Numerical resolution of McKean-Vlasov FBSDEs using neural networks *

Maximilien Germain, Joseph Mikael, Xavier Warin

Numerical resolution of McKean-Vlasov FBSDEs using neural networks *

Maximilien Germain †, Joseph Mikael ‡, Xavier Warin §

August 25, 2021

Abstract

We propose several algorithms to solve McKean-Vlasov Forward Backward Stochastic Differential Equations (FBSDEs). Our schemes rely on the approximating power of neural networks to estimate the solution or its gradient through minimization problems. As a consequence, we obtain methods able to tackle both mean-field games and mean-field control problems in moderate dimension. We analyze the numerical behavior of our algorithms on several examples including non linear quadratic models.

Key words: Neural networks, McKean-Vlasov FBSDEs, Deep BSDE, mean-field games, machine learning.

MSC Classification: 65C30, 68T07, 49N80, 35Q89.

1 Introduction

This paper is dedicated to the numerical resolution in moderate dimension of the following McKean-Vlasov Forward Backward Stochastic Differential Equations (MKV FBSDEs)

$$\begin{cases} X_t &= \xi + \int_0^t b(s, X_s, Y_s, Z_s, \mathcal{L}(X_s), \mathcal{L}(Y_s), \mathcal{L}(Z_s))\ \mathrm{d}s + \int_0^t \sigma(s, X_s, \mathcal{L}(X_s))\ \mathrm{d}W_s \\ Y_t &= g(X_T, \mathcal{L}(X_T)) + \int_t^T f(s, X_s, Y_s, Z_s, \mathcal{L}(X_s), \mathcal{L}(Y_s), \mathcal{L}(Z_s))\ \mathrm{d}s - \int_t^T Z_s\ \mathrm{d}W_s \end{cases} \tag{1}$$

with $b : \mathbb{R} \times \mathbb{R}^d \times \mathbb{R}^k \times \mathbb{R}^{k\times d} \times \mathcal{P}_2(\mathbb{R}^d) \times \mathcal{P}_2(\mathbb{R}^k) \times \mathcal{P}_2(\mathbb{R}^{k\times d}) \mapsto \mathbb{R}^d$, $\sigma : \mathbb{R} \times \mathbb{R}^d \times \mathcal{P}_2(\mathbb{R}^d) \mapsto \mathbb{R}^d$, $g : \mathbb{R}^d \times \mathcal{P}_2(\mathbb{R}^d) \mapsto \mathbb{R}^k$, and $f : \mathbb{R} \times \mathbb{R}^d \times \mathbb{R}^k \times \mathbb{R}^{k\times d} \times \mathcal{P}_2(\mathbb{R}^d) \times \mathcal{P}_2(\mathbb{R}^k) \times \mathcal{P}_2(\mathbb{R}^{k\times d}) \mapsto \mathbb{R}^k$.
W_t is a d-dimensional $\mathcal{F}_t$-Brownian motion where $(\Omega, \mathcal{A}, \mathcal{F}_t, \mathbb{P})$ is a given filtered probability space and $T > 0$.
ξ is a given random variable in $L^2(\Omega, \mathcal{F}, \mathbb{P}; \mathbb{R}^d)$ and $\mathcal{P}_2(\mathbb{R}^n)$ stands for the space of square integrable probability measures over $\mathbb{R}^n$ endowed with the $2-$Wasserstein distance

$$\mathcal{W}_2(\mu, \nu) = \inf \left\{ \sqrt{\mathbb{E}[(X - X')^2]} \mid X, X' \in L^2(\Omega, \mathcal{F}, \mathbb{P}; \mathbb{R}^n),\ \mathcal{L}(X) = \mu,\ \mathcal{L}(X') = \nu \right\}. \tag{2}$$

At last $\mathcal{L}(.)$ is a generic notation for the law of a random variable.

*This work is supported by FiME, Laboratoire de Finance des Marchés de l'Energie.
†EDF R&D, Université de Paris, LPSM mgermain at lpsm.fr
‡EDF R&D joseph.mikael at edf.fr
§EDF R&D & FiME xavier.warin at edf.fr

This kind of equation is linked to non local PDEs kwown as master equations. We refer to [CD18], chapter 4 and 5 of volume 2, for an introduction on the subject. In [CCD15], it is shown for example that under regularity conditions, when the drift b is independent of time and the law $\mathcal{L}(Z_t)$, the driver f does not depend on time and the law $\mathcal{L}(Z_t)$, σ does not depend on time, then the resolution of equation (1) provides a way to estimate the solution of the equation

$$\begin{aligned}\partial_t\mathcal{U}(t,x,\mu)+&b(x,\mathcal{U}(t,x,\mu),\partial_x\mathcal{U}(t,x,\mu)\sigma(x,\mu),\mu,\eta).\partial_x\mathcal{U}(t,x,\mu)\\&+\frac{1}{2}\operatorname{Tr}[\partial^2_{xx}\mathcal{U}(t,x,\mu)\sigma\sigma^\top(x,\mu)]+f(x,\mathcal{U}(t,x,\mu),\partial_x\mathcal{U}(t,x,\mu)\sigma(x,\mu),\mu,\eta)\\&+\int_{\mathbb{R}^d}\partial_\mu\mathcal{U}(t,x,\mu)(y).b(y,\mathcal{U}(t,y,\mu),\partial_x\mathcal{U}(t,x,\mu)\sigma(x,\mu),\mu,\eta)\ \mathrm{d}\mu(y)\\&+\int_{\mathbb{R}^d}\frac{1}{2}\operatorname{Tr}[\partial_x\partial_\mu\mathcal{U}(t,x,\mu)(y)\sigma\sigma^\top(y,\mu)]\ \mathrm{d}\mu(y)=0 \text{ on } [0,T]\times\mathbb{R}^d\times\mathcal{P}_2(\mathbb{R}^d)\end{aligned}\tag{3}$$

with initial condition $\mathcal{U}(0,x,\mu)=g(x,\mu)$ on $\mathbb{R}^d\times\mathcal{P}_2(\mathbb{R}^d)$, and where η is a notation for the image of the probability measure μ by the mapping $x\in\mathbb{R}^d\longrightarrow\mathcal{U}(t,x,\mu)\in\mathbb{R}^k$. Under suitable assumptions, [CCD15] proves that the solution to (1) admits the so-called decoupling field representation $Y_s=U(s,X_s,\mathcal{L}(X_s))$ where $(t,x,\mu)\in[0,T]\times\mathbb{R}^d\times\mathcal{P}_2(\mathbb{R}^d)\mapsto U(T-t,x,\mu)$ is a classical solution to (3). See Theorem 2.9 and equation (2.12) in [CCD15]. Their results are stated in the more general of dynamics depending in the joint law of (X_t,Y_t) but we state them here with the particular case of dependence in the marginal laws $\mathcal{L}(X_t),\mathcal{L}(Y_t)$ to be coherent with (1). This MKV FBSDE representation is used in [CCD19] to build a numerical scheme for the approximation of (3) when the drift b is independent of Z. The equation above is non local due to the integral terms and the term $\partial_\mu\mathcal{U}(t,x,\mu)(y)$ stands for the Wasserstein derivative of $\mathcal{U}$ in the direction of the measure at point (t,x,μ) and evaluated at the continuous coordinate y.

Equations (1) appear as well as probabilistic formulations of mean-field games or mean-field controls characterizing the value function V of the game. Mean-field games are introduced by [LL06a] and [LL06b] to model games with interactions between many similar players. In this theory, each player's dynamics and cost take into account the empirical distribution of all agents. At the limit of an infinite number of players, the search for a Nash equilibrium with close loop controls boils down to a control problem concerning a representative player whose law enters in the cost and dynamics. Two probabilistic approaches based on Forward Backward Stochastic Differential Equations can be used to solve these problems:

- A first approach called the **Pontryagin approach** consists as shown in [CD13] in applying the strong Pontryagin principle to these control problems. Under regularity and convexity conditions, Y_t appears to be a stochastic representation of the gradient of the value function V. In this case the coefficients b,f of the related MKV FBSDE (1) do not depend on $Z_t,\mathcal{L}(Y_t),\mathcal{L}(Z_t)$ and $k=d$.

- Another approach called the **Weak approach** permits to solve the optimization problem by estimating directly Y_t as the value function V of the problem as shown in [CL15]. In this case the coefficients b,f of the related MKV FBSDE (1) do not depend on $Y_t,\mathcal{L}(Y_t),\mathcal{L}(Z_t)$ and $k=1$.

The numerical resolution of equations (1) is rather difficult since:

- The dynamics are coupled through both the drift and the driver of the BSDE.

- The McKean-Vlasov structure of the problem requires to solve a fixed point in probability spaces.

In the linear-quadratic setting (with quadratic cost to minimize but linear dynamics) the weak approach applied to mean-field problems gives a problem in low dimension but with quadratic coupling in Z_t appearing in the backward dynamic. In contrast, the Pontryagin approach exhibits a problem in potentially high dimension (the Z component is a $d \times d$ matrix in this case) but with a linear coupling in Y_t which is easier to solve numerically.
In the case of mean-field games, only the law of X_t is present in the dynamic of (1). In other applications, individuals interact through their controls instead of their states as in the application of trade crowding in [CL18]. The law of the control thus appears in the dynamic of (1) and may give rise to some FBSDE depending on the law of Z_t in the weak approach or the law of Y_t in the Pontryagin approach.

Existence and uniqueness of a solution to the fully coupled system (1) are studied by [CD18] when the drift does not depend on the law of Z. Their Theorem 4.29 gives existence of a solution under a non-degeneracy condition. However, uniqueness is a priori only expected to hold in small time, as stated in Theorem 4.24 from [CD18]. In the latter we will assume that existence and uniqueness hold for the MKV FBSDE we aim to solve.

In [CCD19] and [Ang+19], tree and grid algorithms are proposed and tested in dimension 1. It is worth mentioning that these techniques suffer from the so-called curse of dimensionality and cannot be applied when the dimension describing a player state is too high (typically greater than 3 or 4). This is due to the discretization of the state space.

However, new approaches using machine learning are developed since 2017 to solve non linear parabolic PDEs through a BSDE representation. Two kinds of methods have emerged:

- The first to appear are **global methods** first proposed in [HJE17] to solve semi-linear PDEs. They rely on a single high-dimensional optimization problem whose resolution is difficult. It consists in the training of as many neural networks as time steps by solving in a forward way the backward representation of the PDE solution. The Z_t process is represented by a different neural network Z_i^θ with parameters θ at each date t_i. Instead of solving the BSDE starting from the terminal condition, the method writes it down as a forward equation and an optimization problem aiming to reach the terminal condition $g(X_T)$ by minimizing a mean squared error $\mathbb{E}|Y_T - g(X_T)|^2$. The approach is extended to fully nonlinear equations (nonlinear in the solution, its gradient and hessian) in [BEJ19] and the authors show that the methodology can solve some equations in high dimension. [CWNMW19] showed that it is more effective to used a single network for all dates and besides proposed an original fixed point algorithm to solve semilinear PDEs.

- A second kind of **local methods** first proposed in [HPW20] is based on local optimization problems solved at each time step in a backward way. Contrarily to the global method, the successive optimization problems are here in moderate dimension. Each optimization step at date t_i consists in the training of only two local neural networks Y_i^θ, Z_i^θ with parameters θ. For instance, instead of solving 1 optimization problem with N neural networks in the global method, [HPW20] solves N learning problems with 2 neural networks. Moreover the resolution is simplified by the initialization of the neural networks at time t_i to their previously computed values at time t_{i+1}, which provides a good approximation for the current value. This strategy is inspired by the standard backward resolution of BSDE with conditional expectations from [BT04] and [GLW05]. The methodology is extended to the much more challenging case of fully nonlinear PDEs in [PWG21] by combining it with some ideas proposed in [Bec+19]. Extensive tests performed in [HPW20] show that the local method gives better results than the global

one, such as [PWG21] in the case of fully nonlinear dynamics. Especially, these papers show that local methods can be used with a larger time horizon T than the global method.

Machine learning techniques to solve coupled FBSDEs are investigated by several authors in [HL20] and [Ji+20], and a first method for McKean-Vlasov FBSDEs with delay is studied by [FZ20] for a linear quadratic equation in dimension one. Similar and more general ideas are presented and tested in dimension one in [CL19] alongside convergence results, together with an additional method directly solving mean-field control problems by minimizing the cost without writing down optimality conditions. The resulting algorithms proposed all rely on the global approach first initiated in [HJE17].

Our paper aims to extend these methods and to propose new ones for the resolution of McKean-Vlasov FBSDEs in moderate dimension, and go beyond one dimensional examples for which standard methods are already available (see [ACD10; Lau21]). In fact, one major advantage for the use of neural networks for solving control problems is their ability to efficiently represent high-dimensional functions without using space grids. We also study the influence of the maturity T on the algorithms. We first propose to modify the previously proposed algorithm to stabilize its convergence. Our modification allows us to reduce the variance of the estimators used in the dynamic of X_t and Y_t. Then we propose a second algorithm relaxing the fixed point iteration algorithm by adding a neural network learning the distribution of the solution thanks to a penalization in the loss function. At last we propose a resolution scheme based on some local resolution as in [HPW20].

To simplify the presentation, we consider first order interaction models, that is to say that the dependency of the drift and cost function with respect to the laws $\mathcal{L}(X_t), \mathcal{L}(Y_t), \mathcal{L}(Z_t)$ only concerns expectations in the form

$$u_t = (u_t^X, u_t^Y, u_t^Z) := (\mathbb{E}[\varphi_1(X_t)], \mathbb{E}[\varphi_2(Y_t)], \mathbb{E}[\varphi_3(Z_t)]), \tag{4}$$

for some continuous functions $\varphi_1, \varphi_2, \varphi_3$ with adequate domains and codomains. In this framework we can rewrite by abuse of notation

$$\begin{aligned} b(s, X_s, Y_s, Z_s, \mathcal{L}(X_s), \mathcal{L}(Y_s), \mathcal{L}(Z_s)) &= b(s, X_s, Y_s, Z_s, u_s) \\ \sigma(s, X_s, \mathcal{L}(X_s)) &= \sigma(s, X_s, u_s^X) \\ g(X_T, \mathcal{L}(X_T)) &= g(X_T, u_T^X) \\ f(s, X_s, Y_s, Z_s, \mathcal{L}(X_s), \mathcal{L}(Y_s), \mathcal{L}(Z_s)) &= f(s, X_s, Y_s, Z_s, u_s). \end{aligned}$$

For instance when $\varphi_1, \varphi_2, \varphi_3$ are power functions with positive integers as exponents we recover probability distribution moments. See Remark 1 for more general cases beyond first order interaction.

We provide multidimensional tests to show how these machine learning approaches can overcome the curse of dimensionality on some test cases first coming from a mean-field game of controls: we solve the FBSDE derived from the weak approach and the Pontryagin approach. We also consider an example arising from a non linear quadratic mean-field game. Then we compare all the methods on some general test cases of FBSDE involving linear or quadratic dependence on the processes X_t, Y_t, Z_t and on their distributions.

The structure of the paper is the following: in sections 2 and 3 we describe the proposed schemes, and in section 4 we provide a numerical study of our methods in dimension 10 (except for the one-dimensional Example of Section 4.2). We show that our algorithms can solve non linear-quadratic models with small maturities.

2 Machine learning global solvers

In this section we propose three **global algorithms** based on the approach in [HJE17].

2.1 Algorithm principle

We propose a generalized and refined version of the Algorithm 2 from [CL19]. We recall that a similar technique with additional networks is used in [FZ20] for delayed McKean-Vlasov equations but is tested only on a one dimensional linear quadratic example. Our methods also take advantage of different expectation computation methods, introduced in section 2.2. We present in section 4 several tests in dimension 10 where the laws of X, Y, Z are involved.

We consider the Euler-Maruyama discretized FBSDE system (1) on a regular time grid $t_k = \frac{kT}{N}$ for $k \in [\![0, N]\!]$:

$$\begin{cases} X_{t_{i+1}} &= X_{t_i} + b\left(t_i, X_{t_i}, Y_{t_i}, Z_{t_i}, u_{t_i}\right)\ \Delta t + \sigma\left(t_i, X_{t_i}, u^X_{t_i}\right) \Delta W_i \\ Y_{t_{i+1}} &= Y_{t_i} - f\left(t_i, X_{t_i}, Y_{t_i}, Z_{t_i}, u_{t_i}\right)\ \Delta t + Z_{t_i} \Delta W_i. \end{cases} \tag{5}$$

with terminal condition $Y_{t_N} = g(X_{t_N}, u^X_{t_N})$ and initial condition $X_0 = \xi$. We recall that u_t is defined in (4). We note $\Delta t := t_{i+1} - t_i = \frac{T}{N}$ and $(\Delta W_i)_{i=0,\cdots,N-1} := (W_{t_{i+1}} - W_{t_i})_{i=0,\cdots,N-1}$ the Brownian increments. In the FBSDE theory, one requires the processes $(X_{t_i}, Y_{t_i}, Z_{t_i})$ to be $\mathcal{F}_{t_i}$-adapted. Therefore the backward part of the system can also be written in the conditional expectation form

$$\begin{cases} Y_{t_i} = \mathbb{E}[Y_{t_{i+1}} + f\left(t_i, X_{t_i}, Y_{t_i}, Z_{t_i}, u_{t_i}\right)\ \Delta t | \mathcal{F}_{t_i}] \\ Z_{t_i} = \mathbb{E}[Y_{t_{i+1}} \frac{\Delta W_i}{\Delta t} | \mathcal{F}_{t_i}], \end{cases} \tag{6}$$

where we see how the process Z is defined. This process, specific to the stochastic case, allows the Y component to be $\mathcal{F}_t$-adapted, even though we fix its terminal condition. It is a major difference between backward ordinary differential equations and backward stochastic differential equations.
We see that the whole system is coupled therefore we need to design a method allowing to solve simultaneously both equations of (5).

We solve the system by the Merged Deep BSDE method introduced in [CWNMW19]. Z_{t_i} is approximated by a single feedforward neural network $\mathcal{Z}_{\theta^z}(t_i, X_{t_i})$ and Y_0 by a neural network $\mathcal{Y}_{\theta^y}(X_0)$ with parameters $\theta = (\theta^y, \theta^z)$. With this point of view, the discretized Brownian motion W_t acts as training data in the language of machine learning, so that we can generate a training set as large as desired. Extensive tests conducted in [CWNMW19] show that the use of a Merged network improves the training in comparison with the Deep BSDE method of [HJE17]. Indeed it lowers the number of parameters to learn hence reduces the complexity of the problem. It empirically improves the accuracy of the method but also makes the training faster. That is why we focus on this architecture. It is also used by [CL19] which considers controls in the form $\left(\zeta(t_i, X_i)\right)_{i=1,\dots,N}$ for a neural network ζ. The use of recurrent networks such as Long Short Term Memory networks as in [FZ20] is possible but tests achieved in [CWNMW19] seem to show that is does not bring more accuracy on Markovian problems. Other alternatives may include the GroupSort network [ALG19] for a better control of the Lipschitz constant of the approximation, or some special networks preserving some properties of the solution but they have not been tested.

The motivation for such an approximation comes from the notion of decoupling field, also used for numerical purposes in [Ang+19] or [CCD19], which gives the existence of functions u, v (see the paragraph below (3)) such that

$$Y_t = u(t, X_t, \mathcal{L}(X_t)),\ Z_t = v(t, X_t, \mathcal{L}(X_t)). \tag{7}$$

Numerically, it is enough to consider Y and Z as a function of the couple (t, X_t). In fact, the law of the solution (and therefore its moments) can be seen as a function of t. That's why we search for a representation

$$Y_t = \widetilde{u}(t, X_t),\ Z_t = \widetilde{v}(t, X_t). \tag{8}$$

The forward-backward system is transformed into a forward system and an optimization problem aiming to satisfy the terminal condition of the BSDE through the loss function $\mathbb{E}\big[\big(Y_T - g\big(X_T, u^X_{t_N}\big)\big)^2\big]$. To simplify notations, $X_i := X_{t_i}$ and similarly for Y and Z.

In practice the loss function is minimized with the Adam gradient descent method [KB14]. In any case, the goal of our scheme is to learn both the optimal control and the distribution of X_t, Y_t, Z_t. In the following, B is the batch size, N is the number of time steps and M is the number of previous batches expectations to keep in memory.

We use for $\mathcal{Z}$ and $\mathcal{Y}$ feedforward neural networks with 3 hidden layers ($d+10$ neurons in each) with hyperbolic tangent function as activation functions and an output layer with identity as activation. It is worth noticing that because the merged neural network takes the couple (t, X) as inputs, we cannot use batch normalization since the distribution of X_i is not stationary over time.

2.2 Estimation of the expectation

A key step for the methods is to estimate the mean-field parameter u. It has a significant effect on the algorithms performances. We note $\theta_m = (\theta^y_m, \theta^z_m)$ the neural network parameters at optimization iteration m and $u_i = (u^X_i, u^Y_i, u^Z_i)$ the estimation of u_{t_i}. In the algorithms described below, the approximated processes are considered as functions of the parameters θ of the neural network.
Several methods can be used to approximate the moments of the solution involved in the stochastic McKean-Vlasov dynamics:

- **Direct**: use the empirical mean of the current batch of particles

$$u_i = \frac{1}{B}\left(\sum_{j=1}^{B}\varphi_1(X^j_i(\theta_m)), \sum_{j=1}^{B}\varphi_2(Y^j_i(\theta_m)), \sum_{j=1}^{B}\varphi_3(Z^j_i(\theta_m))\right),\ i = 0, \cdots, N-1. \tag{9}$$

  Alternatively one could use instead the last batch particles to estimate the law

$$u_i = \frac{1}{B}\left(\sum_{j=1}^{B}\varphi_1(X^j_i(\theta_{m-1})), \sum_{j=1}^{B}\varphi_2(Y^j_i(\theta_{m-1})), \sum_{j=1}^{B}\varphi_3(Z^j_i(\theta_{m-1}))\right),\ i = 0, \cdots, N-1. \tag{10}$$

  The difference lies in the fact that in one case the optimization of the parameters θ_m at iteration m modifies the current estimation of the law whereas using the previously computed parameters θ_{m-1} fixes the law and simplifies the optimization problem. In practice, for the numerical tests of Section 4 we use the formula (9). This approach requires to handle very large batches, typically of the order of $B = 10,000$ sample paths get a reasonable approximation of the laws. This is the approach used by [FZ20] and [CL19].

We solve the following optimization problem

$$
\begin{aligned}
\min_{\theta=(\theta^y,\theta^z)} \frac{1}{B}\sum_{k=1}^{B}\left|Y_N^k(\theta) - g\Big(X_N^k(\theta), \frac{1}{B}\sum_{j=1}^{B}\varphi_1(X_N^j(\theta))\Big)\right|^2 \\
X_{i+1}^j(\theta) = X_i^j(\theta) + b\left(t_i, X_i^j(\theta), Y_i^j(\theta), \mathcal{Z}_{\theta^z}\left(t_i, X_i^j(\theta)\right), u_i\right)\Delta t + \sigma\left(t_i, X_i^j(\theta), u_i^X\right)\Delta W_i^j \\
Y_{i+1}^j(\theta) = Y_i^j(\theta) - f\left(t_i, X_i^j(\theta), Y_i^j(\theta), \mathcal{Z}_{\theta^z}\left(t_i, X_i^j(\theta)\right), u_i\right)\ \Delta t + \mathcal{Z}_{\theta^z}\left(t_i, X_i^j(\theta)\right)\Delta W_i^j \\
u_i = \frac{1}{B}\left(\sum_{j=1}^{B}\varphi_1(X_i^j(\theta)), \sum_{j=1}^{B}\varphi_2(Y_i^j(\theta)), \sum_{j=1}^{B}\varphi_3\left(\mathcal{Z}_{\theta^z}\left(t_i, X_i^j(\theta)\right)\right)\right) \\
X_0^j = \xi^j \sim \xi,\ j = 1, \cdots, B \\
Y_0^j(\theta) = \mathcal{Y}_{\theta^y}(X_0^j), \\
i = 1, \cdots, N-1.
\end{aligned}
$$

The **direct solver** leads to algorithm 1.

Algorithm 1 Direct solver

Let $\mathcal{Y}_{\theta^y}(\cdot)$ be a neural network with parameter θ^y, defined on $\mathbb{R}^d$ and valued in $\mathbb{R}^k$, $\mathcal{Z}_{\theta^z}(\cdot,\cdot)$ be a neural network with parameter θ^z, defined on $\mathbb{R}^+ \times \mathbb{R}^d$ and valued in $\mathbb{R}^{k\times d}$, so that $\theta = (\theta^y, \theta^z)$ is initialized with value $\theta_0 = (\theta_0^y, \theta_0^z)$.
for m from 0 to K **do** ▷ Stochastic gradient iterations
  Sample $(\xi^j)_{j=1,\cdots,B}$ from B independent copies of the initial condition ξ.
  Set $\forall j \in [\![1, B]\!],\ X_0^j(\theta_m) = \xi^j \in \mathbb{R}^d, Y_0^j(\theta_m) = \mathcal{Y}_{\theta_m^y}(\xi^j) \in \mathbb{R}^k$.
  for i from 0 to $N-1$ **do**
    $u_i = (u_i^X, u_i^Y, u_i^Z) = \frac{1}{B}\sum_{j=1}^{B}\left(\varphi_1(X_i^j(\theta_m)), \varphi_2(Y_i^j(\theta_m)), \varphi_3\Big(\mathcal{Z}_{\theta_m^z}\left(t_i, X_i^j(\theta_m)\right)\Big)\right)$
    for j from 1 to B **do**
      Sample δ_i^j from a d-dimensional standard Gaussian vector.
      $X_{i+1}^j(\theta_m) = X_i^j(\theta_m) + b\left(t_i, X_i^j(\theta_m), Y_i^j(\theta_m), \mathcal{Z}_{\theta_m^z}\left(t_i, X_i^j(\theta_m)\right), u_i\right)\Delta t + \sqrt{\Delta t}\ \sigma\left(t_i, X_i^j(\theta_m), u_i^X\right)\delta_i^j$
      $Y_{i+1}^j(\theta_m) = Y_i^j(\theta_m) - f\left(t_i, X_i^j(\theta_m), Y_i^j(\theta_m), \mathcal{Z}_{\theta_m^z}\left(t_i, X_i^j(\theta_m)\right), u_i\right)\Delta t + \sqrt{\Delta t}\ \mathcal{Z}_{\theta_m^z}\left(t_i, X_i^j(\theta_m)\right)\delta_i^j$
    end for
  end for
  $\overline{X_N}(\theta_m) = \frac{1}{B}\sum_{j=1}^{B}\varphi_1(X_N^j(\theta_m))$,
  $J(\theta_m) = \frac{1}{B}\sum_{j=1}^{B}\left(Y_N^j(\theta_m) - g\left(X_N^j(\theta_m), \overline{X_N}(\theta_m)\right)\right)^2$
  Calculate $\nabla J(\theta_m)$ by back-propagation.
  Update $\theta_{m+1} = \theta_m - \rho_m \nabla J(\theta_m)$.
end for

- **Dynamic**: a method which dynamically updates the estimation on $(M+1)B$ samples. The expectations from the last M batches are kept in memory in an array

$$
\begin{matrix}(\zeta_{i,r}) & i = 0, \ldots, N-1, \\ & r = 0, \ldots, M-1\end{matrix}
$$

initialized with values $(\mathbb{E}[\varphi_1(\xi)], \varphi_2(0), \varphi_3(0))^{N\times M}$.
At iteration $m-1$, $\nu_i^{(m-1)}$ is defined as the empirical mean on these previous sample paths. On a new batch, the expectation is computed by averaging the previous estimation $\nu_i^{(m-1)}$ and the current batch empirical mean by the following algorithm used for $i = 0, \cdots, N-1$:

$$\begin{aligned}
\nu_i^{(m-1)} &= \frac{1}{M}\sum_{r=0}^{M-1}\zeta_{i,r},\\
u_i &= \frac{M\nu_i^{(m-1)} + \frac{1}{B}\left(\sum_{j=1}^B \varphi_1(X_i^j(\theta_m)), \sum_{j=1}^B \varphi_2(Y_i^j(\theta_m)), \sum_{j=1}^B \varphi_3(Z_i^j(\theta_m))\right)}{M+1},\\
\zeta_{i,m\%M} &= \frac{1}{B}\left(\sum_{j=1}^B \varphi_1(X_i^j(\theta_m)), \sum_{j=1}^B \varphi_2(Y_i^j(\theta_m)), \sum_{j=1}^B \varphi_3(Z_i^j(\theta_m))\right). \qquad (11)
\end{aligned}$$

The notation $m\%M$ refers to the remainder of the Euclidian division of m by M. This technique allows to use smaller batches of size 100 or 1000. Thus it is more efficient in terms of convergence speed in comparison with the direct approach. This method can be seen as a dynamic fixed point approach.

The idea behind this update rule comes from online learning in machine learning. $1/M$ can be interpreted as a learning rate quantifying the updating speed. From the current estimation of the particles law, we introduce a small correction related to the new observed samples. Therefore the estimation is much more stable through iterations compared to the instantaneous update of the law used by the Direct method. After M batches, the older samples are forgotten, since they don't represent anymore the current law. Indeed we expect the convergence for a good choice of M. If this parameter is too small the stabilization would be inefficient and on the contrary a too large M would slow down the learning process by introducing a bias in the law. For instance in our numerical experiments of Section 4 we use $M = 100$ for a total of 2000 gradient descent iterations.

Remark 1. *If the law dependence is more general than a first order interaction and is given by a continuous function $F : \mu \in \mathcal{P}_2(\mathbb{R}^d) \mapsto \mathbb{R}^k$ then the Direct method can be straightforwardly applied to the equation by estimating $F(\mathcal{L}(X_t))$ by the so-called empirical projection $F(\frac{1}{B}\sum_{j=1}^B \delta_{X_{t_i}^j})$ for identically distributed particles $(X_{t_i}^j)_{j=1,\dots,B}$ on a time grid $t_0, \cdots, t_N$. Concerning the Dynamic approach, it would require to keep in memory the previously computed particles from the last M batches which is costly.*

Remark 2. *The fixed point approach is known to be convergent theoretically only for small maturities. In practice, the theoretical bound on the maturity found on the simple example given for example in paragraph 3.1 in [Ang+19] is far too pessimistic. We will see that the restriction is not relevant on all our test cases.*

For a given iteration m, given the estimations $(\zeta_{i,r})_{\substack{i=0,\dots,N-1,\\ r=0,\dots,M-1}}$ of u_i on the last M

iterations, we perform one gradient descent step for the following optimization problem

$$
\begin{aligned}
\min_{\theta_m=(\theta_m^y,\theta_m^z)} \ & \frac{1}{B}\sum_{k=1}^{B}\left|Y_N^k(\theta_m) - g\Big(X_N^k(\theta_m), \frac{1}{B}\sum_{j=1}^{B}\varphi_1(X_N^j(\theta_m))\Big)\right|^2 \\
X_{i+1}^j(\theta_m) &= X_i^j(\theta_m) + b\left(t_i, X_i^j(\theta_m), Y_i^j(\theta_m), \mathcal{Z}_{\theta_m^z}\left(t_i, X_i^j(\theta_m)\right), \widetilde{u}_i\right)\Delta t \\
&\quad + \sigma\left(t_i, X_i^j(\theta_m), \widetilde{u}_i{}^X(\theta_m)\right)\Delta W_i^j \\
Y_{i+1}^j(\theta_m) &= Y_i^j(\theta_m) - f\left(t_i, X_i^j(\theta_m), Y_i^j(\theta_m), \mathcal{Z}_{\theta_m^z}\left(t_i, X_i^j(\theta_m)\right), \widetilde{u}_i\right)\ \Delta t \\
&\quad + \mathcal{Z}_{\theta_m^z}\left(t_i, X_i^j(\theta_m)\right)\Delta W_i^j \\
u_i &= \frac{1}{B}\left(\sum_{j=1}^{B}\varphi_1(X_i^j(\theta_m)), \sum_{j=1}^{B}\varphi_2(Y_i^j(\theta_m)), \sum_{j=1}^{B}\varphi_3\left(\mathcal{Z}_{\theta_m^z}\left(t_i, X_i^j(\theta_m)\right)\right)\right) \\
\widetilde{u}_i &= \frac{\sum_{r=0}^{M-1}\zeta_{i,r} + u_i}{M+1} \\
X_0^j &= \xi^j \sim \xi,\ j = 1, \cdots, B \\
Y_0^j(\theta_m) &= \mathcal{Y}_{\theta_m^y}(X_0^j) \\
i &= 1, \cdots, N-1.
\end{aligned}
$$

Then we update $(\xi_i)_i$ by forgetting the oldest estimation and keeping in memory the new one, $(u_i)_i$ (see (11)). The **dynamic solver** is given more explicitly in algorithm 2.

Algorithm 2 Dynamic solver

Let $\mathcal{Y}_{\theta^y}(\cdot)$ be a neural network with parameter θ^y, defined on $\mathbb{R}^d$ and valued in $\mathbb{R}^k$, $\mathcal{Z}_{\theta^z}(\cdot,\cdot)$ be a neural network with parameter θ^z, defined on $\mathbb{R}^+ \times \mathbb{R}^d$ and valued in $\mathbb{R}^{k\times d}$, so that $\theta = (\theta^y, \theta^z)$ is initialized with value $\theta_0 = (\theta_0^y, \theta_0^z)$.
Set $\forall i \in [\![0, N-1]\!],\ \forall r \in [\![0, M-1]\!],\ \zeta_{i,r} = (\mathbb{E}[\varphi_1(\xi)], \varphi_2(0), \varphi_3(0))$.
for m from 0 to K **do**
  Sample $(\xi^j)_{j=1,\cdots,B}$ from B independent copies of the initial condition ξ.
  Set $\forall j \in [\![1, B]\!],\ X_0^j(\theta_m) = \xi^j \in \mathbb{R}^d, Y_0^j(\theta_m) = \mathcal{Y}_{\theta_m^y}(\xi^j) \in \mathbb{R}^k$.
  for i from 0 to $N-1$ **do**
    $u_i = \frac{1}{B}\left(\sum_{j=1}^B \varphi_1(X_i^j(\theta_m)), \sum_{j=1}^B \varphi_2(Y_i^j(\theta_m)), \sum_{j=1}^B \varphi_3\Big(\mathcal{Z}_{\theta_m^z}\left(t_i, X_i^j(\theta_m)\right)\Big)\right)$
    $\widetilde{u_i} = (\widetilde{u_i}^X, \widetilde{u_i}^Y, \widetilde{u_i}^Z) = \frac{\sum_{r=0}^{M-1}\zeta_{i,r}+u_i}{M+1}$
    for j from 1 to B **do**
      Sample δ_i^j from a d-dimensional standard Gaussian vector.
      $X_{i+1}^j(\theta_m) = X_i^j(\theta_m) + b\left(t_i, X_i^j(\theta_m), Y_i^j(\theta_m), \mathcal{Z}_{\theta_m^z}\left(t_i, X_i^j(\theta_m)\right), \widetilde{u_i}\right)\Delta t + \sqrt{\Delta t}\,\sigma\left(t_i, X_i^j(\theta_m), \widetilde{u_i}^X\right)\delta_i^j$
      $Y_{i+1}^j(\theta_m) = Y_i^j(\theta_m) - f\left(t_i, X_i^j(\theta_m), Y_i^j(\theta_m), \mathcal{Z}_{\theta_m^z}\left(t_i, X_i^j(\theta_m)\right), \widetilde{u_i}\right)\Delta t + \sqrt{\Delta t}\,\mathcal{Z}_{\theta_m^z}\left(t_i, X_i^j(\theta_m)\right)\delta_i^j$
    end for
    $\zeta_{i,m\%M} = u_i$
  end for
  $\overline{X_N}(\theta_m) = \frac{1}{B}\sum_{j=1}^B \varphi_1(X_N^j(\theta_m))$,
  $J(\theta_m) = \frac{1}{B}\sum_{j=1}^B \left(Y_N^j(\theta_m) - g\left(X_N^j(\theta_m), \overline{X_N}(\theta_m)\right)\right)^2$
  Calculate $\nabla J(\theta_m)$ by back-propagation.
  Update $\theta_{m+1} = \theta_m - \rho_m \nabla J(\theta_m)$.
end for

- **Expectation**: estimate u_t by a neural network Ψ_{θ^Ψ} with input t and parameters θ^Ψ.

$$u_i(\theta^\Psi) = \Psi_{\theta^\Psi}(t_i) = (\Psi^X_{\theta^\Psi}(t_i), \Psi^Y_{\theta^\Psi}(t_i), \Psi^Z_{\theta^\Psi}(t_i)),\ i = 0, \cdots, N. \tag{12}$$

  A penalization term

$$\mathbb{E}\left[\frac{\lambda}{N}\sum_{i=0}^{N-1}\left\|\Psi_{\theta^\Psi}(t_i) - \frac{1}{B}\left(\sum_{j=1}^B \varphi_1(X_i^j(\theta)), \sum_{j=1}^B \varphi_2(Y_i^j(\theta)), \sum_{j=1}^B \varphi_3(Z_i^j(\theta))\right)\right\|_2^2\right],$$

  is added to the loss function. We will see that in practice this method is quite involved to use because the performances heavily depend upon the choice of the parameter λ. This approach provides a relaxation of the fixed point method.

We solve the following optimization problem

$$
\begin{aligned}
\min_{\theta=(\theta^y,\theta^z,\theta^\Psi)} \; & \frac{1}{B}\sum_{k=1}^{B}\left|Y_N^k(\theta) - g\Big(X_N^k(\theta), \frac{1}{B}\sum_{j=1}^{B}\varphi_1(X_N^j(\theta))\Big)\right|^2 \\
& + \frac{\lambda}{N}\sum_{i=0}^{N-1}\left\|u_i(\theta^\Psi) - \Psi_{\theta^\Psi}(t_i)\right\|^2 \\
X_{i+1}^j(\theta) = \; & X_i^j(\theta) + b\left(t_i, X_i^j(\theta), Y_i^j(\theta), \mathcal{Z}_{\theta^z}\left(t_i, X_i^j(\theta)\right), \Psi_{\theta^\Psi}(t_i)\right)\Delta t \\
& + \sigma\left(t_i, X_i^j(\theta), \Psi_{\theta^\Psi}(t_i)^X\right)\Delta W_i^j \\
Y_{i+1}^j(\theta) = \; & Y_i^j(\theta) - f\left(t_i, X_i^j(\theta), Y_i^j(\theta), \mathcal{Z}_{\theta^z}\left(t_i, X_i^j(\theta)\right), \Psi_{\theta^\Psi}(t_i)\right)\Delta t \\
& + \mathcal{Z}_{\theta^z}\left(t_i, X_i^j(\theta)\right)\Delta W_i^j \\
u_i = \; & \frac{1}{B}\left(\sum_{j=1}^{B}\varphi_1(X_i^j(\theta)), \sum_{j=1}^{B}\varphi_2(Y_i^j(\theta)), \sum_{j=1}^{B}\varphi_3\left(\mathcal{Z}_{\theta^z}\left(t_i, X_i^j(\theta)\right)\right)\right) \\
X_0^j = \; & \xi^j \sim \xi, \; j = 1, \cdots, B \\
Y_0^j(\theta) = \; & \mathcal{Y}_{\theta^y}(X_0^j) \\
i = \; & 1, \cdots, N-1.
\end{aligned}
$$

The **expectation solver** is described in algorithm 3. The parameter λ is chosen by trial and error.

Algorithm 3 Expectation solver

Let $\mathcal{Y}_{\theta^y}(\cdot)$ be a neural network with parameter θ^y, defined on $\mathbb{R}^d$ and valued in $\mathbb{R}^k$, $\mathcal{Z}_{\theta^z}(\cdot,\cdot)$ defined on $\mathbb{R}^+ \times \mathbb{R}^d$, $\Psi_{\theta^\Psi}(\cdot) = (\Psi^X_{\theta^\Psi}(\cdot), \Psi^Y_{\theta^\Psi}(\cdot), \Psi^Z_{\theta^\Psi}(\cdot))$ defined on $\mathbb{R}^+$ be neural networks with parameters θ^z, θ^Ψ, taking values respectively in $\mathbb{R}^{k\times d}$ and $\mathbb{R}^d \times \mathbb{R}^k \times \mathbb{R}^{k\times d}$, so that $\theta = (\theta^y, \theta^z, \theta^\Psi)$ is initialized with value $\theta_0 = (\theta^y_0, \theta^z_0, \theta^\Psi_0)$.
for m from 0 to K **do**
  Sample $(\xi^j)_{j=1,\cdots,B}$ from B independent copies of the initial condition ξ.
  Set $\forall j \in [\![1, B]\!]$, $X^j_0(\theta_m) = \xi^j \in \mathbb{R}^d, Y^j_0(\theta_m) = \mathcal{Y}_{\theta^y_m}(\xi^j) \in \mathbb{R}^k$.
  for i from 0 to $N-1$ **do**
    $u_i = \frac{1}{B}\left(\sum_{j=1}^B \varphi_1(X^j_i(\theta_m)), \sum_{j=1}^B \varphi_2(Y^j_i(\theta_m)), \sum_{j=1}^B \varphi_3\Big(\mathcal{Z}_{\theta^z_m}\left(t_i, X^j_i(\theta_m)\right)\Big)\right)$
    for j from 1 to B **do**
      Sample δ^j_i from a d-dimensional Gaussian vector.
      $X^j_{i+1}(\theta_m) = X^j_i(\theta_m) + b\left(t_i, X^j_i(\theta_m), Y^j_i(\theta_m), \mathcal{Z}_{\theta^z_m}\left(t_i, X^j_i(\theta_m)\right), \Psi_{\theta^\Psi_m}(t_i)\right)\Delta t + \sqrt{\Delta t}\,\sigma\left(t_i, X^j_i(\theta_m), \Psi^X_{\theta^\Psi_m}(t_i)\right)\delta^j_i$
      $Y^j_{i+1}(\theta_m) = Y^j_i(\theta_m) - f\left(t_i, X^j_i(\theta_m), Y^j_i(\theta_m), \mathcal{Z}_{\theta^z_m}\left(t_i, X^j_i(\theta_m)\right), \Psi_{\theta^\Psi_m}(t_i)\right)\Delta t + \sqrt{\Delta t}\,\mathcal{Z}_{\theta^z_m}\left(t_i, X^j_i(\theta_m)\right)\delta^j_i$
    end for
  end for
  $\overline{X_N}(\theta_m) = \frac{1}{B}\sum_{j=1}^B \varphi_1(X^j_N(\theta_m))$,
  $J(\theta_m) = \frac{1}{B}\sum_{j=1}^B \left(Y^j_N(\theta_m) - g\left(X^j_N(\theta_m), \overline{X_N}(\theta_m)\right)\right)^2 + \frac{\lambda}{N}\sum_{i=0}^{N-1}\left(u_i - \Psi_{\theta^\Psi_m}(t_i)\right)^2$
  Calculate $\nabla J(\theta_m)$ by back-propagation.
  Update $\theta_{m+1} = \theta_m - \rho_m \nabla J(\theta_m)$.
end for

We will compare the performances of these techniques on several examples in section 4.

3 A local solver

We also propose a local method inspired by the Deep Backward Dynamic Programming introduced by [HPW20] and [PWG21]. It considers local minimization problems between contiguous time steps. In this case there are as many networks as time steps. We replace a global optimization setting by a set of smaller problems.

In this method for $i \in [\![0, N-1]\!]$, Z_i and Y_i are approximated by a neural network $(\mathcal{Z}^i_{\theta^y_i}(\cdot), \mathcal{Y}^i_{\theta^z_i}(\cdot))$ with parameters $\theta = (\theta^y_0, \theta^z_0, \cdots, \theta^y_{N-1}, \theta^z_{N-1})$. At iteration m, with $\theta_m = (\theta^y_{m,0}, \theta^z_{m,0}, \cdots, \theta^y_{m,N-1}, \theta^z_{m,N-1})$, we simulate $X_i(\theta_m)$ with the previously computed parameters θ_m.

$$X^j_{i+1}(\theta_m) = X^j_i(\theta_m) + b\Big(t_i, X^j_i(\theta_m), \mathcal{Y}^i_{\theta^y_{m,i}}\left(X^j_i(\theta_m)\right), \mathcal{Z}^i_{\theta^z_{m,i}}\left(X^j_i(\theta_m)\right), \widetilde{u_i}\Big)\,\Delta t + \sigma\left(t_i, X^j_i(\theta_m), \widetilde{u_i}^X\right)\Delta W^j_i. \tag{13}$$

This first step allows to find the areas visited by the controlled process. Using R samples, we compute the empirical mean $\overline{m_i} = \frac{1}{R}\left(\sum_{j=1}^R X^j_i(\theta_m)\right)$ and variance $V_i = \frac{1}{R}\sum_{j=1}^R \left(X^j_i(\theta_m)\right)^2 -$

$\frac{1}{R^2}\left(\sum_{j=1}^R X_i^j(\theta_m)\right)^2$ of $X_i(\theta_m)$. We estimate u_t as in the Dynamic method (11):

$$u_i = \frac{1}{R}\left(\sum_{j=1}^R \varphi_1(X_i^j(\theta_m)), \sum_{j=1}^R \varphi_2\left(\mathcal{Y}^i_{\theta^y_{m,i}}\left(t_i, X_i^j(\theta_m)\right)\right), \sum_{j=1}^R \varphi_3\left(\mathcal{Z}^i_{\theta^z_{m,i}}\left(t_i, X_i^j(\theta_m)\right)\right)\right)$$
$$\widetilde{u_i} = \frac{\sum_{r=0}^{M-1}\zeta_{i,r} + u_i}{M+1}.$$

A priori R can be different from the batch size B. It is the batch size used for the estimation of the law with the previously computed parameters. In the numerical tests of Section 4 we use $B = 100$ or $B = 300$ for the backward optimization and a larger value $R = 50000$ for the Monte-Carlo forward estimation of the law. Then we solve backward problems to find the θ_{m+1} by sampling B independent copies of X_i through a Gaussian distribution $\mathcal{N}(\overline{m_i}, V_i^m)$ with frozen parameters θ_m :

- First sample B independent copies $X_N^1, \cdots, X_N^B$ of X_N following a Gaussian distribution $\mathcal{N}(\overline{m_N^m}, V_N^m)$. $Y^N_{\theta^y_{m+1,N}}(X_N^j)$ is set to the terminal condition $g\left(X_N^j, u_N^X\right)$.

- For i from $N-1$ to 0:
  Sample B independent copies $X_i^1, \cdots, X_i^B$ of X_i following a Gaussian distribution $\mathcal{N}(\overline{m_i^m}, V_i^m)$. Diffuse according to the dynamics (13) of X, starting from $X_i^1, \cdots, X_i^B$, to obtain $X_{i+1}^1, \cdots, X_{i+1}^B$.
  Solve the local optimization problem

$$\min_{\theta=(\theta^y,\theta^z)} \frac{1}{B}\sum_{j=1}^B \Big| Y^{i+1}_{\theta^y_{m+1,i+1}}\left(X_{i+1}^j\right) - Y^i_{\theta^y}\left(X_i^j\right) + f\left(t_i, X_i^j, \mathcal{Y}^i_{\theta^y}\left(X_i^j\right), \mathcal{Z}^i_{\theta^z}\left(X_i^j\right), \widetilde{u_i}\right)\Delta t$$
$$- \mathcal{Z}^i_{\theta^z}\left(X_i^j\right)\Delta W_i^j\Big|^2,$$

  starting from the parameter value $\theta_{m+1,i+1}$. We can then update the θ value by denoting as $\theta_{m+1,i}$ the argmin value of the minimization problem.

- Repeat the previous steps for the iteration $m+1$ until reaching K iterations.

In the version of the **local solver** given in algorithm 4, we use the dynamic update of the expectations introduced previously in the dynamic solver of section 2. In this algorithm H stands for the number of gradient steps to perform at each step of the algorithm and R is the number of samples for the laws estimation.

Remark 3. *Because we have to learn the dynamic of the forward process, the use of a backward resolution is not as obvious as in [Hur+21; Bac+21]. We have to alternate between forward dynamic estimations and backward resolutions.*
More precisely, here we solve fully coupled FBSDEs, whereas the works [HPW20; PWG21] consider decoupled FBSDEs, where the forward process X can be simulated independently of Y, Z. Estimating the law of X and sampling from a normal distribution allows us to decouple and solve locally the FBSDEs in areas visited by X and its law. However, because of this freezing of the forward dynamics, another fixed point problem has to be solved. Other approaches for fully coupled FBSDEs like [HL20] and [Ji+20] rely on the global machine learning method initiated by [HJE17].

Algorithm 4 Local solver

Let $(\mathcal{Y}^i_{\theta^y_i}(\cdot), \mathcal{Z}^i_{\theta^z_i}(\cdot))$ be some neural networks defined on $\mathbb{R}^d$ with values in $\mathbb{R}^k \times \mathbb{R}^{k\times d}$ for $i = 0, \cdots, N-1$ and parameters $\theta = (\theta^y_0, \theta^z_0, \cdots, \theta^y_{N-1}, \theta^z_{N-1})$ initialized with values $\theta_0 = (\theta^y_{0,0}, \theta^z_{0,0}, \cdots, \theta^y_{0,N-1}, \theta^z_{0,N-1})$.
Set $\forall i \in [\![0, N]\!],\ \forall r \in [\![0, M-1]\!],\ \zeta_{i,r} = (\mathbb{E}[\xi], 0, 0)$.
for m from 0 to K **do**
  Sample δ^j_i from a d-dimensional standard Gaussian vector, $i = 0, \cdots, N$, $j = 1, \cdots, R$.
  Sample $(\xi^j)_{j=1,\cdots,B}$ from R independent copies of the initial condition ξ.
  Set $\forall j \in [\![1, R]\!],\ X^j_0(\theta_m) = \xi^j \in \mathbb{R}^d$.
  for i from 0 to N **do** ▷ Forward estimation of the laws
    $l^m_i = \frac{1}{R}\left(\sum_{j=1}^R X^j_i(\theta_m)\right)$
    $u_i = \frac{1}{R}\left(\sum_{j=1}^R \varphi_1(X^j_i(\theta_m)), \sum_{j=1}^R \varphi_2(\mathcal{Y}^i_{\theta^y_{m,i}}\left(X^j_i(\theta_m)\right)), \sum_{j=1}^R \varphi_3\left(\mathcal{Z}^i_{\theta^z_{m,i}}\left(X^j_i(\theta_m)\right)\right)\right)$
    $V^m_i = \frac{1}{R}\sum_{j=1}^R \left(X^j_i(\theta_m)\right)^2 - \frac{1}{R^2}\left(\sum_{j=1}^R X^j_i(\theta_m)\right)^2$
    $\widetilde{u_i} = \frac{\sum_{r=0}^{M-1}\zeta_{i,r}+u_i}{M+1}$
    $\zeta_{i,m\%M} = u_i(\theta_m)$
    for j from 1 to R **do**
      $X^j_{i+1}(\theta_m) = X^j_i(\theta_m) + b\left(t_i, X^j_i(\theta_m), \mathcal{Y}^i_{\theta^y_{m,i}}\left(X^j_i(\theta_m)\right), \mathcal{Z}^i_{\theta^z_{m,i}}\left(X^j_i(\theta_m)\right), \widetilde{u_i}\right)\Delta t + \sqrt{\Delta t}\,\sigma\left(t_i, X^j_i(\theta_m), \widetilde{u_i}^X\right)\delta^j_i$
    end for
  end for
  for i from $N-1$ to 0 **do** ▷ Backward resolution
    $\hat{\theta}_0 = \theta_{m,i}$
    for h from 0 to $H-1$ **do** ▷ Gradient descent with simulated data for X
      for j from 1 to B **do**
        Sample Ξ^j_i, Θ^j_i from d-dimensional standard Gaussian vectors.
        $x^j_i = l^m_i + \sqrt{V^m_i}\,\Theta^j_i$
        $x^j_{i+1} = x^j_i + b\left(t_i, x^j_i, \mathcal{Y}^i_{\hat{\theta}^y_h}\left(x^j_i\right), \mathcal{Z}^i_{\hat{\theta}^z_h}\left(x^j_i\right), \widetilde{u_i}\right)\Delta t + \sqrt{\Delta t}\,\sigma\left(t_i, x^j_i, \widetilde{u_i}^X\right)\Xi^j_i$
        if $i = N-1$ **then**
          $Y^j_{i+1} = g\left(x^j_N, \widetilde{u_N}^X\right)$
        else
          $Y^j_{i+1} = \mathcal{Y}^{i+1}_{\theta_{m+1,i+1}}\left(x^j_{i+1}\right)$
        end if
      end for
      $J^i(\hat{\theta}_h) = \frac{1}{B}\sum_{j=1}^B \left(f\left(t_i, x^j_i, \mathcal{Y}^i_{\hat{\theta}^y_h}\left(x^j_i\right), \mathcal{Z}^i_{\hat{\theta}^z_h}\left(x^j_i\right), \widetilde{u_i}\right)\Delta t + Y^j_{i+1} - \mathcal{Y}^i_{\hat{\theta}^y_h}\left(x^j_i\right) - \sqrt{\Delta t}\,\mathcal{Z}^i_{\hat{\theta}^z_h}\left(x^j_i\right)\Xi^j_i\right)^2$
      Calculate $\nabla J^i(\hat{\theta}_h)$ by back-propagation.
      Update $\hat{\theta}_{h+1} = \hat{\theta}_h - \rho_h \nabla J^i(\hat{\theta}_h)$.
    end for
    $\theta_{m+1,i} = \hat{\theta}_H$
  end for
end for

4 Numerical results

The algorithms are implemented in Python with the Tensorflow library [Aba+16]. Each numerical experiment is conducted using a node composed of 2 Intel® Xeon® Gold 5122 Processors, 192 Go of RAM, and 2 GPU nVidia® Tesla® V100 16Go. The multi-GPU parallelization on the global solver is conducted using the Horovod library [SDB18]. The methods we test are:

- **Direct**: algorithm 1 at page 7. Batch size $B = 10000$.
- **Dynamic**: algorithm 2 at page 10. Batch size $B = 200$ and $M = 100$.
- **Expectation**: algorithm 3 at page 12. Batch size $B = 2000$.
- **Local**: algorithm 4 at page 14. Batch size $B = 300$ (Weak), $B = 100$ (Pontryagin) and $M = 20$, $R = 50000$.

If the algorithm is applied to equations coming from the Pontryagin (abbreviated in Pont.) or the Weak approach, it is specified in its name.

4.1 Linear price impact model

We use a linear-quadratic mean-field game of controls model studied in [Ang+19] and [CD18] for comparison. This model is useful for numerical tests since the analytic solution is known. The MFG of controls model for the representative player is given by:

$$\begin{aligned} \min_{\alpha\in\mathbb{A}} \quad & \mathbb{E}\left[\int_0^T \left(\frac{c_\alpha}{2}\left\|\alpha_t\right\|^2 + \frac{c_X}{2}\left\|X_t\right\|^2 - \gamma X_t\cdot u_t\right)\,\mathrm{d}t + \frac{c_g}{2}\left\|X_T\right\|^2\right]. \\ \text{subject to} \quad & X_t = x_0 + \int_0^t \alpha_s\,\mathrm{d}s + \sigma\, W_t \end{aligned} \tag{14}$$

and the fixed point $\mathbb{E}[\alpha_t] = u_t$. In this case, the mean-field interaction is exerted through the law of the control process.
The Pontryagin optimality principle gives the system:

$$\begin{cases} \mathrm{d}X_t &= -\frac{1}{c_\alpha}Y_t\,\mathrm{d}t + \sigma\,\mathrm{d}W_t \\ X_0 &= x_0 \\ \mathrm{d}Y_t &= -(c_X X_t + \frac{\gamma}{c_\alpha}\mathbb{E}[Y_t])\,\mathrm{d}t + Z_t\,\mathrm{d}W_t \\ Y_T &= c_g X_T. \end{cases} \tag{15}$$

In this case, the output Z of the neural network is a matrix of size $d\times d$ and Y is a vector of size d. The weak representation of the value function gives:

$$\begin{cases} \mathrm{d}X_t &= -\frac{1}{c_\alpha}\sigma^{-1}Z_t\,\mathrm{d}t + \sigma\,\mathrm{d}W_t \\ X_0 &= x_0 \\ \mathrm{d}Y_t &= -\left(\frac{c_X}{2}\left\|X_t\right\|^2 + \frac{\gamma}{c_\alpha}X_t\cdot\sigma^{-1}\mathbb{E}[Z_t] + \frac{1}{2c_\alpha}\left\|\sigma^{-1}Z_t\right\|^2\right)\,\mathrm{d}t + Z_t\,\mathrm{d}W_t \\ Y_T &= \frac{c_g}{2}\left\|X_T\right\|^2. \end{cases} \tag{16}$$

In this case, the output Z of the neural network is a vector of size d and Y is a scalar. Therefore we may be able work in higher dimensions.

Remark 4. *With LQ models, the dynamics of Y is linear in the Pontryagin approach and quadratic in the Weak approach. Thus the potentially high dimension of one method is counterbalanced by the complex dynamics of the other technique.*

For our numerical experiments we take $c_X = 2, x_0 = 1, \sigma = 0.7, \gamma = 2, c_\alpha = 2/3, c_g = 0.3$. If not stated otherwise, the simulations are conducted with $T = 1, d = 10, \Delta t = 0.01$.

Method \ T	0.25	0.75	1.0	1.5
Reference	**0.7709**	**0.1978**	**0.0811**	**0.0125**
Pontryagin	0.763 (1.3e-03)	0.187 (2.5e-03)	0.075 (2.7e-03)	0.012 (5.0e-03)
Dyn. Pont.	0.762 (2.3e-03)	0.189 (4.0e-03)	0.078 (5.5e-03)	0.013 (6.7e-03)
Exp. Pont. (0.1)	0.763 (1.6e-03)	0.604 (1.1e-01)	0.729 (1.1e-01)	0.803 (1.5e-01)
Exp. Pont. (1.)	0.762 (1.4e-03)	0.251 (2.7e-02)	0.467 (7.6e-02)	0.639 (1.1e-01)
Exp. Pont. (10.)	0.763 (1.5e-03)	0.216 (1.7e-02)	0.275 (3.7e-02)	0.574 (1.7e-01)
Exp. Pont. (100.)	0.776 (8.4e-03)	0.797 (1.1e-01)	1.042 (1.3e-01)	1.613 (2.6e-01)
Weak	0.778 (2.0e-03)	0.200 (1.4e-02)	0.092 (2.9e-02)	0.025 (2.0e-02)
Dyn. Weak	0.775 (4.4e-03)	0.212 (2.0e-02)	0.083 (4.1e-02)	0.016 (5.6e-02)
Exp. Weak (0.1)	0.877 (1.9e-02)	0.654 (9.9e-02)	0.595 (2.3e-01)	0.28 (6.0e-01)
Exp. Weak (1.)	0.901 (2.2e-03)	0.664 (9.8e-02)	0.617 (1.0e-01)	0.507 (1.9e-01)
Exp. Weak (10.)	0.887 (1.1e-02)	0.698 (7.3e-02)	0.6541 (6.3e-02)	0.49 (2.3e-01)
Exp. Weak (100.)	0.887 (2.0e-03)	0.650 (9.4e-02)	0.602 (9.1e-02)	0.492 (2.4e-01)
Pontryagin Loc.	0.767 (3.5e-04)	0.189 (6.3e-04)	0.076 (7.6e-04)	0.011 (7.5e-04)
Weak Loc.	0.944 (8.7e-04)	0.740 (2.6e-02)	0.692 (1.6e-02)	0.625 (2.2e-02)

Table 1: Mean of $\mathbb{E}[X_T]$ over the 10 dimensions (and standard deviation) for several maturities T (2000 iterations for global methods, 20000 iterations for local methods) on the price impact model (14). For the expectation method, the value of the λ penalization parameter is given under parenthesis.

Pontryagin	**1877 s.**
Dyn. Pontryagin	**1336 s.**
Exp. Pontryagin	**1562 s.**
Weak	**2205s.**
Dyn. Weak	**1605 s.**
Exp. Weak	**1670 s.**
Pontryagin Loc.	11627 s.
Weak Loc.	12689 s.

Table 2: Duration times of the methods (2000 iterations for global methods, 20000 iterations for local methods) on the price impact model (14) with $T = 1$. on one run

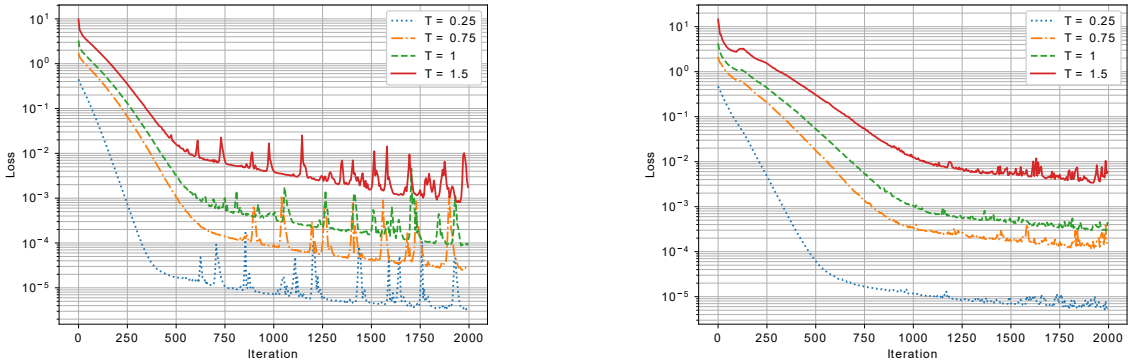

Figure 1: Learning curves for direct (left) and dynamic (right) Pontryagin method on the price impact model (14). The loss is the L^2 error between Y_T and the terminal condition of the backward equation.

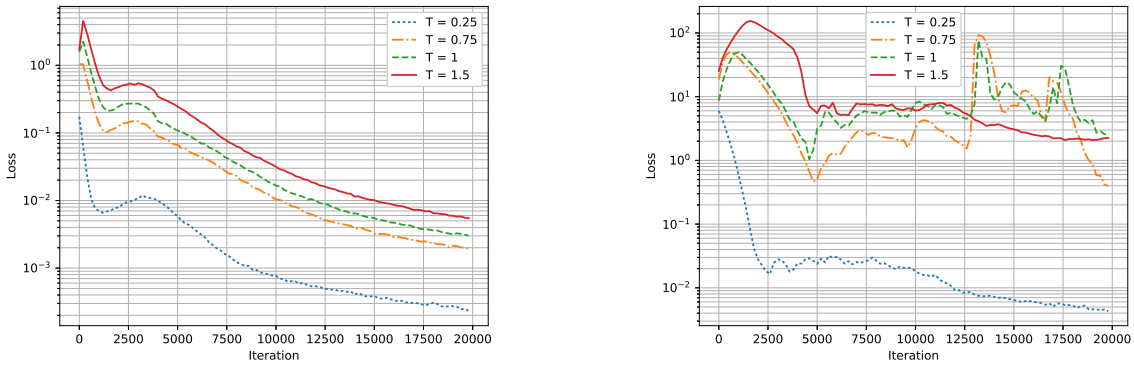

Figure 2: Learning curves for Local Pontryagin (left) and Local Weak (right) method on the price impact model (14). The loss is the sum of the local L^2 errors between the neural network Y and the Euler discretization for all time steps.

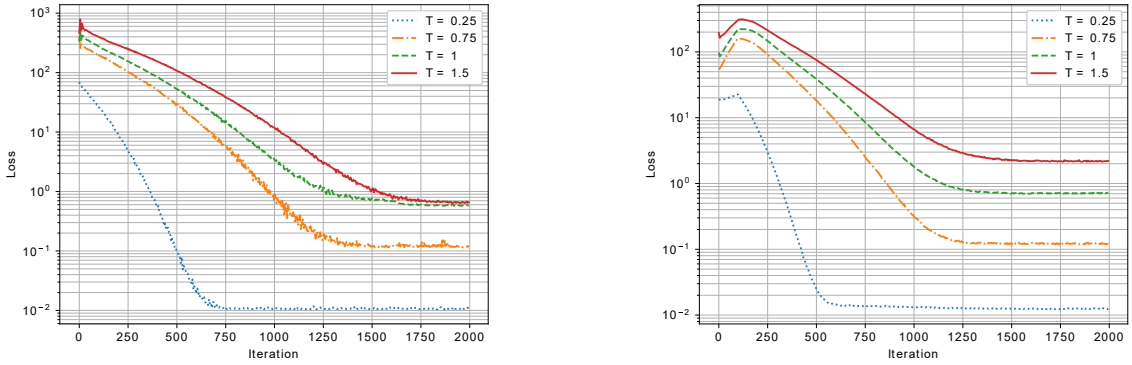

Figure 3: Learning curves for direct (left) and dynamic (right) Weak method on the price impact model (14). The loss is the L^2 error between Y_T and the terminal condition of the backward equation.

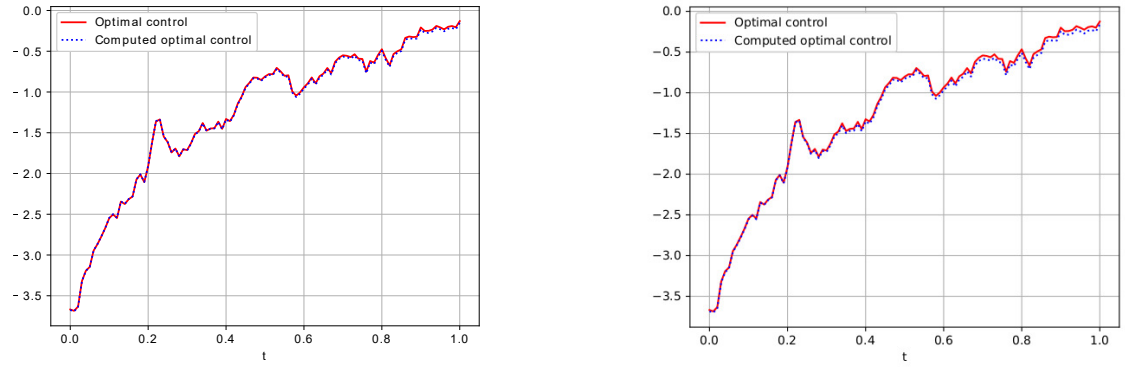

Figure 4: First coordinate of the optimal control evaluated on a sample path for direct (left) and dynamic (right) Pontryagin method after 2000 iterations on the price impact model (14) with $T = 1$.

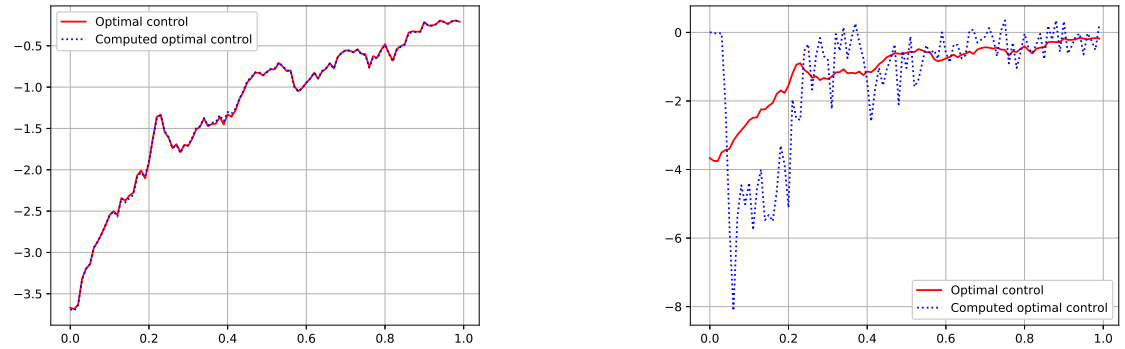

Figure 5: First coordinate of the optimal control evaluated on a sample path for Local Pontryagin (left) and Local Weak (right) Local Weak method after 20000 iterations on the price impact model (14) with $T = 1$.

All methods except Expectation Weak and Local Weak converge to the exact solution for small maturities. These two solvers do not converge to the right solution for any time horizon.

The choice of the parameter λ influences a lot the output of the Pontryagin Expectation scheme, as observed in Table 1. The best results are obtained when λ is of order 10 but we notice that a large range of values seems to work fine for very small time horizon T. However the Weak Expectation scheme never works. We won't test the expectation methods on the other test cases since they are less efficient than the other methods.

We see in Figure 2 that the Pontryagin Local method needs more iterations for the loss to stabilize than the Global method. We cannot hope for more iterations to help the convergence in the Weak method since the loss in the learning curves of Figure 3 reaches a plateau. The algorithms solving the system coming from the Pontryagin principle perform better than the others. The dynamic estimation of the expectation allows to gain training speed and to smooth the loss, as seen in Figure 1 and Table 2. As another accuracy test, we can also plot the optimal control for which we have an analytical expression. We see in Figures 4, 5 that Global and Local Pontryagin methods perform well but that the Local Weak method does not seem to converge, which confirms what is observed in Table 1.

4.2 A one-dimensional mixed model

We consider the following one-dimensional example from [CL19; Ang+19]:

$$\begin{cases} \mathrm{d}X_t &= -\rho Y_t \ \mathrm{d}t + \sigma \ \mathrm{d}W_t, \ X_0 = x_0 \\ \mathrm{d}Y_t &= \arctan(\mathbb{E}[X_t]) \ \mathrm{d}t + Z_t \ \mathrm{d}W_t, \ Y_T = \arctan(X_T). \end{cases} \tag{17}$$

This model comes from the Pontryagin principle applied to the mean-field game problem

$$\min_{\alpha} \mathbb{E}\Big[\int_0^T \big(\frac{1}{2\rho}\alpha_s^2 - X_s \arctan(u_s)\big) \ \mathrm{d}s + g(X_T)\Big]$$
$$\mathrm{d}X_t = \alpha_t \ \mathrm{d}t + \mathrm{d}W_t, \ X_0 = x_0,$$

with the fixed point $u_s = \mathbb{E}[X_s]$, and where g is an antiderivative of arctan. We take the same model parameters as in [CL19] ($T = 1$ and $x_0 = 1$) and obtain in Figure 6 with all our methods the same results as in their Figure 4. For the numerical resolution we choose 100 time steps. Notice that we use 3 hidden layers with 11 neurons in each when [CL19] uses 100 neurons by layer. We see that our smaller number of neurons is enough for this example resolution.

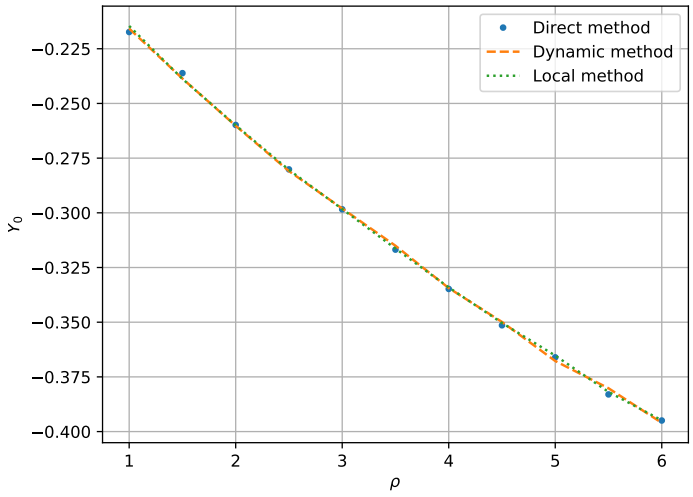

Figure 6: Value of Y_0 as a function of parameter ρ for the model (17)

The duration of the methods are given in Table 3 which illustrates again the speed gain in using the dynamic method.

Direct	**535 s.**
Dynamic	**425 s.**
Local	**10900 s.**

Table 3: Duration times of the algorithms for model (17) on one run
(2000 iterations for global methods, 20000 iterations for the local method)

4.3 Beyond the mean-field games case

In this Section we design non Linear Quadratic models in order to test the limitations of our methods. We construct general MKV FBSDES with explicit solutions following a log-normal distribution. Let X_t^i be defined by

$$\mathrm{d}X_t^i = a^i X_t^i \ \mathrm{d}t + \sigma_t^i X_t^i \ \mathrm{d}W_t^i, \tag{18}$$
$$X_0^i = \xi^i. \tag{19}$$

We obtain explicitly

$$\begin{aligned} X_t^i &= \xi^i e^{(a^i - \frac{(\sigma^i)^2}{2})t + \sigma^i W_t^i}, \\ g_t^i &:= \mathbb{E}[X_t^i] = \xi^i e^{a^i t}, \\ k_t^i &:= \mathbb{E}[(X_t^i)^2] = \xi^i e^{(2a^i + (\sigma^i)^2)t}. \end{aligned}$$

We choose $g : (t,x) \mapsto e^{\alpha t} \log\left(\prod_{i=1}^n x^i\right)$ and the following dynamic for Y_t

$$Y_t = e^{\alpha t} \log\left(\prod_i X_t^i\right) = e^{\alpha t} \sum_i \left[\log(\xi^i) + (a^i - \frac{(\sigma^i)^2}{2})t + \sigma^i W_t^i\right],$$

such that

$$\begin{aligned} c_t &:= \mathbb{E}[Y_t] = e^{\alpha t} \sum_i \left[\log(\xi^i) + \left(a^i - \frac{(\sigma^i)^2}{2}\right) t\right], \\ d_t &:= \mathbb{E}[Y_t^2] = e^{2\alpha t} \left(\left[\sum_i \left(\log(\xi^i) + a^i - \frac{(\sigma^i)^2}{2}\right) t\right]^2 + \sum_i (\sigma^i)^2 t\right). \end{aligned}$$

As we want $Y_t = u(t, X_t)$, we have $Z_t^i = \sigma_t^i X_t^i \partial_x u(t, X_t)$ following:

$$\begin{aligned} Z_t^i &= \sigma_t^i e^{\alpha t} \\ e_t^i &:= \mathbb{E}[Z_t^i] = \sigma_t^i e^{\alpha t}, \\ f_t^i &:= \mathbb{E}[(Z_t^i)^2] = (\sigma_t^i)^2 e^{2\alpha t}. \end{aligned}$$

Introducing

$$\begin{aligned} \phi(t,x) &:= \partial_t u + \sum_i a_i x_i \ \partial_{x_i} u + \sum_i \frac{(\sigma^i x_i)^2}{2} \ \partial^2_{x_i^2} u \\ &= e^{\alpha t} \left(\alpha \log\left(\prod_i x^i\right) + \sum_i \left(a^i - \frac{(\sigma^i)^2}{2}\right)\right), \end{aligned}$$

$u(t, X_t)$ solves the PDE

$$\partial_t u + \sum_i a_i x^i \ \partial_{x_i} u + \sum_i \frac{(\sigma^i)^2}{2} \ \partial^2_{x_i^2} u - \phi(t,x) = 0.$$

This semilinear PDE is related to the BSDE associated with the driver $f(t,x) = -\phi(t,x)$ for forward dynamics (18).
Using some chosen $\mathbb{R}^d$ valued functions ψ^i and $\mathbb{R}^k$ valued functions κ, we express all dynamics in a McKean-Vlasov setting:

$$\begin{cases} \mathrm{d}X_t^i & = (a^i X_t^i + \psi^i(Y_t, Z_t^i, \mathbb{E}[X_t^i], \mathbb{E}[(X_t^i)^2], \mathbb{E}[Y_t], \mathbb{E}[Y_t^2], \mathbb{E}[Z_t^i], \mathbb{E}[(Z_t^i)^2]) \\ & \quad -\psi^i\left(e^{\alpha t} \log\left(\prod_i X_t^i\right), \sigma_t^i e^{\alpha t}, g_t^i, k_t^i, c_t, d_t, e_t^i, f_t^i\right) \ \mathrm{d}t + \sigma_t^i X_t^i \ \mathrm{d}W_t^i \\ X_0^i & = \xi^i \\ \mathrm{d}Y_t & = -f(t, X_t, Y_t, Z_t, \mathbb{E}[X_t], \mathbb{E}[X_t^2], \mathbb{E}[Y_t], \mathbb{E}[Y_t^2], \mathbb{E}[Z_t], \mathbb{E}[Z_t^2]) \ \mathrm{d}t + Z_t \ \mathrm{d}W_t \\ Y_T & = e^{\alpha T} \log\left(\prod_i X_T^i\right) \end{cases} \tag{20}$$

with

$$
\begin{aligned}
&f(t, X_t, Y_t, Z_t, x_1, x_2, y_1, y_2, z_1, z_2) \\
&= -\phi(t,x) + \kappa(Y_t, Z_t, x_1, x_2, y_1, y_2, z_1, z_2) - \kappa\left(e^{\alpha t}\log\left(\prod_i X_t^i\right), \sigma_t^i e^{\alpha t}, g_t^i, k_t^i, c_t, d_t, e_t^i, f_t^i\right).
\end{aligned}
$$

and $f : \mathbb{R} \times \mathbb{R}^d \times \mathbb{R} \times \mathbb{R}^{1\times d} \times \mathbb{R}^d \times \mathbb{R}^d \times \mathbb{R} \times \mathbb{R} \times \mathbb{R}^{1\times d} \times \mathbb{R}^{1\times d} \mapsto \mathbb{R}$.
We consider two models of this kind for numerical tests.

4.3.1 A fully coupled linear example

We consider a linear McKean-Vlasov FBSDE in Y_t, Z_t and their law dynamics for X_t and Y_t:

$$
\begin{cases}
\mathrm{d}X_t^i &= (a^i X_t^i + b(Y_t + Z_t^i + \mathbb{E}[X_t^i] + \mathbb{E}[Y_t] + \mathbb{E}[Z_t^i]) \\
&\quad -b\left(e^{\alpha t}\log\left(\prod_{i=1}^d X_t^i\right) + \sigma_t^i e^{\alpha t} + g_t^i + c_t + e_t^i\right)\,\mathrm{d}t + \sigma_t^i X_t^i\,\mathrm{d}W_t^i \\
X_0^i &= \xi^i \\
\mathrm{d}Y_t &= \Big(\phi(t,X_t) + b(Y_t + \tfrac{1}{d}\sum_{i=1}^d Z_t^i + \tfrac{1}{d}\sum_{i=1}^d \mathbb{E}[X_t^i] + \mathbb{E}[Y_t] + \tfrac{1}{d}\sum_{i=1}^d \mathbb{E}[Z_t^i]) \\
&\quad -b\left(e^{\alpha t}\log\left(\prod_{i=1}^d X_t^i\right) + \tfrac{1}{d}\sum_{i=1}^d \sigma_t^i e^{\alpha t} + \tfrac{1}{d}\sum_{i=1}^d g_t^i + c_t + \tfrac{1}{d}\sum_{i=1}^d e_t^i\right)\Big)\,\mathrm{d}t + Z_t\,\mathrm{d}W_t \\
Y_T &= e^{\alpha T}\log\left(\prod_{i=1}^d X_T^i\right).
\end{cases}
\tag{21}
$$

We take $a = b = 0.1,\ \alpha = 0.5, \sigma = 0.4, \xi = 1$.

Method \ T	0.25	0.75	1.0	1.5
Reference	**1.0253**	**1.0779**	**1.1052**	**1.1618**
Global	1.025 (1.8e-03)	1.076 (3.3e-03)	1.095 (3.9e-03)	1.162 (7.2e-03)
Dyn. Global	1.026 (2.0e-03)	1.077 (3.6e-03)	1.105 (2.9e-03)	1.163 (4.8e-03)
Local	1.025 (2.3e-04)	1.092 (5.0e-04)	1.146 (7.9e-04)	1.28 (1.4e-03)

Table 4: Mean of $\mathbb{E}[X_T]$ over the 10 dimensions (and standard deviation) for several maturities T (2000 iterations for global methods, 20000 iterations for local method) on the fully coupled linear model (21).

Global	Dynamic Global	Local
2081 s.	**1308 s.**	14811 s.

Table 5: Duration times of the methods (2000 iterations for global methods, 20000 iterations for local method) on the fully coupled linear model (21) for $T = 1$.

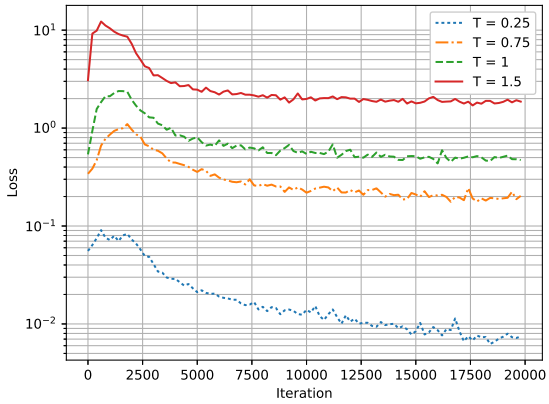

Figure 7: Learning curves for Local method on the fully coupled linear model (21). The loss is the sum of the local L^2 errors between the neural network Y and the Euler discretization for all time steps.

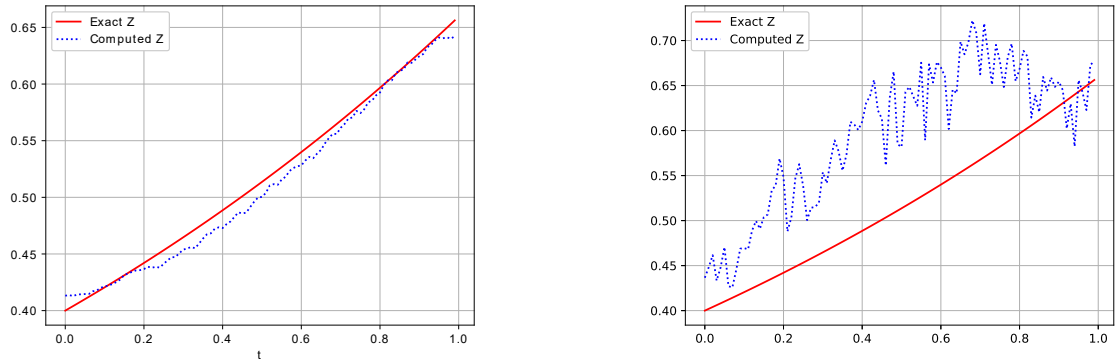

Figure 8: First coordinate of Z_t evaluated on a sample path for Dynamic Global (left) and Local (right) methods after 2000 iterations (respectively 20000) on the fully coupled linear model (21).

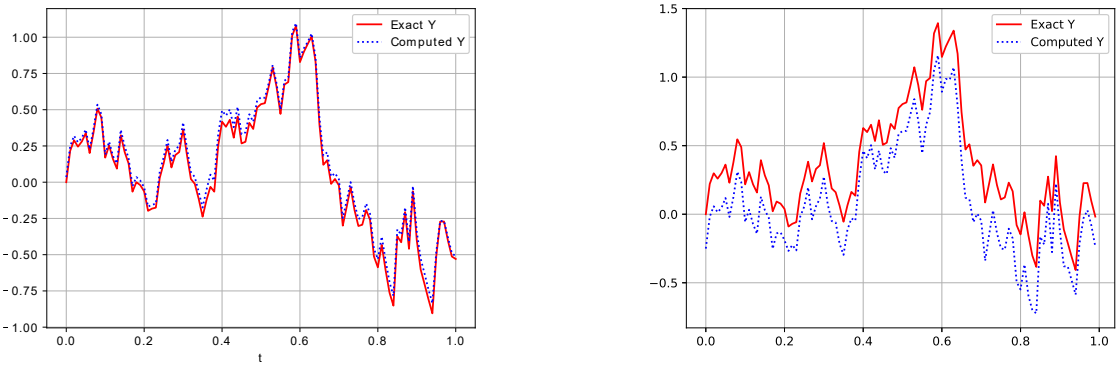

Figure 9: First coordinate of Y_t evaluated on a sample path for Direct Global (left) and Local method (right) after 2000 iterations (respectively 20000) on the fully coupled linear model (21).

The three algorithms demonstrate good performances on this test case. Both processes Y, Z are well represented by the neural network. However the local method is less precise than the global methods when the maturity grows. We see in Table 4, Figure 8, Figure 9 that the Local method is biased when $T = 1$ when the Global methods achieve a great accuracy. It looks like the results of the Local method cannot be improved since the loss flattens in Figure 7.

4.3.2 A fully coupled quadratic example

We consider a quadratic McKean-Vlasov FBSDE in Y_t, Z_t and their law dynamics for X_t and Y_t:

$$\begin{cases} \mathrm{d}X_t^i &= (a^i X_t^i + b(Y_t + Z_t^i + \mathbb{E}[X_t^i] + \mathbb{E}[Y_t] + \mathbb{E}[Z_t^i]) \\ &\quad -b\left(e^{\alpha t}\log\left(\prod_{i=1}^d X_t^i\right) + \sigma_t^i e^{\alpha t} + g_t^i + c_t + e_t^i\right) \\ &\quad +c\Bigg(Y_t^2 + (Z_t^i)^2 + \mathbb{E}[(X_t^i)^2] + \mathbb{E}[Y_t^2] + \mathbb{E}[(Z_t^i)^2]) \\ &\quad -c\left(e^{2\alpha t}\log\left(\prod_{i=1}^d X_t^i\right)^2 + (\sigma_t^i)^2 e^{2\alpha t} + (g_t^i)^2 + c_t^2 + (e_t^i)^2\right)\Bigg)\,\mathrm{d}t + \sigma_t^i X_t^i\,\mathrm{d}W_t^i \\ X_0^i &= \xi^i \\ \mathrm{d}Y_t &= \Bigg(\phi(t, X_t) + b(Y_t + \tfrac{1}{d}\textstyle\sum_{i=1}^d Z_t^i + \tfrac{1}{d}\sum_{i=1}^d \mathbb{E}[X_t^i] + \mathbb{E}[Y_t] + \tfrac{1}{d}\sum_{i=1}^d \mathbb{E}[Z_t^i]) \\ &\quad -b\left(e^{\alpha t}\log\left(\prod_{i=1}^d X_t^i\right) + \tfrac{1}{d}\textstyle\sum_{i=1}^d \sigma_t^i e^{\alpha t} + \tfrac{1}{d}\sum_{i=1}^d g_t^i + c_t + \tfrac{1}{d}\sum_{i=1}^d e_t^i\right) \\ &\quad +c(Y_t^2 + \tfrac{1}{d}\textstyle\sum_{i=1}^d (Z_t^i)^2 + \tfrac{1}{d}\sum_{i=1}^d \mathbb{E}[(X_t^i)^2] + \mathbb{E}[Y_t^2] + \tfrac{1}{d}\sum_{i=1}^d \mathbb{E}[(Z_t^i)^2]) \\ &\quad -c\left(e^{2\alpha t}\log\left(\prod_{i=1}^d X_t^i\right)^2 + \tfrac{1}{d}\textstyle\sum_{i=1}^d (\sigma_t^i)^2 e^{2\alpha t} + \tfrac{1}{d}\sum_{i=1}^d (g_t^i)^2 + c_t^2 + \tfrac{1}{d}\sum_{i=1}^d (e_t^i)^2\right)\Bigg)\,\mathrm{d}t \\ &\quad +Z_t\,\mathrm{d}W_t \\ Y_T &= e^{\alpha T}\log\left(\prod_{i=1}^d X_T^i\right). \end{cases} \tag{22}$$

We take $a = b = c = 0.1,\ \alpha = 0.5, \sigma = 0.4, \xi = 1$.

Method \ T	0.25	0.75	1.0	1.5
Reference	**1.0253**	**1.0779**	**1.1052**	**1.1618**
Global	1.024 (1.8e-03)	1.065 (4.3e-03)	12.776 (3.3e-02)	DV
Dyn. Global	1.025 (2.1e-03)	1.072 (3.1e-03)	0.961 (7.0e-03)	DV
Local	1.024 (1.6e-04)	-7.180 (9.0e-04)	0.411 (1.1e-03)	DV

Table 6: Mean of $\mathbb{E}[X_T]$ over the 10 dimensions (and standard deviation) for several maturities T (2000 iterations for global methods, 20000 iterations for local methods) on the fully coupled quadratic model (22).

Global	Dynamic Global	Local
2072 s.	**1309 s.**	14823 s.

Table 7: Duration times of the methods (2000 iterations for global methods, 20000 iterations for local methods) on the fully coupled quadratic model (22) for $T = 1$

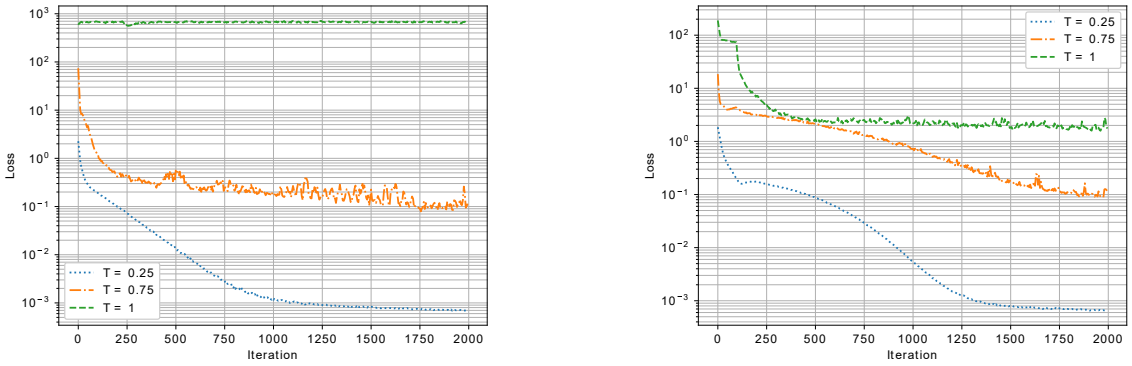

Figure 10: Learning curves for Direct Global (left) and Dynamic Global (right) method on the fully coupled quadratic model (22). The loss is the L^2 error between Y_T and the terminal condition of the backward equation.

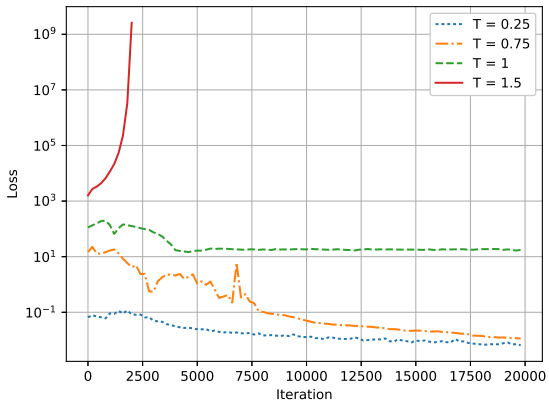

Figure 11: Learning curves for Local method on the fully coupled quadratic model (22). The loss is the sum of the local L^2 errors between the neural network Y and the Euler discretization for all time steps.

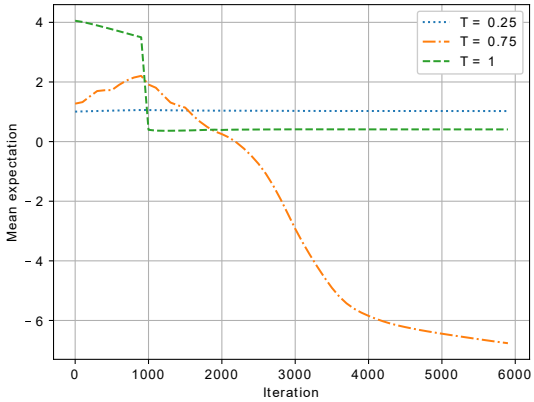

Figure 12: $\mathbb{E}[X_T]$ for Local method on the fully coupled quadratic model (22).

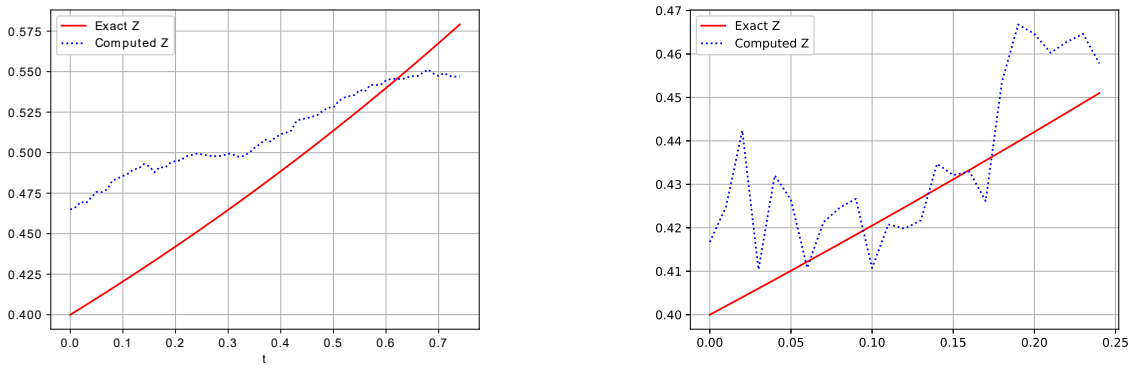

Figure 13: First coordinate of Z_t evaluated on a sample path for Direct Global (left) ($T = 0.75$) and Local method (right)($T = 0.25$) method after 2000 iterations (respectively 20000) on the fully coupled quadratic model (22).

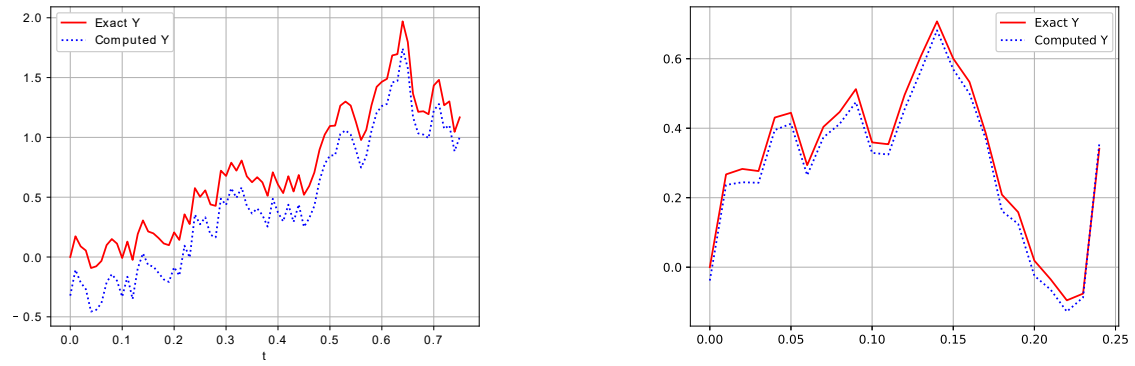

Figure 14: First coordinate of Y_t evaluated on a sample path for Dynamic Global (right) method ($T = 0.75$) and Local method (right) ($T = 0.25$) after 2000 iterations (respectively 20000 for the local method
) on the fully coupled quadratic model (22).

We observe in Table 6 convergence of the methods for small maturities and divergence beyond $T = 1$. Note that the dynamic estimation of the expectation prevents the algorithm to explode for $T = 1$, contrarily to the direct method. However, it does not converge to the true solution in this case. Indeed the loss plateaus at the value 2 in Figure 10 (right), so the terminal condition of the BSDE is not properly respected. The dynamic method also produces better result than the Direct one (see Figure 14 and Table 6). We notice from Figure 13 and Figure 14 that the estimated Y, Z processes have the good shape but some errors are still present after convergence.

Concerning the local method, we see in Figure 12 that the estimated expectations are stable around zero for a few iterations but then become negative. It may be due to the lack of a contraction for the fixed point problem. The loss explodes for $T = 1.5$, as seen on the learning curve from Figure 11. For $T = 1.$, we see that it stays above 10.

5 Conclusion

We have shown that neural network methods can solve some moderate dimensional FBSDE of McKean-Vlasov type. Comparing the different algorithms we find out that

- The dynamic update of the expectation is efficient in terms of computation speed (about 30% faster than direct method) and seems to smooth the learning curve.
- The Pontryagin approach performs better than the Weak one for large maturities. On the contrary, the Weak approach is the best for small maturities.
- For the linear model we observe no convergence problem whereas for the quadratic one we can solve only the equation on a small time horizon. However the local method is not very accurate for larger maturities.
- The local method faces more difficulties for quadratic problems than the global methods do. It also requires more iterations, hence more time, to converge.
- The Expectation methods do not work well and require to empirically choose a proper penalization parameter, which is troublesome.
- The methods can be used in dimension 10, thus applied to more realistic problems than usually. For instance, in the price impact model, the number of dimensions corresponds to the number of assets involved in the trading. Thus, developing methods able to deal with problems in high dimensions can help us to handle large portfolios.

We recommend the use of the Dynamic method which offers the best accuracy and training speed among all the tested methods. For linear quadratic mean-field games, it appears to be better to use the Weak approach for small maturities and the Pontryagin method for larger time horizons. The use of a local method is possible but requires too many iterations to converge hence it is not competitive in terms of computation time.

References

[Aba+16] M. Abadi et al. "TensorFlow: A System for Large-scale Machine Learning". In: *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*. OSDI'16. Savannah, GA, USA: USENIX Association, 2016, pp. 265–283. ISBN: 978-1-931971-33-1. URL: http://dl.acm.org/citation.cfm?id=3026877.3026899.

[ACD10] Y. Achdou and I. Capuzzo-Dolcetta. "Mean Field Games: Numerical Methods". In: *SIAM Journal on Numerical Analysis* 48 (Jan. 2010). DOI: 10.1137/090758477.

[ALG19] C. Anil, J. Lucas, and R. Grosse. "Sorting Out Lipschitz Function Approximation". In: *Proceedings of the 36th ICML*. Ed. by K. Chaudhuri and R. Salakhutdinov. Vol. 97. 2019, pp. 291–301.

[Ang+19] A. Angiuli et al. "Cemracs 2017: numerical probabilistic approach to MFG". In: *ESAIM: Proceedings and Surveys* 65 (Jan. 2019), pp. 84–113. DOI: 10.1051/proc/201965084.

[Bac+21] A. Bachouch et al. "Deep neural networks algorithms for stochastic control problems on finite horizon: numerical computations". In: *Methodol. Comput. Appl. Probab* (2021).

[Bec+19] C. Beck et al. "Deep splitting method for parabolic PDEs". In: *arXiv preprint arXiv:1907.03452* (2019).

[BEJ19] C. Beck, W. E, and A. Jentzen. "Machine Learning Approximation Algorithms for High-Dimensional Fully Nonlinear Partial Differential Equations and Second-order Backward Stochastic Differential Equations". In: *J. Nonlinear Sci.* 29.4 (Aug. 2019), pp. 1563–1619. ISSN: 1432-1467. DOI: 10.1007/s00332-018-9525-3.

[BT04] B. Bouchard and N. Touzi. "Discrete-time approximation and Monte-Carlo simulation of backward stochastic differential equations". In: *Stochastic Process. Appl.* 111.2 (2004), pp. 175 –206.

[CCD15] J.-F. Chassagneux, D. Crisan, and F. Delarue. "A probabilistic approach to classical solutions of the master equation for large population equilibria". In: *to appear in Memoirs of the AMS* (2015).

[CCD19] J.-F. Chassagneux, D. Crisan, and F. Delarue. "Numerical method for FBSDEs of McKean-Vlasov type". In: *Annals of Applied Probability* 29 (2019). DOI: 10.1214/18-AAP1429.

[CD13] R. Carmona and F. Delarue. "Probabilistic Analysis of Mean-Field Games". In: *SIAM Journal on Control and Optimization* 51.4 (2013), pp. 2705–2734. DOI: 10.1137/120883499. eprint: https://doi.org/10.1137/120883499. URL: https://doi.org/10.1137/120883499.

[CD18] R. Carmona and F. Delarue. *Probabilistic Theory of Mean Field Games with Applications I-II*. Springer, 2018.

[CL15] R. Carmona and D. Lacker. "A probabilistic weak formulation of mean field games and applications". In: *Annals of Applied Probability* 25.3 (June 2015), pp. 1189–1231. DOI: 10.1214/14-AAP1020. URL: https://doi.org/10.1214/14-AAP1020.

[CL18] P. Cardaliaguet and C.-A. Lehalle. "Mean field game of controls and an application to trade crowding". In: *Mathematics and Financial Economics* 12.3 (2018), pp. 335–363. DOI: https://doi.org/10.1007/s11579-017-0206-z.

[CL19] R. Carmona and M. Laurière. "Convergence Analysis of Machine Learning Algorithms for the Numerical Solution of Mean Field Control and Games: II – The Finite Horizon Case". In: *arXiv preprint arXiv:1908.01613, to appear in The Annals of Applied Probability* (Aug. 2019).

[CWNMW19] Q. Chan-Wai-Nam, J. Mikael, and X. Warin. "Machine Learning for Semi Linear PDEs". In: *Journal of Scientific Computing* 79 (Feb. 2019), 1667–1712. DOI: 10.1007/s10915-019-00908-3.

[FZ20] J.-P. Fouque and Z. Zhang. "Deep Learning Methods for Mean Field Control Problems with Delay". In: *Frontiers in Applied Mathematics and Statistics* 6 (2020). DOI: https://doi.org/10.3389/fams.2020.00011.

[GLW05] E. Gobet, J-P. Lemor, and X. Warin. "A regression-based Monte Carlo method to solve backward stochastic differential equations". In: *Ann. Appl. Probab.* 15.3 (2005), pp. 2172–2202.

[HJE17] J. Han, A. Jentzen, and W. E. "Solving high-dimensional partial differential equations using deep learning". In: *Proceedings of the National Academy of Sciences* 115 (July 2017). DOI: 10.1073/pnas.1718942115.

[HL20] J. Han and J. Long. "Convergence of the Deep BSDE Method for Coupled FBSDEs". In: *Probability, Uncertainty and Quantitative Risk* 5.1 (2020), pp. 1–33. DOI: https://doi.org/10.1186/s41546-020-00047-w.

[HPW20] C. Huré, H. Pham, and X. Warin. "Deep backward schemes for high-dimensional nonlinear PDEs". In: *Mathematics of Computation* 89.324 (2020), pp. 1547–1579. DOI: https://doi.org/10.1090/mcom/3514.

[Hur+21] C. Huré et al. "Deep neural networks algorithms for stochastic control problems on finite horizon: convergence analysis". In: *SIAM J. Numer. Anal.* 59.1 (2021), 525–557.

[Ji+20] S. Ji et al. "Three Algorithms for Solving High-Dimensional Fully Coupled FBSDEs Through Deep Learning". In: *IEEE Intelligent Systems* 35.3 (2020), pp. 71–84. DOI: 10.1109/MIS.2020.2971597.

[KB14] D. Kingma and J. Ba. "Adam: A Method for Stochastic Optimization". In: *International Conference on Learning Representations* (Dec. 2014).

[Lau21] M. Lauriere. "Numerical Methods for Mean Field Games and Mean Field Type Control". In: *arXiv:2106.06231* (2021).

[LL06a] J.-M. Lasry and P.-L. Lions. "Jeux à champ moyen. I – Le cas stationnaire". In: *Comptes Rendus Mathematique - C R MATH* 343 (Nov. 2006), pp. 619–625. DOI: 10.1016/j.crma.2006.09.019.

[LL06b] J.-M. Lasry and P.-L. Lions. "Jeux à champ moyen. II – Horizon fini et contrôle optimal". In: *Comptes Rendus. Mathématique. Académie des Sciences, Paris* 10 (Nov. 2006). DOI: 10.1016/j.crma.2006.09.018.

[PWG21] H. Pham, X. Warin, and M. Germain. "Neural networks-based backward scheme for fully nonlinear PDEs". In: *SN Partial Differential Equations and Applications* 2 (2021).

[SDB18] A. Sergeev and M. Del Balso. *Horovod: fast and easy distributed deep learning in TensorFlow*. Feb. 2018.