quad \mathcal{Y}_{N_T} = G(X_{N_T}, \mu_{N_T}). \end{cases}$$

This system of equations corresponds to the resolution of the system of Eqs. (2.3), (2.4). Note that in fact, (X_t, P_t) is independent of Y_t . Then the resolution is achieved by calculating the optimal control solving (X_t, P_t) for $t \leq T$. The estimation of Y_t is achieved by using the optimal control with a simple forward simulation and by taking the expectation of Y_t in Eq. (2.3):

$$Y_t = \mathbb{E} \left[\int_t^T f(X_s, \mathbb{P}_{X_s}, \hat{a}(X_s, \mathbb{P}_{X_s}, P_s)) ds + g(X_T, \mathbb{P}_{X_T}) | \mathcal{F}_t \right].$$

4.1 Local algorithms

We adapt the deep backward scheme in [21] to our context. We are given a family of mean-field neural networks $\mathcal{Y}_{\theta_i}(\mu)(x), \mathcal{Z}_{\theta_i}(\mu)(x), i = 0, \dots, N_T - 1$ (by misuse of notation, we also denote by \mathcal{Y} and \mathcal{Z} the neural networks for the approximation of the pair component of the MKV BSDE), and proceed by backward induction as follows: Starting from $\mathcal{Y}_{N_T}^*(\mu)(x) = G(x, \mu)$, we minimize over θ_i , for $i = N_T - 1, \dots, 0$, the loss function

$$\begin{aligned} L^i(\theta_i) = & \mathbb{E} \left| \mathcal{Y}_{i+1}^*(\mu_{i+1})(X_{i+1}) - \mathcal{Y}_{\theta_i}(\mu_i)(X_i) - \mathcal{Z}_{\theta_i}(\mu_i)(X_i)\Delta W_i \right. \\ & \left. - \tilde{\mathbb{E}}[\mathcal{H}(X_i, \mu_i, \mathcal{Y}_{\theta_i}(\mu_i)(X_i), \mathcal{Z}_{\theta_i}(\mu_i)(X_i), \tilde{X}_i, \mathcal{Y}_{\theta_i}(\mu_i)(\tilde{X}_i), \mathcal{Z}_{\theta_i}(\mu_i)(\tilde{X}_i))] \Delta t \right|^2, \end{aligned}$$

where

$$X_{i+1} = X_i + B(X_i, \mu_i, \mathcal{Y}_{\theta_i}(\mu_i)(X_i))\Delta t + \sigma(X_i, \mu_i)\Delta W_i, \quad X_i \sim \mu_i, \quad (4.1)$$

update θ_i^* as the resulting optimal parameter, and set $\mathcal{Y}_i^* = \mathcal{Y}_{\theta_i^*}$. In the practical implementation, we use a training of M probability measures $\mu_i^{(m)} = \mathcal{L}_D(\mathbf{p}_i^{(m)})$ in $\mathcal{D}_2(\mathbb{R}^d)$ with samples $\mathbf{p}_i^{(m)} = (p_{i,k}^{(m)})_{k \in \llbracket 1, K \rrbracket}, m = 1, \dots, M$, in \mathcal{D}_K . For each batch m , one then computes N independent samples $X_i^{(m),(n)}, \tilde{X}_i^{(m),(n)} \sim \mu_i^{(m)}, n = 1, \dots, N, X_{i+1}^{(m),(n)}$ according to (4.1) with estimated probability measure $\hat{\mu}_{i+1}^{(m)}$ as in Section 3.1, and approximate the loss function by

$$\begin{aligned}
L_{M,N}^i(\theta_i) = \frac{1}{MN} \sum_{m=1}^M \sum_{n=1}^N & \left| \mathcal{Y}_{i+1}^*(\hat{\mu}_{i+1}^{(m)})(X_{i+1}^{(m),(n)}) - \mathcal{Y}_{\theta_i}(\mu_i^{(m)})(X_i^{(m),(n)}) \right. \\
& - \mathcal{Z}_{\theta_i}(\mu_i^{(m)})(X_i^{(m),(n)}) \Delta W_i \\
& - \frac{\Delta t}{N} \sum_{n'=1}^N \mathcal{H}(X_i^{(m),(n)}, \mu_i^{(m)}, \mathcal{Y}_{\theta_i}(\mu_i^{(m)})(X_i^{(m),(n)}), \\
& \quad \mathcal{Z}_{\theta_i}(\mu_i^{(m)})(X_i^{(m),(n)}), \tilde{X}_i^{(m),(n')}, \mathcal{Y}_{\theta_i}(\mu_i^{(m)})(\tilde{X}_i^{(m),(n')}), \\
& \quad \left. \mathcal{Z}_{\theta_i}(\mu_i^{(m)})(\tilde{X}_i^{(m),(n')}) \right|^2.
\end{aligned}$$

The pseudo-code is described in Algorithm 4. It is in the spirit of the actor/critic algorithm 3, but now \mathcal{Y} and \mathcal{Z} are learnt simultaneously.

Algorithm 4 Deep Backward Algorithm.

Input data: Mean-field neural networks $\mathcal{Y}_{\theta_i}, \mathcal{Z}_{\theta_i}$.

Initialization: $\mathcal{Y}_{N_T}^*(\mu)(x) = G(x, \mu)$.

for $i = N_T - 1, \dots, 0$ **do**

Initialization: Learning rate γ , and parameter θ_i .

for each epoch do

Generate a batch of M distributions $\mu_i^{(m)}, m = 1, \dots, M$.

for each batch m do

Generate Brownian increments $\Delta W_i^{(m),(n)}, n = 1, \dots, N$.

Compute samples $X_i^{(m),(n)}, \tilde{X}_i^{(m),(n)}, X_{i+1}^{(m),(n)}, n = 1, \dots, N$, and estimate $\hat{\mu}_{i+1}^{(m)}$.

Compute the batch loss $L_{M,N}^i(\theta_i)$ and its gradient $\nabla_{\theta} L_{M,N}^i(\theta_i)$.

Update $\theta_i \leftarrow \theta_i - \gamma \nabla_{\theta} L_{M,N}^i(\theta_i)$.

$\mathcal{Y}_i^* = \mathcal{Y}_{\theta_i^*}$.

Return: The set of optimized parameters $\theta_i^*, i = 0, \dots, N_T - 1$.

We also propose a multi-step version of the above algorithm following the idea in [15], and in the spirit of the policy iteration in Section 3.2. We proceed by backward induction for $i = N_T - 1, \dots, 0$, by keeping track of the approximate optimal mean-field neural networks $\mathcal{Y}_j^*, \mathcal{Z}_j^*, j = i + 1, \dots, N_T - 1$, and minimize over θ_i the loss function

$$\begin{aligned}
\tilde{L}^i(\theta_i) = \mathbb{E} & \left| G(X_{N_T}, \mu_{N_T}) - \sum_{j=i+1}^{N_T-1} \mathcal{Z}_j^*(\mu_j)(X_j) \Delta W_j - \mathcal{Z}_{\theta_i}(\mu_i)(X_i) \Delta W_i - \mathcal{Y}_{\theta_i}(\mu_i)(X_i) \right. \\
& - \sum_{j=i+1}^{N_T-1} \tilde{\mathbb{E}} [\mathcal{H}(X_j, \mu_j, \mathcal{Y}_j^*(\mu_j)(X_j), \mathcal{Z}_j^*(\mu_j)(X_j), \tilde{X}_j, \mathcal{Y}_j^*(\mu_j)(\tilde{X}_j), \mathcal{Z}_j^*(\mu_j)(\tilde{X}_j))] \Delta t \\
& \left. - \tilde{\mathbb{E}} [\mathcal{H}(X_i, \mu_i, \mathcal{Y}_{\theta_i}(\mu_i)(X_i), \mathcal{Z}_{\theta_i}(\mu_i)(X_i), \tilde{X}_i, \mathcal{Y}_{\theta_i}(\mu_i)(\tilde{X}_i), \mathcal{Z}_{\theta_i}(\mu_i)(\tilde{X}_i))] \Delta t \right|^2,
\end{aligned}$$

where

$$\begin{cases} X_{i+1} = X_i + B(X_i, \mu_i, \mathcal{Y}_{\theta_i}(\mu_i)(X_i))\Delta t + \sigma(X_i, \mu_i)\Delta W_i, & X_i \sim \mu_i, \\ X_{j+1} = X_j + B(X_j, \mu_j, \mathcal{Y}_j^*(\mu_j)(X_j))\Delta t + \sigma(X_j, \mu_j)\Delta W_j, & j = i + 1, \dots, N_T - 1. \end{cases} \quad (4.2)$$

In the practical implementation, we use a training of M probability measures $\mu_i^{(m)}, m = 1, \dots, M$, and for each batch m , one then computes N samples $X_i^{(m),(n)}, \tilde{X}_i^{(m),(n)} \sim \mu_i^{(m)}, X_j^{(m),(n)}, \tilde{X}_j^{(m),(n)}, j = i + 1, \dots, N_T - 1$, according to (4.2) with estimated probability measures $\hat{\mu}_j^{(m)} = \mathcal{L}_D(\hat{\rho}_j^{(m)})$, as in Section 3.1, and approximate the loss function by $\tilde{L}_{M,N}^i(\theta_i), i = 0, \dots, N_T - 1$.

The pseudo-code is described in Algorithm 5.

Algorithm 5 Deep Backward Multi-step Algorithm.

Input data: Mean-field neural networks $\mathcal{Y}_{\theta_i}, \mathcal{Z}_{\theta_i}$, and Brownian increments $\Delta W_i, i = 0, \dots, N_T - 1$.

for $i = N_T - 1, \dots, 0$ **do**

Initialization: Learning rate γ , and parameter θ_i .

for each epoch do

Generate a batch of M distributions $\mu_i^{(m)}, m = 1, \dots, M$.

for each batch m do

Generate Brownian increments $\Delta W_k^{(m),(n)}, \tilde{\Delta} W_k^{(m),(n)}, k = i, \dots, N_T - 1, n = 1, \dots, N$.

Compute samples $X_i^{(m),(n)}, \tilde{X}_i^{(m),(n)}, X_j^{(m),(n)}, \tilde{X}_j^{(m),(n)}, n = 1, \dots, N$, and estimate

$\hat{\mu}_j^{(m)}, j = i + 1, \dots, N_T$.

Compute the batch loss $\tilde{L}_{M,N}^i(\theta_i)$ and its gradient $\nabla_{\theta} \tilde{L}_{M,N}^i(\theta_i)$.

Update $\theta_i \leftarrow \theta_i - \gamma \nabla_{\theta} \tilde{L}_{M,N}^i(\theta_i)$.

$\mathcal{Y}_i^* = \mathcal{Y}_{\theta_i^*}, \mathcal{Z}_i^* = \mathcal{Z}_{\theta_i^*}$.

Return: The set $\mathcal{Y}_i^* = \mathcal{Y}_{\theta_i^*}, \mathcal{Z}_i^* = \mathcal{Z}_{\theta_i^*}, i = 0, \dots, N_T - 1$.

The output of these two Algorithms 4 and 5 yields in particular an approximation of the function \mathcal{U} in (2.1) by the mean-field neural network \mathcal{Y}_i^* at any time t_i , hence an approximation of the optimal feedback control defined in (2.2). We can then estimate the value function at any time by regression similarly as in (3.2). Alternately, by considering the value function in the BSDE as in (2.3), we can obtain an approximation of V via the mean-field neural network \mathcal{Y}_i^* at any time t_i . Once again, in a backward resolution process, if the support of the distribution is unknown, it is necessary to employ an iterative algorithm, as suggested in Section 2.2, to explore and identify the distributions of interest.

4.2 Global algorithms

In the spirit of the deep BSDE method in [19], we consider a mean-field neural network $\mathcal{U}_\theta(\mu)(x)$, and time dependent mean-field neural network $\mathcal{Z}_\theta(t, \mu)(x)$, for approximating respectively the initial value of the \mathcal{Y} component, and the \mathcal{Z} component at any time of the MKV BSDE. We then define by forward induction: Starting from $X_0 \sim \mu_0, \mathcal{Y}_0 = \mathcal{U}_\theta(\mu_0)(X_0)$ for $i = 0, \dots, N_T - 1$,

$$\begin{aligned} X_{i+1} &= X_i + B(X_i, \mu_i, \mathcal{Y}_i)\Delta t + \sigma(X_i, \mu_i)\Delta W_i, \\ \mathcal{Y}_{i+1} &= \mathcal{Y}_i + \tilde{\mathbb{E}}[\mathcal{H}(X_i, \mu_i, \mathcal{Y}_i, \mathcal{Z}_\theta(t_i, \mu_i)(X_i), \tilde{X}_i, \tilde{\mathcal{Y}}_i, \mathcal{Z}_\theta(t_i, \mu_i)(\tilde{X}_i))] \Delta t \\ &\quad + \mathcal{Z}_\theta(t_i, \mu_i)(X_i)\Delta W_i, \end{aligned} \quad (4.3)$$

and minimize over θ the global loss function

$$L(\theta) = \mathbb{E}|\mathcal{Y}_{N_T} - G(X_{N_T}, \mu_{N_T})|^2.$$

In practical implementation, we use a training sample of probability measures $\mu_0^{(m)}$, and then for each m, N samples $X_0^{(m),(n)} \sim \mu_0^{(m)}, \mathcal{Y}_0^{(m),(n)} = \mathcal{U}_0(\mu_0^{(m)})(X_0^{(m),(n)}), n = 1, \dots, N$, and for $i = 0, \dots, N_T - 1$,

$$\begin{aligned} X_{i+1}^{(m),(n)} &= X_i^{(m),(n)} + B(X_i^{(m),(n)}, \hat{\mu}_i^{(m)}, \mathcal{Y}_i^{(m),(n)})\Delta t + \sigma(X_i^{(m),(n)}, \hat{\mu}_i^{(m)})\Delta W_i, \\ \mathcal{Y}_{i+1}^{(m),(n)} &= \mathcal{Y}_i^{(m),(n)} + \frac{\Delta t}{N} \sum_{n'=1}^N \mathcal{H}(X_i^{(m),(n)}, \hat{\mu}_i^{(m)}, \mathcal{Y}_i^{(m),(n)}, \mathcal{Z}_\theta(t_i, \hat{\mu}_i^{(m)})(X_i^{(m),(n)}), \\ &\quad \tilde{X}_i^{(m),(n')}, \tilde{\mathcal{Y}}_i^{(m),(n')}, \mathcal{Z}_\theta(t_i, \hat{\mu}_i^{(m)})(\tilde{X}_i^{(m),(n')})) \\ &\quad + \mathcal{Z}_\theta(t_i, \hat{\mu}_i^{(m)})(X_i^{(m),(n)})\Delta W_i, \end{aligned}$$

where $\tilde{X}_i^{(m),(n)}, \tilde{\mathcal{Y}}_i^{(m),(n)}$ are independent copies of $X_i^{(m),(n)}, \mathcal{Y}_i^{(m),(n)}$, while $\hat{\mu}_0^{(m)} = \mu_0^{(m)}, \hat{\mu}_i^{(m)}, i = 1, \dots, N_T$, are estimated as in Section 3.1. The loss function is then approximated by

$$L_{M,N}(\theta) = \frac{1}{MN} \sum_{m=1}^M \sum_{n=1}^N \left| \mathcal{Y}_{N_T}^{(m),(n)} - G(X_{N_T}^{(m),(n)}, \hat{\mu}_{N_T}^{(m)}) \right|^2.$$

The pseudo-code is described in Algorithm 6. The output of this global deep BSDE algorithm is an approximation of the \mathcal{Y} component of the BSDE at initial time $t_0 = 0$ by a mean-field neural network \mathcal{U}_{θ^*} , and yields approximation of the \mathcal{Z} component at times $t_i, i = 0, \dots, N_T - 1$, by mean-field neural networks $\mathcal{Z}_{\theta^*}(t_i, \mu_i)(X_i)$ along the law μ_i , and state X_i explored during the learning algorithm. The value function can then be estimated at any time t_k by regression as follows: We approximate the value function at time t_k by a mean-field neural network $\vartheta_{\eta_k}(\mu)(x)$ valued in \mathbb{R} , and minimize over the parameters η_k of this neural network the quadratic loss function

$$\mathbb{E}|Y_k - \vartheta_{\eta_k}(\mu_k)(X_k)|^2, \quad (4.4)$$

Algorithm 6 Deep MKV BSDE.

Input data: A mean-field neural network $\mathcal{U}_\theta(\mu)(x)$, and a time-dependent mean-field neural network $\mathcal{Z}_\theta(t, \mu)(x)$.

Initialization: Learning rate γ and parameters θ .

for each epoch **do**

Generate a batch of M initial distributions $\mu_0^{(m)}, m = 1, \dots, M$.

for each batch m **do**

Generate Brownian increments $\Delta W_i^{(m),(n)}, i = 0, \dots, N_T - 1, n = 1, \dots, N$.

Compute sample trajectories $X_i^{(m),(n)}, \tilde{X}_i^{(m),(n)}, \mathcal{Y}_i^{(m),(n)}, \tilde{\mathcal{Y}}_i^{(m),(n)}, n = 1, \dots, N$, and

estimate $\hat{\mu}_i^{(m)}, i = 0, \dots, N_T$.

Compute the batch loss $L_{M,N}(\theta)$ and its gradient $\nabla_\theta L_{M,N}(\theta)$.

Update $\theta \leftarrow \theta - \gamma \nabla_\theta L_{M,N}(\theta)$.

Return: The set of optimized parameters θ^* .

where (X_k, Y_k) are generated by using Eq. (4.3) for $i = 0, \dots, k - 1$ (here Y_k is the first component of $\mathcal{Y}_k = (Y_k, P_k)$ in (2.3)-(2.4)), and μ_k is estimated from the distribution of the X_k .

In order to avoid the cost of solving Eq. (4.4) at each time step, we can propose two other global methods permitting to obtain directly the value function.

We first present a variation of the deep BSDE algorithm by considering two time-dependent mean-field neural networks $\mathcal{Y}_\theta(t, \mu)(x)$ and $\mathcal{Z}_\theta(t, \mu)(x)$, for approximating the pair solution of the MKV BSDE at any time. We then define by forward induction: Starting from $X_0 \sim \mu_0$, for $i = 0, \dots, N_T - 1$,

$$X_{i+1} = X_i + B(X_i, \mu_i, \mathcal{Y}_\theta(t_i, \mu_i)(X_i))\Delta t + \sigma(X_i, \mu_i)\Delta W_i, \tag{4.5}$$

and minimize over θ the global loss function as a sum of local loss functions

$$\tilde{L}(\theta) = \mathbb{E} \left[\sum_{i=1}^{N_T-1} \left| \mathcal{Y}_\theta(t_{i+1}, \mu_{i+1})(X_{i+1}) - \mathcal{Y}_\theta(t_i, \mu_i)(X_i) - \mathcal{Z}_\theta(t_i, \mu_i)(X_i)\Delta W_i - \tilde{\mathbb{E}} \left[\mathcal{H}(X_i, \mu_i, \mathcal{Y}_\theta(t_i, \mu_i)(X_i), \mathcal{Z}_\theta(t_i, \mu_i)(X_i), \tilde{X}_i, \mathcal{Y}_\theta(t_i, \mu_i)(\tilde{X}_i), \mathcal{Z}_\theta(t_i, \mu_i)(\tilde{X}_i)) \right] \Delta t \right|^2 \right]$$

with the convention that $\mathcal{Y}_\theta(t_{N_T}, \mu)(x) = G(x, \mu)$. In practical implementation, we use a training sample of probability measures $\mu_0^{(m)}$, and then for each $m = 1, \dots, M, N$ samples $X_0^{(m),(n)} \sim \mu_0^{(m)}, X_i^{(m),(n)}, \tilde{X}_i^{(m),(n)}, n = 1, \dots, N$, according to (4.5), and estimated probability measures $\hat{\mu}_i^{(m)}, i = 1, \dots, N_T$. The loss function is then approximated by $\tilde{L}_{M,N}(\theta)$.

The pseudo-code is described in Algorithm 7. Finally, we present a multi-step version of the deep MKV BSDE algorithm. We consider two time-dependent mean-field neural networks $\mathcal{Y}_\theta(t, \mu)(x)$ and $\mathcal{Z}_\theta(t, \mu)(x)$, for approximating the pair solution of the

Algorithm 7 Deep MKV BSDE Global/Local.

Input data: Two time-dependent mean-field neural network $\mathcal{Y}(t, \mu)(x), \mathcal{Z}_\theta(t, \mu)(x)$.

Initialization: Learning rate γ and parameters θ .

for each epoch do

Generate a batch of M initial distributions $\mu_0^{(m)}, m = 1, \dots, M$.

for each batch m do

Generate Brownian increments $\Delta W_i^{(m),(n)}, i = 0, \dots, N_T - 1, n = 1, \dots, N$.

Compute sample trajectories $X_i^{(m),(n)}, \tilde{X}_i^{(m),(n)}, n = 1, \dots, N$, and estimate $\hat{\mu}_i^{(m)}, i = 0, \dots, N_T$.

Compute the batch loss $\tilde{L}_{M,N}(\theta)$ and its gradient $\nabla_\theta \tilde{L}_{M,N}(\theta)$.

Update $\theta \leftarrow \theta - \gamma \nabla_\theta \tilde{L}_{M,N}(\theta)$.

Return: The set of optimized parameters θ^* .

MKV BSDE at any time, and define by forward induction: Starting from $X_0 \sim \mu_0$ for $i = 0, \dots, N_T - 1$,

$$X_{i+1} = X_i + B(X_i, \mu_i, \mathcal{Y}_\theta(t_i, \mu_i)(X_i))\Delta t + \sigma(X_i, \mu_i)\Delta W_i. \quad (4.6)$$

The global loss function to be minimized is of the form

$$L^{multi}(\theta) = \mathbb{E} \left[\sum_{i=0}^{N_T-1} \left| G(X_{N_T}, \mu_{N_T}) - \sum_{j=i}^{N_T-1} \mathcal{Z}_\theta(t_j, \mu_j)(X_j)\Delta W_j - \mathcal{Y}_\theta(t_i, \mu_i)(X_i) - \sum_{j=i}^{N_T-1} \tilde{\mathbb{E}} \left[\mathcal{H}(X_j, \mu_j, \mathcal{Y}_\theta(t_j, \mu_j)(X_j), \mathcal{Z}_\theta(t_j, \mu_j)(X_j), \tilde{X}_j, \mathcal{Y}_\theta(t_j, \mu_j)(\tilde{X}_j), \mathcal{Z}_\theta(t_j, \mu_j)(\tilde{X}_j)) \right] \Delta t \right|^2 \right].$$

Again, in practical implementation, we use a training sample of probability measures $\mu_0^{(m)}$, and then for each $m \in \{1, \dots, M\}$, N samples $X_0^{(m),(n)} \sim \mu_0^{(m)}, X_i^{(m),(n)}, \tilde{X}_i^{(m),(n)}, n = 1, \dots, N$, according to (4.6), and estimated probability measures $\hat{\mu}_i^{(m)}, i = 1, \dots, N_T$. The loss function is then approximated by $L_{M,N}^{multi}(\theta)$.

The pseudo-code is described in Algorithm 8. The output of Algorithms 7 and 8 is an approximation of the \mathcal{Y} component of the BSDE at initial time $t_0 = 0$ by a mean-field neural network $\mathcal{Y}_{\theta^*}(t_0, \cdot)(\cdot)$, and yields approximation of the \mathcal{Y} , at other times $t_i, i = 1, \dots, N_T - 1$, and \mathcal{Z} at times $t_i, i = 0, \dots, N_T - 1$, by mean-field neural networks $\mathcal{Y}_{\theta_i^*}(t_i, \mu_i)(X_i), \mathcal{Z}_{\theta_i^*}(t_i, \mu_i)(X_i)$ along the law μ_i , and state X_i explored during the learning algorithm.

In the case of global algorithms using the cylindrical network, there is no requirement to adapt the parameter \mathcal{K} . The need for adaptation methods, as proposed in Section 2.2, arises primarily when employing the bin method.

Algorithm 8 Deep multi-step MKV BSDE.

Input data: Two time-dependent mean-field neural networks $\mathcal{Y}(t, \mu)(x), \mathcal{Z}_\theta(t, \mu)(x)$.

Initialization: Learning rate γ and parameters θ .

for each epoch do

Generate a batch of M initial distributions $\mu_0^{(m)}, m = 1, \dots, M$.

for each batch m do

Generate Brownian increments $\Delta W_i^{(m),(n)}, i = 0, \dots, N_T - 1, n = 1, \dots, N$.

Compute sample trajectories $X_i^{(m),(n)}, \tilde{X}_i^{(m),(n)}, n = 1, \dots, N$, and estimate $\hat{\mu}_i^{(m)}, i = 0, \dots, N_T$.

Compute the batch loss $L_{M,N}^{multi}(\theta)$ and its gradient $\nabla_\theta L_{M,N}^{multi}(\theta)$.

Update $\theta \leftarrow \theta - \gamma \nabla_\theta L_{M,N}^{multi}(\theta)$.

Return: The set of optimized parameters θ^* .

5 Numerical examples

We shall illustrate the results of our different algorithms on three test cases. The two first examples are MKV control problems where the diffusion coefficient is constant, and the BSDE approach can be used. The third example is a classical mean variance problem, hence with control on the diffusion coefficient. We then test the three cases using the dynamic programming-based algorithms and for the two first cases using also the backward SDE-based algorithms.

For each problem, we will test the optimized solutions $v(\mu_0)$ found by using different initial distributions μ_0 and compare the result obtained to the analytical solution or the reference calculated by an other method. For all test cases, we keep the same parameters for the neural networks:

- For the bin method, we take 2 layers of 20 neurons.
- For the cylinder method, we take 2 layers of 20 neurons for the two networks.

For both methods we use the tanh activation function. At each iteration of the ADAM gradient method [23], we consider for each of the M tested distributions $N = 100000$ realizations of the process X . These parameters are chosen accordingly the results of [25]. We either take a batch size equal to $M = 5, M = 8, M = 10$ or $M = 20$, using between 30000 to 120000 gradient iterations: we have to adapt the batch size and the number of gradient iterations to be able to solve the problem on the graphic card GPU NVidia V100 32 Gb (except when specified due to memory limitation) and in order to obtain the result in less than 3 days. K in the tables below is the number of bins used, and $\Delta t = T/N_T$ is the time step.

5.1 The test examples

5.1.1 Systemic risk model

We consider a mean-field model of systemic risk introduced in [8]. This model was introduced in the context of mean field games but here we consider a cooperative version. The limit problem (when the number of banks is large) of the social planner (central bank) is formulated as follows. The log-monetary reserve of the representative bank is governed by the mean-reverting controlled McKean-Vlasov dynamics

$$dX_t = [\kappa(\mathbb{E}[X_t] - X_t) + \alpha_t] dt + \sigma dW_t, \quad X_0 \sim \mu_0,$$

where $\alpha = (\alpha_t)_t$ is the control rate of borrowing/lending to a central bank that aims to minimize the functional cost

$$J(\alpha) = \mathbb{E} \left[\int_0^T \tilde{f}(X_t, \mathbb{E}[X_t], \alpha_t) dt + \tilde{g}(X_T, \mathbb{E}[X_T]) \right] \rightarrow v(\mu_0) = \inf_{\alpha} J(\alpha), \quad (5.1)$$

where the running and terminal costs are given by

$$\tilde{f}(x, \bar{x}, a) = \frac{1}{2}a^2 - qa(\bar{x} - x) + \frac{\eta}{2}(\bar{x} - x)^2, \quad \tilde{g}(x, \bar{x}) = \frac{c}{2}(x - \bar{x})^2$$

for some positive constants $q, \eta, c > 0$, with $q^2 \leq \eta$. Notice that in this linear-quadratic example, the objective function is convex with respect to the control process, which ensures the convergence of the global algorithm.

The explicit solution of the linear-quadratic McKean-Vlasov control problem (5.1) is solved via the resolution of a Riccati equation (see [2]), and is analytically given by

$$v(t, \mu) = \int_{\mathbb{R}} V(t, x, \mu) \mu(dx) = Q_t \int_{\mathbb{R}} (x - \bar{\mu})^2 \mu(dx) + \sigma^2 \int_t^T Q_s ds, \quad (5.2)$$

where we set

$$\bar{\mu} := \mathbb{E}_{\xi \sim \mu}[\xi] = \int_{\mathbb{R}} x \mu(dx),$$

and

$$Q_t = -\frac{1}{2} \left[\kappa + q - \frac{\sqrt{\Delta} \sinh(\sqrt{\Delta}(T-t)) + (\kappa + q + c) \cosh(\sqrt{\Delta}(T-t))}{\sqrt{\Delta} \cosh(\sqrt{\Delta}(T-t)) + (\kappa + q + c) \sinh(\sqrt{\Delta}(T-t))} \right]$$

with $\sqrt{\Delta} = \sqrt{(\kappa + q)^2 + \eta - q^2}$, and

$$\int_t^T Q_s ds = \frac{1}{2} \ln \left[\cosh(\sqrt{\Delta}(T-t)) + \frac{\kappa + q + c}{\sqrt{\Delta}} \sinh(\sqrt{\Delta}(T-t)) \right] - \frac{1}{2}(\kappa + q)(T-t).$$

In this example, the function \hat{a} that attains the infimum of the Hamiltonian function is $\hat{a}(x, \mu, p) = q(\bar{\mu} - x) - p$, the function in (2.1) is $\mathcal{U}(t, x, \mu) = 2Q_t(x - \bar{\mu})$, which yields

the optimal feedback control: $\alpha^*(t, x, \mu) = (q + 2Q_t)(\bar{\mu} - x)$. The BSDE (2.3)-(2.4) is then written as

$$\begin{cases} dX_t = [(\kappa + q)(\mathbb{E}[X_t] - X_t) - P_t]dt + \sigma dW_t, & X_0 \sim \mu_0, \\ dY_t = -\left[\frac{1}{2}(\eta - q^2)(\mathbb{E}[X_t] - X_t)^2 + \frac{1}{2}P_t^2\right]dt + Z_t dW_t, & Y_T = \frac{c}{2}(X_T - \mathbb{E}[X_T])^2, \\ dP_t = [-(\kappa + q)(\mathbb{E}[P_t] - P_t) + (\eta - q^2)(\mathbb{E}[X_t] - X_t)]dt + M_t dW_t, & P_T = -c(\mathbb{E}[X_T] - X_T). \end{cases}$$

For the numerical tests of the different methods, we take $\sigma = 1, \kappa = 0.6, q = 0.8, T = 0.2, C = 2, \eta = 2$. We solve the problem (5.1) using our various algorithms and compare the solution obtained at $t = 0$ with $v(0, \mu_0)$ given by (5.2) for different initial distributions μ_0 plotted on Fig. 5.1:

- Case 1. Gaussian with $\bar{\mu}_0 = 0, std(\mu_0) = 0.2$.
- Case 2. Gaussian with $\bar{\mu}_0 = 0.3, std(\mu_0) = 0.05$.
- Case 3. Gaussian with $\bar{\mu}_0 = 0., std(\mu_0) = 0.05$.
- Case 4. Mixture of two Gaussian random variables: $X_0 = P(-k + \theta Y) + (1 - P)(k + \theta \tilde{Y})$ with P a Bernouilli random variable with parameter $1/2, k = \sqrt{3}/10, \theta = 0.1, Y, \tilde{Y} \sim \mathcal{N}(0, 1)$.
- Case 5. Mixture of two Gaussian random variables $X_0 = P(-k + \theta Y) + (1 - P)(-k + \theta \tilde{Y})$ with P a Bernouilli random variable with parameter $1/2, k = 0.25, \theta = 0.1, Y, \tilde{Y} \sim \mathcal{N}(0, 1)$.

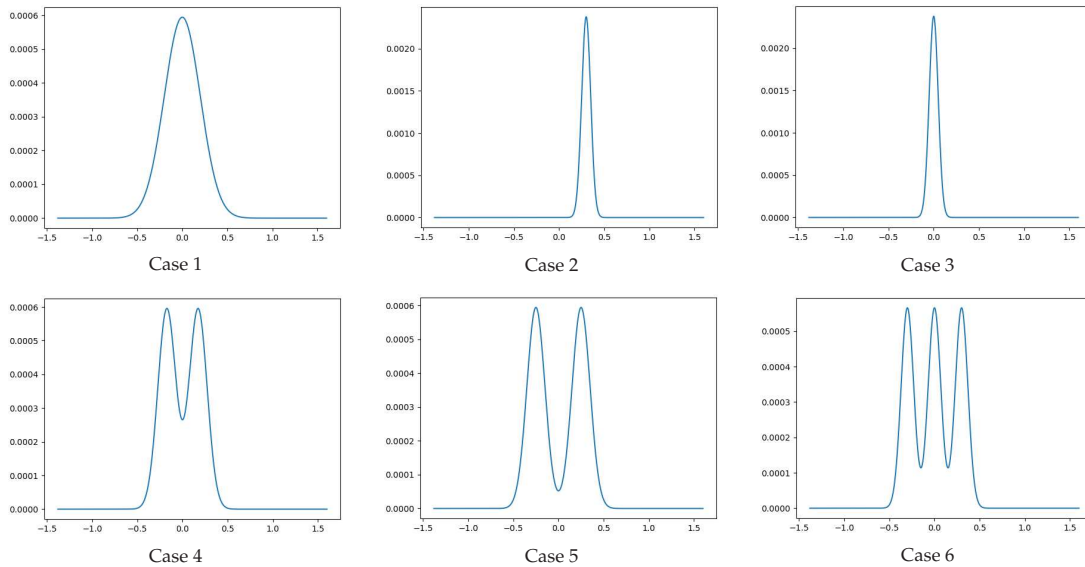


Figure 5.1: Distribution μ_0 tested on the systemic case.

Case 6. Mixture of 3 Gaussian random variables : $X_0 = [-1_{[3U]=0}k + 1_{[3U]=1}k] + \theta Y$ with $U \sim U(0, 1), k = 0.3, \theta = 0.07, \tilde{Y} \sim \mathcal{N}(0, 1)$.

Notice that Cases 1 and 4 have the same variance for μ_0 so that the values $v(0, \mu_0)$ of (5.2) should be the same. Similarly, values of Cases 2 and 3 are the same.

5.1.2 Min/max linear quadratic MKV control

We next consider a mean-field model in which the dynamics is linear, the running cost is quadratic in the position, the control and the expectation of the position, while the terminal cost gives incentive to be close to one of two targets. This type of model is inspired by the min-LQG problem of [28]. More precisely, we consider the following controlled McKean-Vlasov dynamics:

$$dX_t = [AX_t + \bar{A}\mathbb{E}[X_t] + B\alpha_t]dt + \sigma dW_t, \quad X_0 \sim \mu_0,$$

where $\alpha = (\alpha_t)_t$ is the control, and the agent aims to minimize the functional cost

$$J(\alpha) = \mathbb{E} \left[\int_0^T f(X_t, \mathbb{E}[X_t], \alpha_t)dt + g(X_T) \right] \rightarrow v(\mu_0) = \inf_{\alpha} J(\alpha),$$

where the running and terminal costs are given by

$$f(x, \bar{x}, a) = \frac{1}{2}(Qx^2 + \bar{Q}(x - S\bar{x})^2 + Ra^2), \quad g(x) = \min \{ |x - \zeta_1|^2, |x - \zeta_2|^2 \}$$

for some non-negative constants Q, \bar{Q}, S, R , and two real numbers ζ_1 and ζ_2 . Notice that g is not a convex function, and the solution to the MKV BSDE is not necessarily an optimal control.

In this example, the BSDE (2.3)-(2.4) is then written as

$$\begin{cases} dX_t = \left[AX_t + \bar{A}\mathbb{E}[X_t] - \frac{B^2}{R}P_t \right] dt + \sigma dW_t, & X_0 \sim \mu_0, \\ dY_t = -\frac{1}{2} \left[QX_t^2 + \bar{Q}(X_t - S\mathbb{E}[X_t])^2 + \frac{B^2}{R}P_t^2 \right] dt + Z_t dW_t, & Y_T = \min [|X_T - \zeta_1|^2, |X_T - \zeta_2|^2], \\ dP_t = - \left[AP_t + \bar{A}\mathbb{E}[P_t] + QX_t + \bar{Q}(X_t - \mathbb{E}[X_t]) + \bar{Q}(S - 1)^2\mathbb{E}[X_t] \right] dt + M_t dW_t, \\ P_T = 2 \left(X_T - \min(\zeta_1, \zeta_2) 1_{X_T \leq \frac{\zeta_1 + \zeta_2}{2}} - \max(\zeta_1, \zeta_2) 1_{X_T > \frac{\zeta_1 + \zeta_2}{2}} \right). \end{cases}$$

For the numerical tests, we take $A = 1, \bar{A} = 0.5, B = 1, Q = \bar{Q} = R = S = 1, \sigma = 0.5, \zeta_1 = 0.25, \zeta_2 = 1.75$. We first solve the problem (1.1) by the different algorithms and we can compare the solution $v(\mu_0)$ obtained for different distributions μ_0 to a reference calculated using [9] approach. Notice that [9] method needs to be run for each initial distribution tested. We use three different distributions μ_0 plotted on Fig. 5.2:

Case 1. Gaussian distribution $\bar{\mu}_0 = 1, std(\mu_0) = 0.2$. The reference values are 0.484 for $T = 0.2$, and 0.818 for $T = 0.5$.

Case 2. Mixture of two Gaussian random variables : $X_0 = P(\zeta_1 + \theta Y) + (1 - P)(\zeta_2 + \theta \tilde{Y})$ with P a Bernouilli random variable with parameter $1/2$, $\theta = 0.15$, $Y, \tilde{Y}, \tilde{Y} \sim \mathcal{N}(0, 1)$, with reference values 0.494 for $T = 0.2$, and 1.082 for $T = 0.5$.

Case 3. Mixture of three Gaussian random variables: $X_0 = [1_{[5U] < 2} \zeta_1 + 1_{[5U] > 3} \zeta_2 + 1_{2 \leq [5U] \leq 3} (\zeta_1 + \zeta_2)] + \theta Y$ with $U \sim U(0, 1)$, $\theta = 0.05$ with reference values 0.491 for $T = 0.2$, and 0.836 for $T = 0.5$.

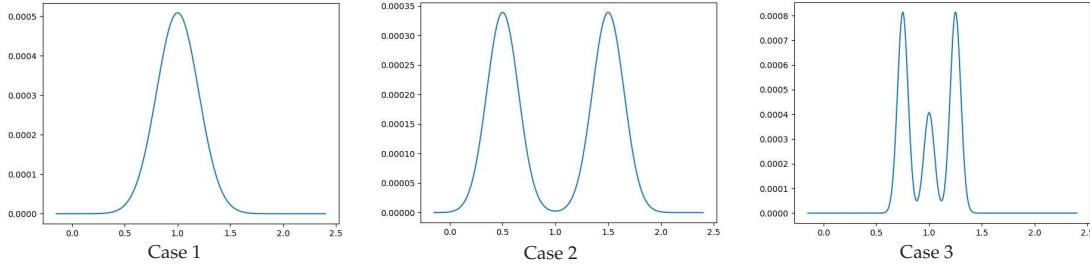


Figure 5.2: Distribution μ_0 tested on the min/max linear case.

5.1.3 Mean-variance problem

We consider the celebrated Markowitz portfolio selection problem where an investor can invest at any time t an amount α_t in a risky asset (assumed for simplicity to follow a Black-Scholes model with constant rate of return β and volatility $\nu > 0$), hence generating a wealth process $X = X^\alpha$ with dynamics

$$dX_t = \alpha_t \beta dt + \alpha_t \nu dW_t, \quad 0 \leq t \leq T, \quad X_0 \sim \mu_0.$$

The goal is then to minimize over portfolio control α the mean-variance criterion

$$J(\alpha) = \lambda \text{Var}(X_T^\alpha) - \mathbb{E}[X_T^\alpha],$$

where $\lambda > 0$ is a parameter related to the risk aversion of the investor.

We refer to [22] for the McKean-Vlasov approach to Markowitz mean-variance problems (in a more general context), and we recall that the solution to the Bellman equation is given by

$$\begin{aligned} V(t, x, \mu) &= \lambda e^{-R(T-t)} (x - \bar{\mu})^2 - x - \frac{1}{4\lambda} [e^{R(T-t)} - 1], \\ \mathcal{U}(t, x, \mu) &= 2\lambda e^{-R(T-t)} (x - \mathbb{E}_\mu[\zeta]) - 1, \end{aligned} \tag{5.3}$$

where we set $R := \beta^2 / \nu^2$. Moreover, the optimal feedback control is given by

$$\alpha^*(t, x, \mu) = -\frac{\beta}{\nu^2} \left(x - \bar{\mu} - \frac{e^{R(T-t)}}{2\lambda} \right).$$

Note that with this model, the BSDE approach cannot be used as the volatility is controlled.

We test our algorithms with the parameters $\beta = 0.1, \nu = 0.4, \lambda = 0.5$. We compare the solutions obtained at $t = 0$ to the analytical solution $v(\mu_0) = \mathbb{E}_{\xi \sim \mu_0}[V(0, \xi, \mu_0)]$ given by (5.3) for different initial distributions μ_0 plotted in Fig. 5.3, and explicitly given by:

Case 1. Gaussian distribution with $\bar{\mu}_0 = 0.1, \text{std}(\mu_0) = 0.2$.

Case 2. Gaussian distribution with $\bar{\mu}_0 = 0.2, \text{std}(\mu_0) = 0.025$.

Case 3. Gaussian distribution with $\bar{\mu}_0 = 0.3, \text{std}(\mu_0) = 0.025$.

Case 4. Mixture of two Gaussian random variables: $X_0 = P(-k + a + \theta Y) + (1 - P)(-k + a + \theta \bar{Y})$ with P a Bernoulli random variable with parameter $1/2, k = \sqrt{3}/10, a = 0.1, \theta = 0.1, Y, \bar{Y} \sim \mathcal{N}(0, 1)$.

Case 5. Mixture of two Gaussian random variables: $X_0 = P(-k + a + \theta Y) + (1 - P)(-k + a + \theta \bar{Y})$ with P a Bernoulli random variable with parameter $1/2, a = 0.05, k = 0.1, \theta = 0.1, Y, \bar{Y} \sim \mathcal{N}(0, 1)$.

Case 6. Mixture of 3 Gaussian random variables: $X_0 = a + [-1_{[5U] < 2} k + 1_{[5U] > 3} k] + \theta Y$ with $U \sim U(0, 1), a = 0.2, k = 0.3, \theta = 0.07, \bar{Y} \sim \mathcal{N}(0, 1)$.

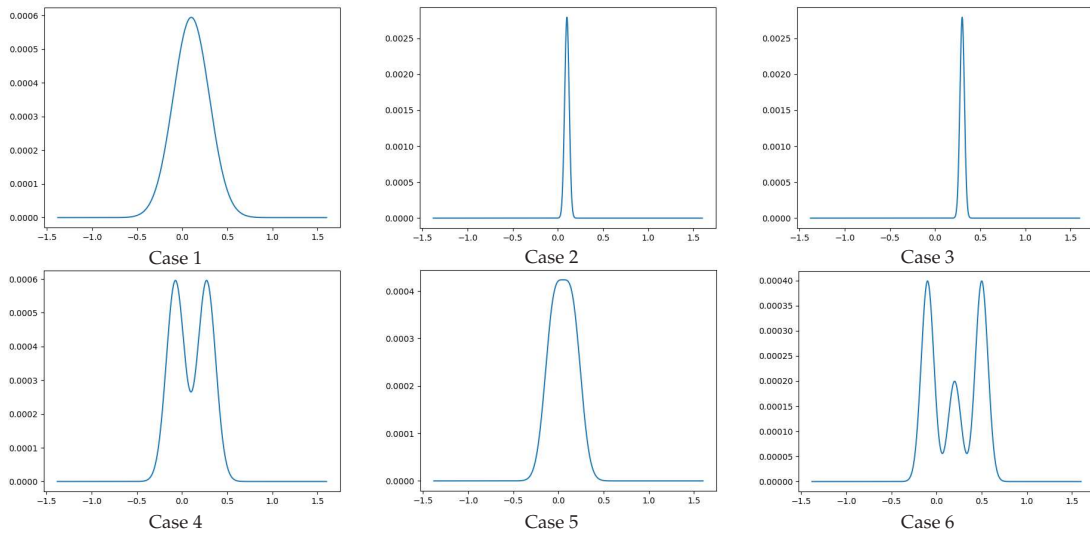


Figure 5.3: Distribution μ_0 tested on the mean variance case.

5.1.4 A toy example of non linear quadratic McKean-Vlasov control problem

We consider a one-dimensional controlled mean-field dynamics of the form

$$dX_t = [\beta(X_t, \mathbb{P}_{X_t}) + \alpha_t]dt + \sigma dW_t, \quad 0 \leq t \leq T, \quad X_0 \sim \mu_0$$

with a cost functional of the form

$$J(\alpha) = \mathbb{E} \left[\int_0^T \left(F(t, X_t, \mathbb{P}_{X_t}) + \frac{1}{2} |\alpha_t|^2 \right) dt + g(X_T, \mathbb{P}_{X_T}) \right] \rightarrow v(\mu_0) = \inf_{\alpha \in \mathcal{A}} J(\alpha),$$

where g is of the form

$$g(x, \mu) = \mathbb{E}_{\xi \sim \mu} [w(x - \xi)]$$

for some smooth C^2 even function w on \mathbb{R} , e.g. $w(x) = \cos(x)$, and F is a function to be chosen later.

In this case, the optimal feedback control valued in $A = \mathbb{R}$ is given by

$$\begin{aligned} \alpha^*(t, x, \mu) &= \hat{\alpha}(t, x, \mathcal{U}(t, x, \mu)) = -\mathcal{U}(t, x, \mu) = -\partial_\mu v(t, \mu)(x), \\ v(t, \mu) &= \mathbb{E}_{\xi \sim \mu} [V(t, \xi, \mu)], \end{aligned}$$

and V is solution to the master Bellman equation

$$\begin{aligned} &\partial_t V(t, x, \mu) + (\beta(x, \mu) - \mathcal{U}(t, x, \mu)) \partial_x V(t, x, \mu) + \frac{\sigma^2}{2} \partial_{xx}^2 V(t, x, \mu) \\ &+ \mathbb{E}_{\xi \sim \mu} \left[(\beta(\xi, \mu) - \mathcal{U}(t, \xi, \mu)) \partial_\mu V(t, x, \mu)(\xi) + \frac{\sigma^2}{2} \partial_{x'} \partial_\mu V(t, x, \mu)(\xi) \right] \\ &+ F(t, x, \mu) + \frac{1}{2} |\mathcal{U}(t, x, \mu)|^2 = 0 \end{aligned} \tag{5.4}$$

with the terminal condition $V(T, x, \mu) = g(x, \mu)$.

We look for a solution to the Master equation of the form

$$V(t, x, \mu) = e^{T-t} \mathbb{E}_{\xi \sim \mu} [w(x - \xi)].$$

For such function V , we have $\partial_t V(t, x, \mu) = -V$,

$$\begin{aligned} \partial_x V(t, x, \mu) &= e^{T-t} \mathbb{E}_{\xi \sim \mu} [w'(x - \xi)], \quad \partial_{xx}^2 V(t, x, \mu) = e^{T-t} \mathbb{E}_{\xi \sim \mu} [w''(x - \xi)] \\ \partial_\mu V(t, x, \mu)(\xi) &= -e^{T-t} w'(x - \xi), \quad \partial_{x'} \partial_\mu V(t, x, \mu)(\xi) = e^{T-t} w''(x - \xi), \end{aligned}$$

and

$$\begin{aligned} \mathcal{U}(t, x, \mu) &= e^{T-t} \mathbb{E}_{\xi \sim \mu} [w'(x - \xi) - w'(\xi - x)] \\ &= 2e^{T-t} \mathbb{E}_{\xi \sim \mu} [w'(x - \xi)] = 2\partial_x V(t, x, \mu) \end{aligned}$$

since w is even. By plugging these derivatives expressions of V into the left-hand side of (5.4), we then see that by choosing F equal to

$$\begin{aligned} F(t, x, \mu) &= e^{T-t} \mathbb{E}_{\xi \sim \mu} [(w - \sigma^2 w'')(x - \xi) + (\beta(\xi, \mu) - \beta(x, \mu)) w'(x - \xi)] \\ &\quad - 2e^{2(T-t)} \mathbb{E}_{(\xi, \xi') \sim \mu \otimes \mu} [w'(x - \xi) w'(\xi - \xi')], \end{aligned}$$

the function V satisfies the master Bellman equation.

For the choice of $w(x) = \cos(x)$, and using trigonometric relations, the function F is written as

$$\begin{aligned}
 F(t, x, \mu) = & \cos(x) \left[e^{T-t} \left((1 + \sigma^2) \mathbb{E}_{\xi \sim \mu} (\cos(\xi)) + \mathbb{E}_{\xi \sim \mu} (\sin(\xi) \beta(\xi, \mu)) \right. \right. \\
 & \left. \left. - \beta(x, \mu) \mathbb{E}_{\xi \sim \mu} (\sin(\xi)) \right) \right. \\
 & \left. - 2e^{2(T-t)} \left(\mathbb{E}_{\xi \sim \mu} (\sin(\xi) \cos(\xi)) \mathbb{E}_{\xi \sim \mu} (\sin(\xi)) \right. \right. \\
 & \left. \left. - \mathbb{E}_{\xi \sim \mu} (\sin^2(\xi)) \mathbb{E}_{\xi \sim \mu} (\cos(\xi)) \right) \right] \\
 & + \sin(x) \left[e^{T-t} \left((1 + \sigma^2) \mathbb{E}_{\xi \sim \mu} (\sin(\xi)) - \mathbb{E}_{\xi \sim \mu} (\beta(\xi, \mu) \cos(\xi)) \right. \right. \\
 & \left. \left. + \beta(x, \mu) \mathbb{E}_{\xi \sim \mu} (\cos(\xi)) \right) \right. \\
 & \left. - 2e^{2(T-t)} \left(\mathbb{E}_{\xi \sim \mu} (\sin(\xi) \cos(\xi)) \mathbb{E}_{\xi \sim \mu} (\cos(\xi)) \right. \right. \\
 & \left. \left. - \mathbb{E}_{\xi \sim \mu} (\cos^2(\xi)) \mathbb{E} (\sin(\xi)) \right) \right]
 \end{aligned}$$

Note that with this model, the BSDE approach cannot be used by lack of convexity.

For this example, we take $T = 0.4, \sigma = 1, \beta(x, \mu) = \bar{\mu} - x$, and we test the three distributions as given in the Min/max example (Section 5.1.2).

5.1.5 A two dimensional example

We consider a multi-dimensional extension of the linear quadratic (LQ) systemic risk model of Section 5.1.1 by supposing that on each dimension, the dynamic satisfies the same equation with independent Brownian motions, and that the cost functions are the sum over each component of the cost function in the univariate model. In this case, the value function is given by $V(t, x, \mu) = \sum_{i=1}^d V_1(t, x_i, \mu_i)$, for $t \in [0, T], x = (x_i)_{i \in [1, d]} \in \mathbb{R}^d, \mu_i$ is the i -th marginal law of $\mu \in \mathcal{P}_2(\mathbb{R}^d)$, and V_1 is the value function in the univariate model given by (5.2).

The parameters of the dynamic in each dimension are $\sigma = 0.5, \kappa = 0.6, q = 0.8, T = 0.2, c = 2, \eta = 2$. We solve the two dimensional version of the problem (5.1) by implementing our various algorithms, and compare the solution obtained at $t = 0$ with $v(0, \mu_0)$ for the initial distributions μ_0 with the same marginals $\mu_0^1 = \mu_0^2$ in the two dimensions plotted on Fig. 5.4:

Case 1. Gaussian marginals with $\bar{\mu}_0^1 = 0, \text{std}(\mu_0^1) = 0.2$.

Case 2. Mixture of two Gaussian random variables giving the marginal: $X_0^1 = P(-k_1 + \theta_1 Y) + (1 - P)(k_2 + \theta_2 \tilde{Y})$ with P a Bernoulli random variable with parameter $1/2, k_1 = 0, k_2 = 0.5, \theta_1 = \theta_2 = 0.15, Y, \tilde{Y} \sim \mathcal{N}(0, 1)$.

Case 3. Mixture of three Gaussian random variables giving the marginal: $X_0^1 = k_P + \theta_P Y$, where P is a random variable taking values $(1, 2, 3)$ with probability $(2/5, 2/5, 1/5), k_1 = -0.05, k_2 = 0, k_3 = 0.5, \theta_1 = \theta_2 = \theta_3 = 0.05, Y \sim \mathcal{N}(0, 1)$.

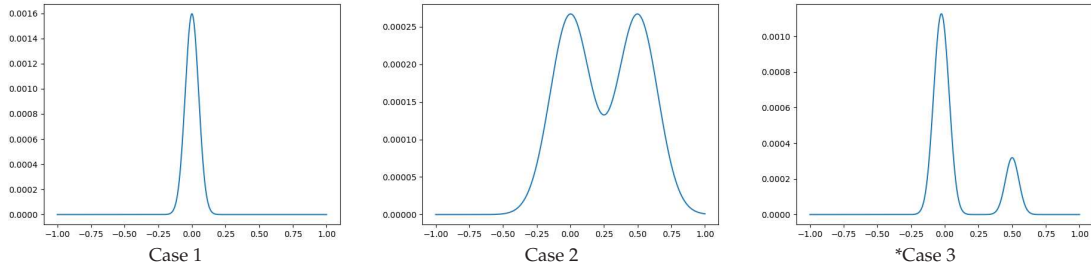


Figure 5.4: Marginals of distribution μ_0 tested on the two dimensional systemic model.

5.2 Results for the systemic risk model

5.2.1 Dynamic programming-based algorithms

We report the results for this model of Section 5.1.1 in Tables 5.1-5.3. It turns out that the results obtained by Algorithms 1 and 2 are excellent and very close. We can see that results with $K = 100$ or $K = 200$ bins for the bins method are very close. Notice that with the bins method, we have to limit the number K of bins due to memory issues for these two algorithms. We clearly see the effect of the convergence of the Euler scheme used to discretized the equations on the convergence rate. The Bins method and the Cylinder method provide very similar results but as the cost of Algorithm 1 is in $\mathcal{O}(N_T)$ while the cost of Algorithm 2 is in $\mathcal{O}(N_T(N_T - 1)/2)$, Algorithm 1 is clearly preferred. The computational time values presented in Table 5.2 provide confirmation that Algorithm 2 becomes impractical and less usable as the number of time steps increases.

The results obtained by the value iteration Algorithm 3 are still good but less accurate than the results obtained by the two other algorithms. The cylinder methods appears to be the best of the two methods. We notice a small degradation of the results as we refine the time step with the bins method. Notice that the memory used by this algorithm is small

Table 5.1: Global Algorithm 1 for systemic risk with $T = 0.2, \mathcal{K} = [-1.38, 1.62]$ using $M = 10$, and 60000 gradient iterations.

Method	K	$\Delta t = T/N_T$	Case 1		Case 2		Case 3		Training time (s)
			Calc	Anal	Calc	Anal	Calc	Anal	
Bins	100	0.02	0.1670	0.1642	0.1495	0.1446	0.1497	0.1446	8160
Bins	100	0.01	0.1651	0.1642	0.1472	0.1446	0.1470	0.1446	16200
Cylinder	500	0.02	0.1684	0.1642	0.1489	0.1446	0.1492	0.1446	8100
Cylinder	500	0.01	0.1665	0.1642	0.1469	0.1446	0.1467	0.1446	15240
Method	K	$\Delta t = T/N_T$	Case 4		Case 5		Case 6		
			Calc	Anal	Calc	Anal	Calc	Anal	
Bins	100	0.02	0.1675	0.1642	0.1824	0.1812	0.1792	0.1772	
Bins	100	0.01	0.1648	0.1642	0.1803	0.1812	0.1766	0.1772	
Cylinder	500	0.02	0.1684	0.1642	0.1848	0.1812	0.1817	0.1772	
Cylinder	500	0.01	0.1660	0.1642	0.1835	0.1812	0.1795	0.1772	

Table 5.2: Policy iteration Algorithm 2 for systemic risk with $T = 0.2, \mathcal{K} = [-1.38, 1.62]$ using $M = 10$, and 30000 gradient iterations.

Method	K	$\Delta t = T/N_T$	Case 1		Case 2		Case 3		Training time (s)
			Calc	Anal	Calc	Anal	Calc	Anal	
Bins	100	0.02	0.1692	0.1642	0.1493	0.1446	0.1495	0.1446	20000
Bins	100	0.01	0.1673	0.1642	0.1478	0.1446	0.1470	0.1446	73300
Bins	200	0.01	0.1674	0.1642	0.1480	0.1446	0.1477	0.1446	108800
Cylinder	500	0.02	0.1688	0.1642	0.1492	0.1446	0.1490	0.1446	46600
Cylinder	500	0.01	0.1662	0.1642	0.1468	0.1446	0.1471	0.1446	160200
Method	K	$\Delta t = T/N_T$	Case 4		Case 5		Case 6		Training time (s)
			Calc	Anal	Calc	Anal	Calc	Anal	
Bins	100	0.02	0.1691	0.1642	0.1862	0.1821	0.1822	0.1772	
Bins	100	0.01	0.1670	0.1642	0.1836	0.1812	0.1799	0.1772	
Bins	200	0.01	0.1675	0.1642	0.1844	0.1812	0.1800	0.1772	
Cylinder	500	0.02	0.1684	0.1642	0.1862	0.1812	0.1819	0.1772	
Cylinder	500	0.01	0.1663	0.1642	0.1836	0.1812	0.1794	0.1772	

Table 5.3: Value iteration Algorithm 3 for systemic risk with $T = 0.2, \mathcal{K} = [-1.38, 1.62]$, $M = 10$, and 30000 gradient iterations.

Method	K	$\Delta t = T/N_T$	Case 1		Case 2		Case 3		Training time (s)
			Calc	Anal	Calc	Anal	Calc	Anal	
Bins	500	0.02	0.1620	0.1642	0.1373	0.1446	0.1698	0.1446	36000
Bins	500	0.01	0.1873	0.1642	0.1673	0.1446	0.1841	0.1446	72000
Cylinder	500	0.02	0.1722	0.1642	0.1540	0.1446	0.1554	0.1446	9300
Cylinder	500	0.01	0.1704	0.1642	0.1520	0.1446	0.1571	0.1446	18600
Method	K	$\Delta t = T/N_T$	Case 4		Case 5		Case 6		Training time (s)
			Calc	Anal	Calc	Anal	Calc	Anal	
Bins	500	0.02	0.1630	0.1642	0.1809	0.1812	0.1755	0.1772	
Bins	500	0.01	0.1880	0.1642	0.2037	0.1812	0.1991	0.1772	
Cylinder	500	0.02	0.1722	0.1642	0.1880	0.1812	0.1843	0.1772	
Cylinder	500	0.01	0.1704	0.1642	0.1864	0.1812	0.1827	0.1772	

compared to the two other algorithms and it permits to take a high number K of bins for the bins method (even if it is not necessary on this case).

In Table 5.4, we provide sensitivity results for Cases 1, 4, and 6 using Algorithms 1 and 3 with different methods (bins and cylinder) and parameter settings ($T/N_T = 0.02, K = 100$ for bins, and $K = 500$ for cylinder). The results are based on 10 runs, and we report the average value obtained along with the standard deviation. It is observed that the results obtained using different methods and algorithms are generally very similar, except for Algorithm 3 with the cylinder network. This particular algorithm shows a higher standard deviation, which is a known characteristic of this approach, as mentioned in [21]. Furthermore, all the results seem to converge to the discrete-time solution of the problem, indicating the reliability and accuracy of the algorithms employed.

Table 5.4: Some sensitivity results using 10 runs.

Alg	Method	Case 1			Case 4			Case 6		
		Calc	Std	Anal	Calc	Std	Anal	Calc	Std	Anal
1	Bins	0.1691	0.0007	0.1642	0.1692	0.0010	0.1642	0.1825	0.0007	0.1772
1	Cyl	0.1687	0.0002	0.1642	0.1686	0.0002	0.1642	0.1816	0.0002	0.1772
2	Bins	0.1694	0.0003	0.1642	0.1694	0.0002	0.1642	0.1821	0.0002	0.1772
2	Cyl	0.1687	0.0002	0.1642	0.1687	0.0001	0.1642	0.1815	0.0002	0.1772
3	Bins	0.1692	0.0092	0.1642	0.1692	0.0093	0.1642	0.1822	0.0089	0.1772
3	Cyl	0.1807	0.0066	0.1642	0.1807	0.0066	0.1642	0.1931	0.0064	0.1772

5.2.2 Results for Backward SDE-based algorithms

Results for the systemic example of Section 5.1.1 are given in Tables 5.5-5.9. All the proposed methods converge very accurately to the solution. As previously seen in the results of the dynamic programming-based algorithms, the number of bins does not need to be large for the bins network. For this test case, the numerical values obtained does not permit to select the best algorithm. As Algorithm 5 is by far the most costly, it should not be the preferred choice. It is difficult to compare the other algorithms in terms of computing time, but all global algorithms have roughly the same cost in terms of time and the local deep backward Algorithm 4 is certainly more costly as we have to achieve an optimization per time step. This drawback due to the number of optimizations is reduced by transfer learning, namely the fact that at each time step the problem is much more smaller to solve as we can initialize the parameters of networks at a given time step by the parameters of networks of the preceding time step. On the other hand, we point out that all the global algorithms are too far memory consuming to be able to compete with the local deep backward Algorithm 4 which seems to be globally the best choice.

Table 5.5: Local deep backward BSDE Algorithm 4, $T = 0.2, \mathcal{K} = [-1.38, 1.62]$, using $M = 10$, and 30000 gradient iterations.

Method	K	$\Delta t = T/N_T$	Case 1		Case 2		Case 3		Training time (s)
			Calc	Anal	Calc	Anal	Calc	Anal	
Bins	200	0.02	0.1709	0.1642	0.1513	0.1446	0.1516	0.1446	30000
Bins	200	0.01	0.1672	0.1642	0.1479	0.1446	0.1475	0.1446	111000
Cylinder	500	0.02	0.1688	0.1642	0.1494	0.1446	0.1489	0.1446	20500
Cylinder	500	0.01	0.1663	0.1642	0.1469	0.1446	0.1472	0.1446	68800
Method	K	$\Delta t = T/N_T$	Case 4		Case 5		Case 6		
			Calc	Anal	Calc	Anal	Calc	Anal	
Bins	200	0.02	0.1711	0.1642	0.1881	0.1812	0.1838	0.1772	
Bins	200	0.01	0.1671	0.1642	0.1845	0.1812	0.1800	0.1772	
Cylinder	500	0.02	0.1686	0.1642	0.1855	0.1812	0.1817	0.1772	
Cylinder	500	0.01	0.1662	0.1642	0.1834	0.1812	0.1787	0.1772	

Table 5.6: Deep backward multi-step Algorithm 5, $T = 0.2, \mathcal{K} = [-1.38, 1.62]$ using $M = 10$, and 30000 gradient iterations.

Method	K	$\Delta t = T/N_T$	Case 1		Case 2		Case 3		Training time (s)
			Calc	Anal	Calc	Anal	Calc	Anal	
Bins	100	0.02	0.1690	0.1642	0.1493	0.1446	0.1497	0.1446	7500
Bins	200	0.02	0.1689	0.1642	0.1494	0.1446	0.1494	0.1446	11700
Bins	100	0.01	0.1664	0.1642	0.1474	0.1446	0.1470	0.1446	15000
Bins	200	0.01	0.1664	0.1642	0.1472	0.1446	0.1471	0.1446	23400
Cylinder	500	0.02	0.1683	0.1642	0.1491	0.1446	0.1492	0.1446	10500
Cylinder	500	0.01	0.1664	0.1642	0.1472	0.1446	0.1466	0.1446	21000
Method	K	$\Delta t = T/N_T$	Case 4		Case 5		Case 6		Training time (s)
			Calc	Anal	Calc	Anal	Calc	Anal	
Bins	100	0.02	0.1690	0.1642	0.1860	0.1812	0.1816	0.1772	
Bins	200	0.02	0.1687	0.1642	0.1853	0.1812	0.1818	0.1772	
Bins	200	0.01	0.1669	0.1642	0.1838	0.1812	0.1801	0.1772	
Bins	200	0.01	0.1666	0.1642	0.1835	0.1812	0.1796	0.1772	
Cylinder	500	0.02	0.1683	0.1642	0.1858	0.1812	0.1816	0.1772	
Cylinder	500	0.01	0.1665	0.1642	0.1837	0.1812	0.1795	0.1772	

Table 5.7: Global deep MKV BSDE Algorithm 6, $T = 0.2, \mathcal{K} = [-1.38, 1.6]$ using $M = 10$, and 30000 gradient iterations.

Method	K	$\Delta t = T/N_T$	Case 1		Case 2		Case 3		Training time (s)
			Calc	Anal	Calc	Anal	Calc	Anal	
Bins	100	0.02	0.1691	0.1642	0.1496	0.1446	0.1498	0.1446	4500
Bins	200	0.02	0.1691	0.1642	0.1495	0.1446	0.1497	0.1446	6400
Bins	200	0.01	0.1663	0.1642	0.1468	0.1446	0.1471	0.1446	12400
Cylinder	500	0.02	0.1686	0.1642	0.1491	0.1446	0.1492	0.1446	4530
Cylinder	500	0.01	0.1665	0.1642	0.1466	0.1446	0.1466	0.1446	8400
Method	K	$\Delta t = T/N_T$	Case 4		Case 5		Case 6		Training time (s)
			Calc	Anal	Calc	Anal	Calc	Anal	
Bins	100	0.02	0.1692	0.1642	0.1858	0.1812	0.1815	0.1772	
Bins	200	0.02	0.1694	0.1642	0.1863	0.1812	0.1824	0.1772	
Bins	200	0.01	0.1668	0.1642	0.1838	0.1812	0.1793	0.1772	
Cylinder	500	0.02	0.1686	0.1642	0.1857	0.1812	0.1816	0.1772	
Cylinder	500	0.01	0.1667	0.1642	0.1836	0.1812	0.1795	0.1772	

In Table 5.10, sensitivity results are presented based on ten runs using the same hyperparameters as in the dynamic programming approach. Similar to the previous table, all the algorithms demonstrate consistent results with low standard deviations, except for Algorithm 8 with bins, which exhibits a higher standard deviation. Additionally, it is ob-

Table 5.8: Global/local deep MKV BSDE Algorithm 7, $T = 0.2, \mathcal{K} = [-1.38, 1.62]$ using $M = 10, 30000$ gradient iterations.

Method	K	$\Delta t = T/N_T$	Case 1		Case 2		Case 3		Training time (s)
			Calc	Anal	Calc	Anal	Calc	Anal	
Bins	200	0.02	0.1753	0.1642	0.1545	0.1446	0.1706	0.1446	6400
Bins	200	0.01	0.1670	0.1642	0.1483	0.1446	0.1597	0.1446	12300
Cylinder	500	0.02	0.1684	0.1642	0.1496	0.1446	0.1491	0.1446	4200
Cylinder	500	0.01	0.1667	0.1642	0.1469	0.1446	0.1468	0.1446	8100
Method	K	$\Delta t = T/N_T$	Case 4		Case 5		Case 6		
			Calc	Anal	Calc	Anal	Calc	Anal	
Bins	200	0.02	0.1758	0.1642	0.1931	0.1812	0.1887	0.1772	
Bins	200	0.01	0.1661	0.1642	0.1841	0.1812	0.1797	0.1772	
Cylinder	500	0.02	0.1687	0.1642	0.1856	0.1812	0.1816	0.1772	
Cylinder	500	0.01	0.1664	0.1642	0.1836	0.1812	0.1793	0.1772	

Table 5.9: Global deep multi-step MKV BSDE Algorithm 8, $T = 0.2, \mathcal{K} = [-1.38, 1.62]$ using $M = 10, 30000$ gradient iterations.

Method	K	$\Delta t = T/N_T$	Case 1		Case 2		Case 3		Training time (s)
			Calc	Anal	Calc	Anal	Calc	Anal	
Bins	200	0.02	0.1689	0.1642	0.1507	0.1446	0.1528	0.1446	6300
Bins	200	0.01	0.1664	0.1642	0.1470	0.1446	0.1469	0.1446	12400
Cylinder	500	0.02	0.1685	0.1642	0.1489	0.1446	0.1494	0.1446	4300
Cylinder	500	0.01	0.1658	0.1642	0.1470	0.1446	0.1468	0.1446	83200
Method	K	$\Delta t = T/N_T$	Case 4		Case 5		Case 6		
			Calc	Anal	Calc	Anal	Calc	Anal	
Bins	200	0.02	0.1692	0.1642	0.1868	0.1812	0.1821	0.1772	
Bins	200	0.01	0.1666	0.1642	0.1829	0.1812	0.1796	0.1772	
Cylinder	500	0.02	0.1687	0.1642	0.1855	0.1812	0.1817	0.1772	
Cylinder	500	0.01	0.1661	0.1642	0.1834	0.1812	0.1795	0.1772	

Table 5.10: Some sensitivity results using 10 runs.

Alg	Method	Case 1			Case 4			Case 6		
		Calc	Std	Anal	Calc	Std	Anal	Calc	Std	Anal
4	Bins	0.1691	0.0002	0.1642	0.1691	0.0003	0.1642	0.1821	0.0003	0.1772
4	Cyl	0.1688	0.0001	0.1642	0.1687	0.0002	0.1642	0.1817	0.0002	0.1772
5	Bins	0.1691	0.0002	0.1642	0.1691	0.0002	0.1642	0.1820	0.0002	0.1772
5	Cyl	0.1686	0.0003	0.1642	0.1689	0.0003	0.1642	0.1816	0.0001	0.1772
6	Bins	0.1693	0.0005	0.1642	0.1693	0.0004	0.1642	0.1821	0.0004	0.1772
6	Cyl	0.1687	0.0002	0.1642	0.1689	0.0003	0.1642	0.1817	0.0002	0.1772
7	Bins	0.1709	0.0013	0.1642	0.1704	0.0013	0.1642	0.1830	0.0010	0.1772
7	Cyl	0.1686	0.0002	0.1642	0.1686	0.0003	0.1642	0.1817	0.0002	0.1772
8	Bins	0.1691	0.0004	0.1642	0.1691	0.0003	0.1642	0.1820	0.0004	0.1772
8	Cyl	0.1687	0.0002	0.1642	0.1687	0.0002	0.1642	0.1815	0.0002	0.1772

served that all the algorithms converge to the same value as in the dynamic programming approach, further confirming their reliability and accuracy.

5.3 Results for the min/max MKV model

5.3.1 Dynamic programming-based algorithms

Results for $T = 0.2$ are reported in Tables 5.11, 5.13, 5.15: They are very good for all algorithms and network used. Results for $T = 0.5$ are reported in Tables 5.12, 5.14, 5.16, and also give excellent results. Notice that with Algorithm 2, it is impossible to solve the problem with $T = 0.5$ using $N_T = 50$ due to memory issues and the time needed limited to 3 days. As we increase the number of time steps for Algorithm 3, we observe for the bins methods, as in the previous test case, a small degradation of the results due to an accumulation of regression error, and therefore Algorithm 1 should be preferred.

It is important to consider that as the maturity increases, the size of \mathcal{K} needs to be adjusted accordingly to ensure that the particles primarily remain within \mathcal{K} . This adjustment

Table 5.11: Global Algorithm 1 with $T = 0.2, \mathcal{K} = [0.21, 2.72]$.

Method	K	T/N_T	Case 1		Case 2		Case 3	
			Calc	Ref	Calc	Ref	Calc	Ref
Bins	100	0.02	0.481	0.483	0.502	0.494	0.489	0.491
Bins	100	0.01	0.481	0.483	0.503	0.494	0.489	0.491
Bins	200	0.01	0.484	0.483	0.498	0.494	0.491	0.491
Cylinder	500	0.02	0.484	0.483	0.493	0.494	0.491	0.491
Cylinder	500	0.01	0.484	0.483	0.494	0.494	0.491	0.491

Table 5.12: Global Algorithm 1 with $T = 0.5, \mathcal{K} = [-0.4, 3.21]$.

Method	K	T/N_T	Case 1		Case 2		Case 3	
			Calc	Ref	Calc	Ref	Calc	Ref
Bins	100	0.02	0.830	0.818	1.100	1.082	0.848	0.836
Bins	100	0.01	0.833	0.818	1.104	1.082	0.850	0.836
Bins	200	0.01	0.831	0.818	1.092	1.082	0.848	0.836
Cylinder	500	0.02	0.814	0.818	1.080	1.082	0.831	0.836
Cylinder	500	0.01	0.819	0.818	1.085	1.082	0.837	0.836

Table 5.13: Policy iteration Algorithm 2 with $T = 0.2, \mathcal{K} = [0.21, 2.72]$.

Method	K	T/N_T	Case 1		Case 2		Case 3	
			Calc	Ref	Calc	Ref	Calc	Ref
Bins	100	0.02	0.480	0.483	0.502	0.494	0.489	0.491
Bins	200	0.02	0.482	0.483	0.496	0.494	0.491	0.491
Cylinder	500	0.02	0.484	0.483	0.493	0.494	0.491	0.491

Table 5.14: Policy iteration Algorithm 2 with $T = 0.5, \mathcal{K} = [-0.4, 3.21]$.

Method	K	T/N_T	Case 1		Case 2		Case 3	
			Calc	Ref	Calc	Ref	Calc	Ref
Bins	100	0.02	0.819	0.818	1.088	1.082	0.836	0.836
Bins	200	0.02	0.818	0.818	1.090	1.082	0.836	0.836
Cylinder	500	0.02	0.814	0.818	1.081	1.082	0.831	0.836

Table 5.15: Value iteration Algorithm 3 with $T = 0.2, \mathcal{K} = [0.21, 2.72]$.

Method	K	T/N_T	Case 1		Case 2		Case 3	
			Calc	Ref	Calc	Ref	Calc	Ref
Bins	100	0.02	0.494	0.483	0.512	0.494	0.502	0.491
Bins	200	0.02	0.490	0.483	0.493	0.494	0.495	0.491
Cylinder	500	0.02	0.486	0.483	0.493	0.494	0.491	0.491

Table 5.16: Value iteration Algorithm 3 with $T = 0.5, \mathcal{K} = [-0.4, 3.21]$.

Method	K	T/N_T	Case 1		Case 2		Case 3	
			Calc	Ref	Calc	Ref	Calc	Ref
Bins	100	0.02	0.800	0.818	1.084	1.082	0.817	0.836
Bins	200	0.02	0.810	0.818	1.079	1.082	0.828	0.836
Bins	200	0.01	0.835	0.818	1.114	1.082	0.853	0.836
Cylinder	500	0.02	0.811	0.818	1.088	1.082	0.829	0.836
Cylinder	500	0.01	0.810	0.818	1.078	1.082	0.827	0.836

is necessary to accommodate the potential expansion of the distribution’s support as the maturity lengthens.

5.3.2 Results for Backward SDE-based algorithms

Results for this example of Section 5.1.2 are reported in Tables 5.17-5.21. All algorithms seem to converge to the good solution except the global deep MKV BSDE Algorithm 6

Table 5.17: Deep backward Algorithm 4, $T = 0.5, \mathcal{K} = [-0.40, 3.21]$.

Method	K	T/N_T	Case 1		Case 2		Case 3	
			Calc	Ref	Calc	Ref	Calc	Ref
Bins	100	0.02	0.8355	0.8180	1.1074	1.0820	0.8537	0.8360
Bins	200	0.02	0.8278	0.8180	1.0962	1.0820	0.8462	0.8360
Bins	200	0.01	0.8343	0.8180	1.0998	1.0820	0.8513	0.8360
Cylinder	500	0.02	0.8249	0.8180	1.0896	1.0820	0.8427	0.8360
Cylinder	500	0.01	0.8312	0.8180	1.0946	1.0820	0.8487	0.8360

Table 5.18: Deep backward multi-step Algorithm 5, $T = 0.5, \mathcal{K} = [-0.40, 3.21]$.

Method	K	T/N_T	Case 1		Case 2		Case 3	
			Calc	Ref	Calc	Ref	Calc	Ref
Bins	200	0.02	0.8277	0.8180	1.0966	1.0820	0.8453	0.8360
Cylinder	500	0.02	0.8259	0.8180	1.0904	1.0820	0.8427	0.8360

Table 5.19: Global deep MKV BSDE Algorithm 6, $T = 0.5, \mathcal{K} = [-0.40, 3.21]$.

Method	K	T/N_T	Case 1		Case 2		Case 3	
			Calc	Ref	Calc	Ref	Calc	Ref
Bins	200	0.02	0.8299	0.8180	1.0977	1.0820	0.8485	0.8360
Bins	100	0.01	0.8447	0.8180	1.1111	1.0820	0.8625	0.8360
Bins	200	0.01	0.8369	0.8180	1.1018	1.0820	0.8566	0.8360
Cylinder	500	0.02	0.7801	0.8180	1.0493	1.0820	0.7968	0.8360
Cylinder	500	0.01	0.7597	0.8180	1.0325	1.0820	0.7767	0.8360

Table 5.20: Global/local deep MKV BSDE Algorithm 7, $T = 0.5, \mathcal{K} = [-0.40, 3.21]$.

Method	K	T/N_T	Case 1		Case 2		Case 3	
			Calc	Ref	Calc	Ref	Calc	Ref
Bins	200	0.02	0.8528	0.8180	1.1003	1.0820	0.8692	0.8360
Bins	100	0.01	0.9146	0.8180	1.1120	1.0820	0.9219	0.8360
Bins	200	0.01	0.8406	0.8180	1.1001	1.0820	0.8560	0.8360
Cylinder	500	0.02	0.8305	0.8180	1.0952	1.0820	0.8466	0.8360
Cylinder	500	0.01	0.8666	0.8180	1.1104	1.0820	0.8817	0.8360

Table 5.21: Global deep multi-step MKV BSDE Algorithm 8, $T = 0.5, \mathcal{K} = [-0.40, 3.21]$.

Method	K	T/N_T	Case 1		Case 2		Case 3	
			Calc	Ref	Calc	Ref	Calc	Ref
Bins	200	0.02	0.8380	0.8180	1.1004	1.0820	0.8497	0.8360
Bins	200	0.01	0.8353	0.8180	1.1002	1.0820	0.8520	0.8360
Cylinder	500	0.02	0.8265	0.8180	1.0902	1.0820	0.8434	0.8360
Cylinder	500	0.01	0.8319	0.8180	1.0951	1.0820	0.8487	0.8360

that always converges on our tests (repeated many times) to a slightly different solution while using the cylinder network. Notice that, by using the bins network, we avoid the problem on this test case. Again it is not feasible to refine the time step when implementing the deep backward multi-step Algorithm 5 due to the computational time taken by the algorithm. The local deep backward Algorithm 4 seems to be the best as the results obtained in Table 5.17 are very good and the memory needed rather small. Either bins or cylinder networks can be used.

5.4 Result on the mean variance problem using the dynamic programming approach

We do not report results from Algorithm 3: Indeed, they diverge for all discretizations tested. Results for the two other algorithms are given in Tables 5.22 and 5.24 for $T = 0.2$, and in Tables 5.23 and 5.25 for $T = 0.5$. Notice that the number of bins taken for the bins network has to be high to get an accurate solution.

Again, in term of accuracy, Algorithms 1 and 2 give similar accurate results and the memory taken by both algorithms is close. However Algorithm 1 has to be preferred as the computation time is far lower when we are interested by computing the solution only at time $t = 0$.

Table 5.22: Global Algorithm 1, $T = 0.2$, $\mathcal{K} = [-0.85, 0.9]$, $T/N_T = 0.02$.

Method	K	Case 1		Case 2		Case 3	
		Calc	Anal	Calc	Anal	Calc	Anal
Bins	100	-0.0954	-0.0865	-0.1147	-0.1059	-0.3139	-0.3050
Bins	200	-0.0907	-0.0865	-0.1104	-0.1059	-0.3094	-0.3050
Bins	400	-0.0882	-0.0865	-0.1081	-0.1059	-0.3071	-0.3050
Cylinder	500	-0.0884	-0.0865	-0.1078	-0.1060	-0.3070	-0.3051
Method	K	Case 4		Case 5		Case 6	
		Calc	Anal	Calc	Anal	Calc	Anal
Bins	100	-0.0952	-0.0865	-0.0547	-0.0464	-0.1769	-0.1683
Bins	200	-0.0908	-0.0865	-0.0510	-0.0464	-0.1724	-0.1683
Bins	400	-0.0894	-0.0865	-0.0487	-0.0464	-0.1703	-0.1683
Cylinder	500	-0.0883	-0.0865	-0.0485	-0.0464	-0.1703	-0.1683

Table 5.23: Global Algorithm 1, $T = 0.5$, $\mathcal{K} = [-0.85, 0.9]$, $T/N_T = 0.02$.

Method	K	Case 1		Case 2		Case 3	
		Calc	Anal	Calc	Anal	Calc	Anal
Bins	200	-0.1018	-0.0965	-0.1214	-0.1156	-0.3200	-0.3147
Bins	400	-0.0976	-0.0965	-0.1163	-0.1156	-0.3149	-0.3147
Cylinder	500	-0.0987	-0.0965	-0.1179	-0.1156	-0.3172	-0.3147
Method	K	Case 4		Case 5		Case 6	
		Calc	Anal	Calc	Anal	Calc	Anal
Bins	200	-0.1022	-0.0965	-0.0613	-0.0562	-0.1842	-0.1786
Bins	400	-0.0969	-0.0965	-0.0562	-0.0562	-0.1788	-0.1786
Cylinder	500	-0.0985	-0.0965	-0.0583	-0.0562	-0.1804	-0.1786

Table 5.24: Policy iteration Algorithm 2, $T = 0.2, \mathcal{K} = [-0.85, 0.9], T/N_T = 0.02$.

Method	K	Case 1		Case 2		Case 3	
		Calc	Anal	Calc	Anal	Calc	Anal
Bins	100	-0.0959	-0.0865	-0.1143	-0.1060	-0.3138	-0.3051
Bins	200	-0.0906	-0.0865	-0.1102	-0.1059	-0.3094	-0.3050
Bins	400	-0.0884	-0.0865	-0.1083	-0.1059	-0.3072	-0.3050
Cylinder	500	-0.0884	-0.0865	-0.1078	-0.1060	-0.3070	-0.3051
Method	K	Case 4		Case 5		Case 6	
		Calc	Anal	Calc	Anal	Calc	Anal
Bins	100	-0.0954	-0.0865	-0.0553	-0.0464	-0.1766	-0.1683
Bins	200	-0.0908	-0.0865	-0.0505	-0.0464	-0.1723	-0.1683
Bins	400	-0.0887	-0.0865	-0.0482	-0.0464	-0.1704	-0.1683
Cylinder	500	-0.0883	-0.0865	-0.0485	-0.0464	-0.1703	-0.1683

Table 5.25: Policy iteration Algorithm 2, $T = 0.5, \mathcal{K} = [-0.85, 0.9], T/N_T = 0.02$.

Method	K	Case 1		Case 2		Case 3	
		Calc	Anal	Calc	Anal	Calc	Anal
Bins	400	-0.0978	-0.0965	-0.1171	-0.1156	-0.3140	-0.3147
Cylinder	500	-0.0986	-0.0965	-0.1175	-0.1156	-0.3164	-0.3147
Method	K	Case 4		Case 5		Case 6	
		Calc	Anal	Calc	Anal	Calc	Anal
Bins	400	-0.0985	-0.0965	-0.0579	-0.0562	-0.1789	-0.1786
Cylinder	500	-0.0986	-0.0965	-0.0583	-0.0562	-0.1807	-0.1786

5.5 Results for the non LQ MKV model using dynamic programming

In Table 5.26, results from different algorithms are provided for one run with a maturity $T = 0.4$. The settings used include $N_T = 20, 30000$ gradient iterations, $M = 10$, and $\mathcal{K} = [-0.81, 2.81]$. Based on the provided results, it is observed that Algorithm 3 yields inaccurate results. However, Algorithms 1 and 2 demonstrate a high accuracy in capturing the desired outcome.

In Table 5.27, the results for Algorithms 1 and 2 are given with a maturity of $T = 0.4$ and $\mathcal{K} = [-1.23, 3.84]$. The results are reported for $N_T = 20$. Based on the provided results, it is observed that Algorithm 1 performs better for larger time steps. This suggests that Algorithm 1 is more effective in capturing the desired results in scenarios with extended time steps.

The results obtained from your experiments confirm that Algorithm 1 is the most effective choice when employing the dynamic programming approach. The algorithm consistently produces the best results, demonstrating its superior performance in solving the problem at hand. These findings validate the selection of Algorithm 1 as the preferred choice within the dynamic programming framework.

Table 5.26: The non Linear Quadratic using $T = 0.2$.

Alg	Method	Case 1		Case 2		Case 3	
		Calc	Anal	Calc	Anal	Calc	Anal
1	Bins	1.1840	1.1735	0.9389	0.9200	1.1695	1.1585
1	Cyl	1.1815	1.1736	0.9233	0.9197	1.1666	1.1585
2	Bins	1.1896	1.1735	0.9479	0.9197	1.1757	1.1584
2	Cyl	1.1791	1.1735	0.9239	0.9196	1.1640	1.1585
3	Bins	0.9356	1.1736	0.8282	0.9198	0.9293	1.1585
3	Cyl	1.0449	1.1734	0.8646	0.9196	1.0356	1.1585

Table 5.27: The non Linear Quadratic using $T = 0.4$.

Alg	Method	Case 1		Case 2		Case 3	
		Calc	Anal	Calc	Anal	Calc	Anal
1	Bins	1.4607	1.4332	1.1645	1.1233	1.4448	1.4151
1	Cyl	1.4517	1.4332	1.1402	1.1237	1.4332	1.4150
2	Bins	1.4905	1.4333	1.1980	1.1233	1.4750	1.4150
2	Cyl	1.4627	1.4333	1.1473	1.1237	1.4447	1.4150

5.6 Results for the two dimensional systemic risk model of Section 5.1.5

The bin method suffers from the curse of dimensionality, and the numerical resolution of multi-dimensional problems is time consuming and memory intensive. Therefore, all experiments are performed in 2D using an NVIDIA H100 80 GB HBM3N graphics card. Since the bin method is only used to sample distributions with the cylinder network, these networks can be used with more bins than the bin networks with a given amount of memory. We test the algorithms using $N_T = 20$, with a resolution range of $[-0.63, 0.96]^2$. For the bin network we use 30×30 bins, while for the cylinder network we sample distributions using 50×50 bins. We first give results and sensitivities for dynamic programming based algorithms except for Algorithm 2 (which is too time consuming) in Table 5.28.

We also report the results obtained with the BSDE methods in Table 5.29 (except for Algorithm 4, which is also time-consuming).

Table 5.28: Results and sensitivities using 10 runs with dynamic programming based methods in dimension 2.

Alg	Method	Case 1			Case 4			Case 6		
		Calc	Std	Anal	Calc	Std	Anal	Calc	Std	Anal
1	Bins	0.1147	0.0003	0.1134	0.1611	0.0005	0.1604	0.1223	0.0003	0.1208
1	Cyl	0.1142	0.0003	0.1134	0.1609	0.0005	0.1604	0.1220	0.0003	0.1208
3	Bins	0.1360	0.0462	0.1134	0.1752	0.0449	0.1604	0.1364	0.0430	0.1208
3	Cyl	0.1276	0.0145	0.1134	0.1645	0.0068	0.1604	0.1339	0.0123	0.1208

Table 5.29: Some sensitivity results using 10 runs in dimension 2.

Alg	Method	Case 1			Case 2			Case 3		
		Calc	Std	Anal	Calc	Std	Anal	Calc	Std	Anal
4	Bins	0.1147	0.0003	0.1134	0.1611	0.0004	0.1604	0.1221	0.0003	0.1208
4	Cyl	0.1141	0.0003	0.1134	0.1613	0.0003	0.1604	0.1220	0.0004	0.1208
6	Bins	0.1147	0.0003	0.1134	0.1610	0.0002	0.1604	0.1222	0.0003	0.1208
6	Cyl	0.1147	0.0002	0.1134	0.1614	0.0004	0.1604	0.1220	0.0002	0.1208
7	Bins	0.1184	0.0007	0.1134	0.1635	0.0006	0.1604	0.1281	0.0015	0.1208
7	Cyl	0.1147	0.0003	0.1134	0.1614	0.0004	0.1604	0.1221	0.0005	0.1208
8	Bins	0.1146	0.0004	0.1134	0.1612	0.0005	0.1604	0.1224	0.0003	0.1208
8	Cyl	0.1148	0.0003	0.1134	0.1613	0.0003	0.1604	0.1219	0.0003	0.1208

The results are all very good, except again for the local Algorithm 3 based on the dynamic programming framework. Among the feasible algorithms, the bin Algorithm 7 is less accurate than the others which yield results close to the exact value, showing that the remaining error is mainly due to the Euler discretization of the scheme.

6 Conclusion

We have tested numerous algorithms to solve the McKean-Vlasov control problem (1.1) by using mean-field neural networks. When the problem admits a Backward SDE representation from the Pontryagin maximum principle, it is clearly more interesting to adopt this approach than the dynamic programming-based approaches for several reasons:

- It is observed that the BSDE approach consistently yields stable results across multiple runs. This stability can be attributed to the fact that, in the Pontryagin principle, the BSDE has a driver that depends on Y instead of the traditional approach where the driver is a function of Z , as highlighted in [14]. Based on these findings, Algorithm 6 emerges as the best compromise in terms of accuracy and computational time. This algorithm strikes a balance between achieving accurate results and maintaining reasonable computational efficiency.
- It is possible to use the local deep backward algorithm [21] (Algorithm 4) that yields very accurate results and is not limited by the number of time steps due to transfer learning. Moreover, the method gives the solution of the problem at each time steps for all the distributions.
- Both networks, either bins or cylinder, can be implemented. Notice that cylinder methods use less memory than bins methods especially when the number of bins has to be high to get a good accuracy.

When the maximum Pontryagin principle is not directly available, we distinguish two cases:

- First case is when the volatility of the forward process is not controlled. Then two options are available:
 - When the number of time steps is not too high, the global learning algorithm [19] (Algorithm 1), [17] seems to be the best in terms of accuracy. Then it is possible to get the function value after $t = 0$ at “visited distributions” by regression.
 - When the number of time steps is too high, memory issues force us to use the control learning by value iteration of [20] (Algorithm 3 may have difficulties to converge as shown in the non Linear Quadratic example and in the two dimensional systemic test case). Another option could be to use an hybrid algorithm as proposed in [29].
- Second case is when there is control on the diffusion coefficient, and then only the global learning algorithm should be implemented.

In conclusion, it is advisable to prioritize global Algorithms 6 and 1. When using the cylindrical network, there is no need to make any assumptions or guesses about the parameter \mathcal{K} . However, it is important to note that the global learning algorithms, as observed in [1, 10, 21], may occasionally converge to incorrect solutions, particularly in the non-mean-field case when there is a poor initialization of Y_0 that is too distant from the solution. Such problems have not been experienced in the mean-field case. To mitigate these convergence issues and ensure the reliability of the global learning algorithm, the control learning by policy iteration, as presented in [20], can be employed to verify convergence. This is particularly relevant when the loss of the global learning algorithm does not tend to zero as the number of time steps increases.

Finally, extending the bin method to dimension 3 is currently out of reach. The use of cylinder networks in higher dimensions would be possible if an effective way could be found to generate distributions that avoid bin sampling.

Acknowledgments

This work is supported by the FiME, Laboratoire de Finance des Marchés de l’Energie, and by the “Finance and Sustainable Development” EDF - CACIB Chair.

References

- [1] K. Andersson, A. Andersson, and C. W. Oosterlee, Convergence of a robust deep FBSDE method for stochastic control, *SIAM J. Sci. Comput.*, 45(1):A226–A255, 2023.
- [2] M. Basei and H. Pham, A weak martingale approach to linear-quadratic McKean-Vlasov stochastic control problem, *J. Optim. Theory Appl.*, 181(2):347–382, 2019.
- [3] C. Beck, M. Hutzenthaler, A. Jentzen, and B. Kuckuck, An overview on deep learning-based approximation methods for partial differential equations, *Discrete Contin. Dyn. Syst. Ser. B*, 28(6):3697–3746, 2020.
- [4] A. Bensoussan, J. Frehse, and P. Yam, *Mean Field Games and Mean Field Type Control Theory*, in: *Springer Briefs in Mathematics*, Springer, 2013.
- [5] R. Carmona and F. Delarue, Forward-backward stochastic differential equations and controlled McKean-Vlasov dynamics, *Ann. Probab.*, 43(5):2647–2700, 2015.

- [6] R. Carmona and F. Delarue, *Probabilistic Theory of Mean Field Games with Applications I: Mean Field FBSDEs, Control, and Games*, Springer, 2018.
- [7] R. Carmona and F. Delarue, *Probabilistic Theory of Mean Field Games with Applications I: Mean Field Games with Common Noise and Master Equations*, Springer, 2018.
- [8] R. Carmona, J.-P. Fouque, and L. Sun, Mean field games and systemic risk, *Commun. Math. Sci.*, 13(4):911–933, 2015.
- [9] R. Carmona and M. Laurière, Convergence analysis of machine learning algorithms for the numerical solution of mean field control and games: II – The finite horizon case, *arXiv:1908.01613*, 2019.
- [10] Q. Chan-Wai-Nam, J. Mikael, and X. Warin, Machine learning for semi linear PDEs, *J. Sci. Comput.*, 79(3):1667–1712, 2019.
- [11] W. E, J. Han, and A. Jentzen, Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations, *Commun. Math. Stat.*, 5(4):349–380, 2017.
- [12] J.-P. Fouque and Z. Zhang, Deep learning methods for mean field control problems with delay, *Frontiers Appl. Math. Stat.*, 6, 2020.
- [13] M. Germain, M. Laurière, H. Pham, and X. Warin, DeepSets and derivative networks for solving symmetric PDEs, *J. Sci. Comput.*, 91(63), 2022.
- [14] M. Germain, J. Mikael, and X. Warin, Numerical resolution of McKean-Vlasov FBSDEs using neural networks, *Methodol. Comput. Appl. Probab.*, 24:2557–2586, 2019.
- [15] M. Germain, H. Pham, and X. Warin, Approximation error analysis of some deep backward schemes for nonlinear PDEs, *SIAM J. Sci. Comput.*, 44(1):A28–A56, 2022.
- [16] M. Germain, H. Pham, and X. Warin, Neural networks based algorithms for stochastic control and PDEs in finance, *arXiv:2101.08068*, 2022.
- [17] E. Gobet and R. Munos, Sensitivity analysis using Itô-Malliavin calculus and martingales, and application to stochastic optimal control, *SIAM J. Control Optim.*, 43(5):1676–1713, 2005.
- [18] J. Han and W. E, Deep learning approximation for stochastic control problems, in: *NIPS Workshop on Deep Reinforcement Learning*, NIPS, 2016.
- [19] J. Han, A. Jentzen, and W. E, Solving high-dimensional partial differential equations using deep learning, *Proc. Natl. Acad. Sci. USA*, 115(34):8505–8510, 2018.
- [20] C. Huré, H. Pham, A. Bachouch, and N. Langrené, Deep neural networks algorithms for stochastic control problems on finite horizon: Convergence analysis, *SIAM J. Numer. Anal.*, 59(1):525–557, 2021.
- [21] C. Huré, H. Pham, and X. Warin, Deep backward schemes for high-dimensional nonlinear PDEs, *Math. Comp.*, 89(324):1547–1579, 2020.
- [22] A. Ismail and H. Pham, Robust Markowitz mean-variance portfolio selection under ambiguous covariance matrix, *Math. Finance*, 29:174–207, 2019.
- [23] D. P. Kingma and J. Ba, Adam: A method for stochastic optimization, *arXiv:1412.6980*, 2014.
- [24] L. Pfeiffer, Numerical methods for mean-field-type optimal control problems, *Pure Appl. Funct. Anal.*, 1(4):629–655, 2016.
- [25] H. Pham and X. Warin, Mean-field neural networks: Learning mappings on Wasserstein space, *Neural Netw.*, 168:380–393, 2023.
- [26] C. Reisinger, W. Stockinger, and Y. Zhang, A fast iterative PDE-based algorithm for feedback controls of nonsmooth mean-field control problems, *arXiv:2108.06740*, 2021.
- [27] L. Ruthotto, S. J. Osher, W. Li, L. Nurbekyan, and S. W. Fung, A machine learning framework for solving high-dimensional mean field game and mean field control problems, *Proc. Natl. Acad. Sci. USA*, 117(17):9183–9193, 2020.
- [28] R. Salhab, R. P. Malhamé, and J. Le Ny, A dynamic game model of collective choice in multi-agent systems, in: *2015 IEEE 54th Annual Conference on Decision and Control (CDC)*, IEEE, 4444–4449, 2015.
- [29] X. Warin, Reservoir optimization and machine learning methods, *arXiv:2106.08097*, 2021.
- [30] X. Warin, Quantile and moment neural networks for learning functionals of distributions, *arXiv:2303.11060*, 2023.