Mean Field Games: Numerical Methods and Applications in Machine Learning

Part 6: Deep Learning for MFG PDEs

Mathieu Laurière

https://mlauriere.github.io/teaching/MFG-PKU-6.pdf

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

RECAP

Outline

1. Deep Galerkin Method for MFG PDEs
   - Warm-up: ODE
   - Solving MFG PDE system
   - Link with Generative Adversarial Networks

2. Master Equation

Outline

1. Deep Galerkin Method for MFG PDEs
   - Warm-up: ODE
   - Solving MFG PDE system
   - Link with Generative Adversarial Networks

2. Master Equation

- Look for $\varphi : \mathbb{R} \ni x \mapsto \varphi(x) \in \mathbb{R}$ s.t.

$$\begin{cases} F(x, \varphi(x), \varphi'(x), \dots) = 0, & x \in [a, b] \\ G(a, \varphi(a), \varphi'(a), \dots) = 0 \end{cases}$$

Solving ODEs with Neural Networks

- Look for $\varphi : \mathbb{R} \ni x \mapsto \varphi(x) \in \mathbb{R}$ s.t.

$$\begin{cases} F(x, \varphi(x), \varphi'(x), \dots) = 0, & x \in [a, b] \\ G(a, \varphi(a), \varphi'(a), \dots) = 0 \end{cases}$$

- Look among NN φ_θ

$$\begin{cases} F(x, \varphi_\theta(x), \varphi_\theta'(x), \dots) = 0, & x \in [a, b] \\ G(a, \varphi_\theta(a), \varphi_\theta'(a), \dots) = 0 \end{cases}$$

Solving ODEs with Neural Networks

- Look for $\varphi : \mathbb{R} \ni x \mapsto \varphi(x) \in \mathbb{R}$ s.t.

$$\begin{cases} F(x, \varphi(x), \varphi'(x), \dots) = 0, & x \in [a, b] \\ G(a, \varphi(a), \varphi'(a), \dots) = 0 \end{cases}$$

- Look among NN φ_θ

$$\begin{cases} F(x, \varphi_\theta(x), \varphi_\theta'(x), \dots) = 0, & x \in [a, b] \\ G(a, \varphi_\theta(a), \varphi_\theta'(a), \dots) = 0 \end{cases}$$

- Rephrase as minimization problem: minimizer over θ

$$\mathbb{E}_{X \sim \mathcal{U}([a,b])} \left[\left| F(X, \varphi_\theta(X), \varphi_\theta'(X), \dots) \right|^2 \right] \\ + \left| G(a, \varphi_\theta(a), \varphi_\theta'(a), \dots) \right|^2$$

Solving ODEs with Neural Networks

- Look for $\varphi : \mathbb{R} \ni x \mapsto \varphi(x) \in \mathbb{R}$ s.t.

$$\begin{cases} F(x, \varphi(x), \varphi'(x), \dots) = 0, & x \in [a,b] \\ G(a, \varphi(a), \varphi'(a), \dots) = 0 \end{cases}$$

- Look among NN φ_θ

$$\begin{cases} F(x, \varphi_\theta(x), \varphi_\theta'(x), \dots) = 0, & x \in [a,b] \\ G(a, \varphi_\theta(a), \varphi_\theta'(a), \dots) = 0 \end{cases}$$

- Rephrase as minimization problem: minimizer over θ

$$\mathbb{E}_{X \sim \mathcal{U}([a,b])} \left[\left| F(X, \varphi_\theta(X), \varphi_\theta'(X), \dots) \right|^2 \right] + \left| G(a, \varphi_\theta(a), \varphi_\theta'(a), \dots) \right|^2$$

- Use SGD

Numerical Illustration

Application to the ODE:

$$\begin{cases} F(x,\varphi(x),\varphi'(x)) = \varphi'(x) - (x-\varphi(x)), & x \in [0,5] \\ \varphi(0) = 1 \end{cases}$$

Numerical Illustration

Application to the ODE:

$$\begin{cases} F(x, \varphi(x), \varphi'(x)) = \varphi'(x) - (x - \varphi(x)), & \qquad x \in [0, 5] \\ \varphi(0) = 1 \end{cases}$$

Solution:

$$\varphi(x) = x - 1 + 2e^{-x}$$

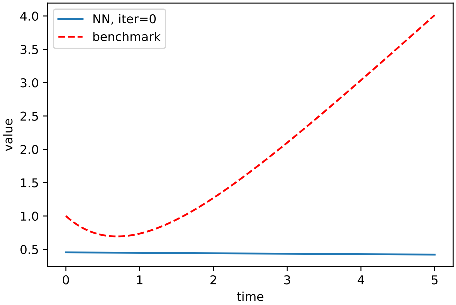
Numerical Illustration

Application to the ODE:

$$\begin{cases} F(x, \varphi(x), \varphi'(x)) = \varphi'(x) - (x - \varphi(x)), & x \in [0, 5] \\ \varphi(0) = 1 \end{cases}$$

Solution:

$$\varphi(x) = x - 1 + 2e^{-x}$$



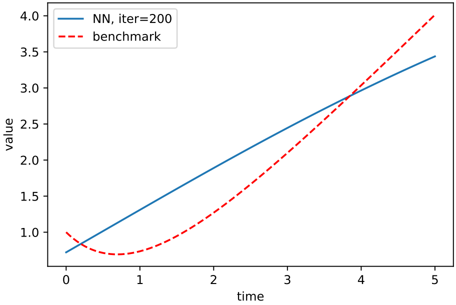
Numerical Illustration

Application to the ODE:

$$\begin{cases} F(x, \varphi(x), \varphi'(x)) = \varphi'(x) - (x - \varphi(x)), & x \in [0, 5] \\ \varphi(0) = 1 \end{cases}$$

Solution:

$$\varphi(x) = x - 1 + 2e^{-x}$$



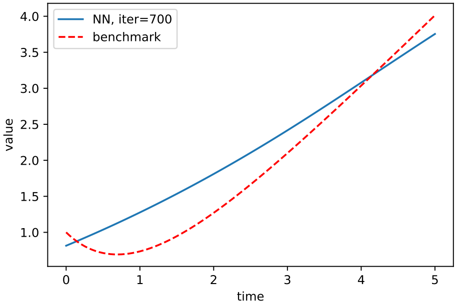
Numerical Illustration

Application to the ODE:

$$\begin{cases} F(x, \varphi(x), \varphi'(x)) = \varphi'(x) - (x - \varphi(x)), & x \in [0, 5] \\ \varphi(0) = 1 \end{cases}$$

Solution:

$$\varphi(x) = x - 1 + 2e^{-x}$$



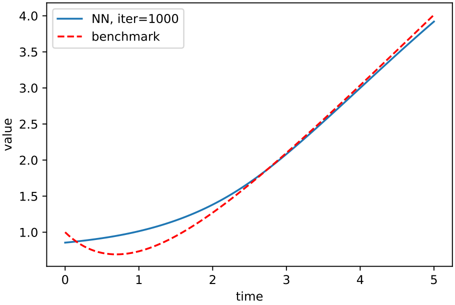
Numerical Illustration

Application to the ODE:

$$\begin{cases} F(x,\varphi(x),\varphi'(x)) = \varphi'(x) - (x-\varphi(x)), & x \in [0,5] \\ \varphi(0) = 1 \end{cases}$$

Solution:

$$\varphi(x) = x - 1 + 2e^{-x}$$



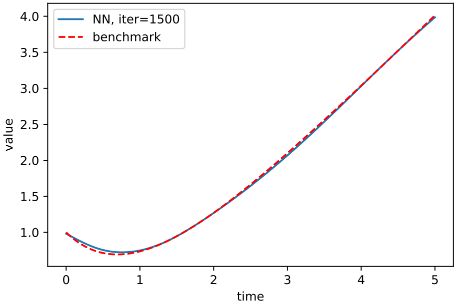
Numerical Illustration

Application to the ODE:

$$\begin{cases} F(x,\varphi(x),\varphi'(x)) = \varphi'(x) - (x - \varphi(x)), & x \in [0,5] \\ \varphi(0) = 1 \end{cases}$$

Solution:

$$\varphi(x) = x - 1 + 2e^{-x}$$



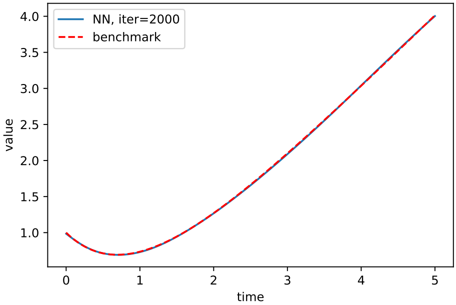
Numerical Illustration

Application to the ODE:

$$\begin{cases} F(x, \varphi(x), \varphi'(x)) = \varphi'(x) - (x - \varphi(x)), & x \in [0, 5] \\ \varphi(0) = 1 \end{cases}$$

Solution:

$$\varphi(x) = x - 1 + 2e^{-x}$$



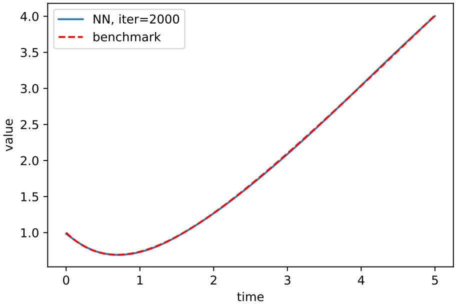
Numerical Illustration

Application to the ODE:

$$\begin{cases} F(x, \varphi(x), \varphi'(x)) = \varphi'(x) - (x - \varphi(x)), & x \in [0,5] \\ \varphi(0) = 1 \end{cases}$$

Solution:

$$\varphi(x) = x - 1 + 2e^{-x}$$



https://colab.research.google.com/drive/1LHuV1oE6eyO6AQgw3joQjow_uozQWSTw?usp=sharing

Outline

Solving PDEs with Neural Networks

Deep Galerkin Method (DGM), proposed by [Sirignano, Spiliopoulos][1]

- Look for $\varphi : \mathbb{R}^d \ni x \mapsto \varphi(x) \in \mathbb{R}$ s.t.

$$\begin{cases} F(x, \varphi(x), D\varphi(x), D^2\varphi(x), \dots) = 0, & x \in \Omega \\ G(x, \varphi(x), D\varphi(x), D^2\varphi(x), \dots) = 0, & x \in \partial\Omega \end{cases}$$

[1] Sirignano, J., & Spiliopoulos, K. (2018). DGM: A deep learning algorithm for solving partial differential equations. *Journal of computational physics*, 375, 1339-1364.

Solving PDEs with Neural Networks

Deep Galerkin Method (DGM), proposed by [Sirignano, Spiliopoulos][1]

- Look for $\varphi : \mathbb{R}^d \ni x \mapsto \varphi(x) \in \mathbb{R}$ s.t.

$$\begin{cases} F(x, \varphi(x), D\varphi(x), D^2\varphi(x), \dots) = 0, & x \in \Omega \\ G(x, \varphi(x), D\varphi(x), D^2\varphi(x), \dots) = 0, & x \in \partial\Omega \end{cases}$$

- Look among NN φ_θ

$$\begin{cases} F(x, \varphi_\theta(x), D\varphi_\theta(x), D^2\varphi_\theta(x), \dots) = 0, & x \in \Omega \\ G(x, \varphi_\theta(x), D\varphi_\theta(x), D^2\varphi_\theta(x), \dots) = 0, & x \in \partial\Omega \end{cases}$$

[1] Sirignano, J., & Spiliopoulos, K. (2018). DGM: A deep learning algorithm for solving partial differential equations. *Journal of computational physics*, 375, 1339-1364.

Solving PDEs with Neural Networks

Deep Galerkin Method (DGM), proposed by [Sirignano, Spiliopoulos][1]

- Look for $\varphi : \mathbb{R}^d \ni x \mapsto \varphi(x) \in \mathbb{R}$ s.t.

$$\begin{cases} F(x, \varphi(x), D\varphi(x), D^2\varphi(x), \dots) = 0, & x \in \Omega \\ G(x, \varphi(x), D\varphi(x), D^2\varphi(x), \dots) = 0, & x \in \partial\Omega \end{cases}$$

- Look among NN φ_θ

$$\begin{cases} F(x, \varphi_\theta(x), D\varphi_\theta(x), D^2\varphi_\theta(x), \dots) = 0, & x \in \Omega \\ G(x, \varphi_\theta(x), D\varphi_\theta(x), D^2\varphi_\theta(x), \dots) = 0, & x \in \partial\Omega \end{cases}$$

- Rephrase as minimization problem: minimizer over θ

$$\mathbb{E}_{X\sim\mathcal{U}(\Omega)}\left[|F(X, \varphi_\theta(X), D\varphi_\theta(X), D^2\varphi_\theta(X), \dots)|^2\right]$$
$$+ \mathbb{E}_{Y\sim\mathcal{U}(\partial\Omega)}\left[|G(Y, \varphi_\theta(Y), D\varphi_\theta(Y), D^2\varphi_\theta(Y), \dots)|^2\right]$$

[1] Sirignano, J., & Spiliopoulos, K. (2018). DGM: A deep learning algorithm for solving partial differential equations. *Journal of computational physics*, 375, 1339-1364.

Solving PDEs with Neural Networks

Deep Galerkin Method (DGM), proposed by [Sirignano, Spiliopoulos][1]

- Look for $\varphi : \mathbb{R}^d \ni x \mapsto \varphi(x) \in \mathbb{R}$ s.t.

$$\begin{cases} F(x, \varphi(x), D\varphi(x), D^2\varphi(x), \dots) = 0, & x \in \Omega \\ G(x, \varphi(x), D\varphi(x), D^2\varphi(x), \dots) = 0, & x \in \partial\Omega \end{cases}$$

- Look among NN φ_θ

$$\begin{cases} F(x, \varphi_\theta(x), D\varphi_\theta(x), D^2\varphi_\theta(x), \dots) = 0, & x \in \Omega \\ G(x, \varphi_\theta(x), D\varphi_\theta(x), D^2\varphi_\theta(x), \dots) = 0, & x \in \partial\Omega \end{cases}$$

- Rephrase as minimization problem: minimizer over θ

$$\mathbb{E}_{X \sim \mathcal{U}(\Omega)} \left[\left| F(X, \varphi_\theta(X), D\varphi_\theta(X), D^2\varphi_\theta(X), \dots) \right|^2 \right] + \mathbb{E}_{Y \sim \mathcal{U}(\partial\Omega)} \left[\left| G(Y, \varphi_\theta(Y), D\varphi_\theta(Y), D^2\varphi_\theta(Y), \dots) \right|^2 \right]$$

- Use SGD

[1] Sirignano, J., & Spiliopoulos, K. (2018). DGM: A deep learning algorithm for solving partial differential equations. *Journal of computational physics*, 375, 1339-1364.

Solving PDEs with Neural Networks

Deep Galerkin Method (DGM), proposed by [Sirignano, Spiliopoulos][1]

- Look for $\varphi : \mathbb{R}^d \ni x \mapsto \varphi(x) \in \mathbb{R}$ s.t.

$$\begin{cases} F(x, \varphi(x), D\varphi(x), D^2\varphi(x), \dots) = 0, & x \in \Omega \\ G(x, \varphi(x), D\varphi(x), D^2\varphi(x), \dots) = 0, & x \in \partial\Omega \end{cases}$$

- Look among NN φ_θ

$$\begin{cases} F(x, \varphi_\theta(x), D\varphi_\theta(x), D^2\varphi_\theta(x), \dots) = 0, & x \in \Omega \\ G(x, \varphi_\theta(x), D\varphi_\theta(x), D^2\varphi_\theta(x), \dots) = 0, & x \in \partial\Omega \end{cases}$$

- Rephrase as minimization problem: minimizer over θ

$$\mathbb{E}_{X\sim\mathcal{U}(\Omega)}\left[|F(X, \varphi_\theta(X), D\varphi_\theta(X), D^2\varphi_\theta(X), \dots)|^2\right]$$
$$+\,\mathbb{E}_{Y\sim\mathcal{U}(\partial\Omega)}\left[|G(Y, \varphi_\theta(Y), D\varphi_\theta(Y), D^2\varphi_\theta(Y), \dots)|^2\right]$$

- Use SGD
- Remarks on the implementation:
  - Choice of distribution
  - Boundary conditions
  - Higher order derivatives computation

[1] Sirignano, J., & Spiliopoulos, K. (2018). DGM: A deep learning algorithm for solving partial differential equations. *Journal of computational physics*, 375, 1339-1364.

DGM Architecture

- Let $\vec{x} = (t, x)$ be the input
- Architecture: $L + 1$ hidden layers ($\odot$ denotes element-wise multiplication):

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

- The parameters are

$$
\theta = \left\{ W^1, b^1, \left(U^{\alpha,\ell}, W^{\alpha,\ell}, b^{\alpha,\ell} \right)_{\ell=1,\ldots,L,\, \alpha \in \{z,g,r,h\}}, W, b \right\}.
$$

- The number of units in each layer is M and $\sigma : \mathbb{R}^M \to \mathbb{R}^M$ is an element-wise nonlinearity:

$$
\sigma(z) = \Big(\phi(z_1), \phi(z_2), \ldots, \phi(z_M) \Big),
$$

where $\phi : \mathbb{R} \to \mathbb{R}$ is a nonlinear activation function.

MFG PDE system

Reminder: (m,u) solving, on $[0,T]\times\mathbb{T}^d$,

$$\begin{cases} 0 = -\dfrac{\partial u}{\partial t}(t,x) - \nu\Delta u(t,x) + H(x, m(t,\cdot), \nabla u(t,x)) \\ 0 = \dfrac{\partial m}{\partial t}(t,x) - \nu\Delta m(t,x) - \operatorname{div}\left(m(t,\cdot)\partial_p H(\cdot, m(t), \nabla u(t,\cdot))\right)(x) \\ \quad u(T,x) = g(x, m(T,\cdot)), \qquad m(0,x) = m_0(x) \end{cases}$$

MFG PDE system

Reminder: (m, u) solving, on $[0, T] \times \mathbb{T}^d$,

$$\begin{cases} 0 = -\dfrac{\partial u}{\partial t}(t,x) - \nu \Delta u(t,x) + H(x, m(t,\cdot), \nabla u(t,x)) \\ 0 = \dfrac{\partial m}{\partial t}(t,x) - \nu \Delta m(t,x) - \operatorname{div}\left(m(t,\cdot)\partial_p H(\cdot, m(t), \nabla u(t,\cdot))\right)(x) \\ \; u(T,x) = g(x, m(T,\cdot)), \qquad m(0,x) = m_0(x) \end{cases}$$

Or ergodic version: (m, u, λ) on $\mathbb{T}^d$

$$\begin{cases} 0 = -\nu \Delta u(x) + H(x, m(\cdot), \nabla u(x)) + \lambda \\ 0 = -\nu \Delta m(x) - \operatorname{div}\left(m(\cdot)\partial_p H(\cdot, m, \nabla u(\cdot))\right)(x) \\ \displaystyle \int u(x)dx = 0, \qquad \int m(x)dx = 1, m > 0 \end{cases}$$

See [Lasry, Lions'07; BFY'13, Chapter 7]

MFG PDE system

Reminder: (m, u) solving, on $[0, T] \times \mathbb{T}^d$,

$$\begin{cases} 0 = -\dfrac{\partial u}{\partial t}(t, x) - \nu \Delta u(t, x) + H(x, m(t, \cdot), \nabla u(t, x)) \\ 0 = \dfrac{\partial m}{\partial t}(t, x) - \nu \Delta m(t, x) - \operatorname{div}\left(m(t, \cdot) \partial_p H(\cdot, m(t), \nabla u(t, \cdot))\right)(x) \\ \quad u(T, x) = g(x, m(T, \cdot)), \qquad m(0, x) = m_0(x) \end{cases}$$

Or ergodic version: (m, u, λ) on $\mathbb{T}^d$

$$\begin{cases} 0 = -\nu \Delta u(x) + H(x, m(\cdot), \nabla u(x)) + \lambda \\ 0 = -\nu \Delta m(x) - \operatorname{div}\left(m(\cdot) \partial_p H(\cdot, m, \nabla u(\cdot))\right)(x) \\ \quad \displaystyle\int u(x) dx = 0, \qquad \int m(x) dx = 1, m > 0 \end{cases}$$

See [Lasry, Lions'07; BFY'13, Chapter 7]

Analogous PDE systems for MFC problems

Numerical Illustration 1: Ergodic Example with Explicit Solution

Example (of MFC) with explicit solution on $\mathbb{T}^d$ ($d = 10$)
Following [Almulla *et al.*'17], take

$$f(x, m, v) = \frac{1}{2}|v|^2 + \tilde{f}(x) + \ln(m(x)),$$

with $\tilde{f}(x) = 2\pi^2 \left[- \sum_{i=1}^d c \sin(2\pi x_i) + \sum_{i=1}^d |c \cos(2\pi x_i)|^2 \right] - 2 \sum_{i=1}^d c \sin(2\pi x_i)$,
then the solution is given by $u(x) = c \sum_{i=1}^d \sin(2\pi x_i)$ and $m(x) = e^{2u(x)} / \int e^{2u}$

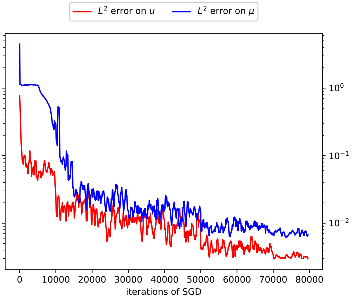
Numerical Illustration 1: Ergodic Example with Explicit Solution

Example (of MFC) with explicit solution on $\mathbb{T}^d$ ($d = 10$)
Following [Almulla *et al.*'17], take

$$f(x, m, v) = \frac{1}{2}|v|^2 + \tilde{f}(x) + \ln(m(x)),$$

with $\tilde{f}(x) = 2\pi^2 \left[-\sum_{i=1}^d c\sin(2\pi x_i) + \sum_{i=1}^d |c\cos(2\pi x_i)|^2 \right] - 2\sum_{i=1}^d c\sin(2\pi x_i)$,
then the solution is given by $u(x) = c\sum_{i=1}^d \sin(2\pi x_i)$ and $m(x) = e^{2u(x)} / \int e^{2u}$

Error vs SGD iterations (see [Carmona, L.'21]):



Relative L^2 error on u and m

Numerical Illustration 2: Ergodic Example without Explicit Solution

Example (of MFG) without explicit solution on $\mathbb{T}^d$ ($d = 30$)
Inspired by [Achdou, Capuzzo-Dolcetta'11], take

$$f(x, m, v) = \frac{1}{2}|v|^2 + \tilde{f}(x) + |m(x)|^2,$$

with $\tilde{f}(x) = 2\pi^2 c \sum_{i=1}^{d} [\sin(2\pi x_i) + \cos(2\pi x_i)]$

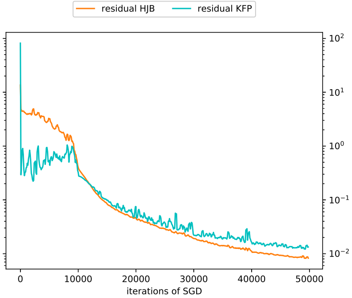
Numerical Illustration 2: Ergodic Example without Explicit Solution

Example (of MFG) without explicit solution on $\mathbb{T}^d$ ($d = 30$)
Inspired by [Achdou, Capuzzo-Dolcetta'11], take

$$f(x, m, v) = \frac{1}{2}|v|^2 + \tilde{f}(x) + |m(x)|^2,$$

with $\tilde{f}(x) = 2\pi^2 c \sum_{i=1}^{d} [\sin(2\pi x_i) + \cos(2\pi x_i)]$

PDE residuals vs SGD iterations (see [Carmona, L.'21]):



L^2 norm of residuals for HJB and KFP

Numerical Illustration 3: Crowd Trading

Model of crowd trading [Cardaliaguet, Lehalle]:

$$\begin{cases} dS_t^{\bar{\nu}} = \gamma \bar{\nu}_t dt + \sigma dW_t & \text{(price)} \\ dQ_t^{v} = v_t dt & \text{(player's inventory)} \\ dX_t^{v,\bar{\nu}} = -v_t (S_t^{\bar{\nu}} + \kappa v_t) dt & \text{(player's wealth)} \end{cases}$$

Objective: given $(\bar{\nu}_t)_t$, maximize

$$\mathbb{E}\left[X_T^{v,\bar{\nu}} + Q_T^{v} S_T^{\bar{\nu}} - A|Q_T^{v}|^2 - \phi \int_0^T |Q_t^{v}|^2 dt \right]$$

where: $\phi, A > 0 \Rightarrow$ penalty for holding inventory

Numerical Illustration 3: Crowd Trading

Model of crowd trading [Cardaliaguet, Lehalle]:

$$\begin{cases} dS_t^{\bar{\nu}} = \gamma \bar{\nu}_t dt + \sigma dW_t & \text{(price)} \\ dQ_t^{v} = v_t dt & \text{(player's inventory)} \\ dX_t^{v,\bar{\nu}} = -v_t (S_t^{\bar{\nu}} + \kappa v_t) dt & \text{(player's wealth)} \end{cases}$$

Objective: given $(\bar{\nu}_t)_t$, maximize

$$\mathbb{E}\left[X_T^{v,\bar{\nu}} + Q_T^{v} S_T^{\bar{\nu}} - A|Q_T^{v}|^2 - \phi \int_0^T |Q_t^{v}|^2 dt \right]$$

where: $\phi, A > 0 \Rightarrow$ penalty for holding inventory

Ansatz [Cartea, Jaimungal]: $V(t,x,s,q) = x + qsu(t,q)$, $\qquad \hat{v}_t(q) = \frac{\partial_q u(t,q)}{2\kappa}$

where $u(\cdot)$ solves

$$-\gamma \bar{\nu} q = \partial_t u - \phi q^2 + \sup_{v} \{ v \partial_q u - \kappa v^2 \}, \qquad u(T,q) = -Aq^2$$

Numerical Illustration 3: Crowd Trading

Model of crowd trading [Cardaliaguet, Lehalle]:

$$\begin{cases} dS_t^{\bar{\nu}} = \gamma \bar{\nu}_t dt + \sigma dW_t & \text{(price)} \\ dQ_t^{v} = v_t dt & \text{(player's inventory)} \\ dX_t^{v,\bar{\nu}} = -v_t(S_t^{\bar{\nu}} + \kappa v_t) dt & \text{(player's wealth)} \end{cases}$$

Objective: given $(\bar{\nu}_t)_t$, maximize

$$\mathbb{E}\left[X_T^{v,\bar{\nu}} + Q_T^{v} S_T^{\bar{\nu}} - A|Q_T^{v}|^2 - \phi \int_0^T |Q_t^{v}|^2 dt \right]$$

where: $\phi, A > 0 \Rightarrow$ penalty for holding inventory

Ansatz [Cartea, Jaimungal]: $V(t,x,s,q) = x + qsu(t,q)$, $\qquad \hat{v}_t(q) = \frac{\partial_q u(t,q)}{2\kappa}$

where $u(\cdot)$ solves

$$-\gamma \bar{\nu} q = \partial_t u - \phi q^2 + \sup_{v} \{ v \partial_q u - \kappa v^2 \}, \qquad u(T,q) = -Aq^2$$

Mean field term: at equilibrium

$$\bar{\nu}_t = \int \hat{v}_t(q) \hat{m}(t,dq) = \int \frac{\partial_q \hat{u}(t,q)}{2\kappa} \hat{m}(t,dq),$$

where $\hat{m}$ solves the KFP equation:

$$m(0,\cdot) = m_0, \qquad \partial_t m + \partial_q \left(m \frac{\partial_q \hat{u}(t,q)}{2\kappa} \right) = 0$$

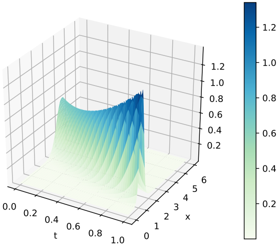
Reduced forward-backward PDE system:

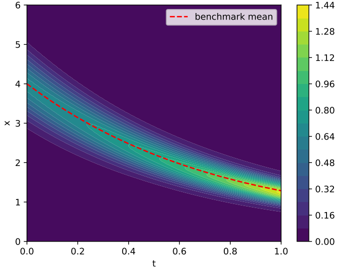
$$\begin{cases} 0 = -\partial_t u(t,q) + \phi q^2 - \dfrac{|\partial_q u(t,q)|^2}{4\kappa} = \gamma \bar{\nu}_t q \\ 0 = \partial_t m(t,q) + \partial_q \left(m(t,q) \dfrac{\partial_q u(t,q)}{2\kappa} \right) \\ \bar{\nu}_t = \displaystyle\int \dfrac{\partial_q u(t,q)}{2\kappa} m(t,q) dq \\ m(0,\cdot) = m_0, u(T,q) = -Aq^2. \end{cases}$$

Numerical Illustration 3: Crowd Trading

Numerical results obtained with DGM & comparison with ODE solution:

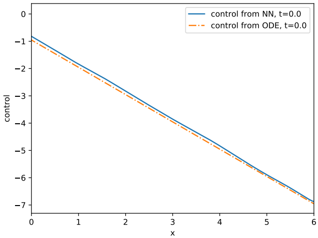
Evolution of m:

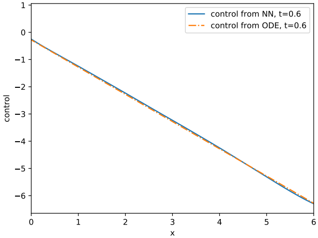


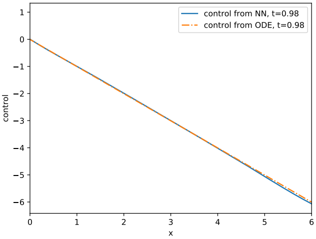


Numerical results obtained with DGM & comparison with ODE solution:

Evolution of equilibrium control $\hat{v}$:







Outline

1. Deep Galerkin Method for MFG PDEs
   - Warm-up: ODE
   - Solving MFG PDE system
   - Link with Generative Adversarial Networks

2. Master Equation

Examples

Examples

thispersondoesnotexist.com

thiscatdoesnotexist.com

Examples

`thispersondoesnotexist.com`

`thiscatdoesnotexist.com`

[Karras *et al.*'20]: Karras, T., Laine, S., Aittala, M., Hellsten, J., Lehtinen, J., & Aila, T. (2020). Analyzing and improving the image quality of stylegan. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 8110-8119).

GANs & MFGs

Generative Adversarial Nets [Goodfellow *et al.*'14]:

Setup: data space $\mathcal{S}$ (e.g. images of fixed size); *unknown* data distribution p_{data}

Goal: be able to generate samples according p_{data}

Given: samples from data, and random noise generator p_z over some space $\mathcal{Z}$

Generative Adversarial Nets [Goodfellow *et al.*'14]:

Setup: data space $\mathcal{S}$ (e.g. images of fixed size); *unknown* data distribution p_{data}

Goal: be able to generate samples according p_{data}

Given: samples from data, and random noise generator p_z over some space $\mathcal{Z}$

Idea: learn $G : \mathcal{Z} \to \mathcal{S}$ such that $p_z \circ G^{-1} \approx p_{data}$

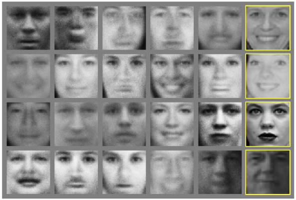
Generative Adversarial Nets [Goodfellow *et al.*'14]:

Setup: data space $\mathcal{S}$ (e.g. images of fixed size); *unknown* data distribution p_{data}

Goal: be able to generate samples according p_{data}

Given: samples from data, and random noise generator p_z over some space $\mathcal{Z}$

Idea: learn $G : \mathcal{Z} \to \mathcal{S}$ such that $p_z \circ G^{-1} \approx p_{data}$



[Goodfellow *et al.*'14]

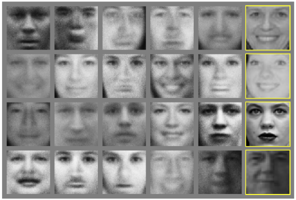
Generative Adversarial Nets [Goodfellow *et al.*'14]:

Setup: data space $\mathcal{S}$ (e.g. images of fixed size); *unknown* data distribution p_{data}

Goal: be able to generate samples according p_{data}

Given: samples from data, and random noise generator p_z over some space $\mathcal{Z}$

Idea: learn $G : \mathcal{Z} \to \mathcal{S}$ such that $p_z \circ G^{-1} \approx p_{data}$



[Goodfellow *et al.*'14]

NVIDIA'19

GANs & MFGs

Generative Adversarial Nets [Goodfellow *et al.*'14]:

Setup: data space $\mathcal{S}$ (e.g. images of fixed size); *unknown* data distribution p_{data}

Goal: be able to generate samples according p_{data}

Given: samples from data, and random noise generator p_z over some space $\mathcal{Z}$

Idea: learn $G : \mathcal{Z} \to \mathcal{S}$ such that $p_z \circ G^{-1} \approx p_{data}$

Idea++: also learn $D : \mathcal{S} \to \mathbb{R}$ to distinguish between samples from $p_z \circ G^{-1}$ and p_{data}

GANs & MFGs

Generative Adversarial Nets [Goodfellow *et al.*'14]:

Setup: data space $\mathcal{S}$ (e.g. images of fixed size); *unknown* data distribution p_{data}

Goal: be able to generate samples according p_{data}

Given: samples from data, and random noise generator p_z over some space $\mathcal{Z}$

Idea: learn $G : \mathcal{Z} \to \mathcal{S}$ such that $p_z \circ G^{-1} \approx p_{data}$

Idea++: also learn $D : \mathcal{S} \to \mathbb{R}$ to distinguish between samples from $p_z \circ G^{-1}$ and p_{data}

Mathematically: min-max game between two neural networks D_δ, G_γ (params: δ, γ)

$$\min_{\gamma} \max_{\delta} \Big\{ \mathbb{E}_{x \sim \mathbb{P}_r}[\log D_\delta(x)] + \mathbb{E}_{z \sim \mathbb{P}_z}[\log(1 - D_\delta(G_\gamma(z)))] \Big\}.$$

GANs & MFGs

Generative Adversarial Nets [Goodfellow *et al.*'14]:

Setup: data space $\mathcal{S}$ (e.g. images of fixed size); *unknown* data distribution p_{data}

Goal: be able to generate samples according p_{data}

Given: samples from data, and random noise generator p_z over some space $\mathcal{Z}$

Idea: learn $G : \mathcal{Z} \to \mathcal{S}$ such that $p_z \circ G^{-1} \approx p_{data}$

Idea++: also learn $D : \mathcal{S} \to \mathbb{R}$ to distinguish between samples from $p_z \circ G^{-1}$ and p_{data}

Mathematically: min-max game between two neural networks D_δ, G_γ (params: δ, γ)

$$\min_{\gamma} \max_{\delta} \Big\{ \mathbb{E}_{x \sim \mathbb{P}_r}[\log D_\delta(x)] + \mathbb{E}_{z \sim \mathbb{P}_z}[\log(1 - D_\delta(G_\gamma(z)))] \Big\}.$$

Variational MFG: $\inf_{u:[0,T]\times\mathbb{R}^d \to \mathbb{R}} \ \sup_{m:[0,T]\times\mathbb{R}^d \to \mathbb{R}} \Phi(m, u)$, where

$$\Phi(m,u) = \int_0^T \int_{\mathbb{T}^d} [m(-\partial_t u - \epsilon \Delta_x u) + m H(x, \nabla_x u, m)] \, dx dt + \int_{\mathbb{T}^d} [m(T) u(T) - m_0 u(0)] \, dx$$

GANs & MFGs

Generative Adversarial Nets [Goodfellow *et al.*'14]:

Setup: data space $\mathcal{S}$ (e.g. images of fixed size); *unknown* data distribution p_{data}

Goal: be able to generate samples according p_{data}

Given: samples from data, and random noise generator p_z over some space $\mathcal{Z}$

Idea: learn $G : \mathcal{Z} \to \mathcal{S}$ such that $p_z \circ G^{-1} \approx p_{data}$

Idea++: also learn $D : \mathcal{S} \to \mathbb{R}$ to distinguish between samples from $p_z \circ G^{-1}$ and p_{data}

Mathematically: min-max game between two neural networks D_δ, G_γ (params: δ, γ)

$$\min_{\gamma} \max_{\delta} \Big\{ \mathbb{E}_{x \sim \mathbb{P}_r} [\log D_\delta(x)] + \mathbb{E}_{z \sim \mathbb{P}_z} [\log(1 - D_\delta(G_\gamma(z)))] \Big\}.$$

Variational MFG: $\inf_{u:[0,T]\times\mathbb{R}^d \to \mathbb{R}} \quad \sup_{m:[0,T]\times\mathbb{R}^d \to \mathbb{R}} \Phi(m, u)$, where

$$\Phi(m,u) = \int_0^T \int_{\mathbb{T}^d} [m(-\partial_t u - \epsilon \Delta_x u) + m H(x, \nabla_x u, m)] \, dx dt + \int_{\mathbb{T}^d} [m(T) u(T) - m_0 u(0)] \, dx$$

$\to$ Conceptual connection GANs/MFGs [Cao, Guo, L.'20]

GANs & MFGs

Generative Adversarial Nets [Goodfellow *et al.*'14]:
Setup: data space $\mathcal{S}$ (e.g. images of fixed size); *unknown* data distribution p_{data}
Goal: be able to generate samples according p_{data}
Given: samples from data, and random noise generator p_z over some space $\mathcal{Z}$

Idea: learn $G : \mathcal{Z} \to \mathcal{S}$ such that $p_z \circ G^{-1} \approx p_{data}$

Idea++: also learn $D : \mathcal{S} \to \mathbb{R}$ to distinguish between samples from $p_z \circ G^{-1}$ and p_{data}

Mathematically: min-max game between two neural networks D_δ, G_γ (params: δ, γ)

$$\min_{\gamma} \max_{\delta} \Big\{ \mathbb{E}_{x \sim \mathbb{P}_r}[\log D_\delta(x)] + \mathbb{E}_{z \sim \mathbb{P}_z}[\log(1 - D_\delta(G_\gamma(z)))] \Big\}.$$

Variational MFG: $\inf_{u:[0,T]\times\mathbb{R}^d \to \mathbb{R}} \ \sup_{m:[0,T]\times\mathbb{R}^d \to \mathbb{R}} \Phi(m, u)$, where

$$\Phi(m,u) = \int_0^T \int_{\mathbb{T}^d} [m(-\partial_t u - \epsilon \Delta_x u) + m H(x, \nabla_x u, m)] \, dx dt + \int_{\mathbb{T}^d} [m(T) u(T) - m_0 u(0)] \, dx$$

$\to$ Conceptual connection GANs/MFGs [Cao, Guo, L.'20]
Related work: [Domingo-Enrich *et al.*, NeurIPS'20; Onken *et al.*'20]

Outline

Master Equation

- Reminder: equilibrium: $(u, \mu) =$ sol. starting with m_0 at $t = 0$
- Idea: express the **value function** of a typical player as $u(t, x) = \mathcal{U}(t, x, \mu_t)$

Master Equation

- Reminder: equilibrium: $(u, \mu) =$ sol. starting with m_0 at $t = 0$
- Idea: express the **value function** of a typical player as $u(t, x) = \mathcal{U}(t, x, \mu_t)$
- Value function $\mathcal{U}$: PDE on the Wasserstein space

Master Equation

- Reminder: equilibrium: $(u, \mu) =$ sol. starting with m_0 at $t = 0$
- Idea: express the **value function** of a typical player as $u(t, x) = \mathcal{U}(t, x, \mu_t)$
- Value function $\mathcal{U}$: PDE on the Wasserstein space
- Motivations:
  - Unknown initial distribution μ_0
  - Macroscopic shocks, common noise
- Main upshot: optimal behavior for every population configuration

Master Equation

- Reminder: equilibrium: $(u, \mu) =$ sol. starting with m_0 at $t = 0$
- Idea: express the **value function** of a typical player as $u(t, x) = \mathcal{U}(t, x, \mu_t)$
- Value function $\mathcal{U}$: PDE on the Wasserstein space
- Motivations:
  - Unknown initial distribution μ_0
  - Macroscopic shocks, common noise
- Main upshot: optimal behavior for every population configuration
- Convergence of N-player games, large deviation principles, ...

Outline

Finite state MFG:

- Finite state space $\mathcal{S}$
- $\mu \in \Delta^{|\mathcal{S}|}$
- $\dot{\mu}_t = \mu_t Q(\mu_t)$, $Q =$ transition rate matrix

Finite state MFG:

- Finite state space $\mathcal{S}$
- $\mu \in \Delta^{|\mathcal{S}|}$
- $\dot{\mu}_t = \mu_t Q(\mu_t)$, $Q =$ transition rate matrix

Master PDE for $\mathcal{U}$:

$$\begin{cases} \mathcal{U}(T, x, \mu) = g(x, \mu) \\ -\partial_t \mathcal{U}(t, x, \mu) = \underbrace{H^*(t, x, \mu, \mathcal{U}(t, \cdot, \mu))}_{\text{Hamiltonian}} + \sum_{x' \in \mathcal{S}} \underbrace{\bar{Q}^*(t, \mu, \mathcal{U}(t, \cdot, \mu))(x')}_{\text{avg transition}} \underbrace{\frac{\partial \mathcal{U}(t, \cdot, \mu)}{\partial \mu(x')}}_{\text{classical deriv.}} \end{cases}$$

for $(t, x, \mu) \in [0, T] \times \mathcal{S} \times \Delta^{|\mathcal{S}|}$

Finite State MFGs

Finite state MFG:

- Finite state space $\mathcal{S}$
- $\mu \in \Delta^{|\mathcal{S}|}$
- $\dot{\mu}_t = \mu_t Q(\mu_t)$, $Q =$ transition rate matrix

Master PDE for $\mathcal{U}$:

$$\begin{cases} \mathcal{U}(T, x, \mu) = g(x, \mu) \\ -\partial_t \mathcal{U}(t, x, \mu) = \underbrace{H^*(t, x, \mu, \mathcal{U}(t, \cdot, \mu))}_{\text{Hamiltonian}} + \sum_{x' \in \mathcal{S}} \underbrace{\bar{Q}^*(t, \mu, \mathcal{U}(t, \cdot, \mu))(x')}_{\text{avg transition}} \underbrace{\frac{\partial \mathcal{U}(t, \cdot, \mu)}{\partial \mu(x')}}_{\text{classical deriv.}} \end{cases}$$

for $(t, x, \mu) \in [0, T] \times \mathcal{S} \times \Delta^{|\mathcal{S}|}$

Numerical solution?

Example: Cyber-Security Model

Cyber-security model (see [Bensoussan, Kolokoltsov'16])

- **State space:** $\mathcal{S} = \{DI, DS, UI, US\}$
  - defended/unprotected
  - infected/susceptible
- **Actions:**
  - $\alpha = 1$ (want to switch level of protection)
  - or 0 (happy)
  - in each case: event happens at rate λ
- **Time:** continuous time, finite time horizon T

Example: Cyber-Security Model

Cyber-security model (see [Bensoussan, Kolokoltsov'16])

- **State space:** $\mathcal{S} = \{DI, DS, UI, US\}$
  - defended/unprotected
  - infected/susceptible
- **Actions:**
  - $\alpha = 1$ (want to switch level of protection)
  - or 0 (happy)
  - in each case: event happens at rate λ
- **Time:** continuous time, finite time horizon T
- **Mean field interactions:** more infected units $\Rightarrow$ higher infection rate

Example: Cyber-Security Model

Cyber-security model (see [Bensoussan, Kolokoltsov'16])

- **State space:** $\mathcal{S} = \{DI, DS, UI, US\}$
  - defended/unprotected
  - infected/susceptible
- **Actions:**
  - $\alpha = 1$ (want to switch level of protection)
  - or 0 (happy)
  - in each case: event happens at rate λ
- **Time:** continuous time, finite time horizon T
- **Mean field interactions:** more infected units $\Rightarrow$ higher infection rate

$$\dot{\mu}(t) = \mu(t) \underbrace{\begin{pmatrix} \dots & q^D_{\text{rec}} & \alpha\lambda & 0 \\ q^D_{\text{inf}} + \beta_{\text{D}}(\mu_{DI}(t) + \mu_{UI}(t)) & \dots & 0 & \alpha\lambda \\ \alpha\lambda & 0 & \dots & q^U_{\text{rec}} \\ 0 & \alpha\lambda & q^U_{\text{inf}} + \beta_{\text{U}}(\mu_{UI}(t) + \mu_{DI}(t)) & \dots \end{pmatrix}}_{\text{transition rates}}$$

Example: Cyber-Security Model

Cyber-security model (see [Bensoussan, Kolokoltsov'16])

- **State space:** $\mathcal{S} = \{DI, DS, UI, US\}$
  - defended/unprotected
  - infected/susceptible
- **Actions:**
  - $\alpha = 1$ (want to switch level of protection)
  - or 0 (happy)
  - in each case: event happens at rate λ
- **Time:** continuous time, finite time horizon T
- **Mean field interactions:** more infected units $\Rightarrow$ higher infection rate

$$\dot{\mu}(t) = \mu(t) \underbrace{\begin{pmatrix} \dots & q^D_{\text{rec}} & \alpha\lambda & 0 \\ q^D_{\text{inf}} + \beta_{\text{D}}(\mu_{DI}(t) + \mu_{UI}(t)) & \dots & 0 & \alpha\lambda \\ \alpha\lambda & 0 & \dots & q^U_{\text{rec}} \\ 0 & \alpha\lambda & q^U_{\text{inf}} + \beta_{\text{U}}(\mu_{UI}(t) + \mu_{DI}(t)) & \dots \end{pmatrix}}_{\text{transition rates}}$$

- **Running cost:**

$$k_D 1_{\{DI,DS\}} + k_I 1_{\{DI,UI\}} = \text{cost of defense} + \text{penalty for being infected}$$

- **Terminal cost:** 0

Numerical Illustration: DGM for Master Equation

We apply the Deep Galerkin Method (see [L.'21 - AMS notes])

- Neural network: $\mathcal{U}_\theta$ to approximate $\mathcal{U}$
- Samples: Pick points $(t, x, \mu) \in [0, T] \times \mathcal{S} \times \Delta^{|\mathcal{S}|}$
- Loss: PDE residual + terminal condition

Comparison:

- $\mathcal{U}_\theta(t, x, \mu(t, \cdot))$
- $\mu(t, x)$, $u(t, x)$: finite state space $\to$ forward-backward ODE system

Numerical Illustration: DGM for Master Equation

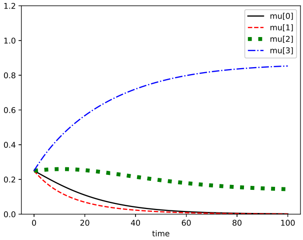
We apply the Deep Galerkin Method (see [L.'21 - AMS notes])

- Neural network: $\mathcal{U}_\theta$ to approximate $\mathcal{U}$
- Samples: Pick points $(t, x, \mu) \in [0, T] \times \mathcal{S} \times \Delta^{|\mathcal{S}|}$
- Loss: PDE residual + terminal condition

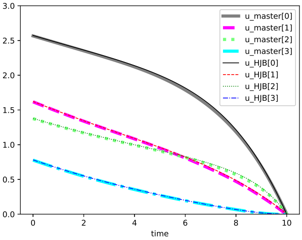
Comparison:

- $\mathcal{U}_\theta(t, x, \mu(t, \cdot))$
- $\mu(t, x)$, $u(t, x)$: finite state space $\to$ forward-backward ODE system

Test 1: $m_0 = (1/4, 1/4, 1/4, 1/4)$



Evolution of μ



Evolution of $u, \mathcal{U}$

Numerical Illustration: DGM for Master Equation

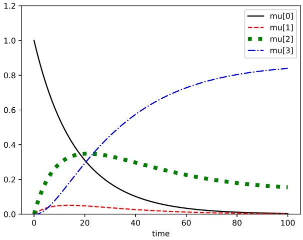
We apply the Deep Galerkin Method (see [L.'21 - AMS notes])

- Neural network: $\mathcal{U}_\theta$ to approximate $\mathcal{U}$
- Samples: Pick points $(t, x, \mu) \in [0, T] \times \mathcal{S} \times \Delta^{|\mathcal{S}|}$
- Loss: PDE residual + terminal condition

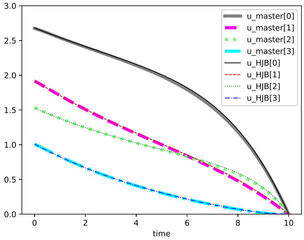
Comparison:

- $\mathcal{U}_\theta(t, x, \mu(t, \cdot))$
- $\mu(t, x)$, $u(t, x)$: finite state space $\to$ forward-backward ODE system

Test 2: $m_0 = (1, 0, 0, 0)$



Evolution of μ



Evolution of $u, \mathcal{U}$

Numerical Illustration: DGM for Master Equation

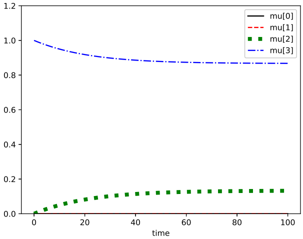
We apply the Deep Galerkin Method (see [L.'21 - AMS notes])

- Neural network: $\mathcal{U}_\theta$ to approximate $\mathcal{U}$
- Samples: Pick points $(t, x, \mu) \in [0, T] \times \mathcal{S} \times \Delta^{|\mathcal{S}|}$
- Loss: PDE residual + terminal condition

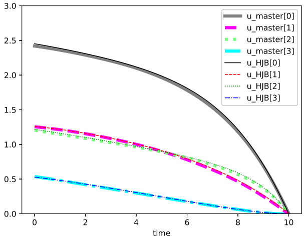
Comparison:

- $\mathcal{U}_\theta(t, x, \mu(t, \cdot))$
- $\mu(t, x)$, $u(t, x)$: finite state space $\rightarrow$ forward-backward ODE system

Test 3: $m_0 = (0, 0, 0, 1)$



Evolution of μ



Evolution of $u, \mathcal{U}$

Outline

1. Deep Galerkin Method for MFG PDEs

2. Master Equation
- Master Equation for Finite State MFG
- Master Bellman PDE of MFC

Master Bellman Equation for MFC

- **MFC problem** with common noise:

$$J^{MFC}(v) = \mathbb{E}\Big[\int_0^T f(X_t, \mathbb{P}^0_{X_t}, v_t)dt + g(X_T, \mathbb{P}^0_{X_T})\Big].$$

subj. to: $dX_t = b(X_t, \mathbb{P}^0_{X_t}, v_t)dt + \sigma dW_t + \sigma_0 dW^0_t$,
where $\mathbb{P}^0_{X_t}$ = conditional law of X_t given the common noise W^0

Master Bellman Equation for MFC

- **MFC problem** with common noise:

$$J^{MFC}(v) = \mathbb{E}\Big[\int_0^T f(X_t, \mathbb{P}^0_{X_t}, v_t)dt + g(X_T, \mathbb{P}^0_{X_T}) \Big].$$

subj. to: $dX_t = b(X_t, \mathbb{P}^0_{X_t}, v_t)dt + \sigma dW_t + \sigma_0 dW_t^0$,
where $\mathbb{P}^0_{X_t}$ = conditional law of X_t given the common noise W^0

- **Master Bellman equation** in the Wasserstein space $\mathcal{P}_2(\mathbb{R}^d)$:

$$\begin{cases} \partial_t V + \mathcal{F}(\mu, V, \partial_\mu V, \partial_x \partial_\mu V, \partial_\mu^2 V) & = & 0, & (t,\mu) \in [0,T) \in \mathcal{P}_2(\mathbb{R}^d) \\ V(T,\mu) & = & \mathcal{G}(\mu), & \mu \in \mathcal{P}_2(\mathbb{R}^d), \end{cases}$$

where:

Master Bellman Equation for MFC

- **MFC problem** with common noise:

$$J^{MFC}(v) = \mathbb{E}\Big[\int_0^T f(X_t, \mathbb{P}^0_{X_t}, v_t)dt + g(X_T, \mathbb{P}^0_{X_T})\Big].$$

  subj. to: $dX_t = b(X_t, \mathbb{P}^0_{X_t}, v_t)dt + \sigma dW_t + \sigma_0 dW^0_t$,
  where $\mathbb{P}^0_{X_t}$ = conditional law of X_t given the common noise W^0

- **Master Bellman equation** in the Wasserstein space $\mathcal{P}_2(\mathbb{R}^d)$:

$$\begin{cases} \partial_t V + \mathcal{F}(\mu, V, \partial_\mu V, \partial_x\partial_\mu V, \partial^2_\mu V) & = & 0, & (t,\mu) \in [0,T) \in \mathcal{P}_2(\mathbb{R}^d) \\ V(T,\mu) & = & \mathcal{G}(\mu), & \mu \in \mathcal{P}_2(\mathbb{R}^d), \end{cases}$$

  where:
  - $\partial_\mu V(\mu)(.) : \mathbb{R}^d \to \mathbb{R}^d$, $\partial_x\partial_\mu V(\mu)(.) : \mathbb{R}^d \to \mathbb{S}^d$, $\partial^2_\mu V(\mu)(.,.) : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{S}^d$, are the L-derivatives of V on $\mathcal{P}_2(\mathbb{R}^d)$ (see [Carmona, Delarue'18])
  - and

$$\mathcal{F}(\mu, y, Z(.), \Gamma(.), \Gamma_0(.,.)) = \int_{\mathbb{R}^d} h(x,\mu,Z(x),\Gamma(x))\mu(dx) \; + \; \int_{\mathbb{R}^d\times\mathbb{R}^d} \tfrac{1}{2}\mathrm{tr}\Big(\sigma_0\sigma_0^\intercal\Gamma_0(x,x')\Big)\mu(dx)\mu(dx'),$$

$$\mathcal{G}(\mu) = \int_{\mathbb{R}^d} g(x,\mu)\mu(dx),$$

$$h(x,\mu,z,\gamma) = \inf_{a\in A}\Big[b(x,\mu,a).z + \frac{1}{2}\mathrm{tr}\Big(\sigma\sigma^\intercal\gamma\Big) \; + \; f(x,\mu,a)\Big].$$

Symmetric Neural Networks

- N agents $\to$ mean field: $\mu^N = \frac{1}{N}\sum_{i=1}^{N} \delta_{x^i}$

$$v^N(t, x, x^1, \ldots, x^N) = V^N(t, x, \mu^N) \to V(t, x, \mu^N)$$

- Approximate $V(t, x, \cdot)$ by a **symmetric** function of N inputs (N large)

2

3

Symmetric Neural Networks

- N agents $\to$ mean field: $\mu^N = \frac{1}{N}\sum_{i=1}^N \delta_{x^i}$

$$v^N(t, x, x^1, \dots, x^N) = V^N(t, x, \mu^N) \to V(t, x, \mu^N)$$

- Approximate $V(t, x, \cdot)$ by a **symmetric** function of N inputs (N large)

- **Symmetric Neural Networks:**
  - Symmetry by construction; e.g. with a sum:

$$(x^i)_{i=1,\dots,N} \mapsto \sum_{i=1}^N \psi_\omega(x^i) \mapsto \varphi_\theta\left(\sum_{i=1}^N \psi_\omega(x^i)\right)$$

  - DeepSets[2], PointNet[3], ...

[2] Zaheer, M., Kottur, S., Ravanbhakhsh, S., Póczos, B., Salakhutdinov, R., & Smola, A. J. (2017, December). Deep Sets. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (pp. 3394-3404).

[3] Qi, C. R., Su, H., Mo, K., & Guibas, L. J. (2017). Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 652-660).

Deep Backward Dynamic Programming for MFC

Deep Learning for MFC based on DPP and Symmetric NN ([Germain et al.'21][4])

- **Symmetric NN:** $\mathcal{V}(t, x^1, \ldots, x^N)$
- **D-Symmetric NN:** sym. except in one space variable:

$$\mathcal{Z}(x^1, \ldots, x^N, x^i) \leftrightarrow \partial_{x^i} \mathcal{V}(x^1, \ldots, x^N) = \frac{1}{N} \partial_\mu \mathcal{V}\left(\frac{1}{N} \sum_j x^j\right)(x^i)$$

Output: $(\widehat{\mathcal{V}}_n, \widehat{\mathcal{Z}}_n)_{n=0,\ldots,N_T}$ s.t. $\widehat{\mathcal{V}}_n(\underline{x}) \approx V(t_n, \mu^N_{\underline{x}})$, $\widehat{\mathcal{Z}}_n(\underline{x}, x^i) \approx \frac{1}{N} \partial_\mu V(t_n, \mu^N_{\underline{x}})(x^i)$

Set $\widehat{\mathcal{V}}_{N_T}(\cdot) = G(\cdot)$

for $n = N_T - 1, N_T - 2, \ldots, 1, 0$ **do**

Compute $(\widehat{\mathcal{V}}_n, \widehat{\mathcal{Z}}_n)$ as a minimizer of:

$$(\mathcal{V}_n, \mathcal{Z}_n) \mapsto \mathbb{E}\Big|\widehat{\mathcal{V}}_{n+1}(\mathbf{X}_{n+1}) - \mathcal{V}_n(\mathbf{X}_n) + H\big(t_n, \mathbf{X}_n, \mathcal{V}_n(\mathbf{X}_n), \mathbf{Z}_n(\mathbf{X}_n)\big)\Delta t - \sum_{i=1}^{N} \sum_{j=0}^{N} \big(\mathcal{Z}_n(\mathbf{X}_n, X^i_n)\big)^\intercal \sigma_{ij} \Delta W^j_n\Big|^2,$$

where $\widehat{\mathcal{V}}_n$ is a sym. NN, $\widehat{\mathcal{Z}}_n$ is a D-sym. NN, $H =$ sym. version of h

return $(\widehat{\mathcal{V}}_n, \widehat{\mathcal{Z}}_n)_{n=0,\ldots,N_T}$

---

[4] Germain, M., Laurière, M., Pham, H., & Warin, X. (2021). DeepSets and their derivative networks for solving symmetric PDEs. arXiv preprint arXiv:2103.00838.

Summary