"Financial Mathematics and Supercomputing"

September 26, 2018

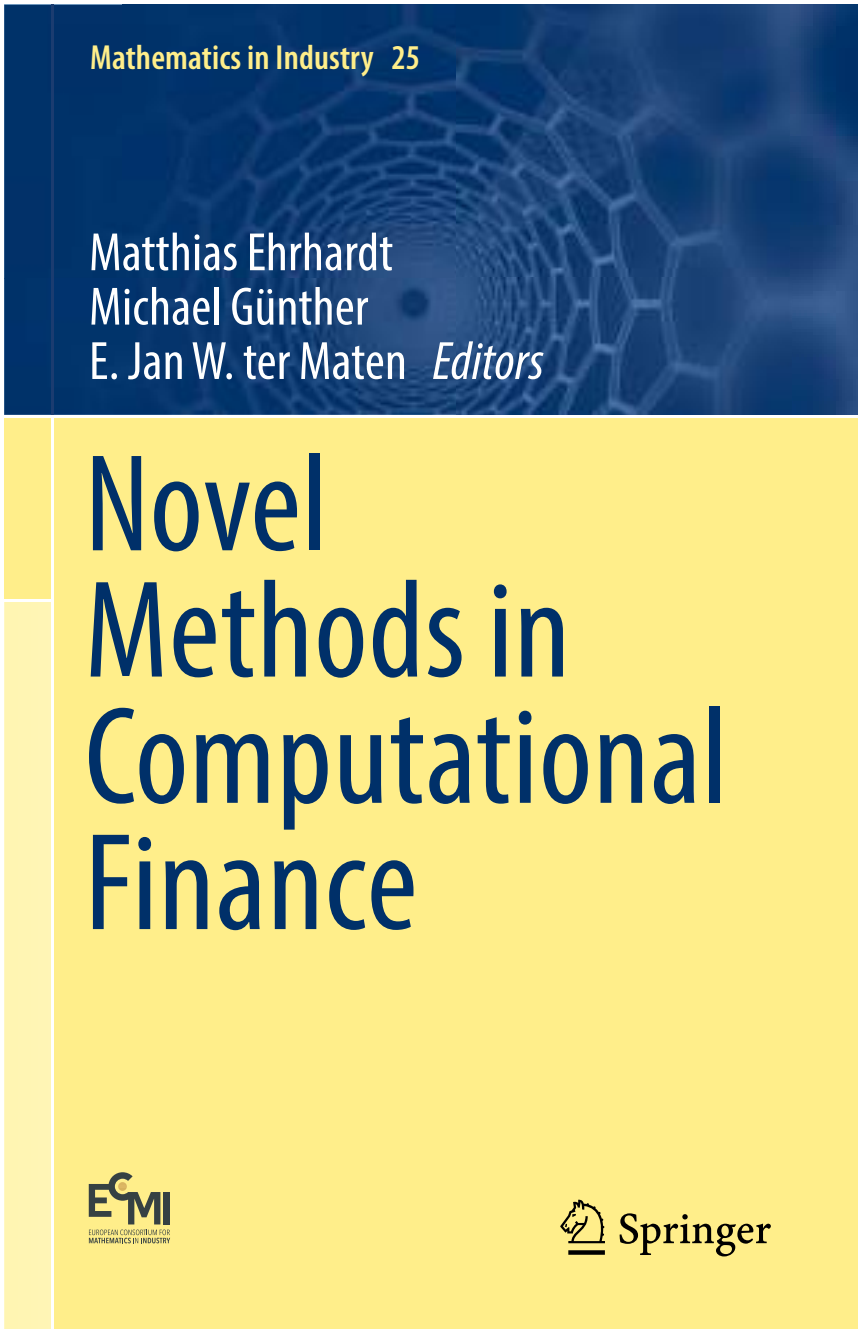
EFFICIENT REGRESSION MONTE CARLO SCHEME FOR SEMILINEAR PDEs AND ITS APPLICATION TO A PROBLEM ARISING IN MATHEMATICAL FINANCE

E. Gobet (L'X), J. G. López-Salas(UDC), P. Turkedjiev (BP), C. Vázquez (UDC)

Contents

Contents

Motivation



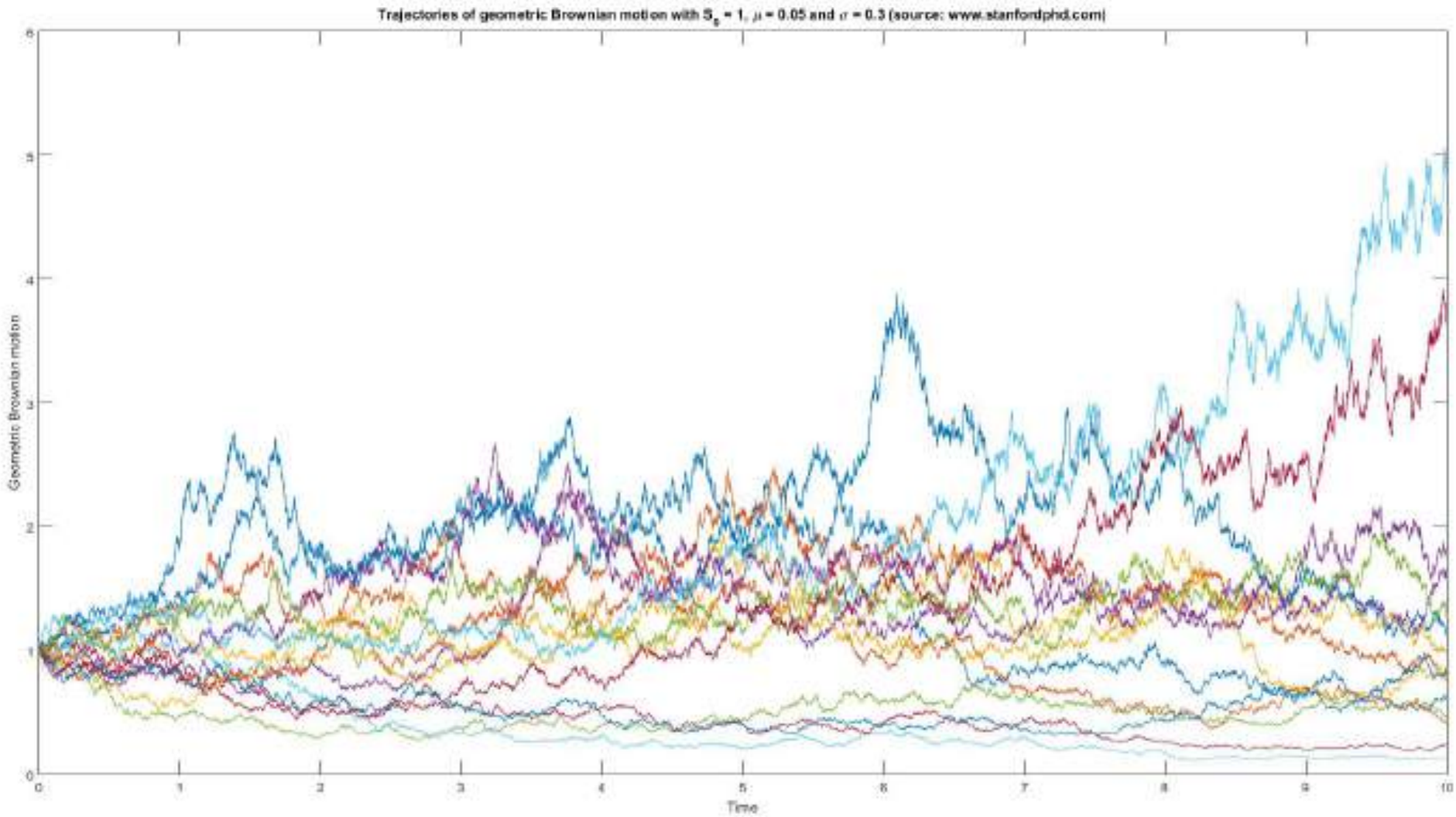
Prof. Peter A. Forsyth (University of Waterloo), January 2017:

"... Consequently, the future of computational finance is bright. We face challenges associated with the development of numerical algorithms for solution of nonlinear PIDEs, possibly using BSDE approaches. However, in order to produce timely results, it will be necessary to exploit the latest hardware advances (e.g. GPUs) and programming environments. This will all have to be coupled with data science methods."

Example

Let us assume that the price of a tradeable risky asset, denoted S, evolves according to the following SDE (Stochastic Differential Equation)

$$dS_t = \mu_t S_t dt + \sigma_t S_t dW_t, \quad S_0 \text{ known.}$$



Example

Consider the situation where a trader wants to:

- Sell a European option with maturity $T > 0$ and payoff $g(S_T)$.
- Hedge the option dynamically with the risky asset S, considering a risk-free interest rate r for both lending and borrowing money.

Denote by:

- V_t the value of the self-financing portfolio at time t.
- π_t the amount of money invested in the risky asset S over time.

In order to ensure the replication of the payoff at maturity, the couple (V, π) should solve the following decoupled forward-backward SDEs:

$$dS_t = \mu S_t dt + \sigma S_t dW_t, \quad S_0 = s_0, \qquad \text{FSDE}$$

$$dV_t = r(V_t - \pi_t)dt + \pi_t \frac{dS_t}{S_t} dS_t = (rV_t + (\mu - r)\pi_t)dt + \sigma \pi_t dW_t, \quad V_T = g(S_T). \qquad \text{BSDE}$$

Find π that leads V to the value $g(S_T)$ at time T.

Contents

BSDEs: Backward SDEs

- Interesting recent concept in financial mathematics:
  - Nonlinear pricing formulas (e.g. CVA), dynamic measures of risk.
  - Portfolio optimization with trading constraints, stochastic optimal control (HJB).
  - Optimal execution of American options.
- BSDEs are directly connected to semilinear PDEs:
  - The solution to these PDEs can be found by solving the corresponding decoupled forward-backward SDE.
- Several advanced probabilistic numerical methods have been developed for FBSDEs:
  - Advanced Monte Carlo methods.
  - Integration methods.
  - Fourier methods.

Design highly efficient Monte Carlo methods.

BSDEs

▹ Notation: Let $(\Omega, \mathscr{F}, (\mathscr{F}_t)_{0\leq t\leq T}, \mathbb{P})$ be a filtered probability space supporting a q-dimensional Brownian motion $(W_t)_{t\geq 0}$, with $(\mathscr{F}_t)_{0\leq t\leq T}$ the natural filtration of the Brownian motion, and T a fixed time horizon.

▹ General BSDE:

$$\boxed{\begin{aligned} -dY_t &= f(t, Y_t, Z_t)dt - Z_t dW_t, \\ Y_T &= \xi, \end{aligned}}$$

where the function f is the so-called driver or generator of the process and the terminal condition ξ is a $\mathscr{F}_T$-measurable random variable.

f and ξ are called *standard parameters* for the BSDE.

BSDEs

▹ Solution: pair of adapted processes (Y, Z) satisfying $\int_0^T |Z_t|^2 dt < \infty$,

$$Y_t = \xi + \int_t^T f(s, Y_s, Z_s) ds - \int_t^T Z_s dW_s, \quad 0 \le t \le T.$$

▹ Result:

N. El Karoui, S. Peng, M. C. Quenez. *Backward stochastic differential equations in finance*, Mathematical Finance, 7(1), 1-71 (1997).

Given a pair of standard parameters (f, ξ) there exists a unique solution $(Y, Z) \in \mathbb{H}_T^2(\mathbb{R}) \times \mathbb{H}_T^2(\mathbb{R})$ to the BSDE.

Decoupled FBSDEs

- Assume $0 \leq t \leq T$, $f(t,w,x,y) = f(t,X_t,Y_t,Z_t)$ and $\xi = g(X_T)$ where X is a forward SDE, i.e.

$$\boxed{\begin{aligned} dX_t &= b(t,X_t)dt + \sigma(t,X_t)dW_t, \quad X_0 = x_0, && \text{FSDE} \\ dY_t &= -f(t,X_t,Y_t,Z_t)dt + Z_t dW_t, \quad Y_T = g(X_T). && \text{BSDE} \end{aligned}}$$

- Let u be the solution to the semilinear parabolic PDE

$$\begin{aligned} &\partial_t u(t,x) + \sum_i b_i(t,x)\partial_{x_i} u(t,x) \\ &\qquad + \frac{1}{2}\sum_{i,j}[\sigma\sigma^*]_{i,j}(t,x)\partial^2_{x_i,x_j} u(t,x) + f\big(t,x,u(t,x),\nabla u(t,x)\sigma(t,x)\big) = 0, \\ &u(T,x) = g(x). \end{aligned}$$

- By Ito's formula $Y_t = u(t,X_t)$ and $Z_t = \nabla u(t,X_t)\sigma(t,X_t)$ solves the BSDE.
- Solving the semilinear PDE and the corresponding decoupled FBSDE result in the same solution:
  - The PDE can be solved by applying numerical discretization techniques.
  - For the FBSDE probabilistic numerical methods are available:
    - Monte Carlo.

Feynman-Kac Representation

Coupling of two stochastic differential equations:

- **FSDE with time ↗:** X unknown,

$$X_t = x_0 + \int_0^t b(s, X_s)ds + \int_0^t \sigma(s, X_s)dW_s.$$

- **BSDE with time ↘:** (Y, Z) unknowns,

$$Y_t = g(X_T) + \int_t^T f(s, X_s, Y_s, Z_s)ds - \int_t^T Z_s dW_s.$$

- The FSDE can be solved without the BSDE.
- The BSDE is coupled to the FSDE.

Feynman-Kac Representation

- FBSDE:

$$X_t = x_0 + \int_0^t b(s, X_s)ds + \int_0^t \sigma(s, X_s)dW_s,$$
$$Y_t = g(X_T) + \int_t^T f(s, X_s, Y_s, Z_s)ds - \int_t^T Z_s dW_s.$$

- Multidimensional unknown: $X \in \mathbb{R}^d$, $Y \in \mathbb{R}$, $Z \in \mathbb{R}^q$.
- Semilinear parabolic PDE: Let $\mathscr{L}$ be the infinitesimal generator of X, $t \in [0, T)$,

$$\partial_t u(t,x) + \mathscr{L}u(t,x) + f\big(t, x, u(t,x), \nabla u(t,x)\sigma(t,x)\big) = 0,$$
$$u(T,x) = g(x).$$

- Then $Y_t = u(t, X_t)$ and $Z_t = \nabla u(t, X_t)\sigma(t, X_t)$ solves the BSDE:

$$Y_t = g(X_T) + \int_t^T f(s, X_s, Y_s, Z_s)ds - \int_t^T Z_s dW_s.$$

- Corollary: $\mathbf{Y_t = u(t,x)} = \mathbb{E}\left(\mathbf{g(X_T)} + \int_{\mathbf{t}}^{\mathbf{T}} \mathbf{f(s, X_s, Y_s, Z_s)ds} \;\middle|\; \mathbf{X_t = x}\right)$,
  $\Longrightarrow$ Probabilistic solution to semilinear PDEs.

Approximation/simulation of the BSDE

$$Y_t = g(X_T) + \int_t^T f(s, X_s, Y_s, Z_s)ds - \int_t^T Z_s dW_s.$$

Two stages:

1. Time-discretization.
   - Numerous works under rather general settings.
2. Solving the dynamic programming equation.
   - Nested conditional expectations, few works.

Time-discretization of $Y_t = g(X_T) + \int_t^T f(s, X_s, Y_s, Z_s)ds - \int_t^T Z_s dW_s$

Discretization along equidistant time grid $\pi := \{0 = t_0, \ldots, t_N = T\}$:

- $(i+1)$-th time-step is $\Delta_i = t_{i+1} - t_i = T/N = \Delta_t$.
- Related Brownian motion increments $\Delta W_i := W_{t_{i+1}} - W_{t_i}$, $\Delta W_i \sim \mathcal{N}\left(0, \sqrt{\Delta_t}\right)$.

Discretization:

1. Euler-Maruyama scheme for X ($X_{t_i} = X_i$):

$$\begin{cases} X_0 &= x_0, \\ X_{i+1} &= X_i + b(t_i, X_i)\Delta_i + \sigma(t_i, X_i)\Delta W_i, \quad i = 0, \ldots, N-1. \end{cases}$$

2. Explicit Euler scheme for Y ($Y_{t_i} = Y_i$, $Z_{t_i} = Z_i$):
   - We start with $Y_i = Y_{i+1} + \int_{t_i}^{t_{i+1}} f(s, X_s, Y_s, Z_s)ds - \int_{t_i}^{t_{i+1}} Z_s dW_s$.
   - Take conditional expectations $\mathbb{E}[\cdot | \mathscr{F}_{t_i}] = \mathbb{E}_i[\cdot]$.
   - $Y_i = \mathbb{E}\left(Y_{i+1} + \int_{t_i}^{t_{i+1}} f(s, X_s, Y_s, Z_s)ds \,\middle|\, \mathscr{F}_{t_i}\right) \approx \mathbb{E}_i\left(Y_{i+1} + f(t_i, X_i, Y_{i+1}, Z_i)\Delta_i\right)$, $i = N-1, \ldots, 0$.
3. Explicit Euler scheme for Z:
   - $\times \Delta W_i$, take conditional expectations $\mathbb{E}_i[\cdot]$, approximate integral.
   - $Z_i \Delta_i \approx \mathbb{E}_i\left(\left[Y_{i+1} + \int_{t_i}^{t_{i+1}} f(s, X_s, Y_s, Z_s)ds\right]\Delta W_i^\top\right) \approx \mathbb{E}_i(Y_{i+1}\Delta W_i^\top), i = N-1, \ldots, 0.$

Dynamic Programming equations

- One-step forward Dynamic Programming (ODP) equation:

$$\begin{aligned} Y_i &= \mathbb{E}_i(Y_{i+1} + f(t_i, X_i, Y_{i+1}, Z_i)\Delta_i), \quad i = N-1, \ldots, 0, \quad Y_N = g(X_N), \\ \Delta_i Z_i &= \mathbb{E}_i(Y_{i+1}\Delta W_i^\top), \quad i = N-1, \ldots, 0. \end{aligned}$$

- Multi-step forward Dynamic Programming (MDP) equation:

$$\begin{aligned} Y_i &= \mathbb{E}_i\left(g(X_N) + \sum_{j=i}^{N-1} f(t_j, X_j, Y_{j+1}, Z_j)\Delta_j\right), \quad i = N-1, \ldots, 0, \quad Y_N = g(X_N), \\ \Delta_i Z_i &= \mathbb{E}_i\left(\left[g(X_N) + \sum_{j=i+1}^{N-1} f(t_j, X_j, Y_{j+1}, Z_j)\Delta_j\right]\Delta W_i^\top\right), \quad i = N-1, \ldots, 0. \end{aligned}$$

- Without extra approximation, ODP $\Longleftrightarrow$ MDP.
- Differences occur when conditional expectations are approximated: MDP $>$ ODP.

Usual Regression Monte Carlo method

- Markovian representations: there exist measurable deterministic functions y_i and z_i such that $Y_i = y_i(X_i)$ and $Z_i = z_i(X_i)$ a.s.
- Computations of y and z on approximation spaces $\mathscr{L}_i^Y$, $\mathscr{L}_i^Z$ (finite dimensional vector spaces: global/local approximations, Fourier basis, wavelets, ...).
  $\Phi := \mathrm{Span}(\Phi_1, \ldots, \Phi_K)$, of finite dimension K.
- $\mathscr{M}$ independent learning samples: at time t_i, $[(X_j^{i,m})_{0\le j\le N}, \Delta W_i^{i,m}]_{1\le m\le \mathscr{M}}$.
- **Algorithm**:
  - Initialization: Set $y_N^{\mathscr{L},\mathscr{M}}(\cdot) = g(\cdot)$.
  - Backward iteration for $i = N-1$ to $i = 0$. Solve the empirical least squares problems:

$$z_i^{\mathscr{L},\mathscr{M}} = \arg\min_{\alpha_z\in\mathbb{R}^K} \sum_{m=1}^{\mathscr{M}} \Big| \Big[g(X_N^{i,m}) + \sum_{j=i+1}^{N-1} f(t_j, X_j^{i,m}, y_{j+1}^{\mathscr{F},\mathscr{M}}(X_{j+1}^{i,m}), z_j^{\mathscr{F},\mathscr{M}}(X_j^{i,m}))\Delta_j \Big] \frac{\Delta W_i^{i,m}}{\Delta_i} - \alpha_z \cdot \phi(X_j^{i,m}) \Big|^2,$$

$$y_i^{\mathscr{L},\mathscr{M}} = \arg\min_{\alpha_y\in\mathbb{R}^K} \sum_{m=1}^{\mathscr{M}} \Big| g(X_N^{i,m}) + \sum_{j=i}^{N-1} f(t_j, X_j^{i,m}, y_{j+1}^{\mathscr{F},\mathscr{M}}(X_{j+1}^{i,m}), z_j^{\mathscr{F},\mathscr{M}}(X_j^{i,m}))\Delta_j - \alpha_y \cdot \phi(X_j^{i,m}) \Big|^2.$$

Least Squares MDP (LSMDP) algorithm

E. Gobet and P. Turkedjiev. *Linear regression MDP scheme for discrete backward stochastic differential equations under general conditions*, Mathematics Of Computation, 85(299), 1359-1391 (2016).

- Solutions approximated on a $\mathscr{K}$-dimensional basis of functions at each time time point t_i, $0 \leq i \leq N$.
- Coefficients of the basis functions computed at every t_i with $\mathscr{M}$ simulations of a Markov chain which approximates X in $[t_i, T]$.
- Memory constraints. We have to store:
  - $\mathscr{K} \times N$ coefficients of the basis functions.
  - $\mathscr{M} \times N$ simulations to compute the coefficients.
- Problems in high dimensions:
  - Dimension of the basis functions $\mathscr{K} = const \times N^{\alpha d}$, so $\mathscr{K}$ increases geometrically with d.
  - Local statistical error is proportional to $N\mathscr{K}/\mathscr{M}$, so $\mathscr{M} = const \times \mathscr{K} N^2$ to ensure convergence $O(N^{-1})$.
- Simulations are the main constraint on the memory consumption.

Currently available algorithms rarely handle the case of dimension greater than 8.

Objectives

Drastically rework the LSMDP algorithm to:

1. Minimize the exposure to the memory due to the storage of simulations.

   $\Longrightarrow$ This will allow computation in larger dimension d.

2. Exploit the parallelism of the algorithm by using GPUs.

   $\Longrightarrow$ Optimize computational time.

3. Support the algorithm by a rigorous theoretical error analysis.

Stratification

Two objectives:

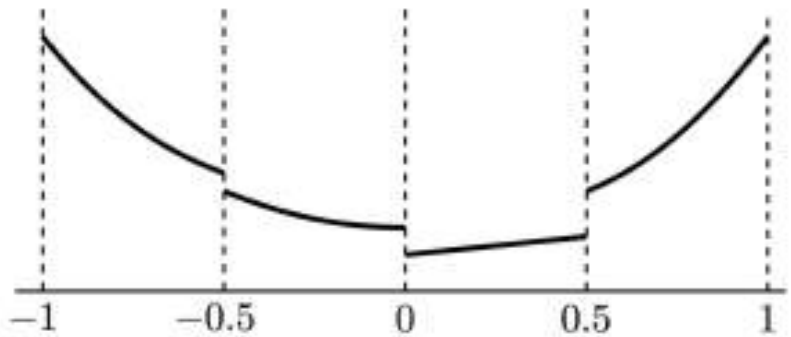
- Relaxing the requirement on $\mathscr{M}$.
- Allow parallel computations.

First choice: local approximations

- Partition of the state space $\mathbb{R}^d$ in strata $\rightarrow$ finite number of disjoint sets $\mathscr{H}_k$.

On each hypercube $\mathscr{H}_k$, (local) polynomial:

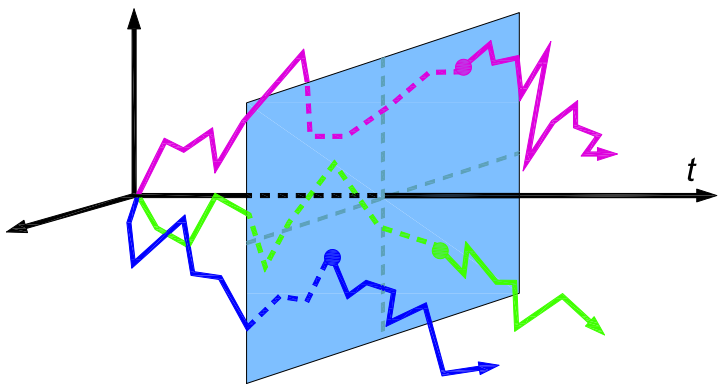
- **LP0**: piecewise constant approximation.
- **LP1**: linear approximation.



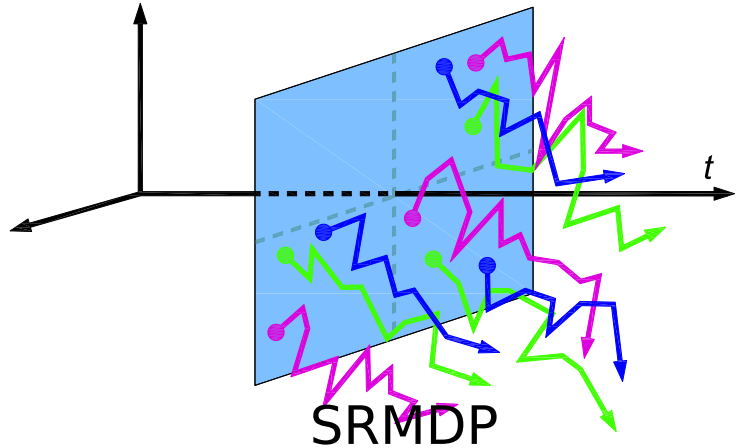
To get a statistical error of order N^{-1}, only N^2 simulations in $\mathscr{H}_k$ are required.

Stratification

Second choice: stratified simulations and regressions



LSMDP



SRMDP

- Paths of X were simulated from a fixed point at time 0.
- One must store all the simulated paths.
- For each hypercube $\mathscr{H}_k$ we simulate M paths of X in $[t_i, T]$ starting from i.i.d random variables valued in $\mathscr{H}_k$.
- We only need to generate samples on one hypercube at a time.
- The computations are performed in parallel across the hypercubes, which allows us for massive parallelization.

$$z_i^{\mathscr{L},\mathscr{M}}\big|_{\mathscr{H}_k} = \arg\min_{\alpha_z \in \mathbb{R}^K} \sum_{m=1}^{M} \Big| \Big[g(X_N^{i,k,m}) + \sum_{j=i+1}^{N-1} f(t_j, X_j^{i,k,m}, y_{j+1}^{\mathscr{F},M}(X_{j+1}^{i,k,m}), z_j^{\mathscr{F},M}(X_j^{i,k,m}))\Delta_j \Big] \frac{\Delta W_i^{i,k,m}}{\Delta_i} - \alpha_z \cdot \phi(X_i^{i,k,m}) \Big|^2,$$

$$y_i^{\mathscr{L},M}\big|_{\mathscr{H}_k} = \arg\min_{\alpha_y \in \mathbb{R}^K} \sum_{m=1}^{M} \Big| g(X_N^{i,k,m}) + \sum_{j=i}^{N-1} f(t_j, X_j^{i,k,m}, y_{j+1}^{\mathscr{F},M}(X_{j+1}^{i,k,m}), z_j^{\mathscr{F},M}(X_j^{i,k,m}))\Delta_j - \alpha_y \cdot \phi(X_i^{i,k,m}) \Big|^2.$$

Conditional logistic distribution

▹ Select a distribution ν whose restriction to the hypercubes $\mathscr{H}_k$, ν_k, can be explicitly computed.

▹ Simulate i.i.d. copies of $X_i^{(i,\nu_k)}$ directly in $\mathscr{H}_k$.

▹ Use the resulting paths to estimate $y_k(\cdot)|_{\mathscr{H}_k}$.

- Let $\mu > 0$. The distribution of $X_i^{(i,\nu_k)}$ is given by $\nu_k(\mathrm{d}x) = \dfrac{\mathbf{1}_{\mathscr{H}_k}(x)\,\nu(\mathrm{d}x)}{\nu(\mathscr{H}_k)}$,

$$\nu(\mathrm{d}x) = p_{logis}^{(\mu)}(x)dx,\ p_{logis}^{(\mu)}(x) := \prod_{l=1}^{d} \frac{\mu \mathrm{e}^{-\mu x_l}}{(1+\mathrm{e}^{-\mu x_l})^2},\ x = (x_1,\ldots,x_d) \in \mathbb{R}^d.$$

- Generation of $X_i^{(i,\nu_k)}$: Draw d independent random variables $(U_1,\ldots,U_d)$ which are uniformly distributed on $[0,1]$ and compute

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

where $F_\nu(x) := \int_{-\infty}^{x} \nu(\mathrm{d}x') = 1/(1+\mathrm{e}^{-\mu x})$ and

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

▹ Logistic distribution is crucial in the error analysis of the SRMDP (Stratified Regression MDP) algorithm.

SRMDP (Stratified Regression MDP) algorithm

N, K, M: **int** ; // *Number of time steps, hypercubes and simulations*
T, Δ_t: **double** ; // *Terminal time and time step*
α: **vector of double** ; // *Set of coefficients* $(\alpha_i^k)_{0 \leq i \leq N-1}$
for i **from** $N-1$ **to** 0 **do**
  for k **from** 1 **to** K **do**
    // *Sampling Euler scheme paths*
    for m **from** 1 **to** M **do**
      Simulate $\underline{\mathbf{X}}_i^{(i,k,m)}$

    // *Compute the responses*
    for m **from** 1 **to** M **do**

$$S_{Y,i}^{(M)}(X_i^{k,m}) = g(X_N^m) + \sum_{j=i}^{N-1} f_j(X_j^m, Y_{j+1}(X_{j+1}^m))\Delta_t$$

    // *Compute the regression coefficients*
    `compute_`α_i^k`()` // *Depending on the selected approximation space*
  end for k
end for i

Explicit solutions to OLS problems

- **LP0:**

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

- **LP1:**
  - Let A be the $\mathbb{R}^M \otimes \mathbb{R}^{d+1}$ matrix, $A[m,j] = 1\mathbf{1}_{\{0\}}(j) + X_{i,j}^{i,k,m}\mathbf{1}_{\{0\}^c}(j)$, where $X_{i,j}^{i,k,m}$ is the j-th component of $\underline{\mathbf{X}}_i^{(i,k,m)}$.
  - Let S be the $\mathbb{R}^M$ vector given by $S[m] = S_{Y,i}^{(M)}(\underline{\mathbf{X}}_i^{(i,k,m)})$.
  - Perform QR factorization $A = QR$ (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) using the Householder reflections method, [Alg 5.3.2] of

    G. H. Golub and C. F. Van Loan. *Matrix computations*, Johns Hopkins University Press, Baltimore, MD, third ed., 1996.

    Crucial for GPU computing: We do not need to store matrix Q.
  - Compute the coefficients $\alpha = (\alpha_0, \ldots, \alpha_d) \in \mathbb{R}^{d+1}$ using QR factorization and backward-substitution method, $R\alpha = Q^T S$.
  - For any vector $x = (x_1, \ldots, x_d)$ in $\mathscr{H}_k$,

$$y_i^{(M)}(x)|_{\mathscr{H}_k} = \mathscr{T}_{C_y}\left(\alpha_0 + \sum_{j=1}^{d} \alpha_j x_j\right).$$

Graphics Processing Units (GPUs)

Exterior view of a NVIDIA GeForce GTX TITAN Black graphics card.

Dimensions:

LENGTH: 26,67 cm.

WIDTH: Approx. 2 cm (Double width).

HEIGHT: 11.16 cm.

Graphics Processing Units (GPUs)

GPUs have a double advantage

Execution parallelism

- A GPU is based in a **Many-core** architecture with a huge number of cores:

  - Up to 2880 in Nvidia Kepler architecture (for example, Nvidia Titan Black).

- A GPU can execute many tasks (execution threads) at a time.

- SIMD **(Single Instruction Multiple Data) paradigm**.

- The same instruction is executed asynchronously by all the computing cores, but over different data.

  We only need to code the set of instructions once, and this computing kernel is executed by all the cores.

Fast memory access

- Much higher memory bandwidth than a CPU (up to 336 GB/s) **...but...**

- Launching threads is not for free: to benefit from this, the programmer is responsible for doing a good job when accessing memory.

Parallelization

- Time loop is not parallelizable.
- The outer hypercubes loop and the inner simulations loop are trivially parallel.

- GPU parallelization.
  - For **LP0** it might be worth to parallelize both loops, blocks of threads parallelize the outer hypercubes loop, while threads inside each block parallelize the inner simulations loop. Reduce operations will be performed on shared memory.
  - For **LP1** it is better to parallelize only over the hypercubes, since some parts of QR factorization and backwards substitution routines are pure sequential.
  - Generation of random numbers using NVIDIA cuRAND library. In order to have a lower global memory footprint, random numbers are generated inside the kernel, although this may cause register pressure.
  - Global memory coalesced access.

- Multicore CPU parallelization.
  - OpenMP.
  - ICC.
  - Intel Math Kernel Library (MKL).
  - Vectorization.

Theorem (Theoretical error for the SRMDP algorithm)

Error estimates for **LP0** and **LP1** spaces are,

$$\mathbb{E}\left[\int_{\mathbb{R}^d}\left|y_i^{\mathscr{F},M}(x)-y_i(x)\right|^2\nu(dx)\right]\leq C\mathscr{E}(i)+C\sum_{j=i}^{N-1}\mathscr{E}(j)\Delta_j,$$

$$\sum_{j=i}^{N-1}\mathbb{E}\left[\int_{\mathbb{R}^d}\left|z_j^{\mathscr{F},M}(x)-z_j(x)\right|^2\nu(dx)\right]\Delta_j\leq C\sum_{j=i}^{N-1}\mathscr{E}(j)\Delta_j,$$

where

$$\mathscr{E}(j):=\sum_k\nu(\mathscr{H}_k)\inf_{\varphi_y\in\mathscr{L}_{Y,k}}\int_{\mathscr{H}_k}\left|\varphi_y(x)-y_j(x)\right|^2\nu_k(dx)$$
$$+\sum_k\nu(\mathscr{H}_k)\inf_{\varphi_z\in\mathscr{L}_{Z,k}}\int_{\mathscr{H}_k}\left|\varphi_z(x)-z_j(x)\right|^2\nu_k(dx)+\frac{\log(M)}{\Delta_j M}.$$

considering some explicit constant C.

Numerical experiments

- We use the Brownian motion model $X = W(d = q)$ and $T = 1$.
- We introduce the function $w(t,x) = \exp(t + \sum_{k=1}^{q} x_k)$.
- BSDE data:
  - $g(x) = w(T,x)(1 + w(T,x))^{-1}$.
  - $f(t,x,y,z) = \left(\sum_{k=1}^{q} z_k\right)\left(y - \frac{2+q}{2q}\right)$, where $z = (z_1, \ldots, z_q)$.
- PDE:

$$\partial_t u(t,x) + \frac{1}{2}\sum_{i,j}[\sigma\sigma^*]_{i,j}(t,x)\partial^2_{x_i,x_j} u(t,x) + f\big(t,x,u(t,x),\nabla u(t,x)\sigma(t,x)\big) = 0,$$

$$u(T,x) = g(x).$$

- Probabilistic formulation:

$$dX_t = dW_t,$$
$$dY_t = -f(t,X_t,Y_t,Z_t)dt + Z_t dW_t, \quad Y_T = g(X_T).$$

- Explicit solutions of this BSDE:

$$y_i(x) = w(t_i,x)(1 + w(t_i,x))^{-1}, z_{k,i}(x) = w(t_i,x)(1 + w(t_i,x))^{-2}.$$

Numerical experiments, Mean square error, MSE

- Hardware:
  - GPU GeForce GTX TITAN Black with 6 Gbytes of global memory.
  - Two multicore Intel Xeon CPUs E5-2620 v2 clocked at 2.10 GHz (6 cores per socket) with 62 GBytes of RAM.
- Logistic distribution parameterized by $\mu = 1$.
- Domain $[-6.5, 6.5]^d$ stratified with uniform hypercubes.
- We consider 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.

Examples with the approximation with **LP0** local polynomials

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: $d = 4$, #C=$\lfloor 4\sqrt{N} \rfloor$, $M = N^2$.

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: $d = 6$, #C=$\lfloor 2\sqrt{N} \rfloor$, $M = N^2$.

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: $d = 11$, #C=$\lfloor \sqrt{N} \rfloor$, $M = N^2$.

Examples with the approximation with **LP1** local polynomials

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: $d = 12$.

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: $d = 13$.

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: $d = 14$.

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: $d = 15, \ldots, 19$, $\Delta_t = 0.2$, #C $= 2$.

Comparison between **LP0** and **LP1** local polynomials

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: **LP0**, $d = 6$, #C=$\lfloor 2\sqrt{N} \rfloor$, $M = N^2$.

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

Article

E. Gobet, J. G. López Salas, P. Turkedjiev and C. Vázquez. *Stratified Regression Monte-Carlo Scheme for Semilinear PDEs and BSDEs with Large Scale Parallelization on GPUs*, SIAM Journal on Scientific Computing, 38(6), C652-C677 (2016).

THANKS FOR YOUR ATTENTION