STRATIFIED REGRESSION MONTE-CARLO SCHEME FOR SEMILINEAR PDES AND BSDES WITH LARGE SCALE PARALLELIZATION ON GPUS

E. GOBET[*], J. G. LÓPEZ-SALAS[†], P. TURKEDJIEV[‡], AND C. VÁZQUEZ[§]

Abstract. In this paper, we design a novel algorithm based on Least-Squares Monte Carlo (LSMC) in order to approximate the solution of discrete time Backward Stochastic Differential Equations (BSDEs). Our algorithm allows massive parallelization of the computations on many core processors such as graphics processing units (GPUs). Our approach consists of a novel method of stratification which appears to be crucial for large scale parallelization. In this way, we minimize the exposure to the memory requirements due to the storage of simulations. Indeed, we note the lower memory overhead of the method compared with previous works.

Key words. Backward stochastic differential equations, dynamic programming equation, empirical regressions, parallel computing, GPUs, CUDA.

AMS subject classifications. 49L20, 62Jxx, 65C30, 93E24, 68W10.

1. Introduction.

The problem. The aim of the algorithm in this paper is to approximate the (Y, Z) components of the solution to the decoupled forward-backward stochastic differential equation (BSDE)

$$Y_t = g(X_T) + \int_t^T f(s, X_s, Y_s, Z_s)\mathrm{d}s - \int_t^T Z_s \mathrm{d}W_s, \tag{1.1}$$

$$X_t = x + \int_0^t b(s, X_s)\mathrm{d}s + \int_0^t \sigma(s, X_s)\mathrm{d}W_s, \tag{1.2}$$

where W is a $q \geq 1$ dimensional Brownian motion. The algorithm will also approximate the solution u to the related semilinear, parabolic partial differential equation (PDE) of the form

$$\partial_t u(t,x) + \mathcal{A}u(t,x) + f(t,x,u(t,x),\nabla_x u\sigma(t,x)) = 0 \quad \text{for } t < T \text{ and } u(T,.) = g(.), \tag{1.3}$$

where $\mathcal{A}$ is the infinitesimal generator of X, through the Feynman-Kac relation $(Y_t, Z_t) = (u(t, X_t), (\nabla_x u\sigma)(t, X_t))$. In recent times, there has been an increasing interest to have algorithms which work efficiently when the dimension d of the space occupied by the process X is large. This interest has been principally driven by the mathematical finance community, where nonlinear valuation rules are becoming increasingly important.

[*]Centre de Mathématiques Appliquées, Ecole Polytechnique and CNRS, route de Saclay, 91128 Palaiseau cedex, France. This research is part of the Chair Financial Risks of the Risk Foundation, the FiME Laboratory and the ANR project CAESARS (ANR-15-CE05-0024). Email: `emmanuel.gobet@polytechnique.edu`

[†]Department of Mathematics, Faculty of Informatics, Universidade da Coruña, Campus de Elviña s/n, 15071 - A Coruña, Spain. This author has been partially funded by Spanish Grant MTM2013-47800-C2-1-P and also by a Spanish FPU grant. Email: `jose.lsalas@udc.es`

[‡]King's College London, Department of Mathematics, Strand, London WC2R 2LS, United Kingdom. This research is supported the Chair Financial Risks of the Risk Foundation and of the FiME Laboratory. Email: `plamen.turkedjiev@kcl.ac.uk`

[§]Department of Mathematics, Faculty of Informatics, Universidade da Coruña, Campus de Elviña s/n, 15071 - A Coruña, Spain. This author has been partially funded by Spanish Grant MTM2013-47800-C2-1-P. Email: `carlosv@udc.es`

In general, currently available algorithms [8, 3, 4, 16, 5, 12, 13, 11] rarely handle the case of dimension greater than 8. The main constraint is not only due to the computational time, but mainly due to memory consumption requirements by the algorithms. For example, the recent work [13] uses a Regression Monte Carlo approach (a.k.a. Least Squares MC), in which the solutions $(u, \nabla_x u\sigma)$ of the semi-linear PDE are approximated on a $\mathcal{K}$-dimensional basis of functions at each point of a time grid of cardinality N. Popular choices of basis functions are global [16] or local polynomials [13]. In both cases, the approximation error behaves in general like $\mathcal{K}^{-\alpha'/d}$ where α' measures the smoothness of the function of interest and d is the dimension (curse of dimensionality): see [7, Theorem 6.2.6] for global polynomials, see [15, Section 11.2] for local polynomials. Later, we use local approximations in order to allow parallel computing. We restrict to affine polynomials for implementation in GPU. The coefficients of the basis functions are computed at every time point t_i with the aid of $\mathcal{M}$ simulations of a discrete time Markov chain (which approximates X) in the interval $[t_i, T]$. The main memory constraints of this scheme are (a) to store the $\mathcal{K} \times N$ coefficients of the basis functions, and (b) to store the $\mathcal{M} \times N$ simulations used to compute the coefficients. To illustrate the problem of high dimension, in order to ensure the convergence the dimension of the basis is typically $\mathcal{K} = const \times N^{\alpha d}$, for some $\alpha > 0$ (which decreases with the regularity of the solution), so $\mathcal{K}$ increases geometrically with d. Moreover, the error analysis of these algorithms demonstrates that the local statistical error is proportional to $N\mathcal{K}/\mathcal{M}$, so that one must choose $\mathcal{M} = const \times \mathcal{K}N^2$ to ensure a convergence $O(N^{-1})$ of the scheme. This implies that the simulations pose by far the most significant constraint on the memory.

Objectives. The purpose of this paper is to drastically rework the algorithm of [13] to first minimize the exposure to the memory due to the storage of simulations. This will allow computation in larger dimension d. Secondly, in this way the algorithm can be implemented in parallel on GPU processors to optimize the computational time.

New Regression Monte Carlo paradigm. We develop a novel algorithm called the Stratified Regression MDP (SRMDP) algorithm; the name is aimed to distinguish from the related LSMDP algorithm [13]. The key technique is to use *stratified simulation* of the paths of X. In order to estimate the solution at t_i, we first define a set of hypercubes $(\mathcal{H}_k \subset \mathbb{R}^d : 1 \leq k \leq K)$. Then, for each hypercube $\mathcal{H}_k$, we simulate M paths of the process X in the interval $[t_i, T]$ starting from i.i.d. random variables valued in $\mathcal{H}_k$; these random variables are distributed according to the conditional logistic distribution, see $(\mathbf{A}_\nu)$ later. By using only the paths starting in $\mathcal{H}_k$, we approximate the solution to the BSDE restricted to $X_{t_i} \in \mathcal{H}_k$ on linear functions spaces $\mathcal{L}_{Y,k}$ and $\mathcal{L}_{Z,k}$ (both of small dimension), see $(\mathbf{A}_{\text{Strat.}})$ later. [1] This allows us to minimize the amount of memory consumed by the simulations, since we only need to generate samples on one hypercube at a time. In Theorem 3.5, we demonstrate that the error of our scheme is proportional to $N \max(\dim(\mathcal{L}_{Y,k}), \dim(\mathcal{L}_{Z,k}))/M$ and, since $\max(\dim(\mathcal{L}_{Y,k}), \dim(\mathcal{L}_{Z,k})) = const$, we require only $M = const \times N^2$ to ensure the convergence $O(N^{-1})$. Therefore, the memory consumption of the algorithm will be dominated by the storage of the coefficients, which equals $const \times N^{\alpha d}$ (the theoretical minimum). Moreover, the computations are performed in parallel across

[1]To distinguish from previous algorithms, we use two notations for the number of simulations in this section: $\mathcal{M}$ and M. $\mathcal{M}$ stands for the overall number of simulations for computing the full approximation in the unstratified algorithms, while M stands for the number of simulations used to evaluate the approximation locally in each stratum (our stratified regression algorithm). Later we will mainly use M.

the hypercubes, which allows for massive parallelization. The speed-up compared to sequential programming increases as the dimension d increases, because of the geometric growth of the number of hypercubes with respect to d. In the subsequent tests (§5), for instance we can solve problems in dimension $d = 11$ within eight seconds using 2000 simulations per hypercube.

This regression Monte Carlo approach is very different from the algorithm proposed in [13]. Although local approximations were already proposed in that work, the paths of the process X were simulated from a fixed point at time 0 rather than directly in the hypercubes. This implies that one must store *all* the simulated paths at any given time, rather than only those for the specific hypercubes. This is because the trajectories are random, and one is not certain which paths will end up in which hypercubes a priori. Therefore, our scheme essentially removes the main constraint on the memory consumption of LSMC algorithms for BSDEs.

The choice of the logistic distribution for the stratification procedure is crucial. Firstly, it is easy to simulate from the conditional distribution. Secondly, it possesses the important USES property (see later $(\mathbf{A}_\nu)$), which enables us to recover equivalent $\mathbf{L}_2$-norms (up to constant) for the marginal of the forward process initialized with the logistic distribution (Proposition 2.1).

Literature review. Parallelization of Monte-Carlo methods for solving non-linear probabilistic equations has been little investigated. Due to the non-linearity, this is a challenging issue. For optimal stopping problems, we can refer to the works [1, 2, 6] with numerical results up to dimension 4. To the best of our knowledge, the only work related to BSDEs in parallel version is [16]. It is based on a Picard iteration for finding the solution, coupled with iterative control variates. The iterative solution is computed through an approximation on sparse polynomial basis. Although the authors report efficient numerical experiments up to dimension 8, this study is not supported by a theoretical error analysis. Due to the stratification, our proposed approach is quite different from [16] and additionally, we provide an error analysis (Theorem 3.5).

Notation.

(i) $|x|$ stands for the Euclidean norm of the vector x.

(ii) $\log(x)$ stands for the natural logarithm of $x \in \mathbb{R}_+$.

(iii) For a multidimensional process $U = (U_i)_{0\le i\le N}$, its l-th component is denoted by $U_l = (U_{l,i})_{0\le i\le N}$.

(iv) For any finite $L > 0$ and $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$, define the truncation function

$$\mathcal{T}_L(x) := (-L \vee x_1 \wedge L, \ldots, -L \vee x_n \wedge L). \tag{1.4}$$

(v) For a probability measure ν on a domain D, and function $h : D \to \mathbb{R}^l$ in $\mathbf{L}_2(D, \nu)$, denote the $\mathbf{L}_2$ norm of h by $|h|_\nu := \sqrt{\int_D |h|^2(x)\nu(\mathrm{d}x)}$.

(vi) For a probability measure ν, disjoint sets $\{\mathcal{H}_1, \ldots, \mathcal{H}_K\}$ in the support of ν, and finite dimensional function spaces $\mathcal{L}\{\mathcal{L}_1, \ldots, \mathcal{L}_K\}$ such that the domain of $\mathcal{L}_k$ is in the respected set $\mathcal{H}_k$

$$\nu(\dim(\mathcal{L})) = \sum_{k=1}^{K} \nu(\mathcal{H}_k)\dim(\mathcal{L}_k).$$

(vii) For function $g : \mathbb{R}_+ \to \mathbb{R}_+$, the order notation $g(x) = O(x)$ means that there exists some universal unspecified constant, *const* > 0, such that $g(x) \leq const \times x$ for all $x \in \mathbb{R}_+$.

2. Mathematical framework and basic properties. We work on a filtered probability space $(\Omega, \mathcal{F}, (\mathcal{F}_t)_{0\leq t\leq T}, \mathbb{P})$ containing a q-dimensional ($q \geq 1$) Brownian motion W. The filtration $(\mathcal{F}_t)_{0\leq t\leq T}$ satisfies the usual hypotheses. The existence of a unique strong solution X to the forward equation (1.2) follows from usual Lipschitz conditions on b and σ, see $(\mathbf{A_X})$. The BSDE (1.1) is approximated using a multistep-forward dynamical programming equation (MDP) studied in [12]. Let $\pi := \{t_i := i\Delta_t : 0 \leq i \leq N\}$ be the uniform time-grid with time step $\Delta_t = T/N$. The solution $(Y_i, Z_i)_{0\leq i\leq N-1}$ of the MDP can be written in the form:

(2.1)
$$\left.\begin{array}{rcl} Y_i &=& \mathbb{E}_i\left(g(X_N) + \sum_{j=i}^{N-1} f_j(X_j, Y_{j+1}, Z_j)\Delta_t\right), \\ \Delta_t Z_i &=& \mathbb{E}_i\left((g(X_N) + \sum_{j=i+1}^{N-1} f_j(X_j, Y_{j+1}, Z_j)\Delta_t)\Delta W_i\right) \end{array}\right\} \quad \text{for } i \in \{0, \ldots, N-1\},$$

where $(X_j)_{i\leq j\leq N}$ is a Markov chain approximating the forward component (1.2) (typically the Euler scheme, see Algorithm 2 below), $\Delta W_i := W_{t_{i+1}} - W_{t_i}$ is the $(i+1)$-th Brownian motion increment, and $\mathbb{E}_i(\cdot) := \mathbb{E}(\cdot \mid \mathcal{F}_{t_i})$ is the conditional expectation. Our working assumptions on the functions g and f are as follows:

$(\mathbf{A_g})$ g is a bounded measurable function from $\mathbb{R}^d$ to $\mathbb{R}$, the upper bound of which is denoted by C_g.

$(\mathbf{A_f})$ for every $i < N$, $f_i(x, y, z)$ is a measurable function $\mathbb{R}^d \times \mathbb{R} \times \mathbb{R}^q$ to $\mathbb{R}$, and there exist two finite constants L_f and C_f such that, for every $i < N$,

$$|f_i(x, y, z) - f_i(x, y', z')| \leq L_f(|y - y'| + |z - z'|),$$
$$\forall (x, y, y', z, z') \in \mathbb{R}^d \times (\mathbb{R})^2 \times (\mathbb{R}^q)^2,$$
$$|f_i(x, 0, 0)| \leq C_f, \quad \forall x \in \mathbb{R}^d.$$

The definition of the Markov chain $(X_j)_j$ is made under the following assumptions.

$(\mathbf{A_X})$ The coefficients functions b and σ satisfy

(i) $b : [0, T] \times \mathbb{R}^d \to \mathbb{R}^d$ and $\sigma : [0, T] \times \mathbb{R}^d \to \mathbb{R}^d \otimes \mathbb{R}^q$ are bounded measurable, uniformly Lipschitz in the space dimensions;

(ii) there exists $\zeta \geq 1$ such that, for all $\xi \in \mathbb{R}^d$, the following inequalities hold: $\zeta^{-1}|\xi|^2 \leq \xi^\top \sigma(t, x)\sigma(t, x)^\top \xi \leq \zeta|\xi|^2$.

Let X_i be a random variable with some distribution η (more details on this to follow). Then X_j for $j > i$ is generated according to one of the two algorithms below:

Algorithm 1 (SDE dynamics). $X_{j+1} = \bar{X}_{t_{j+1}} = X_j + \int_{t_j}^{t_{j+1}} b(s, \bar{X}_s)\mathrm{d}s + \int_{t_j}^{t_{j+1}} \sigma(s, \bar{X}_s)\mathrm{d}W_s$;

Algorithm 2 (Euler dynamics). $X_{j+1} = X_j + b(t_i, X_i)\Delta_t + \sigma(t_i, X_i)\Delta W_i$.

The above ellipticity condition (ii) will be used in the proof of Proposition 2.1.

As in the continuous time framework (1.1), the solution of the MDP (2.1) admits a Markov representation: under $(\mathbf{A_g})$, $(\mathbf{A_f})$ and $(\mathbf{A_X})$ (and using for X either the SDE itself or its Euler scheme), for every i, there exist measurable deterministic functions $y_i : \mathbb{R}^d \to \mathbb{R}$ and $z_i : \mathbb{R}^d \to \mathbb{R}^q$, such that $Y_i = y_i(X_i)$ and $Z_i = z_i(X_i)$, almost surely. In fact, the value functions $y_i(\cdot)$ and $z_i(\cdot)$ are independent of how we initialize the forward component.[2]

[2] Actually under our assumptions, the measurability of y_i and z_i can be easily established by induction on i. More precisely, we can write y_i and z_i as a $(N-i)$-fold integrals in space, using the

For the subsequent stratification algorithm, X_i will be sampled randomly (and independently of the Brownian motion W) according to different squared-integrable distributions η. When $X_i \sim \eta$, we will write $(X_j^{(i,\eta)})_{i\le j\le N}$ the Markov chain given in $(\mathbf{A_X})$, using either the SDE dynamics (better when possible) or the Euler one. One can recover the value functions from the conditional expectations: almost surely,

$$y_i(X_i^{(i,\eta)}) = \mathbb{E}\left(g(X_N^{(i,\eta)}) + \sum_{j=i}^{N-1} f_j(X_j^{(i,\eta)}, y_{j+1}(X_{j+1}^{(i,\eta)}), z_j(X_j^{(i,\eta)}))\Delta_t \mid X_i^{(i,\eta)}\right), \tag{2.2}$$

$$\Delta_t z_i(X_i^{(i,\eta)}) = \mathbb{E}\left((g(X_N^{(i,\eta)}) + \sum_{j=i+1}^{N-1} f_j(X_j^{(i,\eta)}, y_{j+1}(X_{j+1}^{(i,\eta)}), z_j(X_j^{(i,\eta)}))\Delta_t)\Delta W_i \mid X_i^{(i,\eta)}\right);$$

the proof of this is the same as [13, Lemma 4.1].

Approximating the solution to (2.1) is actually achieved by approximating the functions $y_i(\cdot)$ and $z_i(\cdot)$. In this way, we are directly approximating the solution to the semilinear PDE (1.3). Our approach consists in approximating the restrictions of the functions y_i and z_i to subsets of a cubic partition of $\mathbb{R}^d$ using finite dimensional linear function spaces. The basic assumptions for this local approximation approach are given below.

$(\mathbf{A}_{\text{Strat.}})$ There are $K \in \mathbb{N}^*$ disjoint hypercubes $(\mathcal{H}_k : 1 \le k \le K)$, that is

$$\mathcal{H}_k \cap \mathcal{H}_l = \emptyset, \qquad \bigcup_{k=1}^{K} \mathcal{H}_k = \mathbb{R}^d \quad \text{and} \quad \mathcal{H}_k = \prod_{l=1}^{d} [x_{k,l}^-, x_{k,l}^+)$$

for some $-\infty \le x_{k,l}^- < x_{k,l}^+ \le +\infty$. Additionally, there are linear function spaces $\mathcal{L}_{Y,k}$ and $\mathcal{L}_{Z,k}$, valued in $\mathbb{R}$ and $\mathbb{R}^q$ respectively, which are subspaces of $\mathbf{L}_2(\mathcal{H}_k, \nu_k)$ w.r.t. a probability measure ν_k on $\mathcal{H}_k$ defined in $(\mathbf{A}_\nu)$ below.

Common examples of hypercubes are:

(i) Hypercubes of equal size: $x_{k,l}^+ - x_{k,l}^- = \textit{const} > 0$ for all k and l, except for exterior strata that must be infinite.

(ii) Hypercubes of equal probability: $\nu(\mathcal{H}_k) = 1/K$ for some probability ν to be defined later in $(\mathbf{A}_\nu)$.

Common examples of local approximations spaces $\mathcal{L}_{Y,k}$ and $\mathcal{L}_{Z,k}$ are:

(i) Piece-wise constant approximation (**LP0**): $\mathcal{L}_{Y,k} := \text{span}\{\mathbf{1}_{\mathcal{H}_k}\}$, and $\mathcal{L}_{Z,k} := (\mathcal{L}_{Y,k})^q$; $\dim(\mathcal{L}_Y) = 1$ and $\dim(\mathcal{L}_{Z,k}) = q$.

(ii) Affine approximations (**LP1**): $\mathcal{L}_{Y,k} := \text{span}\{\mathbf{1}_{\mathcal{H}_k}, x_1\mathbf{1}_{\mathcal{H}_k}, \dots, x_d\mathbf{1}_{\mathcal{H}_k}\}$, and $\mathcal{L}_{Z,k} := (\mathcal{L}_{Y,k})^q$; $\dim(\mathcal{L}_Y) = d+1$ and $\dim(\mathcal{L}_{Z,k}) = q(d+1)$.

The key idea in this paper is to select a distribution ν, the restriction of which to the hypercubes $\mathcal{H}_k$, ν_k, can be explicitly computed. Then, we can easily simulate i.i.d. copies of $X_i^{(i,\nu_k)}$ directly in $\mathcal{H}_k$ and use the resulting paths of the Markov chain to estimate $y_k(\cdot)|_{\mathcal{H}_k}$. This sampling method is traditionally known as *stratification*, and for this reason we will call the hypercubes in $(\mathbf{A}_{\text{Strat.}})$ the *strata*. For the stratification, the components $X_i^{(i,\nu_k)}$ are sampled as i.i.d. conditional logistic random variables, which is precisely stated in the following assumption.

C^2 transition density of X given in Algorithms 1 or 2. From this, we observe that z_i is a C^2 function of x_i; regarding y_i all the contributions in the sum for $j > i$ are also smooth, and only the $j = i$ term may be non-smooth (because of $x_i \mapsto f_i(x_i, .)$ is only assumed measurable). From this, we easily see that the initialization x_i of X at time i can be made arbitrary, provided that this is independent of W.

$(\mathbf{A}_\nu)$ Let $\mu > 0$. The distribution of $X_i^{(i,\nu_k)}$ is given by $\mathbb{P} \circ (X_i^{(i,\nu_k)})^{-1}(\mathrm{d}x) = \nu_k(\mathrm{d}x)$, where

$$\nu_k(\mathrm{d}x) = \frac{\mathbf{1}_{\mathcal{H}_k}(x)\nu(\mathrm{d}x)}{\nu(\mathcal{H}_k)},$$

and

$$\nu(\mathrm{d}x) = p^{(\mu)}_{\text{logis.}}(x)\mathrm{d}x, \quad p^{(\mu)}_{\text{logis.}}(x) := \prod_{l=1}^{d} \frac{\mu e^{-\mu x_l}}{(1+e^{-\mu x_l})^2}, \quad x = (x_1, \dots, x_d) \in \mathbb{R}^d.$$

Remark 2.1. *An important relation of ν and ν_k is that one has the $\mathbf{L}_2$-norm identity $|\cdot|^2_\nu = \sum_{k=1}^K \nu(\mathcal{H}_k)\, |\cdot|^2_{\nu_k}$.*

In order to generate the random variable $X_i^{(i,\nu_k)}$, we make use of the *inverse* conditional distribution function of ν_k and the simulation of uniform random variables, as shown in the following algorithm:

Algorithm 3. *Draw d independent random variables $(U_1, \dots, U_d)$ which are uniformly distributed on $[0,1]$, and compute*

$$X_i^{(i,\nu_k)} := \left(F^{-1}_{\nu,[x^-_{k,1}, x^+_{k,1})}(U_1), \dots, F^{-1}_{\nu,[x^-_{k,d}, x^+_{k,d})}(U_d)\right) \overset{d}{\sim} \nu_k,$$

where we use the functions $F_\nu(x) := \int_{-\infty}^x \nu(\mathrm{d}x') = 1/\left(1+\exp(-\mu x)\right)$ and

$$F^{-1}_{\nu,[x^-,x^+)}(U) = -\frac{1}{\mu}\log\left(\frac{1}{F_\nu(x^-) + U(F_\nu(x^+) - F_\nu(x^-))} - 1\right).$$

A further reason for the choice of the logistic distribution is that it induces the following stability property on the $\mathbf{L}_2$ norms of the Markov chain $(X_j^{(i,\nu)})_{i\le j\le N}$; this property will be crucial for the error analysis of the stratified regression scheme in §3.2. The proof is postponed to Appendix A.1.

Proposition 2.1. *Suppose that ν is the logistic distribution defined in $(\mathbf{A}_\nu)$. There is a constant $c_{(\mathbf{A}_\nu)} \in [1, +\infty)$ such that, for any function $h : \mathbb{R}^d \mapsto \mathbb{R}$ or $\mathbb{R}^q$ in $\mathbf{L}_2(\nu)$, for any $0 \le i \le N$, and for any $i \le j \le N-1$, we have*

$$\frac{1}{c_{(\mathbf{A}_\nu)}}\mathbb{E}[|h(X_j^{(i,\nu)})|^2] \le |h|^2_\nu \le c_{(\mathbf{A}_\nu)}\mathbb{E}[|h(X_j^{(i,\nu)})|^2].$$

To conclude this section, we recall standard uniform absolute bounds for the functions $y_i(\cdot)$ and $z_i(\cdot)$.

Proposition 2.2 (*a.s.* upper bounds, [13, Proposition 3.3]). *For N large enough such that $\frac{T}{N}L_f^2 \le \frac{1}{12q}$, we have for any $x \in \mathbb{R}^d$ and any $0 \le i \le N-1$,*

$$|y_i(x)| \le C_y := e^{\frac{T}{4}+6q(1\vee L_f^2)(T\vee 1)}\left(C_g + \frac{T}{2\sqrt{q}}C_f\right), \qquad |z_{l,i}(x)| \le C_z := \frac{C_y}{\sqrt{\Delta_t}}. \tag{2.3}$$

3. Stratified algorithm and convergence results.

3.1. Algorithm. In this section, we define the SRMDP algorithm mathematically, and then expose in §4 how to efficiently perform it using GPUs. Our algorithm involves solving a sequence of Ordinary linear Least Squares regression (**OLS**) problems. For a precise mathematical statement, we recall the seemingly abstract but very convenient definition from [13]; explicit algorithms for the computation of **OLS** solutions are exposed in §4.1.

Definition 3.1 (Ordinary linear least-squares regression). *For $l, l' \geq 1$ and for probability spaces $(\tilde{\Omega}, \tilde{\mathcal{F}}, \tilde{\mathbb{P}})$ and $(\mathbb{R}^l, \mathcal{B}(\mathbb{R}^l), \eta)$, let S be a $\tilde{\mathcal{F}} \otimes \mathcal{B}(\mathbb{R}^l)$-measurable $\mathbb{R}^{l'}$-valued function such that $S(\omega, \cdot) \in \mathbf{L}_2(\mathcal{B}(\mathbb{R}^l), \eta)$ for $\tilde{\mathbb{P}}$-a.e. $\omega \in \tilde{\Omega}$, and $\mathcal{L}$ a linear vector subspace of $\mathbf{L}_2(\mathcal{B}(\mathbb{R}^l), \eta)$ spanned by deterministic $\mathbb{R}^{l'}$-valued functions $\{p_k(.), k \geq 1\}$. The least squares approximation of S in the space $\mathcal{L}$ with respect to η is the ($\tilde{\mathbb{P}} \times \eta$-a.e.) unique, $\tilde{\mathcal{F}} \otimes \mathcal{B}(\mathbb{R}^l)$-measurable function $S^\star$ given by*

$$S^\star(\omega, \cdot) = \arg \inf_{\phi \in \mathcal{L}} \int |\phi(x) - S(\omega, x)|^2 \eta(\mathrm{d}x).$$

We say that $S^\star$ solves $\mathbf{OLS}(S, \mathcal{L}, \eta)$.

On the other hand, suppose that $\eta_M = \frac{1}{M} \sum_{m=1}^M \delta_{\mathcal{X}^{(m)}}$ is a discrete probability measure on $(\mathbb{R}^l, \mathcal{B}(\mathbb{R}^l))$, where δ_x is the Dirac measure on x and $\mathcal{X}^{(1)}, \dots, \mathcal{X}^{(M)} : \tilde{\Omega} \to \mathbb{R}^l$ are i.i.d. random variables. For an $\tilde{\mathcal{F}} \otimes \mathcal{B}(\mathbb{R}^l)$-measurable $\mathbb{R}^{l'}$-valued function S such that $|S(\omega, \mathcal{X}^{(m)}(\omega))| < \infty$ for any m and $\tilde{\mathbb{P}}$-a.e. $\omega \in \tilde{\Omega}$, the least squares approximation of S in the space $\mathcal{L}$ with respect to η_M is the ($\tilde{\mathbb{P}}$-a.e.) unique, $\tilde{\mathcal{F}} \otimes \mathcal{B}(\mathbb{R}^l)$-measurable function $S^\star$ given by

$$S^\star(\omega, \cdot) = \arg \inf_{\phi \in \mathcal{L}} \frac{1}{M} \sum_{m=1}^{M} |\phi(\mathcal{X}^{(m)}(\omega)) - S(\omega, \mathcal{X}^{(m)}(\omega))|^2.$$

We say that $S^\star$ solves $\mathbf{OLS}(S, \mathcal{L}, \eta_M)$.

Definition 3.2 (Simulations and empirical measures). *Recall the Markov chain $(X_j^{(i,\nu_k)})_{i \leq j \leq N}$ initialized as in $(\mathbf{A}_\nu)$. For any $i \in \{0, \dots, N-1\}$ and $k \in \{1, \dots, K\}$, define $M \geq \dim(\mathcal{L}_{Y,k}) \vee \dim(\mathcal{L}_{Z,k})$ independent copies of $(\Delta W_i, (X_j^{i,\nu_k})_{i \leq j \leq N})$ that we denote by*

$$\mathcal{C}_{i,k} := \left\{ (\Delta W_i^{(i,k,m)}, (X_j^{(i,k,m)})_{i \leq j \leq N}) \ : \ m = 1, \dots, M \right\}.$$

The random variables $\mathcal{C}_{i,k}$ form a cloud of simulations *used for the regression at time i and in the stratum k. Furthermore, we assume that the clouds of simulations $(\mathcal{C}_{i,k} : 0 \leq i \leq N-1, 1 \leq k \leq K)$ are independently generated. All these random variables are defined on a probability space $(\Omega^{(M)}, \mathcal{F}^{(M)}, \mathbb{P}^{(M)})$. Denote by $\nu_{i,k,M}$ the empirical probability measure of the $\mathcal{C}_{i,k}$-simulations, i.e.*

$$\nu_{i,k,M} = \frac{1}{M} \sum_{m=1}^{M} \delta_{(\Delta W_i^{(i,k,m)}, X_i^{(i,k,m)}, \dots, X_N^{(i,k,m)})}.$$

Denoting by $(\Omega, \mathcal{F}, \mathbb{P})$ the probability space supporting $(\Delta W_i, X^{i,\nu_k} : 0 \leq i \leq N-1, 1 \leq k \leq K)$, which serves as a generic element for the clouds of simulations $\mathcal{C}_{i,k}$, the full probability space used to analyze our algorithm is the product space $(\bar{\Omega}, \bar{\mathcal{F}}, \bar{\mathbb{P}}) = (\Omega, \mathcal{F}, \mathbb{P}) \otimes (\Omega^{(M)}, \mathcal{F}^{(M)}, \mathbb{P}^{(M)})$. By a slight abuse of notation, we write $\mathbb{P}$ (resp. $\mathbb{E}$) to mean $\bar{\mathbb{P}}$ (resp. $\bar{\mathbb{E}}$) from now on.

We now come to the definition of the stratified LSMDP algorithm, which computes random approximations $y_i^{(M)}(.)$ and $z_i^{(M)}(.)$

Algorithm 4 (SRMDP). *Recall the linear spaces $\mathcal{L}_{Y,k}$ and $\mathcal{L}_{Z,k}$ from ($\mathbf{A}_{\text{Strat.}}$), the bounds (2.3) and the truncation function $\mathcal{T}_L$ (see (1.4)).*

Initialization. *Set $y_N^{(M)}(\cdot) := g(\cdot)$.*

Backward iteration for $i = N-1$ to $i = 0$. *For any stratum index $k \in \{1, \dots, K\}$, generate the empirical measure $\nu_{i,k,M}$ as in Definition 3.2, and define*

(3.1)
$$\begin{cases} \psi_{Z,i,k}^{(M)}(\cdot) \quad \textit{solution of} \quad \mathbf{OLS}(S_{Z,i}^{(M)}(w, \underline{\mathbf{x}}_i)\ ,\ \mathcal{L}_{Z,k}\ ,\ \nu_{i,k,M}) \\ \qquad \textit{for} \quad S_{Z,i}^{(M)}(w, \underline{\mathbf{x}}_i) := \dfrac{1}{\Delta_t} S_{Y,i+1}^{(M)}(\underline{\mathbf{x}}_i)\ w, \\ z_i^{(M)}(\cdot)|_{\mathcal{H}_k} := \mathcal{T}_{C_z}\big(\psi_{Z,i,k}^{(M)}(\cdot)\big) \quad (\textit{truncation}), \\ \psi_{Y,i,k}^{(M)}(\cdot) \quad \textit{solution of} \quad \mathbf{OLS}(S_{Y,i}^{(M)}(\underline{\mathbf{x}}_i)\ ,\ \mathcal{L}_{Y,k}\ ,\ \nu_{i,k,M}) \\ \qquad \textit{for} \quad S_{Y,i}^{(M)}(\underline{\mathbf{x}}_i) := g(x_N) + \displaystyle\sum_{j=i}^{N-1} f_j\big(x_j, y_{j+1}^{(M)}(x_{j+1}), z_j^{(M)}(x_j)\big)\Delta_t, \\ y_i^{(M)}(\cdot)|_{\mathcal{H}_k} := \mathcal{T}_{C_y}\big(\psi_{Y,i,k}^{(M)}(\cdot)\big) \quad (\textit{truncation}), \end{cases}$$

where $w \in \mathbb{R}^q$ and $\underline{\mathbf{x}}_i = (x_i, \dots, x_N) \in (\mathbb{R}^d)^{N-i+1}$.

An important difference between SRMDP and established Monte Carlo algorithms [9, 18, 12, 13] is that the number of simulations falling in each hypercube is no more random but fixed and equal to M. Observe first that this is likely to improve the numerical stability of the regression algorithm: there is no risk that too few simulations will land in the hypercube, leading to under-fitting. Later, in §4, we shall explain how to implement Algorithm 4 on a GPU device. The key point is that the calculations at every time point are fully independent between the different hypercubes, so that we can perform them in parallel across the hypercubes. The choice of M independent on k is made in order to maintain a computational effort equal on each of the strata. In this way, the gain in parallelization is likely to be the largest. However, the subsequent mathematical analysis can be easily adapted to make the number of simulations vary with k whenever necessary.

An easy but important consequence of Algorithm 4 and of the bounds of Proposition 2.2 is the following absolute bound; the proof is analogous to that of [13, Lemma 4.7].

Lemma 3.3. *With the above notation, we have*

$$\sup_{0 \le i \le N} \sup_{\underline{\mathbf{x}_i} \in (\mathbb{R}^d)^{N-i+1}} |S_{Y,i}^{(M)}(\underline{\mathbf{x}_i})| \le C_{3.3} := C_g + T\left(L_f C_y \left[1 + \frac{\sqrt{q}}{\sqrt{\Delta_t}}\right] + C_f\right).$$

3.2. Error analysis. The analysis will be performed according to several $\mathbf{L}_2$-norms, either w.r.t. the probability measure ν, or the empirical norm related to the cloud simulations. They are defined as follows:

$$\mathcal{E}(Y, M, i) := \sum_{k=1}^{K} \nu(\mathcal{H}_k)\mathbb{E}\left(\left|y_i^{(M)}(\cdot) - y_i(\cdot)\right|_{i,k,M}^2\right),$$

$$\bar{\mathcal{E}}(Y, M, i) := \sum_{k=1}^{K} \nu(\mathcal{H}_k)\mathbb{E}\left(\left|y_i^{(M)}(\cdot) - y_i(\cdot)\right|_{\nu_k}^2\right) = \mathbb{E}\left(\left|y_i^{(M)}(\cdot) - y_i(\cdot)\right|_{\nu}^2\right),$$

$$\begin{aligned}
\mathcal{E}(Z, M, i) &:= \sum_{k=1}^{K} \nu(\mathcal{H}_k) \mathbb{E}\left(\left| z_i^{(M)}(\cdot) - z_i(\cdot) \right|_{i,k,M}^2 \right), \\
\bar{\mathcal{E}}(Z, M, i) &:= \sum_{k=1}^{K} \nu(\mathcal{H}_k) \mathbb{E}\left(\left| z_i^{(M)}(\cdot) - z_i(\cdot) \right|_{\nu_k}^2 \right) = \mathbb{E}\left(\left| z_i^{(M)}(\cdot) - z_i(\cdot) \right|_{\nu}^2 \right),
\end{aligned}$$

where

$$|h|_{i,k,M} := \left(\int |h|^2(\omega, \underline{\mathbf{x}}_i) \nu_{i,k,M}(\mathrm{d}\omega, \mathrm{d}\underline{\mathbf{x}}_i) \right)^{1/2}.$$

In fact, the norms $\mathcal{E}(., M, i)$ and $\bar{\mathcal{E}}(., M, i)$ are related through model-free concentration-of-measures inequalities. This relation is summarized in the proposition below.

Proposition 3.4. *For each $i \in \{0, \dots, N-1\}$, we have*

$$\begin{aligned}
\bar{\mathcal{E}}(Y, M, i) &\le 2\mathcal{E}(Y, M, i) + \frac{2028 C_y^2 \log(3M)}{M} \left(\nu(\dim(\mathcal{L}_{Y,.})) + 1 \right), \\
\bar{\mathcal{E}}(Z, M, i) &\le 2\mathcal{E}(Z, M, i) + \frac{2028 q C_y^2 \log(3M)}{\Delta_t M} \left(\nu(\dim(\mathcal{L}_{Z,.})) + 1 \right).
\end{aligned}$$

Proof. It is clearly sufficient to show that

$$\begin{aligned}
\mathbb{E}\left(\left| y_i^{(M)}(\cdot) - y_i(\cdot) \right|_{\nu_k}^2 \right) &\le 2\mathbb{E}\left(\left| y_i^{(M)}(\cdot) - y_i(\cdot) \right|_{i,k,M}^2 \right) \\
&\quad + \frac{2028 C_y^2 \log(3M)}{M} \left(\dim(\mathcal{L}_{Y,.}) + 1 \right), \\
\mathbb{E}\left(\left| z_i^{(M)}(\cdot) - z_i(\cdot) \right|_{\nu_k}^2 \right) &\le 2\mathbb{E}\left(\left| z_i^{(M)}(\cdot) - z_i(\cdot) \right|_{i,k,M}^2 \right) \\
&\quad + \frac{2028 q C_y^2 \log(3M)}{\Delta_t M} \left(\dim(\mathcal{L}_{Z,.}) + 1 \right),
\end{aligned}$$

which follows exactly as in the proof of [13, Proposition 4.10]. □

From the previous proposition, the controls on $\bar{\mathcal{E}}(Y, M, i)$ and $\bar{\mathcal{E}}(Z, M, i)$ stem from those on $\mathcal{E}(Y, M, i)$ and $\mathcal{E}(Z, M, i)$, which are handled in Theorem 3.5 below. In order to study the impact of basis selection, we define the squared *quadratic approximation errors* associated to the basis in hypercube $\mathcal{H}_k$ by

$$T_{i,k}^Y := \inf_{\phi \in \mathcal{L}_{Y,k}} |\phi - y_i|_{\nu_k}^2, \quad T_{i,k}^Z := \inf_{\phi \in \mathcal{L}_{Z,k}} |\phi - z_i|_{\nu_k}^2.$$

These terms are the minimal error that can possibly be achieved by the basis $\mathcal{L}_{Y,k}$ (resp. $\mathcal{L}_{Z,k}$) in order to approximate the restriction $y_i(\cdot)|_{\mathcal{H}_k}$ (resp. $z_i(\cdot)|_{\mathcal{H}_k}$) in the $\mathbf{L}_2$ norm. Consequently, the global squared quadratic approximation error is given by

$$T_i^Y := \sum_{k=1}^{K} \nu(\mathcal{H}_k) T_{i,k}^Y = \inf_{\phi \text{ s.t. } \phi|_{\mathcal{H}_k} \in \mathcal{L}_{Y,k}} |\phi - y_i|_{\nu}^2, \tag{3.2}$$

$$T_i^Z := \sum_{k=1}^{K} \nu(\mathcal{H}_k) T_{i,k}^Z = \inf_{\phi \text{ s.t. } \phi|_{\mathcal{H}_k} \in \mathcal{L}_{Z,k}} |\phi - z_i|_{\nu}^2. \tag{3.3}$$

As we shall see in Theorem 3.5 below, the terms T_i^Y and T_i^Z are closely associated to the limit of the expected quadratic error of the numerical scheme in the asymptotic $M \to \infty$; for this reason, these terms are usually called *bias* terms.

Now, we are in the position to state our main result giving non-asymptotic error estimates.

Theorem 3.5 (Error for the Stratified LSMDP scheme). *Recall the constants C_y from Proposition 2.2, $C_{3.3}$ from Lemma 3.3, and $c_{(\mathbf{A}_\nu)}$ from Proposition 2.1. For each $i \in \{0, \ldots, N-1\}$, define*

$$\begin{aligned}\mathcal{E}(i) := 2\sum_{j=i}^{N-1} \Delta_t \Big(T_j^Y &+ 3C_{3.3}^2 \frac{\nu(\dim(\mathcal{L}_{Y,.}))}{M} + 12168 L_f^2 \Delta_t \frac{(\nu(\dim(\mathcal{L}_{Z,.})) + 1) q C_y^2 \log(3M)}{M} \\ &+ 3T_j^Z + 6qC_{3.3}^2 \frac{\nu(\dim(\mathcal{L}_{Z,.}))}{\Delta_t M} \Big) \\ &+ (T - t_i) \frac{1014 C_y^2 \log(3M)}{M} \left((\nu(\dim(\mathcal{L}_{Y,.})) + 1) + \frac{q}{\Delta_t} (\nu(\dim(\mathcal{L}_{Z,.})) + 1) \right).\end{aligned}$$

For Δ_t small enough such that $L_f \Delta_t \leq \sqrt{\frac{2}{15}}$ and $\Delta_t L_f^2 \leq \frac{1}{288 c_{(\mathbf{A}_\nu)}^2 C_{A.1}(1+T)}$, we have, for all $0 \leq i \leq N-1$,

$$\begin{aligned}\mathcal{E}(Y, M, i) \leq T_i^Y &+ 3C_{3.3}^2 \frac{\nu(\dim(\mathcal{L}_{Y,.}))}{M} + 12168 L_f^2 \Delta_t \frac{(\nu(\dim(\mathcal{L}_{Z,.})) + 1) q C_y^2 \log(3M)}{M} \\ &+ (1 + 15 L_f^2 \Delta_t) C_{3.5} \mathcal{E}(i), \qquad (3.4)\end{aligned}$$

$$\sum_{j=i}^{N-1} \Delta_t \mathcal{E}(Z, M, j) \leq C_{3.5} \mathcal{E}(i), \qquad (3.5)$$

where $C_{3.5} := \exp(288 c_{(\mathbf{A}_\nu)}^2 C_{A.1}(1+T) L_f^2 T)$.

3.3. Proof of Theorem 3.5. We start by obtaining estimates on the *local* empirical quadratic errors terms

$$\mathbb{E}\left(\left| y_i^{(M)}(\cdot) - y_i(\cdot) \right|_{i,k,M}^2 \right), \quad \mathbb{E}\left(\left| z_i^{(M)}(\cdot) - z_i(\cdot) \right|_{i,k,M}^2 \right),$$

on each of the hypercubes $\mathcal{H}_k$ $(k = 1, \ldots, K)$. We first reformulate (2.2) with $\eta = \nu_k$ in terms of the Definition 3.1 of OLS. For each $i \in \{0, \ldots, N-1\}$ and $k \in \{1, \ldots, K\}$, let $\nu_{i,k} := \mathbb{P} \circ (\Delta W_i, X_i^{i,\nu_k}, \ldots, X_N^{i,\nu_k})^{-1}$, so that we have

$$\begin{cases} y_i(\cdot)|_{\mathcal{H}_k} \text{ solution of } \mathbf{OLS}(\ S_{Y,i}(\underline{\mathbf{x}}_i)\ ,\ \mathcal{L}_k^{(1)}\ ,\ \nu_{i,k}\ ) \\ \qquad \text{where } S_{Y,i}(\underline{\mathbf{x}}_i) := g(x_N) + \displaystyle\sum_{j=i}^{N-1} f_j\big(x_j, y_{j+1}(x_{j+1}), z_j(x_j)\big)\Delta_t, \\ z_i(\cdot)|_{\mathcal{H}_k} \text{ solution of } \mathbf{OLS}(\ S_{Z,i}(w, \underline{\mathbf{x}}_i)\ ,\ \mathcal{L}_k^{(q)}\ ,\ \nu_{i,k}\ ) \\ \qquad \text{where } S_{Z,i}(w, \underline{\mathbf{x}}_i) := \dfrac{1}{\Delta_t} S_{Y,i+1}(\underline{\mathbf{x}}_i)\ w, \end{cases}$$

where $w \in \mathbb{R}^q$, $\underline{\mathbf{x}}_i := (x_i, \ldots, x_N) \in (\mathbb{R}^d)^{N-i+1}$ and where $\mathcal{L}_k^{(l')}$ is any dense separable subspace in the $\mathbb{R}^{l'}$-valued functions belonging to $\mathbf{L}_2(\mathcal{B}(\mathcal{H}_k), \nu_k)$. The above OLS solutions and those defined in (3.1) will be compared with other intermediate OLS solutions given by

$$\begin{cases} \psi_{Y,i,k}(\cdot) \quad \text{solution of} \quad \mathbf{OLS}(\ S_{Y,i}(\underline{\mathbf{x}}_i)\ ,\ \mathcal{L}_{Y,k}\ ,\ \nu_{i,k,M}), \\ \psi_{Z,i,k}(\cdot) \quad \text{solution of} \quad \mathbf{OLS}(\ S_{Z,i}(w,\underline{\mathbf{x}}_i)\ ,\ \mathcal{L}_{Z,k}\ ,\ \nu_{i,k,M}). \end{cases}$$

In order to handle the dependence on the simulation clouds, we define the following σ-algebras.

DEFINITION 3.6. *Define the σ-algebras*

$$\mathcal{F}_i^{(*)} := \sigma(\mathcal{C}_{i+1,k},\dots,\mathcal{C}_{N-1,k} : 1 \le k \le K), \quad \mathcal{F}_{i,k}^{(M)} := \mathcal{F}_i^{(*)} \vee \sigma(X_i^{(i,k,m)}\ :\ 1 \le m \le M).$$

For every $i \in \{0,\dots,N-1\}$ and $k \in \{1,\dots,K\}$, let $\mathbb{E}_{i,k}^{(M)}(\cdot)$ (resp. $\mathbb{P}_{i,k}^{M}(\cdot)$) with respect to $\mathcal{F}_{i,k}^{(M)}$. Defining additionally the functions

$$\xi^*_{Y,i}(x) := \mathbb{E}\left(S_{Y,i}^{(M)}(\underline{\mathbf{X}}_i) - S_{Y,i}(\underline{\mathbf{X}}_i) \mid X_i = x, \mathcal{F}^{(M)}\right),$$
$$\xi^*_{Z,i}(x) := \mathbb{E}\left(S_{Z,i}^{(M)}(\Delta W_i, \underline{\mathbf{X}}_i) - S_{Z,i}(\Delta W_i, \underline{\mathbf{X}}_i) \mid X_i = x, \mathcal{F}^{(M)}\right),$$

now we are in the position to prove that

$$\begin{aligned}\mathbb{E}\left(\left|y_i(\cdot) - y_i^{(M)}(\cdot)\right|^2_{i,k,M}\right) &\le T^Y_{i,k} + 6\mathbb{E}\left(\left|\xi^*_{Y,i}(\cdot)\right|^2_{\nu_k}\right) + 3C^2_{3.3}\frac{\dim(\mathcal{L}_{Y,k})}{M}, \\ &+ 15L_f^2\Delta_t^2\mathbb{E}\left(\left|z_i(\cdot) - z_i^{(M)}(\cdot)\right|^2_{i,k,M}\right) \\ &+ 12168 L_f^2 \Delta_t \frac{(\dim(\mathcal{L}_{Z,k})+1)qC_y^2\log(3M)}{M},\end{aligned} \tag{3.6}$$

$$\mathbb{E}\left(\left|z_i(\cdot) - z_i^{(M)}(\cdot)\right|^2_{i,k,M}\right) \le T^Z_{i,k} + 2\mathbb{E}\left(\left|\xi^*_{Z,i}(\cdot)\right|^2_{\nu_k}\right) + 2qC^2_{3.3}\frac{\dim(\mathcal{L}_{Z,k})}{\Delta_t M}. \tag{3.7}$$

In fact, the proof of (3.6)–(3.7) follows analogously the proof of [13, (4.12)–(4.13)]; in order to follow the steps of that proof, one must note that the term R_π of that paper is equal to 1 here, C_π is equal to Δ_t, and $\theta_L = 1$. Moreover, one must exchange all norms, **OLS** problems, σ-algebras, and empirical functions from the reference to the localized versions defined in the preceding paragraphs. Indeed, the proof method of [13, (4.12)–(4.13)] is model free in the sense that it does not care about the distribution of the Markov chain at time t_i.

We now aim at aggregating the previous estimates across the strata and propagating them along time. For this, let

$$\begin{aligned}\mathcal{E}_1(i) := \sum_{j=i}^{N-1}\Delta_t\Big(T_j^Y &+ 3C^2_{3.3}\frac{\nu(\dim(\mathcal{L}_{Y,\cdot}))}{M} + 12168L_f^2\Delta_t\frac{(\nu(\dim(\mathcal{L}_{Z,\cdot}))+1)qC_y^2\log(3M)}{M} \\ &+ 3T_j^Z + 6qC^2_{3.3}\frac{\nu(\dim(\mathcal{L}_{Z,\cdot}))}{\Delta_t M}\Big)\Gamma_j,\end{aligned} \tag{3.8}$$

where $\Gamma_i := (1+\gamma\Delta_t)^i$ with γ to be determined below. Next, defining

$$\gamma := 288 c^2_{(\mathbf{A}_\nu)} C_{A.1}(1+T)L_f^2. \tag{3.9}$$

and recalling that $\Delta_t L_f^2 \leq \frac{1}{288 c_{(\mathbf{A}_\nu)}^2 C_{A.1}(1+T)}$, then γ and Δ_t satisfy

$$\max\left(\frac{1}{\gamma} \times 12 c_{(\mathbf{A}_\nu)}^2 C_{A.1}(1+T) L_f^2, \Delta_t \times 12 c_{(\mathbf{A}_\nu)}^2 C_{A.1}(1+T) L_f^2\right) \leq \frac{1}{6} \times \frac{1}{4}. \tag{3.10}$$

Additionally, $\Gamma_i \leq \exp(\gamma T) := C_{3.5}$ for every $0 \leq i \leq N$. Now, multiply (3.6) and (3.7) by $\nu(\mathcal{H}_k)\Delta_t \Gamma_i$ and sum them up over i and k to ascertain that

$$\begin{aligned}
&\sum_{j=i}^{N-1} \Delta_t \mathcal{E}(Y,M,j)\Gamma_j + \sum_{j=i}^{N-1} \Delta_t \mathcal{E}(Z,M,j)\Gamma_j \\
&\qquad \leq \sum_{j=i}^{N-1} \Delta_t \left(T_j^Y + 3C_{3.3}^2 \frac{\nu(\dim(\mathcal{L}_{Y,k}))}{M} + 12168 L_f^2 \Delta_t \frac{(\nu(\dim(\mathcal{L}_{Z,k}))+1) q C_y^2 \log(3M)}{M}\right)\Gamma_j \\
&\quad + \sum_{j=i}^{N-1} \Delta_t \left\{\left(T_j^Z + 2qC_{3.3}^2 \frac{\nu(\dim(\mathcal{L}_{Z,k}))}{\Delta_t M} + 2\mathbb{E}\left(\left|\xi_{Z,j}^*(\cdot)\right|_\nu^2\right)\right)(1+15L_f^2\Delta_t^2) + 6\mathbb{E}\left(\left|\xi_{Y,j}^*(\cdot)\right|_\nu^2\right)\right\}\Gamma_j \\
&\leq \mathcal{E}_1(i) + 6\sum_{j=i}^{N-1} \Delta_t \left(\mathbb{E}\left(\left|\xi_{Y,j}^*(\cdot)\right|_\nu^2\right) + \mathbb{E}\left(\left|\xi_{Z,j}^*(\cdot)\right|_\nu^2\right)\right)\Gamma_j\,,
\end{aligned} \tag{3.11}$$

where we have used $(1+15L_f^2\Delta_t^2) \leq 3$ (since $L_f\Delta_t \leq \sqrt{\frac{2}{15}}$), and the term $\mathcal{E}_1$ from (3.8) above. Next, from Proposition 2.1, we have

$$\mathbb{E}\left(\left|\xi_{Y,j}^*(\cdot)\right|_\nu^2\right) + \mathbb{E}\left(\left|\xi_{Z,j}^*(\cdot)\right|_\nu^2\right) \leq c_{(\mathbf{A}_\nu)}\left(\mathbb{E}\left(|\xi_{Y,j}^*(X_j^{0,\nu})|^2\right) + \mathbb{E}\left(|\xi_{Z,j}^*(X_j^{0,\nu})|^2\right)\right).$$

Furthermore, note that $(\xi_{Y,j}^*(X_j^{0,\nu}), \xi_{Z,j}^*(X_j^{0,\nu}) : 0 \leq j \leq N-1)$ solves a discrete BSDE (in the sense of Appendix A.2) with terminal condition 0 and driver

$$f_{\xi^*,j}(y,z) := f_j(X_j^{0,\nu}, y_{j+1}^{(M)}(X_{j+1}^{0,\nu}), z_j^{(M)}(X_j^{0,\nu})) - f_j(X_j^{0,\nu}, y_{j+1}(X_{j+1}^{0,\nu}), z_j(X_j^{0,\nu})).$$

This allows the application of Proposition A.1, with the first BSDE $(\xi_{Y,j}^*(X_j^{0,\nu}), \xi_{Z,j}^*(X_j^{0,\nu}) : 0 \leq j \leq N-1)$, and the second one equal to 0: since $L_{f_2} = 0$, any choice of $\gamma > 0$ is valid and we take γ as in (3.9). We obtain

$$\begin{aligned}
&\sum_{j=i}^{N-1} \Delta_t \left(\mathbb{E}\left(\left|\xi_{Y,j}^*(\cdot)\right|_\nu^2\right) + \mathbb{E}\left(\left|\xi_{Z,j}^*(\cdot)\right|_\nu^2\right)\right)\Gamma_j \\
&\leq 6 c_{(\mathbf{A}_\nu)} C_{A.1}(1+T)\left(\frac{1}{\gamma} + \Delta_t\right) L_f^2 \sum_{j=i}^{N-1} \Delta_t \\
&\times \left[\mathbb{E}\left(|y_{j+1}^{(M)}(X_{j+1}^{0,\nu}) - y_{j+1}(X_{j+1}^{0,\nu})|^2\right) + \mathbb{E}\left(|z_j^{(M)}(X_j^{0,\nu}) - z_j(X_j^{0,\nu})|^2\right)\right]\Gamma_j.
\end{aligned}$$

Now, Proposition 2.1 yields to

$$\begin{aligned}
&\mathbb{E}\left(|y_{j+1}^{(M)}(X_{j+1}^{0,\nu}) - y_{j+1}(X_{j+1}^{0,\nu})|^2\right) + \mathbb{E}\left(|z_j^{(M)}(X_j^{0,\nu}) - z_j(X_j^{0,\nu})|^2\right) \\
&\leq c_{(\mathbf{A}_\nu)}[\bar{\mathcal{E}}(Y,M,j+1) + \bar{\mathcal{E}}(Z,M,j)] \\
&\leq 2c_{(\mathbf{A}_\nu)}[\mathcal{E}(Y,M,j+1) + \mathcal{E}(Z,M,j)] + c_{(\mathbf{A}_\nu)} \frac{2028 C_y^2 \log(3M)}{M}(\nu(\dim(\mathcal{L}_{Y,\cdot})) + 1)
\end{aligned}$$

$$+ c_{(\mathbf{A}_\nu)} \frac{2028 q C_y^2 \log(3M)}{\Delta_t M} \left(\nu(\dim(\mathcal{L}_{Z,.})) + 1\right) ,$$

where the last inequality follows from the concentration-measure inequalities in Proposition 3.4. In order to summarize this, we define

$$\mathcal{E}_2(i) := \frac{1014 C_y^2 \log(3M)}{M} \left(\sum_{j=i}^{N-1} \Delta_t \Gamma_j \right) \left((\nu(\dim(\mathcal{L}_{Y,.})) + 1) + \frac{q}{\Delta_t} (\nu(\dim(\mathcal{L}_{Z,.})) + 1) \right)$$

and make use of (3.10), and that $\Gamma_j \le \Gamma_{j+1}$ in order to ascertain that we have

$$\begin{aligned}
&\sum_{j=i}^{N-1} \Delta_t \left(\mathbb{E}\left(\left|\xi^*_{Y,j}(\cdot)\right|^2_\nu \right) + \mathbb{E}\left(\left|\xi^*_{Z,j}(\cdot)\right|^2_\nu \right) \right) \Gamma_j \\
&\le 12 c^2_{(\mathbf{A}_\nu)} C_{A.1} (1+T) \left(\frac{1}{\gamma} + \Delta_t \right) L_f^2 \left[\sum_{j=i}^{N-1} \Delta_t \left(\mathcal{E}(Y,M,j) + \mathcal{E}(Z,M,j) \right) \Gamma_j + \mathcal{E}_2(i) \right] \\
&\le \frac{1}{6} \times \frac{1}{2} \left[\sum_{j=i}^{N-1} \Delta_t \left(\mathcal{E}(Y,M,j) + \mathcal{E}(Z,M,j) \right) \Gamma_j + \mathcal{E}_2(i) \right] .
\end{aligned}$$

By plugging this into (3.11) readily yields to

$$\begin{aligned}
&\sum_{j=i}^{N-1} \Delta_t \mathcal{E}(Y,M,j) \Gamma_j + \sum_{j=i}^{N-1} \Delta_t \mathcal{E}(Z,M,j) \Gamma_j \\
&\qquad\qquad \le \mathcal{E}_1(i) + \frac{1}{2} \left[\sum_{j=i}^{N-1} \Delta_t \left(\mathcal{E}(Y,M,j) + \mathcal{E}(Z,M,j) \right) \Gamma_j + \mathcal{E}_2(i) \right]
\end{aligned}$$

and therefore

$$\sum_{j=i}^{N-1} \Delta_t \mathcal{E}(Y,M,j) \Gamma_j + \sum_{j=i}^{N-1} \Delta_t \mathcal{E}(Z,M,j) \Gamma_j \le 2\mathcal{E}_1(i) + \mathcal{E}_2(i). \tag{3.12}$$

This completes the proof of the estimate (3.5) on z as stated in Theorem 3.5, using $1 \le \Gamma_i \le C_{3.5}$ and $2\mathcal{E}_1(i) + \mathcal{E}_2(i) \le C_{3.5}\mathcal{E}(i)$. It remains to derive (3.4). Starting from (3.6), multiplying by $\nu(\mathcal{H}_k)$ and summing over k yields to

$$\begin{aligned}
\mathcal{E}(Y,M,i) \le & T_i^Y + 6\mathbb{E}\left(\left|\xi^*_{Y,i}(\cdot)\right|^2_\nu \right) + 3C^2_{3.3} \frac{\nu(\dim(\mathcal{L}_{Y,.}))}{M} \\
& + 15 L_f^2 \Delta_t (2\mathcal{E}_1(i) + \mathcal{E}_2(i)) \\
& + 12168 L_f^2 \Delta_t \frac{(\nu(\dim(\mathcal{L}_{Z,.})) + 1) q C_y^2 \log(3M)}{M}
\end{aligned} \tag{3.13}$$

where we use the inequality (3.12) to control $\Delta_t \mathcal{E}(Z,M,i)$. Using the same arguments as before, we upper bound $\mathbb{E}\left(\left|\xi^*_{Y,i}(\cdot)\right|^2_\nu \right)$ by

$$6 c^2_{(\mathbf{A}_\nu)} C_{A.1} \left(\frac{1}{\gamma} + \Delta_t \right) L_f^2 \sum_{j=i}^{N-1} \Delta_t \left(\bar{\mathcal{E}}(Y,M,j) + \bar{\mathcal{E}}(Z,M,j) \right) \Gamma_j .$$

By additionally bounding $\bar{\mathcal{E}}(Y,M,j)$ and $\bar{\mathcal{E}}(Z,M,j)$ using the concentration-measure inequalities of Proposition 3.4 and plugging this in (3.13), we finally obtain

$$\begin{aligned}\mathcal{E}(Y,M,i) \leq{}& T^Y_{1,i} + 3C^2_{3.3}\frac{\nu(\dim(\mathcal{L}_{Y,.}))}{M} + 15L_f^2\Delta_t\left(2\mathcal{E}_1(i)+\mathcal{E}_2(i)\right)\\ &+ 12168L_f^2\Delta_t\frac{(\nu(\dim(\mathcal{L}_{Z,.}))+1)qC_y^2\log(3M)}{M}\\ &+ 72c^2_{(\mathbf{A}_\nu)}C_{A.1}\left(\frac{1}{\gamma}+\Delta_t\right)L_f^2\left[\sum_{j=i}^{N-1}\Delta_t\left(\mathcal{E}(Y,M,j)+\mathcal{E}(Z,M,j)\right)\Gamma_j+\mathcal{E}_2(i)\right].\end{aligned}$$

From (3.10) and (3.12), the last term in previous inequality is bounded by

$$\left(\frac{1}{4(1+T)}+\frac{1}{4(1+T)}\right)(2\mathcal{E}_1(i)+\mathcal{E}_2(i)+\mathcal{E}_2(i)) \leq \mathcal{E}_1(i)+\mathcal{E}_2(i) \leq 2\mathcal{E}_1(i)+\mathcal{E}_2(i).$$

This completes the proof of (3.6), using again $2\mathcal{E}_1(i)+\mathcal{E}_2(i) \leq C_{3.5}\mathcal{E}(i)$. □

4. GPU implementation.

In this section, we consider the computation of $y_i^{(M)}(\cdot)$ for a given stratum $\mathcal{H}_k$ and time point i. The calculation of $z_i^{(M)}(\cdot)$ is rather similar, only requiring component-wise calculations to be taken into account, so that we do not provide details. The theoretical description of the calculation was given in §3.1. In this section, we first describe the required computations to implement the approximations with **LP0** and **LP1** local polynomials in §4.1, and then present their implementation on the GPU in §4.2.[3]

4.1. Explicit solutions to OLS in Algorithm 4.

LP0. This piecewise solution is given by the simple formula [15, Ch. 4]

$$y_i^{(M)}(\cdot)|_{\mathcal{H}_k} = \mathcal{T}_{C_y}\left(\frac{\sum_{m=1}^M S^{(M)}_{Y,i}(\underline{\mathbf{X}}_i^{(i,k,m)})}{M}\right). \tag{4.1}$$

Observe that there will be a memory consumption of $O(1)$ per hypercube to store the simulations needed for the computation of $S^{(M)}_{Y,i}(\underline{\mathbf{X}}_i^{(i,k,m)})$. Once added in the sum (4.1), their allocation can be freed.

LP1. Let A be the $\mathbb{R}^M \otimes \mathbb{R}^{d+1}$ matrix, the components of which are given by $A[m,j] = 1\mathbf{1}_{\{0\}}(j)+X^{(i,k,m)}_{i,j}\mathbf{1}_{\{0\}^c}(j)$, where $X^{(i,k,m)}_{i,j}$ is the j-th component of $\underline{\mathbf{X}}_i^{(i,k,m)}$, and let S be the $\mathbb{R}^M$ vector given by $S[m] = S^{(M)}_{Y,i}(\underline{\mathbf{X}}_i^{(i,k,m)})$. In order to compute $y_i^{(M)}(\cdot)|_{\mathcal{H}_k}$, we first perform a QR-factorization $A = QR$, where Q is an $\mathbb{R}^M \otimes \mathbb{R}^M$ orthogonal matrix, and R is an $\mathbb{R}^M \otimes \mathbb{R}^{d+1}$ upper triangular matrix. The computational cost to compute this factorization is $(d+1)^2(M-(d+1)/3)$ flops using the Householder reflections method [14, Alg. 5.3.2]. Using the form of **LP1** and the density of ν_k, we can prove that the rank of A is $d+1$ with probability 1, i.e. R is invertible a.s. (the OLS problem is non-degenerate).

[3] Theoretically, we are not restricted from going to higher order local polynomials. We restrict to **LP1** for implementation in GPU due to memory limitations. In our forthcoming numerical examples, the GPU's global memory is limited to 6GB. Higher order polynomials would require not only more memory per hypercubes but also more memory for storing the regression coefficients, therefore we would be able to estimate over fewer hypercubes in parallel. Note that this is not an issue for parallel computing on CPUs.

Then, we obtain the approximation $y_i^{(M)}(\cdot)|_{\mathcal{H}_k}$ by computing the coefficients $\alpha = (\alpha_0, \dots, \alpha_d) \in \mathbb{R}^{d+1}$ using the QR factorization and backward-substitution method as follows:

$$R\alpha = Q^\top S, \qquad y_i^{(M)}(x(k)) = \mathcal{T}_{C_y}\left(\alpha_0 + \sum_{j=1}^{d} \alpha_j \times x_j(k)\right), \tag{4.2}$$

for any vector $(x(k) = (x_1(k), \dots, x_d(k))$ in $\mathcal{H}_k$. By using the Householder reflection algorithm for computing the QR-factorization, there will be memory consumption of $O\left(M \times (d+1)\right)$ for the storage of the matrix A on each hypercube. This memory can be deallocated once the computation (4.2) is completed. We remark that the memory consumption is considerably lower than other alternative QR-factorization methods, as for example the Givens rotations method [14, Alg. 5.2.2], which requires a memory consumption $O(M^2)$ to store the matrix Q. This reduced memory consumption is instrumental in the GPU approach, as we explain in forthcoming §5.2.2.

4.2. Pseudo-algorithms for GPU. Algorithm 4 will be implemented on an NVIDIA GPU device. The device architecture is built around a scalable array of multithreaded Streaming Multiprocessors (SMs); each multiprocessor is designed to execute hundreds of threads concurrently. To manage such a large amount of threads, it employs a unique architecture called SIMT (Single-Instruction, Multiple-Thread). The code execution unit is called a kernel and is executed simultaneously on all SMs by independent blocks of threads. Each thread is assigned to a single processor and executes within its own execution environment. Thus, all threads run the same instruction at a time, although over different data. In this section we briefly describe pseudo-codes for the Algorithm 4.

The algorithm has been programmed using the Compute Unified Device Architecture (CUDA) toolkit, specially designed for NVIDIA GPUs, see [20]. The code was built from an optimized C code. The below pseudo-algorithms reflect this programming feature. For the generation of the random numbers in parallel we took advantage of the NVIDIA CURAND library, see [21].

The time loop corresponding to the backward iteration of Algorithm 4 is shown in Listing 1; the kernel corresponds to the use of either the **LP0** or the **LP1** basis. In Listing 2, a sketch for the **LP0** kernel is given. Notice that we are paralellizing the loop for any stratum index $k \in \{1, \dots, K\}$ in the Algorithm 4; the terms $S_{Y,i}^{(M)}(\underline{\mathbf{x}}_i)$ and $S_{Z,i}^{(M)}(w, \underline{\mathbf{x}}_i)$ are computed in the `compute_responses_i` function, and the coefficients for $\psi_{Y,i,k}^{(M)}(\cdot)$ and $\psi_{Z,i,k}^{(M)}(\cdot)$ are computed in `compute_psi_Y` and `compute_psi_Z`, respectively, according to (4.1). Having in view an optimal performance, matrices storing the simulations, responses and regression coefficients are fully interleaved, thus allowing coalesced memory accesses, see [20]. Note that all device memory accesses are coalesced except those accesses to the regression coefficients in the resimulation stage during the computation of the responses, because one is not certain in which hypercube up each path will land. In Listing 3, the sketch for the **LP1** kernel is given. Additionally to the tasks of the kernel in Listing 2, each thread builds the matrix A and applies a QR factorization, as detailed in §4.1. Note that in addition to the matrices just explained in **LP0**, the matrix A is fully interleaved thus allowing fully coalesced accesses. The global memory is allocated at the beginning of the program and is freed at the end, thus allowing kernels to reuse already-allocated memory wherever possible. In addition to global memory, kernels are also using local memory, for example for storing the resimulated forward paths used for computing the responses. The coefficients for $\psi^{(M)}_{Y,i,k}(\cdot)$ and $\psi^{(M)}_{Z,i,k}(\cdot)$ are computed according to (4.2).

```
int i
curandState *devStates
Initialize devStates
Initialize n_blocks, n_threads_per_block
for(i=N-1; i>=0; i--)
  kernel_bsde<<<n_blocks,n_threads_per_block>>>(i, devStates, ...)
```

Listing 1
Backward iteration for $i = N - 1$ to $i = 0$.

```
__global__ void kernel_bsde_LP0(int i, curandState* devStates, ...) {
  const unsigned int global_tid = blockDim.x * blockIdx.x + threadIdx.x
  curandState localState = devStates[global_tid]

  unsigned long long int bin
  for(bin=global_tid; bin<K; bin+=n_blocks*n_threads_per_block) {
    simulates_x(&localState, global_tid, bin, ...)

    compute_responses_i(&localState, global_tid, i, ...)

    compute_psi_Z(global_tid, bin, i, ...)

    compute_psi_Y(global_tid, bin, i, ...)
  }
  devStates[global_tid] = localState
}
```

Listing 2
Kernel for the approximation with **LP0**.

```
__global__ void kernel_bsde_LP1(int i, curandState *devStates, ...) {
  const unsigned int global_tid = blockDim.x * blockIdx.x + threadIdx.x
  curandState localState = devStates[global_tid]

  unsigned long long int bin
  for(bin=global_tid;bin<K;bin+=n_blocks*n_threads_per_block) {
    simulates_x(&localState, global_tid, bin, ...)

    compute_responses_i(&localState, global_tid, i, ...)

    build_d_A(global_tid, d_A, ...)

    qr(global_tid, d_A, ...)

    compute_psi_Z(global_tid, bin, i, d_A, ...)

    compute_psi_Y(global_tid, bin, i, d_A, ...)
  }
  devStates[global_tid] = localState
}
```

Listing 3
Kernel for the approximation with **LP1**.

4.3. Theoretical complexity analysis. In this section, we assume that the functions $y_i(\cdot)$ and $z_i(\cdot)$ are smooth, namely globally Lipschitz (resp. C^1 and the first derivatives are globally Lipschitz) in the **LP0** (resp. **LP1**) case. The strata will be composed of uniform hypercubes of side length $\delta > 0$ in the domain $[-L, L]^d$, where $L = \log(N)/\mu$ and μ is the parameter of the logistic distribution. This choice ensures $\nu\left(\mathbb{R}^d \backslash [-L, L]^d\right) \leq 2d \exp(-\mu L) = O(N^{-1})$. Our aim is to calibrate the numerical parameters (number of simulations and number of strata) so that the error given in Theorem 3.5 is $O(N^{-1})$, where N is the number of time-steps. This tolerance error is the one we usually obtain after time discretization with N time points [23, 10, 22]. In the following, we focus on polynomial dependency w.r.t. N, keeping only the highest degree, ignoring constants and $\log(N)$ terms.

Squared bias errors $T^Y_{1,i}$ and $T^Z_{1,i}$ in (3.2)-(3.3). First, we remark that the approximation error of the numerical scheme, namely the error due to basis selection, depends principally on the size δ of strata. In the case of **LP0**, the squared bias error is proportional to the squared hypercube diameter plus the tail contribution, i.e. $O\left(\delta^2 + \nu\left(\mathbb{R}^d \backslash [-L, L]^d\right)\right)$; to calibrate this bias to $O(N^{-1})$, we require $\delta = O(N^{-1/2})$. In contrast, the squared bias in $[-L, L]^d$ using **LP1** is proportional to the fourth power of the hypercube diameter, whence $\delta = O(N^{-1/4})$. As a result, ignoring the log terms the number of required hypercubes is

$$\mathbf{LP0}: \quad K = O(N^{d/2}), \qquad\qquad \mathbf{LP1}: \quad K = O(N^{d/4}),$$

in both cases.

Statistical and interdependence errors. These error terms depend on the number of local polynomials, as well as on the number of simulations. Indeed, denoting $K' = \dim(\mathcal{L}_{Y \text{ or } Z,.})$ the number of local polynomials and M the number of simulations in the hypercube, then both errors are dominated by $O\left(NK' \log(M)/M\right)$, where the factor N comes from the Z part of the solution (see $\mathcal{E}(i)$ in Theorem 3.5). For **LP0** (resp. **LP1**), $K' = 1$ or q (resp. $K' = d+1$ or $q(d+1)$). This implies to select

$$\mathbf{LP0}: \quad M = O(N^2), \qquad\qquad \mathbf{LP1}: \quad M = O(N^2),$$

again omitting the log terms.

Computational cost. The computational cost (in flops) of the simulations per hypercube is equal to $O(M \times N)$, because we simulate M paths (of length N) of the process X. The cost of the regression per hypercube is $O\left(M \times N\right)$, see §4.1, and thus equivalent to the simulation cost. Putting in the values of M from the last paragraph, the overall computational cost $\mathcal{C}_{\text{cost}}$ (summed over all hypercubes and time steps) is

$$\mathbf{LP0}: \quad \mathcal{C}^{\texttt{SEQ}}_{\text{cost}} = O(N^{4+d/2}), \qquad\qquad \mathbf{LP1}: \quad \mathcal{C}^{\texttt{SEQ}}_{\text{cost}} = O(N^{4+d/4}).$$

This quantity is related to the computational time for a sequential system (SEQ implementation) where there is no parallel computing. For the GPU implementation, described in §4.2, there is an additional computational time improvement since the computations on the hypercubes will be threaded across the cores of the card. Thus, the computational cost on GPU is

$$\mathbf{LP0}: \quad \mathcal{C}^{\texttt{GPU}}_{\text{cost}} = O(N^{4+d/2})/\mathcal{C}_{\text{Load factor}}, \quad \mathbf{LP1}: \quad \mathcal{C}^{\texttt{GPU}}_{\text{cost}} = O(N^{4+d/4})/\mathcal{C}_{\text{Load factor}}.$$

where the load factor $\mathcal{C}_{\text{Load factor}}$ is ideally the number of threads on the device.

Finally, we quantify the improvement in memory consumption offered by the SRMDP algorithm compared to the LSMDP algorithm of [13]. This is a very important improvement, because, as explained in the introduction, the memory is the key constraint in solving problems in high dimension. We only compare sequential versions of the algorithms, meaning that the computational costs will be the same. The main difference between the two schemes is then in the number of simulations that must be stored in the machine at any given time. We summarize this in Table 1 below.

In SRMDP, the memory consumption is mainly related to storing coefficients representing the solutions on hypercubes, that is $O(N \times \dim(\mathcal{L}_{Y \text{ or } Z,.}) \times K)$; if one is using the **LP1** basis, one must also take into account the memory consumption per

Algorithm	Number of simulations		Computational cost	
	LP0	**LP1**	**LP0**	**LP1**
SRMDP	N^2	N^2	$N^{4+d/2}$	$N^{4+d/4}$
LSMDP	$N^{2+d/2}$	$N^{2+d/4}$	$N^{4+d/2}$	$N^{4+d/4}$

TABLE 1
Comparison of numerical parameters with or without stratified sampling, as a function of N.

strata $M \times (d+1) = O(N^2)$ for the QR factorization, explained in §4.1. In contrast, the memory consumption for LSMDP is mainly $O(K \times N^2)$, which represents the number of simulated paths of the Markov chains that must be stored in the machine at any given time. We summarize the memory consumption of the two algorithms in Table 2.

Algorithm	**LP0**	**LP1**
SRMDP	$N^{1+d/2}$	$N^{1+d/4} \vee N^2$
LSMDP	$N^{2+d/2}$	$N^{2+d/4}$

TABLE 2
Comparison of memory requirement as a function of N.

Observe that SRMDP requires N times less memory than LSMDP with the **LP0** basis. This implicitly implies a gain of 2 on the dimension d that can be handled. On the other hand, if the **LP1** basis is used, the SRMDP requires $O(N^{d/4})$ less memory for $d \leq 4$ than LSMDP, and N times less memory for $d \geq 4$. Therefore, there is an implicit gain of 4 in the dimension that can be handled by the algorithm.

5. Numerical experiments.

5.1. Model, stratification, and performance benchmark.

We use the Brownian motion model $X = W$ $(d = q)$. Moreover, the numerical experiments will consider the performance according to the dimension d. We introduce the function $\omega(t,x) = \exp(t + \sum_{k=1}^{q} x_k)$. We perform numerical experiments on the BSDE with data $g(x) = \omega(T,x)(1+\omega(T,x))^{-1}$ and

$$f(t,x,y,z) = \left(\sum_{k=1}^{q} z_k\right)\left(y - \frac{2+q}{2q}\right),$$

where $z = (z_1, \dots, z_q)$. The BSDE has explicit solutions in this framework, given by

$$y_i(x) = \omega(t_i,x)(1+\omega(t_i,x))^{-1}, \qquad z_{k,i}(x) = \omega(t_i,x)(1+\omega(t_i,x))^{-2},$$

where $z_{k,i}(x)$ is the k-th component of the q-dimensional cylindrical function $z_i(x) \in \mathbb{R}^q$.

The logistic distribution for Algorithm 4 is parameterized by $\mu = 1$ and we consider $T = 1$. For the least-squares Monte Carlo, we stratify the domain $[-6.5, 6.5]^q$ with uniform hypercubes. To assess the performance of the algorithm, we compute the average mean squared error (MSE) over 10^3 independent runs of the algorithm for three error indicators:

$$MSE_{Y,\max} := \ln\left\{10^{-3} \max_{0\le i\le N-1} \sum_{m=1}^{10^3} |y_i(R_{i,m}) - y_i^{(M)}(R_{i,m})|^2\right\},$$

$$MSE_{Y,\mathrm{av}} := \ln\left\{10^{-3}N^{-1} \sum_{m=1}^{10^3}\sum_{i=0}^{N-1} |y_i(R_{i,m}) - y_i^{(M)}(R_{i,m})|^2\right\},$$

$$MSE_{Z,\mathrm{av}} := \ln\left\{10^{-3}N^{-1} \sum_{m=1}^{10^3}\sum_{i=0}^{N-1} |z_i(R_{i,m}) - z_i^{(M)}(R_{i,m})|^2\right\},$$

where the simulations $\{R_{i,m}; i = 0,\dots,N-1,\ m = 1,\dots,10^3\}$ are independent and identically ν-distributed, and independently drawn from the simulations used for the LSMC scheme. We parameterize the hypercubes according to the instructions given in the theoretical complexity analysis, see §4.3. In particular, we consider different values of N and always set $K = O(N^{d/2})$ in **LP0** (resp. $O(N^{d/4})$ in **LP1**) and $M = O(N^2)$. Note, however, that we do not specify the value of δ, but rather the number of hypercubes per dimension $K^{1/q}$, which we denote `#C` in what follows; this being equivalent to setting δ, but is more convenient to program. As we shall illustrate, the error converges as predicted as N increases, although the exact error values will depend on the constants that we choose in the parameterization of K and M.

5.2. CPU and GPU performance. In this section, several experiments based on §5.1 are presented to assess the performance of CUDA implementation of Algorithm 4; the pseudo-algorithms are given in §4.2. We shall compare its performance with a version of SRMDP implemented to run on multicore CPUs. For the design of this comparison we have followed some ideas in [17]. Moreover, in order to test the theoretical results of §4.3, we compare the performance of the two algorithms according to the choice of the basis functions, the impact of this choice on the convergence of the approximation of the BSDE, and the impact of this choice on the computational performance in terms of computational time and memory consumption.

There are two types of basis functions we investigate: **LP0** in §5.2.1, and **LP1** in §5.2.2. As explained in §4.3, the **LP0** basis is highly suited to GPU implementation because it has a very low memory requirement per thread of computation. On the other hand, it has a very high global memory requirement for storing coefficients. This represents a problem in high dimensions because one needs many coefficients to obtain a good accuracy. On the other hand, the **LP1** basis involves a higher cost per thread, although requires a far lower global memory for storing coefficients; this implies that the impact of the GPU implementation is lower in moderate dimensional problems, but that one can solve problems in higher dimension. Moreover, the full performance impact of the GPU implementation on the **LP1** basis is in high dimension, where the number of strata is very high and therefore the GPU is better saturated with computations. We illustrate numerically all of these effects in the following sections.

The numerical experiments have been performed with the following hardware and software configurations: a GPU GeForce GTX TITAN Black with 6 GBytes of global memory (see [19] for details in the architecture), two recent multicore Intel Xeon CPUs E5-2620 v2 clocked at 2.10 GHz (6 cores per socket) with 62 GBytes of RAM, CentOS Linux, NVIDIA CUDA SDK 7.5 and INTEL C compiler 15.0.6. The CPU programs were optimized and parallelized using OpenMP [24]. Since the CUDA code has been derived from an optimized C code, both codes perform the same algorithms, and their performance can be fairly compared according to computational times; the multicore CPUs time (CPU) and the GPU time (GPU) will all be measured in seconds in the forthcoming tables. CPU times correspond to executions using 24 threads so as to take advantage of Intel Hyperthreading. The speedups of the CPU parallel version with respect to pure sequential CPU code are around 16. The results are obtained in single precision, both in CPU and GPU.

5.2.1. Examples with the approximation with LP0 local polynomials. All examples will be run using 64 thread blocks each with 256 threads. In Table 3 we show results for $d = 4$, with `#C`=$\left\lfloor 4\sqrt{N} \right\rfloor$ and $M = N^2$. Except for the cases $\Delta_t = 0.2$ and $\Delta_t = 0.1$ where there are not enough strata to fully take advantage of the GPU, the GPU implementation provides a significant reduction in the computational time: the GPU speed-up reaches the value 18.90. Moreover, the speed-up improves as we increase the `#C`.

Δ_t	`#C`	K	M	$MSE_{Y,\max}$	$MSE_{Y,\text{av}}$	$MSE_{Z,\text{av}}$	CPU	GPU
0.2	8	4096	25	-3.712973	-3.774071	-0.964842	0.23	2.00
0.1	12	20736	100	-4.066741	-4.303750	-1.607104	5.23	2.20
0.05	17	83521	400	-4.337988	-4.698645	-2.302092	171.92	12.39
0.02	28	614656	2500	-4.472564	-4.988069	-3.225411	58066.33	3070.92

Table 3
LP0 *local polynomials,* $d = 4$, `#C`=$\left\lfloor 4\sqrt{N} \right\rfloor$, $M = N^2$.

Tables 4 and 5 show results for $d = 6$ with `#C`=$\left\lfloor \sqrt{N} \right\rfloor$ and `#C`=$\left\lfloor 2\sqrt{N} \right\rfloor$, respectively. Convergence is clearly improved by doubling `#C`. In Table 5 the case of $\Delta_t = 0.02$ is not shown due to insufficient GPU global memory [4]. In Table 4, the GPU speed-up reaches 15.93, whereas in Table 5 it reaches 14.85. As in Table 3, the increase in the speed-up is explained due to the increased number of hypercubes, thus demonstrating how important it is to have many hypercubes in the GPU implementation. However, the finer basis requires 2^6 times as much memory for storing coefficients.

Δ_t	`#C`	K	M	$MSE_{Y,\max}$	$MSE_{Y,\text{av}}$	$MSE_{Z,\text{av}}$	CPU	GPU
0.2	2	64	25	-2.392320	-2.451332	-0.431059	0.21	1.99
0.1	3	729	100	-2.440274	-2.500775	-1.096603	0.47	2.05
0.05	4	4096	400	-2.829757	-2.905192	-1.687142	17.21	3.15
0.02	7	117649	2500	-3.235130	-3.539011	-2.557686	13930.70	874.25

Table 4
LP0 *local polynomials,* $d = 6$, `#C`=$\left\lfloor \sqrt{N} \right\rfloor$, $M = N^2$.

Table 6 shows that the algorithm can work for $d = 11$ in several seconds with a reasonable accuracy in a GPU. The corresponding speed-up with respect to CPU version is around 13.35. For the execution with $\Delta_t = 0.1$ we are going to report the GFlop rate, and also the memory transfer to/from the global memory. Inside the kernel, the functions computing the regression coefficients (denoted by `compute_psi_Z`

[4]For $\Delta_t = 0.02$, the array for storing the regression coefficients will be of size $N \times K \times (D + 1)$, i.e. $50 \times 14^6 \times 7 \times 4$ bytes using single precision, which equals 9.81 GBytes, much greater than the available 6 GBytes of device memory.

Δ_t	#C	K	M	$MSE_{Y,\max}$	$MSE_{Y,\text{av}}$	$MSE_{Z,\text{av}}$	CPU	GPU
0.2	4	4096	25	−2.707882	−2.784022	−0.477751	0.29	1.94
0.1	6	46656	100	−3.195937	−3.294488	−1.133834	13.72	2.44
0.05	8	262144	400	−3.505867	−3.664396	−1.795697	775.33	52.20

TABLE 5
LP0 *local polynomials,* $d = 6$, #C$=\left\lfloor 2\sqrt{N}\right\rfloor$, $M = N^2$.

and `compute_psi_Y` in the Listing 2) are memory bounded, reaching 236.795 GBytes/s when reading/writing from/to global memory. The rest of the kernel is more computationally limited. In the overall kernel, the memory transfer from/to global memory is around 160 GBytes/s and the Gflop rate is around 238 GFlop/s, although around the 30% of the instructions executed by the kernel are integer instructions to assign the simulations to the strata in the resimulation stage during the computation of the responses.

Δ_t	#C	K	M	$MSE_{Y,\max}$	$MSE_{Y,\text{av}}$	$MSE_{Z,\text{av}}$	CPU	GPU
0.2	2	2048	25	−2.152253	−2.202357	0.211590	0.27	1.99
0.1	3	177147	100	−2.144843	−2.267742	−0.469759	67.96	6.29
0.05	4	4194304	400	−2.484169	−2.633602	−1.070096	28154.07	2108.64

TABLE 6
LP0 *local polynomials,* $d = 11$, #C$=\left\lfloor \sqrt{N}\right\rfloor$, $M = N^2$.

5.2.2. Examples with the approximation with LP1 local polynomials. In this section we show the results corresponding to the approximation with the **LP1** basis. Compared to **LP0**, this basis consumes much less global memory to store coefficients, because it requires far fewer hypercubes, see §4.3. On the other hand, the approximation with **LP1** basis demands higher thread memory due to the storage of a large matrix for each hypercube, as explained in §4.3. This may have an impact on the computational time on the GPU: recalling from §4.2 the GPU handles multiple hypercubes at any given time, each one requiring the storage of a matrix A, the global memory capacity of the GPU device restricts the number of threads we can handle at any given time. This issue is much less significant with the **LP0** basis. In order to optimize the performance of the **LP1** basis, we must minimize the thread memory storage. We implement the Householder reflection method for QR-factorization, [14, Alg. 5.3.2]. For this, we must store a matrix containing $M \times (d+1) = O(N^2)$ floating point values per thread on the GPU memory. Thanks to the reduced global memory storage for coefficients, we are able to work in a rather high dimension $d = 19$.

REMARK 5.1. *There are many methods to implement QR-factorization. However, the choice of method has a substantial impact on the performance of the GPU implementation. For example, the Givens rotation method [14, Alg. 5.2.2] requires the storage of an $M \times M$ matrix, which corresponds to $O(N^4)$ floating points. This is rather more than the required $O(N^2)$ for the Householder reflection method given in Section 4.1. Therefore, the Givens rotation method would be far slower when implemented on a GPU than the Householder reflection method, because it may not be possible to use an optimal thread configuration.*

REMARK 5.2. *In the forthcoming examples, we use more simulations per stratum for the* **LP1** *basis compared to the equivalent results for* **LP0**. *This is to account for the additional statistical and interdependence errors, as explained in §4.3.*

In Table 7, we present results for $d = 4$. These results are to be compared with Table 3, where in particular the $MSE_{Z,\text{av}}$ results are closer line to line. The computational time is substantially improved for the CPU and GPU calculations. Also note that, unlike for the Z component, the accuracy for the Y component is substantially better for the **LP1** basis than for the **LP0** one. The difference in the accuracy results between the Y and Z components is likely explained by the fact that the function $x \mapsto z_i(x)$ is rather flat, so it is much better approximated by **LP0** basis functions than $x \mapsto y_i(x)$. The GPU speed-up reaches 8.05.

Δ_t	#C	K	M	$MSE_{Y,\max}$	$MSE_{Y,\text{av}}$	$MSE_{Z,\text{av}}$	CPU	GPU
0.2	3	81	125	-4.021483	-4.131725	-0.900286	0.11	0.23
0.1	5	625	500	-4.290881	-4.695769	-1.551480	1.26	0.79
0.05	7	2401	2000	-4.541253	-5.022405	-2.281332	43.56	7.83
0.02	10	10000	12500	-4.574551	-5.143310	-3.228237	6827.98	847.83

Table 7

LP1 *local polynomials,* $d = 4$, #C$=\left\lfloor 3\sqrt{d\sqrt{N}} - 5\right\rfloor$, $M = (d+1)N^2$.

Next, results for $d = 6$ are shown. Thus, we compare Table 8 below with Table 5. For a given precision on the Z component of the solution, we observe substantial improvements in the CPU codes, but no such gains on the GPU version. In contrast, the accuracy of the Y approximation is, as in the $d = 4$ case, substantially better. Moreover, whereas we were not able to do computations for $\Delta_t = 0.02$ with the **LP0** basis due to insufficient GPU memory, we are now able to make these calculations with the **LP1** basis. The GPU speed-up reaches 6.13, which is lower than the **LP0** basis speed-up factor, as expected.

Δ_t	#C	K	M	$MSE_{Y,\max}$	$MSE_{Y,\text{av}}$	$MSE_{Z,\text{av}}$	CPU	GPU
0.2	2	64	175	-3.504153	-3.668801	-0.461077	0.20	0.32
0.1	3	729	700	-3.804091	-3.911488	-1.133263	1.84	1.66
0.05	4	4096	2800	-4.075928	-4.231639	-1.791519	125.81	20.50
0.02	6	46656	17500	-3.809734	-4.529827	-2.689432	82529.21	15283.18

Table 8

LP1 *local polynomials,* $d = 6$, #C$=\left\lfloor 1.5\sqrt{d\sqrt{N}} - 3\right\rfloor$, $M = (d+1)N^2$.

In the high dimensional $d = 11$ setting shown in Table 9, we compare with Table 6. We observe a speed-up of order 5.63 compared to the CPU implementation.

Δ_t	#C	K	M	$MSE_{Y,\max}$	$MSE_{Y,\text{av}}$	$MSE_{Z,\text{av}}$	CPU	GPU
0.2	2	2048	2000	-3.271648	-3.368051	-1.455388	10.33	3.41
0.2	3	177147	4000	-3.269004	-3.403994	-1.975300	1635.95	290.56

Table 9

LP1 *local polynomials,* $d = 11$.

In the remainder of this section, we present results in dimension $d = 12$ to $d = 19$ (in Tables 10, 11, 12 and 13, respectively) for which the capacity of the GPU is maximally used to provide the highest possible accuracy. The GPU speed-up reaches up to 5.67 compared to the CPU implementation. Note that for the example with $d = 19$ in Table 13 the LSMDP algorithm would require 118 GBytes of memory to store all the simulations at a given time, whereas the here proposed SRMDP algorithm can be executed with less than 6 GBytes and with much less computational time owing to it does not need to associate the simulations to hypercubes. Finally, for the example with $\Delta_t = 0.2$, `#C`$= 2$ and $M = 4000$ of Table 11 we next report the GFlop rate and the memory transfer from/to global memory. In the overall kernel, the memory transfer from/to global memory is around 132 GBytes/s and the GFlop rate is around 136 GFlop/s. In order to understand why the device memory bandwidth used by LP1 kernel is lower than the one used by LP0 kernel, observe that at any given time, for each strata we are accessing $(d + 1)$ times more elements in the LP1 framework in the re-simulation stage of the responses computation. Moreover, these accesses are potentially non-coalesced, because the forward process is randomly re-simulated and we do not know a priori in which strata is going to fall.

Δ_t	#C	K	M	$MSE_{Y,\max}$	$MSE_{Y,\text{av}}$	$MSE_{Z,\text{av}}$	CPU	GPU
0.2	2	4096	2000	−3.111153	−3.232051	−1.297737	22.29	4.95
0.2	3	531441	4000	−3.214096	−3.272644	−1.821935	5554.49	1196.28

TABLE 10
LP1 *local polynomials,* $d = 12$.

Δ_t	#C	K	M	$MSE_{Y,\max}$	$MSE_{Y,\text{av}}$	$MSE_{Z,\text{av}}$	CPU	GPU
0.2	2	8192	3000	−2.995413	−3.153302	−1.460911	69.45	12.46
0.2	2	8192	4000	−3.022855	−3.158471	−1.649632	94.07	16.58

TABLE 11
LP1 *local polynomials,* $d = 13$.

Δ_t	#C	K	M	$MSE_{Y,\max}$	$MSE_{Y,\text{av}}$	$MSE_{Z,\text{av}}$	CPU	GPU
0.2	2	16384	2000	−3.011673	−3.092870	−1.026128	102.11	19.55
0.2	2	16384	4000	−3.029663	−3.105833	−1.558935	205.82	50.62

TABLE 12
LP1 *local polynomials,* $d = 14$.

Appendix A.

A.1. Proof of Proposition 2.1. It is known from [11, Proposition 3.1] that it is sufficient to show that there is a continuous $C_\rho : \mathbb{R} \to [1, \infty)$ such that, for all $\Lambda \geq 0$, $\lambda \in [0, \Lambda]$, and $y \in \mathbb{R}^d$,

$$\frac{p^{(\mu)}_{\text{logis.}}(y)}{C_\rho(\Lambda)} \leq \int_{\mathbb{R}^d} p^{(\mu)}_{\text{logis.}}(y + z\sqrt{\lambda}) \exp(-\frac{|z|^2}{2})\mathrm{d}z \leq C_\rho(\Lambda) p^{(\mu)}_{\text{logis.}}(y). \tag{A.1}$$

The proof is given for $d = 1$; generalization to higher dimensions is obvious because the multidimensional density is just the product of the one-dimensional densities over the components. Moreover, for simplicity the proof is given for $\mu = 1$, as generality in this parameter does not change the proof. For simplicity, we will write $p^{(\mu)}_{\text{logis.}}(x) = p(x)$ in what follows.

In terms of the hyperbolic cosine function, the density can be expressed as

$$p(x) = \frac{\exp(-x)}{\big(1 + \exp(-x)\big)^2} = \Big(\exp(\frac{x}{2}) + \exp(-\frac{x}{2})\Big)^{-1} = \Big(2\cosh(\frac{x}{2})\Big)^{-1}.$$

d	K	M	$MSE_{Y,\max}$	$MSE_{Y,\text{av}}$	$MSE_{Z,\text{av}}$	CPU	GPU
15	32768	5000	−2.981181	−3.106590	−1.574532	578.88	139.60
16	65536	6000	−2.795353	−2.959375	−1.588716	1411.75	429.53
17	131072	5000	−2.772595	−2.936549	−1.371146	2580.06	793.61
18	262144	4000	−2.845755	−2.918057	−1.114600	4275.13	1589.30
19	524288	3200	−2.726427	−2.851617	−0.839849	7245.91	4370.31

Table 13
LP1 *local polynomials,* $d = 15, \dots, 19$, $\Delta_t = 0.2$, $\#C = 2$.

We define $I(y,\lambda) := 2\int_{\mathbb{R}} p(y + z\sqrt{\lambda}) \exp(-\frac{z^2}{2})\mathrm{d}z$, so that from the relation $\cosh(x + y) = \cosh(x)\cosh(y) + \sinh(x)\sinh(y)$, we have that

$$I(y,\lambda) = \int_{\mathbb{R}} \frac{\exp(-\frac{z^2}{2})}{\cosh(\frac{y}{2})\cosh(\frac{z\sqrt{\lambda}}{2}) + \sinh(\frac{y}{2})\sinh(\frac{z\sqrt{\lambda}}{2})}\mathrm{d}z := I_+(y,\lambda) + I_-(y,\lambda)$$

where $I_{+,-}$ denotes respectively the integral on $\mathbb{R}^+$ and $\mathbb{R}^-$.

Upper bound. Suppose that $y \geq 0$. Then, if $z \geq 0$, it follows that $\sinh(y/2)\sinh(z\sqrt{\lambda}/2) \geq 0$, whence

$$I_+(y,\lambda) \leq \int_{\mathbb{R}_+} \frac{\exp(-\frac{z^2}{2})}{\cosh(\frac{y}{2})\cosh(\frac{z\sqrt{\lambda}}{2})}\mathrm{d}z = 2\int_{\mathbb{R}_+} e^{-\frac{z^2}{2}}\mathrm{d}z \times p(y).$$

On the other hand, if $z \leq 0$, then $\sinh(\frac{y}{2})\sinh(\frac{z\sqrt{\lambda}}{2}) \geq \cosh(\frac{y}{2})\sinh(\frac{z\sqrt{\lambda}}{2})$, therefore

$$I_-(y,\lambda) \leq \int_{\mathbb{R}_-} \frac{\exp(-\frac{z^2}{2})}{\cosh(\frac{y}{2})\{\cosh(\frac{z\sqrt{\lambda}}{2}) + \sinh(\frac{z\sqrt{\lambda}}{2})\}}\mathrm{d}z = 2\int_{\mathbb{R}_-} \exp\left(-\frac{z^2}{2} - \frac{z\sqrt{\lambda}}{2}\right)\mathrm{d}z \times p(y).$$

Therefore, if $y \geq 0$ then $I(y,\lambda) \leq 2\int_{\mathbb{R}} \exp(\frac{-z^2+(z)_-\sqrt{\Lambda}}{2})\mathrm{d}z \times p(y)$. Observing that $I(y,\lambda)$ is symmetric in y, thus the upper bound (A.1) is proved.

Lower bound. Suppose that $y \geq 0$. For $z \leq 0$, observe that $\sinh(\frac{y}{2})\sinh(\frac{z\sqrt{\lambda}}{2}) \leq 0$, whence

$$I_-(y,\lambda) \geq \int_{\mathbb{R}_-} \frac{\exp(-\frac{z^2}{2})}{\cosh(\frac{y}{2})\cosh(\frac{z\sqrt{\lambda}}{2})}\mathrm{d}z \geq 2\int_{\mathbb{R}_-} \frac{\exp(-\frac{z^2}{2})}{\cosh(\frac{z\sqrt{\Lambda}}{2})}\mathrm{d}z \times p(y).$$

For $z \geq 0$, we use that $\sinh(\frac{y}{2})\sinh(\frac{z\sqrt{\lambda}}{2}) \leq \cosh(\frac{y}{2})\sinh(\frac{z\sqrt{\lambda}}{2})$ to obtain

$$\begin{aligned} I_+(y,\lambda) &\geq \int_{\mathbb{R}_+} \frac{\exp(-\frac{z^2}{2})}{\cosh(\frac{y}{2})\{\cosh(\frac{z\sqrt{\lambda}}{2}) + \sinh(\frac{z\sqrt{\lambda}}{2})\}}\mathrm{d}z \\ &\geq 2\int_{\mathbb{R}_+} \exp\left(-\frac{z^2}{2} - \frac{z\sqrt{\Lambda}}{2}\right)\mathrm{d}z \times p(y). \end{aligned}$$

The result on $y < 0$ follows again from the symmetry of $I(y,\lambda)$. □

A.2. Stability results for discrete BSDE. We recall standard results borrowed to [13] and adapted to our setting, they are aimed at comparing two solutions of discrete BSDEs of the form (2.1) with different data. Namely, consider two discrete BSDEs, $(Y_{1,i}, Z_{1,i})_{0\le i<N}$ and $(Y_{2,i}, Z_{2,i})_{0\le i<N}$, given by

$$Y_{l,i} = \mathbb{E}_i\left(g(X_N) + \sum_{j=i}^{N-1} f_{l,j}(X_j, Y_{l,j+1}, Z_{l,j})\Delta_t\right),$$

$$\Delta_t Z_{l,i} = \mathbb{E}_i\left((g(X_N) + \sum_{j=i+1}^{N-1} f_{l,j}(X_j, Y_{l,j+1}, Z_{l,j})\Delta_t)\Delta W_i\right),$$

for $i \in \{0, \dots, N-1\}$, $l \in \{1, 2\}$.

To allow the driver $f_{1,i}$ to depend on the clouds of simulations (necessary in the analysis), we require that it is measurable w.r.t. $\mathcal{F}_T$ instead of $\mathcal{F}_{t_i}$ as usually.

Proposition A.1. *Assume that* $(\mathbf{A_g})$ *and* $(\mathbf{A_X})$*hold. Moreover, for each* $i \in \{0, \dots, N-1\}$*, assume that* $f_{1,i}(X_i, Y_{1,i+1}, Z_{1,i}) \in \mathbf{L}_2(\mathcal{F}_T)$ *and* f_2 *satisfies* $(\mathbf{A_f})$ *with constants* L_{f_2} *and* C_{f_2}*. Then, for any* $\gamma \in (0, +\infty)$ *satisfying* $6q(\Delta_t + \frac{1}{\gamma})L_{f_2}^2 \le 1$*, we have for* $0 \le i < N$

$$|Y_{1,i} - Y_{2,i}|^2\Gamma_i + \sum_{j=i}^{N-1} \Delta_t\mathbb{E}_i\left(|Z_{1,j} - Z_{2,j}|^2\right)\Gamma_j$$
$$\le 3C_{A.1}\left(\frac{1}{\gamma} + \Delta_t\right)\sum_{j=i}^{N-1} \Delta_t\mathbb{E}_i\left(|f_{1,j}(X_j, Y_{1,j+1}, Z_{1,j}) - f_{2,j}(X_j, Y_{1,j+1}, Z_{1,j})|^2\right)\Gamma_j,$$

where $\Gamma_i := (1 + \gamma\Delta_t)^i$ *and* $C_{A.1} := 2q + (1 + T)e^{T/2}$.

Acknowledgments

The authors are gratefull to the three anonymous referees whose remarks have helped us to improve the final version of the paper.

REFERENCES

[1] L. Abbas-Turki and B. Lapeyre, *American options pricing on multi-core graphic cards*, in Business Intelligence and Financial Engineering, 2009. BIFE'09. International Conference on, IEEE, 2009, pp. 307–311.

[2] L. Abbas-Turki, S. Vialle, B. Lapeyre, and P. Mercier, *High dimensional pricing of exotic European contracts on a GPU cluster, and comparison to a CPU cluster*, in Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on, IEEE, 2009, pp. 1–8.

[3] C. Bender and J. Steiner, *Least-squares Monte Carlo for BSDEs*, in Numerical Methods in Finance, R. Carmona, P. Del Moral, P. Hu, and N. Oudjane, eds., Series: Springer Proceedings in Mathematics, Vol. 12, 2012, pp. 257–289.

[4] B. Bouchard and X. Warin, *Monte-Carlo valuation of American options: facts and new algorithms to improve existing methods*, in Numerical Methods in Finance, R. Carmona, P. Del Moral, P. Hu, and N. Oudjane, eds., Series: Springer Proceedings in Mathematics, Vol. 12, 2012, pp. 215–255.

[5] P. Briand and C. Labart, *Simulation of BSDEs by Wiener Chaos Expansion*, Annals of Applied Probability, 24 (2014), pp. 1129–1171.

[6] M. Fatica and E. Phillips, *Pricing American options with least squares Monte Carlo on GPUs*, Proceeding of the 6th Workshop on High Performance Computational Finance, ACM (2013). doi>10.1145/2535557.2535564.

[7] D. Funaro, *Polynomial approximation of differential equations*, Lecture Notes in Physics. New Series m: Monographs, Volume 8, Springer-Verlag, Berlin, 1992.

[8] E. Gobet and C. Labart, *Solving BSDE with adaptive control variate*, SIAM Numerical Analysis, 48 (2010), pp. 257–277.

[9] E. Gobet, J-P. Lemor, and X. Warin, *A regression-based Monte Carlo method to solve backward stochastic differential equations*, Annals of Applied Probability, 15 (2005), pp. 2172–2202.

[10] E. Gobet and A. Makhlouf, *L_2-time regularity of BSDEs with irregular terminal functions*, Stochastic Processes and their Applications, 120 (2010), pp. 1105–1132.

[11] E. Gobet and P. Turkedjiev, *Adaptive importance sampling in least-squares Monte Carlo algorithms for backward stochastic differential equations*, Preprint, hal-01169119, accepted for publication in Stochastic Processes and Applications, (2015).

[12] ———, *Approximation of BSDEs using Malliavin weights and least-squares regression*, Bernoulli, 22(1) (2016), pp. 530–562.

[13] ———, *Linear regression MDP scheme for discrete backward stochastic differential equations under general conditions*, Mathematics of Computation, 85(299) (2016), pp. 1359–1391.

[14] G. H. Golub and C. F. Van Loan, *Matrix computations*, Johns Hopkins Studies in the Mathematical Sciences, Johns Hopkins University Press, Baltimore, MD, third ed., 1996.

[15] L. Györfi, M. Kohler, A. Krzyżak, and H. Walk, *A distribution-free theory of nonparametric regression*, Springer Series in Statistics, Springer-Verlag, New York, 2002.

[16] C. Labart and J. Lelong, *A parallel algorithm for solving BSDEs*, Monte Carlo Methods Appl., 19 (2013), pp. 11–39.

[17] V-W. Lee, C. Kim, J. Chhugani, M. Deisher, D. Kim, A-D. Nguyen, N. Satish, M. Smelyanskiy, S. Chennupaty, P. Hammarlund, R. Singhal, P. Dubey, *Debunking the 100X GPU vs. CPU myth: an evaluation of throughput computing on CPU and GPU*, Newsletter ACM SIGARCH Computer Architecture News - ISCA '10, 38 (3) (2010), pp. 451–460.

[18] J-P. Lemor, E. Gobet, and X. Warin, *Rate of convergence of an empirical regression method for solving generalized backward stochastic differential equations*, Bernoulli, 12 (2006), pp. 889–916.

[19] NVIDIA, *NVIDIA's Next Generation CUDA Compute Architecture: Kepler GK110/210.*

[20] ———, *CUDA C Programming Guide*, March 2015.

[21] ———, *CURAND Library*, March 2015.

[22] P. Turkedjiev, *Two algorithms for the discrete time approximation of Markovian backward stochastic differential equations under local conditions*, Electronic Journal of Probability, 20 (2015).

[23] J. Zhang, *A numerical scheme for BSDEs*, The Annals of Applied Probability, 14 (2004), pp. 459–488.

[24] OpenMP, `www.openmp.org`