proceedings

MDPI

Proceedings

Quasi-Regression Monte-Carlo Method for Semi-Linear PDEs and BSDEs †

Emmanuel Gobet [1], José Germán López Salas [2,*] and Carlos Vázquez [2]

[1] Centre de Mathématiques Appliquées, École Polytechnique and CNRS, route de Saclay, 91128 Palaiseau CEDEX, France
[2] Department of Mathematics, Faculty of Informatics, Universidade da Coruña, Campus de Elviña s/n, 15071 A Coruña, Spain
* Correspondence: jose.lsalas@udc.es
† Presented at the 2nd XoveTIC Conference, A Coruña, Spain, 5–6 September 2019.

Published: 6 August 2019

Abstract: In this work we design a novel and efficient quasi-regression Monte Carlo algorithm in order to approximate the solution of discrete time backward stochastic differential equations (BSDEs), and we analyze the convergence of the proposed method. With the challenge of tackling problems in high dimensions we propose suitable projections of the solution and efficient parallelizations of the algorithm taking advantage of powerful many core processors such as graphics processing units (GPUs).

Keywords: BSDEs; semi-linear PDEs; parallel computing; GPUs; CUDA

1. Introduction

In this work we are interested in numerically approximating the solution (X, Y, Z) of a decoupled forward-backward stochastic differential equation

$$Y_t = g(X_T) + \int_t^T f(s, X_s, Y_s)\mathrm{d}s - \int_t^T Z_s \mathrm{d}W_s, \tag{1}$$

$$X_t = x + \int_0^t b(s, X_s)\mathrm{d}s + \int_0^t \sigma(s, X_s)\mathrm{d}W_s. \tag{2}$$

The terminal time $T > 0$ is fixed. These equations are considered in a filtered probability space $(\Omega, \mathcal{F}, \mathbb{P}, (\mathcal{F}_t)_{0\le t\le T})$ supporting a $q \ge 1$ dimensional Brownian motion W. In this representation, X is a d-dimensional adapted continuous process (called the forward component), Y is a scalar adapted continuous process and Z is a q-dimensional progressively measurable process. Regarding terminology, $g(X_T)$ is called *terminal condition* and f the *driver*.

2. Results

Our aim is to solve

$$Y_i = \mathbb{E}\left[g(X_N) + \sum_{j=i}^{N-1} f_j(X_j, Y_{j+1})\Delta \mid \mathcal{F}_{t_i}\right] \quad \text{for } i \in \{N-1, \dots, 0\}, \tag{3}$$

where $f_j(x,y) := f(t_j,x,y)$, f being the driver in (1). In other words, our subsequent scheme will approximate the solutions to

$$X_t = x + \int_0^t b(s,X_s)\mathrm{d}s + \int_0^t \sigma(s,X_s)\mathrm{d}W_s, \quad Y_t = \mathbb{E}\left[g(X_T) + \int_t^T f(s,X_s,Y_s)\mathrm{d}s \mid \mathcal{F}_t\right], \tag{4}$$

and

$$\partial_t u(t,x) + \mathcal{A}u(t,x) + f(t,x,u(t,x)) = 0 \text{ for } t < T \text{ and } u(T,.) = g(.). \tag{5}$$

One important observation is that, due to the Markov property of the Euler scheme, for every i, there exist measurable deterministic functions $y_i : \mathbb{R}^d \to \mathbb{R}$, such that $Y_i = y_i(X_i)$, almost surely. A second crucial observation is that the value functions $y_i(\cdot)$ are independent of how we initialize the forward component. Our subsequent algorithm takes advantage of this observation. For instance, let X_i^i be a random variable in $\mathbb{R}^d$ with some distribution ν and let X_j^i be the Euler scheme evolution of X_j starting from X_i; it writes

$$X_{j+1}^i = X_j^i + b(t_j,X_j^i)\Delta + \sigma(t_j,X_j^i)(W_{t_{j+1}} - W_{t_j}), \quad j \geq i. \tag{6}$$

This flexibility property w.r.t. the initialization then writes

$$y_i(X_i^i) := \mathbb{E}\left[g(X_N^i) + \sum_{j=i}^{N-1} f_j\left(X_j^i, y_{j+1}(X_{j+1}^i)\right)\Delta \mid X_i^i\right]. \tag{7}$$

Approximating the solution to (3) is actually achieved by approximating the functions $y_i(\cdot)$. In this way, we are directly approximating the solution to the semi-linear PDE (5). In order to control better the truncation error we define a weighted modification of y_i by $y_i^{(q)}(\mathbf{x}) = \dfrac{y_i(\mathbf{x})}{(1+|\mathbf{x}|^2)^{q/2}}$ for a damping exponent $q \geq 0$. For $q = 0$, $y_i^{(q)}$ and y_i coincide. The previous DPE (7) becomes

$$y_i^{(q)}(X_i^i) := \mathbb{E}\left[\frac{g(X_N^i)}{(1+|X_i^i|^2)^{q/2}} + \sum_{j=i}^{N-1} \frac{f_j\left(X_j^i, y_{j+1}^{(q)}(X_{j+1}^i)(1+|X_{j+1}^i|^2)^{q/2}\right)}{(1+|X_i^i|^2)^{q/2}}\Delta \mid X_i^i\right]. \tag{8}$$

The introduction of the polynomial factor $(1+|X_i^i|^2)^{q/2}$ gives higher flexibility in the error analysis, it ensures that $y_i^{(q)}$ decreases faster at infinity, which will provide nicer estimates on the approximation error when dealing with Fourier-basis.

Then we define some proper basis functions $\phi_{\mathbf{k}}$ which satisfy orthogonality properties in $\mathbb{R}^d$ and which span some L^2 space. It turns out that the choice of measure for defining the L^2 space has to coincide with the sampling measure of $X_i^i \sim \nu$. Our strategy for defining such basis functions is to start from trigonometric basis on $[0,1]^d$ and then to apply appropriate changes of variable: later, this transform will allow to easily quantify the approximation error when truncating the basis. Using the notation

$$S_i^{(q)}(x_{i:N}^i) := \frac{g(x_N^i)}{(1+|x_i^i|^2)^{q/2}} + \sum_{j=i}^{N-1} \frac{f_j\left(x_j^i, y_{j+1}^{(q)}(x_{j+1}^i)(1+|x_{j+1}^i|^2)^{q/2}\right)}{(1+|x_i^i|^2)^{q/2}}\Delta,$$

we can rewrite the exact solution as $y_i^{(q)}(\mathbf{x}) = \mathbb{E}\left[S_i^{(q)}\left(X_{i:N}^i\right) | X_i^i = \mathbf{x}\right], \quad \mathbf{x} \in \mathbb{R}^d$. Under mild conditions on f, g and ν, $S_i^{(q)}(X_{i:N}^i)$ is square-integrable, and therefore $y_i^{(q)}$ is in $L^2_\nu(\mathbb{R}^d)$, thus $y_i^{(q)}(\mathbf{x}) = \sum_{\mathbf{k}\in\mathbb{N}^d} \alpha_{i,\mathbf{k}}^{(q)}\phi_{\mathbf{k}}(\mathbf{x})$, for some coefficients $(\alpha_{i,\mathbf{k}}^{(q)} : \mathbf{k} \in \mathbb{N}^d)$. Using the orthonormality property of the

basis functions $\phi_{\mathbf{k}}$'s, $\alpha_{i,\mathbf{k}}^{(q)} = (y_i^{(q)}, \phi_{\mathbf{k}})_{L_v^2(\mathbb{R}^d)} = \mathbb{E}\left[y_i^{(q)}(X_i^i)\phi_{\mathbf{k}}(X_i^i)\right] = \mathbb{E}\left[\mathbb{E}\left[S_i^{(q)}\left(X_{i:N}^i\right)|X_i^i\right]\phi_{\mathbf{k}}(X_i^i)\right] = \mathbb{E}\left[S_i^{(q)}\left(X_{i:N}^i\right)\phi_{\mathbf{k}}(X_i^i)\right]$, thus allowing us to the use of Monte Carlo simulation in order to compute the Fourier coefficients. The resulting Algorithm 1 is shown below.

Algorithm 1. Global Quasi-Regression Multistep-forward Dynamical Programming (GQRMDP) algorithm

Initialization. Set $\bar{y}_N^{(q,M)}(x_N) := \frac{g(x_N)}{(1+|x_N|^2)^{q/2}}$.
Backward iteration for $i = N-1$ **to** $i = 0$**,**

$$\bar{y}_i^{(q,M)}(\cdot) := \sum_{\mathbf{k}\in\Gamma} \bar{\alpha}_{i,\mathbf{k}}^{(q,M)} \phi_{\mathbf{k}}(\cdot), \tag{9}$$

where for all $\mathbf{k} \in \Gamma$,

$$\bar{\alpha}_{i,\mathbf{k}}^{(q,M)} := \frac{1}{M}\sum_{m=1}^{M} S_i^{(q,M)}(X_{i:N}^{i,m})\phi_{\mathbf{k}}(X_i^{i,m}), \tag{10}$$

and

$$S_i^{(q,M)}(x_{i:N}^i) := \frac{g(x_N^i)}{(1+|x_i^i|^2)^{q/2}} \sum_{j=i}^{N-1} \frac{f_j\left(x_j^i, \mathcal{T}_{L^\star}\left(\bar{y}_{j+1}^{(q,M)}(x_{j+1}^i)(1+|x_{j+1}^i|^2)^{q/2}\right)\right)}{(1+|x_i^i|^2)^{q/2}}\Delta.$$

3. Discussion

A implementation on GPUs of the GQRMDP algorithm is proposed. It includes two kernels, one simulates the paths of the forward process and computes the associated responses, the other one computes the regression coefficients $(\alpha_{i,\mathbf{k}}^{(q)}, \mathbf{k} \in \Gamma)$. In the first kernel the initial value of each simulated path of the forward process is stored in a device vector in global memory, it will be read later in the second kernel. In order to minimize the number of memory transactions and therefore maximize performance, all accesses to global memory have been implemented in a coalesced way. The random numbers needed for the path generation of the forward process were generated on the fly (inline generation) taking advantage of the NVIDIA cuRAND library [1] and the generator MRG32k3a proposed by L'Ecuyer in [2]. Therefore, inside this kernel the random number generator is called as needed. Another approach would be the pre-generation of the random numbers in a separate previous kernel, storing them in GPU global memory and reading them back from this device memory in the next kernel. Both alternatives have advantages and drawbacks. In this work we have chosen inline generation having in mind that this option is faster and saves global memory. Besides, register swapping was not observed on the implementation and the quality of the obtained solutions is similar to the accuracy of pure sequential traditional CPU solutions achieved employing more complex random number generators. In the second kernel, in order to compute the regression coefficients, a parallelization not only over the multi-indices $\mathbf{k} \in \Gamma$ but also over the simulations $1 \leq m \leq M$ was proposed. Thus, blocks of threads parallelize the outer for loop $\forall \mathbf{k} \in \Gamma$, whilst the threads inside each block carry out in parallel the inner loop traversing the vectors of the responses and the simulations.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. NVIDIA cuRAND Web Page. Available online: https://developer.nvidia.com/curand (accessed on 5 October 2018).
2. L'Ecuyer, P. Good parameters and implementations for combined multiple recursive random number generators. *Oper. Res.* **1999**, *47*, 159–164.



© 2019 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).