Deep Learning Numerical Methods on High-Dimensional PDEs -- The Deep BSDE method

Presenter: Chenyang Fang
10/31/19

Papers of Interest

- Bing, Yu , Xiaojing Xing , Agus Sudjianto, Deep-learning based numerical BSDE method for barrier options, *arxiv,* 2017
- Weinan E , Jiequn Han , and Arnulf Jentzen, Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations -- *Communications in Mathematics and Statistics* 2017
- Côme Huré, Huyên Pham, Xavier Warin, Some machine learning schemes for high-dimensional nonlinear PDEs, *arxiv*, 2019

Wharton
University of Pennsylvania

Outline

- Introduction
  - Problems of High Dimensional Nonlinear PDEs
  - Solution
  - Background
- Neural Network & Deep Learning Schemes
  - Neural Network as Approximation
  - Deep learning-based schemes for semi-linear PDEs
    - DBSP
    - DBDP1 & DBDP2
    - RDBDP
- Convergence Analysis
  - DBDP1
  - DBDP2
- Applications

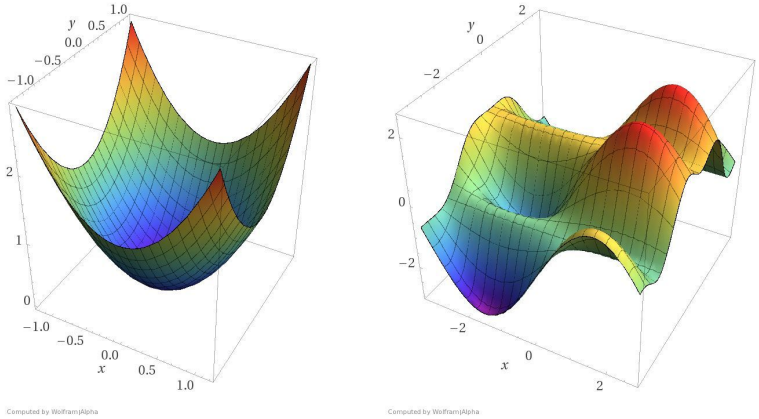
Wharton
University of Pennsylvania

Problem of High Dimensional PDEs

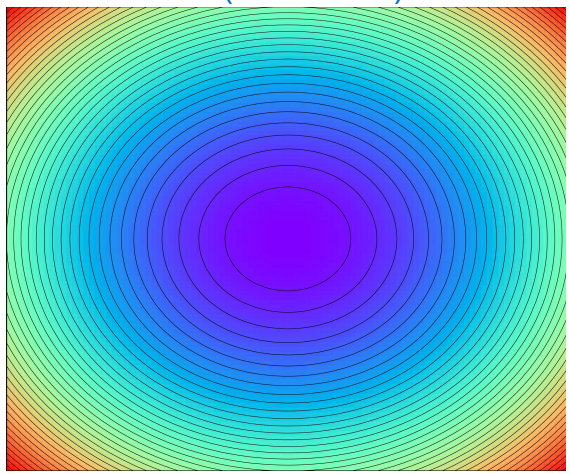
Part 1

Problems of High Dimensional Nonlinear PDE

- "curse of dimensionality"
  - The BSDE (Backward Stochastic Differential Equation) approach has recently been developed to tackle this. Using the time discretization scheme, some effective algorithms based on regression manage to solve nonlinear PDEs in dimension above 4. However this approach is still not implementable in dimension above 6 or 7: the number of basis functions used for the regression still explodes with the dimension(Huré et al.)



High Dimensional Gradient Descent



High Dimensional Hamilton-Jacobi PDEs

Wharton
University of Pennsylvania

An Elegant, Practical Solution

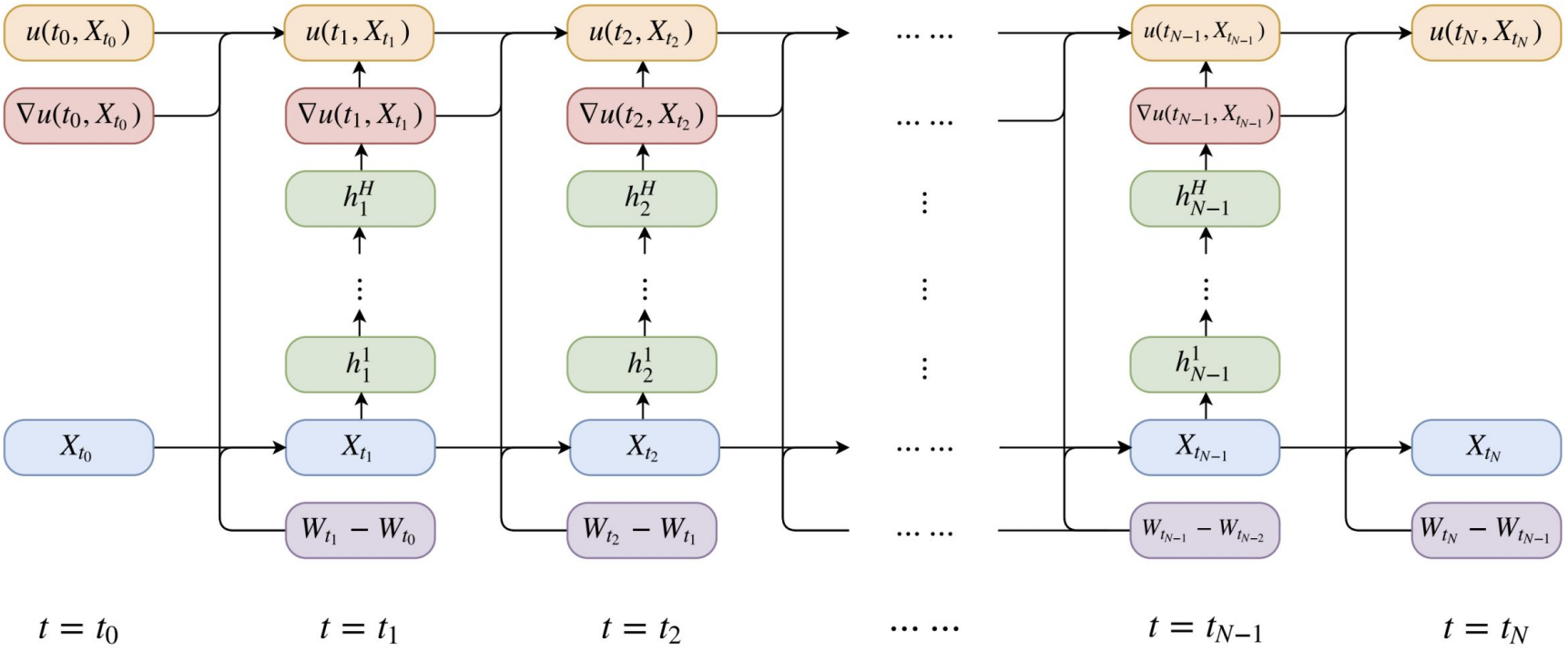


Fig. 1: Graphical illustration of the neural network diagram with H hidden layers and N time steps.

Details will be introduced

Figure adapted from *Solving high-dimensional partial differential equations using deep learning*, Han et al. PNAS 2018

Wharton
University of Pennsylvania

Neural Network & Deep Learning Based Schemes for Semi-linear PDEs

Part 2

High Dimensional Semilinear PDE

$$\frac{\partial u}{\partial t}(t,x) + \frac{1}{2}\mathrm{Tr}\Big(\sigma\sigma^{\mathrm{T}}(t,x)(\mathrm{Hess}_x u)(t,x)\Big) + \nabla u(t,x)\cdot\mu(t,x) + f\big(t,x,u(t,x),\sigma^{\mathrm{T}}(t,x)\nabla u(t,x)\big) = 0$$

- Terminal Condition: u(T, x) = g(x)
- To fix ideas, we are interested in the solution at t = 0, x = ξ for some vector $\xi \in \mathbb{R}^d$

Wharton
University of Pennsylvania

Linear Quadratic Gaussian Control example for High Dimensional Semilinear PDE

- Consider a stochastic optimal control problem in R^d

$$dX_t = 2\sqrt{\lambda}\, m_t\, dt + \sqrt{2}\, dW_t$$

$$J(\{m_t\}_{0\le t\le T}) = \mathbb{E}\big[\int_0^T \|m_t\|_2^2\, dt + g(X_T)\big]$$

- the optimal control is

$$m_t^* = \frac{\nabla u(t,x)}{\sqrt{2\lambda}}$$

$$u(t,x) = -\frac{1}{\lambda}\ln\Big(\mathbb{E}\Big[\exp\Big(-\lambda g(x+\sqrt{2}W_{T-t})\Big)\Big]\Big)$$

- via the Hamilton-Jacobi-Bellman equation,

$$\frac{\partial u}{\partial t}(t,x) + \Delta u(t,x) - \lambda\|\nabla u(t,x)\|_2^2 = 0$$

Wharton
University of Pennsylvania

Two Solutions

(1) Estimate the solution and its gradient by a neural network.

(2) Approximate the solution by a neural network while its gradient is estimated directly with some numerical differentiation techniques.

I will be focusing on 1 due to time

Wharton
University of Pennsylvania

Backward SDE

- Find a solution to this backward stochastic differential equation :

$$y(t) = (ry(t) + az(t)) * dt + z(t)dW_t$$

- with the terminal condition

$$y(T) = \xi$$

- with ξ is a square integrable, F measurable random variable in a filtered probability space and W is a one dimensional brownian motion. in addition r and a are constant

Wharton
University of Pennsylvania

Nonlinear Feynman-Kac Formula

- Markovian Backward SDEs give a nonlinear Feynman-Kac representation of some nonlinear parabolic PDEs (*Backward Stochastic Differential Equations in Finance, N. et al. Mathematical Finance Vol 7. 1997)*
- solution to the BSDE,

$$Y_t = g(\mathcal{X}_T) + \int_t^T f(s, \mathcal{X}_s, Y_s, Z_s)\,\mathrm{d}s - \int_t^T Z_s^\intercal\,\mathrm{d}W_s, \quad 0 \le t \le T$$

- related to the solution u of the form,

$$\begin{cases} \partial_t u + \mathcal{L}u + f(.,.,u,\sigma^\intercal D_x u) = 0, & \text{on } [0,T) \times \mathbb{R}^d, \\ u(T,.) = g, & \text{on } \mathbb{R}^d, \end{cases}$$

- via

$$Y_t = u(t, \mathcal{X}_t), \quad 0 \le t \le T$$

$$Z_t = \sigma^\intercal(t, \mathcal{X}_t) D_x u(t, \mathcal{X}_t), \quad 0 \le t \le T$$

Wharton
University of Pennsylvania

Connection Between PDEs and BSDEs

- Consider the following BSDE,

$$\begin{cases} X_t = \xi + \int_0^t \mu(s, X_s)\, ds + \int_0^t \sigma(s, X_s)\, dW_s, \\ Y_t = g(X_T) + \int_t^T f(s, X_s, Y_s, Z_s)\, ds - \int_t^T (Z_s)^{\mathrm{T}}\, dW_s, \end{cases}$$

- The solution is an adaptive process {(Xt, Yt, Zt)}
- Under suitable regularity assumptions, the BSDE is well-posed and related to the PDE in the sense that for all t ∈ [0, T] it holds a.s. that

$$Y_t = u(t, X_t) \qquad \text{and} \qquad Z_t = \sigma^{\mathrm{T}}(t, X_t)\, \nabla u(t, X_t)$$

- the solution of PDE satisfies the following SDE

$$\begin{aligned} & u(t, X_t) - u(0, X_0) \\ = & -\int_0^t f(s, X_s, u(s, X_s), \sigma^{\mathrm{T}}(s, X_s)\, \nabla u(s, X_s))\, ds \\ & + \int_0^t [\nabla u(s, X_s)]^{\mathrm{T}}\, \sigma(s, X_s)\, dW_s. \end{aligned}$$

Wharton
University of Pennsylvania

Deep Learning Scheme

- Feedforward Neural Network
  - define linear feedforward neural network

$$x \in \mathbb{R}^d \longmapsto A_L \circ \varrho \circ A_{L-1} \circ \ldots \circ \varrho \circ A_1(x) \in \mathbb{R}^{d_1}$$

Here A_ℓ, $\ell = 1, \ldots, L$ are affine transformations: A_1 maps from $\mathbb{R}^d$ to $\mathbb{R}^m$, $A_2, \ldots, A_{L-1}$ map from $\mathbb{R}^m$ to $\mathbb{R}^m$, and A_L maps from $\mathbb{R}^m$ to $\mathbb{R}^{d_1}$, represented by

$$A_\ell(x) = \mathcal{W}_\ell x + \beta_\ell,$$

Wharton
University of Pennsylvania

Time Discretization

- consider the simple Euler scheme of the BSDE, with a partition of the time interval [0, T], 0 = t0 < t1 < . . . < tN = T

$$X_{t_{n+1}} - X_{t_n} \approx \mu(t_n, X_{t_n})\,\Delta t_n + \sigma(t_n, X_{t_n})\,\Delta W_n$$

$$\begin{aligned} & u(t_{n+1}, X_{t_{n+1}}) - u(t_n, X_{t_n}) \\ \approx & -f\big(t_n, X_{t_n}, u(t_n, X_{t_n}), \sigma^{\mathrm{T}}(t_n, X_{t_n})\,\nabla u(t_n, X_{t_n})\big)\,\Delta t_n \\ & + [\nabla u(t_n, X_{t_n})]^{\mathrm{T}}\,\sigma(t_n, X_{t_n})\,\Delta W_n, \end{aligned}$$

$$\Delta t_n = t_{n+1} - t_n, \quad \Delta W_n = W_{t_{n+1}} - W_{t_n}$$

Wharton
University of Pennsylvania

Architecture

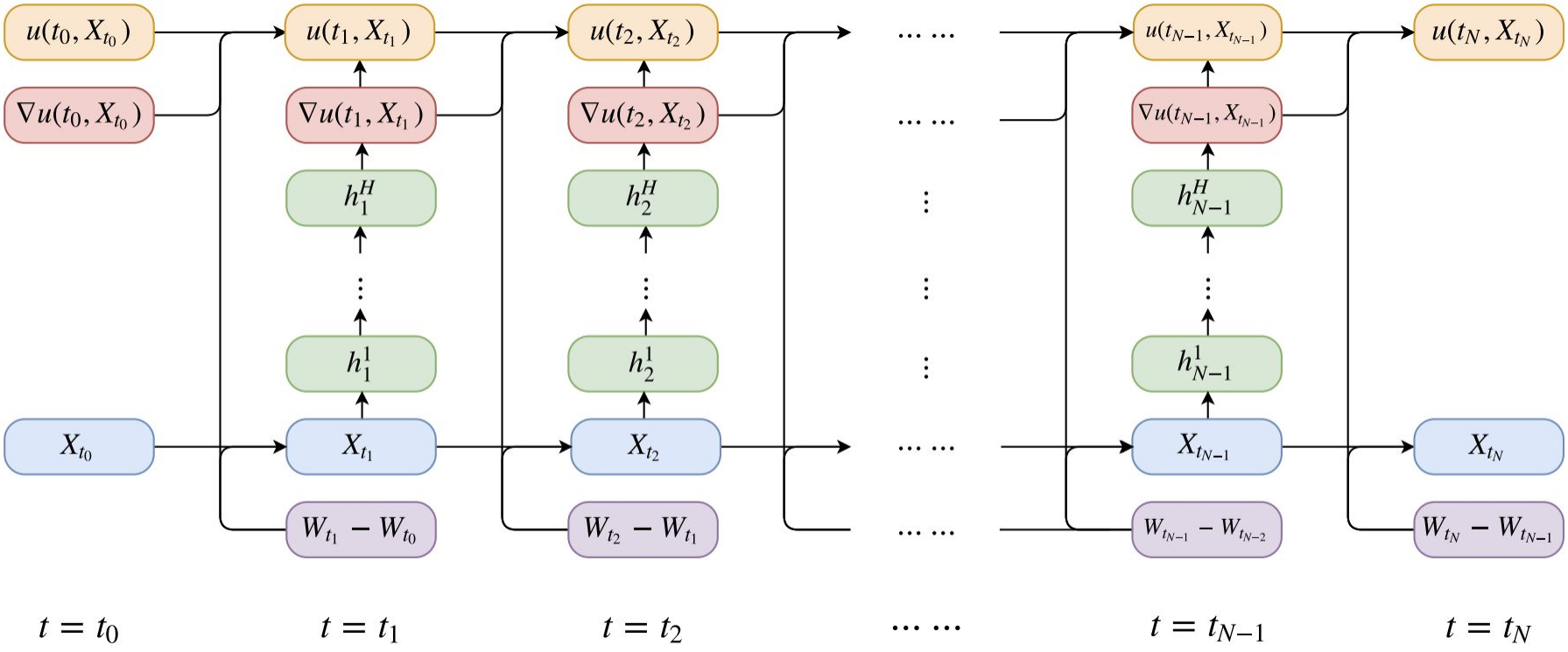


Figure adapted from *Solving high-dimensional partial differential equations using deep learning*, Han et al. PNAS 2018

- Each column corresponds to a subnetwork at time t = tn. The whole network has (H + 1)(N − 1) layers in total that involve free parameters to be optimized simultaneously

Wharton
University of Pennsylvania

Approximation

- The most important step by Han et al. is to approximate

$$x \mapsto \sigma^{\mathrm{T}}(t,x)\,\nabla u(t,x)$$

  at each discretized time step tn by a feedforward neural network

$$\begin{aligned}\sigma^{\mathrm{T}}(t_n, X_{t_n})\,\nabla u(t_n, X_{t_n}) &= (\sigma^{\mathrm{T}}\nabla u)(t_n, X_{t_n}) \\ &\approx (\sigma^{\mathrm{T}}\nabla u)(t_n, X_{t_n}|\theta_n),\end{aligned}$$

- “We can stack all the subnetworks together to form a deep neural network (DNN) as a whole, based on the time discretization”

Wharton
University of Pennsylvania

DBSP (this part is adapted from Han et al.'s presentation on the paper)

- According to Han et al. , the error in the matching of given terminal condition defines the expected loss function,
- This network takes the paths {Xtn }0≤n≤N and {Wtn }0≤n≤N as the input data and gives the final output, denoted by uˆ({Xtn }0≤n≤N , {Wtn }0≤n≤N), as an approximation to u(tN , XtN)

$$l(\theta) = \mathbb{E}\left[\left|g(X_{t_N}) - \hat{u}(\{X_{t_n}\}_{0\le n\le N}, \{W_{t_n}\}_{0\le n\le N})\right|^2\right]$$

- The paths can be simulated easily. Therefore the commonly used SGD algorithm fits this problem well.
- Call the introduced methodology deep BSDE method since we use the BSDE and DNN as essential tool (Han, et al.)

Wharton
University of Pennsylvania

The Intuition

- Question:
  - Such deep neural networks can be trained?
  - Intuition: there are skip connections between different subnetworks (Han, et al.)

$$
\begin{aligned}
& u(t_{n+1}, X_{t_{n+1}}) - u(t_n, X_{t_n}) \\
\approx & - f\left(t_n, X_{t_n}, u(t_n, X_{t_n}), (\sigma^{\mathrm{T}} \nabla u)(t_n, X_{t_n} | \theta_n)\right) \Delta t_n \\
& + (\sigma^{\mathrm{T}} \nabla u)(t_n, X_{t_n} | \theta_n) \, \Delta W_n,
\end{aligned}
$$

Wharton
University of Pennsylvania

Analogy to Deep Reinforcement Learning

- We want to draw insight from represent policy function (control) through neural networks in Deep Reinforcement Learning

Deep BSDE method		DRL
BSDE	$\longleftrightarrow$	Markov decision model
gradient of the solution	$\longleftrightarrow$	optimal policy function

Wharton
University of Pennsylvania

DBDP1

- backward dynamic programming
  - use independent neural networks respectively for the approximation of u and for the approximation of σ.

- Initialize from an estimation $\widehat{\mathcal{U}}_N^{(1)}$ of $u(t_N, .)$ with $\widehat{\mathcal{U}}_N^{(1)} = g$
- For $i = N-1, \dots, 0$, given $\widehat{\mathcal{U}}_{i+1}^{(1)}$, use a pair of deep neural networks $(\mathcal{U}_i(.;\theta), \mathcal{Z}_i(.;\theta)) \in \mathcal{NN}_{d,1,L,m}^{\varrho}(\mathbb{R}^{N_m}) \times \mathcal{NN}_{d,d,L,m}^{\varrho}(\mathbb{R}^{N_m})$ for the approximation of $(u(t_i, .), \sigma^\intercal(t_i, .) D_x u(t_i, .))$, and compute (by SGD) the minimizer of the expected quadratic loss function

$$\begin{cases} \hat{L}_i^{(1)}(\theta) := \mathbb{E}\left|\widehat{\mathcal{U}}_{i+1}^{(1)}(X_{t_{i+1}}) - F(t_i, X_{t_i}, \mathcal{U}_i(X_{t_i};\theta), \mathcal{Z}_i(X_{t_i};\theta), \Delta t_i, \Delta W_{t_i})\right|^2 \\ \theta_i^* \in \arg\min_{\theta \in \mathbb{R}^{N_m}} \hat{L}_i^1(\theta). \end{cases}$$

Hure et al.

Then, update: $\widehat{\mathcal{U}}_i^{(1)} = \mathcal{U}_i(.;\theta_i^*)$, and set $\widehat{\mathcal{Z}}_i^{(1)} = \mathcal{Z}_i(.;\theta_i^*)$.

Wharton
University of Pennsylvania

DBDP as to DBSDE

- The advantages of the two schemes, compared to the Deep BSDE algorithm, are the following:
  - We solve smaller problems that are less prone to be trapped in local mimimizer. The memory needed in [HJE17] can be a problem when taking too many time steps.
  - at each time step, we initialize the weights and bias of the neural network to the weights and bias of the previous time step treated : this methodology always used in iterative solvers in PDE methods permits to have a starting point close to the solution, and then to avoid local minima too far away from the true solution. Besides the number of gradient iterations to achieve is rather small after the first resolution step (Hure et al.)

Wharton
University of Pennsylvania

Convergence Analysis

Part 3

Convergence of DBDP1

- Proof is LOOOOOONG
  - Approximation error of u and z goes to zero as n goes to infinity according to the universal approximation theorem (HSW, 89).

$$\varepsilon_i^{\mathcal{N},v} := \inf_{\xi} \mathbb{E}\left|\hat{v}_i(X_{t_i}) - \mathcal{U}_i(X_{t_i};\xi)\right|^2, \qquad \varepsilon_i^{\mathcal{N},z} := \inf_{\eta} \mathbb{E}\left|\overline{\hat{z}_i}(X_{t_i}) - \mathcal{Z}_i(X_{t_i};\eta)\right|^2.$$

  - Consistency of DBDP1

Theorem 4.1. (Consistency of DBDP1) *Under* **(H1)**, *there exists a constant* $C > 0$, *independent of* π, *such that*

$$\mathcal{E}\left[(\widehat{\mathcal{U}}^{(1)}, \widehat{\mathcal{Z}}^{(1)}), (Y, Z)\right] \leq C\Big(\mathbb{E}\left|g(\mathcal{X}_T) - g(X_T)\right|^2 + |\pi| + \varepsilon^Z(\pi) + \sum_{i=0}^{N-1}\left(N\varepsilon_i^{\mathcal{N},v} + \varepsilon_i^{\mathcal{N},z}\right)\Big). \tag{4.8}$$

Application

Part 4

American Option

- Augment the classical Black Scholes model by some important factors in real markets, including defaultable securities, transactions costs, uncertainties in the model parameters, etc.
- “Ideally the pricing models should take into account the whole basket of financial derivative underlyings, resulting in high-dimensional nonlinear PDEs.”(Han et al.)
- To test the deep BSDE method, we study a special case of the recursive valuation model with default risk (Duffie et al. 1996, Bender et al. 2015)

Formulation

- Risk neutral stock price,

$$dX_t^i = rX_t^i dt + \sigma_i X_t^i dW_t^i,$$

- The value at time t of an American option with payoff g and maturity T is given,

$$u(t,x) = \sup_{\tau \in \mathcal{T}_{t,T}} \mathbb{E}[e^{-r\tau} g(X_\tau)]$$

- τt,T is the solution to the variational inequality,

$$\begin{cases} \min\left[-\partial_t u - \hat{\mathcal{L}}u, u - g\right] &= 0, & \text{on } [0,T) \times (0,\infty)^d \\ u(T,.) &= g, & \text{on } (0,\infty)^d, \end{cases}$$

- Solve for the BSDE,

$$\hat{\mathcal{L}}u(t,x) = \frac{1}{2}\sum_{i=1}^{d} \sigma_i^2 x_i^2 D_{x_i}^2 u(t,x) + r\sum_{i=1}^{d} x_i D_{x_i} u(t,x) - ru(t,x)$$

Performance

	Nonlinear BS
ReLU	0.46% (0.0008)
Tanh	0.44% (0.0006)
Sigmoid	0.46% (0.0004)
Softplus	0.45% (0.0007)

Han, et al.

Number of layers†	29	58	87	116	145
Mean of relative error	2.29%	0.90%	0.60%	0.56%	0.53%
Std. of relative error	0.0026	0.0016	0.0017	0.0017	0.0014

Huré, et al.

Dimension	nb step	value	std	reference
1	10	0.06047	0.00023	0.060903
1	20	0.060789	0.00021	0.060903
1	40	0.061122	0.00015	0.060903
1	80	0.0613818	0.00019	0.060903
5	10	0.10537	0.00014	0.10738
5	20	0.10657	0.00011	0.10738
5	40	0.10725	0.00012	0.10738
5	80	0.107650	0.00016	0.10738
10	10	0.12637	0.00014	0.12996
10	20	0.128292	0.00011	0.12996
10	40	0.12937	0.00014	0.12996
10	80	0.129923	0.00016	0.12996
20	10	0.1443	0.00014	0.1510
20	20	0.147781	0.00012	0.1510
20	40	0.149560	0.00012	0.1510
20	80	0.15050	0.00010	0.1510
40	10	0.15512	0.00018	0.1680
40	20	0.16167	0.00015	0.1680
40	40	0.16487	0.00011	0.1680
40	80	0.16665	0.00013	0.1680
40	160	0.16758	0.00016	0.1680

As stated in the paper, “As it is done in python, the global graph creation takes much time as it is repeated for each time step and the global resolution is a little bit time consuming : as the dimension of the problem increases, the time difference decreases and it becomes hard to compare the computational time for a given accuracy when the dimension is above 5”, there is no time provided for computing the example

Wharton
University of Pennsylvania

Linear Quadratic Gaussian Control

- Consider a stochastic optimal control problem in R^d

$$dX_t = 2\sqrt{\lambda}\, m_t\, dt + \sqrt{2}\, dW_t$$

$$J(\{m_t\}_{0\le t\le T}) = \mathbb{E}\big[\int_0^T \|m_t\|_2^2\, dt + g(X_T)\big]$$

- the optimal control is

$$m_t^* = \frac{\nabla u(t,x)}{\sqrt{2\lambda}}$$

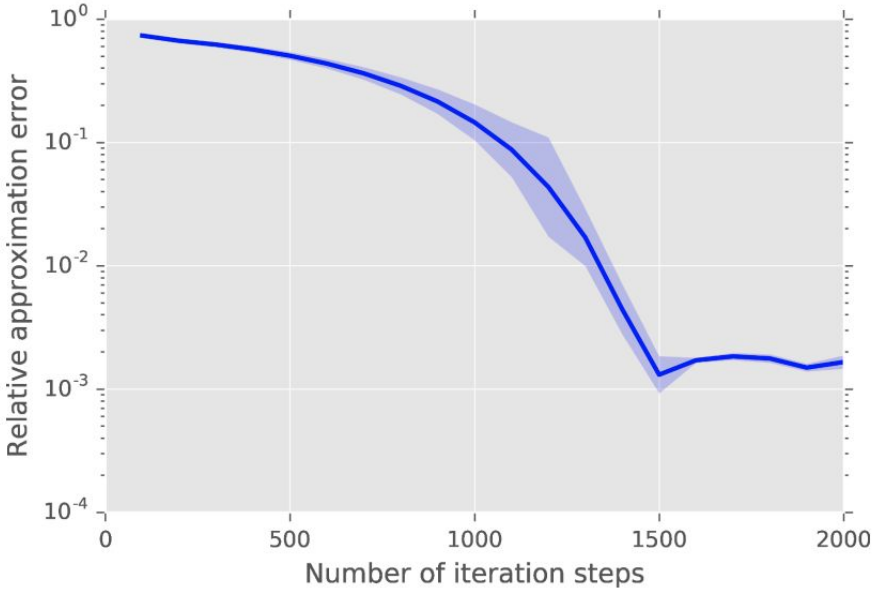
$$u(t,x) = -\frac{1}{\lambda}\ln\Big(\mathbb{E}\Big[\exp\Big(-\lambda g(x+\sqrt{2}W_{T-t})\Big)\Big]\Big)$$

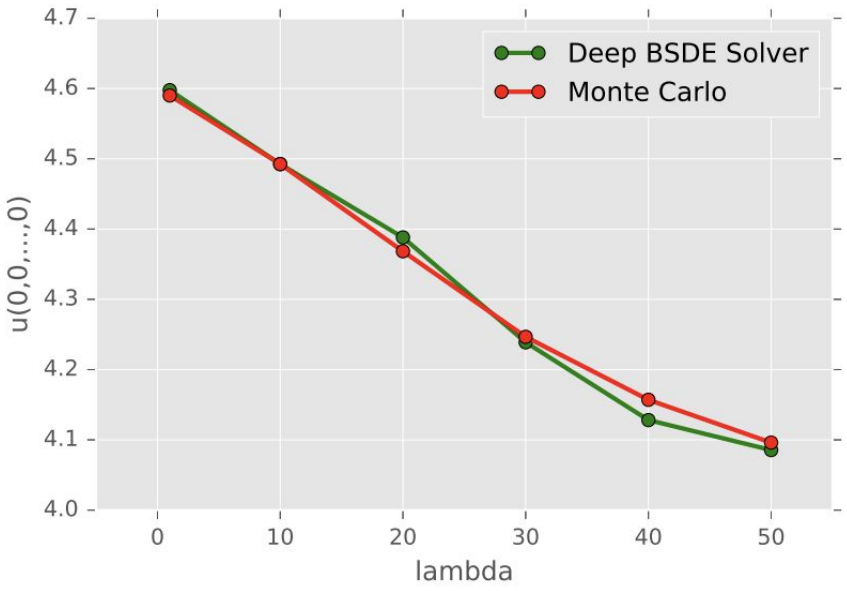
- via the Hamilton-Jacobi-Bellman equation,

$$\frac{\partial u}{\partial t}(t,x) + \Delta u(t,x) - \lambda\|\nabla u(t,x)\|_2^2 = 0$$

Wharton
University of Pennsylvania

Result





Wharton
University of Pennsylvania

THANK YOU !