# Deep neural networks algorithms for stochastic control problems on finite horizon: numerical applications

Achref Bachouch, Côme Huré, Nicolas Langrené, Huyen Pham

# Deep neural networks algorithms for stochastic control problems on finite horizon: numerical applications*

Achref Bachouch † Côme Huré ‡ Nicolas Langrené § Huyên Pham ¶

January 25, 2020

**Abstract**

This paper presents several numerical applications of deep learning-based algorithms for discrete-time stochastic control problems in finite time horizon that have been introduced in [Hur+18]. Numerical and comparative tests using TensorFlow illustrate the performance of our different algorithms, namely control learning by performance iteration (algorithms NNcontPI and ClassifPI), control learning by hybrid iteration (algorithms Hybrid-Now and Hybrid-LaterQ), on the 100-dimensional nonlinear PDEs examples from [EHJ17] and on quadratic backward stochastic differential equations as in [CR16]. We also performed tests on low-dimension control problems such as an option hedging problem in finance, as well as energy storage problems arising in the valuation of gas storage and in microgrid management. Numerical results and comparisons to quantization-type algorithms Qknn, as an efficient algorithm to numerically solve low-dimensional control problems, are also provided.

**Keywords:** Deep learning, policy learning, performance iteration, value iteration, Monte Carlo, quantization.

*We are grateful to both referees for helpful comments and remarks.

†Department of Mathematics, University of Oslo, Norway. The author's research is carried out with support of the Norwegian Research Council, within the research project Challenges in Stochastic Control, Information and Applications (STOCONINF), project number 250768/F20 achrefb at math.uio.no

‡LPSM, University Paris Diderot hure at lpsm.paris

§CSIRO Data61, RiskLab Australia Nicolas.Langrene at data61.csiro.au

¶LPSM, University Paris-Diderot and CREST-ENSAE, pham at lspm.paris The work of this author is supported by the ANR project CAESARS (ANR-15-CE05-0024), and also by FiME and the "Finance and Sustainable Development" EDF - CACIB Chair

# 1 Introduction

This paper is devoted to the numerical resolution of discrete-time stochastic control problem over a finite horizon. The dynamics of the controlled state process $X = (X_n)_n$ valued in $\mathbb{R}^d$ is given by

$$X_{n+1} = F(X_n, \alpha_n, \varepsilon_{n+1}), \quad n = 0, \dots, N-1, \; X_0 = x_0 \in \mathbb{R}^d, \tag{1.1}$$

where $(\varepsilon_n)_n$ is a sequence of i.i.d. random variables valued in some Borel space $(E, \mathcal{B}(E))$, and defined on some probability space $(\Omega, \mathcal{F}, \mathbb{P})$ equipped with the filtration $\mathbb{F} = (\mathcal{F}_n)_n$ generated by the noise $(\varepsilon_n)_n$ ($\mathcal{F}_0$ is the trivial $\sigma$-algebra), the control $\alpha = (\alpha_n)_n$ is an $\mathbb{F}$-adapted process valued in $\mathbb{A} \subset \mathbb{R}^q$, and $F$ is a measurable function from $\mathbb{R}^d \times \mathbb{R}^q \times E$ into $\mathbb{R}^d$ which is known by the agent. Given a running cost function $f$ defined on $\mathbb{R}^d \times \mathbb{R}^q$ and a terminal cost function $g$ defined on $\mathbb{R}^d$, the cost functional associated with a control process $\alpha$ is

$$J(\alpha) = \mathbb{E}\left[\sum_{n=0}^{N-1} f(X_n, \alpha_n) + g(X_N)\right]. \tag{1.2}$$

In this framework, we assume $f$ and $g$ to be known by the agent. The set $\mathcal{A}$ of admissible controls is the set of control processes $\alpha$ satisfying some integrability conditions ensuring that the cost functional $J(\alpha)$ is well-defined and finite. The control problem, also called Markov decision process (MDP), is formulated as

$$V_0(x_0) := \inf_{\alpha \in \mathcal{A}} J(\alpha), \tag{1.3}$$

and the goal is to find an optimal control $\alpha^* \in \mathcal{A}$, i.e., attaining the optimal value: $V_0(x_0) = J(\alpha^*)$. Notice that problem (1.1)-(1.3) may also be viewed as the time discretization of a continuous time stochastic control problem, in which case, $F$ is typically the Euler scheme for a controlled diffusion process.

It is well-known that the global dynamic optimization problem (1.3) can be reduced to local optimization problems via the dynamic programming (DP) approach, which allows to determine the value function in a backward recursion by

$$\begin{aligned} V_N(x) &= g(x), \quad x \in \mathbb{R}^d, \\ V_n(x) &= \inf_{a \in \mathbb{A}} Q_n(x, a), \\ \text{with } \; Q_n(x, a) &= f(x, a) + \mathbb{E}\big[V_{n+1}(X_{n+1}) \big| X_n = x, \alpha_n = a\big], \quad (x, a) \in \mathbb{R}^d \times \mathbb{A}. \end{aligned} \tag{1.4}$$

Moreover, when the infimum is attained in the DP formula (1.4) at any time $n$ by $a_n^*(x) \in \arg\min_{a \in \mathbb{A}} Q_n(x, a)$, we get an optimal control in feedback form (policy) given by: $\alpha^* = (a_n^*(X_n^*))_n$ where $X^*$ is the Markov process defined by

$$X_{n+1}^* = F(X_n^*, a_n^*(X_n^*), \varepsilon_{n+1}), \quad n = 0, \dots, N-1, \; X_0^* = x_0.$$

The practical implementation of the DP formula may suffer from the curse of dimensionality and large complexity when the state space dimension $d$ and the control space

dimension are high. In [Hur+18], we proposed algorithms relying on deep neural networks for approximating/learning the optimal policy and then eventually the value function by performance/policy iteration or hybrid iteration with Monte Carlo regressions now or later. This research led to three algorithms, namely algorithms NNcontPI, Hybrid-Now and Hybrid-LaterQ that are recalled in Section 2, and which can be seen as a natural extension of actor-critic methods, developed in the reinforcement learning community for stationary stochastic problem ([SB98]), to finite-horizon control problems. Note that for stationary control problem, it is usual to use techniques such as temporal difference learning, which relies on the fact that the value function and the optimal control do not depend on time, to improve the learning of the latter. Such techniques do not apply to finite horizon control problems. In Section 3, we perform some numerical and comparative tests to illustrate the efficiency of our different algorithms, on 100-dimensional nonlinear PDEs examples as in [EHJ17] and quadratic Backward Stochastic Differential equations as in [CR16], as well as on high-dimensional linear quadratic stochastic control problems. We present numerical results for an option hedging problem in finance, and energy storage problems arising in the valuation of gas storage and in microgrid management. Numerical results and comparisons to quantization-type algorithms Qknn, introduced in this paper as an efficient algorithm to numerically solve low-dimensional control problems, are also provided. Finally, we conclude in Section 4 with some comments about possible extensions and improvements of our algorithms.

## 2 Algorithms

We introduce in this section four neural network-based algorithms for solving the discrete-time stochastic control problem (1.1)-(1.3). The convergence of these algorithms have been analyzed in detail in our companion paper [Hur+18], and for self-contained purpose, we recall in this section the description of these algorithms and the convergence results. We also introduce at the end of this section a quantization and $k$-nearest-neighbor-based algorithm (Qknn) that will be used as benchmark when testing our algorithms on low-dimensional control problems.

We are given a class of deep neural networks (DNN) for the control policy represented by the parametric functions $x \in \mathbb{R}^d \mapsto A(x;\beta) \in \mathbb{A}$, with parameters $\beta \in \mathbb{R}^q$, and a class of DNN for the value function represented by the parametric functions: $x \in \mathbb{R}^d \mapsto \Phi(x;\theta) \in \mathbb{R}$, with parameters $\theta \in \mathbb{R}^p$. Recall that these DNN functions $A$ and $\Phi$ are compositions of linear combinations and nonlinear activation functions, see [GBC16].

Additionally, we shall be given a sequence of probability measures on the state space $\mathbb{R}^d$, that we call training measure and denoted $(\mu_n)_{n=0}^{N-1}$, which should be seen as dataset providers to learn the optimal strategies and the value functions at time $n = 0, \ldots, N-1$.

**Remark 2.1 (Training sets design)** The choice of the training sets is critical for numerical efficiency. This problem has been largely investigated in the reinforcement learning community, notably with multi-armed bandits algorithms [ACBF02], and more recently in the numerical probability literature, see [LM19], but remains a challenging issue. Here, two

cases are considered for the choice of the training measure $\mu_n$ used to generate the training sets on which the estimates at time $n$ will be computed. The first one is a knowledge-based selection, relevant when the controller knows with a certain degree of confidence where the process has to be driven in order to optimize her cost functional. The second case is when the controller has no idea where or how to drive the process to optimize the cost functional.

(1) Exploitation only strategy

In the knowledge-based setting, there is no need for exhaustive and expensive (in time mainly) exploration of the state space, and the controller can take a training measure $\mu_n$ that assigns more points in the region of the state space that is likely to be visited by the optimally-driven process.

In practice, at time $n$, assuming we know that the optimal process is likely to lie in a region $\mathcal{D}$, we choose a training measure in which the density assigns a lot of weight to the points of $\mathcal{D}$, for example $\mathcal{U}(\mathcal{D})$, the uniform distribution in $\mathcal{D}$.

(2) Explore first, exploit later When the controller has no idea where or how to drive the process to optimize the cost functional, we suggest to build the training measures as empirical measures of the process, driven by estimates of the optimal control computed using alternative methods.

(i) *Explore first:* Use an alternative method to obtain good estimates of the optimal strategy. In high-dimension: one can for example think of approximating the control at all time by neural network, and obtain a good estimate of the optimal control by performing a global optimization of the function:

$$J(\theta_0, \dots, \theta_{N+1}) := \mathbb{E}\left[\sum_{n=0}^{N-1} f(X_n, A(X_n; \theta_n)) + g(X_N)\right],$$

where $X$ is the process controlled by the feedback control $A(.; \theta_n)$ at time $n$.

(ii) *Exploit later:* Take the training measures $\mu_n := \mathbb{P}_{X_n}$, for $n = 0, \dots, N-1$, where $X$ is driven using the optimal control estimated in step (i); and apply the procedure (1). Such an idea has been recently exploited in [KPX18].

**Remark 2.2 (Choice of Neural Networks)** Unless otherwise specified, we use feed-forward Neural Networks with two or three hidden layers and $d$+10 neurons per hidden layer, since we noticed empirically that these parameters were enough to approximate the relatively smooth objective functions considered here. We tried sigmoid, tanh, ReLU and ELU activation functions and noticed that ELU is most often the one providing the best results in our applications. We normalize the input data of each neural network in order to speed up the training of the latter. □

**Remark 2.3 (Neural Networks Training)** We use the Adam optimizer, as implemented in TensorFlow, with initial learning-rate set to 0.001 or 0.005, which are the default values in TensorFlow, to train by gradient-descent the optimal strategy and the value

function defined in the algorithms described later. TensorFlow takes care of the Adam gradient-descent procedure by automatic differentiation when the function to optimize is an expectation of TensorFlow functions, such as the usual differentiable activation functions $\sin, \log, \exp$ but also popular non-differentiable activation functions such as ReLu: $x \mapsto \max(0, x)$.

In order to force the weights and biases of the neurons to stay small, we use an $\mathbb{L}^2$ regularization with parameter mainly set to 0.01, but the value can change in order to make sure that the regularization term is neither too strong or too weak when added to the loss when training neural networks.

We consider a large enough number of mini-batches of size 64 or 128 for the training, depending essentially empirically on the dimension of the problem. We use at least 10 epochs[a] and stop the training when the loss computed on a validation set of size 100 stops decreasing. We noticed that taking more than one epoch really improves the quality of the estimates. □

**Remark 2.4 (Constraints)** The proposed algorithms can deal with state and control constraints at any time, which is useful in several applications:

$$(X_n^\alpha, \alpha_n) \quad \in \quad \mathcal{S} \quad a.s., \quad n \in \mathbb{N},$$

where $\mathcal{S}$ is some given subset of $\mathbb{R}^d \times \mathbb{R}^q$. In this case, in order to ensure that the set of admissible controls is not empty, we assume that the sets

$$\mathbb{A}(x) \quad := \quad \Big\{ a \in \mathbb{R}^q : (F(x, a, \varepsilon_1), a) \in \mathcal{S} \ a.s. \Big\}$$

are non empty for all $x \in \mathcal{S}$, and the DP formula now reads

$$V_n(x) \quad = \quad \inf_{a \in \mathbb{A}(x)} \big[ f(x, a) + P^a V_{n+1}(x) \big], \quad x \in \mathcal{S}.$$

From a computational point of view, it may be more convenient to work with unconstrained state/control variables, hence by relaxing the state/control constraint and introducing into the running cost a penalty function $L(x, a)$: $f(x, a) \leftarrow f(x, a) + L(x, a)$, and $g(x) \leftarrow g(x) + L(x, a)$. For example, if the constraint set $\mathcal{S}$ is in the form: $\mathcal{S} = \{(x, a) \in \mathbb{R}^d \times \mathbb{R}^q : h_k(x, a) = 0, k = 1, \ldots, p, \ h_k(x, a) \geq 0, k = p + 1, \ldots, q\}$, for some functions $h_k$, then one can take as penalty functions:

$$L(x, a) \quad = \quad \sum_{k=1}^{p} \mu_k |h_k(x, a)|^2 + \sum_{k=p+1}^{q} \mu_k \max(0, -h_k(x, a)).$$

where $\mu_k > 0$ are penalization coefficients (large in practice). □

## 2.1 Control Learning by Performance Iteration

We present in this section Algorithm 1, which combines an optimal policy estimation by neural networks and the dynamic programming principle. We rely on the performance iteration procedure, i.e. paths are always recomputed up to the terminal time $N$.

[a] We denote by epoch one pass of the full training set.

### 2.1.1 Algorithm NNContPI

Our first algorithm, referred to as NNContPI, is well-designed for control problems with continuous control space such as $\mathbb{R}^q$ or a ball in $\mathbb{R}^q$. The main idea is:

1. Represent the controls at time $n = 0, \dots, N-1$ by neural networks in which the activation function for the output layers takes values in the control space. For example, one can take the identity function as activation function for the output layer if the control space is $\mathbb{R}^q$; or the sigmod function if the control space is $[0, 1]$.

2. Learn sequentially in time, and in a backward way, the optimal parameters $\hat{\beta}_n$ for the representation of the optimal control. In particular, notice that the learning of the optimal control at time $n$ highly relies on the accuracy of the estimates of the optimal controls at time $k = n+1, \dots, N-1$, computed previously.

**Algorithm 1:** NNContPI

**Input:** the training distributions $(\mu_n)_{n=0}^{N-1}$;
**Output:** estimates of the optimal strategy $(\hat{a}_n)_{n=0}^{N-1}$;
**for** $n = N-1, \dots, 0$ **do**
Compute

$$\hat{\beta}_n \in \underset{\beta \in \mathbb{R}^q}{\operatorname{argmin}}\, \mathbb{E}\left[ f\big(X_n, A(X_n;\beta)\big) + \sum_{k=n+1}^{N-1} f\big(X_k^\beta, \hat{a}_k(X_k^\beta)\big) + g\big(X_N^\beta\big) \right] \tag{2.1}$$

where $X_n \sim \mu_n$ and where $\big(X_k^\beta\big)_{k=n+1}^{N}$ is defined by induction as:

$$\begin{cases} X_{n+1}^\beta &= F\big(X_n, A\big(X_n;\beta\big), \varepsilon_{n+1}\big) \\ X_{k+1}^\beta &= F\big(X_k^\beta, \hat{a}_k\big(X_k^\beta\big), \varepsilon_{k+1}\big), \quad \text{for } k = n+1, \dots, N-1. \end{cases}$$

Set $\hat{a}_n = A(.;\hat{\beta}_n)$.
▷ $\hat{a}_n$ is the estimate of the optimal policy at time $n$

### 2.1.2 Algorithm ClassifPI

In the special case where the control space $\mathbb{A}$ is finite, i.e., $\mathrm{Card}(\mathbb{A}) = L < \infty$ with $\mathbb{A} = \{a_1, \dots, a_L\}$, a classification method can be used: consider a DNN that takes state $x$ as input and returns a probability vector $p(x;\beta) = (p_\ell(x;\beta))_{\ell=1}^L$ with parameters $\beta$. Such a usual DNN can be build using $k$ hidden layers with ReLu activation functions, an output layer with $L$ neurons, and a Softmax[b] activation function for the output layer. Algorithm 2, presented below, is based on this idea, and is called ClassifPI.

[b]The Softmax function is defined as follows: $x \mapsto \left( \frac{e^{\beta_1 x}}{\sum_{k=1}^L e^{\beta_k x}}, \dots, \frac{e^{\beta_1 x}}{\sum_{k=1}^L e^{\beta_k x}} \right)$ where $\beta_1, \dots, \beta_L$ are part of the parameters that will be learned by gradient-descent.

**Algorithm 2:** ClassifPI

**Input:** the training distributions $(\mu_n)_{n=0}^{N-1}$;

**Output:** estimates of optimal strategies $(\hat{a}_n)_{n=0}^{N-1}$ and probabilities $p_l(.;\hat{\beta}_n)$;

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

Represent the discrete control at time $n$ by neural network with parameter $\beta_n$:

$$a_n(x) \;=\; a_{\ell_n(x)} \;\text{ with }\; \ell_n(x) \;\in\; \operatorname*{argmax}_{\ell=1,\ldots,L} p_\ell(x;\beta_n),$$

and compute the optimal parameter:

$$\hat{\beta}_n \;\in\; \operatorname*{argmin}_{\beta\in\mathbb{R}^q} \mathbb{E}\left[\sum_{\ell=1}^{L} p_\ell(X_n;\beta)\Big(f(X_n,a_\ell) + \sum_{k=n+1}^{N-1} f\big(X_k^\ell, \hat{a}_k(X_k^\ell)\big) \;+\; g(X_N^\ell)\Big)\right], \tag{2.2}$$

where $X_n \sim \mu_n$ on $\mathbb{R}^d$, $X_{n+1}^\ell = F(X_n, a_\ell, \varepsilon_{n+1})$, $X_{k+1}^\ell = F(X_k^\ell, \hat{a}_k(X_k^\ell), \varepsilon_{k+1})$, for $k = n+1, \ldots, N-1$ and $\ell = 1, \ldots, L$;

Set $\hat{a}_n(.) = a_{\hat{\ell}_n(.)}$ with $\hat{\ell}_n(x) \in \operatorname{argmax}_{\ell=1,\ldots,L} p_\ell(x;\hat{\beta}_n)$;

▷ $\hat{a}_n$ is the estimate of the optimal policy at time $n$

Note that, when using Algorithms 1 and 2, the estimate of the optimal strategy at time $n$ highly relies on the estimates of the optimal strategy at time $n+1, \ldots, N-1$, that have been computed previously. In particular, the practitioner who wants to use Algorithms 1 and 2 needs to keep track of the estimates of the optimal strategy at time $n+1, \ldots, N-1$ in order to compute the estimate of the optimal strategy at time $n$.

**Remark 2.5** In practice, for $n = N-1, ..., 0$, one should minimize the expectations (2.1) and (2.2) by stochastic gradient-descent, where mini-batches of finite number of paths $(X_k^\beta)_{k=n+1}^N$ are generated by drawing independent samples under $\mu_n$ for the initial position at time $n$, and independent samples under $\varepsilon_k$, for $k = n+1, \ldots, N$. The convergence of Algorithms 1 and 2 is analyzed in [Hur+18] in terms of the error approximation of the optimal control by neural networks, and in terms of the estimation error by stochastic gradient descent methods, see their Theorem 4.7. □

## 2.2 Control and value function learning by double DNN

We present in this section two algorithms, which in contrast with Algorithms 1 or 2, only keep track of the estimates of the value function and optimal control at time $n+1$ in order to build an estimate of the value function and optimal control at time $n$.

### 2.2.1 Regress Now (Hybrid-Now)

The Algorithm 3, refereed to as Hybrid-Now, combines optimal policy estimation by neural networks and dynamic programming principle, and relies on an hybrid procedure between value and performance iteration.

**Algorithm 3:** Hybrid-Now

**Input:** the training distributions $(\mu_n)_{n=0}^{N-1}$;
**Output:**
– estimate of the optimal strategy $(\hat{a}_n)_{n=0}^{N-1}$;
– estimate of the value function $(\hat{V}_n)_{n=0}^{N-1}$;
Set $\hat{V}_N = g$;
**for** $n = N-1, \ldots, 0$ **do**
Compute:

$$\hat{\beta}_n \in \operatorname*{argmin}_{\beta \in \mathbb{R}^q} \mathbb{E}\Big[ f\big(X_n, A(X_n;\beta)\big) + \hat{V}_{n+1}(X_{n+1}^{\beta}) \Big] \tag{2.3}$$

where $X_n \sim \mu$, and $X_{n+1}^{\beta} = F\big(X_n, A(X_n;\beta)\big), \varepsilon_{n+1})$;
Set $\hat{a}_n = A(.;\hat{\beta}_n)$; ▷ $\hat{a}_n$ is the estimate of the optimal policy at time $n$
Compute

$$\hat{\theta}_n \in \operatorname*{argmin}_{\theta \in \mathbb{R}^p} \mathbb{E}\left[ \Big( (f(X_n, \hat{a}_n(X_n)) + \hat{V}_{n+1}(X_{n+1}^{\hat{\beta}_n}) - \Phi(X_n;\theta) \Big)^2 \right]. \tag{2.4}$$

Set $\hat{V}_n = \Phi(.;\hat{\theta}_n)$; ▷ $\hat{V}_n$ is the estimate of the value function at time $n$

**Remark 2.6** One can combine different features from Algorithms 1, 2 and 3 to solve specific problems, as it has been done for example in Section 3.5, where we designed Algorithm 6 to solve a smart grid management problem. □

### 2.2.2 Regress Later and Quantization (Hybrid-LaterQ)

The Algorithm 4, called Hybrid-LaterQ, combines regress-later and quantization methods to build estimates of the value function. The main idea behind Algorithm 4 is to first interpolate the value function at time $n+1$ by a set of basis functions, which is in the spirit of the regress-later-based algorithms, and secondly regress the interpolation at time $n$ using quantization. The usual regress-later approach requires the ability to compute closed-form conditional expectations, which limits the stochastic dynamics and regression bases that can be considered. The use of quantization avoids this limitation and makes the regress-later algorithm more generally applicable.

Let us first recall the basic ingredients of quantization. We denote by $\hat{\varepsilon}$ a $K$-quantizer of the $\mathbb{R}^d$-valued random variable $\varepsilon_{n+1} \sim \varepsilon_1$ (typically a Gaussian random variable), that is a discrete random variable on a grid $\Gamma = \{e_1, \ldots, e_K\} \subset (\mathbb{R}^d)^K$ defined by

$$\hat{\varepsilon} = \mathrm{Proj}_{\Gamma}(\varepsilon_1) := \sum_{\ell=1}^{K} e_\ell 1_{\varepsilon_1 \in C_\ell(\Gamma)},$$

where $C_1(\Gamma)$, …, $C_K(\Gamma)$ are Voronoi tesselations of $\Gamma$, i.e., Borel partitions of the Euclidian space $(\mathbb{R}^d, |.|)$ satisfying

$$C_\ell(\Gamma) \subset \Big\{ e \in \mathbb{R}^d : |e - e_\ell| = \min_{j=1,\ldots,K} |e - e_j| \Big\}.$$

The discrete law of $\hat{\varepsilon}$ is then characterized by

$$\hat{p}_\ell \;:=\; \mathbb{P}[\hat{\varepsilon} = e_\ell] \;=\; \mathbb{P}[\varepsilon_1 \in C_\ell(\Gamma)], \quad \ell = 1, \ldots, K.$$

The grid points $(e_\ell)$ which minimize the $L^2$-quantization error $\|\varepsilon_1 - \hat{\varepsilon}\|_2$ lead to the so-called optimal $K$-quantizer, and can be obtained by a stochastic gradient descent method, known as Kohonen algorithm or competitive learning vector quantization (CLVQ) algorithm, which also provides as a byproduct an estimation of the associated weights $(\hat{p}_\ell)$. We refer to [PPP04] for a description of the algorithm, and mention that for the normal distribution, the optimal grids and the weights of the Voronoi tesselations are precomputed on the website `http://www.quantize.maths-fi.com`.

**Algorithm 4:** Hybrid-LaterQ

**Input:**

– the training distributions $(\mu_n)_{n=0}^{N-1}$;

– The grid $\{e_1, \ldots, e_K\}$ of $K$ points in $\mathbb{R}^d$, with weights $p_1, \ldots, p_K$ for the quantization of the noise $\varepsilon_n$;

**Output:**

– estimate of the optimal strategy $(\hat{a}_n)_{n=0}^{N-1}$;

– estimate of the value function $(\hat{V}_n)_{n=0}^{N-1}$;

Set $\hat{V}_N = g$;

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

Compute:

$$\hat{\beta}_n \;\in\; \underset{\beta \in \mathbb{R}^q}{\operatorname{argmin}}\, \mathbb{E}\Big[ f\big(X_n, A(X_n;\beta)\big) + \hat{V}_{n+1}(X_{n+1}^{\beta}) \Big] \tag{2.5}$$

where $X_n \sim \mu_n$, and $X_{n+1}^{\beta} = F\big(X_n, A(X_n;\beta)\big), \varepsilon_{n+1})$;

Set $\hat{a}_n = A(.;\hat{\beta}_n)$; ▷ $\hat{a}_n$ is the estimate of the optimal policy at time $n$

Compute

$$\hat{\theta}_{n+1} \;\in\; \underset{\theta \in \mathbb{R}^p}{\operatorname{argmin}}\, \mathbb{E}\left[ \Big( \hat{V}_{n+1}(X_{n+1}^{\hat{\beta}_n}) - \Phi(X_{n+1};\theta) \Big)^2 \right] \tag{2.6}$$

and set $\tilde{V}_{n+1} = \Phi(.;\hat{\theta}_{n+1})$;

▷ interpolation at time $n+1$

Set

$$\hat{V}_n(x) \;=\; f(x, \hat{a}_n(x)) + \sum_{\ell=1}^{K} p_\ell \tilde{V}_{n+1}\big(F(x, \hat{a}_n(x), e_\ell)\big);$$

▷ $\hat{V}_n$ is the estimate by quantization of the value function at time $n$

Quantization is mainly used in Algorithm 4 to efficiently approximate the expectations: recalling the dynamics (1.1), the conditional expectation operator for any functional $W$ is equal to

$$P^{\hat{a}_n^M(x)} W(x) \;=\; \mathbb{E}\big[W(X_{n+1}^{\hat{a}_n^M}) | X_n = x\big] \;=\; \mathbb{E}\big[W(F(x, \hat{a}_n^M(x), \varepsilon_1))\big], \;\; x \in \mathbb{R}^d,$$

that we shall approximate analytically by quantization via:

$$\widehat{P}^{\hat{a}_n^M(x)}W(x) \;:=\; \mathbb{E}\big[W(F(x,\hat{a}_n^M(x),\hat{\varepsilon}))\big] \;=\; \sum_{\ell=1}^{K}\hat{p}_\ell W\left(F(x,\hat{a}_n^M(x),e_\ell)\right).$$

Observe that the solution to (2.6) actually provides a neural network $\Phi(.;\hat{\theta}_{n+1})$ that interpolates $\hat{V}_{n+1}$. Hence the Algorithm 4 contains an interpolation step, and moreover, any kind of distance in $\mathbb{R}^d$ can be chosen as a loss to compute $\hat{\theta}_{n+1}$. In (2.6), we decide to take the $\mathbb{L}^2$-loss, mainly because it is the one that worked the best in our applications.

**Remark 2.7 (Quantization)** In dimension 1, we used the optimal grids and weights with $K = 21$ points, to quantize the reduced and centered normal law $\mathcal{N}(0,1)$; and took 100 points to quantize the reduced and centered normal law in dimension 2, i.e. $\mathcal{N}_2(0,1)$. All the grids and weights for the optimal quantization of the normal law in dimension $d$ are available in `http://www.quantize.maths-fi.com` for $d = 1,\dots,100$. □

### 2.2.3 Some remarks on Algorithms 3 and 4

As in Remark 2.5, all the expectations written in our pseudo-codes in Algorithm 3 and 4 should be approximated by empirical mean using a finite training set. The convergence of these algorithms has been analyzed in [Hur+18] in terms of the approximation error of the optimal control and value function by neural networks, in terms of the estimation error by stochastic gradient descent methods, and in terms of the quantization error (for Algorithm 4, see their Theorems 4.14 and 4.19).

Algorithms 3 or 4 are quite efficient to use in the usual case where the value function and the optimal control at time $n$ are very close to the value function and the optimal control at time $n+1$, which happens e.g. when the value function and the optimal control are approximations of the time discretization of a continuous in time value function and an optimal control. In this case, it is recommended to follow this two-step procedure:

(i) initialize the parameters (i.e. weights and bias) of the neural network approximations of the value function and the optimal control at time $n$ to the ones of the neural network approximations of the value function and the optimal control at time $n+1$.

(ii) take a very small learning rate parameter, for the Adam optimizer, that guarantees the stability of the parameters' updates from the gradient-descent based learning procedure.

Doing so, one obtains stable estimates of the value function and optimal control, which is desirable. We highlight the fact that this stability procedure is applicable here since the stochastic gradient descent method benefits from good initial guesses of the parameters to be optimized. It is an advantage compared to alternative methods proposed in the literature, such as classical polynomial regressions.

## 2.3 Quantization with k-nearest-neighbors (Qknn-algorithm)

Algorithm 5 presents the pseudo-code of an algorithm based on the quantization and $k$-nearest neighbors methods, called Qknn, which will be the benchmark in all the low-

dimensional control problems that will be considered in Section 3 to test NNContPI, ClassifPI, Hybrid-Now and Hybrid-Later. Also, comparisons of Algorithm 5 to other well-known algorithms on various control problems in low-dimension are performed in [Bal+19], which show in particular that Algorithm 5 works very well to solve low-dimensional control problems. Actually, in our experiments, Algorithm 5 always outperforms the other algorithms based either on regress-now or regress-later methods whenever the dimension of the problem is low enough for Algorithm 5 to be feasible.

As done in Section 2.2.2, we consider a $K$-optimal quantizer of the noise $\varepsilon_n$, i.e. a discrete random variable $\hat{\varepsilon}_n$ valued in a grid $\{e_1, \ldots, e_K\}$ of $K$ points in $E$, and with weights $p_1, \ldots, p_K$. We also consider grids $\Gamma_n$, $n = 0, \ldots, N$ of points in $\mathbb{R}^d$, which are assumed to properly cover the region of $\mathbb{R}^d$ that is likely to be visited by the optimally driven process $X$ at time $n = 0, \ldots, N-1$. These grids can be viewed as samples of well-chosen training distributions where more points are taken in the region that is likely to be visited by the optimally driven controlled process (see Remark 2.1 for details on the choice of the training measure).

**Algorithm 5:** Qknn

**Input:**

– Grids $\Gamma_k$, $k = 0, \ldots, N$ in $\mathbb{R}^d$;

– Grid $\{e_1, \ldots, e_K\}$ of $K$ points in $E$, with weights $p_1, \ldots, p_K$ for the quantization of $\varepsilon_n$

**Output:**

– estimate of the optimal strategy $(\hat{a}_n)_{n=0}^{N-1}$;

– estimate of the value function $(\hat{V}_n)_{n=0}^{N-1}$;

Set $\hat{V}_N = g$;

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

Compute for $(z, a) \in \Gamma_n \times A$,

$$\hat{Q}_n(z,a) \;=\; f(z,a) + \sum_{\ell=1}^{K} p_\ell \widehat{V}_{n+1}\Big(\mathrm{Proj}_{\Gamma_{n+1}}\big(F(z,a,e_\ell)\big)\Big), \tag{2.7}$$

where $\mathrm{Proj}_{\Gamma_{n+1}}$ is the Euclidean projection over $\Gamma_{n+1}$;

▷ $\hat{Q}_n$ is the approximated $Q$-value[c] at time $n$

Compute the optimal control at time $n$

$$\hat{A}_n(z) \in \operatorname*{argmin}_{a \in A} \big[\hat{Q}_n(z,a)\big], \quad \forall z \in \Gamma_n; \tag{2.8}$$

▷ use classical optimization algorithms of deterministic functions for this step

Set $\widehat{V}_n(z) \;=\; \hat{Q}_n\big(z, \hat{A}_n(z)\big), \quad \forall z \in \Gamma_n;$

▷ $\widehat{V}_n$ is the estimate by quantization of the value function

[c]The $Q$-value at time $n$, denoted by $Q_n$, is defined as the function that takes the couple state-action $(x, a)$ as argument, and returns the expected optimal reward earned from time $n$ to time $N$ when the process $X$ is at state $x$ and action $a$ is chosen at time $n$; i.e. $Q_n : \mathbb{R}^d \times \mathbb{R}^q \ni (x, a) \mapsto f(x, a) + \mathbb{E}^a_{n,x}[V_{n+1}(X_{n+1})]$.

**Remark 2.8** The estimate of the Q-value at time $n$ given by (2.7) is not continuous w.r.t. the control variable $a$, which might cause some stability issues when running Qknn, especially during the optimization procedure (2.8). We refer to Section 3.2.2. in [Bal+19] for a detailed presentation of an extension of Algorithm 5 where the estimates of the $Q$ value function $Q_n$ is continuous w.r.t. the control variable. □

# 3 Numerical applications

In this section, we test the Neural-Networks-based algorithms presented in Section 2 on different examples. In high-dimension, we first took the same example as already considered in [EHJ17] so that we can directly compare our results to theirs, and take another example from linear quadratic control problem with explicit analytic solution that is served as reference value. In low-dimension, we compared the results of our algorithms to the ones provided by Qknn, which has been introduced in Section 2 as an excellent benchmark for low-dimensional control problems.

## 3.1 A semilinear PDE

We consider the following semilinear PDE with quadratic growth in the gradient:

$$\left\{ \begin{array}{rcl} \dfrac{\partial v}{\partial t} + \Delta_x v \ - \ |D_x v|^2 & = & 0, \quad (t,x) \in [0,T) \times \mathbb{R}^d, \\ v(T,x) & = & g(x), \quad x \in \mathbb{R}^d. \end{array} \right. \tag{3.1}$$

By observing that for any $p \in \mathbb{R}^d$, $-|p|^2 = \inf_{a \in \mathbb{R}^d} [|a|^2 + 2a.p]$, the PDE (3.1) can be written as a Hamilton-Jacobi-Bellman equation

$$\left\{ \begin{array}{rcl} \dfrac{\partial v}{\partial t} + \Delta_x v \ + \ \inf\limits_{a \in \mathbb{R}^d} \left[ |a|^2 + 2a.D_x v \right] & = & 0, \quad (t,x) \in [0,T) \times \mathbb{R}^d, \\ v(T,x) & = & g(x), \quad x \in \mathbb{R}^d, \end{array} \right. \tag{3.2}$$

hence associated with the stochastic control problem

$$v(t,x) \ = \ \inf_{\alpha \in \mathcal{A}} \mathbb{E} \left[ \int_t^T |\alpha_s|^2 ds + g(X_T^{t,x,\alpha}) \right], \tag{3.3}$$

where $X = X^{t,x,\alpha}$ is the controlled process governed by

$$dX_s \ = \ 2\alpha_s ds \ + \ \sqrt{2} dW_s, \quad t \leq s \leq T, \ X_t = x,$$

$W$ is a $d$-dimensional Brownian motion, and the control process $\alpha$ is valued in $A = \mathbb{R}^d$. The time discretization (with time step $h = T/N$) of the control problem (3.3) leads to the discrete-time control problem (1.1)-(1.2)-(1.3) with

$$X_{n+1}^\alpha \ = \ X_n^\alpha + 2\alpha_n h + \sqrt{2h} \varepsilon_{n+1} \ =: \ F(X_n^\alpha, \alpha_n, \varepsilon_{n+1}), \quad n = 0, \ldots, N-1,$$

where $(\varepsilon_n)_n$ is a sequence of i.i.d. random variables with law $\mathcal{N}(0, \mathbb{I}_d)$, and the cost functional

$$J(\alpha) \ = \ \mathbb{E} \left[ \sum_{n=0}^{N-1} h |\alpha_n|^2 \ + \ g(X_N^\alpha) \right].$$

On the other hand, it is known that an explicit solution to (3.1) (or equivalently (3.2)) can be obtained via a Hopf-Cole transformation (see e.g. [CR16]), and is given by

$$v(t,x) \quad = \quad -\ln\Big(\mathbb{E}\Big[\exp\big(-g(x+\sqrt{2}W_{T-t})\big)\Big]\Big), \quad (t,x)\in[0,T]\times\mathbb{R}^d. \tag{3.4}$$

We choose to run tests on two different examples that have already been considered in the literature:

**Test 1** Some recent numerical results have been obtained in [EHJ17] (see Section 4.3 in [EHJ17]) when $T = 1$ and $g(x) = \ln(\frac{1}{2}(1+|x|^2))$ in dimension $d = 100$ (see Table 2 and Figure 3 in [EHJ17]). Their method is based on neural network regression to solve the BSDE representation associated with the PDE (3.1), and provide estimates of the value function at time 0 and state 0 for different values of a coefficient $\gamma$. We plotted the results of the Hybrid-Now algorithm in Figure 1. Hybrid-Now took one hour to achieves a relative error of 0.11%, using a 4-cores 3GHz intel Core i7 CPU. We want to highlight the fact that the algorithm presented in [EHJ17] only needed 330 seconds to provide a relative error of 0.17%. However, in our experience, it is difficult to reduce the relative error from 0.17% to 0.11% using their algorithm. Also, we believe that the computation time of our algorithm can easily be reduced; some ideas in this direction are discussed in Section 4. The main trick that can be used is the transfer learning (also referred to as pre-training in the literature): we rely on the continuity of the value function and the optimal control w.r.t. time to claim that the value function and the optimal control at time $n$ are very close to the ones at time $n+1$. Hence, one can initialize the weights of the value function and optimal control at time $n$ with the optimal ones estimated at step $n+1$, reduce the learning rate of the optimizer algorithm, and reduce the number of steps for the gradient descent algorithm. All this procedure really speeds up the learning of the value function and the optimal control, and insures stability of the estimates. Doing so, we were able to reduce the computation time from one hour to twenty minutes.

We also considered the same problem in dimension $d = 2$, for which we plotted the first component of $X$ w.r.t. time in Figure 2, for five different paths of the Brownian motion, where for each $\omega$, the agent follows either the naive ($\alpha = 0$) or the Hybrid-Now strategy. One can see that both strategies are very similar when the terminal time is far; but the Hybrid-Now strategy clearly forces $X$ to get closer to 0 when the terminal time gets closer, in order to reduce the terminal cost.

Let us provide further implementation details on the algorithms presented in Test 1:

- As one can guess from the representation of $v$ in (3.3), it is probably optimal to drive the process $X$ around 0. Hence we decided to take $\mu_n := (\frac{nT}{N})^{1/2}\mathcal{N}_d(0, I_d)$ as a training measure at time $n$ to learn the optimal strategy and value function at time $n$, for $n = 0, \dots, N-1$.
- We tested the algorithm with 1, 2 and 3 layers for the representation of the value function and the optimal control by neural networks, and noticed that the quality of the estimate significantly improves when using more than one layer, but does not vary significantly when considering more than 3 layers.

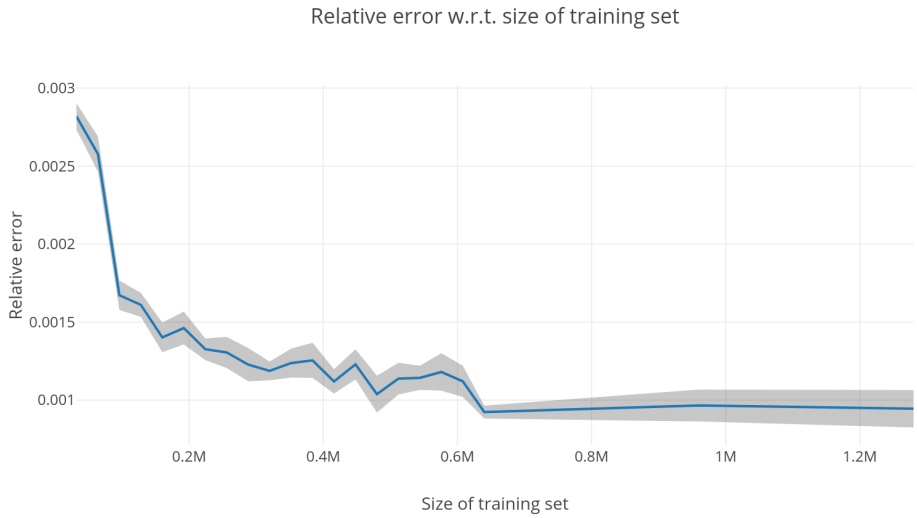


Figure 1: Relative error of the Hybrid-Now estimate of the value function at time 0 w.r.t the number of mini-batches used to build the Hybrid-Now estimators of the optimal strategy. The value functions have been computed running three times a forward Monte Carlo with a sample of size 10,000, following the optimal strategy estimated by the Hybrid-Now algorithm.

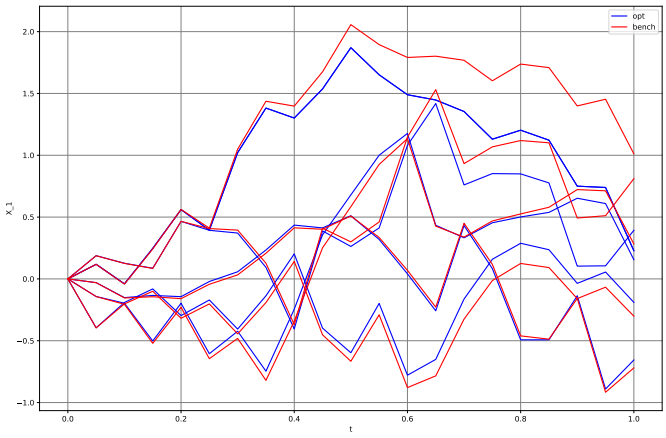


Figure 2: Five forward simulations of the first component of $X$ w.r.t. time, when the agent follows the optimal strategy estimated by the Hybrid-Now (opt in blue) and the naive strategy $\alpha = 0$ (bench in red). We consider the problem in dimension $d$=2. Observe that the optimal strategy (estimated by Hybrid-Now) is to do nothing when the terminal time is far in order to avoid any running cost, i.e. $\alpha^{opt} = 0$; and push $X$ toward 0 when the terminal time is close, in order to minimize the terminal cost.

**Test 2** Tests of the algorithms are proposed in dimension 1 with the terminal cost $g(x) = -x^\gamma \mathbb{1}_{0\leq x\leq 1} - \mathbb{1}_{1\leq x}$ and $\gamma \in (0,1)$. This problem was already considered in [Ric10], where the author proposed an algorithm based on a smart temporal discretization of the BSDE representation of the PDE (3.1) in order to deal with the quadratic growth of the driver

of the BSDE, and usual projection on basis functions techniques for the approximation of conditional expectations that appear in the dynamic programming equation associated with the BSDE. We refer to equations (13),(14),(15) in [Ric11] for details on the proposed algorithm, and its Theorem 4.14 for the convergence result. Their estimates of the value function at time 0 and state 0, when $\gamma = 1, 0.5, 0.1, 0$, are available in [Ric10], and have been reported in the column $Y\&R$ of Table 1. Also, the exact values for the value function have been computed for these values of $\gamma$ by Monte Carlo using the closed-form formula (3.4), and are reported in the column *Bench* of Table 1. Tests of the Hybrid-Now and Hybrid-LaterQ algorithms have been run, and the estimates of the value function at time 0 and state $x = 0$ are reported in the Hybrid-Now and Hybrid-LaterQ columns. We also tested Qknn and reported its results in column Qknn. Note that Qknn is particularly well-suited to 1-dimensional control problems. In particular, it is not time-consuming since the dimension of the state space is $d$=1. Actually, it provides the fastest results, which is not surprising since the other algorithms need time to learn the optimal strategy and value function through gradient-descent method at each time step $n = 0, \dots, N-1$. Moreover, Table 1 reveals that Qknn is the most accurate algorithm on this example, probably because it uses local methods in space to estimate the conditional expectation that appears in the expression of the $Q$-value.

Table 1: Value function at time 0 and state 0 w.r.t. $\gamma$, computed with the Y&R, Hybrid-Now, Hybrid-Later and Qknn algorithms. Bench reports the MC estimates of the closed-form formula (3.4).

| $\gamma$ | Y&R | Hybrid-LaterQ | Hybrid-Now | Qknn | Bench |
|---|---|---|---|---|---|
| 1.0 | -0.402 | -0.456 | -0.460 | -0.461 | -0.464 |
| 0.5 | -0.466 | -0.495 | -0.507 | -0.508 | -0.509 |
| 0.1 | -0.573 | -0.572 | -0.579 | -0.581 | -0.586 |
| 0.0 | -0.620 | -1.000 | -1.000 | -1.000 | -1.000 |

We end this paragraph by giving some implementation details for the different algorithms as part of Test 2:

- *Y&R:* The algorithm Y&R converged only when using a Lipschitz version of $g$. The following approximation was used to obtain the results in Table 1:

$$g_N(x) = \begin{cases} g(x) & \text{if } x \notin [0, N^{\frac{-1}{1-\gamma}}] \\ -Nx & \text{otherwise.} \end{cases}$$

- *Hybrid-Now:* We used $N = 40$ time steps for the time-discretization of $[0, T]$. The value functions and optimal controls at time $n = 0, \dots, N-1$ are estimated using neural networks with 3 hidden layers and 10+5+5 neurons.

- *Hybrid-LaterQ:* We used $N = 40$ time steps for the time-discretization of $[0, T]$. The value functions and optimal controls at time $n = 0, \dots, N-1$ are estimated using neural networks with 3 hidden layers containing 10+5+5 neurons; and 51 points for the quantization of the exogenous noise.

- *Qknn:* We used $N = 40$ time steps for the time-discretization of $[0, T]$. We take 51 points to quantize the exogenous noise, $\varepsilon_n \sim \mathcal{N}(0, 1)$, for $n = 0, \ldots, N$; and decided to use the 200 points of the optimal grid of $\mathcal{N}_2(0, 1)$ for the state space discretization.

The main conclusion regarding the results in this semilinear PDE problem is that Hybrid-Now provides better estimates of the solution to the PDE in dimension $d$=100 than the previous results available in [EHJ17] but requires more time to do so.

Hybrid-Now and Hybrid-Later provide better results than those available in [Ric11] to solve the PDE in dimension 2; but are outperformed by Qknn, which is arguably very accurate.

### 3.2 A linear quadratic stochastic test case

We consider a linear controlled process with dynamics in $\mathbb{R}^d$ according to

$$dX_t \quad = \quad (BX_t + C\alpha_t)dt + \sum_{j=1}^{p} D_j \alpha_t dW_t^j, \tag{3.5}$$

where $W^j$, $j = 1, \ldots, p$, are independent real-Brownian motion, the control process $\alpha \in \mathcal{A}$ is valued in $\mathbb{R}^m$, and the constant coefficients $B \in \mathbb{R}^{d\times d}$, $C, D_j \in \mathbb{R}^{d\times m}$, $j = 1, \ldots, p$. The value function of the linear quadratic stochastic control problem is

$$v(t, x) \quad = \quad \inf_{\alpha\in\mathcal{A}} \mathbb{E}\Big[ \int_t^T (X_s^{t,x,\alpha}.QX_s^{t,x,\alpha} + \lambda|\alpha_t|^2)dt + X_T^{t,x,\alpha}.PX_T^{t,x,\alpha}\Big], \quad (t, x) \in [0, T] \times \mathbb{R}^d,$$

where $X^{t,x,\alpha}$ is the solution to (3.5) starting from $x$ at time $t$, given a control process $\alpha \in \mathcal{A}$, $P, Q$ are nonnegative symmetric $d \times d$ matrices, and $\lambda > 0$. The Bellman equation associated with this stochastic control problem is a fully nonlinear equation in the form

$$\begin{aligned} \frac{\partial v}{\partial t} + x.Qx + \inf_{a\in\mathbb{R}} \big[(Bx + Ca).D_x v + a^\intercal \big(\lambda I_m + \sum_{j=1}^{p} \frac{D_j^\intercal D_x^2 v D_j}{2}\big) a\big] \quad &= \quad 0, \quad \text{on } [0, T) \times \mathbb{R}^d, \\ v(T, x) \quad &= \quad x.Px, \quad x \in \mathbb{R}^d, \end{aligned}$$

and it is well-known, see e.g. [YZ99], that an explicit solution is given by

$$v(t, x) = x.K(t)x, \tag{3.6}$$

where $K(t)$ is a nonnegative symmetric $d \times d$ matrix, solution to the Riccati equation

$$\dot{K} + B^\intercal K + KB + Q - KC(\lambda I_m + \sum_{j=1}^{p} D_j^\intercal K D_j)^{-1} C^\intercal K \quad = \quad 0, \quad K(T) = P, \tag{3.7}$$

while an optimal feedback control is equal to

$$a^*(t, x) = -\big(\lambda I_m + \sum_{j=1}^{p} D_j^\intercal K(t) D_j\big)^{-1} C^\intercal K(t)x, \quad (t, x) \in [0, T) \times \mathbb{R}^d. \tag{3.8}$$

We numerically solve this problem by considering a time discretization (with time step $h = T/N$), which leads to the discrete-time control problem with dynamics

$$X^\alpha_{n+1} \;=\; X^\alpha_n + (BX^\alpha_n + C\alpha_n)h + D\alpha_n\sqrt{h}\varepsilon_{n+1} \;=:\; F(X^\alpha_n, \alpha_n, \varepsilon_{n+1}), \quad n = 0, \dots, N-1,$$

where $(\varepsilon_n)_n$ is a sequence of i.i.d. random variables with law $\mathcal{N}(0,1)$, and cost functional

$$J(\alpha) \;=\; \mathbb{E}\left[\sum_{n=0}^{N-1} \big(X^\alpha_n.QX^\alpha_n + \lambda|\alpha_n|^2\big)h \;+\; X^\alpha_N.PX^\alpha_N\right].$$

For the numerical tests, we take $m = 1$, $p = d$, and the following parameters:

$$T \;=\; 1,\; N = 20, \qquad B = I_d,\;\; C = \mathbb{1}_d,\;\; D_j = (0, \dots, \underbrace{1}_{\text{j-th term}}, \dots, 0)^\top,\;\; j = 1, \dots, p,$$
$$Q \;=\; P \;= I_d, \quad \lambda = 1,$$

where we denote $\mathbb{1}_d := (\underbrace{1, \dots, 1}_{d \text{ times}})^\top$.

**Numerical results** We implement our algorithms in dimension $d = 1, 10, 100$, and compare our solutions with the analytic solution via the Riccati equation (3.7) solved by Matlab[d].

- For $d = 1$, we plotted the estimates of the optimal control at time $n = 0, \dots, N-1$ in Figure 3 and the value function in Figure 4. Observe that, as expected, the estimated optimal control is linear and the estimated value function is quadratic at each time.

- For $d = 10$, we reported in Table 2 the estimates of $v(0, X_0)$, computed by running forward simulations of $X$ using the estimated optimal strategy. "Riccati" is $v(0, X_0)$ computed by solving (3.7) with Matlab. We set the initial position to $X_0 = \mathbb{1}_d$. We also plotted in Figure 5 a forward simulation of the components of $X$ optimally controlled. Observe that NNContPI is more accurate than Hybrid-Now. Notice that the estimates provided by the algorithms are biased, which is due to the time discretization.

- For $d = 100$, we reported in Table 3 the estimates of the value function, computed by running forward simulations of $X$ using the estimated optimal strategy. "Riccati" is $v(0, X_0)$ computed by solving (3.7) with Matlab. We set the initial position to $X_0 = 0.1\mathbb{1}_d$ and $X_0 = 0.5\mathbb{1}_d$. Once again, NNContPI is slightly more accurate than Hybrid-Now, and the estimates provided by the latter are biased due to the time discretization.

*Implementation details:* We implemented Hybrid-Now and NNContPI using training sets from the distribution $\mu_n := \mathcal{N}_d(0,1)$ for $n = 0, \dots, N-1$. We represented the value function and optimal control at time $n$, $n = 0, \dots, N-1$ using two hidden layers with d+20 and d+10 neurons, and 1 neuron for the output layers. We used Elu as activation function for the hidden layers, and identity for the output layer.

[d] We solved (3.7) with the Matlab method ode45.

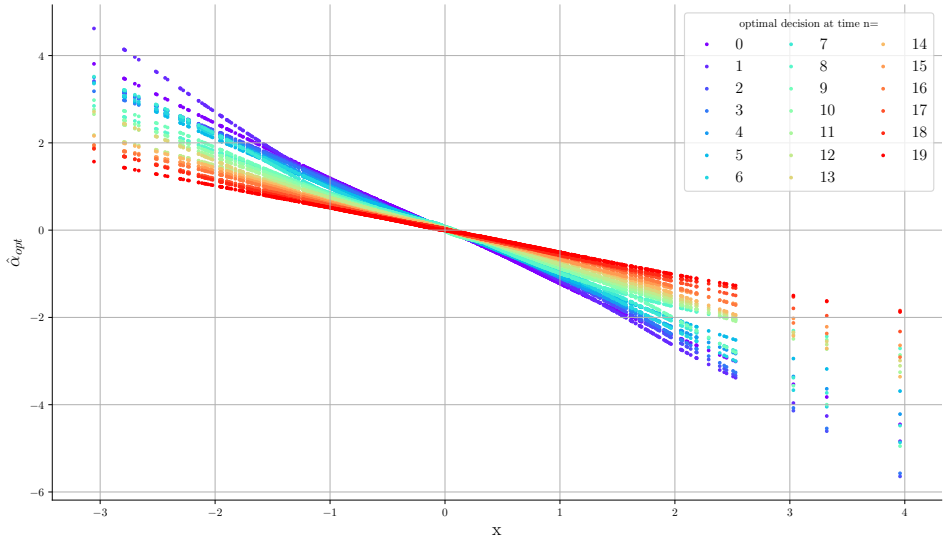


Figure 3: Optimal decision estimated by Hybrid-Now at time $n = 0, \ldots, N-1$. We took $d = 1$, $N = 20$. We observe that the estimates are linear, as expected given the closed-form formula (3.8) for the optimal control.

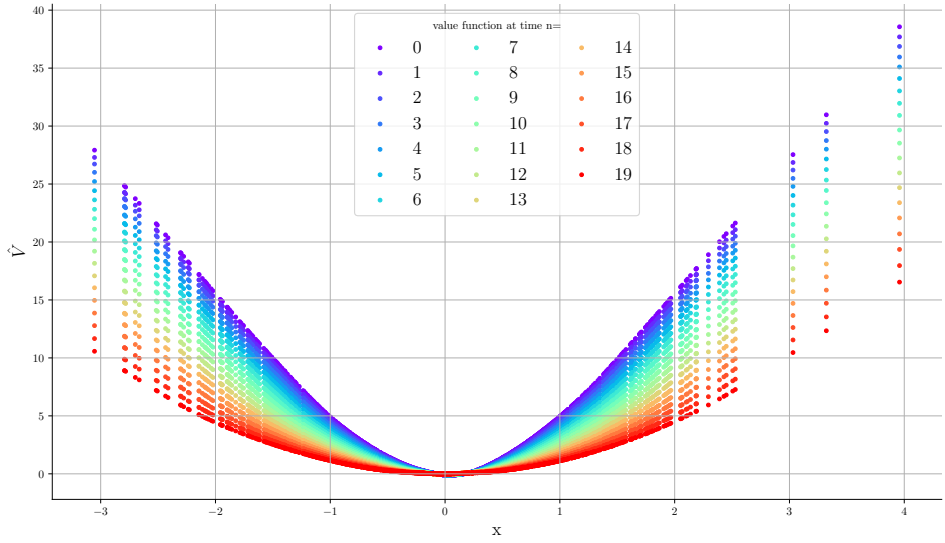


Figure 4: Value function w.r.t. $x$, estimated by Hybrid-Now at time $n = 0, \ldots, N-1$. We took $d = 1$, $N = 20$. We observe that the estimates are quadratic, as expected given the closed-form formula (3.6) for the value function.

*Comments on the algorithms:* Hybrid-Now behaved similarly as for the SemiLinear PDE example, and we can make the same remarks. NNContPI is much slower than Hybrid-Now, because the data have to go through the $N-n-1$ neural networks that represent the optimal controls at time $n+1, \dots, N-1$, in order to estimate the optimal control at time $n$.

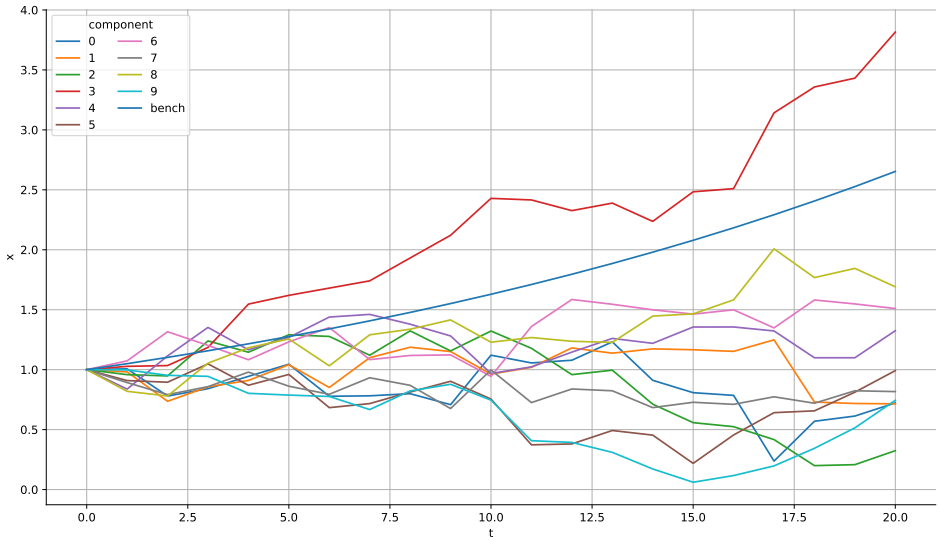


Figure 5: Forward simulation of $X$ w.r.t. time, when $X_0 = 1_d$ and $d = 10$, driven optimally using Hybrid estimates. The first ten curves represent the ten components of $X$. The bench curve represents one of the identical component of $X$ when it is driven using the strategy $\alpha = 0$. One can see that the optimal control tends to reduce the norm of each component of $X$.

Table 2: Estimate of $v(0, X_0)$ obtained by forward simulation of the process controlled by the optimal strategy estimated by Hybrid-Now and NNContPI. "Riccati" is $v(0, X_0)$ computed by solving (3.7) with Matlab. We took $d = 10$, and $X_0 = \mathbb{1}_d$. Mean and standard deviation are computed on 10 sets of 10,000 simulations each.

| | Mean | std |
|---|---|---|
| Hybrid-Now | 56.0 | 0.6 |
| NNContPI | 54.3 | 0.1 |
| Riccati | 57.1 | - |

Table 3: Estimate of $v(0, X_0)$ obtained by forward simulation of the process controlled by the optimal strategy estimated by Hybrid-Now and NNContPI. "Riccati" is $v(0, X_0)$ computed by solving (3.7) with Matlab. We took $d = 100$, and initial position $X_0 = 0.5\mathbb{1}_d$ and $X_0 = 0.1\mathbb{1}_d$. Mean and standard deviation are computed on 10 sets of 10,000 simulations each.

| | Mean | std |
|---|---|---|
| Hybrid-Now | 5.7 | 7e-3 |
| NNContPI | 5.4 | 7e-3 |
| Riccati | 5.7 | - |

Case $X_0 = 0.1\mathbb{1}_d$

| | Mean | std |
|---|---|---|
| Hybrid-Now | 137.1 | 1.3e-1 |
| NNContPI | 137.4 | 1.4e-1 |
| Riccati | 142.7 | - |

Case $X_0 = 0.5\mathbb{1}_d$

### 3.3 Option hedging

Our third example comes from a classical hedging problem in finance. We consider an investor who trades in $q$ stocks with (positive) price process $(P_n)_n$, and we denote by $(\alpha_n)$ valued in $\mathbb{A} \subset \mathbb{R}^q$ the amount held in these assets over the period $(n, n+1]$. We assume for simplicity that the price of the riskless asset is constant equal to 1 (zero interest rate). It is convenient to introduce the return process as: $R_{n+1} = \mathrm{diag}(P_n)^{-1}(P_{n+1} - P_n)$, $n = 0, \ldots, N-1$, so that the self-financed wealth process of the investor with a portfolio strategy $\alpha$, and starting from some capital $w_0$, is governed by

$$\mathcal{W}^\alpha_{n+1} \;=\; \mathcal{W}^\alpha_n + \alpha_n.R_{n+1}, \quad n = 0, \ldots, N-1, \;\; \mathcal{W}^\alpha_0 = w_0.$$

Given an option payoff $h(P_N)$, the objective of the agent is to minimize over her portfolio strategies $\alpha$ her expected square replication error

$$V_0 \;=\; \inf_{\alpha \in \mathcal{A}} \mathbb{E}\Big[\ell\big(h(P_N) - \mathcal{W}^\alpha_N\big)\Big],$$

where $\ell$ is a convex function on $\mathbb{R}$. Assuming that the returns $R_n$, $n = 1, \ldots, N$ are i.i.d, we are in a $(q+1)$-dimensional framework of Section 1 with $X^\alpha = (\mathcal{W}^\alpha, P)$ with $\varepsilon_n = R_n$ valued in $E \subset \mathbb{R}^q$, with the dynamics function

$$F(w, p, a, r) \;=\; \begin{cases} w + a.r \\ p + \mathrm{diag}(p)r, \end{cases} \qquad x = (w, p) \in \mathbb{R} \times \mathbb{R}^q, \; a \in \mathbb{R}^q, \; r \in E,$$

the running cost function $f = 0$ and the terminal cost $g(w, p) = \ell(h(p) - w)$. We test our algorithm in the case of a square loss function, i.e. $\ell(w) = w^2$, and when there is no portfolio constraints $\mathbb{A} = \mathbb{R}^q$, and compare our numerical results with the explicit solution derived in [BKL01]: denote by $\nu(dr)$ the distribution of $R_n$, by $\bar{\nu} = \mathbb{E}[R_n] = \int r\nu(dr)$ its mean, and by $\bar{M}_2 = \mathbb{E}[R_n R_n^\intercal]$ assumed to be invertible; we then have

$$V_n(w, p) \;=\; K_n w^2 - 2Z_n(p)w + C_n(p)$$

where the functions $K_n > 0$, $Z_n(p)$ and $C_n(p)$ are given in backward induction, starting from the terminal condition

$$K_N \;=\; 1, \;\; Z_N(p) \;=\; h(p), \;\; C_N(p) \;=\; h^2(p),$$

and for $n = N-1, \dots, 0$, by

$$\begin{aligned}
K_n &= K_{n+1}\big(1 - \bar{\nu}^\intercal \bar{M}_2^{-1} \bar{\nu}\big), \\
Z_n(p) &= \int Z_{n+1}(p + \mathrm{diag}(p)r)\nu(dr) - \bar{\nu}^\intercal \bar{M}_2^{-1} \int Z_{n+1}(p + \mathrm{diag}(p)r) r \nu(dr), \\
C_n(p) &= \int C_{n+1}(p + \mathrm{diag}(p)r)\nu(dr) \\
&\qquad - \frac{1}{K_{n+1}} \Big( \int Z_{n+1}(p + \mathrm{diag}(p)r) r \nu(dr) \Big)^\intercal \bar{M}_2^{-1} \Big( \int Z_{n+1}(p + \mathrm{diag}(p)r) r \nu(dr) \Big),
\end{aligned}$$

so that $V_0 = K_0 w_0^2 - 2Z_0(p_0)w_0 + C_0(p_0)$, where $p_0$ is the initial stock price. Moreover, the optimal portfolio strategy is given in feedback form by $\alpha_n^* = a_n^*(\mathcal{W}_n^*, P_n)$, where $a_n^*(w,s)$ is the function

$$a_n^*(w,p) = \bar{M}_2^{-1} \left[ \frac{\int Z_{n+1}(p + \mathrm{diag}(p)r) r \nu(dr)}{K_{n+1}} - \bar{\nu} w \right],$$

and $\mathcal{W}^*$ is the optimal wealth associated with $\alpha^*$, i.e., $\mathcal{W}_n^* = \mathcal{W}_n^{\alpha^*}$. Moreover, the initial capital $w_0^*$ that minimizes $V_0 = V_0(w_0, p_0)$, and called (quadratic) hedging price is given by

$$w_0^* = \frac{Z_0(p_0)}{K_0}.$$

**Test** Take $N = 6$, and consider one asset $q = 1$ with returns modeled by a trinomial tree:

$$\nu(dr) = \pi_+ \delta_{r_+} + \pi_0 \delta_0 + \pi_- \delta_{r_-}, \quad \pi_0 + \pi_+ + \pi_- = 1,$$

with $r_+ = 5\%$, $r_- = -5\%$, $\pi_+ = 60\%$, $\pi_- = 30\%$. Take $p_0 = 100$, and consider the call option $h(p) = (p - \kappa)_+$ with $\kappa = 100$. The price of this option is defined as the initial value of the portfolio that minimizes the terminal quadratic loss of the agent when the latter follows the optimal strategy associated with the initial value of the portfolio. In this test, we want to determine the price of the call and the associated optimal strategy using different algorithms.

**Remark 3.1** The option hedging problem is linear-quadratic, hence belongs to the class of problems where the agent has ansatzes on the optimal control and the value function. Indeed, we expect here the optimal control to be affine w.r.t. $w$ and the value function to be quadratic w.r.t. $w$. For these kind of problems, the algorithms presented in Section 2 can easily be adapted so that the expressions of the estimators satisfy the ansatzes. See (3.9) and (3.10) for the option hedging problem. □

**Numerical results** In Figure 6, we plot the value function at time 0 w.r.t $w_0$, the initial value of the portfolio, when the agent follows the theoretical optimal strategy (benchmark), and the optimal strategy estimated by the Hybrid-Now or Hybrid-LaterQ algorithms. We perform forward Monte Carlo using 10,000 samples to approximate the lower bound of the

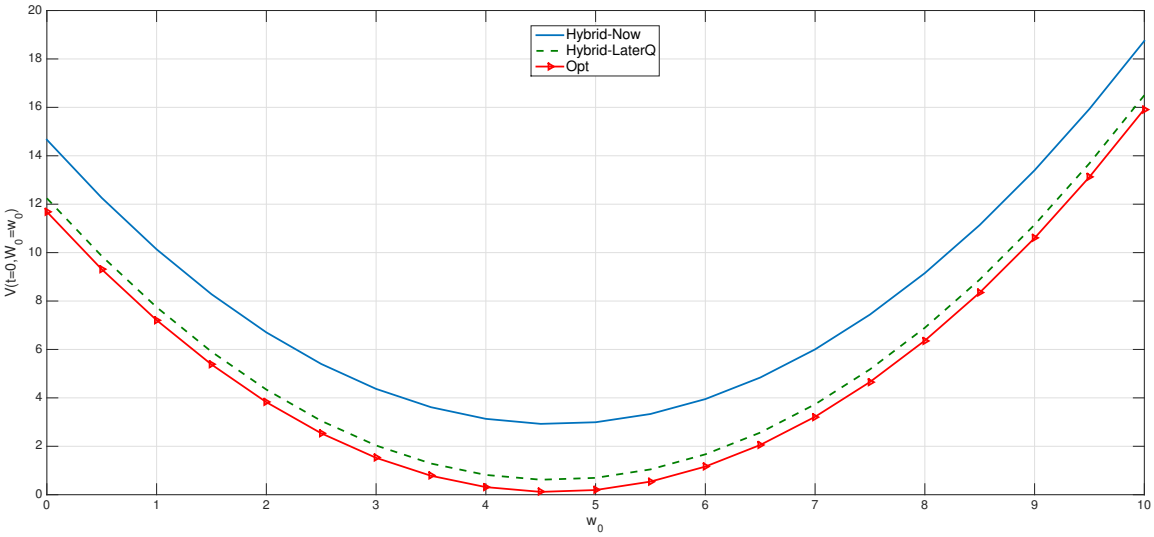


Figure 6: Estimates of the value function at time 0 w.r.t. $w_0$ using Hybrid-Now (blue line) or Hybrid-LaterQ (green dashes). We draw the value function in red for comparison. One can observe that all the algorithms estimate the price to be 4.5, but Hybrid-LaterQ is better than Hybrid-Now at reducing the quadratic risk.

value function at time 0 (see [HL17] for details on how to get an approximation of the upper-bound of the value function via duality). One can observe that while all the algorithms give a call option price approximately equal to 4.5, Hybrid-LaterQ clearly provides a better strategy than Hybrid-Now to reduce the quadratic risk of the terminal loss.

We plot in Figure 7 three different paths of the value of the portfolio w.r.t the time $n$, when the agent follows either the theoretical optimal strategy (red), or the estimated one using Hybrid-Now (blue) or Hybrid-LaterQ (green). We set $w_0 = 100$ for these simulations.

**Comments on Hybrid-Now and Hybrid-LaterQ** The Option Hedging problem belongs to the class of linear-quadratic control problems for which we expect the optimal control to be affine w.r.t. $w$ and the value function to be quadratic w.r.t. $w$. It is then natural to consider the following classes of controls $\mathcal{A}_M$ and functions $\mathcal{F}_M$ to properly approximate the optimal controls and the values functions at time $n{=}0, \dots, N-1$:

$$\mathcal{A}_M := \left\{(w,p) \mapsto A(x;\beta) \cdot \big(1,w\big)^\intercal; \quad \beta \in \mathbb{R}^p\right\}, \tag{3.9}$$

$$\mathcal{F}_M := \left\{(w,p) \mapsto \Phi(x;\theta) \cdot \big(1,w,w^2\big)^\intercal; \quad \theta \in \mathbb{R}^p\right\}, \tag{3.10}$$

where $\beta$ describes the parameters (weights+bias) associated with the neural network $A$ and $\theta$ describes those associated with the neural network $\Phi$. The notation $^\intercal$ stands for the transposition, and $\cdot$ for the inner product. Note that there are 2 (resp. 3) neurons in the output layer of $A$ (resp. $\Phi$), so that the inner product is well-defined in (3.10) and (3.9).

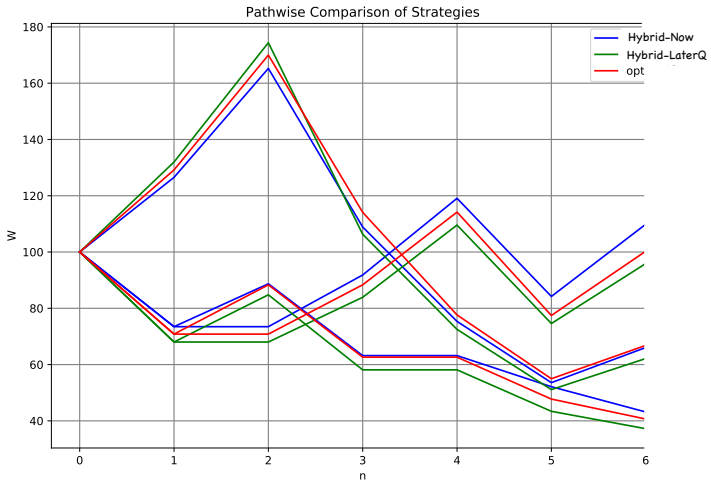


Figure 7: Three simulations of the agent's wealth w.r.t. time $n$ when, for each $\omega$, the latter follows the theoretical optimal strategy (red), the estimated one using Hybrid-Now (blue) and the one using Hybrid-LaterQ (green). We took $w_0 = 100$. Observe that the process is driven similarly to the optimally controlled process, when the agent follows the estimated optimal strategy using Hybrid-LaterQ or Hybrid-Now.

### 3.4 Valuation of energy storage

We present a discrete-time version of the energy storage valuation problem studied in [CL10]. We consider a commodity (gas) that has to be stored in a cave, e.g. salt domes or aquifers. The manager of such a cave aims to maximize the real options value by optimizing over a finite horizon $N$ the dynamic decisions to inject or withdraw gas as time and market conditions evolve. We denote by $(P_n)$ the gas price, which is an exogenous real-valued Markov process modeled by the following mean-reverting process:

$$P_{n+1} \quad = \quad \bar{p}(1-\beta) + \beta P_n + \xi_{n+1}, \tag{3.11}$$

where $\beta < 1$, and $\bar{p} > 0$ is the stationary value of the gas price. The current inventory in the gas storage is denoted by $(C_n^\alpha)_n$ and depends on the manager's decisions represented by a control process $\alpha = (\alpha_n)$ valued in $\{-1, 0, 1\}$: $\alpha_n = 1$ (resp. $-1$) means that she injects (resp. withdraws) gas with an injection (resp. withdrawal) rate $a_{in}(C_n^\alpha)$ (resp. $a_{out}(C_n^\alpha)$) requiring (causing) a purchase (resp. sale) of $b_{in}(C_n^\alpha) \geq a_{in}(C_n^\alpha)$ (resp. $b_{out}(C_n^\alpha) \leq a_{out}(C_n^\alpha)$), and $\alpha_n = 0$ means that she is doing nothing. The difference between $b_{in}$ and $a_{in}$ (resp. $b_{out}$ and $a_{out}$) indicates gas loss during injection/withdrawal. The evolution of the inventory is then governed by

$$C_{n+1}^\alpha \quad = \quad C_n^\alpha + h(C_n^\alpha, \alpha_n), \quad n = 0, \ldots, N-1, \; C_0^\alpha \; = \; c_0, \tag{3.12}$$

where we set

$$h(c,a) \;=\; \begin{cases} a_{in}(c) & \text{for } a=1 \\ 0 & \text{for } a=0 \\ -a_{out}(c) & \text{for } a=-1, \end{cases}$$

and we have the physical inventory constraint:

$$C_n^\alpha \;\in\; [C_{min}, C_{max}], \quad n=0,\dots,N.$$

The running gain of the manager at time $n$ is $f(P_n, C_n^\alpha, \alpha_t)$ given by

$$f(p,c,a) \;=\; \begin{cases} -b_{in}(c)p - K_1(c) & \text{for } a=1 \\ -K_0(c) & \text{for } a=0 \\ b_{out}(c)p - K_{-1}(c) & \text{for } a=-1, \end{cases}$$

and $K_i(c)$ represents the storage cost in each regime $i=-1,0,1$. The problem of the manager is then to maximize over $\alpha$ the expected total profit

$$J(\alpha) \;=\; \mathbb{E}\left[\sum_{n=0}^{N-1} f(P_n, C_n^\alpha, \alpha_n) + g(P_N, C_N^\alpha)\right], \tag{3.13}$$

where a common choice for the terminal condition is

$$g(p,c) \;=\; -\mu p(c_0 - c)_+,$$

which penalizes for having less gas than originally, and makes this penalty proportional to the current price of gas ($\mu > 0$). We are then in the 2-dimensional framework of Section 1 with $X^\alpha = (P, C^\alpha)$, and the set of admissible controls in the dynamic programming loop is given by:

$$A_n(c) = \big\{a \in \{-1,0,1\} : c + h(c,a) \in [C_{min}, C_{max}],\ c \in [C_{min}, C_{max}]\big\},\ \ n=0,\dots,N-1.$$

**Test** We fixed the parameters as follows, to run our numerical tests:

$$a_{in}(c) \;=\; b_{in}(c) \;=\; 0.06, \qquad a_{out}(c) \;=\; b_{out}(c) \;=\; 0.25$$
$$K_i(c) \;=\; 0.01c$$

$C_{max} = 8$, $C_{min} = 0$, $c_0 = 4$, $\bar{p} = 5$, $\beta = 0.5$, $\xi_{n+1} \rightsquigarrow \mathcal{N}(0, \sigma^2)$ with $\sigma^2 = 0.05$, and $\mu = 2$ in the terminal penalty function, $N$ =30.

**Numerical results** We plotted in Figure 8 the estimates of the value function at time 0 w.r.t. $a_{in}$ using Qknn, as well as the reward function (3.13) associated with the naive do-nothing strategy $\alpha = 0$ (see Bench in figure 8). As expected, the naive strategy performs well when $a_{in}$ is small compared to $a_{out}$, since, in this case, it takes time to fill the cave, so that the agent is likely to do nothing in order to avoid any penalization at terminal

time. When $a_{in}$ is of the same order as $a_{out}$, it is easy to fill up and empty the cave, so the agent has more freedom to buy and sell gas in the market without worrying about the terminal cost. Observe that the value function is not monotone, due to the fact that the $C$ component in the state space takes its value in a bounded and discrete set (see (3.12)).

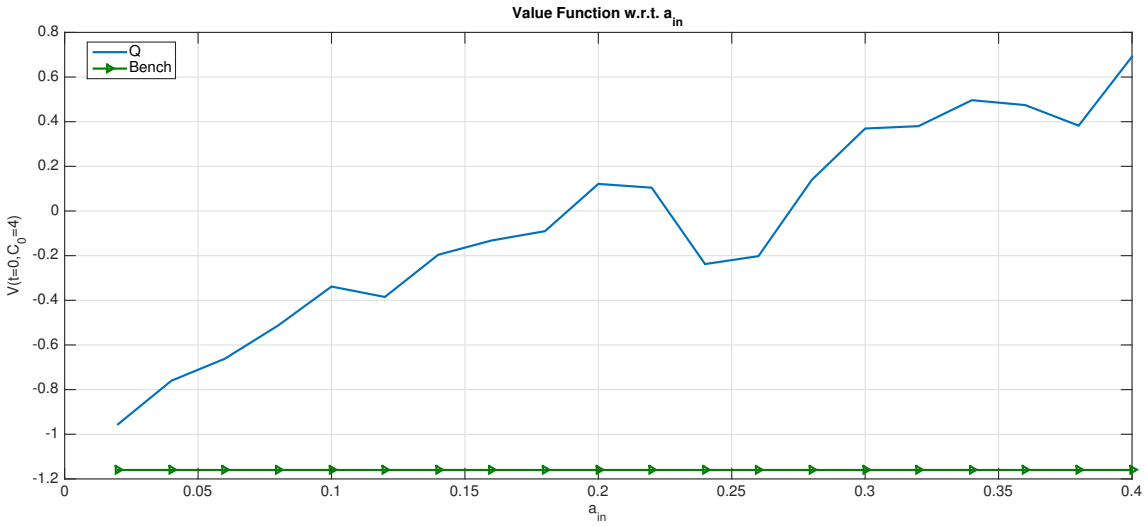


Figure 8: Estimate of the value function at time 0 w.r.t. $a_{in}$, when the agent follows the optimal strategy estimated by Qknn, by running a forward Monte Carlo with a sample of size 100,000 (blue). We also plotted the cost functional associated with the naive passive strategy $\alpha = 0$ (Bench). See that for small values of $a_{in}$ such as 0.06, doing nothing is a reasonable strategy. Observe also that the value function is not monotone w.r.t. $a_{in}$ which is due to the dynamics of $C$ (3.12).

Table 4 provides the estimates of the value function using the ClassifPI, Hybrid-Now and Qknn algorithms. Observe first that the estimates provided by Qknn are larger than those provided by the other algorithms, meaning that Qknn outperforms the other algorithms. The second best algorithm is ClassifPI, while Hybrid-Now performs poorly and clearly suffers from instability, due to the discontinuity of the running rewards w.r.t. the control variable.

Table 4: $V(0, P_0, C_0)$ estimates for different values of $a_{in}$, using the optimal strategy provided by the ClassifPI , Hybrid-Now and Qknn algorithms, with $a_{out} = 0.25$, $P_0 = 4$ and $C_0 = 4$.

| $a_{in}$ | Hybrid-Now | ClassifPI | Qknn | $\alpha = 0$ |
|---|---|---|---|---|
| 0.06 | -0.99 | -0.71 | -0.66 | -1.20 |
| 0.10 | -0.70 | -0.38 | -0.34 | -1.20 |
| 0.20 | -0.21 | 0.01 | 0.12 | -1.20 |
| 0.30 | -0.10 | 0.37 | 0.37 | -1.20 |
| 0.40 | 0.10 | 0.51 | 0.69 | -1.20 |

Finally, Figures 9, 10, 11 provide the optimal decisions w.r.t. $(P, C)$ at times 5, 10, 15, 20, 25, 29 estimated respectively by the Qknn, ClassifPI and Hybrid-Now algorithms.

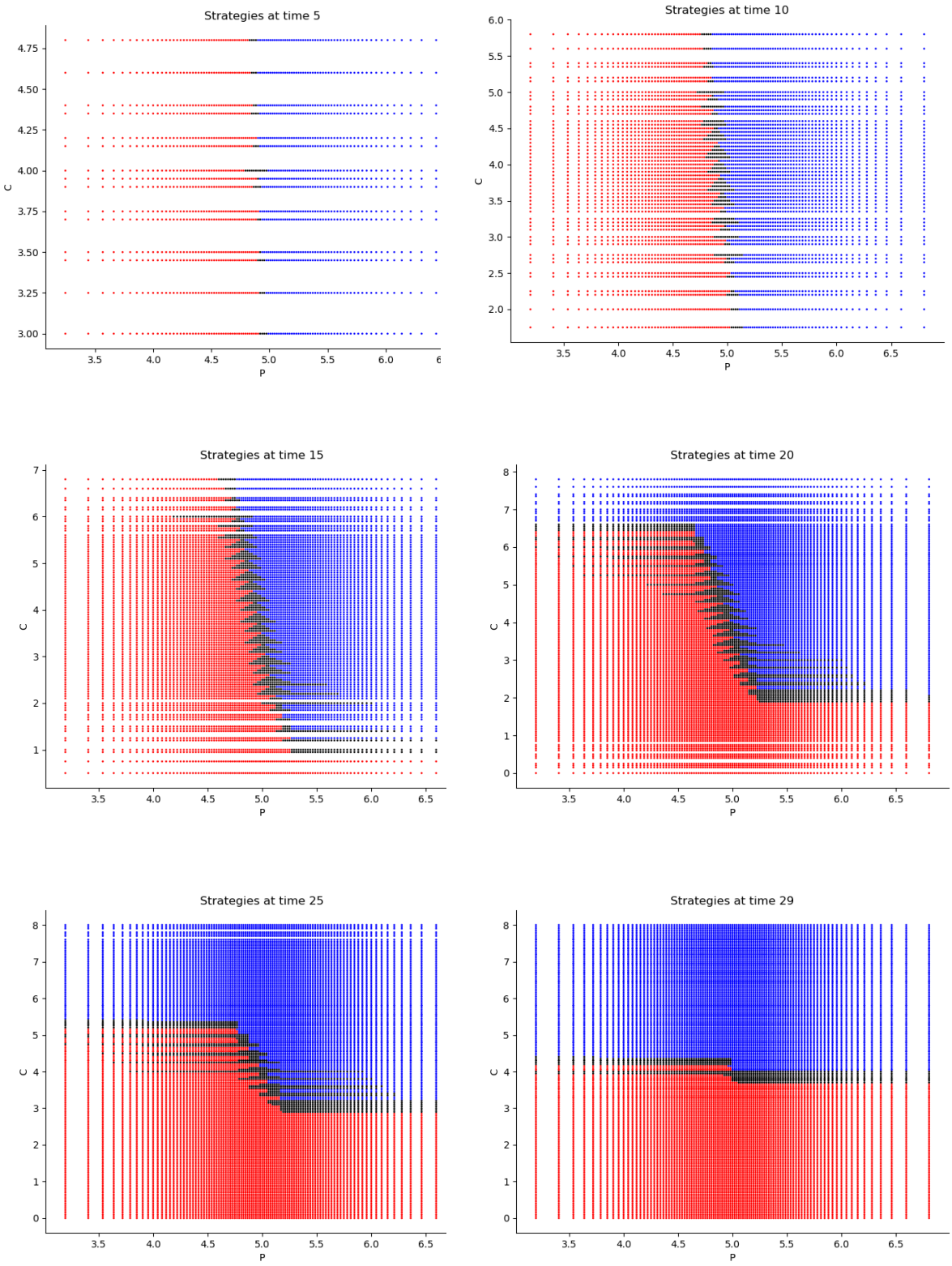


Figure 9: Estimated optimal decisions at times 5, 10, 15, 20, 25, 29 w.r.t. (P,C) for the energy storage valuation problem using **Qknn**. Injection (a=-1) in red, store (a=0) in black and withdraw (a=1) in blue.

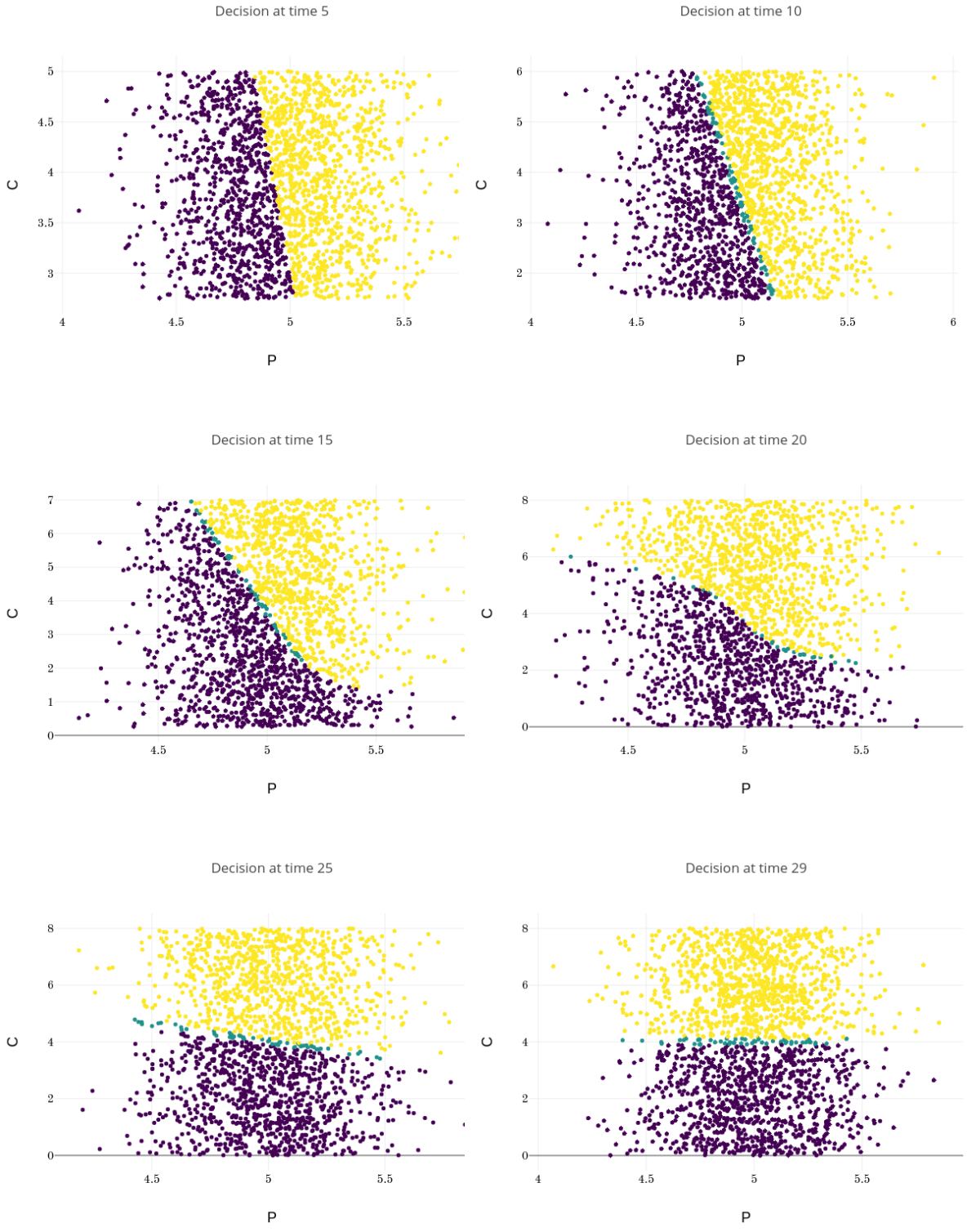


Figure 10: Estimated optimal decisions at times 5, 10, 15, 20, 25, 29 w.r.t. (P,C) for the energy storage valuation problem using **ClassifPI**. Injection (a=-1) in purple, store (a=0) in blue and withdraw (a=1) in yellow.

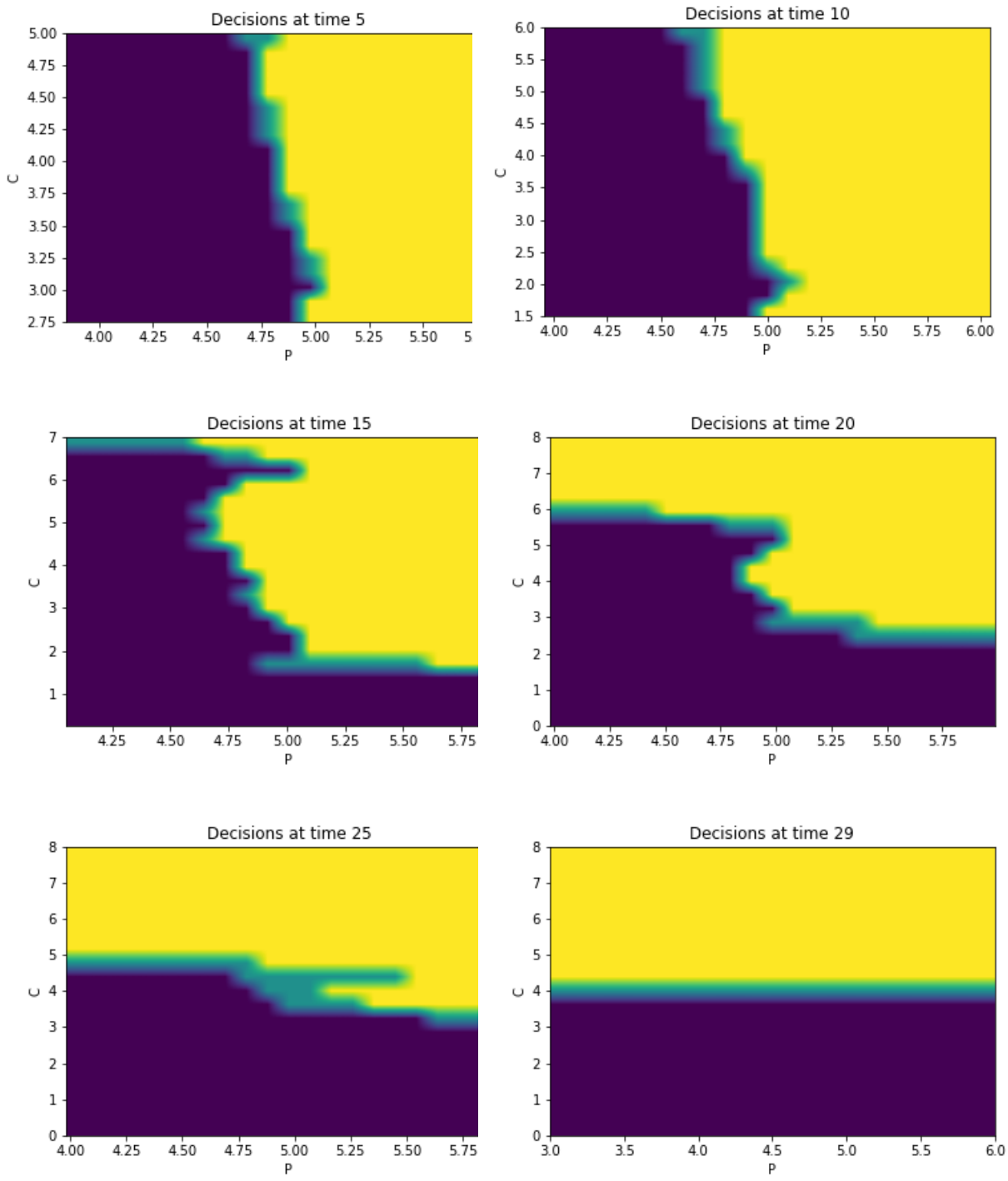


Figure 11: Estimated optimal decisions at times 5, 10, 15, 20, 25, 29 w.r.t. (P,C) for the energy storage valuation problem using **Hybrid-Now**. Injection (a=-1) in purple, store (a=0) in blue and withdraw (a=1) in yellow. Observe the instability in the decisions which come from the fact that we did not pre-train the neural networks (see Section 2.2.3)

As expected, one can observe on each plot that the optimal strategy is to inject gas

when the price is low, to sell gas when the price is high, and to make sure to have a volume of gas greater than $c_0$ in the cave when the terminal time is getting closer to minimize the terminal cost.

Let us now comment on the implementation of the algorithms:

- *Qknn:* Table 4 shows that once again, due to the low-dimensionality of the problem, Qknn provides the best value function estimates. The estimated optimal strategies, shown on Figure 9, are very good estimates of the theoretical ones. The three decision regions on Figure 9 are natural and easy to interpret: basically it is optimal to sell when the price is high, and to buy when it is low. However, a closer look reveals that the waiting region (where it is optimal to do nothing) has an unusual triangular-based shape, due essentially to the discreteness of the space on which the $C$ component of the state space takes its values. We expect this shape to be very hard to reproduce with the DNN-based algorithms proposed in Section 2.

- *ClassifPI:* As shown on Figure 10, the ClassifPI algorithm manages to provide accurate estimates for the optimal controls at time $n = 0, \dots, N-1$. However, the latter is not able to catch the particular triangular-based shape of the waiting region, which explains why Qknn performs better.

- *Hybrid-Now:* As shown on Figure 11, Hybrid-Now only manages to provide relatively poor estimates, compared to ClassifPI and Qknn, of the three different regions at time $n = 0, \dots, N-1$. In particular, the regions suffer from instability.

We end this paragraph by providing some implementation details for the different algorithms we tested.

- *Qknn:* We used the extension of Algorithm 5 introduced in the paragraph "semi-linear interpolation" of the Section 3.2.2. in [Bal+19] and used a projection of each state on its $k$=2-nearest neighbors to get an estimate of the value function which is continuous w.r.t. the control variable at each time $n = 0, \dots, N-1$. The optimal control is computed at each point of the grids using the Brent algorithm, which is a deterministic function optimizer already implemented in Python[e].

- *Implementation details for the neural network-based algorithms:* We use neural networks with two hidden layers, ELU activation functions[f] and 20+20 neurons . The output layer contains 3 neurons with softmax activation function for the ClassifPI algorithm and no activation function for the Hybrid-Now one. We use a training set of size M=60,000 at each time step. Note that given the expression of the terminal cost, the ReLU activation functions (Rectified Linear Units) could have been deemed a better choice to capture the shape of the value functions, but our tests revealed that ELU activation functions

[e]We could have chosen other algorithms to optimize the $Q$-value, but, in our tests, Brent was faster than the other choices that we tried, such as GoldenSearch, and always provided accurate estimates of the optimal controls.

[f]The Exponential Linear Unit (ELU) activation function is defined as $x \mapsto \begin{cases} \exp(x) - 1 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$.

provide better results. At time $n = 0, \ldots, N-1$, we took $\mu_n = \mathcal{U}(C_{min}, C_{max})$ as training measure.

We did not use the pre-train trick discussed in Section 2.2.3, which explains the instability in the decisions that can be observed in Figure 11.

The main conclusion of our numerical comparisons on this energy storage example is that ClassifPI, the DNN-based classification algorithm designed for stochastic control problems with discrete control space, appears to be more accurate than the more general Hybrid-Now. Nevertheless, ClassifPI was not able to capture the unusual triangle-based shape of the optimal control as well as Qknn did.

### 3.5 Microgrid management

Finally, we consider a discrete-time model for power microgrid inspired by the continuous-time models developed in [Hey+18] and [JP15]; see also [Ala+19]. The microgrid consists of a photovoltaic (PV) power plant, a diesel generator and a battery energy storage system (BES), hence using a mix of fuel and renewable energy sources. These generation units are decentralized, i.e., installed at a rather small scale (a few kW power), and physically close to electricity consumers. The PV produces electricity from solar panels with a generation pattern $(P_n)_n$ depending on the weather conditions. The diesel generator has two modes: on and off. Turning it on consumes fuel, and produces an amount of power $\alpha_n$. The BES can store energy for later use but has limited capacity and power. The aim of the microgrid management is to find the optimal planning that meets the power demand, denoted by $(D_n)_n$, while minimizing the operational costs due to the diesel generator. We denote by

$$R_n \;=\; D_n - P_n,$$

the residual demand of power: when $R_n > 0$, one should provide power through diesel or battery, and when $R_n < 0$, one can store the surplus power in the battery.

The optimal control problem over a fixed horizon $N$ is formulated as follows. At any time $n = 0, \ldots, N-1$, the microgrid manager decides the power production of the diesel generator, either by turning it off: $\alpha_n = 0$, or by turning it on, hence generating a power $\alpha_n$ valued in $[A_{min}, A_{max}]$ with $0 < A_{min} < A_{max} < \infty$. There is a fixed cost $\kappa > 0$ associated with switching from the on/off mode to the other one off/on, and we denote by $M_n^\alpha$ the mode valued in $\{0 = \text{off}, 1 = \text{on}\}$ of the generator right before time $n$, i.e., $M_{n+1}^\alpha = 1_{\alpha_n \neq 0}$.

When the diesel generator and renewable provide a surplus of power, the excess can be stored into the battery (up to its limited capacity) for later use, and in case of power insufficiency, the battery is discharged for satisfying the power demand. The input power process $\mathcal{I}^\alpha$ for charging the battery is then given by

$$\mathcal{I}_n^\alpha \;=\; (\alpha_n - R_n)_+ \wedge (C_{max} - C_n^\alpha),$$

where $C_{max}$ is the maximum capacity of the battery with current charge $C^\alpha$, while the output power process $O^\alpha$ for discharging the battery is given by

$$O_n^\alpha \;=\; (R_n - \alpha_n)_+ \wedge C_n^\alpha.$$

Here, we denote $p_+ = \max(p, 0)$. Assuming for simplicity that the battery is fully efficient, the capacity charge $(C_n^\alpha)_n$ of the BES, valued in $[0, C_{max}]$, evolves according to the dynamics

$$C_{n+1}^\alpha \;=\; C_n^\alpha + \mathcal{I}_n^\alpha - O_n^\alpha. \tag{3.14}$$

The imbalance process defined by

$$S_n^\alpha \;=\; R_n - \alpha_n + \mathcal{I}_n^\alpha - O_n^\alpha$$

represents how well we are doing for satisfying electricity supply: the ideal situation occurs when $S_n^\alpha = 0$, i.e., perfect balance between demand and generation. When $S_n^\alpha > 0$, this means that demand is not satisfied, i.e., there is missing power in the microgrid, and when $S_n^\alpha < 0$, there is an excess of electricity. In order to ensure that there is no missing power, we impose the following constraint on the admissible control:

$$S_n^\alpha \;\le\; 0, \quad \text{i.e.} \quad \alpha_n \;\ge\; R_n - C_n^\alpha,$$

but penalize the excess of electricity when $S_n^\alpha < 0$ with a proportional cost $Q^- > 0$. We model the residual demand as a mean-reverting process:

$$R_{n+1} \;=\; \bar{R}(1-\varrho) + \varrho R_n + \varepsilon_{n+1},$$

where $(\varepsilon_n)_n$ are i.i.d., $\bar{R} \in \mathbb{R}$, and $\varrho < 1$. The goal of the microgrid manager is to find the optimal (admissible) decision $\alpha$ that minimizes the functional cost

$$J(\alpha) \;=\; \mathbb{E}\left[\sum_{n=0}^{N-1} \ell(\alpha_n) + \kappa 1_{\{M_n^\alpha \neq M_{n+1}^\alpha\}} + Q^-(S_n^\alpha)_-\right],$$

where $\ell(.)$ is the cost function for fuel consumption: $\ell(0) = 0$, and e.g. $\ell(a) = Ka^\gamma$, with $K > 0$, $\gamma > 0$. This stochastic control problem fits into the 3-dimensional framework of Section 1 (see also Remark 2.4) with control $\alpha$ valued in $\mathbb{A} = \{0\} \times [A_{min}, A_{max}]$, $X^\alpha = (C^\alpha, M^\alpha, R)$, noise $\varepsilon_{n+1}$, starting from an initial value $(C_0^\alpha, M_0^\alpha, R_0) = (c_0, 0, r_0)$ on the state space $[0, C_{max}] \times \{0, 1\} \times \mathbb{R}$, with dynamics function

$$F(x, a, e) \;=\; \begin{pmatrix} F^1(x,a) := c + (a-r)_+ \wedge (C_{max} - c) - (r-a)_+ \wedge c \\ 1_{a \neq 0} \\ \bar{R}(1-\varrho) + \varrho r + e \end{pmatrix},$$

for $x = (c, m, r) \in [0, C_{max}] \times \{0, 1\} \times \mathbb{R}$, $a \in \{0\} \times [A_{min}, A_{max}]$, $e \in \mathbb{R}$, running cost function

$$\begin{aligned} f(x, a) \;&=\; \ell(a) + \kappa 1_{m = 1_{a=0}} + Q^- S(x, a)_-, \\ S(x, a) \;&=\; r - a + (a-r)_+ \wedge (C_{max} - c) - (r-a)_+ \wedge c, \end{aligned}$$

zero terminal cost $g = 0$, and control constraint

$$\begin{aligned} \mathbb{A}_n(x) \;&=\; \Big\{a \in \{0\} \times [A_{min}, A_{max}] : S(x, a) \le 0\Big\} \\ &=\; \Big\{a \in \{0\} \times [A_{min}, A_{max}] : r - c \le a\Big\}. \end{aligned}$$

**Remark 3.2** The state/space constraint is managed in our NN-based algorithm by introducing a penalty function into the running cost (see Remark 2.4): $f(x,a) \leftarrow f(x,a)+L(x,a)$

$$L(x,a) \quad = \quad Q^+\Big(r-c-a\Big)_+$$

with large $Q^+$ taken much larger than $Q^-$. Doing so, the NN-based estimate of the optimal control learns not to take any forbidden decision. □

The control space $\{0\} \cup [A_{\min}, A_{\max}]$ is a mix between a discrete space and a continuous space, which is challenging for algorithms with neural networks. We actually use a mixture of classification and standard DNN for the control: $(p_0(x;\theta), \pi(x;\beta))$ valued in $[0,1] \times [A_{min}, A_{max}]$, where $p_0(x;\theta)$ is the probability of turning off in state $x$, and $\pi(x;\beta)$ is the amount of power when turning on with probability $1-p_0(x;\theta)$. In other words,

$$X_{n+1} \quad = \quad \begin{cases} F(X_n, 0, \varepsilon_{n+1}) & \text{with probability } p_0(X_n;\theta_n) \\ F(X_n, \pi(X_n;\beta_n), \varepsilon_{n+1}) & \text{with probability } 1-p_0(X_n;\theta_n) \end{cases}$$

The pseudo-code of this approach, specifically designed for this problem, is written in Algorithm 6, and we henceforth refer to it as ClassifHybrid. Note in particular that it is an Hybrid version of ClassifPI.

**Algorithm 6:** ClassifHybrid

**Input:** the training distributions $(\mu_n)_{n=0}^{N-1}$;
**Output:**
– estimate of the optimal strategy $(\hat{a}_n)_{n=0}^{N-1}$;
– estimate of the value function $(\hat{V}_n)_{n=0}^{N-1}$;
Set $\hat{V}_N = g$;
**for** $n = N-1, \ldots, 0$ **do**
Compute

$$(\hat{\beta}_n^0, \hat{\beta}_n^1) \in \operatorname*{argmax}_{\beta^0,\beta^1} \mathbb{E}\Bigg[ p_0(X_n;\beta^0)\Big[f(X_n,0) + \hat{V}_{n+1}\Big(f(\hat{X}_{n+1}^0\Big)\Big] + (1-p_0(X_n;\beta^0))\Big[f(X_n,\pi(X_n;\beta^1)) + \hat{V}_{n+1}\Big(\hat{X}_{n+1}^{1,\beta^1}\Big)\Big]\Bigg],$$

where $X_n \rightsquigarrow \mu_n$, $\hat{X}_{n+1}^0 = F(X_n, 0, \varepsilon_{n+1})$, and $\hat{X}_{n+1}^{1,\beta^1} = F(X_n, \pi(X_n;\beta^1), \varepsilon_{n+1})$;
Compute

$$\hat{\theta}_n \in \operatorname*{argmin}_{\theta} \mathbb{E}\Bigg[ p_0\big(X_n;\hat{\beta}_n^0\big)\Big[f(X_n,0) + \hat{V}_{n+1}\Big(f(\hat{X}_{n+1}^0\Big) - \Phi(.;\theta)\Big]^2 + \big(1-p_0\big(X_n;\hat{\beta}_n^0\big)\big)\Big[f(X_n,\pi(X_n;\beta_n^1)) + \hat{V}_{n+1}\Big(\hat{X}_{n+1}^{1,\hat{\beta}_n^1}\Big) - \Phi(.;\theta)\Big]^2\Bigg];$$

Set $\hat{V}_n = \Phi(.;\hat{\theta}_n)$; ▷ $\hat{V}_n$ is the estimate of the value function at time $n$

**Test** We set the parameters to the following values to compare Qknn and ClassifHybrid:

$$\begin{array}{llllllll} N = 30 \text{ or } 200, & \bar{R} = 0.1, & \varrho = 0.9, & \sigma = 0.2, \\ C_{\min} = 0, & C_{\max} = 1 \text{ or } 4, & C_0 = 0, & K = 2, \\ \gamma = 2, & \kappa = 0.2, & Q^- = 10, & R_0 = 0.1, \\ A_{\min} = 0.05, & A_{\max} = 10 & Q^+ = 1000. & \end{array}$$

**Results** Figure 12 shows the Qknn-estimated optimal decisions to take at times $n = 1, 10, 28$ in the cases where $m = M_n = 0$ and $m = M_n = 1$. If the generator is off at time $n$, i.e. $m = 0$, the blue curve separates the region where it is optimal to keep it off and the one where it is optimal to generate power. If the generator is on at time $n$, i.e. $m = 1$, the blue curve separates the region where it is optimal to turn it off and the one where it is optimal to generate power. A colorscale is available on the right to inform how much power it is optimal to generate in both cases. Observe that the optimal decisions are quite intuitive: for example, if the demand is high and the battery is empty, then it is optimal to generate a lot of energy. Moreover, it is optimal to turn the generator off if the demand is negative or if the battery is charged enough to meet the demand.
We plot in Figure 13 the estimated optimal decisions at times $n = 1, 10, 28$, using the Hybrid-Now algorithm, with $N = 30$ time steps. See that the decisions are similar to the ones given using Qknn.
Note that the plots in Figure 12 and 13 look much better than the ones obtained in [Ala+19] in which algorithms based on regress-now or regress-later are used (see in particular Figure 4 in [Ala+19]); hence Qknn and ClassifHybrid seem more stable than the algorithms proposed in [Ala+19].

We report in Table 5 the result for the estimates of the value function with $N$=30 time steps, obtained by running 10 times a forward Monte Carlo with 10,000 simulations using the optimal strategy estimated using Qknn and ClassifHybrid algorithms. Observe that Hybrid-Now performs better than Qknn. However, Qknn run in less than a minute whereas Hybrid-Now needed seven minutes to run.
We also report in Table 6 the value function estimates with $N$=200 time steps, obtained by running 20 times a forward Monte Carlo with 10,000 simulations using the Qknn-estimated optimal strategy.

Table 5: Estimates of the value function at time 0 and state $(C_0 = 0, M_0 = 0, R_0 = 0.1)$, for $N = 30$ and $C_{max} = 1$, using Qknn and ClassifHybrid algorithms. Note that ClassifHybrid achieved better results than Qknn on this problem.

| | Mean | std |
|---|---|---|
| ClassifHybrid | 33.34 | 0.31 |
| Qknn | 35.37 | 0.34 |

Table 6: Qknn-estimates of the value function at time 0 and state $(C_0 = 0, M_0 = 0, R_0 = 0.1)$, for $N = 200$.

| Mean | Standard Deviation |
|---|---|
| 231.8 | 1.2 |

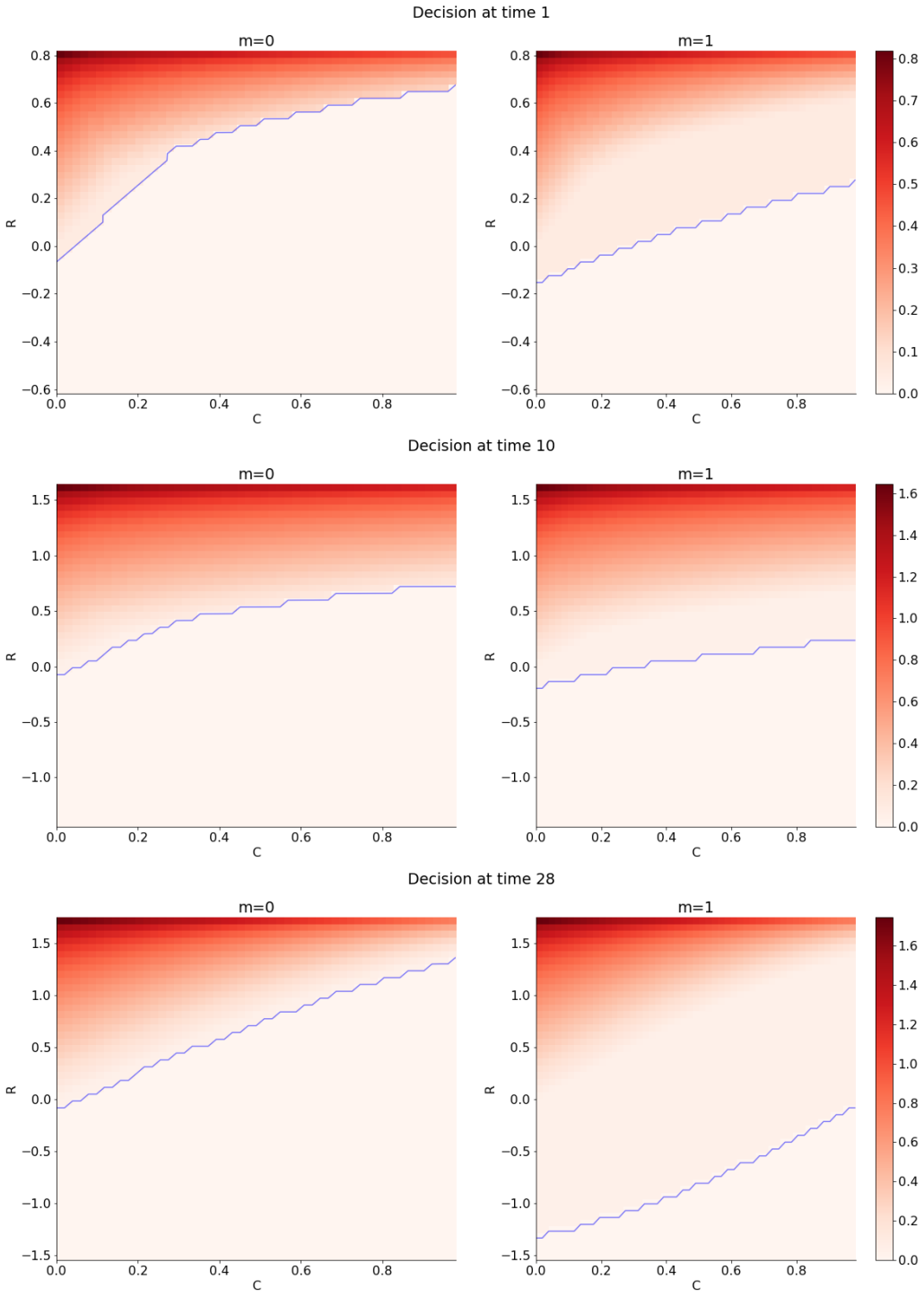


Figure 12: Estimated optimal decisions at time 1, 10 and 28, using Qknn, with $N = 30$ time steps. The region under the blue line is the one where it is optimal to turn the generator off if $m$=1 (i.e. the generator was on at time $n$-1), or keep it off if $m = 0$ (i.e. the generator was off at time $n$-1).

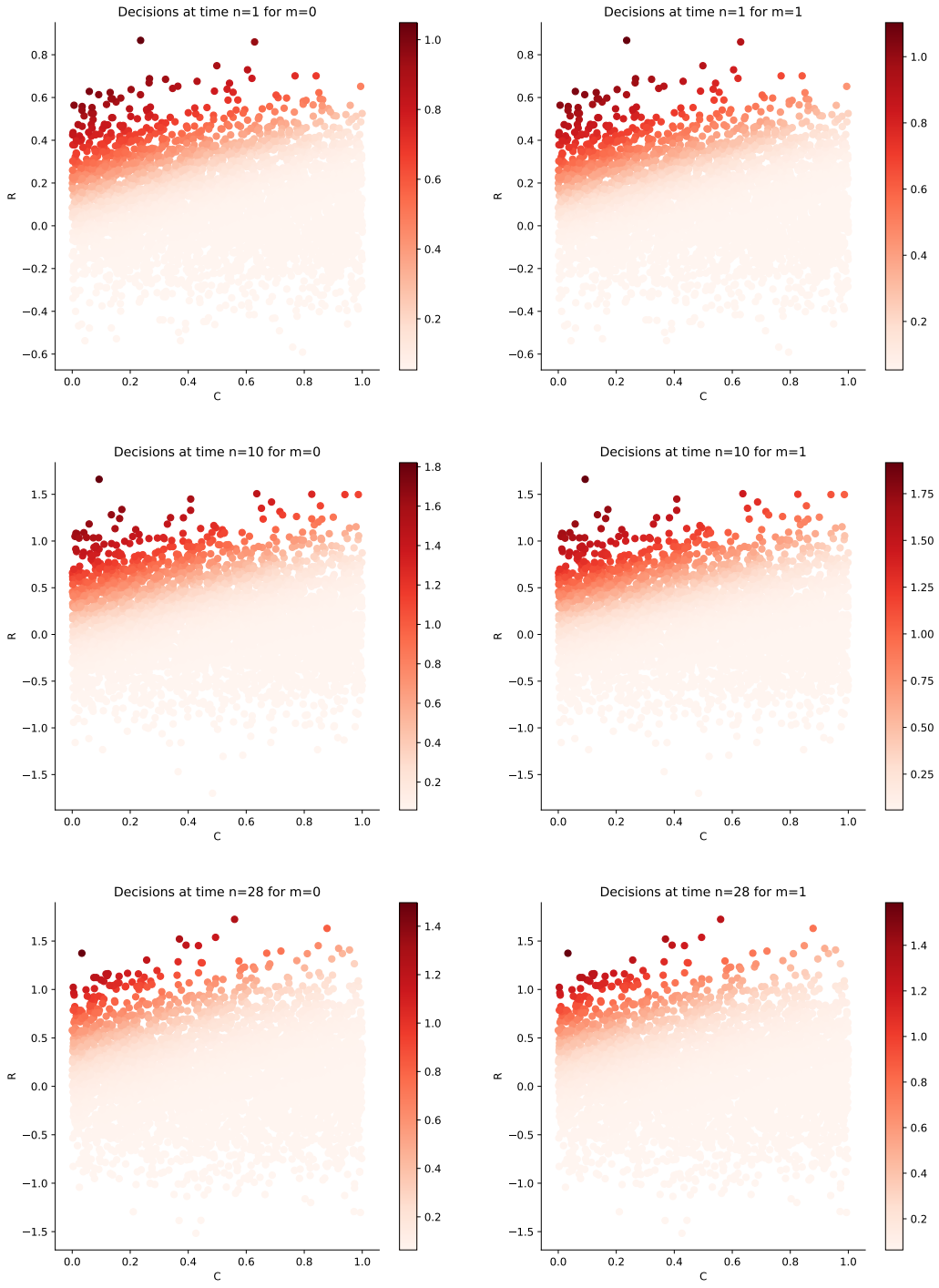


Figure 13: Estimated optimal decisions at time 1, 10 and 28, using ClassifHybrid, with $N = 30$ time steps.

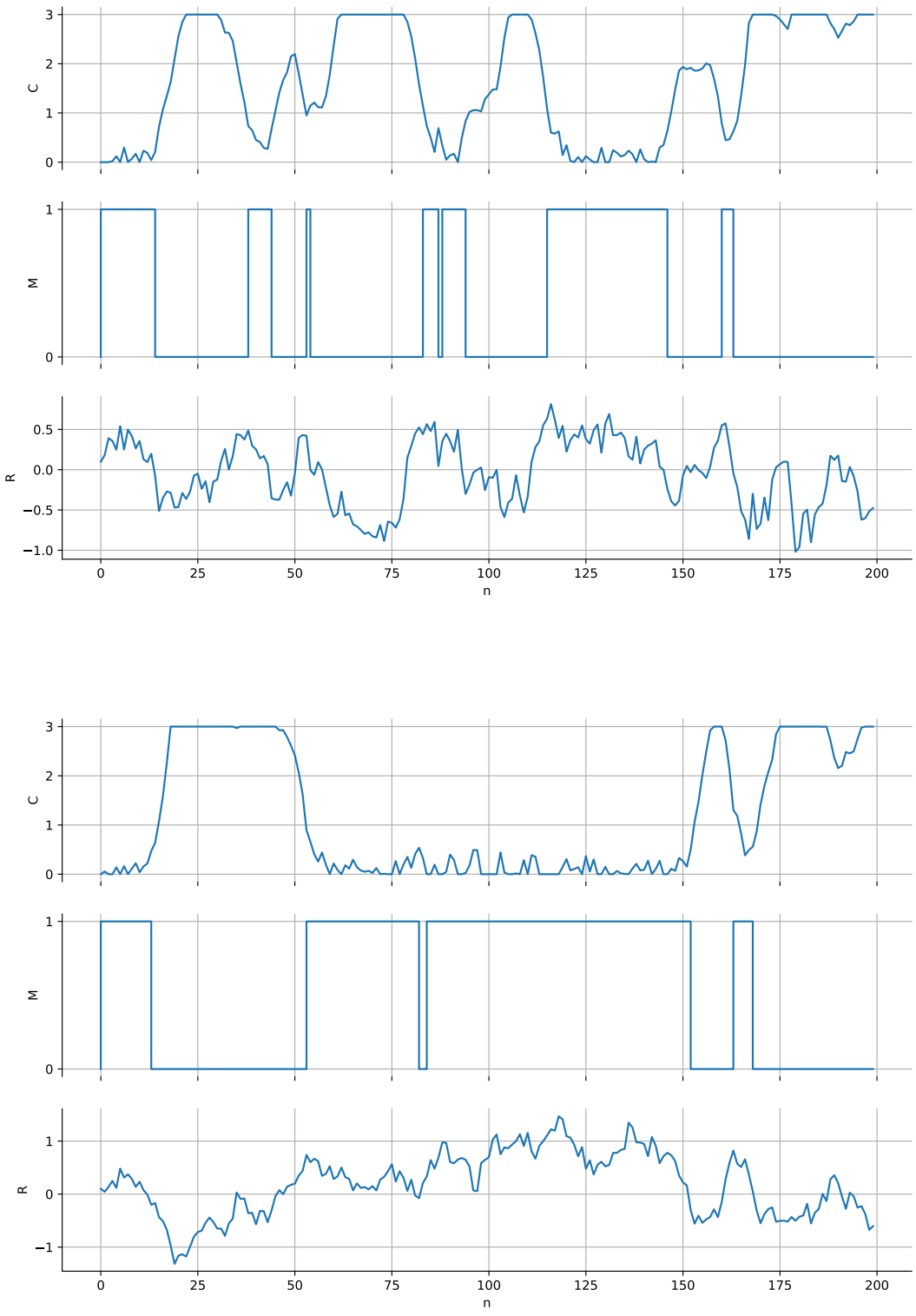


Figure 14: Two simulations of $(C, M, R)$ optimally controlled using Qknn, with $N = 200$ and $C_{\max} = 4$.

Figure 14 shows two simulations of $(C, M, R)$ controlled using the Qknn-estimated optimal strategy, where $N = 200$ has been chosen. Observe in particular the natural behavior of the Qknn-decisions which consists in turning the generator on when the demand cannot be met by the battery, and turn it off when the demand is negative or when the battery is charged enough to meet the demand. Note that the plots are similar to the ones plotted in Figure 9 of [Ala+19].

**Comments on Qknn:** Note that there is no need to use a penalization method with the Qknn-algorithm to constrain the control to stay in $\mathbb{A}_n(x)$, where $x$ is the state at time $n$, since, for all state $x$, we can simply search for the optimal control associated in $\mathbb{A}_n(x)$, using e.g. the Brent algorithm. For $n = 0, \ldots, N-1$, we took the training set as follows: $\Gamma_n := \Gamma_C \times \{0,1\} \times \Gamma_R^n$; where $\Gamma_C := \{C_{min} + \frac{i}{50}(C_{max} - C_{min}), i = 0, \ldots, 50\}$, $\Gamma_R^n := \rho^n R_0 + \sigma \frac{1-\rho^n}{1-\rho} \Gamma_1$ and where $\Gamma_1$ is the optimal grid for the quantization of $\mathcal{N}(0,1)$, available in `http://www.quantize.maths-fi.com`, with 51 points. This choice of training points for the $C$ component corresponds to the exploration procedure discussed in Remark 2.1, whereas we chose the best grid with 51 points for the (uncontrolled) $R$ component.
**Comments on ClassifHybrid:** We took 100 mini-batches of size 300 and took 100 epochs to run the algorithm. We chose the following training distribution at time $n$: $\mu_n = \mathcal{U}(C_{min}, C_{max}) \times \mathcal{U}(\{0,1\}) \times \mathbb{P}_{R_n}$, where $\mathbb{P}_{R_n}$ is the law of the (uncontrolled) residual demand at time $t_n$. Note that such a choice of training distribution means that we want to explore all the available states for the controlled components of the controlled process $(C, M, R)$ in order to learn the optimal strategy globally.

The microgrid management problem is very challenging for our algorithms because the control space $\{0\} \cup [a_{\min}, a_{\max}]$ is a mix of discrete and continuous space, moreover the choice of the optimal control is subject to constraints. We designed ClassifHybrid, an Hybrid version of ClassifPI, to solve this problem. ClassifHybrid provided very good estimates and actually managed to perform better than Qknn.

# 4 Discussion and conclusion

Our proposed algorithms are well-designed and provide accurate estimates of optimal control and value function associated with various high-dimensional control problems. Also, when tested on low-dimensional problems, they performed as well as the Monte Carlo-based or quantization-based methods, which have shown their efficiency in low dimension, see e.g. [Bal+19] and [Ala+19].

The presented algorithms suffer from a rather high time-consuming cost due to the expensive training of $2(N-1)$ neural networks to learn the value functions and optimal controls at times $n = 0, \ldots, N-1$. However, the agent can easily alleviate the computation time. A first trick consists in reducing the number of neural networks by partially or totally ignoring the dynamic programming principle (DPP), as it has been done e.g. in [EHJ17]. The use of one unique Recurrent Neural Networks (RNN) (in the case where the DPP is totally ignored) or a few of them (in the partial-ignored case) can also be considered to learn the optimal controls, either all at the same time (first case), or group by group

in a backward way (second case). We refer to [WNMW19] for algorithms in this spirit. Another trick consists in learning faster the value functions and optimal controls at times $n = 0, \ldots, N-1$ by pre-training the neural networks. The way to proceed in that direction is to initialize at time $n$ the weights and bias of the value function estimator $\hat{V}_n$ to the ones of $\hat{V}_{n+1}$. We then rely on the continuity of the value function w.r.t. the time $n$ to expect that the weights will not change much from time $n$ to $n+1$, hence trainable very quickly by reducing the learning rate of the Adam algorithm for the gradient descent, and using an early-stop procedure as implemented in Keras[g]. Another benefit from the pre-training task is to get the stability of the estimates w.r.t. time, which is also a pleasant feature.

## References


[ACBF02] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. "Finite-time Analysis of the Multiarmed Bandit Problem". In: *Machine Learning* 47.2 (2002), pp. 235–256. ISSN: 1573-0565. DOI: 10.1023/A:1013689704352. URL: https://doi.org/10.1023/A:1013689704352.

[Ala+19] Clemence Alasseur, Alessandro Balata, Sahar Ben Aziza, Aditya Maheshwari, Peter Tankov, and Xavier Warin. "Regression Monte Carlo for Microgrid Management". In: *ESAIM Proceedings and Surveys, CEMRACS 2017* (2019), pp. 46–67.

[Bal+19] Alessandro Balata, Côme Huré, Mathieu Laurière, Huyên Pham, and Isaque Pimentel. "A Class of Finite-Dimensional Numerically Solvable McKean-Vlasov Control Problems". In: *ESAIM Proceedings and Surveys, CEMRACS 2017* 19 (2019), pp. 114–144.

[BKL01] Dimitris Bertsimas, Leonid Kogan, and Andrew W. Lo. "Hedging derivative securities and incomplete markets: an $\varepsilon$-arbitrage approach". In: *Operations Research* 49.3 (2001), pp. 372–397.

[CL10] René Carmona and Mike Ludkovski. "Valuation of energy storage: an optimal switching approach". In: *Quantitative Finance* 26.1 (2010), pp. 262–304.

[CR16] Jean-Francois Chassagneux and Adrien Richou. "Numerical Simulation of Quadratic BSDEs". In: *The Annals of Applied Probabilities* 26.1 (2016), pp. 262–304.

[EHJ17] Weinan E, Jiequn Han, and Arnulf Jentzen. "Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations". In: *Communications in Mathematics and Statistics 5* 5 (2017), pp. 349–380.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT Press, 2016.


[g] See EarlyStopping callback in Keras

[Hey+18] Benjamin Heymann, J. Frédéric Bonnans, Pierre Martinon, Francisco J. Silva, Fernando Lanas, and Guillermo Jiménez-Estévez. "Continuous optimal control approaches to microgrid energy management". In: *Energy Systems* 9.1 (2018), pp. 59–77.

[HL17] Pierre Henry-Labordere. "Deep Primal-Dual Algorithm for BSDEs: Applications of Machine Learning to CVA and IM". In: *SSRN:3071506* (2017).

[Hur+18] Côme Huré, Huyên Pham, Achref Bachouch, and Nicolas Langrené. "Deep neural networks algorithms for Stochastic Control Problems on finite horizon, part I: convergence analysis". In: *arXiv:1812.04300* (2018).

[JP15] Daniel R. Jiang and Warren B. Powell. "An approximate dynamic programming algorithm for monotone value functions". In: *Operations Research* 63.6 (2015), pp. 1489–1511.

[KPX18] Steven Kou, Xianhua Peng, and Xingbo Xu. "A general Monte Carlo algorithm with monotonicity for stochastic control problems". 2018 IMS Annual Meeting on Probability and Statistics. 2018.

[LM19] Michael Ludkovski and Aditya Maheshwari. "Simulation methods for stochastic storage problems: a statistical learning perspective". In: *Energy Systems* (2019). ISSN: 1868-3975. DOI: 10.1007/s12667-018-0318-4. URL: https://doi.org/10.1007/s12667-018-0318-4.

[PPP04] Gilles Pagès, Huyên Pham, and Jacques Printems. "Optimal quantization methods and applications to numerical problems in finance". In: *Handbook of computational and numerical methods in finance* (2004), pp. 253–297.

[Ric10] Adrien Richou. "Etude théorique et numérique des équations différentielles stochastiques rétrogrades". PhD thesis. Université de Rennes 1, 2010.

[Ric11] Adrien Richou. "Numerical Simulation of BSDEs with Drivers of Quadratic Growth". In: *The Annals of Applied Probability* 21.5 (2011), pp. 1933–1964.

[SB98] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning*. The MIT Press, 1998.

[WNMW19] Quentin Chan Wai-Nam, Joseph Mikael, and Xavier Warin. "Machine Learning for semi linear PDEs". In: *Journal of Scientific Computing* 79.3 (2019), pp. 1667–1712.

[YZ99] Jiongmin Yong and Xunyu Zhou. *Stochastic Controls Hamiltonian Systems and HJB Equations*. Springer, 1999.