Mean Field Games: Numerical Methods and Applications in Machine Learning

Part 5: Deep Learning for MFC and MKV FBSDE

Mathieu LAURIÈRE

https://mlauriere.github.io/teaching/MFG-PKU-5.pdf

Peking University
Summer School on Applied Mathematics
July 26 – August 6, 2021

RECAP

Numerical Methods for MFG: Some references

Methods based on a deterministic approach:

- Finite diff. & Newton meth.: [Achdou, Capuzzo-Dolcetta'10; Achdou, Capuzzo-Dolcetta, Camilli'13; . . .]
- Gradient descent: [L., Pironneau'14; Pfeiffer'16]
- Semi-Lagrangian scheme: [Carlini, Silva'14; Carlini, Silva'15]
- Augmented Lagrangian & ADMM: [Benamou, Carlier'14; Achdou, L.'16; Andreev'17]
- Primal-dual algo.: [Briceño-Arias, Kalise, Silva'18; BAKS + Kobeissi, L., Mateos González'18]
- Monotone operators: [Almulla *et al.*'17; Gomes, Saúde'18; Gomes, Yang'18]

Methods based on a probabilistic approach:

- Cubature: [Chaudru de Raynal, Garcia Trillos'15]
- Recursion: [Chassagneux *et al.*'17; Angiuli *et al.*'18]
- MC & Regression: [Balata, Huré, L., Pham, Pimentel'18]

Surveys and lecture notes: [Achdou'13 (LNM); Achdou, L.'20 (Cetraro); L.'21 (AMS)]

Numerical Methods for MFG: Some references

Methods based on a deterministic approach:

- Finite diff. & Newton meth.: [Achdou, Capuzzo-Dolcetta'10; Achdou, Capuzzo-Dolcetta, Camilli'13; . . .]
- Gradient descent: [L., Pironneau'14; Pfeiffer'16]
- Semi-Lagrangian scheme: [Carlini, Silva'14; Carlini, Silva'15]
- Augmented Lagrangian & ADMM: [Benamou, Carlier'14; Achdou, L.'16; Andreev'17]
- Primal-dual algo.: [Briceño-Arias, Kalise, Silva'18; BAKS + Kobeissi, L., Mateos González'18]
- Monotone operators: [Almulla *et al.*'17; Gomes, Saúde'18; Gomes, Yang'18]

Methods based on a probabilistic approach:

- Cubature: [Chaudru de Raynal, Garcia Trillos'15]
- Recursion: [Chassagneux *et al.*'17; Angiuli *et al.*'18]
- MC & Regression: [Balata, Huré, L., Pham, Pimentel'18]

Surveys and lecture notes: [Achdou'13 (LNM); Achdou, L.'20 (Cetraro); L.'21 (AMS)]

Limitations:

- dimensionality (typically: state in dimension ≤ 3)
- structure of the problem (typically: simple costs, dynamics and noises)

Numerical Methods for MFG: Some references

Methods based on a deterministic approach:

- Finite diff. & Newton meth.: [Achdou, Capuzzo-Dolcetta'10; Achdou, Capuzzo-Dolcetta, Camilli'13; ...]
- Gradient descent: [L., Pironneau'14; Pfeiffer'16]
- Semi-Lagrangian scheme: [Carlini, Silva'14; Carlini, Silva'15]
- Augmented Lagrangian & ADMM: [Benamou, Carlier'14; Achdou, L.'16; Andreev'17]
- Primal-dual algo.: [Briceño-Arias, Kalise, Silva'18; BAKS + Kobeissi, L., Mateos González'18]
- Monotone operators: [Almulla *et al.*'17; Gomes, Saúde'18; Gomes, Yang'18]

Methods based on a probabilistic approach:

- Cubature: [Chaudru de Raynal, Garcia Trillos'15]
- Recursion: [Chassagneux *et al.*'17; Angiuli *et al.*'18]
- MC & Regression: [Balata, Huré, L., Pham, Pimentel'18]

Surveys and lecture notes: [Achdou'13 (LNM); Achdou, L.'20 (Cetraro); L.'21 (AMS)]

Limitations:

- dimensionality (typically: state in dimension ≤ 3)
- structure of the problem (typically: simple costs, dynamics and noises)

Recent progress: extending the toolbox with tools from **machine learning**:

- approximation without a grid (mesh-free methods): **opt. control** & **distribution**
  $\longrightarrow$ [Carmona, L.; Al-Aradi *et al.*; Fouque *et al.*; Germain *et al.*; Ruthotto *et al.*; Agram *et al.*; ...]
- even when the **dynamics / cost are not known** (model-free methods)
  $\longrightarrow$ [Guo *et al.*; Subramanian *et al.*; Elie *et al.*; Carmona *et al.*; Pham *et al.*; Yang *et al.*; ...]

Outline

Ingredient 1: Neural Networks

- **Goal:** Minimize over $\varphi(\cdot)$, $\mathbb{J}(\varphi) := \mathbb{E}_\xi[\mathbb{L}(\varphi, \xi)]$
- **Ex.:** Regression: $\xi = (x, f(x))$ for some f, $\mathbb{L}(\varphi, \xi) = \|\varphi(x) - f(x)\|^2$

Ingredient 1: Neural Networks

- **Goal:** Minimize over $\varphi(\cdot)$, $\mathbb{J}(\varphi) := \mathbb{E}_\xi[\mathbb{L}(\varphi, \xi)]$
- **Ex.:** Regression: $\xi = (x, f(x))$ for some f, $\mathbb{L}(\varphi, \xi) = \|\varphi(x) - f(x)\|^2$
- **Idea:** Instead of min. over all $\varphi(\cdot)$, min. over parameters θ of $\varphi_\theta(\cdot)$
- **Ex.: Feedforward fully-connected neural network:**
  φ_θ with weights & biases $\theta = (\beta^{(k)}, w^{(k)})_{k=1,\dots,\ell}$

$$\underbrace{\varphi_\theta(x)}_{\varphi(\theta,x)} = \psi^{(\ell)}\left(\beta^{(\ell)} + w^{(\ell)} \dots \psi^{(2)}\left(\beta^{(2)} + w^{(2)} \underbrace{\psi^{(1)}(\beta^{(1)} + w^{(1)}x)}_{\text{one hidden layer}}\right) \dots\right)$$

  where $\psi^{(i)} \in \{$ sigmoid, ReLU, $\dots\}$: non-linear activation functions (coordinate-wise)
- Depth = number of layers; width of a layer = dimension of bias vector

Ingredient 1: Neural Networks

- **Goal:** Minimize over $\varphi(\cdot)$, $\mathbb{J}(\varphi) := \mathbb{E}_\xi[\mathbb{L}(\varphi, \xi)]$
- **Ex.:** Regression: $\xi = (x, f(x))$ for some f, $\mathbb{L}(\varphi, \xi) = \|\varphi(x) - f(x)\|^2$
- **Idea:** Instead of min. over all $\varphi(\cdot)$, min. over parameters θ of $\varphi_\theta(\cdot)$
- **Ex.: Feedforward fully-connected neural network:**
  φ_θ with weights & biases $\theta = (\beta^{(k)}, w^{(k)})_{k=1,\dots,\ell}$

$$\underbrace{\varphi_\theta(x)}_{\varphi(\theta, x)} = \psi^{(\ell)}\Big(\beta^{(\ell)} + w^{(\ell)} \dots \psi^{(2)}\Big(\beta^{(2)} + w^{(2)} \underbrace{\psi^{(1)}(\beta^{(1)} + w^{(1)} x)}_{\text{one hidden layer}}\Big) \dots\Big)$$

  where $\psi^{(i)} \in \{$ sigmoid, ReLU, $\dots\}$: non-linear activation functions (coordinate-wise)
- Depth = number of layers; width of a layer = dimension of bias vector
- Other architectures

Ingredient 1: Neural Networks – Universal Approximation

Ingredient 1: Neural Networks – Gradients

Differentiation: can compute partial derivatives by automatic differentiation (AD) at every (θ, x):

- With respect to parameters: $\nabla_\theta \varphi(\theta, x)$

$$\nabla_{\beta^{(\ell)}} \varphi(\theta, x) = \ldots, \qquad \nabla_{w^{(2)}} \varphi(\theta, x) = \ldots$$

$\Rightarrow$ can perform **SGD** on these parameters

Ingredient 1: Neural Networks – Gradients

Differentiation: can compute partial derivatives by automatic differentiation (AD) at every (θ, x):

- With respect to parameters: $\nabla_\theta \varphi(\theta, x)$

$$\nabla_{\beta^{(\ell)}} \varphi(\theta, x) = \dots, \qquad \nabla_{w^{(2)}} \varphi(\theta, x) = \dots$$

  $\Rightarrow$ can perform **SGD** on these parameters

- With respect to state variable: $\nabla_x \varphi(\theta, x)$ can be computed by AD too

$$\partial_{x_1} \varphi(\theta, x) = \dots$$

  $\Rightarrow$ can be used in PDEs

Ingredient 2: Stochastic Gradient Descent

Goal: Minimize over $\varphi(\cdot)$, $\mathbb{J}(\varphi) := \mathbb{E}_\xi[\mathbb{L}(\varphi, \xi)]$

Parameterization: $\widetilde{\mathbb{J}}(\theta) := \mathbb{E}_\xi[\widetilde{\mathbb{L}}(\theta, \xi)]$, where $\widetilde{\mathbb{L}}(\theta, \xi) := \mathbb{L}(\varphi_\theta, \xi)$

Ingredient 2: Stochastic Gradient Descent

Goal: Minimize over $\varphi(\cdot)$, $\mathbb{J}(\varphi) := \mathbb{E}_\xi[\mathbb{L}(\varphi, \xi)]$

Parameterization: $\widetilde{\mathbb{J}}(\theta) := \mathbb{E}_\xi[\widetilde{\mathbb{L}}(\theta, \xi)]$, where $\widetilde{\mathbb{L}}(\theta, \xi) := \mathbb{L}(\varphi_\theta, \xi)$

Setting: the distribution of ξ is unknown, but

- we have some samples (i.e. random realizations) of ξ
- we know $\mathbb{L}$

Goal: Minimize over $\varphi(\cdot)$, $\mathbb{J}(\varphi) := \mathbb{E}_\xi[\mathbb{L}(\varphi, \xi)]$

Parameterization: $\widetilde{\mathbb{J}}(\theta) := \mathbb{E}_\xi[\widetilde{\mathbb{L}}(\theta, \xi)]$, where $\widetilde{\mathbb{L}}(\theta, \xi) := \mathbb{L}(\varphi_\theta, \xi)$

Setting: the distribution of ξ is unknown, but

- we have some samples (i.e. random realizations) of ξ
- we know $\mathbb{L}$

Ex: Regression: $\xi = (x, f(x))$, $\widetilde{\mathbb{J}}(\theta) := \mathbb{E}_\xi[\, \|\varphi_\theta(x) - f(x)\|^2\,]$

Ingredient 2: Stochastic Gradient Descent

Goal: Minimize over $\varphi(\cdot)$, $\mathbb{J}(\varphi) := \mathbb{E}_\xi[\mathbb{L}(\varphi, \xi)]$

Parameterization: $\widetilde{\mathbb{J}}(\theta) := \mathbb{E}_\xi[\widetilde{\mathbb{L}}(\theta, \xi)]$, where $\widetilde{\mathbb{L}}(\theta, \xi) := \mathbb{L}(\varphi_\theta, \xi)$

Setting: the distribution of ξ is unknown, but

- we have some samples (i.e. random realizations) of ξ
- we know $\mathbb{L}$

Ex: Regression: $\xi = (x, f(x))$, $\widetilde{\mathbb{J}}(\theta) := \mathbb{E}_\xi[\,\|\varphi_\theta(x) - f(x)\|^2\,]$

Input: Initial param. θ_0; data $S = (\xi_s)_{s=1,\dots,|S|}$; nb of steps $\mathtt{K}$; learning rates $(\eta^{(\mathtt{k})})_\mathtt{k}$

Output: Parameter $\theta^\star$ s.t. $\varphi_{\theta^\star}$ (approximately) minimizes $\widetilde{\mathbb{J}}$

Initialize $\theta^{(0)} = \theta_0$

for $\mathtt{k} = 0, 1, 2, \dots, \mathtt{K} - 1$ **do**

  Pick $s \in S$ randomly

  Compute the gradient $\nabla_\theta \widetilde{\mathbb{L}}(\theta^{(\mathtt{k}-1)}, \xi_s) = \frac{d}{d\theta}\mathbb{L}(\varphi_{\theta^{(\mathtt{k}-1)}}, \xi_s)$

  Set $\theta^{(\mathtt{k})} = \theta^{(\mathtt{k}-1)} - \eta^{(\mathtt{k})} \nabla_\theta \widetilde{\mathbb{L}}(\theta^{(\mathtt{k}-1)}, \xi_s)$

return $\theta^{(\mathtt{K})}$

Ingredient 2: Stochastic Gradient Descent – cont.

- Many variants:
  - Learning rate: `ADAM` (Adaptive Moment Estimation), ...
  - Samples: Mini-batches, ...

Ingredient 2: Stochastic Gradient Descent – cont.

- Many variants:
  - Learning rate: `ADAM` (Adaptive Moment Estimation), ...
  - Samples: Mini-batches, ...

- Generator for $\xi \Rightarrow$ can generate Monte Carlo samples on the fly

Ingredient 2: Stochastic Gradient Descent – cont.

- Many variants:
  - Learning rate: `ADAM` (Adaptive Moment Estimation), ...
  - Samples: Mini-batches, ...
- Generator for $\xi \Rightarrow$ can generate Monte Carlo samples on the fly
- Robbins-Monro

Ingredient 2: Stochastic Gradient Descent – cont.

- Many variants:
  - Learning rate: `ADAM` (Adaptive Moment Estimation), ...
  - Samples: Mini-batches, ...
- Generator for $\xi \Rightarrow$ can generate Monte Carlo samples on the fly
- Robbins-Monro
- Links with convex minimization & stochastic approximation

Analysis: Error Types

- Consider the task: minimize over φ the **population risk**:

$$\mathcal{R}(\varphi) = \mathbb{E}_{x,y}[L(\varphi(x), y)]$$

with $x \sim \mu$ and $y = f(x) + \epsilon$ for some noise ϵ where f is unknown

Analysis: Error Types

- Consider the task: minimize over φ the **population risk**:

$$\mathcal{R}(\varphi) = \mathbb{E}_{x,y}[L(\varphi(x), y)]$$

  with $x \sim \mu$ and $y = f(x) + \epsilon$ for some noise ϵ where f is unknown
- In practice:
  - minimize over a **hypothesis class** $\mathcal{F}$ of φ
  - finite number of samples, $S = (x_m, y_m)_{m=1,\ldots,M}$: (regularized) **empirical risk**:

$$\hat{\mathcal{R}}_S(\varphi) = \frac{1}{M} \sum_{m=1}^{M} L(\varphi(x_m), y_m) \qquad (+\text{ regu})$$

  - finite number of **optimization steps**, say `k`

Analysis: Error Types

- Consider the task: minimize over φ the **population risk**:

$$\mathcal{R}(\varphi) = \mathbb{E}_{x,y}[L(\varphi(x), y)]$$

  with $x \sim \mu$ and $y = f(x) + \epsilon$ for some noise ϵ where f is unknown
- In practice:
  - minimize over a **hypothesis class** $\mathcal{F}$ of φ
  - finite number of samples, $S = (x_m, y_m)_{m=1,\dots,M}$: (regularized) **empirical risk**:

$$\hat{\mathcal{R}}_S(\varphi) = \frac{1}{M}\sum_{m=1}^{M} L(\varphi(x_m), y_m) \qquad \text{(+ regu)}$$

  - finite number of **optimization steps**, say $\mathtt{k}$
- We are interested in:
  - **Approximation error:** Letting $\varphi^* = \operatorname{argmin}_{\varphi\in\mathcal{F}} \operatorname{dist}(\varphi, f)$,

$$\epsilon_{\text{approx}} = \operatorname{dist}(\varphi^*, f)$$

  - **Estimation error:** Letting $\hat{\varphi}_S = \operatorname{argmin}_{\varphi\in\mathcal{F}} \hat{\mathcal{R}}_S(\varphi)$

$$\epsilon_{\text{estim}} = \operatorname{dist}(\hat{\varphi}_S, \varphi^*)$$

  - **Optimization error:** After $\mathtt{k}$ steps, we get $\varphi_S^{(\mathtt{k})}$;

$$\epsilon_{\text{optim}} = \operatorname{dist}(\varphi_S^{(\mathtt{k})}, \hat{\varphi}_S)$$

Analysis: Error Types

- Consider the task: minimize over φ the **population risk**:

$$\mathcal{R}(\varphi) = \mathbb{E}_{x,y}[L(\varphi(x), y)]$$

  with $x \sim \mu$ and $y = f(x) + \epsilon$ for some noise ϵ where f is unknown
- In practice:
  - minimize over a **hypothesis class** $\mathcal{F}$ of φ
  - finite number of samples, $S = (x_m, y_m)_{m=1,\dots,M}$: (regularized) **empirical risk**:

$$\hat{\mathcal{R}}_S(\varphi) = \frac{1}{M}\sum_{m=1}^{M} L(\varphi(x_m), y_m) \qquad (+\text{ regu})$$

  - finite number of **optimization steps**, say $\mathtt{k}$
- We are interested in:
  - **Approximation error:** Letting $\varphi^* = \operatorname{argmin}_{\varphi\in\mathcal{F}} \operatorname{dist}(\varphi, f)$,

$$\epsilon_{\text{approx}} = \operatorname{dist}(\varphi^*, f)$$

  - **Estimation error:** Letting $\hat{\varphi}_S = \operatorname{argmin}_{\varphi\in\mathcal{F}} \hat{\mathcal{R}}_S(\varphi)$

$$\epsilon_{\text{estim}} = \operatorname{dist}(\hat{\varphi}_S, \varphi^*)$$

  - **Optimization error:** After $\mathtt{k}$ steps, we get $\varphi_S^{(\mathtt{k})}$;

$$\epsilon_{\text{optim}} = \operatorname{dist}(\varphi_S^{(\mathtt{k})}, \hat{\varphi}_S)$$

  - **Generalization error** of the learnt $\varphi_S^{(\mathtt{k})}$:

$$\epsilon_{\text{gene}} = \epsilon_{\text{approx}} + \epsilon_{\text{estim}} + \epsilon_{\text{optim}}$$

Outline

Stochastic optimal control problem:

Minimize over $v(\cdot,\cdot)$

$$J(v(\cdot,\cdot)) = \mathbb{E}\Big[\int_0^T f(X_t, v(t,X_t))\,dt + g(X_T)\Big],$$

with

$$X_0 \sim m_0\,, \quad dX_t = b(X_t, v(t,X_t))\,dt + \sigma dW_t$$

Stochastic Optimal Control: Approximate Problem

Stochastic optimal control problem: (2) neural network φ_θ,

Minimize over **neural network** parameters θ

$$J(\theta) = \mathbb{E}\Big[\int_0^T f\left(X_t, \varphi_\theta(t, X_t)\right)\, dt + g\left(X_T\right) \Big],$$

with

$$X_0 \sim m_0\,, \quad dX_t = b(X_t, \varphi_\theta(t, X_t))\, dt + \sigma dW_t$$

Stochastic Optimal Control: Approximate Problem

Stochastic optimal control problem: (2) neural network φ_θ, (3) time discretization

Minimize over **neural network** parameters θ and N_T time steps

$$J^{N_T}(\theta) = \mathbb{E}\Big[\sum_{n=0}^{N_T-1} f\left(X_n, \varphi_\theta(t_n, X_n)\right) \Delta t + g\left(X_{N_T}\right) \Big],$$

with

$$X_0 \sim m_0\,, \quad X_{n+1} - X_n = b(X_n, \varphi_\theta(t_n, X_n))\Delta t + \sigma \Delta W_n$$

Stochastic Optimal Control: Approximate Problem

Stochastic optimal control problem: (2) neural network φ_θ, (3) time discretization

Minimize over **neural network** parameters θ and N_T time steps

$$J^{N_T}(\theta) = \mathbb{E}\left[\sum_{n=0}^{N_T-1} f\left(X_n, \varphi_\theta(t_n, X_n)\right) \Delta t + g\left(X_{N_T}\right) \right],$$

with

$$X_0 \sim m_0\,, \quad X_{n+1} - X_n = b(X_n, \varphi_\theta(t_n, X_n))\Delta t + \sigma \Delta W_n$$

$\rightarrow$ neural network induces an approximation error

$\rightarrow$ time discretization induce extra errors

MFC problem:

Minimize over $v(\cdot,\cdot)$

$$J(v(\cdot,\cdot)) = \mathbb{E}\Big[\int_0^T f(X_t, \mu_t, v(t, X_t))\,dt + g(X_T, \mu_T)\Big],$$

where $\mu_t = \mathcal{L}(X_t)$ with

$$X_0 \sim m_0\,, \quad dX_t = b(X_t, \mu_t, v(t, X_t))\,dt + \sigma dW_t$$

MFC problem: (1) Finite pop.,

Minimize over **decentralized** controls $v(\cdot,\cdot)$ with N agents

$$J^N(v(\cdot,\cdot)) = \mathbb{E}\Big[\frac{1}{N}\sum_{i=1}^{N}\int_0^T f\left(X_t^i, \mu_t^N, v(t, X_t^i)\right)\,dt + g\left(X_T^i, \mu_T^N\right)\Big],$$

where $\mu_t^N = \frac{1}{N}\sum_{j=1}^N \delta_{X_t^j}$, with

$$X_0^j \sim m_0\,, \quad dX_t^j = b(X_t^j, \mu_t^N, v(t, X_t^j))\,dt + \sigma dW_t^j$$

MFC: Approximate Problem

MFC problem: (1) Finite pop., (2) neural network φ_θ,

Minimize over **neural network** parameters θ with N agents

$$J^N(\theta) = \mathbb{E}\Big[\frac{1}{N}\sum_{i=1}^{N}\int_0^T f\left(X_t^i, \mu_t^N, \varphi_\theta(t, X_t^i)\right)\, dt + g\left(X_T^i, \mu_T^N\right)\Big],$$

where $\mu_t^N = \frac{1}{N}\sum_{j=1}^N \delta_{X_t^j}$, with

$$X_0^j \sim m_0\,, \quad dX_t^j = b(X_t^j, \mu_t^N, \varphi_\theta(t, X_t^j))\, dt + \sigma dW_t^j$$

MFC problem: (1) Finite pop., (2) neural network φ_θ, (3) time discretization

Minimize over **neural network** parameters $\theta \in \Theta$ with N agents and N_T time steps

$$J^{N,N_T}(\theta) = \mathbb{E}\Big[\frac{1}{N}\sum_{i=1}^{N}\sum_{n=0}^{N_T-1} f\left(X_n^i, \mu_n^N, \varphi_\theta(t_n, X_n^i)\right)\Delta t + g\left(X_{N_T}^i, \mu_{N_T}^N\right)\Big],$$

where $\mu_n^N = \frac{1}{N}\sum_{j=1}^{N}\delta_{X_n^j}$, with

$$X_0^j \sim m_0\,, \quad X_{n+1}^j - X_n^j = b(X_n^j, \mu_n^N, \varphi_\theta(t_n, X_n^j))\Delta t + \sigma\Delta W_n^j$$

MFC: Approximate Problem

MFC problem: (1) Finite pop., (2) neural network φ_θ, (3) time discretization

Minimize over **neural network** parameters $\theta \in \Theta$ with N agents and N_T time steps

$$J^{N,N_T}(\theta) = \mathbb{E}\Big[\frac{1}{N}\sum_{i=1}^{N}\sum_{n=0}^{N_T-1} f\left(X_n^i, \mu_n^N, \varphi_\theta(t_n, X_n^i)\right)\Delta t + g\left(X_{N_T}^i, \mu_{N_T}^N\right)\Big],$$

where $\mu_n^N = \frac{1}{N}\sum_{j=1}^{N}\delta_{X_n^j}$, with

$$X_0^j \sim m_0\,, \quad X_{n+1}^j - X_n^j = b(X_n^j, \mu_n^N, \varphi_\theta(t_n, X_n^j))\Delta t + \sigma\Delta W_n^j$$

$\rightarrow$ neural network induces an approximation error

$\rightarrow$ Finite population and time discretization induce extra errors

MFC: Approximate Problem

MFC problem: (1) Finite pop., (2) neural network φ_θ, (3) time discretization

Minimize over **neural network** parameters $\theta \in \Theta$ with N agents and N_T time steps

$$J^{N,N_T}(\theta) = \mathbb{E}\Big[\frac{1}{N}\sum_{i=1}^{N}\sum_{n=0}^{N_T-1} f\left(X_n^i, \mu_n^N, \varphi_\theta(t_n, X_n^i)\right)\Delta t + g\left(X_{N_T}^i, \mu_{N_T}^N\right)\Big],$$

where $\mu_n^N = \frac{1}{N}\sum_{j=1}^N \delta_{X_n^j}$, with

$$X_0^j \sim m_0\,, \quad X_{n+1}^j - X_n^j = b(X_n^j, \mu_n^N, \varphi_\theta(t_n, X_n^j))\Delta t + \sigma \Delta W_n^j$$

→ neural network induces an approximation error

→ Finite population and time discretization induce extra errors

N.B.: **decentralized** control

Convergence Analysis

- The following kind of convergence result (bound on the **approximation error**) can be proved (see [Carmona, L.'19][1]):
  Under suitable assumptions (in particular regularity of the value function),

$$\left| \inf_{v(\cdot,\cdot)} J(v(\cdot,\cdot)) - \inf_{\theta\in\Theta} J^{N,N_T}(\theta) \right| \leq \epsilon_1(N) + \epsilon_2(\dim(\theta)) + \epsilon_3(N_T)$$

[1] Carmona, R., & Laurière, M. (2019). Convergence Analysis of Machine Learning Algorithms for the Numerical Solution of Mean Field Control and Games: II–The Finite Horizon Case. arXiv preprint arXiv:1908.01613. To appear in *Annals of Applied Probability*

[2] Carmona, R., & Laurière, M. (2021). Convergence Analysis of Machine Learning Algorithms for the Numerical Solution of Mean Field Control and Games I: The Ergodic Case. *SIAM Journal on Numerical Analysis*, 59(3), 1455-1485.

Convergence Analysis

- The following kind of convergence result (bound on the **approximation error**) can be proved (see [Carmona, L.'19][1]):

  Under suitable assumptions (in particular regularity of the value function),

$$\left| \inf_{v(\cdot,\cdot)} J(v(\cdot,\cdot)) - \inf_{\theta\in\Theta} J^{N,N_T}(\theta) \right| \leq \epsilon_1(N) + \epsilon_2(\dim(\theta)) + \epsilon_3(N_T)$$

- The **estimation error** for shallow neural networks can be analyzed using techniques similar to [Carmona, L.'21][2]

[1] Carmona, R., & Laurière, M. (2019). Convergence Analysis of Machine Learning Algorithms for the Numerical Solution of Mean Field Control and Games: II–The Finite Horizon Case. arXiv preprint arXiv:1908.01613. To appear in *Annals of Applied Probability*

[2] Carmona, R., & Laurière, M. (2021). Convergence Analysis of Machine Learning Algorithms for the Numerical Solution of Mean Field Control and Games I: The Ergodic Case. *SIAM Journal on Numerical Analysis*, 59(3), 1455-1485.

Convergence Analysis

- The following kind of convergence result (bound on the **approximation error**) can be proved (see [Carmona, L.'19][1]):

  Under suitable assumptions (in particular regularity of the value function),

$$\left| \inf_{v(\cdot,\cdot)} J(v(\cdot,\cdot)) - \inf_{\theta\in\Theta} J^{N,N_T}(\theta) \right| \leq \epsilon_1(N) + \epsilon_2(\dim(\theta)) + \epsilon_3(N_T)$$

- The **estimation error** for shallow neural networks can be analyzed using techniques similar to [Carmona, L.'21][2]

- The **optimization error** remains to be studied

[1] Carmona, R., & Laurière, M. (2019). Convergence Analysis of Machine Learning Algorithms for the Numerical Solution of Mean Field Control and Games: II–The Finite Horizon Case. arXiv preprint arXiv:1908.01613. To appear in *Annals of Applied Probability*

[2] Carmona, R., & Laurière, M. (2021). Convergence Analysis of Machine Learning Algorithms for the Numerical Solution of Mean Field Control and Games I: The Ergodic Case. *SIAM Journal on Numerical Analysis*, 59(3), 1455-1485.

- The following kind of convergence result (bound on the **approximation error**) can be proved (see [Carmona, L.'19][1]):
  Under suitable assumptions (in particular regularity of the value function),

$$\left| \inf_{v(\cdot,\cdot)} J(v(\cdot,\cdot)) - \inf_{\theta\in\Theta} J^{N,N_T}(\theta) \right| \leq \epsilon_1(N) + \epsilon_2(\dim(\theta)) + \epsilon_3(N_T)$$

- The **estimation error** for shallow neural networks can be analyzed using techniques similar to [Carmona, L.'21][2]

- The **optimization error** remains to be studied

- Many extensions to be studied

[1] Carmona, R., & Laurière, M. (2019). Convergence Analysis of Machine Learning Algorithms for the Numerical Solution of Mean Field Control and Games: II–The Finite Horizon Case. arXiv preprint arXiv:1908.01613. To appear in *Annals of Applied Probability*

[2] Carmona, R., & Laurière, M. (2021). Convergence Analysis of Machine Learning Algorithms for the Numerical Solution of Mean Field Control and Games I: The Ergodic Case. *SIAM Journal on Numerical Analysis*, 59(3), 1455-1485.

Approximation Error Analysis: Main Ingredients of the Proof

Proposition 1 (N agents & decentralized controls):

Under suitable assumptions, there exists a decentralized control v^* s.t. *(d = dimension of X_t)*

$$\left| \inf_{v(\cdot)} J(v(\cdot)) - J^N(v^*(\cdot)) \right| \leq \epsilon_1(N) \in \widetilde{O}\left(N^{-1/d}\right).$$

Proof: propagation of chaos type argument [Carmona, Delarue'18]

Approximation Error Analysis: Main Ingredients of the Proof

Proposition 1 (N agents & decentralized controls):

Under suitable assumptions, there exists a decentralized control v^* s.t. *(d = dimension of X_t)*

$$\left| \inf_{v(\cdot)} J(v(\cdot)) - J^N(v^*(\cdot)) \right| \leq \epsilon_1(N) \in \widetilde{O}\left(N^{-1/d}\right).$$

Proof: propagation of chaos type argument [Carmona, Delarue'18]

Proposition 2 (approximation by neural networks): Under suitable assumptions

There exists a set of parameters $\theta \in \Theta$ for a one-hidden layer $\hat{\varphi}_\theta$ s.t.

$$\left| J^N(v^*(\cdot)) - J^N(\hat{\varphi}_\theta(\cdot)) \right| \leq \epsilon_2(\dim(\theta)) \in O\left(\dim(\theta)^{-\frac{1}{3(d+1)}}\right).$$

Proof: Key difficulty: approximate $v^*(\cdot)$ by $\hat{\varphi}_\theta(\cdot)$ while controlling $\|\nabla \hat{\varphi}_\theta(\cdot)\|$ by $\|\nabla v^*(\cdot)\|$

$\rightarrow$ universal approximation without rate of convergence is not enough

$\rightarrow$ approximation rate for the derivative too, e.g. from [Mhaskar, Micchelli'95]

Approximation Error Analysis: Main Ingredients of the Proof

Proposition 1 (N agents & decentralized controls):

Under suitable assumptions, there exists a decentralized control v^* s.t. *(d = dimension of X_t)*

$$\left| \inf_{v(\cdot)} J(v(\cdot)) - J^N(v^*(\cdot)) \right| \leq \epsilon_1(N) \in \widetilde{O}\left(N^{-1/d}\right).$$

Proof: propagation of chaos type argument [Carmona, Delarue'18]

Proposition 2 (approximation by neural networks): Under suitable assumptions

There exists a set of parameters $\theta \in \Theta$ for a one-hidden layer $\hat{\varphi}_\theta$ s.t.

$$\left| J^N(v^*(\cdot)) - J^N(\hat{\varphi}_\theta(\cdot)) \right| \leq \epsilon_2(\dim(\theta)) \in O\left(\dim(\theta)^{-\frac{1}{3(d+1)}}\right).$$

Proof: Key difficulty: approximate $v^*(\cdot)$ by $\hat{\varphi}_\theta(\cdot)$ while controlling $\|\nabla \hat{\varphi}_\theta(\cdot)\|$ by $\|\nabla v^*(\cdot)\|$

$\rightarrow$ universal approximation without rate of convergence is not enough

$\rightarrow$ approximation rate for the derivative too, e.g. from [Mhaskar, Micchelli'95]

Proposition 3 (Euler-Maruyama scheme):

For a specific neural network $\hat{\varphi}_\theta(\cdot)$,

$$\left| J^N(\hat{\varphi}_\theta(\cdot)) - J^{N,N_T}(\hat{\varphi}_\theta(\cdot)) \right| \leq \epsilon_3(N_T) \in O\left(N_T^{-1/2}\right).$$

Key point: $O(\cdot)$ independent of N and $\dim(\theta)$

Proof: analysis of **strong error rate** for Euler scheme (reminiscent of [Bossy, Talay'97])

Implementation

- Key idea: replace optimal control problem by (finite dim.) optimization problem:
  - Loss function = cost: $J^{N,N_T}(\theta) = \mathbb{E}[\mathbb{L}(\varphi_\theta, \xi)]$
  - One sample: $\xi = \left(X_0^j, (\Delta W_n^j)_{n=0,\dots,N_T-1}\right)_{j=1,\dots,N}$

$\rightarrow$ can use **Stochastic Gradient Descent**

Implementation

- Key idea: replace optimal control problem by (finite dim.) optimization problem:
  - Loss function = cost: $J^{N,N_T}(\theta) = \mathbb{E}[\mathbb{L}(\varphi_\theta, \xi)]$
  - One sample: $\xi = \left(X_0^j, (\Delta W_n^j)_{n=0,\dots,N_T-1}\right)_{j=1,\dots,N}$

  $\rightarrow$ can use **Stochastic Gradient Descent**
- Related work:
  - Generalizes standard stochastic control problems [. . . ; Gobet, Munos'05; Han, E'16]
  - Related work with mean field: [Fouque, Zhang'19; Germain *et al.*'19; . . .]

Implementation

- Key idea: replace optimal control problem by (finite dim.) optimization problem:
  - Loss function = cost: $J^{N,N_T}(\theta) = \mathbb{E}[\mathbb{L}(\varphi_\theta, \xi)]$
  - One sample: $\xi = \left(X_0^j, (\Delta W_n^j)_{n=0,\ldots,N_T-1}\right)_{j=1,\ldots,N}$

  $\rightarrow$ can use **Stochastic Gradient Descent**
- Related work:
  - Generalizes standard stochastic control problems [. . . ; Gobet, Munos'05; Han, E'16]
  - Related work with mean field: [Fouque, Zhang'19; Germain *et al.*'19; . . .]
- Structure:

Implementation

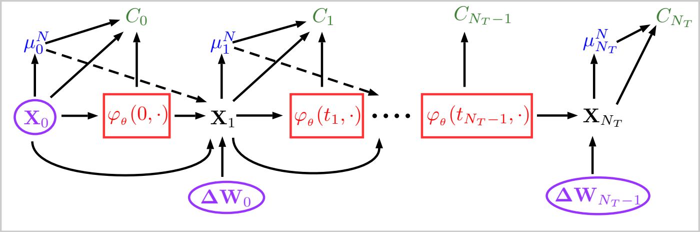
- Key idea: replace optimal control problem by (finite dim.) optimization problem:
  - Loss function = cost: $J^{N,N_T}(\theta) = \mathbb{E}[\mathbb{L}(\varphi_\theta, \xi)]$
  - One sample: $\xi = \left(X_0^j, (\Delta W_n^j)_{n=0,\dots,N_T-1}\right)_{j=1,\dots,N}$

  $\rightarrow$ can use **Stochastic Gradient Descent**
- Related work:
  - Generalizes standard stochastic control problems [. . . ; Gobet, Munos'05; Han, E'16]
  - Related work with mean field: [Fouque, Zhang'19; Germain *et al.*'19; . . .]
- Structure:

Implementation

- Key idea: replace optimal control problem by (finite dim.) optimization problem:
  - Loss function = cost: $J^{N,N_T}(\theta) = \mathbb{E}[\mathbb{L}(\varphi_\theta, \xi)]$
  - One sample: $\xi = \left(X_0^j, (\Delta W_n^j)_{n=0,\dots,N_T-1}\right)_{j=1,\dots,N}$

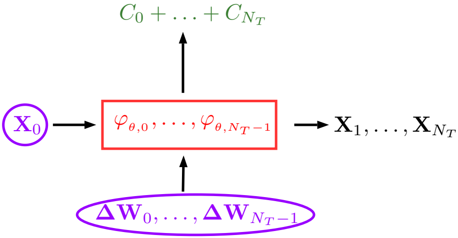
  $\rightarrow$ can use **Stochastic Gradient Descent**
- Related work:
  - Generalizes standard stochastic control problems [. . . ; Gobet, Munos'05; Han, E'16]
  - Related work with mean field: [Fouque, Zhang'19; Germain *et al.*'19; . . .]
- Structure:



Implementation

- Key idea: replace optimal control problem by (finite dim.) optimization problem:
  - Loss function = cost: $J^{N,N_T}(\theta) = \mathbb{E}[\mathbb{L}(\varphi_\theta, \xi)]$
  - One sample: $\xi = \left(X_0^j, (\Delta W_n^j)_{n=0,\dots,N_T-1}\right)_{j=1,\dots,N}$

  $\rightarrow$ can use **Stochastic Gradient Descent**
- Related work:
  - Generalizes standard stochastic control problems [. . . ; Gobet, Munos'05; Han, E'16]
  - Related work with mean field: [Fouque, Zhang'19; Germain *et al.*'19; . . .]
- Structure:

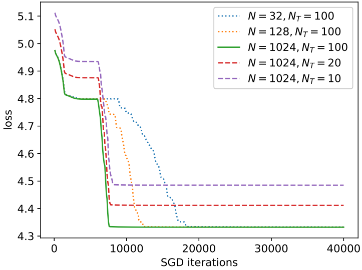


Benchmark to assess **empirical convergence of SGD:** LQ problem with explicit sol.

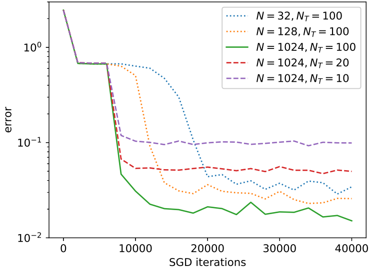
Example: Linear dynamics, quadratic costs of the type

$$f(x,\mu,v) = \underbrace{(\bar\mu - x)^2}_{\text{distance to mean position}} + \underbrace{v^2}_{\text{cost of moving}}, \qquad \bar\mu = \underbrace{\int \mu(\xi)d\xi}_{\text{mean position}}, \qquad g(x) = x^2$$

Numerical example with $d = 10$ (see [Carmona, L.'19]):



total cost (= loss function)



L^2-error on the control

Numerical Illustration 2: min-LQ MFC with common noise

MFC with simple CN (inspired by [Salhab, Malhamé, Le Ny] and [Achdou, Lasry]):

- $dX_t = \phi_t(X_t, \epsilon^0_t)dt + \sigma dW_t$, $\epsilon^0_t = 0$ until $t = T/2$, and then ξ_1 or ξ_2 w.p. $1/2$
- running cost $|\phi_t(X_t, \epsilon^0_t)|^2$, final cost $(X_T - \epsilon^0_T)^2 + \bar{Q}_T(\bar{m}_T - X_T)^2$

- Ex.: $\sigma = 0.1$, $T = 1$, $\xi_1 = -1.5$, $\xi_2 = +1.5$
- Numerics: **neural network** $\varphi_\theta(t, X_t, \epsilon^0_t)$ VS benchmark with **system of 6 PDEs**

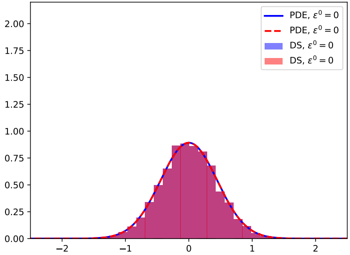
(More details in [Carmona, L.'19])

Numerical Illustration 2: min-LQ MFC with common noise

MFC with simple CN (inspired by [Salhab, Malhamé, Le Ny] and [Achdou, Lasry]):

- $dX_t = \phi_t(X_t, \epsilon_t^0)dt + \sigma dW_t$, $\epsilon_t^0 = 0$ until $t = T/2$, and then ξ_1 or ξ_2 w.p. $1/2$
- running cost $|\phi_t(X_t, \epsilon_t^0)|^2$, final cost $(X_T - \epsilon_T^0)^2 + \bar{Q}_T(\bar{m}_T - X_T)^2$

- Ex.: $\sigma = 0.1$, $T = 1$, $\xi_1 = -1.5$, $\xi_2 = +1.5$
- Numerics: **neural network** $\varphi_\theta(t, X_t, \epsilon_t^0)$ VS benchmark with **system of 6 PDEs**



$t = 0$

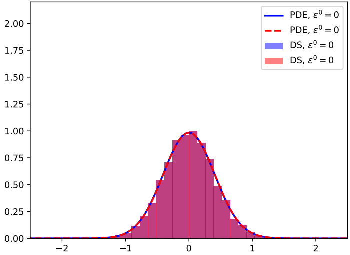
(More details in [Carmona, L.'19])

Numerical Illustration 2: min-LQ MFC with common noise

MFC with simple CN (inspired by [Salhab, Malhamé, Le Ny] and [Achdou, Lasry]):

- $dX_t = \phi_t(X_t, \epsilon_t^0)dt + \sigma dW_t$, $\epsilon_t^0 = 0$ until $t = T/2$, and then ξ_1 or ξ_2 w.p. $1/2$
- running cost $|\phi_t(X_t, \epsilon_t^0)|^2$, final cost $(X_T - \epsilon_T^0)^2 + \bar{Q}_T(\bar{m}_T - X_T)^2$

- Ex.: $\sigma = 0.1$, $T = 1$, $\xi_1 = -1.5$, $\xi_2 = +1.5$
- Numerics: **neural network** $\varphi_\theta(t, X_t, \epsilon_t^0)$ VS benchmark with **system of 6 PDEs**



$t = 0.1$

- Until $T/2$: concentrate around mid-point $= 0$

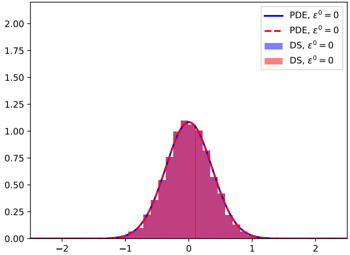
(More details in [Carmona, L.'19])

Numerical Illustration 2: min-LQ MFC with common noise

MFC with simple CN (inspired by [Salhab, Malhamé, Le Ny] and [Achdou, Lasry]):

- $dX_t = \phi_t(X_t, \epsilon_t^0)dt + \sigma dW_t$, $\epsilon_t^0 = 0$ until $t = T/2$, and then ξ_1 or ξ_2 w.p. $1/2$
- running cost $|\phi_t(X_t, \epsilon_t^0)|^2$, final cost $(X_T - \epsilon_T^0)^2 + \bar{Q}_T(\bar{m}_T - X_T)^2$

- Ex.: $\sigma = 0.1$, $T = 1$, $\xi_1 = -1.5$, $\xi_2 = +1.5$
- Numerics: **neural network** $\varphi_\theta(t, X_t, \epsilon_t^0)$ VS benchmark with **system of 6 PDEs**



$t = 0.2$

- Until $T/2$: concentrate around mid-point $= 0$

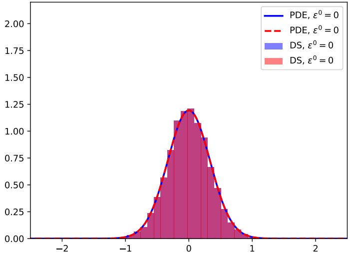
(More details in [Carmona, L.'19])

Numerical Illustration 2: min-LQ MFC with common noise

MFC with simple CN (inspired by [Salhab, Malhamé, Le Ny] and [Achdou, Lasry]):

- $dX_t = \phi_t(X_t, \epsilon_t^0)dt + \sigma dW_t$, $\epsilon_t^0 = 0$ until $t = T/2$, and then ξ_1 or ξ_2 w.p. $1/2$
- running cost $|\phi_t(X_t, \epsilon_t^0)|^2$, final cost $(X_T - \epsilon_T^0)^2 + \bar{Q}_T(\bar{m}_T - X_T)^2$

- Ex.: $\sigma = 0.1$, $T = 1$, $\xi_1 = -1.5$, $\xi_2 = +1.5$
- Numerics: **neural network** $\varphi_\theta(t, X_t, \epsilon_t^0)$ VS benchmark with **system of 6 PDEs**



$t = 0.3$

- Until $T/2$: concentrate around mid-point $= 0$

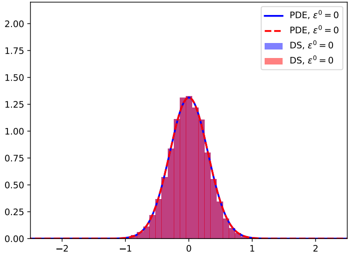
(More details in [Carmona, L.'19])

Numerical Illustration 2: min-LQ MFC with common noise

MFC with simple CN (inspired by [Salhab, Malhamé, Le Ny] and [Achdou, Lasry]):

- $dX_t = \phi_t(X_t, \epsilon_t^0)dt + \sigma dW_t$, $\epsilon_t^0 = 0$ until $t = T/2$, and then ξ_1 or ξ_2 w.p. $1/2$
- running cost $|\phi_t(X_t, \epsilon_t^0)|^2$, final cost $(X_T - \epsilon_T^0)^2 + \bar{Q}_T(\bar{m}_T - X_T)^2$

- Ex.: $\sigma = 0.1$, $T = 1$, $\xi_1 = -1.5$, $\xi_2 = +1.5$
- Numerics: **neural network** $\varphi_\theta(t, X_t, \epsilon_t^0)$ VS benchmark with **system of 6 PDEs**



$t = 0.4$

- Until $T/2$: concentrate around mid-point $= 0$

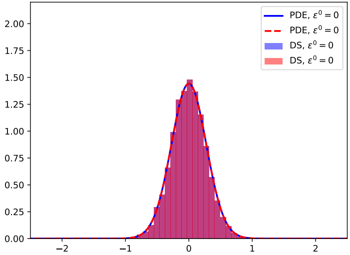
(More details in [Carmona, L.'19])

Numerical Illustration 2: min-LQ MFC with common noise

MFC with simple CN (inspired by [Salhab, Malhamé, Le Ny] and [Achdou, Lasry]):

- $dX_t = \phi_t(X_t, \epsilon^0_t)dt + \sigma dW_t$, $\epsilon^0_t = 0$ until $t = T/2$, and then ξ_1 or ξ_2 w.p. $1/2$
- running cost $|\phi_t(X_t, \epsilon^0_t)|^2$, final cost $(X_T - \epsilon^0_T)^2 + \bar{Q}_T(\bar{m}_T - X_T)^2$

- Ex.: $\sigma = 0.1$, $T = 1$, $\xi_1 = -1.5$, $\xi_2 = +1.5$
- Numerics: **neural network** $\varphi_\theta(t, X_t, \epsilon^0_t)$ VS benchmark with **system of 6 PDEs**



$t = 0.5$

- Until $T/2$: concentrate around mid-point $= 0$

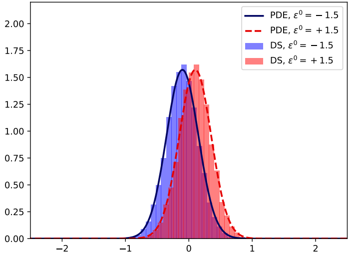
(More details in [Carmona, L.'19])

Numerical Illustration 2: min-LQ MFC with common noise

MFC with simple CN (inspired by [Salhab, Malhamé, Le Ny] and [Achdou, Lasry]):

- $dX_t = \phi_t(X_t, \epsilon^0_t)dt + \sigma dW_t$, $\epsilon^0_t = 0$ until $t = T/2$, and then ξ_1 or ξ_2 w.p. $1/2$
- running cost $|\phi_t(X_t, \epsilon^0_t)|^2$, final cost $(X_T - \epsilon^0_T)^2 + \bar{Q}_T(\bar{m}_T - X_T)^2$

- Ex.: $\sigma = 0.1$, $T = 1$, $\xi_1 = -1.5$, $\xi_2 = +1.5$
- Numerics: **neural network** $\varphi_\theta(t, X_t, \epsilon^0_t)$ VS benchmark with **system of 6 PDEs**



$t = 0.6$

- Until $T/2$: concentrate around mid-point $= 0$
- After $T/2$: move towards the target selected by common noise

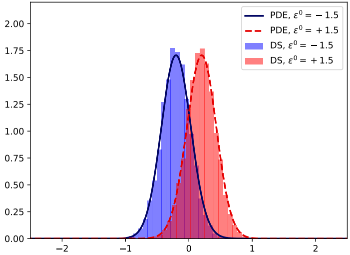
(More details in [Carmona, L.'19])

Numerical Illustration 2: min-LQ MFC with common noise

MFC with simple CN (inspired by [Salhab, Malhamé, Le Ny] and [Achdou, Lasry]):

- $dX_t = \phi_t(X_t, \epsilon^0_t)dt + \sigma dW_t$, $\epsilon^0_t = 0$ until $t = T/2$, and then ξ_1 or ξ_2 w.p. $1/2$
- running cost $|\phi_t(X_t, \epsilon^0_t)|^2$, final cost $(X_T - \epsilon^0_T)^2 + \bar{Q}_T(\bar{m}_T - X_T)^2$

- Ex.: $\sigma = 0.1$, $T = 1$, $\xi_1 = -1.5$, $\xi_2 = +1.5$
- Numerics: **neural network** $\varphi_\theta(t, X_t, \epsilon^0_t)$ VS benchmark with **system of 6 PDEs**



$t = 0.7$

- Until $T/2$: concentrate around mid-point $= 0$
- After $T/2$: move towards the target selected by common noise

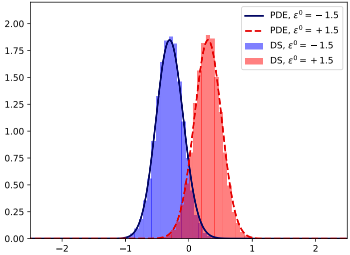
(More details in [Carmona, L.'19])

Numerical Illustration 2: min-LQ MFC with common noise

MFC with simple CN (inspired by [Salhab, Malhamé, Le Ny] and [Achdou, Lasry]):

- $dX_t = \phi_t(X_t, \epsilon_t^0)dt + \sigma dW_t$, $\epsilon_t^0 = 0$ until $t = T/2$, and then ξ_1 or ξ_2 w.p. $1/2$
- running cost $|\phi_t(X_t, \epsilon_t^0)|^2$, final cost $(X_T - \epsilon_T^0)^2 + \bar{Q}_T(\bar{m}_T - X_T)^2$

- Ex.: $\sigma = 0.1$, $T = 1$, $\xi_1 = -1.5$, $\xi_2 = +1.5$
- Numerics: **neural network** $\varphi_\theta(t, X_t, \epsilon_t^0)$ VS benchmark with **system of 6 PDEs**



$t = 0.8$

- Until $T/2$: concentrate around mid-point $= 0$
- After $T/2$: move towards the target selected by common noise

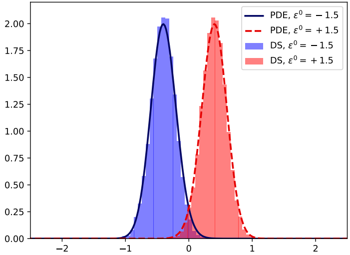
(More details in [Carmona, L.'19])

Numerical Illustration 2: min-LQ MFC with common noise

MFC with simple CN (inspired by [Salhab, Malhamé, Le Ny] and [Achdou, Lasry]):

- $dX_t = \phi_t(X_t, \epsilon^0_t)dt + \sigma dW_t$, $\epsilon^0_t = 0$ until $t = T/2$, and then ξ_1 or ξ_2 w.p. $1/2$
- running cost $|\phi_t(X_t, \epsilon^0_t)|^2$, final cost $(X_T - \epsilon^0_T)^2 + \bar{Q}_T(\bar{m}_T - X_T)^2$

- Ex.: $\sigma = 0.1$, $T = 1$, $\xi_1 = -1.5$, $\xi_2 = +1.5$
- Numerics: **neural network** $\varphi_\theta(t, X_t, \epsilon^0_t)$ VS benchmark with **system of 6 PDEs**



$t = 0.9$

- Until $T/2$: concentrate around mid-point $= 0$
- After $T/2$: move towards the target selected by common noise

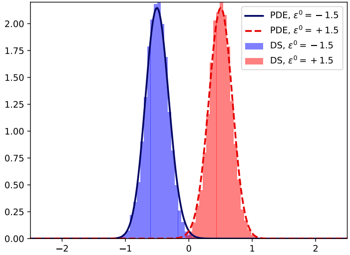
(More details in [Carmona, L.'19])

Numerical Illustration 2: min-LQ MFC with common noise

MFC with simple CN (inspired by [Salhab, Malhamé, Le Ny] and [Achdou, Lasry]):

- $dX_t = \phi_t(X_t, \epsilon^0_t)dt + \sigma dW_t$, $\epsilon^0_t = 0$ until $t = T/2$, and then ξ_1 or ξ_2 w.p. $1/2$
- running cost $|\phi_t(X_t, \epsilon^0_t)|^2$, final cost $(X_T - \epsilon^0_T)^2 + \bar{Q}_T(\bar{m}_T - X_T)^2$

- Ex.: $\sigma = 0.1$, $T = 1$, $\xi_1 = -1.5$, $\xi_2 = +1.5$
- Numerics: **neural network** $\varphi_\theta(t, X_t, \epsilon^0_t)$ VS benchmark with **system of 6 PDEs**



$t = 1$

- Until $T/2$: concentrate around mid-point $= 0$
- After $T/2$: move towards the target selected by common noise

(More details in [Carmona, L.'19])

Numerical Illustration 3: MFC with Interactions Through the Controls

Price Impact Model (see [Carmona, Lacker; Carmona, Delarue]):

Price process: with ν^{v} = population's distribution over actions,

$$dS_t^{v} = \gamma \int_{\mathbb{R}} a d\nu_t^{v}(a) dt + \sigma_0 dW_t^0$$

Typical agent's inventory: $dX_t^{v} = v_t dt + \sigma dW_t$

Typical agent's wealth: $dK_t^{v} = -\big(v_t S_t^{v} + c_v(v_t)\big)dt$

Typical agent's portfolio value: $V_t^{v} = K_t^{v} + X_t^{v} S_t^{v}$

Numerical Illustration 3: MFC with Interactions Through the Controls

Price Impact Model (see [Carmona, Lacker; Carmona, Delarue]):

Price process: with ν^v = population's distribution over actions,

$$dS_t^v = \gamma \int_{\mathbb{R}} a d\nu_t^v(a) dt + \sigma_0 dW_t^0$$

Typical agent's inventory: $dX_t^v = v_t dt + \sigma dW_t$

Typical agent's wealth: $dK_t^v = -\big(v_t S_t^v + c_v(v_t)\big) dt$

Typical agent's portfolio value: $V_t^v = K_t^v + X_t^v S_t^v$

Objective: minimize

$$J(v) = \mathbb{E}\Big[\int_0^T c_X(X_t^v) dt + g(X_T^v) - V_T^v \Big]$$

Numerical Illustration 3: MFC with Interactions Through the Controls

Price Impact Model (see [Carmona, Lacker; Carmona, Delarue]):

Price process: with $\nu^v =$ population's distribution over actions,

$$dS_t^v = \gamma \int_{\mathbb{R}} a d\nu_t^v(a) dt + \sigma_0 dW_t^0$$

Typical agent's inventory: $dX_t^v = v_t dt + \sigma dW_t$

Typical agent's wealth: $dK_t^v = -\big(v_t S_t^v + c_v(v_t)\big) dt$

Typical agent's portfolio value: $V_t^v = K_t^v + X_t^v S_t^v$

Objective: minimize

$$J(v) = \mathbb{E}\Big[\int_0^T c_X(X_t^v) dt + g(X_T^v) - V_T^v \Big]$$

Equivalent problem:

$$J(v) = \mathbb{E}\Big[\int_0^T \left(c_v(v_t) + c_X(X_t^v) - \gamma X_t^v \int_{\mathbb{R}} a d\nu_t^v(a) \right) dt + g(X_T^v) \Big]$$

Numerical Illustration 3: MFC with Interactions Through the Controls

Price Impact Model (see [Carmona, Lacker; Carmona, Delarue]):

Price process: with ν^v = population's distribution over actions,

$$dS_t^v = \gamma \int_{\mathbb{R}} a d\nu_t^v(a) dt + \sigma_0 dW_t^0$$

Typical agent's inventory: $dX_t^v = v_t dt + \sigma dW_t$

Typical agent's wealth: $dK_t^v = -\Big(v_t S_t^v + c_v(v_t)\Big) dt$

Typical agent's portfolio value: $V_t^v = K_t^v + X_t^v S_t^v$

Objective: minimize

$$J(v) = \mathbb{E}\Big[\int_0^T c_X(X_t^v) dt + g(X_T^v) - V_T^v\Big]$$

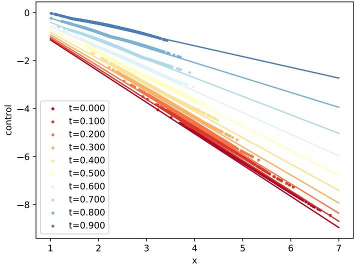
Equivalent problem:

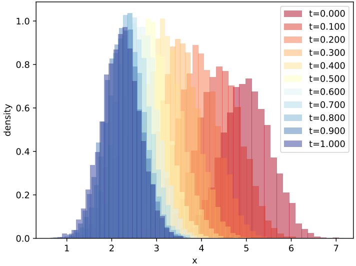
$$J(v) = \mathbb{E}\Big[\int_0^T \left(c_v(v_t) + c_X(X_t^v) - \gamma X_t^v \int_{\mathbb{R}} a d\nu_t^v(a)\right) dt + g(X_T^v)\Big]$$

Take: $c_v(v) = \frac{1}{2} c_v v^2$, $c_X(x) = \frac{1}{2} c_X x^2$ and $g(x) = \frac{1}{2} c_g x^2$

Numerical Illustration 3: MFC with Interactions Through the Controls

Control learnt (left) and associated state distribution (right)

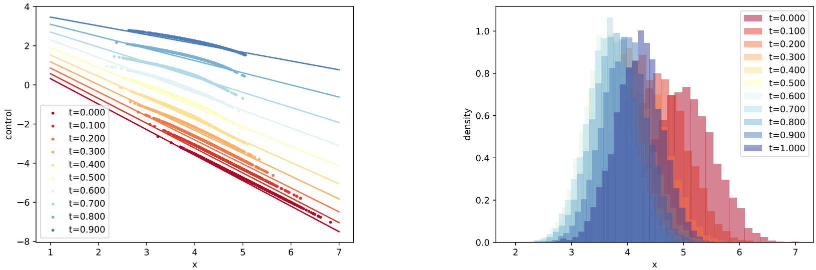




$$T = 1,\ c_X = 2,\ c_v = 1,\ c_g = 0.3,\ \sigma = 0.5,\ \gamma = 0.2$$

Numerical Illustration 3: MFC with Interactions Through the Controls

Control learnt (left) and associated state distribution (right)



$$T = 1,\ c_X = 2,\ c_v = 1,\ c_g = 0.3,\ \sigma = 0.5,\ \gamma = 1$$

Outline

DeepBSDE: Shooting Method for FBSDE

Solutions of sto. control problems can be characterized by **FBSDEs** of the form

$$\begin{cases} dX_t = B(t, X_t, Y_t)dt + dW_t, & X_0 \sim m_0 & \to \text{state} \\ dY_t = -F(t, X_t, Y_t)dt + Z_t \cdot dW_t, & Y_T = G(X_T) & \to \text{control/cost} \end{cases}$$

(stemming from sto. Pontryagin's or Bellman's principle: $F = f$ or $F = \partial_x H$)

[3] E, W., Han, J., & Jentzen, A. (2017). Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. *Communications in Mathematics and Statistics*, 5(4), 349-380.

DeepBSDE: Shooting Method for FBSDE

Solutions of sto. control problems can be characterized by **FBSDEs** of the form

$$\begin{cases} dX_t = B(t, X_t, Y_t)dt + dW_t, & X_0 \sim m_0 & \rightarrow \text{state} \\ dY_t = -F(t, X_t, Y_t)dt + Z_t \cdot dW_t, & Y_T = G(X_T) & \rightarrow \text{control/cost} \end{cases}$$

(stemming from sto. Pontryagin's or Bellman's principle: $F = f$ or $F = \partial_x H$)

Shooting: Guess Y_0 and $(Z_t)_t$ [Ma,Yong; Sannikov; Han, Jentzen, E'17; ...][3]
$\rightarrow$ recover sol. (X, Y, Z) is found by opt. control of 2 **forward** SDEs

[3] E, W., Han, J., & Jentzen, A. (2017). Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. *Communications in Mathematics and Statistics*, 5(4), 349-380.

DeepBSDE: Shooting Method for FBSDE

Solutions of sto. control problems can be characterized by **FBSDEs** of the form

$$\begin{cases} dX_t = B(t, X_t, Y_t)dt + dW_t, & X_0 \sim m_0 & \to \text{state} \\ dY_t = -F(t, X_t, Y_t)dt + Z_t \cdot dW_t, & Y_T = G(X_T) & \to \text{control/cost} \end{cases}$$

(stemming from sto. Pontryagin's or Bellman's principle: $F = f$ or $F = \partial_x H$)

Shooting: Guess Y_0 and $(Z_t)_t$ [Ma,Yong; Sannikov; Han, Jentzen, E'17; ...][3]
$\to$ recover sol. (X, Y, Z) is found by opt. control of 2 **forward** SDEs

Reformulation as a new **optimal control problem**

Minimize over $y_0(\cdot)$ and $\mathbf{z}(\cdot) = (z_t(\cdot))_{t\geq 0}$

$$\mathfrak{J}(y_0(\cdot), \mathbf{z}(\cdot)) = \mathbb{E}\left[\|Y_T^{y_0,\mathbf{z}} - G(X_T^{y_0,\mathbf{z}})\|^2 \right],$$

under the constraint that $(X^{y_0,\mathbf{z}}, Y^{y_0,\mathbf{z}})$ solve: $\forall t \in [0, T]$

$$\begin{cases} dX_t = B(t, X_t, Y_t)dt + dW_t, & X_0 \sim m_0, \\ dY_t = -F(t, X_t, Y_t)dt + z(t, X_t) \cdot dW_t, & Y_0 = y_0(X_0). \end{cases}$$

[3] E, W., Han, J., & Jentzen, A. (2017). Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. *Communications in Mathematics and Statistics*, 5(4), 349-380.

DeepBSDE: Shooting Method for FBSDE

Solutions of sto. control problems can be characterized by **FBSDEs** of the form

$$\begin{cases} dX_t = B(t, X_t, Y_t)dt + dW_t, & X_0 \sim m_0 & \to \text{state} \\ dY_t = -F(t, X_t, Y_t)dt + Z_t \cdot dW_t, & Y_T = G(X_T) & \to \text{control/cost} \end{cases}$$

(stemming from sto. Pontryagin's or Bellman's principle: $F = f$ or $F = \partial_x H$)

Shooting: Guess Y_0 and $(Z_t)_t$ [Ma,Yong; Sannikov; Han, Jentzen, E'17; ...][3]
$\to$ recover sol. (X, Y, Z) is found by opt. control of 2 **forward** SDEs

Reformulation as a new **optimal control problem**

Minimize over $y_0(\cdot)$ and $\mathbf{z}(\cdot) = (z_t(\cdot))_{t \geq 0}$

$$\mathfrak{J}(y_0(\cdot), \mathbf{z}(\cdot)) = \mathbb{E}\left[\|Y_T^{y_0,\mathbf{z}} - G(X_T^{y_0,\mathbf{z}})\|^2 \right],$$

under the constraint that $(X^{y_0,\mathbf{z}}, Y^{y_0,\mathbf{z}})$ solve: $\forall t \in [0, T]$

$$\begin{cases} dX_t = B(t, X_t, Y_t)dt + dW_t, & X_0 \sim m_0, \\ dY_t = -F(t, X_t, Y_t)dt + z(t, X_t) \cdot dW_t, & Y_0 = y_0(X_0). \end{cases}$$

$\to$ New optimal control problem: apply previous method, replacing $y_0(\cdot), z(\cdot, \cdot)$ by NN

[3] E, W., Han, J., & Jentzen, A. (2017). Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. *Communications in Mathematics and Statistics*, 5(4), 349-380.

DeepBSDE: Shooting Method for FBSDE

Solutions of sto. control problems can be characterized by **FBSDEs** of the form

$$\begin{cases} dX_t = B(t, X_t, Y_t)dt + dW_t, & X_0 \sim m_0 & \to \text{state} \\ dY_t = -F(t, X_t, Y_t)dt + Z_t \cdot dW_t, & Y_T = G(X_T) & \to \text{control/cost} \end{cases}$$

(stemming from sto. Pontryagin's or Bellman's principle: $F = f$ or $F = \partial_x H$)

Shooting: Guess Y_0 and $(Z_t)_t$ [Ma,Yong; Sannikov; Han, Jentzen, E'17; ...][3]
$\to$ recover sol. (X, Y, Z) is found by opt. control of 2 **forward** SDEs

Reformulation as a new **optimal control problem**

Minimize over $y_0(\cdot)$ and $\mathbf{z}(\cdot) = (z_t(\cdot))_{t \geq 0}$

$$\mathfrak{J}(y_0(\cdot), \mathbf{z}(\cdot)) = \mathbb{E}\left[\|Y_T^{y_0,\mathbf{z}} - G(X_T^{y_0,\mathbf{z}})\|^2 \right],$$

under the constraint that $(X^{y_0,\mathbf{z}}, Y^{y_0,\mathbf{z}})$ solve: $\forall t \in [0, T]$

$$\begin{cases} dX_t = B(t, X_t, Y_t)dt + dW_t, & X_0 \sim m_0, \\ dY_t = -F(t, X_t, Y_t)dt + z(t, X_t) \cdot dW_t, & Y_0 = y_0(X_0). \end{cases}$$

$\to$ New optimal control problem: apply previous method, replacing $y_0(\cdot), z(\cdot,\cdot)$ by NN
NB: This problem is *not* the original stochastic control problem !

[3] E, W., Han, J., & Jentzen, A. (2017). Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. *Communications in Mathematics and Statistics*, 5(4), 349-380.

Application to Solve PDEs

Feynman-Kac formula: correspondence $u(t, X_t) = Y_t$ where

Application to Solve PDEs

Feynman-Kac formula: correspondence $u(t, X_t) = Y_t$ where

- u solves the PDE

$$\begin{cases} u(T, x) = G(x) \\ \frac{\partial u}{\partial t}(t,x) + B(t,x)\frac{\partial u}{\partial x}(t,x) + \frac{1}{2}\sigma^2 \frac{\partial^2 u}{\partial x \partial x}(t,x) + F(t,x) = 0 \end{cases}$$

- X solves the SDE:

$$dX_t = B(t,x)dt + \sigma dW_t$$

- (Y, Z) solves the BSDE:

$$\begin{cases} Y_T = G(X_T) \\ dY_t = -F(t, X_t)dt + Z_t dW_t \end{cases}$$

Application to Solve PDEs

Feynman-Kac formula: correspondence $u(t, X_t) = Y_t$ where

- u solves the PDE

$$\begin{cases} u(T, x) = G(x) \\ \frac{\partial u}{\partial t}(t, x) + B(t, x)\frac{\partial u}{\partial x}(t, x) + \frac{1}{2}\sigma^2 \frac{\partial^2 u}{\partial x \partial x}(t, x) + F(t, x) = 0 \end{cases}$$

- X solves the SDE:

$$dX_t = B(t, x)dt + \sigma dW_t$$

- (Y, Z) solves the BSDE:

$$\begin{cases} Y_T = G(X_T) \\ dY_t = -F(t, X_t)dt + Z_t dW_t \end{cases}$$

- In fact $Z_t = \sigma \partial_x u(t, X_t)$

Application to Solve PDEs

Feynman-Kac formula: correspondence $u(t, X_t) = Y_t$ where

- u solves the PDE

$$\begin{cases} u(T,x) = G(x) \\ \frac{\partial u}{\partial t}(t,x) + B(t,x)\frac{\partial u}{\partial x}(t,x) + \frac{1}{2}\sigma^2\frac{\partial^2 u}{\partial x \partial x}(t,x) + F(t,x) = 0 \end{cases}$$

- X solves the SDE:

$$dX_t = B(t,x)dt + \sigma dW_t$$

- (Y, Z) solves the BSDE:

$$\begin{cases} Y_T = G(X_T) \\ dY_t = -F(t, X_t)dt + Z_t dW_t \end{cases}$$

- In fact $Z_t = \sigma \partial_x u(t, X_t)$
- Connection also works with $dX_t = dW_t$ and a different Y_t ...

Application to Solve PDEs

Feynman-Kac formula: correspondence $u(t, X_t) = Y_t$ where

- u solves the PDE

$$\begin{cases} u(T,x) = G(x) \\ \frac{\partial u}{\partial t}(t,x) + B(t,x)\frac{\partial u}{\partial x}(t,x) + \frac{1}{2}\sigma^2\frac{\partial^2 u}{\partial x \partial x}(t,x) + F(t,x) = 0 \end{cases}$$

- X solves the SDE:

$$dX_t = B(t,x)dt + \sigma dW_t$$

- (Y, Z) solves the BSDE:

$$\begin{cases} Y_T = G(X_T) \\ dY_t = -F(t, X_t)dt + Z_t dW_t \end{cases}$$

- In fact $Z_t = \sigma \partial_x u(t, X_t)$
- Connection also works with $dX_t = dW_t$ and a different Y_t ...
- Application: solve a PDE by solving the corresponding (F)BSDE

Application to Solve PDEs

Feynman-Kac formula: correspondence $u(t, X_t) = Y_t$ where

- u solves the PDE

$$\begin{cases} u(T,x) = G(x) \\ \frac{\partial u}{\partial t}(t,x) + B(t,x)\frac{\partial u}{\partial x}(t,x) + \frac{1}{2}\sigma^2 \frac{\partial^2 u}{\partial x \partial x}(t,x) + F(t,x) = 0 \end{cases}$$

- X solves the SDE:

$$dX_t = B(t,x)dt + \sigma dW_t$$

- (Y, Z) solves the BSDE:

$$\begin{cases} Y_T = G(X_T) \\ dY_t = -F(t, X_t)dt + Z_t dW_t \end{cases}$$

- In fact $Z_t = \sigma \partial_x u(t, X_t)$
- Connection also works with $dX_t = dW_t$ and a different Y_t ...
- Application: solve a PDE by solving the corresponding (F)BSDE
- Ex. HJB equation. Many variations/extensions

Shooting Method for MKV FBSDE

Solutions of MFG (and MFC) can be characterized by **MKV FBSDEs** of the form

$$\begin{cases} dX_t = B(t, X_t, \mathcal{L}(X_t), Y_t)dt + dW_t, & X_0 \sim m_0 & \to \text{state} \\ dY_t = -F(t, X_t, \mathcal{L}(X_t), Y_t)dt + Z_t \cdot dW_t, & Y_T = G(X_T, \mathcal{L}(X_T)) & \to \text{control/cost} \end{cases}$$

(stemming from sto. Pontryagin's or Bellman's principle: $F = f$ or $F = \partial_x H$)

Shooting Method for MKV FBSDE

Solutions of MFG (and MFC) can be characterized by **MKV FBSDEs** of the form

$$\begin{cases} dX_t = B(t, X_t, \mathcal{L}(X_t), Y_t)dt + dW_t, & X_0 \sim m_0 & \rightarrow \text{state} \\ dY_t = -F(t, X_t, \mathcal{L}(X_t), Y_t)dt + Z_t \cdot dW_t, & Y_T = G(X_T, \mathcal{L}(X_T)) & \rightarrow \text{control/cost} \end{cases}$$

(stemming from sto. Pontryagin's or Bellman's principle: $F = f$ or $F = \partial_x H$)

Shooting: Guess Y_0 and $(Z_t)_t$ [Ma,Yong; Sannikov; Han, Jentzen, E'17; . . .]
$\rightarrow$ recover sol. (X, Y, Z) is found by opt. control of 2 **forward** SDEs

Shooting Method for MKV FBSDE

Solutions of MFG (and MFC) can be characterized by **MKV FBSDEs** of the form

$$\begin{cases} dX_t = B(t, X_t, \mathcal{L}(X_t), Y_t)dt + dW_t, & X_0 \sim m_0 & \to \text{state} \\ dY_t = -F(t, X_t, \mathcal{L}(X_t), Y_t)dt + Z_t \cdot dW_t, & Y_T = G(X_T, \mathcal{L}(X_T)) & \to \text{control/cost} \end{cases}$$

(stemming from sto. Pontryagin's or Bellman's principle: $F = f$ or $F = \partial_x H$)

Shooting: Guess Y_0 and $(Z_t)_t$ [Ma,Yong; Sannikov; Han, Jentzen, E'17; . . .]
$\to$ recover sol. (X, Y, Z) is found by opt. control of 2 **forward** SDEs

Reformulation as a **MFC problem**

Minimize over $y_0(\cdot)$ and $\mathbf{z}(\cdot) = (z_t(\cdot))_{t\geq 0}$

$$\mathfrak{J}(y_0(\cdot), \mathbf{z}(\cdot)) = \mathbb{E}\Big[\|Y_T^{y_0,\mathbf{z}} - G(X_T^{y_0,\mathbf{z}}, \mathcal{L}(X_T^{y_0,\mathbf{z}}))\|^2 \Big],$$

under the constraint that $(X^{y_0,\mathbf{z}}, Y^{y_0,\mathbf{z}})$ solve: $\forall t \in [0, T]$

$$\begin{cases} dX_t = B(t, X_t, \mathcal{L}(X_t), Y_t)dt + dW_t, & X_0 \sim m_0, \\ dY_t = -F(t, X_t, \mathcal{L}(X_t), Y_t)dt + z(t, X_t) \cdot dW_t, & Y_0 = y_0(X_0). \end{cases}$$

Shooting Method for MKV FBSDE

Solutions of MFG (and MFC) can be characterized by **MKV FBSDEs** of the form

$$\begin{cases} dX_t = B(t, X_t, \mathcal{L}(X_t), Y_t)dt + dW_t, & X_0 \sim m_0 & \to \text{state} \\ dY_t = -F(t, X_t, \mathcal{L}(X_t), Y_t)dt + Z_t \cdot dW_t, & Y_T = G(X_T, \mathcal{L}(X_T)) & \to \text{control/cost} \end{cases}$$

(stemming from sto. Pontryagin's or Bellman's principle: $F = f$ or $F = \partial_x H$)

Shooting: Guess Y_0 and $(Z_t)_t$ [Ma,Yong; Sannikov; Han, Jentzen, E'17; ...]
$\to$ recover sol. (X, Y, Z) is found by opt. control of 2 **forward** SDEs

Reformulation as a **MFC problem**

Minimize over $y_0(\cdot)$ and $\mathbf{z}(\cdot) = (z_t(\cdot))_{t \geq 0}$

$$\mathfrak{J}(y_0(\cdot), \mathbf{z}(\cdot)) = \mathbb{E}\left[\| Y_T^{y_0, \mathbf{z}} - G(X_T^{y_0, \mathbf{z}}, \mathcal{L}(X_T^{y_0, \mathbf{z}})) \|^2 \right],$$

under the constraint that $(X^{y_0, \mathbf{z}}, Y^{y_0, \mathbf{z}})$ solve: $\forall t \in [0, T]$

$$\begin{cases} dX_t = B(t, X_t, \mathcal{L}(X_t), Y_t)dt + dW_t, & X_0 \sim m_0, \\ dY_t = -F(t, X_t, \mathcal{L}(X_t), Y_t)dt + z(t, X_t) \cdot dW_t, & Y_0 = y_0(X_0). \end{cases}$$

$\to$ MFC problem: apply previous method, replacing $y_0(\cdot), z(\cdot, \cdot)$ by NN

Shooting Method for MKV FBSDE

Solutions of MFG (and MFC) can be characterized by **MKV FBSDEs** of the form

$$\begin{cases} dX_t = B(t, X_t, \mathcal{L}(X_t), Y_t)dt + dW_t, & X_0 \sim m_0 & \rightarrow \text{state} \\ dY_t = -F(t, X_t, \mathcal{L}(X_t), Y_t)dt + Z_t \cdot dW_t, & Y_T = G(X_T, \mathcal{L}(X_T)) & \rightarrow \text{control/cost} \end{cases}$$

(stemming from sto. Pontryagin's or Bellman's principle: $F = f$ or $F = \partial_x H$)

Shooting: Guess Y_0 and $(Z_t)_t$ [Ma,Yong; Sannikov; Han, Jentzen, E'17; ...]
$\rightarrow$ recover sol. (X, Y, Z) is found by opt. control of 2 **forward** SDEs

Reformulation as a **MFC problem**

Minimize over $y_0(\cdot)$ and $\mathbf{z}(\cdot) = (z_t(\cdot))_{t \geq 0}$

$$\mathfrak{J}(y_0(\cdot), \mathbf{z}(\cdot)) = \mathbb{E}\left[\|Y_T^{y_0,\mathbf{z}} - G(X_T^{y_0,\mathbf{z}}, \mathcal{L}(X_T^{y_0,\mathbf{z}}))\|^2 \right],$$

under the constraint that $(X^{y_0,\mathbf{z}}, Y^{y_0,\mathbf{z}})$ solve: $\forall t \in [0, T]$

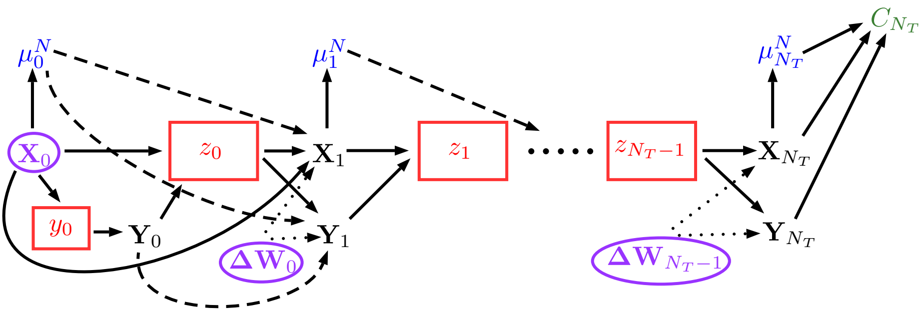
$$\begin{cases} dX_t = B(t, X_t, \mathcal{L}(X_t), Y_t)dt + dW_t, & X_0 \sim m_0, \\ dY_t = -F(t, X_t, \mathcal{L}(X_t), Y_t)dt + z(t, X_t) \cdot dW_t, & Y_0 = y_0(X_0). \end{cases}$$

$\rightarrow$ MFC problem: apply previous method, replacing $y_0(\cdot), z(\cdot, \cdot)$ by NN

NB: This problem is *not* the original MFG or MFC

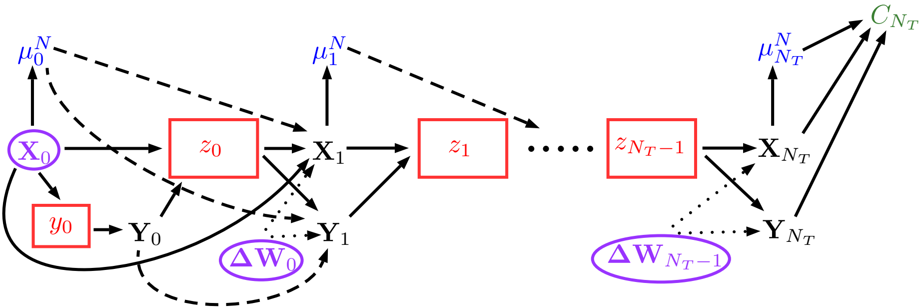
Analysis?

Implementation



- **Inputs:** initial positions $\mathbf{X}_0 = (X_0^i)_i$, BM increments: $\mathbf{\Delta W}_n = (\Delta W_n^i)_i$, for all n
- **Loss function:** total cost = C_{N_T} $=$ terminal penalty; state = (X_n, Y_n)
- **SGD** to optimize over the param. θ_y, θ_z of 2 NN for $y_{\theta_y}(\cdot) \approx y_0(\cdot), z_{\theta_z}(\cdot, \cdot) \approx z(\cdot, \cdot)$

Implementation



- **Inputs:** initial positions $\mathbf{X}_0 = (X_0^i)_i$, BM increments: $\mathbf{\Delta W}_n = (\Delta W_n^i)_i$, for all n
- **Loss function:** total cost = C_{N_T} = terminal penalty; state = (X_n, Y_n)
- **SGD** to optimize over the param. θ_y, θ_z of 2 NN for $y_{\theta_y}(\cdot) \approx y_0(\cdot), z_{\theta_z}(\cdot,\cdot) \approx z(\cdot,\cdot)$
- Alternative implementation: $1 + N_T$ NNs for $y_0(\cdot), z_0(\cdot), \dots, z_{N_T-1}(\cdot)$

Numerical Illustration 1: Comparison with Picard Solver

Example of MKV FBSDE from [Chassagneux *et al.*'17] ($\rho =$ coupling parameter)

$$dX_t = -\rho Y_t dt + \sigma dW_t, \qquad X_0 = x_0$$
$$dY_t = \operatorname{atan}(\mathbb{E}[X_t])dt + Z_t dW_t, \qquad Y_T = G'(X_T) := \operatorname{atan}(X_T)$$

Comes from the **MFG** defined by $dX_t^v = v_t dt + dW_t$ and

$$J(v;\mu) = \mathbb{E}\left[G(X_T^v) + \int_0^T \left(\frac{1}{2\rho} v_t^2 + X_t^v \operatorname{atan}\left(\int x\mu_t(dx)\right)\right) dt\right]$$

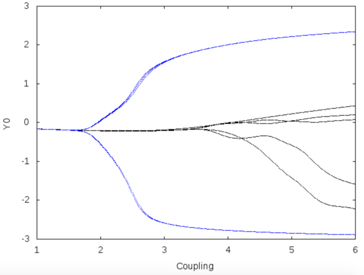
Numerical Illustration 1: Comparison with Picard Solver

Example of MKV FBSDE from [Chassagneux *et al.*'17] (ρ = coupling parameter)

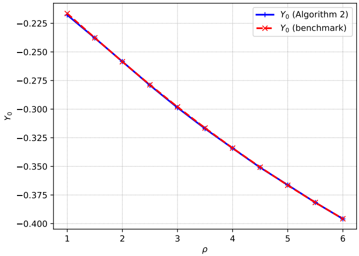
$$dX_t = -\rho Y_t dt + \sigma dW_t, \qquad X_0 = x_0$$
$$dY_t = \mathrm{atan}(\mathbb{E}[X_t])dt + Z_t dW_t, \qquad Y_T = G'(X_T) := \mathrm{atan}(X_T)$$

Comes from the **MFG** defined by $dX_t^v = v_t dt + dW_t$ and

$$J(v;\mu) = \mathbb{E}\left[G(X_T^v) + \int_0^T \left(\frac{1}{2\rho} v_t^2 + X_t^v \mathrm{atan}\left(\int x\mu_t(dx)\right)\right)dt\right]$$



Results from [Chassagneux *et al.*]



NN (FBSDE system)

(More details in [Carmona, L.'19])

Numerical Illustration 2: LQ MFG with Common Noise

Example: MFG for inter-bank borrowing/lending [Carmona, Fouque, Sun]

X = log-monetary reserve, v = rate of borrowing/lending to central bank, cost:

$$J(v;\bar{m}) = \mathbb{E}\left[\int_0^T \left[\frac{1}{2}v_t^2 - q v_t(\bar{m}_t - X_t) + \frac{\epsilon}{2}(\bar{m}_t - X_t)^2\right] dt + \frac{c}{2}(\bar{m}_T - X_T)^2\right]$$

where $\bar{m} = (\bar{m}_t)_{t\geq 0}$ = conditional mean of the population states given W^0, and

$$dX_t = [a(\bar{m}_t - X_t) + v_t]dt + \sigma\left(\sqrt{1-\rho^2}dW_t + \rho\, dW_t^0\right)$$

Numerical Illustration 2: LQ MFG with Common Noise

Example: MFG for inter-bank borrowing/lending [Carmona, Fouque, Sun]

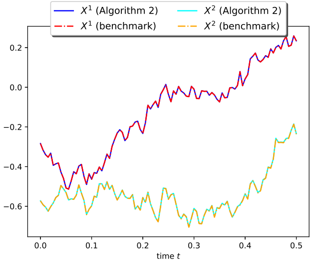
X = log-monetary reserve, v = rate of borrowing/lending to central bank, cost:

$$J(v;\bar m) = \mathbb{E}\left[\int_0^T \left[\frac{1}{2}v_t^2 - q v_t(\bar m_t - X_t) + \frac{\epsilon}{2}(\bar m_t - X_t)^2\right] dt + \frac{c}{2}(\bar m_T - X_T)^2\right]$$

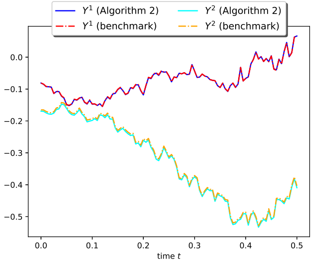
where $\bar m = (\bar m_t)_{t\ge 0}$ = conditional mean of the population states given W^0, and

$$dX_t = [a(\bar m_t - X_t) + v_t]dt + \sigma\left(\sqrt{1-\rho^2}dW_t + \rho\, dW_t^0\right)$$

NN for FBSDE system VS (semi) analytical solution (LQ structure)



Samples of X



Samples of Y

(More details in [Carmona, L.'19])

Numerical Illustration 2: LQ MFG with Common Noise

Example: MFG for inter-bank borrowing/lending [Carmona, Fouque, Sun]

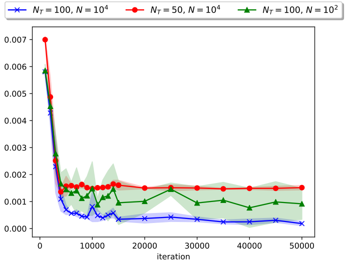
X = log-monetary reserve, v = rate of borrowing/lending to central bank, cost:

$$J(v;\bar{m}) = \mathbb{E}\left[\int_0^T \left[\frac{1}{2}v_t^2 - q v_t(\bar{m}_t - X_t) + \frac{\epsilon}{2}(\bar{m}_t - X_t)^2\right] dt + \frac{c}{2}(\bar{m}_T - X_T)^2\right]$$

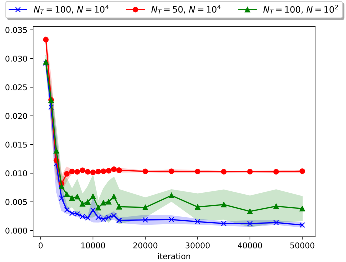
where $\bar{m} = (\bar{m}_t)_{t\geq 0}$ = conditional mean of the population states given W^0, and

$$dX_t = [a(\bar{m}_t - X_t) + v_t]dt + \sigma\left(\sqrt{1-\rho^2}dW_t + \rho\, dW_t^0\right)$$

NN for FBSDE system VS (semi) analytical solution (LQ structure)



L^2 error on X



L^2 error on Y

(More details in [Carmona, L.'19])

Code Samples

- Deep learning (Policy Gradient) for Mean Field Control / MKV control:

  `https://colab.research.google.com/drive/1Di1gP3W6rXXgIVoRqUxLmNyvUYdwQ9XO?usp=sharing`

- Deep learning for MKV FBSDE via shooting method:

  `https://colab.research.google.com/drive/1OMkjzbHorLDyQbQ13vW2nEcQAOsK9s-a?usp=sharing`

- ...

Outline

Methods Based on Dynamic Programming – NNContPI

Method (NNContPI) [Bachouch, Huré, Langrené, Pham'21][4] to minimize:

$$J^{N_T}(v) = \mathbb{E}\left[\sum_{n=0}^{N_T-1} f(X_n, v_n(X_n)) + g(X_{N_T})\right]$$

where $$X_{n+1} = X_n + b(X_n, v_n(X_n)) + \epsilon_{n+1}.$$

[4] Bachouch, A., Huré, C., Langrené, N., & Pham, H. (2021). Deep neural networks algorithms for stochastic control problems on finite horizon: numerical applications. *Methodology and Computing in Applied Probability*, 1-36.

Method (NNContPI) [Bachouch, Huré, Langrené, Pham'21][4] to minimize:

$$J^{N_T}(v) = \mathbb{E}\left[\sum_{n=0}^{N_T-1} f(X_n, v_n(X_n)) + g(X_{N_T})\right]$$

where $$X_{n+1} = X_n + b(X_n, v_n(X_n)) + \epsilon_{n+1}.$$

Input: Training distributions $(\mu_n)_{n=0,\dots,N_T}$

Output: Parameters $(\theta^\star_n)_{n=0,\dots,N_T}$ s.t. $(\varphi_{\theta^\star_n})_{n=0,\dots,N_T}$ (approximately) minimizes J^{N_T}

for $\mathtt{n} = N_T-1, N_T-2, \dots, 1, 0$ **do**

Compute (e.g., using SGD) θ^*_n minimizing:

$$\theta \mapsto \mathbb{E}\left[f(X_n, \varphi_{\theta_n}(X_n)) + \sum_{n'=n+1}^{N_T-1} f(X^\theta_{n'}, \varphi_{\theta^*_{n'}}(X^\theta_{n'})) + g(X^v_{N_T})\right]$$

where $X_n \sim \mu_n$ and

$$\begin{cases} X^\theta_{n+1} = X^\theta_n + b(X^\theta_n, \varphi_{\theta_n}(X^\theta_n)) + \epsilon_{n+1}, \\ X^\theta_{n'+1} = X^\theta_{n'} + b(X^\theta_{n'}, \varphi_{\theta^*_{n'}}(X^\theta_{n'})) + \epsilon_{n'+1}, \qquad n' > n. \end{cases}$$

return $(\theta^*_n)_{n=0,\dots,N_T-1}$

[4] Bachouch, A., Huré, C., Langrené, N., & Pham, H. (2021). Deep neural networks algorithms for stochastic control problems on finite horizon: numerical applications. *Methodology and Computing in Applied Probability*, 1-36.

Methods Based on Dynamic Programming – Hybrid-Now

Method (Hybrid-Now) [Bachouch, Huré, Langrené, Pham'21] to minimize:

$$J^{N_T}(v) = \mathbb{E}\left[\sum_{n=0}^{N_T-1} f(X_n, v_n(X_n)) + g(X_{N_T})\right]$$

where $$X_{n+1} = X_n + b(X_n, v_n(X_n)) + \epsilon_{n+1}.$$

Methods Based on Dynamic Programming – Hybrid-Now

Method (Hybrid-Now) [Bachouch, Huré, Langrené, Pham'21] to minimize:

$$J^{N_T}(v) = \mathbb{E}\left[\sum_{n=0}^{N_T-1} f(X_n, v_n(X_n)) + g(X_{N_T})\right]$$

$$\text{where} \qquad X_{n+1} = X_n + b(X_n, v_n(X_n)) + \epsilon_{n+1}.$$

Value function $V_n(x) = \inf_v \mathbb{E}\left[\sum_{n'=n}^{N_T-1} f(X_{n'}, v_{n'}(X_{n'})) + g(X_{N_T})\right]$

Methods Based on Dynamic Programming – Hybrid-Now

Method (Hybrid-Now) [Bachouch, Huré, Langrené, Pham'21] to minimize:

$$J^{N_T}(v) = \mathbb{E}\left[\sum_{n=0}^{N_T-1} f(X_n, v_n(X_n)) + g(X_{N_T})\right]$$

where $\quad X_{n+1} = X_n + b(X_n, v_n(X_n)) + \epsilon_{n+1}.$

Value function $V_n(x) = \inf_v \mathbb{E}\left[\sum_{n'=n}^{N_T-1} f(X_{n'}, v_{n'}(X_{n'})) + g(X_{N_T})\right]$

Input: Training distributions $(\mu_n)_{n=0,\dots,N_T}$

Output: Parameters $(\theta_n^\star)_{n=0,\dots,N_T}$ s.t. $(\varphi_{\theta_n^\star})_{n=0,\dots,N_T}$ (approximately) minimizes J^{N_T}; Parameters $(\omega_n^*)_{n=0,\dots,N_T}$ such that $\psi_{\omega_n^*}$ approximates the value function V_n at time n

Set $\hat{V}_{N_T} = g$

for $\mathtt{n} = N_T - 1, N_T - 2, \dots, 1, 0$ **do**

Compute θ_n^* minimizing:

$$\theta \mapsto \mathbb{E}\left[f(X_n, \varphi_{\theta_n}(X_n)) + \hat{V}_{n+1}(X_{n+1}^\theta)\right]$$

where $X_n \sim \mu_n$ and $X_{n+1}^\theta = X_n^\theta + b(X_n^\theta, \varphi_{\theta_n}(X_n^\theta)) + \epsilon_{n+1}$

Methods Based on Dynamic Programming – Hybrid-Now

Method (Hybrid-Now) [Bachouch, Huré, Langrené, Pham'21] to minimize:

$$J^{N_T}(v) = \mathbb{E}\left[\sum_{n=0}^{N_T-1} f(X_n, v_n(X_n)) + g(X_{N_T})\right]$$

where $\quad X_{n+1} = X_n + b(X_n, v_n(X_n)) + \epsilon_{n+1}.$

Value function $V_n(x) = \inf_v \mathbb{E}\left[\sum_{n'=n}^{N_T-1} f(X_{n'}, v_{n'}(X_{n'})) + g(X_{N_T})\right]$

Input: Training distributions $(\mu_n)_{n=0,\dots,N_T}$

Output: Parameters $(\theta_n^\star)_{n=0,\dots,N_T}$ s.t. $(\varphi_{\theta_n^\star})_{n=0,\dots,N_T}$ (approximately) minimizes J^{N_T}; Parameters $(\omega_n^*)_{n=0,\dots,N_T}$ such that $\psi_{\omega_n^*}$ approximates the value function V_n at time n

1 Set $\hat{V}_{N_T} = g$

2 **for** $\mathtt{n} = N_T - 1, N_T - 2, \dots, 1, 0$ **do**

3 Compute θ_n^* minimizing:

$$\theta \mapsto \mathbb{E}\left[f(X_n, \varphi_{\theta_n}(X_n)) + \hat{V}_{n+1}(X_{n+1}^\theta)\right]$$

where $X_n \sim \mu_n$ and $X_{n+1}^\theta = X_n^\theta + b(X_n^\theta, \varphi_{\theta_n}(X_n^\theta)) + \epsilon_{n+1}$

4 Compute ω_n^* minimizing:

$$\mathbb{E}\left[\left|f(X_n, \varphi_{\theta_n^*}(X_n)) + \hat{V}_{n+1}(X_{n+1}^{\theta_n^*}) - \psi_{\omega_n^*}(X_n)\right|^2\right]$$

5 **return** $(\theta_n^*)_{n=0,\dots,N_T-1}$, $(\omega_n^*)_{n=0,\dots,N_T}$

Methods Based on Dynamic Programming – DBDP

Deep Backward Dynamic Programming (DBDP) [Huré, Pham, Warin'20][5]

Idea: learn Y_n and Z_n at each n as functions of X_n, backward in time:

- Initialize $\hat{Y}_{N_T} = g$ and then, for $n = N_T - 1, \ldots, 0$, either:
- Version 1: Let $(\hat{Y}_n, \hat{Z}_n)$ = minimizer over (Y_n, Z_n) of:

$$\mathbb{E}\left[|\hat{Y}_{n+1}(X_{n+1}) - Y_n(X_n) - f(t_n, X_n, Y_n(X_n), Z_n(X_n))\Delta t - Z_n(X_n) \cdot \Delta W_{n+1}|\right]$$

[5] Huré, C., Pham, H. & Warin, X. . Deep backward schemes for highdimensional nonlinear PDEs. In: *Math. Comp.* 89.324 (2020), pp. 1547– 1580.

[6] Germain, M, Pham, H., & Warin, X.. Neural networks-based algorithms for stochastic control and PDEs in finance. arXiv preprint arXiv:2101.08068 (2021).

Methods Based on Dynamic Programming – DBDP

Deep Backward Dynamic Programming (DBDP) [Huré, Pham, Warin'20][5]

Idea: learn Y_n and Z_n at each n as functions of X_n, backward in time:

- Initialize $\hat{Y}_{N_T} = g$ and then, for $n = N_T - 1, \ldots, 0$, either:
- Version 1: Let $(\hat{Y}_n, \hat{Z}_n)$ = minimizer over (Y_n, Z_n) of:

$$\mathbb{E}\left[|\hat{Y}_{n+1}(X_{n+1}) - Y_n(X_n) - f(t_n, X_n, Y_n(X_n), Z_n(X_n))\Delta t - Z_n(X_n) \cdot \Delta W_{n+1}|\right]$$

- or Version 2: Let $(\hat{Y}_n, \hat{Z}_n)$ = minimizer over (Y_n, Z_n) of:

$$\mathbb{E}\left[|\hat{Y}_{n+1}(X_{n+1}) - Y_n(X_n) - f(t_n, X_n, Y_n(X_n), \sigma^\top D_x Y_n(X_n))\Delta t - D_x Y_n(X_n)^\top \sigma \Delta W_{n+1}|\right]$$

[5] Huré, C., Pham, H. & Warin, X. . Deep backward schemes for highdimensional nonlinear PDEs. In: *Math. Comp.* 89.324 (2020), pp. 1547– 1580.

[6] Germain, M, Pham, H., & Warin, X.. Neural networks-based algorithms for stochastic control and PDEs in finance. arXiv preprint arXiv:2101.08068 (2021).

Methods Based on Dynamic Programming – DBDP

Deep Backward Dynamic Programming (DBDP) [Huré, Pham, Warin'20][5]

Idea: learn Y_n and Z_n at each n as functions of X_n, backward in time:

- Initialize $\hat{Y}_{N_T} = g$ and then, for $n = N_T - 1, \ldots, 0$, either:
- Version 1: Let $(\hat{Y}_n, \hat{Z}_n)$ = minimizer over (Y_n, Z_n) of:

$$\mathbb{E}\left[|\hat{Y}_{n+1}(X_{n+1}) - Y_n(X_n) - f(t_n, X_n, Y_n(X_n), Z_n(X_n))\Delta t - Z_n(X_n) \cdot \Delta W_{n+1}|\right]$$

- or Version 2: Let $(\hat{Y}_n, \hat{Z}_n)$ = minimizer over (Y_n, Z_n) of:

$$\mathbb{E}\left[|\hat{Y}_{n+1}(X_{n+1}) - Y_n(X_n) - f(t_n, X_n, Y_n(X_n), \sigma^\top D_x Y_n(X_n))\Delta t - D_x Y_n(X_n)^\top \sigma \Delta W_{n+1}|\right]$$

For more details on deep learning methods for (non-mean field) optimal control problems, see e.g. [Germain, Pham, Warin'21][6]

[5] Huré, C., Pham, H. & Warin, X. . Deep backward schemes for highdimensional nonlinear PDEs. In: *Math. Comp.* 89.324 (2020), pp. 1547– 1580.

[6] Germain, M, Pham, H., & Warin, X.. Neural networks-based algorithms for stochastic control and PDEs in finance. arXiv preprint arXiv:2101.08068 (2021).

Summary