Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations

Weinan E[1], Jiequn Han[2], and Arnulf Jentzen[3]

[1]Beijing Institute of Big Data Research (China), Princeton University (USA), and Peking University (China), e-mail: weinan (at) math.princeton.edu

[2]Princeton University (USA), e-mail: jiequnh (at) princeton.edu

[3]ETH Zurich (Switzerland), e-mail: arnulf.jentzen (at) sam.math.ethz.ch

June 16, 2017

Abstract

We propose a new algorithm for solving parabolic partial differential equations (PDEs) and backward stochastic differential equations (BSDEs) in high dimension, by making an analogy between the BSDE and reinforcement learning with the gradient of the solution playing the role of the policy function, and the loss function given by the error between the prescribed terminal condition and the solution of the BSDE. The policy function is then approximated by a neural network, as is done in deep reinforcement learning. Numerical results using TENSORFLOW illustrate the efficiency and accuracy of the proposed algorithms for several 100-dimensional nonlinear PDEs from physics and finance such as the Allen-Cahn equation, the Hamilton-Jacobi-Bellman equation, and a nonlinear pricing model for financial derivatives.

Contents

1

1 Introduction

Developing efficient numerical algorithms for high dimensional (say, hundreds of dimensions) partial differential equations (PDEs) has been one of the most challenging tasks in applied mathematics. As is well-known, the difficulty lies in the "curse of dimensionality" [1], namely, as the dimensionality grows, the complexity of the algorithms grows exponentially. For this reason, there are only a limited number of cases where practical

high dimensional algorithms have been developed. For linear parabolic PDEs, one can use the Feynman-Kac formula and Monte Carlo methods to develop efficient algorithms to evaluate solutions at any given space-time locations. For a class of inviscid Hamilton-Jacobi equations, Darbon & Osher have recently developed an algorithm which performs numerically well in the case of such high dimensional inviscid Hamilton-Jacobi equations; see [9]. Darbon & Osher's algorithm is based on results from compressed sensing and on the Hopf formulas for the Hamilton-Jacobi equations. A general algorithm for (nonlinear) parabolic PDEs based on the Feynman-Kac and Bismut-Elworthy-Li formula and a multi-level decomposition of Picard iteration was developed in [11] and has been shown to be quite efficient on a number examples in finance and physics. The complexity of the algorithm is shown to be $O(d\varepsilon^{-4})$ for semilinear heat equations, where d is the dimensionality of the problem and ε is the required accuracy.

In recent years, a new class of techniques, called deep learning, have emerged in machine learning and have proven to be very effective in dealing with a large class of high dimensional problems in computer vision (cf., e.g., [23]), natural language processing (cf., e.g., [20]), time series analysis, etc. (cf., e.g., [15, 24]). This success fuels in speculations that deep learning might hold the key to solve the curse of dimensionality problem. It should be emphasized that at the present time, there are no theoretical results that support such claims although the practical success of deep learning has been astonishing. However, this should not prevent us from trying to apply deep learning to other problems where the curse of dimensionality has been the issue.

In this paper, we explore the use of deep learning for solving general high dimensional PDEs. To this end, it is necessary to formulate the PDEs as a learning problem. Motivated by ideas in [16] where deep learning-based algorithms were developed for high dimensional stochastic control problems, we explore a connection between (nonlinear) parabolic PDEs and backward stochastic differential equations (BSDEs) (see [26, 28, 25]) since BSDEs share a lot of common features with stochastic control problems.

2   Main ideas of the algorithm

We will consider a fairly general class of nonlinear parabolic PDEs (see (30) in Subsection 4.1 below). The proposed algorithm is based on the following set of ideas:

(i) Through the so-called nonlinear Feynman-Kac formula, we can formulate the PDEs equivalently as BSDEs.

(ii) One can view the BSDE as a stochastic control problem with the gradient of the solution being the policy function. These stochastic control problems can then be viewed as model-based reinforcement learning problems.

(iii) The (high dimensional) policy function can then be approximated by a deep neural network, as has been done in deep reinforcement learning.

3

Instead of formulating initial value problems, as is commonly done in the PDE literature, we consider the set up with terminal conditions since this facilitates making connections with BSDEs. Terminal value problems can obviously be transformed to initial value problems and vice versa.

In the remainder of this section we present a rough sketch of the derivation of the proposed algorithm, which we refer to as *deep BSDE solver*. In this derivation we restrict ourself to a specific class of nonlinear PDEs, that is, we restrict ourself to semilinear heat equations (see (PDE) below) and refer to Subsections 3.2 and 4.1 below for the general introduction of the deep BSDE solver.

2.1 An example: a semilinear heat partial differential equation (PDE)

Let $T \in (0, \infty)$, $d \in \mathbb{N}$, $\xi \in \mathbb{R}^d$, let $f \colon \mathbb{R} \times \mathbb{R}^d \to \mathbb{R}$ and $g \colon \mathbb{R}^d \to \mathbb{R}$ be continuous functions, and let $u = (u(t,x))_{t \in [0,T], x \in \mathbb{R}^d} \in C^{1,2}([0,T] \times \mathbb{R}^d, \mathbb{R})$ satisfy for all $t \in [0,T]$, $x \in \mathbb{R}^d$ that $u(T, x) = g(x)$ and

$$\frac{\partial u}{\partial t}(t,x) + \frac{1}{2}(\Delta_x u)(t,x) + f\big(u(t,x), (\nabla_x u)(t,x)\big) = 0. \tag{PDE}$$

A key idea of this work is to reformulate the PDE (PDE) as an appropriate *stochastic control problem*.

2.2 Formulation of the PDE as a suitable stochastic control problem

More specifically, let $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space, let $W \colon [0,T] \times \Omega \to \mathbb{R}^d$ be a d-dimensional standard Brownian motion on $(\Omega, \mathcal{F}, \mathbb{P})$, let $\mathbb{F} = (\mathbb{F}_t)_{t \in [0,T]}$ be the normal filtration on $(\Omega, \mathcal{F}, \mathbb{P})$ generated by W, let $\mathcal{A}$ be the set of all $\mathbb{F}$-adapted $\mathbb{R}^d$-valued stochastic processes with continuous sample paths, and for every $y \in \mathbb{R}$ and every $Z \in \mathcal{A}$ let $Y^{y,Z} \colon [0,T] \times \Omega \to \mathbb{R}$ be an $\mathbb{F}$-adapted stochastic process with continuous sample paths which satisfies that for all $t \in [0,T]$ it holds $\mathbb{P}$-a.s. that

$$Y_t^{y,Z} = y - \int_0^t f\big(Y_s^{y,Z}, Z_s\big)\, ds + \int_0^t \langle Z_s, dW_s\rangle_{\mathbb{R}^d}. \tag{1}$$

We now view the solution $u \in C^{1,2}([0,T] \times \mathbb{R}^d, \mathbb{R})$ of (PDE) and its spatial derivative as the solution of a stochastic control problem associated to (1). More formally, under suitable regularity hypotheses on the nonlinearity f it holds that the pair consisting of $u(0, \xi) \in \mathbb{R}$ and $((\nabla_x u)(t, \xi + W_t))_{t \in [0,T]} \in \mathcal{A}$ is the (up to indistinguishability) unique global minimum of the function

$$\mathbb{R} \times \mathcal{A} \ni (y, Z) \mapsto \mathbb{E}\big[|Y_T^{y,Z} - g(\xi + W_T)|^2\big] \in [0, \infty]. \tag{2}$$

One can also view the stochastic control problem (1)–(2) (with Z being the control) as a model-based reinforcement learning problem. In that analogy, we view Z as the policy and we approximate $Z \in \mathcal{A}$ using feedforward neural networks (see (11) and Section 4 below for further details). The process $u(t, \xi + W_t)$, $t \in [0, T]$, corresponds to the value function associated to the stochastic control problem and can be computed approximatively by employing the policy Z (see (9) below for details). The connection between the PDE (PDE) and the stochastic control problem (1)–(2) is based on the nonlinear Feynman-Kac formula which links PDEs and BSDEs (see (BSDE) and (3) below).

2.3 The nonlinear Feynman-Kac formula

Let $Y \colon [0, T] \times \Omega \to \mathbb{R}$ and $Z \colon [0, T] \times \Omega \to \mathbb{R}^d$ be $\mathbb{F}$-adapted stochastic processes with continuous sample paths which satisfy that for all $t \in [0, T]$ it holds $\mathbb{P}$-a.s. that

$$Y_t = g(\xi + W_T) + \int_t^T f(Y_s, Z_s)\, ds - \int_t^T \langle Z_s, dW_s \rangle_{\mathbb{R}^d}. \tag{BSDE}$$

Under suitable additional regularity assumptions on the nonlinearity f we have that the nonlinear parabolic PDE (PDE) is related to the BSDE (BSDE) in the sense that for all $t \in [0, T]$ it holds $\mathbb{P}$-a.s. that

$$Y_t = u(t, \xi + W_t) \in \mathbb{R} \qquad \text{and} \qquad Z_t = (\nabla_x u)(t, \xi + W_t) \in \mathbb{R}^d \tag{3}$$

(cf., e.g., [25, Section 3] and [27]). The first identity in (3) is sometimes referred to as *nonlinear Feynman-Kac formula* in the literature.

2.4 Forward discretization of the backward stochastic differential equation (BSDE)

To derive the deep BSDE solver, we first plug the second identity in (3) into (BSDE) to obtain that for all $t \in [0, T]$ it holds $\mathbb{P}$-a.s. that

$$Y_t = g(\xi + W_T) + \int_t^T f\big(Y_s, (\nabla_x u)(s, \xi + W_s)\big)\, ds - \int_t^T \langle (\nabla_x u)(s, \xi + W_s), dW_s \rangle_{\mathbb{R}^d}. \tag{4}$$

In particular, we obtain that for all $t_1, t_2 \in [0, T]$ with $t_1 \leq t_2$ it holds $\mathbb{P}$-a.s. that

$$Y_{t_2} = Y_{t_1} - \int_{t_1}^{t_2} f\big(Y_s, (\nabla_x u)(s, \xi + W_s)\big)\, ds + \int_{t_1}^{t_2} \langle (\nabla_x u)(s, \xi + W_s), dW_s \rangle_{\mathbb{R}^d}. \tag{5}$$

Next we apply a time discretization to (5). More specifically, let $N \in \mathbb{N}$ and let $t_0, t_1, \ldots, t_N \in [0, T]$ be real numbers which satisfy

$$0 = t_0 < t_1 < \ldots < t_N = T \tag{6}$$

and observe that (5) suggests for $N \in \mathbb{N}$ sufficiently large that

$$Y_{t_{n+1}} \tag{7}$$
$$\approx Y_{t_n} - f\big(Y_{t_n}, (\nabla_x u)(t_n, \xi + W_{t_n})\big)(t_{n+1} - t_n) + \big\langle (\nabla_x u)(t_n, \xi + W_{t_n}), W_{t_{n+1}} - W_{t_n} \big\rangle_{\mathbb{R}^d}.$$

2.5 Deep learning-based approximations

In the next step we employ a deep learning approximation for

$$(\nabla_x u)(t_n, x) \in \mathbb{R}^d, \qquad x \in \mathbb{R}^d, \qquad n \in \{0, 1, \ldots, N\}, \tag{8}$$

but not for $u(t_n, x) \in \mathbb{R}$, $x \in \mathbb{R}^d$, $n \in \{0, 1, \ldots, N\}$. Approximations for $u(t_n, x) \in \mathbb{R}$, $x \in \mathbb{R}^d$, $n \in \{0, 1, \ldots, N\}$, in turn, can be computed recursively by using (7) together with deep learning approximations for (8). More specifically, let $\rho \in \mathbb{N}$, let $\mathcal{U}^\theta \in \mathbb{R}$, $\theta \in \mathbb{R}^\rho$, be real numbers, let $\mathcal{V}_n^\theta \colon \mathbb{R}^d \to \mathbb{R}^d$, $n \in \{0, 1, \ldots, N-1\}$, $\theta \in \mathbb{R}^\rho$, be continuous functions, and let $\mathcal{Y}^\theta \colon \{0, 1, \ldots, N\} \times \Omega \to \mathbb{R}$, $\theta \in \mathbb{R}^\rho$, be stochastic processes which satisfy for all $\theta \in \mathbb{R}^\rho$, $n \in \{0, 1, \ldots, N-1\}$ that $\mathcal{Y}_0^\theta = \mathcal{U}^\theta$ and

$$\mathcal{Y}_{n+1}^\theta = \mathcal{Y}_n^\theta - f\big(\mathcal{Y}_n^\theta, \mathcal{V}_n^\theta(\xi + W_{t_n})\big)(t_{n+1} - t_n) + \big\langle \mathcal{V}_n^\theta(\xi + W_{t_n}), W_{t_{n+1}} - W_{t_n} \big\rangle_{\mathbb{R}^d}. \tag{9}$$

We think of $\rho \in \mathbb{N}$ as the number of parameters in the neural network, for all appropriate $\theta \in \mathbb{R}^\rho$ we think of $\mathcal{U}^\theta \in \mathbb{R}$ as suitable approximations

$$\mathcal{U}^\theta \approx u(0, \xi) \tag{10}$$

of $u(0, \xi)$, and for all appropriate $\theta \in \mathbb{R}^\rho$, $x \in \mathbb{R}^d$, $n \in \{0, 1, \ldots, N-1\}$ we think of $\mathcal{V}_n^\theta(x) \in \mathbb{R}^{1 \times d}$ as suitable approximations

$$\mathcal{V}_n^\theta(x) \approx (\nabla_x u)(t_n, x) \tag{11}$$

of $(\nabla_x u)(t_n, x)$.

2.6 Stochastic optimization algorithms

The "appropriate" $\theta \in \mathbb{R}^\rho$ can be obtained by minimizing the expected loss function through stochastic gradient descent-type algorithms. For the loss function we pick the squared approximation error associated to the terminal condition of the BSDE (BSDE). More precisely, assume that the function $\mathbb{R}^\rho \ni \theta \mapsto \mathbb{E}\big[|\mathcal{Y}_N^\theta - g(\mathcal{X}_N)|^2\big] \in [0, \infty]$ has a unique global minimum and let $\Lambda \in \mathbb{R}^\rho$ be the real vector for which the function

$$\mathbb{R}^\rho \ni \theta \mapsto \mathbb{E}\big[|\mathcal{Y}_N^\theta - g(\mathcal{X}_N)|^2\big] \in [0, \infty] \tag{12}$$

is minimal. Minimizing the function (12) is inspired by the fact that

$$\mathbb{E}\big[|Y_T - g(X_T)|^2\big] = 0 \tag{13}$$

according to (BSDE) above (cf. (2) above). Under suitable regularity assumptions, we approximate the vector $\Lambda \in \mathbb{R}^\rho$ through stochastic gradient descent-type approximation methods and thereby we obtain random approximations $\Theta_0, \Theta_1, \Theta_2, \ldots : \Omega \to \mathbb{R}^\rho$ of $\Lambda \in \mathbb{R}^{\rho_\vartheta}$. For sufficiently large $N, \rho, m \in \mathbb{N}$ we then employ the random variable $\mathcal{U}^{\Theta_m} : \Omega \to \mathbb{R}$ as a suitable implementable approximation

$$\mathcal{U}^{\Theta_m} \approx u(0, \xi) \tag{14}$$

of $u(0, \xi)$ (cf. (10) above) and for sufficiently large $N, \rho, m \in \mathbb{N}$ and all $x \in \mathbb{R}^d$, $n \in \{0, 1, \ldots, N-1\}$ we use the random variable $\mathcal{V}_n^{\Theta_m}(x) : \Omega \to \mathbb{R}^{1 \times d}$ as a suitable implementable approximation

$$\mathcal{V}_n^{\Theta_m}(x) \approx (\nabla_x u)(t_n, x) \tag{15}$$

of $(\nabla_x u)(t_n, x)$ (cf. (11) above). In the next section the proposed approximation method is described in more detail.

To simplify the presentation we have restricted us in (PDE), (1), (2), (BSDE) above and Subsection 3.1 below to semilinear heat equations. We refer to Subsection 3.2 and Section 4 below for the general description of the deep BSDE solver.

3 Details of the algorithm

3.1 Formulation of the proposed algorithm in the case of semi-linear heat equations

In this subsection we describe the algorithm proposed in this article in the specific situation where (PDE) is the PDE under consideration, where *batch normalization* (see Ioffe & Szegedy [21]) is not employed, and where the plain-vanilla stochastic gradient descent approximation method with a constant learning rate $\gamma \in (0, \infty)$ and without mini-batches is the employed stochastic algorithm. The general framework, which includes the setting in this subsection as a special case, can be found in Subsection 3.2 below.

Framework 3.1 (Specific case)**.** *Let $T, \gamma \in (0, \infty)$, $d, \rho, N \in \mathbb{N}$, $\xi \in \mathbb{R}^d$, let $f \colon \mathbb{R} \times \mathbb{R}^d \to \mathbb{R}$ and $g \colon \mathbb{R}^d \to \mathbb{R}$ be functions, let $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space, let $W^m \colon [0, T] \times \Omega \to \mathbb{R}^d$, $m \in \mathbb{N}_0$, be independent d-dimensional standard Brownian motions on $(\Omega, \mathcal{F}, \mathbb{P})$, let $t_0, t_1, \ldots, t_N \in [0, T]$ be real numbers with*

$$0 = t_0 < t_1 < \ldots < t_N = T, \tag{16}$$

for every $\theta \in \mathbb{R}^\rho$ let $\mathcal{U}^\theta \in \mathbb{R}$, for every $\theta \in \mathbb{R}^\rho$, $n \in \{0, 1, \ldots, N-1\}$ let $\mathcal{V}_n^\theta \colon \mathbb{R}^d \to \mathbb{R}^d$ be a function, for every $m \in \mathbb{N}_0$, $\theta \in \mathbb{R}^\rho$ let $\mathcal{Y}^{\theta,m} \colon \{0, 1, \ldots, N\} \times \Omega \to \mathbb{R}^k$ be the stochastic process which satisfies for all $n \in \{0, 1, \ldots, N-1\}$ that $\mathcal{Y}_0^{\theta,m} = \mathcal{U}^\theta$ and

$$\mathcal{Y}_{n+1}^{\theta,m} = \mathcal{Y}_n^{\theta,m} - f\big(\mathcal{Y}_n^{\theta,m}, \mathcal{V}_n^\theta(\xi + W_{t_n}^m)\big)(t_{n+1} - t_n) + \big\langle \mathcal{V}_n^\theta(\xi + W_{t_n}^m), W_{t_{n+1}}^m - W_{t_n}^m \big\rangle_{\mathbb{R}^d}, \tag{17}$$

7

for every $m \in \mathbb{N}_0$ let $\phi^m \colon \mathbb{R}^\rho \times \Omega \to \mathbb{R}$ be the function which satisfies for all $\theta \in \mathbb{R}^\rho$, $\omega \in \Omega$ that

$$\phi^m(\theta, \omega) = \left| \mathcal{Y}_N^{\theta,m}(\omega) - g(\xi + W_T^m(\omega)) \right|^2, \tag{18}$$

for every $m \in \mathbb{N}_0$ let $\Phi^m \colon \mathbb{R}^\rho \times \Omega \to \mathbb{R}^\rho$ be a function which satisfies for all $\omega \in \Omega$, $\theta \in \{v \in \mathbb{R}^\rho \colon (\mathbb{R}^\rho \ni w \mapsto \phi_{\mathbf{s}}^m(w, \omega) \in \mathbb{R} \text{ is differentiable at } v \in \mathbb{R}^\rho)\}$ that

$$\Phi^m(\theta, \omega) = (\nabla_\theta \phi^m)(\theta, \omega), \tag{19}$$

and let $\Theta \colon \mathbb{N}_0 \times \Omega \to \mathbb{R}^\rho$ be a stochastic process which satisfy for all $m \in \mathbb{N}$ that

$$\Theta_m = \Theta_{m-1} - \gamma \cdot \Phi^m(\Theta_{m-1}). \tag{20}$$

Under suitable further hypotheses (cf. Sections 4 and 5 below), we think in the case of sufficiently large $\rho, N, m \in \mathbb{N}$ and sufficiently small $\gamma \in (0, \infty)$ of $\mathcal{U}^{\Theta_m} \in \mathbb{R}$ as an appropriate approximation

$$u(0, \xi) \approx \mathcal{U}^{\Theta_m} \tag{21}$$

of the solution $u(t, x) \in \mathbb{R}$, $(t, x) \in [0, T] \times \mathbb{R}^d$, of the PDE

$$\frac{\partial u}{\partial t}(t, x) + \frac{1}{2}(\Delta_x u)(t, x) + f\big(u(t, x), (\nabla_x u)(t, x)\big) = 0 \tag{22}$$

for $(t, x) \in [0, T] \times \mathbb{R}^d$.

3.2 Formulation of the proposed algorithm in the general case

Framework 3.2 (General case). *Let $T \in (0, \infty)$, $d, k, \rho, \varrho, N, \varsigma \in \mathbb{N}$, $\xi \in \mathbb{R}^d$, let $f \colon [0, T] \times \mathbb{R}^d \times \mathbb{R}^k \times \mathbb{R}^{k \times d} \to \mathbb{R}$, $g \colon \mathbb{R}^d \to \mathbb{R}^k$, and $\Upsilon \colon [0, T]^2 \times \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^d$ be functions, let $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space, let $W^{m,j} \colon [0, T] \times \Omega \to \mathbb{R}^d$, $m, j \in \mathbb{N}_0$, be independent d-dimensional standard Brownian motions on $(\Omega, \mathcal{F}, \mathbb{P})$, let $t_0, t_1, \ldots, t_N \in [0, T]$ be real numbers with*

$$0 = t_0 < t_1 < \ldots < t_N = T, \tag{23}$$

for every $\theta \in \mathbb{R}^\rho$ let $\mathcal{U}^\theta \in \mathbb{R}^k$, for every $\theta \in \mathbb{R}^\rho$, $\mathbf{s} \in \mathbb{R}^\varsigma$, $n \in \{0, 1, \ldots, N-1\}$, $j \in \mathbb{N}_0$ let $\mathcal{V}_{n,j}^{\theta,\mathbf{s}} \colon (\mathbb{R}^d)^{\mathbb{N}} \to \mathbb{R}^{k \times d}$ be a function, for every $m, j \in \mathbb{N}_0$ let $\mathcal{X}^{m,j} \colon \{0, 1, \ldots, N\} \times \Omega \to \mathbb{R}^d$ and $\mathcal{Y}^{\theta,\mathbf{s},m,j} \colon \{0, 1, \ldots, N\} \times \Omega \to \mathbb{R}^k$, $\theta \in \mathbb{R}^\rho$, $\mathbf{s} \in \mathbb{R}^\varsigma$, be stochastic processes which satisfy for all $\theta \in \mathbb{R}^\rho$, $\mathbf{s} \in \mathbb{R}^\varsigma$, $n \in \{0, 1, \ldots, N-1\}$ that

$$\mathcal{X}_0^{m,j} = \xi, \qquad \mathcal{Y}_0^{\theta,\mathbf{s},m,j} = \mathcal{U}^\theta, \qquad \mathcal{X}_{n+1}^{m,j} = \Upsilon\big(t_n, t_{n+1}, \mathcal{X}_n^{m,j}, W_{t_{n+1}}^{m,j} - W_{t_n}^{m,j}\big), \tag{24}$$

$$\begin{aligned}
\mathcal{Y}_{n+1}^{\theta,\mathbf{s},m,j} = \mathcal{Y}_n^{\theta,\mathbf{s},m,j} &- f\big(t_n, \mathcal{X}_n^{m,j}, \mathcal{Y}_n^{\theta,\mathbf{s},m,j}, \mathcal{V}_{n,j}^{\theta,\mathbf{s}}(\{\mathcal{X}_n^{m,i}\}_{i \in \mathbb{N}})\big)(t_{n+1} - t_n) \\
&+ \mathcal{V}_{n,j}^{\theta,\mathbf{s}}(\{\mathcal{X}_n^{m,i}\}_{i \in \mathbb{N}})(W_{t_{n+1}}^{m,j} - W_{t_n}^{m,j}), \quad (25)
\end{aligned}$$

for every $m, j \in \mathbb{N}_0$, $\mathbf{s} \in \mathbb{R}^\varsigma$ *let* $\phi_{\mathbf{s}}^{m,j} \colon \mathbb{R}^\rho \times \Omega \to \mathbb{R}$ *be the function which satisfies for all* $\theta \in \mathbb{R}^\rho$, $\omega \in \Omega$ *that*

$$\phi_{\mathbf{s}}^{m,j}(\theta, \omega) = \|\mathcal{Y}_N^{\theta,\mathbf{s},m,j}(\omega) - g(\mathcal{X}_N^{m,j}(\omega))\|_{\mathbb{R}^k}^2, \tag{26}$$

for every $m, j \in \mathbb{N}_0$, $\mathbf{s} \in \mathbb{R}^\varsigma$ *let* $\Phi_{\mathbf{s}}^{m,j} \colon \mathbb{R}^\rho \times \Omega \to \mathbb{R}^\rho$ *be a function which satisfies for all* $\omega \in \Omega$, $\theta \in \{v \in \mathbb{R}^\rho \colon (\mathbb{R}^\rho \ni w \mapsto \phi_{\mathbf{s}}^{m,j}(w, \omega) \in \mathbb{R}$ *is differentiable at* $v \in \mathbb{R}^\rho)\}$ *that*

$$\Phi_{\mathbf{s}}^{m,j}(\theta, \omega) = (\nabla_\theta \phi_{\mathbf{s}}^{m,j})(\theta, \omega), \tag{27}$$

let $\mathcal{S} \colon \mathbb{R}^\varsigma \times \mathbb{R}^\rho \times (\mathbb{R}^d)^{\{0,1,\dots,N-1\} \times \mathbb{N}} \to \mathbb{R}^\varsigma$ *be a function, for every* $m \in \mathbb{N}$ *let* $\psi_m \colon \mathbb{R}^\varrho \to \mathbb{R}^\rho$ *and* $\Psi_m \colon \mathbb{R}^\varrho \times (\mathbb{R}^\rho)^{\mathbb{N}} \to \mathbb{R}^\varrho$ *be functions, and let* $\mathbb{S} \colon \mathbb{N}_0 \times \Omega \to \mathbb{R}^\varsigma$, $\Xi \colon \mathbb{N}_0 \times \Omega \to \mathbb{R}^\varrho$, *and* $\Theta \colon \mathbb{N}_0 \times \Omega \to \mathbb{R}^\rho$ *be stochastic processes which satisfy for all* $m \in \mathbb{N}$ *that*

$$\mathbb{S}_m = \mathcal{S}\big(\mathbb{S}_{m-1}, \Theta_{m-1}, \{\mathcal{X}_n^{m-1,i}\}_{(n,i) \in \{0,1,\dots,N-1\} \times \mathbb{N}}\big), \tag{28}$$

$$\Xi_m = \Psi_m\big(\Xi_{m-1}, \{\Phi_{\mathbb{S}_m}^{m-1,j}(\Theta_{m-1})\}_{j \in \mathbb{N}}\big), \qquad and \qquad \Theta_m = \Theta_{m-1} - \psi_m(\Xi_m). \tag{29}$$

3.3 Comments on the proposed algorithm

The dynamics in (24) associated to the stochastic processes $(\mathcal{X}_n^{m,j})_{n \in \{0,1,\dots,N\}}$ for $m, j \in \mathbb{N}_0$ allows us to incorporate different algorithms for the discretization of the considered forward stochastic differential equation (SDE) into the deep BSDE solver in Subsection 3.2. The dynamics in (29) associated to the stochastic processes Ξ_m, $m \in \mathbb{N}_0$, and Θ_m, $m \in \mathbb{N}_0$, allows us to incorporate different stochastic approximation algorithms such as

- stochastic gradient descent with or without mini-batches (see Subsection 5.1 below) as well as

- adaptive moment estimation (Adam) with mini-batches (see Kingma & Jimmy [22] and Subsection 5.2 below) into the deep BSDE solver in Subsection 3.2.

The dynamics in (28) associated to the stochastic process $\mathbb{S}_m$, $m \in \mathbb{N}_0$, allows us to incorporate the standardization procedure in *batch normalization* (see Ioffe & Szegedy [21] and also Section 4 below) into the deep BSDE solver in Subsection 3.2. In that case we think of $\mathbb{S}_m$, $m \in \mathbb{N}_0$, as approximatively calculated means and standard deviations.

4   Examples for nonlinear partial differential equations (PDEs) and nonlinear backward stochastic differential equations (BSDEs)

In this section we illustrate the algorithm proposed in Subsection 3.2 using several concrete example PDEs. In the examples below we will employ the general approximation

method in Subsection 3.2 in conjunction with the Adam optimizer (cf. Example 5.2 below and Kingma & Ba [22]) with mini-batches with 64 samples in each iteration step (see Subsection 4.1 for a detailed description).

In our implementation we employ $N-1$ fully-connected feedforward neural networks to represent $\mathcal{V}_{n,j}^{\theta}$ for $n \in \{1, 2, \ldots, N-1\}$, $j \in \{1, 2, \ldots, 64\}$, $\theta \in \mathbb{R}^{\rho}$ (cf. also Figure 1 below for a rough sketch of the architecture of the deep BSDE solver). Each of the neural networks consists of 4 layers (1 input layer [d-dimensional], 2 hidden layers [both $d+10$-dimensional], and 1 output layer [d-dimensional]). The number of hidden units in each hidden layer is equal to $d + 10$. We also adopt batch normalization (BN) (see Ioffe & Szegedy [21]) right after each matrix multiplication and before activation. We employ the rectifier function $\mathbb{R} \ni x \mapsto \max\{0, x\} \in [0, \infty)$ as our activation function for the hidden variables. All the weights in the network are initialized using a normal or a uniform distribution without any pre-training. Each of the numerical experiments presented below is performed in PYTHON using TENSORFLOW on a MACBOOK PRO with a 2.90 Gigahertz (GHz) INTEL CORE i5 micro processor and 16 gigabytes (GB) of 1867 Megahertz (MHz) double data rate type three synchronous dynamic random-access memory (DDR3-SDRAM). We also refer to the PYTHON code 1 in Subsection 6.1 below for an implementation of the deep BSDE solver in the case of the 100-dimensional Allen-Cahn PDE (35).

4.1 Setting

Assume the setting in Subsection 3.2, assume for all $\theta = (\theta_1, \ldots, \theta_{\rho}) \in \mathbb{R}^{\rho}$ that $k = 1$, $\rho = d + 1 + (N-1)(2d(d+10) + (d+10)^2 + 4(d+10) + 2d)$, $\varrho = 2\rho$, $\mathcal{U}^{\theta} = \theta_1$, $\Xi_0 = 0$, let $\mu \colon [0, T] \times \mathbb{R}^d \to \mathbb{R}^d$ and $\sigma \colon [0, T] \times \mathbb{R}^d \to \mathbb{R}^{d \times d}$ be functions, let $u \colon [0, T] \times \mathbb{R}^d \to \mathbb{R}$ be a continuous and at most polynomially growing function which satisfies for all $(t, x) \in [0, T) \times \mathbb{R}^d$ that $u|_{[0,T) \times \mathbb{R}^d} \in C^{1,2}([0, T) \times \mathbb{R}^d, \mathbb{R})$, $u(T, x) = g(x)$, and

$$\frac{\partial u}{\partial t}(t, x) + \frac{1}{2} \operatorname{Trace}\big(\sigma(t, x)\,[\sigma(t, x)]^*\,(\operatorname{Hess}_x u)(t, x)\big) + \langle \mu(t, x), (\nabla_x u)(t, x) \rangle$$
$$+ f\big(t, x, u(t, x), [(\nabla_x u)(t, x)]^* \sigma(t, x)\big) = 0, \quad (30)$$

let $\varepsilon = 10^{-8}$, $\mathbb{X} = \frac{9}{10}$, $\mathbb{Y} = \frac{999}{1000}$, $J = 64$, $(\gamma_m)_{m \in \mathbb{N}} \subseteq (0, \infty)$, let $\operatorname{Pow}_r \colon \mathbb{R}^{\rho} \to \mathbb{R}^{\rho}$, $r \in (0, \infty)$, be the functions which satisfy for all $r \in (0, \infty)$, $x = (x_1, \ldots, x_{\rho}) \in \mathbb{R}^{\rho}$ that

$$\operatorname{Pow}_r(x) = (|x_1|^r, \ldots, |x_{\rho}|^r), \quad (31)$$

and assume for all $m \in \mathbb{N}$, $x, y \in \mathbb{R}^{\rho}$, $(\varphi_j)_{j \in \mathbb{N}} \in (\mathbb{R}^{\rho})^{\mathbb{N}}$ that

$$\Psi_m(x, y, (\varphi_j)_{j \in \mathbb{N}}) = \big(\mathbb{X}x + (1 - \mathbb{X})\big(\tfrac{1}{J}\textstyle\sum_{j=1}^{J}\varphi_j\big), \mathbb{Y}y + (1 - \mathbb{Y})\operatorname{Pow}_2\big(\tfrac{1}{J}\textstyle\sum_{j=1}^{J}\varphi_j\big)\big) \quad (32)$$

and

$$\psi_m(x, y) = \left[\varepsilon + \operatorname{Pow}_{1/2}\left(\frac{y}{(1 - \mathbb{Y}^m)}\right)\right]^{-1} \frac{\gamma_m x}{(1 - \mathbb{X}^m)}. \quad (33)$$

(cf. Example 5.2 below and Kingma & Ba [22]).

10

Remark 4.1. *In this remark we illustrate the specific choice of the dimension $\rho \in \mathbb{N}$ of $\theta = (\theta_1, \ldots, \theta_\rho) \in \mathbb{R}^\rho$ in the framework in Subsection 4.1 above.*

(i) *The first component of $\theta = (\theta_1, \ldots, \theta_\rho) \in \mathbb{R}^\rho$ is employed for approximating the real number $u(0, \xi) \in \mathbb{R}$.*

(ii) *The next d-components of $\theta = (\theta_1, \ldots, \theta_\rho) \in \mathbb{R}^\rho$ are employed for approximating the components of the $(1 \times d)$-matrix $(\frac{\partial}{\partial x} u)(0, \xi) \, \sigma(0, \xi) \in \mathbb{R}^{1 \times d}$.*

(iii) *In each of the employed $N - 1$ neural networks we use $d(d + 10)$ components of $\theta = (\theta_1, \ldots, \theta_\rho) \in \mathbb{R}^\rho$ to describe the linear transformation from the d-dimensional first layer (input layer) to the $(d + 10)$-dimensional second layer (first hidden layer) (to uniquely describe a real $(d + 10) \times d$-matrix).*

(iv) *In each of the employed $N - 1$ neural networks we use $(d + 10)^2$ components of $\theta = (\theta_1, \ldots, \theta_\rho) \in \mathbb{R}^\rho$ to uniquely describe the linear transformation from the $(d+10)$-dimensional second layer (first hidden layer) to the $(d + 10)$-dimensional third layer (second hidden layer) (to uniquely describe a real $(d + 10) \times (d + 10)$-matrix).*

(v) *In each of the employed $N - 1$ neural networks we use $d(d + 10)$ components of $\theta = (\theta_1, \ldots, \theta_\rho) \in \mathbb{R}^\rho$ to describe the linear transformation from the $(d + 10)$-dimensional third layer (second hidden layer) to the d-dimensional fourth layer (output layer) (to uniquely describe a real $d \times (d + 10)$-matrix).*

(vi) *After each of the linear transformations in items (iii)–(v) above we employ a componentwise affine linear transformation (multiplication with a diagonal matrix and addition of a vector) within the batch normalization procedure, i.e., in each of the employed $N-1$ neural networks, we use $2(d+10)$ components of $\theta = (\theta_1, \ldots, \theta_\rho) \in \mathbb{R}^\rho$ for the componentwise affine linear transformation between the first linear transformation (see item (iii)) and the first application of the activation function, we use $2(d + 10)$ components of $\theta = (\theta_1, \ldots, \theta_\rho) \in \mathbb{R}^\rho$ for the componentwise affine linear transformation between the second linear transformation (see item (iv)) and the second application of the activation function, and we use $2d$ components of $\theta = (\theta_1, \ldots, \theta_\rho) \in \mathbb{R}^\rho$ for the componentwise affine linear transformation after the third linear transformation (see item (v)).*

Summing (i)–(vi) results in

$$
\rho = \underbrace{1 + d}_{\text{items (i)–(ii)}} + \underbrace{(N - 1)\left(d(d + 10) + (d + 10)^2 + d(d + 10)\right)}_{\text{items (iii)–(v)}}
$$

$$
+ \underbrace{(N - 1)\left(2(d + 10) + 2(d + 10) + 2d\right)}_{\text{item (vi)}} \tag{34}
$$

$$
= d + 1 + (N - 1)\left(2d(d + 10) + (d + 10)^2 + 4(d + 10) + 2d\right).
$$

11

Figure 1: Rough sketch of the architecture of the deep BSDE solver.

4.2 Allen-Cahn equation

In this section we test the deep BSDE solver in the case of an 100-dimensional Allen-Cahn PDE with a cubic nonlinearity (see (35) below).

More specifically, assume the setting in the Subsection 4.1 and assume for all $s, t \in [0, T]$, $x, w \in \mathbb{R}^d$, $y \in \mathbb{R}$, $z \in \mathbb{R}^{1 \times d}$, $m \in \mathbb{N}$ that $\gamma_m = 5 \cdot 10^{-4}$, $d = 100$, $T = \frac{3}{10}$, $N = 20$, $\mu(t, x) = 0$, $\sigma(t, x)w = \sqrt{2}\, w$, $\xi = (0, 0, \ldots, 0) \in \mathbb{R}^d$, $\Upsilon(s, t, x, w) = x + \sqrt{2}\, w$, $f(t, x, y, z) = y - y^3$, and $g(x) = \left[2 + \frac{2}{5}\|x\|_{\mathbb{R}^d}^2\right]^{-1}$. Note that the solution u of the PDE (30) then satisfies for all $t \in [0, T)$, $x \in \mathbb{R}^d$ that $u(T, x) = g(x)$ and

$$\frac{\partial u}{\partial t}(t, x) + u(t, x) - [u(t, x)]^3 + (\Delta_x u)(t, x) = 0. \tag{35}$$

In Table 1 we approximatively calculate the mean of $\mathcal{U}^{\Theta_m}$, the standard deviation of $\mathcal{U}^{\Theta_m}$, the relative L^1-approximatin error associated to $\mathcal{U}^{\Theta_m}$, the standard deviation of the relative L^1-approximatin error associated to $\mathcal{U}^{\Theta_m}$, and the runtime in seconds needed to calculate one realization of $\mathcal{U}^{\Theta_m}$ against $m \in \{0, 1000, 2000, 3000, 4000\}$ based on 5 independent realizations (5 independent runs) (see also the PYTHON code 1 below). Table 1 also depicts the mean of the loss function associated to Θ_m and the standard deviation of the loss function associated to Θ_m against $m \in \{0, 1000, 2000, 3000, 4000\}$ based on 256 Monte Carlo samples and 5 independent realizations (5 independent runs). In addition, the relative L^1-approximation error associated to $\mathcal{U}^{\Theta_m}$ against $m \in \{1, 2, 3, \ldots, 4000\}$ is pictured on the left hand side of Figure 2 based on 5 independent realizations (5 independent runs) and the mean of the loss function associated to Θ_m against $m \in \{1, 2, 3, \ldots, 4000\}$ is pictured on the right hand side of Figure 2 based on 256 Monte Carlo samples and 5 independent realizations (5 independent runs). In the approximative computations of the relative L^1-

12

approximation errors in Table 1 and Figure 2 the value $u(0, \xi) = u(0, 0, \ldots, 0)$ of the exact solution u of the PDE (35) is replaced by the value 0.052802 which, in turn, is calculated by means of the Branching diffusion method (see the MATLAB code 2 below and see, e.g., [17, 19, 18] for analytical and numerical results for the Branching diffusion method in the literature).

Number of iteration steps m	Mean of $\mathcal{U}^{\Theta_m}$	Standard deviation of $\mathcal{U}^{\Theta_m}$	Relative L^1-appr. error	Standard deviation of the relative L^1-appr. error	Mean of the loss function	Standard deviation of the loss function	Runtime in sec. for one realization of $\mathcal{U}^{\Theta_m}$
0	0.4740	0.0514	7.9775	0.9734	0.11630	0.02953	
1000	0.1446	0.0340	1.7384	0.6436	0.00550	0.00344	201
2000	0.0598	0.0058	0.1318	0.1103	0.00029	0.00006	348
3000	0.0530	0.0002	0.0050	0.0041	0.00023	0.00001	500
4000	0.0528	0.0002	0.0030	0.0022	0.00020	0.00001	647

Table 1: Numerical simulations for the deep BSDE solver in Subsection 3.2 in the case of the PDE (35).

4.3 A Hamilton-Jacobi-Bellman (HJB) equation

In this subsection we apply the deep BSDE solver in Subsection 3.2 to a Hamilton-Jacobi-Bellman (HJB) equation which admits an explicit solution that can be obtained through the Cole-Hopf transformation (cf., e.g., Chassagneux & Richou [7, Section 4.2] and Debnath [10, Section 8.4]).

Assume the setting in the Subsection 4.1 and assume for all $s, t \in [0, T]$, $x, w \in \mathbb{R}^d$, $y \in \mathbb{R}$, $z \in \mathbb{R}^{1 \times d}$, $m \in \mathbb{N}$ that $d = 100$, $T = 1$, $N = 20$, $\gamma_m = \frac{1}{100}$, $\mu(t, x) = 0$, $\sigma(t, x)w = \sqrt{2}\, w$, $\xi = (0, 0, \ldots, 0) \in \mathbb{R}^d$, $\Upsilon(s, t, x, w) = x + \sqrt{2}\, w$, $f(t, x, y, z) = -\|z\|^2_{\mathbb{R}^{1 \times d}}$, and $g(x) = \ln(\frac{1}{2}[1 + \|x\|^2_{\mathbb{R}^d}])$. Note that the solution u of the PDE (30) then satisfies for all $t \in [0, T)$, $x \in \mathbb{R}^d$ that $u(T, x) = g(x)$ and

$$\frac{\partial u}{\partial t}(t, x) + (\Delta_x u)(t, x) = \|(\nabla_x u)(t, x)\|^2_{\mathbb{R}^d}. \tag{36}$$

In Table 2 we approximatively calculate the mean of $\mathcal{U}^{\Theta_m}$, the standard deviation of $\mathcal{U}^{\Theta_m}$, the relative L^1-approximatin error associated to $\mathcal{U}^{\Theta_m}$, the standard deviation of the relative L^1-approximatin error associated to $\mathcal{U}^{\Theta_m}$, and the runtime in seconds needed to

(a) Relative L^1-approximation error
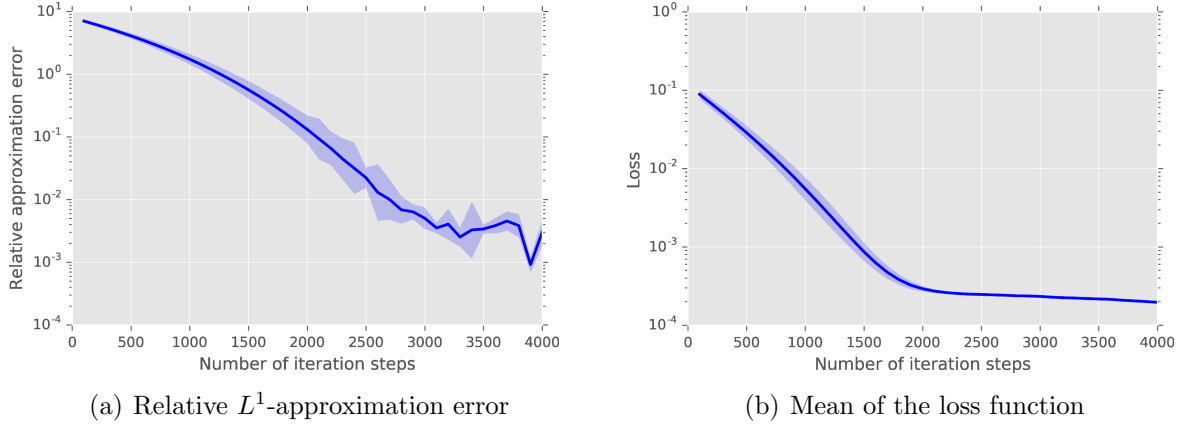

(b) Mean of the loss function

Figure 2: Relative L^1-approximation error of $\mathcal{U}^{\Theta_m}$ and mean of the loss function against $m \in \{1, 2, 3, \ldots, 4000\}$ in the case of the PDE (35). The deep BSDE approximation $\mathcal{U}^{\Theta_{4000}} \approx u(0, \xi)$ achieves a relative L^1-approximation error of size 0.0030 in a runtime of 595 seconds.

calculate one realization of $\mathcal{U}^{\Theta_m}$ against $m \in \{0, 500, 1000, 1500, 2000\}$ based on 5 independent realizations (5 independent runs). Table 2 also depicts the mean of the loss function associated to Θ_m and the standard deviation of the loss function associated to Θ_m against $m \in \{0, 500, 1000, 1500, 2000\}$ based on 256 Monte Carlo samples and 5 independent realizations (5 independent runs). In addition, the relative L^1-approximation error associated to $\mathcal{U}^{\Theta_m}$ against $m \in \{1, 2, 3, \ldots, 2000\}$ is pictured on the left hand side of Figure 3 based on 5 independent realizations (5 independent runs) and the mean of the loss function associated to Θ_m against $m \in \{1, 2, 3, \ldots, 2000\}$ is pictured on the right hand side of Figure 3 based on 256 Monte Carlo samples and 5 independent realizations (5 independent runs). In the approximative computations of the relative L^1-approximation errors in Table 2 and Figure 3 the value $u(0, \xi) = u(0, 0, \ldots, 0)$ of the exact solution u of the PDE (35) is replaced by the value 4.5901 which, in turn, is calculated by means of Lemma 4.2 below (with $d = 100$, $T = 1$, $\alpha = 1$, $\beta = -1$, $g = \mathbb{R}^d \ni x \mapsto \ln(\frac{1}{2}[1 + \|x\|^2_{\mathbb{R}^d}]) \in \mathbb{R}$ in the notation of Lemma 4.2) and a classical Monte Carlo method (see the MATLAB code 3 below).

Lemma 4.2 (Cf., e.g., Section 4.2 in [7] and Section 8.4 in [10]). *Let $d \in \mathbb{N}$, $T, \alpha \in (0, \infty)$, $\beta \in \mathbb{R} \backslash \{0\}$, let $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space, let $W \colon [0, T] \times \Omega \to \mathbb{R}^d$ be a d-dimensional standard Brownian motion, let $g \in C^2(\mathbb{R}^d, \mathbb{R})$ be a function which satisfies $\sup_{x \in \mathbb{R}^d} [\beta g(x)] < \infty$, let $f \colon [0, T] \times \mathbb{R}^d \times \mathbb{R} \times \mathbb{R}^d \to \mathbb{R}$ be the function which satisfies for all $t \in [0, T]$, $x = (x_1, \ldots, x_d), z = (z_1, \ldots, z_d) \in \mathbb{R}^d$, $y \in \mathbb{R}$ that*

$$f(t, x, y, z) = \beta \|z\|^2_{\mathbb{R}^d} = \beta \sum_{i=1}^d |z_i|^2, \tag{37}$$

14

Number of iteration steps m	Mean of $\mathcal{U}^{\Theta_m}$	Standard deviation of $\mathcal{U}^{\Theta_m}$	Relative L^1-appr. error	Standard deviation of the relative L^1-appr. error	Mean of the loss function	Standard deviation of the loss function	Runtime in sec. for one realization of $\mathcal{U}^{\Theta_m}$
0	0.3167	0.3059	0.9310	0.0666	18.4052	2.5090	
500	2.2785	0.3521	0.5036	0.0767	2.1789	0.3848	116
1000	3.9229	0.3183	0.1454	0.0693	0.5226	0.2859	182
1500	4.5921	0.0063	0.0013	0.006	0.0239	0.0024	248
2000	4.5977	0.0019	0.0017	0.0004	0.0231	0.0026	330

Table 2: Numerical simulations for the deep BSDE solver in Subsection 3.2 in the case of the PDE (36).

and let $u\colon [0,T] \times \mathbb{R}^d \to \mathbb{R}$ be the function which satisfies for all $(t,x) \in [0,T] \times \mathbb{R}^d$ that

$$u(t,x) = \frac{\alpha}{\beta} \ln\left(\mathbb{E}\left[\exp\left(\frac{\beta g(x + W_{T-t}\sqrt{2\alpha})}{\alpha} \right) \right] \right). \tag{38}$$

Then

(i) *it holds that $u\colon [0,T] \times \mathbb{R}^d \to \mathbb{R}$ is a continuous function,*

(ii) *it holds that $u|_{[0,T)\times\mathbb{R}^d} \in C^{1,2}([0,T) \times \mathbb{R}^d, \mathbb{R})$, and*

(iii) *it holds for all $(t,x) \in [0,T) \times \mathbb{R}^d$ that $u(T,x) = g(x)$ and*

$$
\begin{aligned}
&\frac{\partial u}{\partial t}(t,x) + \alpha(\Delta_x u)(t,x) + f\big(t,x,u(t,x),(\nabla_x u)(t,x)\big) \\
&= \frac{\partial u}{\partial t}(t,x) + \alpha(\Delta_x u)(t,x) + \beta\|(\nabla_x u)(t,x)\|^2_{\mathbb{R}^d} \\
&= \frac{\partial u}{\partial t}(t,x) + \alpha(\Delta_x u)(t,x) + \beta\sum_{j=1}^{d}\left|\frac{\partial u}{\partial x_j}(t,x)\right|^2 = 0.
\end{aligned}
\tag{39}
$$

Proof of Lemma 4.2. Throughout this proof let $c = \frac{\alpha}{\beta} \in \mathbb{R}\backslash\{0\}$ and let $\mathcal{V}\colon \mathbb{R}^d \to (0,\infty)$ and $v\colon [0,T] \times \mathbb{R}^d \to (0,\infty)$ be the functions which satisfy for all $t \in [0,T]$, $x \in \mathbb{R}^d$ that

$$\mathcal{V}(x) = \exp\left(\frac{g(x)}{c}\right) = \exp\left(\frac{\beta g(x)}{\alpha}\right) \quad \text{and} \quad v(t,x) = \mathbb{E}\big[\mathcal{V}\big(x + W_{T-t}\sqrt{2\alpha}\big)\big]. \tag{40}$$

15

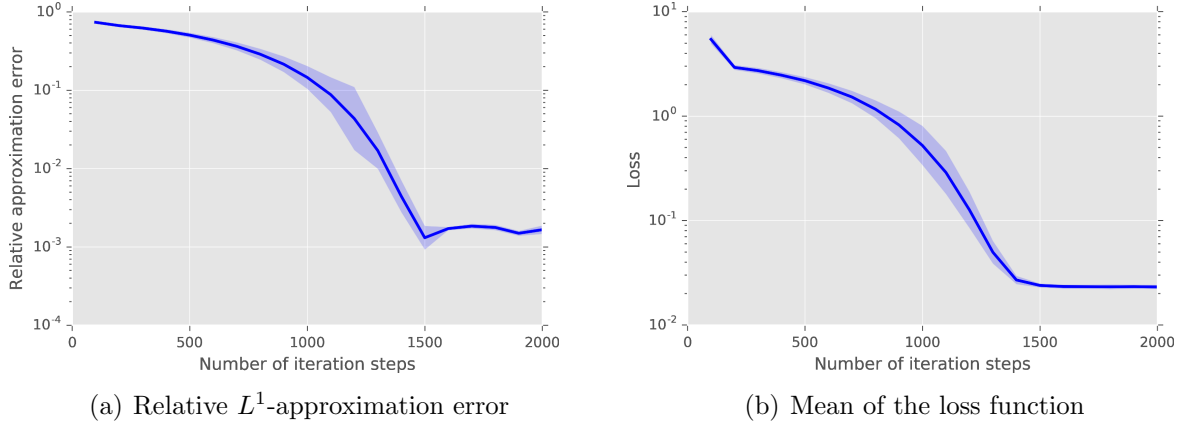(a) Relative L^1-approximation error

(b) Mean of the loss function

Figure 3: Relative L^1-approximation error of $\mathcal{U}^{\Theta_m}$ and mean of the loss function against $m \in \{1, 2, 3, \ldots, 2000\}$. The deep BSDE approximation $\mathcal{U}^{\Theta_{2000}} \approx u(0, \xi)$ achieves a relative L^1-approximation error of size 0.0017 in a runtime of 283 seconds.

Observe that the hypothesis that $\sup_{x \in \mathbb{R}^d} [\beta g(x)] < \infty$ ensures that for all $\omega \in \Omega$ it holds that

$$
\begin{aligned}
\sup_{t \in [0,T]} \sup_{x \in \mathbb{R}^d} \left| \mathcal{V}\big(x + W_{T-t}(\omega)\sqrt{2\alpha}\big) \right| &\leq \sup_{x \in \mathbb{R}^d} |\mathcal{V}(x)| = \sup_{x \in \mathbb{R}^d} \mathcal{V}(x) \\
&= \exp\left(\frac{\sup_{x \in \mathbb{R}^d} [\beta g(x)]}{\alpha} \right) < \infty.
\end{aligned}
\tag{41}
$$

Combining this with Lebesgue's theorem of dominated convergence ensures that $v \colon [0, T] \times \mathbb{R}^d \to (0, \infty)$ is a continuous function. This and the fact that

$$
\forall\, (t, x) \in [0, T] \times \mathbb{R}^d \colon u(t, x) = c \ln(v(t, x))
\tag{42}
$$

establish Item (i). Next note that the Feynman-Kac formula ensures that for all $t \in [0, T)$, $x \in \mathbb{R}^d$ it holds that $v|_{[0,T) \times \mathbb{R}^d} \in C^{1,2}([0, T) \times \mathbb{R}^d, (0, \infty))$ and

$$
\frac{\partial v}{\partial t}(t, x) + \alpha (\Delta_x v)(t, x) = 0.
\tag{43}
$$

This and (42) demonstrate Item (ii). It thus remains to prove Item (iii). For this note that the chain rule and (42) imply that for all $t \in [0, T)$, $x = (x_1, \ldots, x_d) \in \mathbb{R}^d$, $i \in \{1, 2, \ldots, d\}$ it holds that

$$
\frac{\partial u}{\partial t}(t, x) = \frac{c}{v(t, x)} \cdot \frac{\partial v}{\partial t}(t, x) \qquad \text{and} \qquad \frac{\partial u}{\partial x_i}(t, x) = \frac{c}{v(t, x)} \cdot \frac{\partial v}{\partial x_i}(t, x).
\tag{44}
$$

16

Again the chain rule and (42) hence ensure that for all $t \in [0, T)$, $x = (x_1, \ldots, x_d) \in \mathbb{R}^d$, $i \in \{1, 2, \ldots, d\}$ it holds that

$$\frac{\partial u^2}{\partial x_i^2}(t, x) = \frac{c}{v(t, x)} \cdot \frac{\partial v^2}{\partial x_i^2}(t, x) - \frac{c}{[v(t, x)]^2} \cdot \left[\frac{\partial v}{\partial x_i}(t, x)\right]^2. \tag{45}$$

This assures that for all $t \in [0, T)$, $x = (x_1, \ldots, x_d) \in \mathbb{R}^d$, $i \in \{1, 2, \ldots, d\}$ it holds that

$$\begin{aligned}
\alpha(\Delta_x u)(t, x) &= \frac{\alpha c}{v(t, x)} \cdot (\Delta_x v)(t, x) - \frac{\alpha c}{[v(t, x)]^2} \cdot \sum_{i=1}^{d} \left[\frac{\partial v}{\partial x_i}(t, x)\right]^2 \\
&= \frac{\alpha c (\Delta_x v)(t, x)}{v(t, x)} - \frac{\alpha c \left\|(\nabla_x v)(t, x)\right\|_{\mathbb{R}^d}^2}{[v(t, x)]^2}.
\end{aligned} \tag{46}$$

Combining this with (44) demonstrates that for all $t \in [0, T)$, $x \in \mathbb{R}^d$ it holds that

$$\begin{aligned}
&\frac{\partial u}{\partial t}(t, x) + \alpha(\Delta_x u)(t, x) + \beta \left\|(\nabla_x u)(t, x)\right\|_{\mathbb{R}^d}^2 \\
&= \frac{c}{v(t, x)} \cdot \frac{\partial v}{\partial t}(t, x) + \frac{\alpha c (\Delta_x v)(t, x)}{v(t, x)} - \frac{\alpha c \left\|(\nabla_x v)(t, x)\right\|_{\mathbb{R}^d}^2}{[v(t, x)]^2} + \beta \left\|(\nabla_x u)(t, x)\right\|_{\mathbb{R}^d}^2.
\end{aligned} \tag{47}$$

Equation (43) hence shows that for all $t \in [0, T)$, $x = (x_1, \ldots, x_d) \in \mathbb{R}^d$ it holds that

$$\begin{aligned}
&\frac{\partial u}{\partial t}(t, x) + \alpha(\Delta_x u)(t, x) + \beta \left\|(\nabla_x u)(t, x)\right\|_{\mathbb{R}^d}^2 \\
&= \beta \left\|(\nabla_x u)(t, x)\right\|_{\mathbb{R}^d}^2 - \frac{\alpha c \left\|(\nabla_x v)(t, x)\right\|_{\mathbb{R}^d}^2}{[v(t, x)]^2} \\
&= \beta \left[\sum_{i=1}^{d} \left|\frac{\partial u}{\partial x_i}(t, x)\right|^2\right] - \frac{\alpha c \left\|(\nabla_x v)(t, x)\right\|_{\mathbb{R}^d}^2}{[v(t, x)]^2}.
\end{aligned} \tag{48}$$

This and (44) demonstrate that for all $t \in [0, T)$, $x = (x_1, \ldots, x_d) \in \mathbb{R}^d$ it holds that

$$\begin{aligned}
&\frac{\partial u}{\partial t}(t, x) + \alpha(\Delta_x u)(t, x) + \beta \left\|(\nabla_x u)(t, x)\right\|_{\mathbb{R}^d}^2 \\
&= \beta \left[\sum_{i=1}^{d} \left|\frac{c}{v(t, x)} \cdot \frac{\partial v}{\partial x_i}(t, x)\right|^2\right] - \frac{\alpha c \left\|(\nabla_x v)(t, x)\right\|_{\mathbb{R}^d}^2}{[v(t, x)]^2} \\
&= \frac{c^2 \beta}{[v(t, x)]^2} \left[\sum_{i=1}^{d} \left|\frac{\partial v}{\partial x_i}(t, x)\right|^2\right] - \frac{\alpha c \left\|(\nabla_x v)(t, x)\right\|_{\mathbb{R}^d}^2}{[v(t, x)]^2} \\
&= \frac{[c^2 \beta - c\alpha] \left\|(\nabla_x v)(t, x)\right\|_{\mathbb{R}^d}^2}{[v(t, x)]^2} = 0.
\end{aligned} \tag{49}$$

This and the fact that

$$\forall\, x \in \mathbb{R}^d\colon u(T,x) = c \ln(v(T,x)) = c \ln(\mathcal{V}(x)) = c \ln\left(\exp\left(\frac{g(x)}{c} \right) \right) = g(x) \qquad (50)$$

establish Item (iii). The proof of Lemma 4.2 is thus completed. $\qquad\square$

4.4 Pricing of European financial derivatives with different interest rates for borrowing and lending

In this subsection we apply the deep BSDE solver to a pricing problem of an European financial derivative in a financial market where the risk free bank account used for the hedging of the financial derivative has different interest rates for borrowing and lending (see Bergman [4] and, e.g., [12, 2, 3, 5, 8, 11] where this example has been used as a test example for numerical methods for BSDEs).

Assume the setting in Subsection 4.1, let $\bar{\mu} = \frac{6}{100}$, $\bar{\sigma} = \frac{2}{10}$, $R^l = \frac{4}{100}$, $R^b = \frac{6}{100}$, and assume for all $s,t \in [0,T]$, $x = (x_1, \ldots, x_d), w = (w_1, \ldots, w_d) \in \mathbb{R}^d$, $y \in \mathbb{R}$, $z \in \mathbb{R}^{1 \times d}$, $m \in \mathbb{N}$ that $d = 100$, $T = \frac{1}{2}$, $N = 20$, $\gamma_m = 5 \cdot 10^{-3} = 0.005$, $\mu(t,x) = \bar{\mu}x$, $\sigma(t,x) = \bar{\sigma}\, \mathrm{diag}_{\mathbb{R}^{d \times d}}(x_1, \ldots, x_d)$, $\xi = (100, 100, \ldots, 100) \in \mathbb{R}^d$, and

$$g(x) = \max\left\{ \left[\max_{1 \le i \le 100} x_i \right] - 120, 0 \right\} - 2\max\left\{ \left[\max_{1 \le i \le 100} x_i \right] - 150, 0 \right\}, \qquad (51)$$

$$\Upsilon(s,t,x,w) = \exp\left(\left(\bar{\mu} - \frac{\bar{\sigma}^2}{2} \right)(t-s) \right) \exp\left(\bar{\sigma}\, \mathrm{diag}_{\mathbb{R}^{d \times d}}(w_1, \ldots, w_d) \right) x, \qquad (52)$$

$$f(t,x,y,z) = -R^l y - \frac{(\bar{\mu} - R^l)}{\bar{\sigma}} \sum_{i=1}^{d} z_i + (R^b - R^l)\max\left\{ 0, \left[\frac{1}{\bar{\sigma}} \sum_{i=1}^{d} z_i \right] - y \right\}. \qquad (53)$$

Note that the solution u of the PDE (30) then satisfies for all $t \in [0,T)$, $x = (x_1, x_2, \ldots, x_d) \in \mathbb{R}^d$ that $u(T,x) = g(x)$ and

$$\frac{\partial u}{\partial t}(t,x) + f\big(t,x,u(t,x), \bar{\sigma}\, \mathrm{diag}_{\mathbb{R}^{d \times d}}(x_1, \ldots, x_d)(\nabla_x u)(t,x)\big) + \bar{\mu} \sum_{i=1}^{d} x_i \frac{\partial u}{\partial x_i}(t,x)$$

$$+ \frac{\bar{\sigma}^2}{2} \sum_{i=1}^{d} |x_i|^2 \frac{\partial^2 u}{\partial x_i^2}(t,x) = 0. \qquad (54)$$

Hence, we obtain for all $t \in [0,T)$, $x = (x_1, x_2, \ldots, x_d) \in \mathbb{R}^d$ that $u(T,x) = g(x)$ and

$$\frac{\partial u}{\partial t}(t,x) + \frac{\bar{\sigma}^2}{2} \sum_{i=1}^{d} |x_i|^2 \frac{\partial^2 u}{\partial x_i^2}(t,x)$$

$$+ \max\left\{ R^b\left(\left[\sum_{i=1}^{d} x_i\left(\frac{\partial u}{\partial x_i} \right)(t,x) \right] - u(t,x) \right), R^l\left(\left[\sum_{i=1}^{d} x_i\left(\frac{\partial u}{\partial x_i} \right)(t,x) \right] - u(t,x) \right) \right\} = 0. \qquad (55)$$

This shows that for all $t \in [0, T)$, $x = (x_1, x_2, \ldots, x_d) \in \mathbb{R}^d$ it holds that $u(T, x) = g(x)$ and

$$\frac{\partial u}{\partial t}(t, x) + \frac{\bar{\sigma}^2}{2} \sum_{i=1}^{d} |x_i|^2 \frac{\partial^2 u}{\partial x_i^2}(t, x)$$

$$- \min\left\{ R^b\left(u(t, x) - \sum_{i=1}^{d} x_i \frac{\partial u}{\partial x_i}(t, x)\right), R^l\left(u(t, x) - \sum_{i=1}^{d} x_i \frac{\partial u}{\partial x_i}(t, x)\right) \right\} = 0. \quad (56)$$

In Table 3 we approximatively calculate the mean of $\mathcal{U}^{\Theta_m}$, the standard deviation of $\mathcal{U}^{\Theta_m}$, the relative L^1-approximatin error associated to $\mathcal{U}^{\Theta_m}$, the standard deviation of the relative L^1-approximatin error associated to $\mathcal{U}^{\Theta_m}$, and the runtime in seconds needed to calculate one realization of $\mathcal{U}^{\Theta_m}$ against $m \in \{0, 1000, 2000, 3000, 4000\}$ based on 5 independent realizations (5 independent runs). Table 3 also depicts the mean of the loss function associated to Θ_m and the standard deviation of the loss function associated to Θ_m against $m \in \{0, 1000, 2000, 3000, 4000\}$ based on 256 Monte Carlo samples and 5 independent realizations (5 independent runs). In addition, the relative L^1-approximation error associated to $\mathcal{U}^{\Theta_m}$ against $m \in \{1, 2, 3, \ldots, 4000\}$ is pictured on the left hand side of Figure 4 based on 5 independent realizations (5 independent runs) and the mean of the loss function associated to Θ_m against $m \in \{1, 2, 3, \ldots, 4000\}$ is pictured on the right hand side of Figure 4 based on 256 Monte Carlo samples and 5 independent realizations (5 independent runs). In the approximative computations of the relative L^1-approximation errors in Table 3 and Figure 4 the value $u(0, \xi) = u(0, 0, \ldots, 0)$ of the exact solution u of the PDE (56) is replaced by the value 21.299 which, in turn, is calculated by means of the multilevel-Picard approximation method in E et al. [11] (see [11, $\rho = 7$ in Table 6 in Section 4.3]).

Number of iteration steps m	Mean of $\mathcal{U}^{\Theta_m}$	Standard deviation of $\mathcal{U}^{\Theta_m}$	Relative L^1-appr. error	Standard deviation of the relative L^1-appr. error	Mean of the loss function	Standard deviation of the loss function	Runtime in sec. for one realization of $\mathcal{U}^{\Theta_m}$
0	16.964	0.882	0.204	0.041	53.666	8.957	
1000	20.309	0.524	0.046	0.025	30.886	3.076	194
2000	21.150	0.098	0.007	0.005	29.197	3.160	331
3000	21.229	0.034	0.003	0.002	29.070	3.246	470
4000	21.217	0.043	0.004	0.002	29.029	3.236	617

Table 3: Numerical simulations for the deep BSDE solver in Subsection 3.2 in the case of the PDE (56).

19

(a) Relative L^1-approximation error
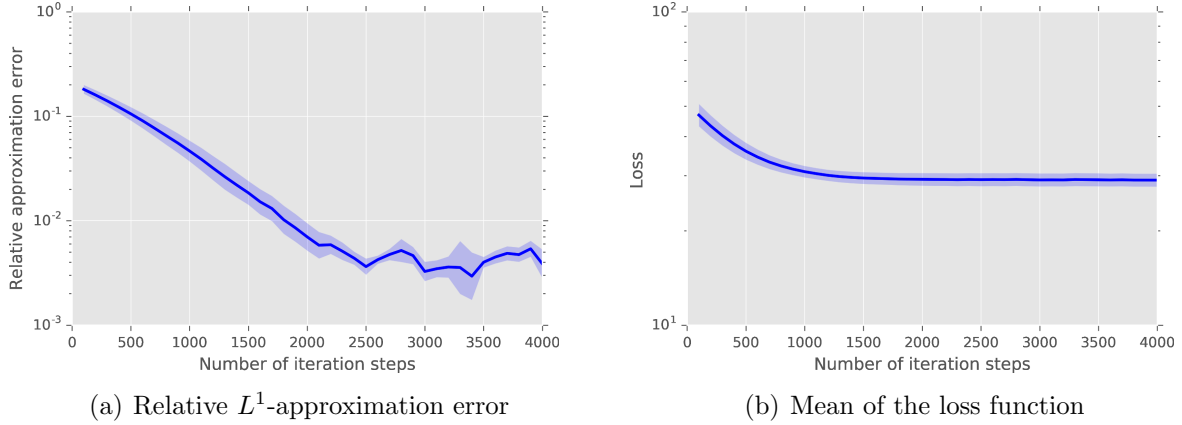
(b) Mean of the loss function

Figure 4: Relative L^1-approximation error of $\mathcal{U}^{\Theta_m}$ and mean of the loss function against $m \in \{1, 2, 3, \ldots, 4000\}$ in the case of the PDE (56). The deep BSDE approximation $\mathcal{U}^{\Theta_{4000}} \approx u(0, \xi)$ achieves a relative L^1-approximation error of size 0.0039 in a runtime of 566 seconds.

4.5 Multidimensional Burgers-type PDEs with explicit solutions

In this subsection we consider a high-dimensional version of the example analyzed numerically in Chassagneux [6, Example 4.6 in Subsection 4.2].

More specifically, assume the setting in Subsection 4.1, and assume for all $s, t \in [0, T]$, $x = (x_1, \ldots, x_d), w = (w_1, \ldots, w_d) \in \mathbb{R}^d$, $y \in \mathbb{R}$, $z = (z_i)_{i \in \{1,2,\ldots,d\}} \in \mathbb{R}^{1 \times d}$ that $\mu(t, x) = 0$, $\sigma(t, x)w = \frac{d}{\sqrt{2}}w$, $\xi = (0, 0, \ldots, 0) \in \mathbb{R}^d$, $\Upsilon(s, t, x, w) = x + \frac{d}{\sqrt{2}}w$, and

$$g(x) = \frac{\exp(T + \frac{1}{d}\sum_{i=1}^d x_i)}{\left(1 + \exp(T + \frac{1}{d}\sum_{i=1}^d x_i)\right)}, \qquad f(t, x, y, z) = \left(y - \frac{2+d}{2d}\right)\left(\sum_{i=1}^d z_i\right). \qquad (57)$$

Note that the solution u of the PDE (30) then satisfies for all $t \in [0, T)$, $x = (x_1, x_2, \ldots, x_d) \in \mathbb{R}^d$ that $u(T, x) = g(x)$ and

$$\frac{\partial u}{\partial t}(t, x) + \frac{d^2}{2}(\Delta_x u)(t, x) + \left(u(t, x) - \frac{2+d}{2d}\right)\left(d\sum_{i=1}^d \frac{\partial u}{\partial x_i}(t, x)\right) = 0 \qquad (58)$$

(cf. Lemma 4.3 below [with $\alpha = d^2$, $\kappa = 1/d$ in the notation of Lemma 4.3 below]). On the left hand side of Figure 5 we present approximatively the relative L^1-approximatin error associated to $\mathcal{U}^{\Theta_m}$ against $m \in \{1, 2, 3, \ldots, 60\,000\}$ based on 5 independent realizations (5 independent runs) in the case

$$T = 1, \qquad d = 20, \qquad N = 80, \qquad \forall\, m \in \mathbb{N}: \gamma_m = 10^{(\mathbb{1}_{[1,30000]}(m) + \mathbb{1}_{[1,50000]}(m) - 4)}. \qquad (59)$$

20

On the right hand side of Figure 5 we present approximatively the mean of the loss function associated to Θ_m against $m \in \{1, 2, 3, \ldots, 60\,000\}$ based on 256 Monte Carlo samples and 5 independent realizations (5 independent runs) in the case (59). On the left hand side of Figure 6 we present approximatively the relative L^1-approximatin error associated to $\mathcal{U}^{\Theta_m}$ against $m \in \{1, 2, 3, \ldots, 30\,000\}$ based on 5 independent realizations (5 independent runs) in the case

$$T = \frac{2}{10}, \qquad d = 50, \qquad N = 30, \qquad \forall\, m \in \mathbb{N}\colon \gamma_m = 10^{(\mathbb{1}_{[1,15000]}(m) + \mathbb{1}_{[1,25000]}(m) - 4)}. \tag{60}$$

On the right hand side of Figure 6 we present approximatively the mean of the loss function associated to Θ_m against $m \in \{1, 2, 3, \ldots, 30\,000\}$ based on 256 Monte Carlo samples and 5 independent realizations (5 independent runs) in the case (60).

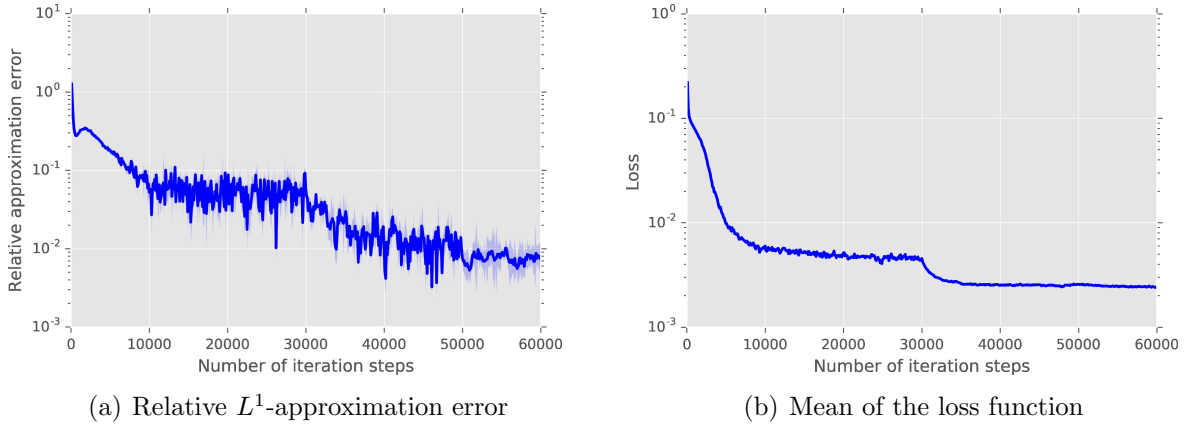(a) Relative L^1-approximation error

(b) Mean of the loss function

Figure 5: Relative L^1-approximation error of $\mathcal{U}^{\Theta_m}$ and mean of the loss function against $m \in \{1, 2, 3, \ldots, 60\,000\}$ in the case of the PDE (58) with (59). The deep BSDE approximation $\mathcal{U}^{\Theta_{60\,000}} \approx u(0, \xi)$ achieves a relative L^1-approximation error of size 0.0073 in a runtime of $20\,389$ seconds.

Lemma 4.3 (Cf. Example 4.6 in Subsection 4.2 in [6]). *Let $\alpha, \kappa, T \in (0, \infty)$, $d \in \mathbb{N}$, let $u\colon [0, T] \times \mathbb{R}^d \to \mathbb{R}$ be the function which satisfies for all $t \in [0, T]$, $x = (x_1, \ldots, x_d) \in \mathbb{R}^d$ that*

$$u(t, x) = 1 - \frac{1}{(1 + \exp(t + \kappa \sum_{i=1}^{d} x_i))} = \frac{\exp(t + \kappa \sum_{i=1}^{d} x_i)}{(1 + \exp(t + \kappa \sum_{i=1}^{d} x_i))}, \tag{61}$$

and let $f\colon [0, T] \times \mathbb{R}^d \times \mathbb{R}^{1+d} \to \mathbb{R}$ be the function which satisfies for all $t \in [0, T]$, $x \in \mathbb{R}^d$, $y \in \mathbb{R}$, $z = (z_1, \ldots, z_d) \in \mathbb{R}^d$ that

$$f(t, x, y, z) = \left(\alpha \kappa y - \frac{1}{d\kappa} - \frac{\alpha \kappa}{2} \right) \left(\sum_{i=1}^{d} z_i \right). \tag{62}$$

21

(a) Relative L^1-approximation error

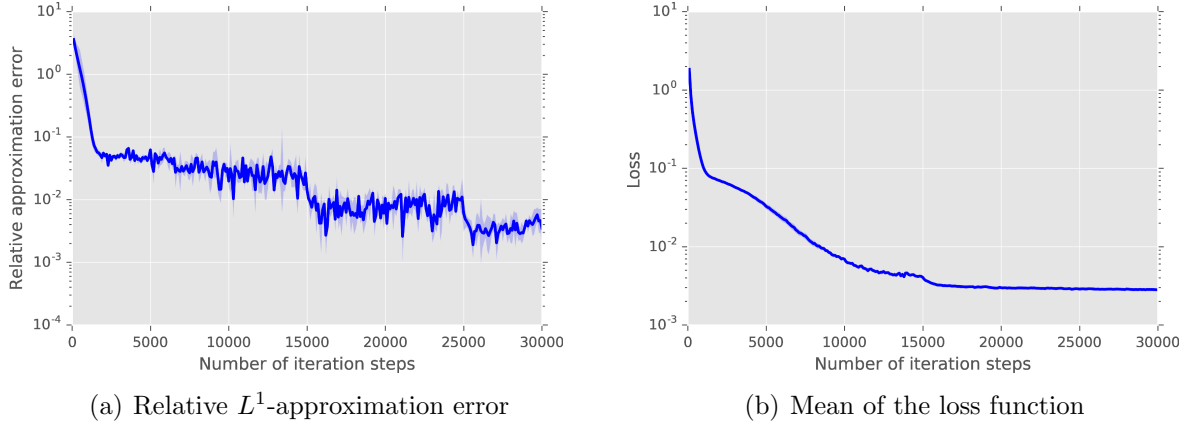(b) Mean of the loss function

Figure 6: Relative L^1-approximation error of $\mathcal{U}^{\Theta_m}$ and mean of the loss function against $m \in \{1, 2, 3, \ldots, 30\,000\}$ in the case of the PDE (58) with (60). The deep BSDE approximation $\mathcal{U}^{\Theta_{30\,000}} \approx u(0, \xi)$ achieves a relative L^1-approximation error of size 0.0035 in a runtime of 4281 seconds.

Then it holds for all $t \in [0, T]$, $x \in \mathbb{R}^d$ that

$$\frac{\partial u}{\partial t}(t, x) + \frac{\alpha}{2}(\Delta_x u)(t, x) + f\big(t, x, u(t, x), (\nabla_x u)(t, x)\big) = 0. \tag{63}$$

Proof of Lemma 4.3. Throughout this proof let $\beta, \gamma \in (0, \infty)$ be the real numbers given by

$$\beta = \alpha\kappa \qquad \text{and} \qquad \gamma = \frac{1}{d\kappa} + \frac{\alpha\kappa}{2} \tag{64}$$

and let $w \colon [0, T] \times \mathbb{R}^d \to (0, \infty)$ be the function which satisfies for all $t \in [0, T]$, $x = (x_1, \ldots, x_d) \in \mathbb{R}^d$ that

$$w(t, x) = \exp\left(t + \kappa \sum_{i=1}^{d} x_i\right). \tag{65}$$

Observe that for all $t \in [0, T]$, $x = (x_1, \ldots, x_d) \in \mathbb{R}^d$, $i \in \{1, 2, \ldots, d\}$ it holds that

$$u(t, x) = 1 - [1 + w(t, x)]^{-1} = \frac{[1 + w(t, x)]}{[1 + w(t, x)]} - \frac{1}{[1 + w(t, x)]} = \frac{w(t, x)}{1 + w(t, x)}, \tag{66}$$

$$\frac{\partial u}{\partial t}(t, x) = [1 + w(t, x)]^{-2} \cdot \frac{\partial w}{\partial t}(t, x) = \frac{w(t, x)}{[1 + w(t, x)]^2}, \tag{67}$$

and

$$\frac{\partial u}{\partial x_i}(t, x) = [1 + w(t, x)]^{-2} \cdot \frac{\partial w}{\partial x_i}(t, x) = \kappa\, w(t, x) \, [1 + w(t, x)]^{-2}. \tag{68}$$

22

Note that (66), (67), and (68) ensure that for all $t \in [0, T]$, $x \in \mathbb{R}^d$ it holds that

$$
\begin{aligned}
&\frac{\partial u}{\partial t}(t, x) + \frac{\alpha}{2}(\Delta_x u)(t, x) + f\big(t, x, u(t, x), (\nabla_x u)(t, x)\big) \\
&= \frac{\partial u}{\partial t}(t, x) + \frac{\alpha}{2}(\Delta_x u)(t, x) + (\beta u(t, x) - \gamma)\left(\sum_{i=1}^{d} \frac{\partial u}{\partial x_i}(t, x)\right) \\
&= \frac{\partial u}{\partial t}(t, x) + \frac{\alpha}{2}(\Delta_x u)(t, x) + d\,\frac{\partial u}{\partial x_1}(t, x)\big(\beta u(t, x) - \gamma\big) \\
&= \frac{w(t, x)}{[1 + w(t, x)]^2} + \frac{\alpha}{2}(\Delta_x u)(t, x) + \frac{d\kappa w(t, x)}{[1 + w(t, x)]^2}\left(\frac{\beta w(t, x)}{[1 + w(t, x)]} - \gamma\right).
\end{aligned}
\tag{69}
$$

Moreover, observe that (68) demonstrates that for all $t \in [0, T]$, $x \in \mathbb{R}^d$ it holds that

$$
\begin{aligned}
\frac{\partial^2 u}{\partial x_i^2}(t, x) &= \kappa\,\frac{\partial w}{\partial x_i}(t, x)\,[1 + w(t, x)]^{-2} - 2\kappa w(t, x)\,[1 + w(t, x)]^{-3}\,\frac{\partial w}{\partial x_i}(t, x) \\
&= \frac{\kappa^2 w(t, x)}{[1 + w(t, x)]^2} - \frac{2\kappa^2 |w(t, x)|^2}{[1 + w(t, x)]^3} = \frac{\kappa^2 w(t, x)}{[1 + w(t, x)]^2}\left[1 - \frac{2w(t, x)}{[1 + w(t, x)]}\right].
\end{aligned}
\tag{70}
$$

Hence, we obtain that for all $t \in [0, T]$, $x \in \mathbb{R}^d$ it holds that

$$
\frac{\alpha}{2}(\Delta_x u)(t, x) = \frac{d\alpha}{2}\frac{\partial^2 u}{\partial x_1^2}(t, x) = \frac{d\alpha\kappa^2 w(t, x)}{2\,[1 + w(t, x)]^2}\left[1 - \frac{2w(t, x)}{[1 + w(t, x)]}\right].
\tag{71}
$$

Combining this with (69) implies that for all $t \in [0, T]$, $x \in \mathbb{R}^d$ it holds that

$$
\begin{aligned}
&\frac{\partial u}{\partial t}(t, x) + \frac{\alpha}{2}(\Delta_x u)(t, x) + f\big(t, x, u(t, x), (\nabla_x u)(t, x)\big) \\
&= \frac{w(t, x)\,[1 - d\kappa\gamma]}{[1 + w(t, x)]^2} + \frac{\alpha}{2}(\Delta_x u)(t, x) + \frac{d\beta\kappa |w(t, x)|^2}{[1 + w(t, x)]^3} \\
&= \frac{w(t, x)\,[1 - d\kappa\gamma]}{[1 + w(t, x)]^2} + \frac{d\alpha\kappa^2 w(t, x)}{2\,[1 + w(t, x)]^2}\left[1 - \frac{2w(t, x)}{[1 + w(t, x)]}\right] + \frac{d\beta\kappa |w(t, x)|^2}{[1 + w(t, x)]^3} \\
&= \frac{w(t, x)\big[1 - d\kappa\gamma + \frac{d\alpha\kappa^2}{2}\big]}{[1 + w(t, x)]^2} - \frac{d\alpha\kappa^2 |w(t, x)|^2}{[1 + w(t, x)]^3} + \frac{d\beta\kappa |w(t, x)|^2}{[1 + w(t, x)]^3}.
\end{aligned}
\tag{72}
$$

The fact that $\alpha\kappa^2 = \beta\kappa$ hence demonstrates that for all $t \in [0, T]$, $x \in \mathbb{R}^d$ it holds that

$$
\begin{aligned}
&\frac{\partial u}{\partial t}(t, x) + \frac{\alpha}{2}(\Delta_x u)(t, x) + f\big(t, x, u(t, x), (\nabla_x u)(t, x)\big) \\
&= \frac{w(t, x)\big[1 - d\kappa\gamma + \frac{d\alpha\kappa^2}{2}\big]}{[1 + w(t, x)]^2}.
\end{aligned}
\tag{73}
$$

23

This and the fact that $1 + \frac{d\alpha\kappa^2}{2} = d\kappa\gamma$ show that for all $t \in [0, T]$, $x \in \mathbb{R}^d$ it holds that

$$\frac{\partial u}{\partial t}(t, x) + \frac{\alpha}{2}(\Delta_x u)(t, x) + f\big(t, x, u(t, x), (\nabla_x u)(t, x)\big) = 0. \tag{74}$$

The proof of Lemma 4.3 is thus completed. $\qquad\square$

4.6 An example PDE with quadratically growing derivatives and an explicit solution

In this subsection we consider a high-dimensional version of the example analyzed numerically in Gobet & Turkedjiev [13, Section 5]. More specifically, Gobet & Turkedjiev [13, Section 5] employ the PDE in (76) below as a numerical test example but with the time horizon $T = {}^2/_{10}$ instead of $T = 1$ in this article and with the dimension $d \in \{3, 5, 7\}$ instead of $d = 100$ in this article.

Assume the setting in Subsection 4.1, let $\alpha = {}^4/_{10}$, let $\psi\colon [0, T] \times \mathbb{R}^d \to \mathbb{R}$ be the function which satisfies for all $(t, x) \in [0, T] \times \mathbb{R}^d$ that $\psi(t, x) = \sin\big([T - t + \|x\|_{\mathbb{R}^d}^2]^\alpha\big)$, and assume for all $s \in [0, T]$, $t \in [0, T)$, $x, w \in \mathbb{R}^d$, $y \in \mathbb{R}$, $z \in \mathbb{R}^{1 \times d}$, $m \in \mathbb{N}$ that $T = 1$, $d = 100$, $N = 30$, $\gamma_m = 5 \cdot 10^{-3} = \frac{5}{1000} = 0.005$, $\mu(t, x) = 0$, $\sigma(t, x)w = w$, $\xi = (0, 0, \ldots, 0) \in \mathbb{R}^d$, $\Upsilon(t, s, x, w) = x + w$, $g(x) = \sin(\|x\|_{\mathbb{R}^d}^{2\alpha})$, and

$$f(t, x, y, z) = \|z\|_{\mathbb{R}^{1 \times d}}^2 - \|(\nabla_x \psi)(t, x)\|_{\mathbb{R}^d}^2 - \frac{\partial \psi}{\partial t}(t, x) - \frac{1}{2}(\Delta_x \psi)(t, x). \tag{75}$$

Note that the solution u of the PDE (30) then satisfies for all $t \in [0, T)$, $x = (x_1, x_2, \ldots, x_d) \in \mathbb{R}^d$ that $u(T, x) = g(x)$ and

$$\frac{\partial u}{\partial t}(t, x) + \|(\nabla_x u)(t, x)\|_{\mathbb{R}^d}^2 + \frac{1}{2}(\Delta_x u)(t, x) = \frac{\partial \psi}{\partial t}(t, x) + \|(\nabla_x \psi)(t, x)\|_{\mathbb{R}^d}^2 + \frac{1}{2}(\Delta_x \psi)(t, x). \tag{76}$$

On the left hand side of Figure 7 we present approximatively the relative L^1-approximatin error associated to $\mathcal{U}^{\Theta_m}$ against $m \in \{1, 2, 3, \ldots, 4000\}$ based on 5 independent realizations (5 independent runs). On the right hand side of Figure 7 we present approximatively the mean of the loss function associated to Θ_m against $m \in \{1, 2, 3, \ldots, 4000\}$ based on 256 Monte Carlo samples and 5 independent realizations (5 independent runs).

4.7 Time-dependent reaction-diffusion-type example PDEs with oscillating explicit solutions

In this subsection we consider a high-dimensional version of the example PDE analyzed numerically in Gobet & Turkedjiev [14, Subsection 6.1]. More specifically, Gobet & Turkedjiev [14, Subsection 6.1] employ the PDE in (78) below as a numerical test example
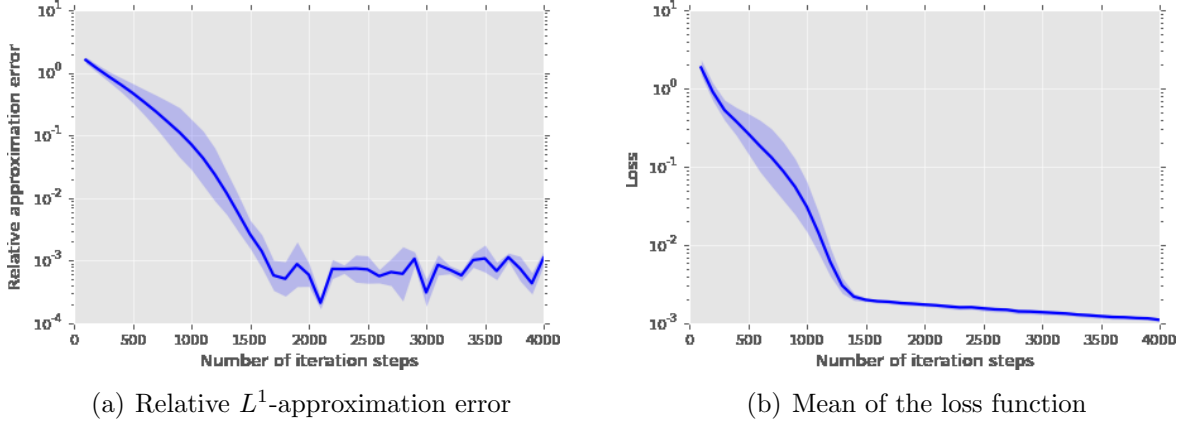
24

(a) Relative L^1-approximation error

(b) Mean of the loss function

Figure 7: Relative L^1-approximation error of $\mathcal{U}^{\Theta_m}$ and mean of the loss function against $m \in \{1, 2, 3, \ldots, 4000\}$ in the case of the PDE (76). The deep BSDE approximation $\mathcal{U}^{\Theta_{4000}} \approx u(0, \xi)$ achieves a relative L^1-approximation error of size 0.0009 in a runtime of 957 seconds.

but in two space-dimensions ($d = 2$) instead of in hundred space-dimensions ($d = 100$) as in this article.

Assume the setting in Subsection 4.1, let $\kappa = {}^6\!/_{10}$, $\lambda = {}^1\!/\!\sqrt{d}$, assume for all $s, t \in [0, T]$, $x = (x_1, \ldots, x_d), w = (w_1, \ldots, w_d) \in \mathbb{R}^d$, $y \in \mathbb{R}$, $z \in \mathbb{R}^{1 \times d}$, $m \in \mathbb{N}$ that $\gamma_m = \frac{1}{100} = 0.01$, $T = 1$, $d = 100$, $N = 30$, $\mu(t, x) = 0$, $\sigma(t, x)w = w$, $\xi = (0, 0, \ldots, 0) \in \mathbb{R}^d$, $\Upsilon(s, t, x, w) = x + w$, $g(x) = 1 + \kappa + \sin(\lambda \sum_{i=1}^d x_i)$, and

$$f(t, x, y, z) = \min\left\{1, \left[y - \kappa - 1 - \sin\left(\lambda \sum_{i=1}^d x_i\right) \exp\left(\tfrac{\lambda^2 d(t-T)}{2}\right)\right]^2\right\}. \tag{77}$$

Note that the solution u of the PDE (30) then satisfies for all $t \in [0, T)$, $x = (x_1, x_2, \ldots, x_d) \in \mathbb{R}^d$ that $u(T, x) = g(x)$ and

$$\frac{\partial u}{\partial t}(t, x) + \min\left\{1, \left[u(t, x) - \kappa - 1 - \sin\left(\lambda \sum_{i=1}^d x_i\right) \exp\left(\tfrac{\lambda^2 d(t-T)}{2}\right)\right]^2\right\} + \tfrac{1}{2}\left(\Delta_x u\right)(t, x) = 0 \tag{78}$$

(cf. Lemma 4.4 below). On the left hand side of Figure 7 we present approximatively the relative L^1-approximatin error associated to $\mathcal{U}^{\Theta_m}$ against $m \in \{1, 2, 3, \ldots, 24000\}$ based on 5 independent realizations (5 independent runs). On the right hand side of Figure 7 we present approximatively the mean of the loss function associated to Θ_m against $m \in \{1, 2, 3, \ldots, 24000\}$ based on 256 Monte Carlo samples and 5 independent realizations (5 independent runs).

Lemma 4.4 (Cf. Subsection 6.1 in [14]). *Let* $T, \kappa, \lambda \in (0, \infty)$, $d \in \mathbb{N}$ *and let* $u \colon [0, T] \times \mathbb{R}^d \to \mathbb{R}$ *be the function which satisfies for all* $t \in [0, T]$, $x = (x_1, \ldots, x_d) \in \mathbb{R}^d$ *that*

$$u(t, x) = 1 + \kappa + \sin\left(\lambda \sum_{i=1}^d x_i\right) \exp\left(\tfrac{\lambda^2 d(t-T)}{2}\right). \tag{79}$$

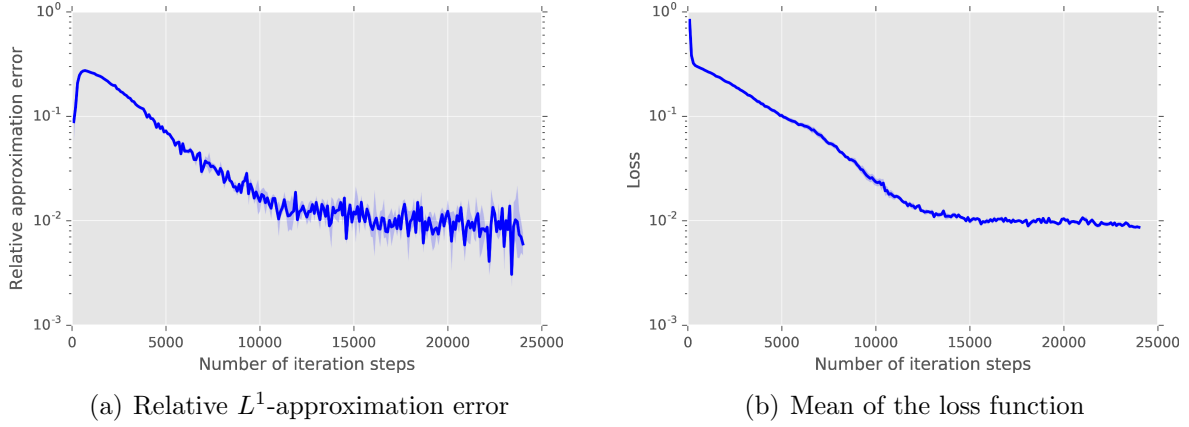(a) Relative L^1-approximation error

(b) Mean of the loss function

Figure 8: Relative L^1-approximation error of $\mathcal{U}^{\Theta_m}$ and mean of the loss function against $m \in \{1, 2, 3, \ldots, 24000\}$ in the case of the PDE (78). The deep BSDE approximation $\mathcal{U}^{\Theta_{24000}} \approx u(0, \xi)$ achieves a relative L^1-approximation error of size 0.0060 in a runtime of 4872 seconds.

Then it holds for all $t \in [0, T]$, $x = (x_1, \ldots, x_d) \in \mathbb{R}^d$ that $u \in C^{1,2}([0, T] \times \mathbb{R}^d, \mathbb{R})$, $u(T, x) = 1 + \kappa + \sin(\lambda \sum_{i=1}^d x_i)$, and

$$\frac{\partial u}{\partial t}(t, x) + \min\left\{1, \left[u(t, x) - \kappa - 1 - \sin\left(\lambda \sum_{i=1}^d x_i\right) \exp\left(\tfrac{\lambda^2 d(t-T)}{2}\right)\right]^2\right\} + \tfrac{1}{2}(\Delta_x u)(t, x) = 0. \quad (80)$$

Proof of Lemma 4.4. Note that for all $t \in [0, T]$, $x = (x_1, \ldots, x_d) \in \mathbb{R}^d$ it holds that

$$\frac{\partial u}{\partial t}(t, x) = \frac{\lambda^2 d}{2} \sin\left(\lambda \sum_{i=1}^d x_i\right) \exp\left(\tfrac{\lambda^2 d(t-T)}{2}\right). \quad (81)$$

In addition, observe that for all $t \in [0, T]$, $x = (x_1, \ldots, x_d) \in \mathbb{R}^d$, $k \in \{1, 2, \ldots, d\}$ it holds that

$$\frac{\partial u}{\partial x_k}(t, x) = \lambda \cos\left(\lambda \sum_{i=1}^d x_i\right) \exp\left(\tfrac{\lambda^2 d(t-T)}{2}\right). \quad (82)$$

Hence, we obtain that for all $t \in [0, T]$, $x = (x_1, \ldots, x_d) \in \mathbb{R}^d$, $k \in \{1, \ldots, d\}$ it holds that

$$\frac{\partial^2 u}{\partial x_k^2}(t, x) = -\lambda^2 \sin\left(\lambda \sum_{i=1}^d x_i\right) \exp\left(\tfrac{\lambda^2 d(t-T)}{2}\right). \quad (83)$$

This ensures that for all $t \in [0, T]$, $x = (x_1, \ldots, x_d) \in \mathbb{R}^d$ it holds that

$$(\Delta_x u)(t, x) = -d\,\lambda^2 \sin\left(\lambda \sum_{i=1}^d x_i\right) \exp\left(\tfrac{\lambda^2 d(t-T)}{2}\right). \quad (84)$$

Combining this with (81) proves that for all $t \in [0, T]$, $x = (x_1, \ldots, x_d) \in \mathbb{R}^d$ it holds that

$$\frac{\partial u}{\partial t}(t, x) + \frac{1}{2}(\Delta_x u)(t, x) = 0. \tag{85}$$

This demonstrates that for all $t \in [0, T]$, $x = (x_1, \ldots, x_d) \in \mathbb{R}^d$ it holds that

$$\frac{\partial v}{\partial t}(t, x) + \min\left\{1, \left[v(t, x) - \kappa - 1 - \sin\left(\lambda \sum_{i=1}^{d} x_i\right) \exp\left(\frac{\lambda^2 d(t-T)}{2}\right)\right]^2\right\} + \frac{1}{2}(\Delta_x v)(t, x)$$
$$= \frac{\partial v}{\partial t}(t, x) + \frac{1}{2}(\Delta_x v)(t, x) = 0. \tag{86}$$

The proof of Lemma 4.4 is thus completed. $\qquad\square$

5 Appendix A: Special cases of the proposed algorithm

In this subsection we illustrate the general algorithm in Subsection 3.2 in several special cases. More specifically, in Subsections 5.1 and 5.2 we provide special choices for the functions ψ_m, $m \in \mathbb{N}$, and Ψ_m, $m \in \mathbb{N}$, employed in (29) and in Subsections 5.3 and 5.4 we provide special choices for the function Υ in (24).

5.1 Stochastic gradient descent (SGD)

Example 5.1. *Assume the setting in Subsection 3.2, let $(\gamma_m)_{m \in \mathbb{N}} \subseteq (0, \infty)$, and assume for all $m \in \mathbb{N}$, $x \in \mathbb{R}^\varrho$, $(\varphi_j)_{j \in \mathbb{N}} \in (\mathbb{R}^\rho)^{\mathbb{N}}$ that*

$$\varrho = \rho, \qquad \Psi_m(x, (\varphi_j)_{j \in \mathbb{N}}) = \varphi_1, \qquad and \qquad \psi_m(x) = \gamma_m x. \tag{87}$$

Then it holds for all $m \in \mathbb{N}$ that

$$\Theta_m = \Theta_{m-1} - \gamma_m \Phi_{m-1,1}(\Theta_{m-1}). \tag{88}$$

5.2 Adaptive Moment Estimation (Adam) with mini-batches

In this subsection we illustrate how the so-called Adam optimizer (see [22]) can be employed in conjunction with the deep BSDE solver in Subsection 3.2 (cf. also Subsection 4.1 above).

Example 5.2. *Assume the setting in Subsection 3.2, assume that $\varrho = 2\rho$, let $\mathrm{Pow}_r \colon \mathbb{R}^\rho \to \mathbb{R}^\rho$, $r \in (0, \infty)$, be the functions which satisfy for all $r \in (0, \infty)$, $x = (x_1, \ldots, x_\rho) \in \mathbb{R}^\rho$ that*

$$\mathrm{Pow}_r(x) = (|x_1|^r, \ldots, |x_\rho|^r), \tag{89}$$

let $\varepsilon \in (0, \infty)$, $(\gamma_m)_{m \in \mathbb{N}} \subseteq (0, \infty)$, $(J_m)_{m \in \mathbb{N}_0} \subseteq \mathbb{N}$, $\mathbb{X}, \mathbb{Y} \in (0, 1)$, let $\mathbf{m}, \mathbb{M} \colon \mathbb{N}_0 \times \Omega \to \mathbb{R}^\wp$ be the stochastic processes which satisfy for all $m \in \mathbb{N}_0$ that $\Xi_m = (\mathbf{m}_m, \mathbb{M}_m)$, and assume for all $m \in \mathbb{N}$, $x, y \in \mathbb{R}^\wp$, $(\varphi_j)_{j \in \mathbb{N}} \in (\mathbb{R}^\wp)^{\mathbb{N}}$ that

$$\Psi_m(x, y, (\varphi_j)_{j \in \mathbb{N}}) = \big(\mathbb{X}x + (1 - \mathbb{X})\big(\tfrac{1}{J_m}\textstyle\sum_{j=1}^{J_m}\varphi_j\big), \mathbb{Y}y + (1 - \mathbb{Y})\operatorname{Pow}_2\big(\tfrac{1}{J_m}\textstyle\sum_{j=1}^{J_m}\varphi_j\big)\big) \quad (90)$$

and

$$\psi_m(x, y) = \left[\varepsilon + \operatorname{Pow}_{1/2}\left(\frac{y}{(1 - \mathbb{Y}^m)}\right)\right]^{-1}\frac{\gamma_m x}{(1 - \mathbb{X}^m)}. \quad (91)$$

Then it holds for all $m \in \mathbb{N}$ that

$$\Theta_m = \Theta_{m-1} - \left[\varepsilon + \operatorname{Pow}_{1/2}\left(\frac{\mathbb{M}_m}{(1 - \mathbb{Y}^m)}\right)\right]^{-1}\frac{\gamma_m \mathbf{m}_m}{(1 - \mathbb{X}^m)},$$

$$\mathbf{m}_m = \mathbb{X}\mathbf{m}_{m-1} + \frac{(1 - \mathbb{X})}{J_m}\left(\sum_{j=1}^{J_m}\Phi_{\mathbb{S}_m}^{m-1,j}(\Theta_{m-1})\right), \quad (92)$$

$$\mathbb{M}_m = \mathbb{Y}\mathbb{M}_{m-1} + (1 - \mathbb{Y})\operatorname{Pow}_2\left(\frac{1}{J_m}\sum_{j=1}^{J_m}\Phi_{\mathbb{S}_m}^{m-1,j}(\Theta_{m-1})\right).$$

5.3 Geometric Brownian motion

Example 5.3. *Assume the setting in Section 3.2, let $\bar{\mu}, \bar{\sigma} \in \mathbb{R}$, and assume for all $s, t \in [0, T]$, $x = (x_1, \ldots, x_d)$, $w = (w_1, \ldots, w_d) \in \mathbb{R}^d$ that*

$$\Upsilon(s, t, x, w) = \exp\left(\left(\bar{\mu} - \frac{\bar{\sigma}^2}{2}\right)(t - s)\right)\exp(\bar{\sigma}\operatorname{diag}_{\mathbb{R}^{d \times d}}(w_1, \ldots, w_d))\,x. \quad (93)$$

Then it holds for all $m, j \in \mathbb{N}_0$, $n \in \{0, 1, \ldots, N\}$ that

$$\mathcal{X}_n^{\theta,m,j} = \exp\left(\left(\bar{\mu} - \frac{\bar{\sigma}^2}{2}\right)t_n\operatorname{Id}_{\mathbb{R}^d} + \bar{\sigma}\operatorname{diag}_{\mathbb{R}^{d \times d}}(W_{t_n}^{m,j})\right)\xi. \quad (94)$$

In the setting of Example 5.3 we consider under suitable further hypotheses (cf. Subsection 4.4 above) for every sufficiently large $m \in \mathbb{N}_0$ the random variable $\mathcal{U}^{\Theta_m}$ as an approximation of $u(0, \xi)$ where $u \colon [0, T] \times \mathbb{R}^d \to \mathbb{R}^k$ is a suitable solution of the PDE

$$\frac{\partial u}{\partial t}(t, x) + \frac{\bar{\sigma}^2}{2}\sum_{i=1}^{d}|x_i|^2\left(\frac{\partial^2 u}{\partial x_i^2}\right)(t, x) + \bar{\mu}\sum_{i=1}^{d}x_i\left(\frac{\partial u}{\partial x_i}\right)(t, x)$$
$$+ f\big(t, x, u(t, x), \bar{\sigma}\left(\tfrac{\partial u}{\partial x}\right)(t, x)\operatorname{diag}_{\mathbb{R}^{d \times d}}(x_1, \ldots, x_d)\big) = 0 \quad (95)$$

with $u(T, x) = g(x)$ for $t \in [0, T]$, $x = (x_1, \ldots, x_d) \in \mathbb{R}^d$.

5.4 Euler-Maruyama scheme

Example 5.4. *Assume the setting in Section 3.2, let $\mu\colon [0,T] \times \mathbb{R}^d \to \mathbb{R}^d$ and $\sigma\colon [0,T] \times \mathbb{R}^d \to \mathbb{R}^d$ be functions, and assume for all $s,t \in [0,T]$, $x,w \in \mathbb{R}^d$ that*

$$\Upsilon(s,t,x,w) = x + \mu(s,x)(t-s) + \sigma(s,x)w. \tag{96}$$

Then it holds for all $m,j \in \mathbb{N}_0$, $n \in \{0,1,\ldots,N-1\}$ that

$$\mathcal{X}_n^{m,j} = \mathcal{X}_n^{m,j} + \mu(t_n, \mathcal{X}_n^{m,j})(t_{n+1} - t_n) + \sigma(t_n, \mathcal{X}_n^{m,j})(W_{t_{n+1}} - W_{t_n}). \tag{97}$$

In the setting of Example 5.4 we consider under suitable further hypotheses for every sufficiently large $m \in \mathbb{N}_0$ the random variable $\mathcal{U}^{\Theta_m}$ as an approximation of $u(0,\xi)$ where $u\colon [0,T] \times \mathbb{R}^d \to \mathbb{R}^k$ is a suitable solution of the PDE

$$\frac{\partial u}{\partial t}(t,x) + \tfrac{1}{2}\sum_{j=1}^d (\frac{\partial^2 u}{\partial x^2})(t,x)\big(\sigma(t,x)\, e_j^{(d)}, \sigma(t,x)\, e_j^{(d)}\big) + (\frac{\partial u}{\partial x})(t,x)\, \mu(t,x)$$

$$+ f\big(t,x,u(t,x),(\tfrac{\partial u}{\partial x})(t,x)\,\sigma(t,x)\big) = 0 \tag{98}$$

with $u(T,x) = g(x)$, $e_1^{(d)} = (1,0,\ldots,0)$, $\ldots$, $e_d^{(d)} = (0,\ldots,0,1) \in \mathbb{R}^d$ for $t \in [0,T]$, $x = (x_1,\ldots,x_d) \in \mathbb{R}^d$ (cf. (PDE) in Section 2 above).

6 Appendix B: Python and Matlab source codes

6.1 Python source code for an implementation of the deep BSDE solver in the case of the Allen-Cahn PDE (35) in Subsection 4.2

```
1  import time
2  import math
3  import tensorflow as tf
4  import numpy as np
5  from tensorflow.python.training.moving_averages \
6      import assign_moving_average
7  from scipy.stats import multivariate_normal as normal
8  from tensorflow.python.ops import control_flow_ops
9  from tensorflow import random_normal_initializer as norm_init
10 from tensorflow import random_uniform_initializer as unif_init
11 from tensorflow import constant_initializer as const_init
12
13 class SolveAllenCahn(object):
14     """The fully-connected neural network model."""
15     def __init__(self, sess):
```

```
16          self.sess = sess
17          # parameters for the PDE
18          self.d = 100
19          self.T = 0.3
20          # parameters for the algorithm
21          self.n_time = 20
22          self.n_layer = 4
23          self.n_neuron = [self.d, self.d+10, self.d+10, self.d]
24          self.batch_size = 64
25          self.valid_size = 256
26          self.n_maxstep = 4000
27          self.n_displaystep = 100
28          self.learning_rate = 5e-4
29          self.Yini = [0.3, 0.6]
30          # some basic constants and variables
31          self.h = (self.T+0.0)/self.n_time
32          self.sqrth = math.sqrt(self.h)
33          self.t_stamp = np.arange(0, self.n_time)*self.h
34          self._extra_train_ops = []
35
36      def train(self):
37          start_time = time.time()
38          # train operations
39          self.global_step = \
40              tf.get_variable('global_step', [],
41                              initializer=tf.constant_initializer(1),
42                              trainable=False, dtype=tf.int32)
43          trainable_vars = tf.trainable_variables()
44          grads = tf.gradients(self.loss, trainable_vars)
45          optimizer = tf.train.AdamOptimizer(self.learning_rate)
46          apply_op = \
47              optimizer.apply_gradients(zip(grads, trainable_vars),
48                                        global_step=self.global_step)
49          train_ops = [apply_op] + self._extra_train_ops
50          self.train_op = tf.group(*train_ops)
51          self.loss_history = []
52          self.init_history = []
53          # for validation
54          dW_valid, X_valid = self.sample_path(self.valid_size)
55          feed_dict_valid = {self.dW: dW_valid,
56                             self.X: X_valid,
57                             self.is_training: False}
58          # initialization
59          step = 1
60          self.sess.run(tf.global_variables_initializer())
61          temp_loss = self.sess.run(self.loss,
62                                    feed_dict=feed_dict_valid)
63          temp_init = self.Y0.eval()[0]
64          self.loss_history.append(temp_loss)
```

```
65          self.init_history.append(temp_init)
66          print "step: %5u,   loss: %.4e,   " % \
67                      (0, temp_loss) + \
68                      "Y0: %.4e,   runtime: %4u" % \
69                      (temp_init, time.time()-start_time+self.t_bd)
70          # begin sgd iteration
71          for _ in range(self.n_maxstep+1):
72              step = self.sess.run(self.global_step)
73              dW_train, X_train = self.sample_path(self.batch_size)
74              self.sess.run(self.train_op,
75                          feed_dict={self.dW: dW_train,
76                                        self.X: X_train,
77                                        self.is_training: True})
78              if step % self.n_displaystep == 0:
79                  temp_loss = self.sess.run(self.loss,
80                                          feed_dict=feed_dict_valid)
81                  temp_init = self.Y0.eval()[0]
82                  self.loss_history.append(temp_loss)
83                  self.init_history.append(temp_init)
84                  print "step: %5u,   loss: %.4e,   " % \
85                          (step, temp_loss) + \
86                          "Y0: %.4e,   runtime: %4u" % \
87                          (temp_init, time.time()-start_time+self.t_bd)
88              step += 1
89          end_time = time.time()
90          print "running time: %.3f s" % \
91              (end_time-start_time+self.t_bd)
92
93      def build(self):
94          start_time = time.time()
95          # build the whole network by stacking subnetworks
96          self.dW = tf.placeholder(tf.float64,
97                                      [None, self.d, self.n_time],
98                                      name='dW')
99          self.X = tf.placeholder(tf.float64,
100                                     [None, self.d, self.n_time+1],
101                                     name='X')
102         self.is_training = tf.placeholder(tf.bool)
103         self.Y0 = tf.Variable(tf.random_uniform([1],
104                                                 minval=self.Yini[0],
105                                                 maxval=self.Yini[1],
106                                                 dtype=tf.float64));
107         self.Z0 = tf.Variable(tf.random_uniform([1, self.d],
108                                                 minval=-.1,
109                                                 maxval=.1,
110                                                 dtype=tf.float64))
111         self.allones = \
112             tf.ones(shape=tf.pack([tf.shape(self.dW)[0], 1]),
113                     dtype=tf.float64)
```

```
114        Y = self.allones * self.Y0
115        Z = tf.matmul(self.allones, self.Z0)
116        with tf.variable_scope('forward'):
117            for t in xrange(0, self.n_time-1):
118                Y = Y - self.f_tf(self.t_stamp[t],
119                                   self.X[:, :, t], Y, Z)*self.h
120                Y = Y + tf.reduce_sum(Z*self.dW[:, :, t], 1,
121                                       keep_dims=True)
122                Z = self._one_time_net(self.X[:, :, t+1],
123                                        str(t+1))/self.d
124            # terminal time
125            Y = Y - self.f_tf(self.t_stamp[self.n_time-1],
126                               self.X[:, :, self.n_time-1],
127                               Y, Z)*self.h
128            Y = Y + tf.reduce_sum(Z*self.dW[:, :, self.n_time-1], 1,
129                                   keep_dims=True)
130            term_delta = Y - self.g_tf(self.T,
131                                        self.X[:, :, self.n_time])
132            self.clipped_delta = \
133                tf.clip_by_value(term_delta, -50.0, 50.0)
134            self.loss = tf.reduce_mean(self.clipped_delta**2)
135        self.t_bd = time.time()-start_time
136
137    def sample_path(self, n_sample):
138        dW_sample = np.zeros([n_sample, self.d, self.n_time])
139        X_sample = np.zeros([n_sample, self.d, self.n_time+1])
140        for i in xrange(self.n_time):
141            dW_sample[:, :, i] = \
142                np.reshape(normal.rvs(mean=np.zeros(self.d),
143                                       cov=1,
144                                       size=n_sample)*self.sqrth,
145                            (n_sample, self.d))
146            X_sample[:, :, i+1] = X_sample[:, :, i] + \
147                                   np.sqrt(2) * dW_sample[:, :, i]
148        return dW_sample, X_sample
149
150    def f_tf(self, t, X, Y, Z):
151        # nonlinear term
152        return Y-tf.pow(Y, 3)
153
154    def g_tf(self, t, X):
155        # terminal conditions
156        return 0.5/(1 + 0.2*tf.reduce_sum(X**2, 1, keep_dims=True))
157
158    def _one_time_net(self, x, name):
159        with tf.variable_scope(name):
160            x_norm = self._batch_norm(x, name='layer0_normal')
161            layer1 = self._one_layer(x_norm, self.n_neuron[1],
162                                      name='layer1')
```

```
163             layer2 = self._one_layer(layer1, self.n_neuron[2],
164                                      name='layer2')
165             z = self._one_layer(layer2, self.n_neuron[3],
166                                 activation_fn=None, name='final')
167         return z
168
169     def _one_layer(self, input_, out_sz,
170                    activation_fn=tf.nn.relu,
171                    std=5.0, name='linear'):
172         with tf.variable_scope(name):
173             shape = input_.get_shape().as_list()
174             w = tf.get_variable('Matrix',
175                                 [shape[1], out_sz], tf.float64,
176                                 norm_init(stddev= \
177                                     std/np.sqrt(shape[1]+out_sz)))
178             hidden = tf.matmul(input_, w)
179             hidden_bn = self._batch_norm(hidden, name='normal')
180         if activation_fn != None:
181             return activation_fn(hidden_bn)
182         else:
183             return hidden_bn
184
185     def _batch_norm(self, x, name):
186         """Batch normalization"""
187         with tf.variable_scope(name):
188             params_shape = [x.get_shape()[-1]]
189             beta = tf.get_variable('beta', params_shape,
190                                    tf.float64,
191                                    norm_init(0.0, stddev=0.1,
192                                        dtype=tf.float64))
193             gamma = tf.get_variable('gamma', params_shape,
194                                     tf.float64,
195                                     unif_init(0.1, 0.5,
196                                         dtype=tf.float64))
197             mv_mean = tf.get_variable('moving_mean',
198                                       params_shape,
199                                       tf.float64,
200                                       const_init(0.0, tf.float64),
201                                       trainable=False)
202             mv_var = tf.get_variable('moving_variance',
203                                      params_shape,
204                                      tf.float64,
205                                      const_init(1.0, tf.float64),
206                                      trainable=False)
207             # These ops will only be preformed when training
208             mean, variance = tf.nn.moments(x, [0], name='moments')
209             self._extra_train_ops.append(\
210                 assign_moving_average(mv_mean, mean, 0.99))
211             self._extra_train_ops.append(\
```

```
212                    assign_moving_average(mv_var, variance, 0.99))
213                mean, variance = \
214                    control_flow_ops.cond(self.is_training,
215                                          lambda: (mean, variance),
216                                          lambda: (mv_mean, mv_var))
217            y = tf.nn.batch_normalization(x, mean, variance,
218                                          beta, gamma, 1e-6)
219            y.set_shape(x.get_shape())
220            return y
221
222  def main():
223      tf.reset_default_graph()
224      with tf.Session() as sess:
225          tf.set_random_seed(1)
226          print "Begin to solve Allen-Cahn equation"
227          model = SolveAllenCahn(sess)
228          model.build()
229          model.train()
230          output = np.zeros((len(model.init_history), 3))
231          output[:, 0] = np.arange(len(model.init_history)) \
232                              * model.n_displaystep
233          output[:, 1] = model.loss_history
234          output[:, 2] = model.init_history
235          np.savetxt("./AllenCahn_d100.csv",
236                      output,
237                      fmt=['%d', '%.5e', '%.5e'],
238                      delimiter=",",
239                      header="step, loss function, " + \
240                              "target value, runtime",
241                      comments='')
242
243  if __name__ == '__main__':
244      np.random.seed(1)
245      main()
```

MATLAB code 1: A PYTHON code for the deep BSDE solver in Subsection 3.2 in the case of the PDE (35).

6.2   MATLAB source code for the Branching diffusion method used in Subsection 4.2

```
1  function Branching_Matlab()
2  % Parameters for the model
3    T = 0.3; t0 = 0; x0 = 0; d = 100; m = d;
4    mu = zeros(d,1); sigma = eye(d)*sqrt(2);
5    a = [0 2 0 -1]';
```

```matlab
 6 |   g = @(x) 1./(1+0.2*norm(x)^2)*1/2;
 7 |
 8 | % Parameters for the algorithm
 9 |   rng('default'); M = 10^7; beta = 1; p = [0 0.5 0 0.5]';
10 |
11 | % Branching method
12 |   tic;
13 |   [mn,sd] = MC_BM( mu, sigma, beta, p, a, t0, x0, T, g, M );
14 |   runtime = toc;
15 |
16 | % Output
17 |   disp(['Terminal condition: u(T,x0) = ' num2str(g(x0)) ';']);
18 |   disp(['Branching method: u(0,x0) ~ ' num2str(mn) ';']);
19 |   disp(['Estimated standard deviation: ' num2str(sd) ';']);
20 |   disp(['Estimated L2-appr. error = ' num2str(sd/sqrt(M)) ';']);
21 |   disp(['Elapsed runtime = ' num2str(runtime) ';']);
22 | end
23 |
24 | function [mn,sd] = MC_BM(mu, sigma, beta, p, a, t0, x0, T, g, M)
25 |   mn = 0; sd = 0;
26 |   for m=1:M
27 |     result = BM_Eval(mu, sigma, beta, p, a, t0, x0, T, g);
28 |     mn = mn + result;
29 |     sd = sd + result^2;
30 |   end
31 |   mn = mn/M; sd = sqrt( (sd - mn^2/M)/M );
32 | end
33 |
34 | function result = BM_Eval(mu, sigma, beta, p, a, t0, x0, T, g)
35 |   bp = BP(mu, sigma, beta, p, t0, x0, T);
36 |   result = 1;
37 |   for k=1:size(bp{1},2)
38 |     result = result * g( bp{1}(:,k) );
39 |   end
40 |   if norm(a-p) > 0
41 |     for k=1:length(a)
42 |       if p(k) > 0
43 |         result = result * ( a(k)/p(k) )^( bp{2}(k) );
44 |       elseif a(k) ~= 0
45 |         error('a(k) zero but p(k) non-zero');
46 |       end
47 |     end
48 |   end
49 | end
50 |
51 | function bp = BP(mu, sigma, beta, p, t0, x0, T)
52 |   bp = cell(2,1);
53 |   bp{2} = p*0;
54 |   tau = exprnd(1/beta);
```

```
55   new_t0 = min( tau + t0, T );
56   delta_t = new_t0 - t0;
57   m = size(sigma,2);
58   new_x0 = x0 + mu*delta_t + sigma*sqrt(delta_t)*randn(m,1);
59   if tau >= T - t0
60     bp{1} = new_x0;
61   else
62     [tmp,nonlinearity] = max(mnrnd(1,p));
63     bp{2}(nonlinearity) = bp{2}(nonlinearity) + 1;
64     for k=1:nonlinearity-1
65       tmp = BP(mu, sigma, beta, p, new_t0, new_x0, T);
66       bp{1} = [ bp{1} tmp{1} ];
67       bp{2} = bp{2} + tmp{2};
68     end
69   end
70 end
```

MATLAB code 2: A MATLAB code for the Branching diffusion method in the case of the PDE (35) based on $M = 10^7$ independent realizations.

6.3   MATLAB source code for the classical Monte Carlo method used in Subsection 4.3

```
1  function MonteCarlo_Matlab()
2    rng('default');
3
4  % Parameters for the model
5    d = 100;
6    g = @(x) log( (1+norm(x)^2)/2 );
7    T = 1;
8    M = 10^7;
9    t = 0;
10
11 % Classical Monte Carlo
12   tic;
13   MC = 0;
14   for m=1:M
15     dW = randn(1,d)*sqrt(T-t);
16     MC = MC + exp( - g( dW * sqrt(2) ) );
17   end
18   MC = -log(MC/M);
19   runtime = toc;
20
21 % Output
22   disp(['Solution: u(T,0) = ' num2str(g(0)) ';']);
23   disp(['Solution: u(0,0) = ' num2str(MC) ';']);
```

```
24    disp(['Time = ' num2str(runtime) ';']);
25  end
```

MATLAB code 3: A MATLAB code for a Monte Carlo method related to the PDE (36) based on $M = 10^7$ independent realizations.

References

[1] BELLMAN, R. *Dynamic programming*. Princeton Landmarks in Mathematics. Princeton University Press, Princeton, NJ, 2010. Reprint of the 1957 edition, With a new introduction by Stuart Dreyfus.

[2] BENDER, C., AND DENK, R. A forward scheme for backward SDEs. *Stochastic Processes and their Applications 117*, 12 (2007), 1793–1812.

[3] BENDER, C., SCHWEIZER, N., AND ZHUO, J. A primal-dual algorithm for BSDEs. *arXiv:1310.3694* (2014), 36 pages.

[4] BERGMAN, Y. Z. Option pricing with differential interest rates. *Review of Financial Studies 8*, 2 (1995), 475–500.

[5] BRIAND, P., AND LABART, C. Simulation of BSDEs by Wiener chaos expansion. *Ann. Appl. Probab. 24*, 3 (06 2014), 1129–1171.

[6] CHASSAGNEUX, J.-F. Linear multistep schemes for BSDEs. *SIAM J. Numer. Anal. 52*, 6 (2014), 2815–2836.

[7] CHASSAGNEUX, J.-F., AND RICHOU, A. Numerical simulation of quadratic BSDEs. *Ann. Appl. Probab. 26*, 1 (2016), 262–304.

[8] CRISAN, D., AND MANOLARAKIS, K. Solving backward stochastic differential equations using the cubature method: Application to nonlinear pricing. *SIAM Journal on Financial Mathematics 3*, 1 (2012), 534–571.

[9] DARBON, J., AND OSHER, S. Algorithms for overcoming the curse of dimensionality for certain Hamilton-Jacobi equations arising in control theory and elsewhere. *Res. Math. Sci. 3* (2016), Paper No. 19, 26.

[10] DEBNATH, L. *Nonlinear partial differential equations for scientists and engineers*, third ed. Birkhäuser/Springer, New York, 2012.

[11] E, W., HUTZENTHALER, M., JENTZEN, A., AND KRUSE, T. On full history recursive multilevel Picard approximations and numerical approximations for high-dimensional nonlinear parabolic partial differential equations and high-dimensional nonlinear backward stochastic differential equations. *arXiv:1607.03295* (2017), 46 pages.

[12] GOBET, E., LEMOR, J.-P., AND WARIN, X. A regression-based Monte Carlo method to solve backward stochastic differential equations. *Ann. Appl. Probab. 15*, 3 (2005), 2172–2202.

[13] GOBET, E., AND TURKEDJIEV, P. Linear regression MDP scheme for discrete backward stochastic differential equations under general conditions. *Math. Comp. 85*, 299 (2016), 1359–1391.

[14] GOBET, E., AND TURKEDJIEV, P. Adaptive importance sampling in least-squares Monte Carlo algorithms for backward stochastic differential equations. *Stochastic Process. Appl. 127*, 4 (2017), 1171–1203.

[15] GOODFELLOW, I., BENGIO, Y., AND COURVILLE, A. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[16] HAN, J., AND E, W. Deep Learning Approximation for Stochastic Control Problems. *arXiv:1611.07422* (2016), 9 pages.

[17] HENRY-LABORDÈRE, P. Counterparty risk valuation: a marked branching diffusion approach. *arXiv:1203.2369* (2012), 17 pages.

[18] HENRY-LABORDÈRE, P., OUDJANE, N., TAN, X., TOUZI, N., AND WARIN, X. Branching diffusion representation of semilinear PDEs and Monte Carlo approximation. *arXiv:1603.01727* (2016), 30 pages.

[19] HENRY-LABORDÈRE, P., TAN, X., AND TOUZI, N. A numerical algorithm for a class of BSDEs via the branching process. *Stochastic Process. Appl. 124*, 2 (2014), 1112–1140.

[20] HINTON, G. E., DENG, L., YU, D., DAHL, G., MOHAMED, A., JAITLY, N., SENIOR, A., VANHOUCKE, V., NGUYEN, P., SAINATH, T., AND KINGSBURY, B.

Deep neural networks for acoustic modeling in speech recognition. *Signal Processing Magazine 29* (2012), 82–97.

[21] Ioffe, S., and Szegedy, C. *Batch normalization: accelerating deep network training by reducing internal covariate shift.* Proceedings of The 32nd International Conference on Machine Learning (ICML), June 2015.

[22] Kingma, D., and Ba, J. *Adam: a method for stochastic optimization.* Proceedings of the International Conference on Learning Representations (ICLR), May 2015.

[23] Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems 25* (2012), 1097–1105.

[24] LeCun, Y., Bengio, Y., and Hinton, G. Deep learning. *Nature 521* (2015), 436–444.

[25] Pardoux, É., and Peng, S. Backward stochastic differential equations and quasilinear parabolic partial differential equations. In *Stochastic partial differential equations and their applications (Charlotte, NC, 1991)*, vol. 176 of *Lecture Notes in Control and Inform. Sci.* Springer, Berlin, 1992, pp. 200–217.

[26] Pardoux, É., and Peng, S. G. Adapted solution of a backward stochastic differential equation. *Systems Control Lett. 14*, 1 (1990), 55–61.

[27] Pardoux, E., and Tang, S. Forward-backward stochastic differential equations and quasilinear parabolic PDEs. *Probab. Theory Related Fields 114*, 2 (1999), 123–150.

[28] Peng, S. G. Probabilistic interpretation for systems of quasilinear parabolic partial differential equations. *Stochastics Stochastics Rep. 37*, 1-2 (1991), 61–74.