Deep splitting method for parabolic PDEs

Christian Beck[1], Sebastian Becker[2], Patrick Cheridito[3], Arnulf Jentzen[4], and Ariel Neufeld[5]

[1] Department of Mathematics, ETH Zurich, Switzerland, e-mail: christian.beck@math.ethz.ch

[2] ZENAI AG, Zurich, Switzerland, e-mail: sebastian.becker@zenai.ch

[3] RiskLab, Department of Mathematics, ETH Zurich, Switzerland, e-mail: patrick.cheridito@math.ethz.ch

[4] Seminar for Applied Mathematics, Department of Mathematics, ETH Zurich, Switzerland, e-mail: arnulf.jentzen@sam.math.ethz.ch

[5] Division of Mathematical Sciences, School of Physical and Mathematical Sciences, Nanyang Technological University, Singapore, e-mail: ariel.neufeld@ntu.edu.sg

July 9, 2019

Abstract

In this paper we introduce a numerical method for parabolic PDEs that combines operator splitting with deep learning. It divides the PDE approximation problem into a sequence of separate learning problems. Since the computational graph for each of the subproblems is comparatively small, the approach can handle extremely high-dimensional PDEs. We test the method on different examples from physics, stochastic control, and mathematical finance. In all cases, it yields very good results in up to 10,000 dimensions with short run times.

Contents

1 Introduction

In this paper we derive a numerical scheme for parabolic PDEs of the form

$$\tfrac{\partial}{\partial t}u(t,x) = F\big(x, u(t,x), \nabla_x u(t,x)\big) + \tfrac{1}{2}\operatorname{Trace}\big(\sigma(x)[\sigma(x)]^* \operatorname{Hess}_x u(t,x)\big) \tag{1}$$

with $u(0,x) = \varphi(x)$ for $(t,x) \in [0,T] \times \mathbb{R}^d$ where $F\colon \mathbb{R}^d \times \mathbb{R} \times \mathbb{R}^d \to \mathbb{R}$, $\sigma\colon \mathbb{R}^d \to \mathbb{R}^{d\times d}$, and $\varphi\colon \mathbb{R}^d \to \mathbb{R}$ are suitable continuous functions. Such PDEs describe various phenomena in nature, engineering, economics, and finance. They typically do not admit closed form solutions and, therefore, have to be solved numerically. In some applications, the dimension d can be high. For instance, in physics and engineering applications, $x \in \mathbb{R}^d$ typically models the coordinates of all components of a given system and in derivative pricing and optimal investment problems, d usually is the number of underlying assets. Many classical PDEs, such as the standard heat and Black–Scholes equations are linear.

Using the Feynman–Kac representation, their solutions can efficiently be approximated in high dimensions with simple Monte Carlo averages. But if constraints or frictions are taken into account or the PDE describes a control problem, the function F is no longer linear and equation (1) becomes much more challenging to solve for large d.

Numerical methods for PDEs have a long history. Classical approaches like finite differences and finite elements (see, e.g., [11, 69, 93]) are deterministic. In their standard form, they work well for $d = 1, 2$ and 3, but their complexity grows exponentially in d. To tackle higher dimensional problems, different simulation-based approaches have been developed that exploit a stochastic representation of the solution of the PDE. For instance, the articles [1, 5, 6, 10, 13, 14, 15, 16, 17, 18, 19, 20, 22, 24, 34, 35, 36, 37, 38, 39, 52, 68, 71, 72, 75, 76, 77, 82, 83, 85, 87, 88, 89, 94, 96] use BSDE representations of PDEs and introduce and study approximation methods based on recursive polynomial regressions, the articles [48, 50, 51, 80, 91, 95] suggest and analyze approximation methods based on branching diffusion processes, and the articles [27, 28, 55, 57, 58] propose and investigate full-history recursive multilevel Picard (MLP) approximation methods. Recently, numerical methods for possibly high-dimensional PDEs have been proposed in [26, 46] based on the idea to reformulate the PDE as a stochastic learning problem, which opens the door to the application of deep learning. We also refer, e.g., to [2, 3, 4, 8, 12, 29, 31, 32, 40, 47, 49, 53, 60, 73, 74, 78, 86, 90] for modifications and extensions of such deep learning based approximation methods for PDEs. There are also already a few rigorous mathematical results in the scientific literature which provide, at least partially, convergence analyses for such deep learning based approximation methods for PDEs. In particular, we refer, e.g., to [47, 90] for mathematical convergence results for such deep learning based PDE approximation methods with no information on the convergence speed and we refer, e.g., to [9, 30, 42, 54, 61, 67] for mathematical convergence and tractability results for such deep learning based PDE approximation methods with dimension-independent convergence rates and error constants which depend only polynomially on the dimension.

In this paper we develop a new deep learning method for parabolic PDEs that splits the differential operator into a linear and a nonlinear part. More precisely, we write

$$F\big(x, u(t,x), \nabla_x u(t,x)\big) = \langle \mu(x), \nabla_x u(t,x)\rangle_{\mathbb{R}^d} + f\big(x, u(t,x), \nabla_x u(t,x)\big) \tag{2}$$

for $(t,x) \in [0,T]\times\mathbb{R}^d$ and suitable continuous functions $\mu\colon \mathbb{R}^d \to \mathbb{R}^d$ and $f\colon \mathbb{R}^d\times\mathbb{R}\times\mathbb{R}^d \to \mathbb{R}$. This decomposition is not unique. But the idea is that μ is chosen such that the nonlinearity $f\big(x, u(t,x), \nabla_x u(t,x)\big)$ becomes small. Then we solve the PDE iteratively over small time intervals by approximating $f\big(x, u(t,x), \nabla_x u(t,x)\big)$ and using the Feynman–Kac representation locally. This requires a recursive computation of conditional expectations. We approximate them by formulating them as minimization problems that can be approached with deep learning. This decomposes the PDE approximation problem into a sequence of separate learning problems. Since the computational graph for each of the subproblems is comparatively small, the method works for very high-dimensional problems.

The rest of the paper is organized as follows. In Section 2 we introduce the framework and derive the deep splitting method. In Section 3 we test the approach on four different high-dimensional examples: a Hamilton–Jacobi–Bellman (HJB) equation, a nonlinear Black–Scholes equation, an Allen–Cahn equation, and a nonlinear heat equation.

2 Derivation of the proposed approximation algorithm

Let $T \in (0, \infty)$, $d \in \mathbb{N}$, let $\varphi\colon \mathbb{R}^d \to \mathbb{R}$ be a continuous function, let $\mu\colon \mathbb{R}^d \to \mathbb{R}^d$ and $\sigma\colon \mathbb{R}^d \to \mathbb{R}^{d\times d}$ be Lipschitz continuous functions, let $f\colon \mathbb{R}^d \times \mathbb{R} \times \mathbb{R}^d \to \mathbb{R}$ be a continuous and at most polynomially growing function, and let $u = (u(t,x))_{(t,x)\in[0,T]\times\mathbb{R}^d} \in C^{1,2}([0,T]\times\mathbb{R}^d, \mathbb{R})$ be a function with at most polynomially growing derivatives which satisfies for every $t \in [0,T]$, $x \in \mathbb{R}^d$ that $u(0,x) = \varphi(x)$ and

$$\begin{aligned}(\tfrac{\partial}{\partial t}u)(t,x) = f\big(x, u(t,x), (\nabla_x u)(t,x)\big) + \big\langle \mu(x), (\nabla_x u)(t,x)\big\rangle_{\mathbb{R}^d} \\ + \tfrac{1}{2}\operatorname{Trace}\big(\sigma(x)[\sigma(x)]^*(\operatorname{Hess}_x u)(t,x)\big).\end{aligned} \tag{3}$$

Our goal is to approximately calculate under suitable hypotheses the solution $u\colon [0,T]\times\mathbb{R}^d \to \mathbb{R}$ of the PDE (3).

2.1 Temporal discretization

In this subsection we discretize the PDE (3) in time by employing the splitting-up method (cf., for example, [41, 43, 44]) to obtain a semi-discrete approximation problem. To this end let $N \in \mathbb{N}$, $t_0, t_1, \ldots, t_N \in [0,T]$ be real numbers such that

$$0 = t_0 < t_1 < \ldots < t_N = T. \tag{4}$$

Observe that (3) yields that for every $t \in [0,T]$, $x \in \mathbb{R}^d$ it holds that

$$\begin{aligned}u(t,x) = \varphi(x) + \int_0^t f\big(x, u(s,x), (\nabla_x u)(s,x)\big)\,ds \\ + \int_0^t \Big[\tfrac{1}{2}\operatorname{Trace}\big(\sigma(x)[\sigma(x)]^*(\operatorname{Hess}_x u)(s,x)\big) + \big\langle \mu(x), (\nabla_x u)(s,x)\big\rangle_{\mathbb{R}^d}\Big]\,ds.\end{aligned} \tag{5}$$

Hence, we obtain that for every $n \in \{0,1,\ldots,N-1\}$, $t \in [t_n, t_{n+1}]$, $x \in \mathbb{R}^d$ it holds that

$$\begin{aligned}u(t,x) = u(t_n,x) + \int_{t_n}^t f\big(x, u(s,x), (\nabla_x u)(s,x)\big)\,ds \\ + \int_{t_n}^t \Big[\tfrac{1}{2}\operatorname{Trace}\big(\sigma(x)[\sigma(x)]^*(\operatorname{Hess}_x u)(s,x)\big) + \big\langle \mu(x), (\nabla_x u)(s,x)\big\rangle_{\mathbb{R}^d}\Big]\,ds.\end{aligned} \tag{6}$$

This illustrates for every $n \in \{0, 1, \dots, N-1\}$, $t \in [t_n, t_{n+1}]$, $x \in \mathbb{R}^d$ that

$$\begin{aligned} u(t,x) &\approx u(t_n,x) + \int_{t_n}^{t_{n+1}} f\big(x, u(t_n,x), (\nabla_x u)(t_n,x)\big)\, ds \\ &\quad + \int_{t_n}^{t} \Big[\tfrac{1}{2} \operatorname{Trace}\big(\sigma(x)[\sigma(x)]^*(\operatorname{Hess}_x u)(s,x)\big) + \big\langle \mu(x), (\nabla_x u)(s,x)\big\rangle_{\mathbb{R}^d}\Big]\, ds. \end{aligned} \tag{7}$$

This, in turn, suggests for every $n \in \{0, 1, \dots, N-1\}$, $t \in [t_n, t_{n+1}]$, $x \in \mathbb{R}^d$ that

$$\begin{aligned} u(t,x) &\approx u(t_n,x) + f\big(x, u(t_n,x), (\nabla_x u)(t_n,x)\big)\,(t_{n+1}-t_n) \\ &\quad + \int_{t_n}^{t} \Big[\tfrac{1}{2} \operatorname{Trace}\big(\sigma(x)[\sigma(x)]^*(\operatorname{Hess}_x u)(s,x)\big) + \big\langle \mu(x), (\nabla_x u)(s,x)\big\rangle_{\mathbb{R}^d}\Big]\, ds. \end{aligned} \tag{8}$$

To derive the splitting-up approximation let $U\colon (0,T] \times \mathbb{R}^d \to \mathbb{R}$ be a function which satisfies for every $n \in \{0, 1, \dots, N-1\}$ that $(U_t(x))_{(t,x)\in(t_n,t_{n+1}]\times\mathbb{R}^d} \in C^{1,2}((t_n, t_{n+1}] \times \mathbb{R}^d, \mathbb{R})$ has at most polynomially growing derivatives, which satisfies for every $x \in \mathbb{R}^d$ that $\int_0^T \|(\operatorname{Hess} U_s)(x)\|_{\mathbb{R}^{d\times d}} + \|(\nabla U_s)(x)\|_{\mathbb{R}^d}\, ds < \infty$, and which satisfies for every $n \in \{0, 1, \dots, N-1\}$, $t \in (t_n, t_{n+1}]$, $x \in \mathbb{R}^d$ that

$$\begin{aligned} U_t(x) &= u(t_n,x) + f\big(x, u(t_n,x), (\nabla_x u)(t_n,x)\big)\,(t_{n+1}-t_n) \\ &\quad + \int_{t_n}^{t} \Big[\tfrac{1}{2} \operatorname{Trace}\big(\sigma(x)[\sigma(x)]^*(\operatorname{Hess} U_s)(x)\big) + \big\langle \mu(x), (\nabla U_s)(x)\big\rangle_{\mathbb{R}^d}\Big]\, ds \end{aligned} \tag{9}$$

(cf., for example, Hairer et al. [45, Section 4.4], Deck & Kruse [21], Krylov [65, Chapter 8], and Krylov [66, Theorem 4.32] for existence, uniqueness, and regularity results for (9)). Note that (8) and (9) suggest for every $n \in \{1, 2, \dots, N\}$, $x \in \mathbb{R}^d$ that

$$U_{t_n}(x) \approx u(t_n, x). \tag{10}$$

Next let $V\colon [0,T] \times \mathbb{R}^d \to \mathbb{R}$ be a function which satisfies for every $n \in \{0, 1, \dots, N-1\}$ that $(V_t(x))_{(t,x)\in(t_n,t_{n+1}]\times\mathbb{R}^d} \in C^{1,2}((t_n, t_{n+1}] \times \mathbb{R}^d, \mathbb{R})$ has at most polynomially growing derivatives, which satisfies for every $x \in \mathbb{R}^d$ that $\int_0^T \|(\operatorname{Hess} V_s)(x)\|_{\mathbb{R}^{d\times d}} + \|(\nabla V_s)(x)\|_{\mathbb{R}^d}\, ds < \infty$, and which satisfies for every $n \in \{0, 1, \dots, N-1\}$, $t \in (t_n, t_{n+1}]$, $x \in \mathbb{R}^d$ that $V_0(x) = \varphi(x)$ and

$$\begin{aligned} V_t(x) &= V_{t_n}(x) + f\big(x, V_{t_n}(x), (\nabla V_{t_n})(x)\big)\,(t_{n+1}-t_n) \\ &\quad + \int_{t_n}^{t} \Big[\tfrac{1}{2} \operatorname{Trace}\big(\sigma(x)[\sigma(x)]^*(\operatorname{Hess} V_s)(x)\big) + \big\langle \mu(x), (\nabla V_s)(x)\big\rangle_{\mathbb{R}^d}\Big]\, ds \end{aligned} \tag{11}$$

(cf., for example, Hairer et al. [45, Section 4.4], Deck & Kruse [21], Krylov [65, Chapter 8], and Krylov [66, Theorem 4.32] for existence, uniqueness, and regularity results for (11)). Note that (9) and (11) suggest for every $n \in \{1, 2, \dots, N\}$, $x \in \mathbb{R}^d$ that

$$V_{t_n}(x) \approx U_{t_n}(x). \tag{12}$$

Combining this with (10), in turn, suggests for every $n \in \{0, 1, \ldots, N\}$, $x \in \mathbb{R}^d$ that

$$V_{t_n}(x) \approx u(t_n, x). \tag{13}$$

Observe that the function V is a specific splitting-up type approximation for the function u (cf., for example, [23, 41, 43, 44]). In the next subsection we derive a Feynman–Kac representation for V (cf., for example, Milstein & Tretyakov [84, Section 2]).

2.2 An approximate Feynman–Kac representation

In the following we introduce artificial stochastic processes in order to incorporate a Feynman–Kac type representation into (11). Let $(\Omega, \mathcal{F}, \mathbb{P}, (\mathcal{F}_t)_{t\in[0,T]})$ be a filtered probability space which fulfills the usual conditions, let $B\colon [0,T] \times \Omega \to \mathbb{R}^d$ be a standard $(\Omega, \mathcal{F}, \mathbb{P}, (\mathcal{F}_t)_{t\in[0,T]})$-Brownian motion, let $\xi\colon \Omega \to \mathbb{R}^d$ be an $\mathcal{F}_0/\mathcal{B}(\mathbb{R}^d)$-measurable function which satisfies for every $p \in (0,\infty)$ that $\mathbb{E}[\|\xi\|^p_{\mathbb{R}^d}] < \infty$, and let $Y\colon [0,T] \times \Omega \to \mathbb{R}^d$ be an $(\mathcal{F}_t)_{t\in[0,T]}$-adapted stochastic process with continuous sample paths which satisfies that for every $t \in [0,T]$ it holds $\mathbb{P}$-a.s. that

$$Y_t = \xi + \int_0^t \mu(Y_s)\, ds + \int_0^t \sigma(Y_s)\, dB_s. \tag{14}$$

Note that the assumption that for every $p \in (0,\infty)$ it holds that $\mathbb{E}[\|\xi\|^p_{\mathbb{R}^d}] < \infty$ and the assumption that $\mu\colon \mathbb{R}^d \to \mathbb{R}^d$ and $\sigma\colon \mathbb{R}^d \to \mathbb{R}^{d\times d}$ are Lipschitz continuous functions ensure that for every $p \in (0,\infty)$ it holds that

$$\sup_{t\in[0,T]} \mathbb{E}\big[\|Y_t\|^p_{\mathbb{R}^d}\big] < \infty \tag{15}$$

(cf., for example, Stroock [92, Section 1.2]). Moreover, observe that (11) implies that for every $n \in \{0, 1, \ldots, N-1\}$, $t \in (t_n, t_{n+1})$, $x \in \mathbb{R}^d$ it holds that

$$\tfrac{\partial}{\partial t}\big[V_t(x)\big] = \big\langle \mu(x), (\nabla V_t)(x)\big\rangle_{\mathbb{R}^d} + \tfrac{1}{2}\operatorname{Trace}\big(\sigma(x)[\sigma(x)]^*(\operatorname{Hess} V_t)(x)\big). \tag{16}$$

This, in turn, assures that for every $n \in \{0, 1, \ldots, N-1\}$, $t \in (T - t_{n+1}, T - t_n)$, $x \in \mathbb{R}^d$ it holds that

$$\tfrac{\partial}{\partial t}\big[V_{T-t}(x)\big] + \big\langle \mu(x), (\nabla V_{T-t})(x)\big\rangle_{\mathbb{R}^d} + \tfrac{1}{2}\operatorname{Trace}\big(\sigma(x)[\sigma(x)]^*(\operatorname{Hess} V_{T-t})(x)\big) = 0. \tag{17}$$

Next note that Itô's formula, the hypothesis that for every $n \in \{0, 1, \ldots, N-1\}$ it holds that $(V_t(x))_{(t,x)\in(t_n,t_{n+1}]\times\mathbb{R}^d} \in C^{1,2}((t_n, t_{n+1}] \times \mathbb{R}^d, \mathbb{R})$ (cf. (11)), and (14) guarantee that for every $n \in \{0, 1, \ldots, N-1\}$, $r, t \in [T - t_{n+1}, T - t_n)$ with $r < t$ it holds $\mathbb{P}$-a.s. that

$$\begin{aligned} V_{T-t}(Y_t) &= V_{T-r}(Y_r) + \int_r^t \big\langle (\nabla V_{T-s})(Y_s), \sigma(Y_s)\, dB_s\big\rangle_{\mathbb{R}^d} + \int_r^t \big(\tfrac{\partial}{\partial s}\big[V_{T-s}\big]\big)(Y_s)\, ds \\ &\quad + \int_r^t \tfrac{1}{2}\operatorname{Trace}\big(\sigma(Y_s)[\sigma(Y_s)]^*(\operatorname{Hess} V_{T-s})(Y_s)\big)\, ds \\ &\quad + \int_r^t \big\langle \mu(Y_s), (\nabla V_{T-s})(Y_s)\big\rangle_{\mathbb{R}^d}\, ds. \end{aligned} \tag{18}$$

Combining this with (17) implies that for every $n \in \{0, 1, \ldots, N-1\}$, $r, t \in [T-t_{n+1}, T-t_n)$ with $r < t$ it holds $\mathbb{P}$-a.s. that

$$V_{T-t}(Y_t) = V_{T-r}(Y_r) + \int_r^t \big\langle (\nabla V_{T-s})(Y_s), \sigma(Y_s)\, dB_s \big\rangle_{\mathbb{R}^d}. \tag{19}$$

Hence, we obtain that for every $n \in \{0, 1, \ldots, N-1\}$, $t \in (T-t_{n+1}, T-t_n)$ it holds $\mathbb{P}$-a.s. that

$$V_{T-t}(Y_t) = V_{t_{n+1}}(Y_{T-t_{n+1}}) + \int_{T-t_{n+1}}^t \big\langle (\nabla V_{T-s})(Y_s), \sigma(Y_s)\, dB_s \big\rangle_{\mathbb{R}^d}. \tag{20}$$

Furthermore, note that (15), the hypothesis that $\sigma\colon \mathbb{R}^d \to \mathbb{R}^{d\times d}$ is a Lipschitz continuous function, and the fact that for every $n \in \{0, 1, \ldots, N-1\}$ it holds that $(t_n, t_{n+1}] \times \mathbb{R}^d \ni (t, x) \mapsto (\nabla V_t)(x) \in \mathbb{R}^d$ is an at most polynomially growing function assure that for every $n \in \{0, 1, \ldots, N-1\}$, $t \in (T-t_{n+1}, T-t_n)$ it holds that

$$\int_{T-t_{n+1}}^t \mathbb{E}\Big[\big\| [\sigma(Y_s)]^* (\nabla V_{T-s})(Y_s) \big\|_{\mathbb{R}^d}^2 \Big]\, ds < \infty. \tag{21}$$

Therefore, we obtain that for every $n \in \{0, 1, \ldots, N-1\}$, $t \in (T-t_{n+1}, T-t_n)$ it holds $\mathbb{P}$-a.s. that

$$\mathbb{E}\bigg[\int_{T-t_{n+1}}^t \big\langle (\nabla V_{T-s})(Y_s), \sigma(Y_s)\, dB_s \big\rangle_{\mathbb{R}^d} \,\Big|\, \mathcal{F}_{T-t_{n+1}} \bigg] = 0. \tag{22}$$

This and (20) demonstrate that for every $n \in \{0, 1, \ldots, N-1\}$, $t \in [T-t_{n+1}, T-t_n)$ it holds $\mathbb{P}$-a.s. that

$$\mathbb{E}\Big[V_{T-t}(Y_t) \,\Big|\, \mathcal{F}_{T-t_{n+1}} \Big] = \mathbb{E}\Big[V_{t_{n+1}}(Y_{T-t_{n+1}}) \,\Big|\, \mathcal{F}_{T-t_{n+1}} \Big]. \tag{23}$$

The fact that for every $n \in \{0, 1, \ldots, N-1\}$ it holds that the function $\Omega \ni \omega \mapsto V_{t_{n+1}}(Y_{T-t_{n+1}}(\omega)) \in \mathbb{R}$ is $\mathcal{F}_{T-t_{n+1}}/\mathcal{B}(\mathbb{R})$-measurable hence implies that for every $n \in \{0, 1, \ldots, N-1\}$, $t \in [T-t_{n+1}, T-t_n)$ it holds $\mathbb{P}$-a.s. that

$$\mathbb{E}\Big[V_{T-t}(Y_t) \,\Big|\, \mathcal{F}_{T-t_{n+1}} \Big] = V_{t_{n+1}}(Y_{T-t_{n+1}}). \tag{24}$$

In the next step we combine the hypothesis that for every $n \in \{0, 1, \ldots, N-1\}$ it holds that $(V_t(x))_{(t,x)\in(t_n,t_{n+1}]\times\mathbb{R}^d} \in C^{1,2}((t_n, t_{n+1}] \times \mathbb{R}^d, \mathbb{R})$ has at most polynomially growing derivatives, (11), the fact that for every $\omega \in \Omega$ it holds that $[0, T] \ni t \mapsto Y_t(\omega) \in \mathbb{R}^d$ is a continuous function, and the hypothesis that for every $x \in \mathbb{R}^d$ it holds that $\int_0^T \|(\operatorname{Hess} V_s)(x)\|_{\mathbb{R}^{d\times d}} + \|(\nabla V_s)(x)\|_{\mathbb{R}^d}\, ds < \infty$ to obtain that for every $\omega \in \Omega$, $n \in \{0, 1, \ldots, N-1\}$ it holds that

$$\begin{aligned} &\limsup_{t \nearrow T-t_n} \Big| V_{T-t}(Y_t(\omega)) - \big[V_{t_n}(Y_{T-t_n}(\omega)) \\ &+ f\big(Y_{T-t_n}(\omega), V_{t_n}(Y_{T-t_n}(\omega)), (\nabla V_{t_n})(Y_{T-t_n}(\omega)) \big)\, (t_{n+1} - t_n) \big] \Big| = 0. \end{aligned} \tag{25}$$

In addition, note that the fact that $[0,T]\times\mathbb{R}^d \ni (t,x) \mapsto V_t(x) \in \mathbb{R}$ is an at most polynomially growing function and the fact that for every $p \in (0,\infty)$ it holds that $\sup_{t\in[0,T]} \mathbb{E}[\|Y_t\|_{\mathbb{R}^d}^p] < \infty$ guarantee that for every $t \in [0,T]$, $p \in (1,\infty)$ it holds that

$$\mathbb{E}\big[|V_{T-t}(Y_t)|^p\big] < \infty. \tag{26}$$

Combining (25) and, e.g., Hutzenthaler et al. [56, Proposition 4.5] therefore demonstrates that for every $n \in \{0,1,\ldots,N-1\}$ it holds that

$$\limsup_{t\nearrow T-t_n} \mathbb{E}\Big[\big|V_{T-t}(Y_t) - \big[V_{t_n}(Y_{T-t_n}) + f\big(Y_{T-t_n}, V_{t_n}(Y_{T-t_n}), (\nabla V_{t_n})(Y_{T-t_n})\big)\,(t_{n+1}-t_n)\big]\big|\Big] = 0. \tag{27}$$

This and (24) yield that for every $n \in \{0,1,\ldots,N-1\}$ it holds $\mathbb{P}$-a.s. that

$$\begin{aligned} &V_{t_{n+1}}(Y_{T-t_{n+1}}) \\ &= \mathbb{E}\Big[V_{t_n}(Y_{T-t_n}) + f\big(Y_{T-t_n}, V_{t_n}(Y_{T-t_n}), (\nabla V_{t_n})(Y_{T-t_n})\big)\,(t_{n+1}-t_n) \,\Big|\, \mathcal{F}_{T-t_{n+1}}\Big]. \end{aligned} \tag{28}$$

The tower property for conditional expectations therefore assures that for every $n \in \{0,1,\ldots,N-1\}$ it holds $\mathbb{P}$-a.s. that

$$\begin{aligned} &\mathbb{E}\Big[V_{t_{n+1}}(Y_{T-t_{n+1}}) \,\Big|\, Y_{T-t_{n+1}}\Big] \\ &= \mathbb{E}\Big[V_{t_n}(Y_{T-t_n}) + f\big(Y_{T-t_n}, V_{t_n}(Y_{T-t_n}), (\nabla V_{t_n})(Y_{T-t_n})\big)\,(t_{n+1}-t_n) \,\Big|\, Y_{T-t_{n+1}}\Big]. \end{aligned} \tag{29}$$

In addition, observe that the fact that for every $n \in \{0,1,\ldots,N-1\}$ it holds that the function $\Omega \ni \omega \mapsto Y_{T-t_{n+1}}(\omega) \in \mathbb{R}^d$ is $\mathfrak{S}(Y_{T-t_{n+1}})/\mathcal{B}(\mathbb{R}^d)$-measurable assures that for every $n \in \{0,1,\ldots,N-1\}$ it holds $\mathbb{P}$-a.s. that

$$V_{t_{n+1}}(Y_{T-t_{n+1}}) = \mathbb{E}\Big[V_{t_{n+1}}(Y_{T-t_{n+1}}) \,\Big|\, Y_{T-t_{n+1}}\Big]. \tag{30}$$

This and (29) imply that for every $n \in \{0,1,\ldots,N-1\}$ it holds $\mathbb{P}$-a.s. that

$$\begin{aligned} &V_{t_{n+1}}(Y_{T-t_{n+1}}) \\ &= \mathbb{E}\Big[V_{t_n}(Y_{T-t_n}) + f(Y_{T-t_n}, V_{t_n}(Y_{T-t_n}), (\nabla V_{t_n})(Y_{T-t_n}))\,(t_{n+1}-t_n) \,\Big|\, Y_{T-t_{n+1}}\Big]. \end{aligned} \tag{31}$$

Equation (31) constitutes the Feynman–Kac type representation we were aiming at. Note that in (3) the coefficient functions $\mu\colon \mathbb{R}^d \to \mathbb{R}^d$ and $\sigma\colon \mathbb{R}^d \to \mathbb{R}^{d\times d}$ and the nonlinearity $f\colon \mathbb{R}^d \times \mathbb{R} \times \mathbb{R}^d \to \mathbb{R}$ do not depend on $t \in [0,T]$. The above sketched derivation, however, could under suitable assumptions also be performed in the case of time-dependent coefficient functions and nonlinearities as the classical Feynman–Kac formula holds also in the time-dependent case. In the following subsection we employ the factorization lemma (cf., for example, Klenke [63, Corollary 1.97]) and the L^2-minimality property of conditional expectations (cf., for example, Klenke [63, Corollary 8.17]) to reformulate (31) as recursive minimization problems.

2.3 Formulation as recursive minimization problems

In this subsection we reformulate (31) as recursive minimization problems. For this we combine the fact that for every $n \in \{0, 1, \ldots, N\}$ it holds that $\mathbb{R}^d \ni x \mapsto V_{t_n}(x) + f(x, V_{t_n}(x), (\nabla V_{t_n})(x))(t_{n+1} - t_n) \in \mathbb{R}$ is an at most polynomially growing function and the fact that for every $p \in (0, \infty)$ it holds that $\sup_{t\in[0,T]} \mathbb{E}[\|Y_t\|^p_{\mathbb{R}^d}] < \infty$ to obtain that for every $n \in \{0, 1, \ldots, N-1\}$ it holds that

$$\mathbb{E}\Big[\big|V_{t_n}(Y_{T-t_n}) + f\big(Y_{T-t_n}, V_{t_n}(Y_{T-t_n}), (\nabla V_{t_n})(Y_{T-t_n})\big)\,(t_{n+1} - t_n)\big|^2\Big] < \infty. \tag{32}$$

The factorization lemma, the L^2-minimality property for conditional expectations, e.g., in Klenke [63, Corollary 8.17] (with $X = \Omega \ni \omega \mapsto V_{t_n}(Y_{T-t_n}(\omega)) + f(Y_{T-t_n}(\omega), V_{t_n}(Y_{T-t_n}(\omega)), (\nabla V_{t_n})(Y_{T-t_n}(\omega)))\,(t_{n+1} - t_n) \in \mathbb{R}$, $\mathcal{F} = \mathfrak{S}(Y_{T-t_{n+1}})$, $\mathcal{A} = \mathcal{F}$ in the notation of [63, Corollary 8.17]), the fact that for every $n \in \{0, 1, \ldots, N-1\}$ it holds that $\mathbb{R}^d \ni x \mapsto V_{t_{n+1}}(x) \in \mathbb{R}$ is a continuous function, and (31) hence imply that for every $n \in \{0, 1, \ldots, N-1\}$ it holds that

$$\begin{aligned}(V_{t_{n+1}}(x))_{x\in\mathrm{supp}(Y_{T-t_{n+1}}(\mathbb{P}))} = \operatorname*{argmin}_{v\in C(\mathrm{supp}(Y_{T-t_{n+1}}(\mathbb{P})),\mathbb{R})} \mathbb{E}\Big[\big|v(Y_{T-t_{n+1}}) - \big[V_{t_n}(Y_{T-t_n}) \\ + f\big(Y_{T-t_n}, V_{t_n}(Y_{T-t_n}), (\nabla V_{t_n})(Y_{T-t_n})\big)\,(t_{n+1} - t_n)\big]\big|^2\Big]. \end{aligned} \tag{33}$$

Therefore, we obtain that for every $n \in \{1, 2, \ldots, N\}$ it holds that

$$\begin{aligned}(V_{t_n}(x))_{x\in\mathrm{supp}(Y_{T-t_n}(\mathbb{P}))} = \operatorname*{argmin}_{v\in C(\mathrm{supp}(Y_{T-t_n}(\mathbb{P})),\mathbb{R})} \mathbb{E}\Big[\big|v(Y_{T-t_n}) - \big[V_{t_{n-1}}(Y_{T-t_{n-1}}) \\ + f\big(Y_{T-t_{n-1}}, V_{t_{n-1}}(Y_{T-t_{n-1}}), (\nabla V_{t_{n-1}})(Y_{T-t_{n-1}})\big)\,(t_n - t_{n-1})\big]\big|^2\Big]. \end{aligned} \tag{34}$$

In the following subsections we approximate for every $n \in \{1, 2, \ldots, N\}$ the function $\mathrm{supp}(Y_{T-t_n}(\mathbb{P})) \ni x \mapsto V_{t_n}(x) \in \mathbb{R}$ by suitable deep artificial neural networks.

2.4 Deep artificial neural network approximations

In this subsection we employ for every $n \in \{1, 2, \ldots, N\}$ suitable approximations for the function

$$\mathrm{supp}(Y_{T-t_n}(\mathbb{P})) \ni x \mapsto V_{t_n}(x) \in \mathbb{R}. \tag{35}$$

More specifically, let $\nu \in \mathbb{N}$ and let $\mathbb{V}_n = (\mathbb{V}_n(\theta, x))_{(\theta,x)\in\mathbb{R}^\nu\times\mathbb{R}^d} \colon \mathbb{R}^\nu \times \mathbb{R}^d \to \mathbb{R}$, $n \in \{0, 1, \ldots, N\}$, be continuously differentiable functions which satisfy for every $\theta \in \mathbb{R}^\nu$, $x \in \mathbb{R}^d$ that $\mathbb{V}_0(\theta, x) = \varphi(x)$. For every $n \in \{1, 2, \ldots, N\}$, $x \in \mathrm{supp}(Y_{T-t_n}(\mathbb{P}))$ we think for *suitable* $\theta \in \mathbb{R}^\nu$ of $\mathbb{V}_n(\theta, x) \in \mathbb{R}$ as an appropriate approximation

$$\mathbb{V}_n(\theta, x) \approx V_{t_n}(x) \tag{36}$$

of $V_{t_n}(x)$. We suggest to choose the functions $\mathbb{V}_n\colon \mathbb{R}^\nu \times \mathbb{R}^d \to \mathbb{R}$, $n \in \{1, 2, \ldots, N\}$, as artificial neural networks (cf., for example, [7, 70]). For example, for every $k \in \mathbb{N}$ let $\mathcal{L}_k\colon \mathbb{R}^k \to \mathbb{R}^k$ be the multidimensional version of the standard logistic function which satisfies for every $x = (x_1, x_2, \ldots, x_k) \in \mathbb{R}^k$ that

$$\mathcal{L}_k(x) = \left(\frac{\exp(x_1)}{\exp(x_1)+1}, \frac{\exp(x_2)}{\exp(x_2)+1}, \ldots, \frac{\exp(x_k)}{\exp(x_k)+1} \right), \tag{37}$$

for every $\theta = (\theta_1, \theta_2, \ldots, \theta_\nu) \in \mathbb{R}^\nu$, $v \in \mathbb{N}_0 = \{0\} \cup \mathbb{N}$, $k, l \in \mathbb{N}$ with $v + l(k+1) \leq \nu$ let $A^{\theta,v}_{k,l}\colon \mathbb{R}^k \to \mathbb{R}^l$ be the function which satisfies for every $x = (x_1, x_2, \ldots, x_k)$ that

$$A^{\theta,v}_{k,l}(x) = \begin{pmatrix} \theta_{v+1} & \theta_{v+2} & \ldots & \theta_{v+k} \\ \theta_{v+k+1} & \theta_{v+k+2} & \ldots & \theta_{v+2k} \\ \theta_{v+2k+1} & \theta_{v+2k+2} & \ldots & \theta_{v+3k} \\ \vdots & \vdots & \vdots & \vdots \\ \theta_{v+(l-1)k+1} & \theta_{v+(l-1)k+2} & \ldots & \theta_{v+lk} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_k \end{pmatrix} + \begin{pmatrix} \theta_{v+lk+1} \\ \theta_{v+lk+2} \\ \theta_{v+lk+3} \\ \vdots \\ \theta_{v+lk+l} \end{pmatrix}, \tag{38}$$

let $s \in \{3, 4, 5, 6, \ldots\}$, assume that $s(N+1)d(d+1) \leq \nu$, and let $\mathbb{V}_n\colon \mathbb{R}^\nu \times \mathbb{R}^d \to \mathbb{R}$, $n \in \{0, 1, \ldots, N\}$, be the functions which satisfy for every $n \in \{1, 2, \ldots, N\}$, $\theta \in \mathbb{R}^\nu$, $x \in \mathbb{R}^d$ that $\mathbb{V}_0(\theta, x) = \varphi(x)$ and

$$\begin{aligned} &\mathbb{V}_n(\theta, x) = \\ &\big(A^{\theta,(sn+s-1)d(d+1)}_{d,1} \circ \mathcal{L}_d \circ A^{\theta,(sn+s-2)d(d+1)}_{d,d} \circ \ldots \circ \mathcal{L}_d \circ A^{\theta,(sn+1)d(d+1)}_{d,d} \circ \mathcal{L}_d \circ A^{\theta,snd(d+1)}_{d,d} \big)(x). \end{aligned} \tag{39}$$

For every $n \in \{1, 2, \ldots, N\}$ the function $\mathbb{V}_n\colon \mathbb{R}^\nu \times \mathbb{R}^d \to \mathbb{R}$ in (39) describes an artificial neural network with $s+1$ layers (1 input layer with d neurons, $s-1$ hidden layers with d neurons each, and 1 output layer with 1 neuron) and multidimensional versions of the standard logistic function as activation functions (see (37)). In our numerical simulations we use as activation functions multidimensional versions of the standard rectifier function instead of multidimensional versions of the standard logistic function. To avoid technical difficulties, which would arise due to the fact that the rectifier function is not everywhere differentiable, we restrict ourselves in (37)–(39) in this illustrative section to the smooth standard logistic function instead of the rectifier function.

2.5 Stochastic gradient descent based minimization

We intend to find *suitable* $\theta^1, \theta^2, \ldots, \theta^N \in \mathbb{R}^\nu$ in (36) by recursive minimization. More precisely, we intend to find for $n \in \{1, 2, \ldots, N\}$, $\theta^0, \theta^1, \ldots, \theta^{n-1} \in \mathbb{R}^\nu$ a suitable $\theta^n \in \mathbb{R}^\nu$ as an approximate minimizer of the function

$$\begin{aligned} \mathbb{R}^\nu \ni \theta \mapsto \mathbb{E}\Big[\big| \mathbb{V}_n(\theta, Y_{T-t_n}) - \big[\mathbb{V}_{n-1}(\theta^{n-1}, Y_{T-t_{n-1}}) \\ + f\big(Y_{T-t_{n-1}}, \mathbb{V}_{n-1}(\theta^{n-1}, Y_{T-t_{n-1}}), (\nabla_x \mathbb{V}_{n-1})(\theta^{n-1}, Y_{T-t_{n-1}})\big)(t_n - t_{n-1}) \big] \big|^2 \Big] \in \mathbb{R} \end{aligned} \tag{40}$$

(cf. (34) and (36) above). To this end let $B^m\colon [0,T]\times\Omega\to\mathbb{R}^d$, $m\in\mathbb{N}_0$, be i.i.d. standard $(\Omega,\mathcal{F},\mathbb{P},(\mathcal{F}_t)_{t\in[0,T]})$-Brownian motions, let $\xi^m\colon\Omega\to\mathbb{R}^d$, $m\in\mathbb{N}_0$, be i.i.d. $\mathcal{F}_0/\mathcal{B}(\mathbb{R}^d)$-measurable functions, for every $m\in\mathbb{N}_0$ let $Y^m\colon[0,T]\times\Omega\to\mathbb{R}^d$ be an $(\mathcal{F}_t)_{t\in[0,T]}$-adapted stochastic process with continuous sample paths which satisfies that for every $t\in[0,T]$ it holds $\mathbb{P}$-a.s. that

$$Y^m_t=\xi^m+\int_0^t\mu(Y^m_s)\,ds+\int_0^t\sigma(Y^m_s)\,dB^m_s, \tag{41}$$

let $\gamma\in(0,\infty)$, $M\in\mathbb{N}$, and let $\vartheta^n=(\vartheta^n_m)_{m\in\mathbb{N}_0}\colon\mathbb{N}_0\times\Omega\to\mathbb{R}^\nu$, $n\in\{0,1,\ldots,N\}$, be stochastic processes which satisfy for every $n\in\{1,2,\ldots,N\}$, $m\in\mathbb{N}_0$ that

$$\begin{aligned}\vartheta^n_{m+1}=\vartheta^n_m-2\gamma\cdot(\nabla_\theta\mathbb{V}_n)(\vartheta^n_m,Y^m_{T-t_n})\cdot\Big[&\mathbb{V}_n(\vartheta^n_m,Y^m_{T-t_n})-\mathbb{V}_{n-1}(\vartheta^{n-1}_M,Y^m_{T-t_{n-1}})\\&-f\big(Y^m_{T-t_{n-1}},\mathbb{V}_{n-1}(\vartheta^{n-1}_M,Y^m_{T-t_{n-1}}),(\nabla_x\mathbb{V}_{n-1})(\vartheta^{n-1}_M,Y^m_{T-t_{n-1}})\big)\,(t_n-t_{n-1})\Big].\end{aligned} \tag{42}$$

2.6 Discretization of the auxiliary stochastic process Y

Equation (42) provides an implementable numerical algorithm in the special case where one can simulate exactly from the solution processes $Y^m\colon[0,T]\times\Omega\to\mathbb{R}^d$, $m\in\mathbb{N}_0$, of the SDEs in (41) (cf. also (14) above). In the case where it is not possible to simulate exactly from the solution processes $Y^m\colon[0,T]\times\Omega\to\mathbb{R}^d$, $m\in\mathbb{N}_0$, of the SDEs in (41), one can employ a numerical approximation method for SDEs, say, the Euler–Maruyama scheme, to approximatively simulate from the solution processes $Y^m\colon[0,T]\times\Omega\to\mathbb{R}^d$, $m\in\mathbb{N}_0$, of the SDEs in (41). This is the subject of this subsection. More formally, note that (41) implies that for every $m\in\mathbb{N}_0$, $r,t\in[0,T]$ with $r\leq t$ it holds $\mathbb{P}$-a.s. that

$$Y^m_t=Y^m_r+\int_r^t\mu(Y^m_s)\,ds+\int_r^t\sigma(Y^m_s)\,dB^m_s. \tag{43}$$

Hence, we obtain that for every $m\in\mathbb{N}_0$, $n\in\{0,1,\ldots,N-1\}$ it holds $\mathbb{P}$-a.s. that

$$Y^m_{T-t_n}=Y^m_{T-t_{n+1}}+\int_{T-t_{n+1}}^{T-t_n}\mu(Y^m_s)\,ds+\int_{T-t_{n+1}}^{T-t_n}\sigma(Y^m_s)\,dB^m_s. \tag{44}$$

This shows that for every $m\in\mathbb{N}_0$, $n\in\{0,1,\ldots,N-1\}$ it holds $\mathbb{P}$-a.s. that

$$Y^m_{T-t_{N-(n+1)}}=Y^m_{T-t_{N-n}}+\int_{T-t_{N-n}}^{T-t_{N-(n+1)}}\mu(Y^m_s)\,ds+\int_{T-t_{N-n}}^{T-t_{N-(n+1)}}\sigma(Y^m_s)\,dB^m_s. \tag{45}$$

Next we introduce suitable real numbers which allow us to formulate (45) in a more compact way. More formally, let $\tau_n\in[0,T]$, $n\in\{0,1,\ldots,N\}$, be the real numbers which satisfy for every $n\in\{0,1,\ldots,N\}$ that

$$\tau_n=T-t_{N-n}. \tag{46}$$

Observe that (4) ensures that

$$0 = \tau_0 < \tau_1 < \cdots < \tau_N = T. \tag{47}$$

Moreover, note that (45) and (46) demonstrate that for every $m \in \mathbb{N}_0$, $n \in \{0, 1, \ldots, N-1\}$ it holds $\mathbb{P}$-a.s. that

$$Y^m_{\tau_{n+1}} = Y^m_{\tau_n} + \int_{\tau_n}^{\tau_{n+1}} \mu(Y^m_s)\, ds + \int_{\tau_n}^{\tau_{n+1}} \sigma(Y^m_s)\, dB^m_s. \tag{48}$$

This suggests for every $m \in \mathbb{N}_0$, $n \in \{0, 1, \ldots, N-1\}$ that

$$Y^m_{\tau_{n+1}} \approx Y^m_{\tau_n} + \mu(Y^m_{\tau_n})\,(\tau_{n+1} - \tau_n) + \sigma(Y^m_{\tau_n})\,(B^m_{\tau_{n+1}} - B^m_{\tau_n}). \tag{49}$$

Based on (49) we now introduce suitable Euler–Maruyama approximations for the solution processes $Y^m \colon [0,T] \times \Omega \to \mathbb{R}^d$, $m \in \mathbb{N}_0$, of the SDEs in (41). More formally, for every $m \in \mathbb{N}_0$ let $\mathcal{Y}^m = (\mathcal{Y}^m_n)_{n \in \{0,1,\ldots,N\}} \colon \{0, 1, \ldots, N\} \times \Omega \to \mathbb{R}^d$ be the stochastic process which satisfies for every $n \in \{0, 1, \ldots, N-1\}$ that

$$\mathcal{Y}^m_{n+1} = \mathcal{Y}^m_n + \mu(\mathcal{Y}^m_n)\,(\tau_{n+1} - \tau_n) + \sigma(\mathcal{Y}^m_n)\,(B^m_{\tau_{n+1}} - B^m_{\tau_n}). \tag{50}$$

Observe that (46), (49), and (50) suggest for every $m \in \mathbb{N}_0$, $n \in \{0, 1, \ldots, N\}$ that

$$\mathcal{Y}^m_n \approx Y^m_{\tau_n} = Y^m_{T - t_{N-n}}. \tag{51}$$

This, in turn, suggests for every $m \in \mathbb{N}_0$, $n \in \{0, 1, \ldots, N\}$ that

$$Y^m_{T-t_n} \approx \mathcal{Y}^m_{N-n}. \tag{52}$$

In the next step we employ (52) to derive approximations of the stochastic processes $\vartheta^n \colon N_0 \times \Omega \to \mathbb{R}^\nu$, $n \in \{0, 1, \ldots, N\}$, in (42) which are also implementable in the case where one cannot simulate exactly from the solution processes $Y^m \colon [0,T] \times \Omega \to \mathbb{R}^d$, $m \in \mathbb{N}_0$, of the SDEs in (41). More precisely, let $\Theta^n = (\Theta^n_m)_{m \in \mathbb{N}_0} \colon \mathbb{N}_0 \times \Omega \to \mathbb{R}$, $n \in \{0, 1, \ldots, N\}$, be stochastic processes which satisfy for every $n \in \{1, 2, \ldots, N\}$, $m \in \mathbb{N}_0$ that

$$\begin{aligned} \Theta^n_{m+1} = \Theta^n_m - 2\gamma \cdot (\nabla_\theta \mathbb{V}_n)(\Theta^n_m, \mathcal{Y}^m_{N-n}) \cdot \Big[& \mathbb{V}_n(\Theta^n_m, \mathcal{Y}^m_{N-n}) - \mathbb{V}_{n-1}(\Theta^{n-1}_M, \mathcal{Y}^m_{N-n+1}) \\ & - f\big(\mathcal{Y}^m_{N-n+1}, \mathbb{V}_{n-1}(\Theta^{n-1}_M, \mathcal{Y}^m_{N-n+1}), (\nabla_x \mathbb{V}_{n-1})(\Theta^{n-1}_M, \mathcal{Y}^m_{N-n+1})\big)\,(t_n - t_{n-1})\Big]. \end{aligned} \tag{53}$$

Note that (42), (52), and (53) suggest for every $n \in \{1, 2, \ldots, N\}$ and every sufficiently large $m \in \mathbb{N}_0$ that

$$\Theta^n_m \approx \vartheta^n_m. \tag{54}$$

In the following two subsections (Subsection 2.7 and Subsection 2.8) we merge the above derivations to precisely formulate the proposed approximation algorithm, first, in a special case (Subsection 2.7) and, thereafter, in the general case (Subsection 2.8).

2.7 Description of the algorithm in a special case

In this subsection we depict the deep splitting method in the special case where the standard Euler–Maruyama scheme (cf., e.g., [64, 79, 81]) is the used approximation scheme for discretizing (41) (cf. (50)) and where the plain vanilla stochastic gradient descent method with constant learning rate $\gamma \in (0, \infty)$ and batch size 1 is the used minimization algorithm. A more general description of the deep splitting method, which allows to include more advanced machine learning approximation techniques such as batch normalization (cf., for instance, Ioffe & Szegedy [59]) and the Adam optimizer (cf., for example, Kingma & Ba [62]), can be found in Subsection 2.8 below.

Framework 2.1 (Special case)**.** *Let $T, \gamma \in (0, \infty)$, $d, N, M \in \mathbb{N}$, $\varphi \in C^2(\mathbb{R}^d, \mathbb{R})$, $s \in \{3, 4, 5, \ldots\}$, $\nu = s(N+1)d(d+1)$, $t_0, t_1, \ldots, t_N \in [0, T]$ satisfy*

$$0 = t_0 < t_1 < \ldots < t_N = T, \tag{55}$$

let $\tau_0, \tau_1, \ldots, \tau_n \in [0, T]$ satisfy for every $n \in \{0, 1, \ldots, N\}$ that $\tau_n = T - t_{N-n}$, let $f \colon \mathbb{R}^d \times \mathbb{R} \times \mathbb{R}^d \to \mathbb{R}$, $\mu \colon \mathbb{R}^d \to \mathbb{R}^d$, $\sigma \colon \mathbb{R}^d \to \mathbb{R}^{d \times d}$ be continuous functions, let $(\Omega, \mathcal{F}, \mathbb{P}, (\mathcal{F}_t)_{t \in [0,T]})$ be a filtered probability space, let $\xi^m \colon \Omega \to \mathbb{R}^d$, $m \in \mathbb{N}_0$, be i.i.d. $\mathcal{F}_0/\mathcal{B}(\mathbb{R}^d)$-measurable random variables, let $B^m \colon [0, T] \times \Omega \to \mathbb{R}^d$, $m \in \mathbb{N}_0$, be i.i.d. standard $(\Omega, \mathcal{F}, \mathbb{P}, (\mathcal{F}_t)_{t \in [0,T]})$-Brownian motions, for every $m \in \mathbb{N}_0$ let $\mathcal{Y}^m \colon \{0, 1, \ldots, N\} \times \Omega \to \mathbb{R}^d$ be the stochastic process which satisfies for every $k \in \{0, 1, \ldots, N-1\}$ that $\mathcal{Y}^m_0 = \xi^m$ and

$$\mathcal{Y}^m_{k+1} = \mathcal{Y}^m_k + \mu(\mathcal{Y}^m_k)\,(\tau_{k+1} - \tau_k) + \sigma(\mathcal{Y}^m_k)\,(B^m_{\tau_{k+1}} - B^m_{\tau_k}), \tag{56}$$

let $\mathcal{L}_d \colon \mathbb{R}^d \to \mathbb{R}^d$ be the function which satisfies for every $x = (x_1, x_2, \ldots, x_d) \in \mathbb{R}^d$ that

$$\mathcal{L}_d(x) = \left(\frac{\exp(x_1)}{\exp(x_1) + 1}, \frac{\exp(x_2)}{\exp(x_2) + 1}, \ldots, \frac{\exp(x_d)}{\exp(x_d) + 1} \right), \tag{57}$$

for every $\theta = (\theta_1, \theta_2, \ldots, \theta_\nu) \in \mathbb{R}^\nu$, $k, l \in \mathbb{N}$, $v \in \mathbb{N}_0 = \{0\} \cup \mathbb{N}$ with $v + l(k+1) \leq \nu$ let $A^{\theta,v}_{k,l} \colon \mathbb{R}^k \to \mathbb{R}^l$ be the function which satisfies for every $x = (x_1, x_2, \ldots, x_k) \in \mathbb{R}^k$ that

$$A^{\theta,v}_{k,l}(x) = \left(\theta_{v+kl+1} + \left[\sum_{i=1}^{k} x_i\, \theta_{v+i} \right], \ldots, \theta_{v+kl+l} + \left[\sum_{i=1}^{k} x_i\, \theta_{v+(l-1)k+i} \right] \right), \tag{58}$$

let $\mathbb{V}_n \colon \mathbb{R}^\nu \times \mathbb{R}^d \to \mathbb{R}$, $n \in \{0, 1, \ldots, N\}$, be the functions which satisfy for every $n \in \{1, 2, \ldots, N\}$, $\theta \in \mathbb{R}^\nu$, $x \in \mathbb{R}^d$ that $\mathbb{V}_0(\theta, x) = \varphi(x)$ and

$$\begin{aligned} &\mathbb{V}_n(\theta, x) = \\ &\big(A^{\theta,(sn+s-1)d(d+1)}_{d,1} \circ \mathcal{L}_d \circ A^{\theta,(sn+s-2)d(d+1)}_{d,d} \circ \ldots \circ \mathcal{L}_d \circ A^{\theta,(sn+1)d(d+1)}_{d,d} \circ \mathcal{L}_d \circ A^{\theta,snd(d+1)}_{d,d} \big)(x), \end{aligned} \tag{59}$$

let $\Theta^n\colon \mathbb{N}_0\times\Omega\to\mathbb{R}^\nu$, $n\in\{0,1,\ldots,N\}$, *be stochastic processes, for every* $n\in\{1,2,\ldots,N\}$, $m\in\mathbb{N}_0$ *let* $\phi^{n,m}\colon\mathbb{R}^\nu\times\Omega\to\mathbb{R}$ *be the function which satisfies for every* $\theta\in\mathbb{R}^\nu$, $\omega\in\Omega$ *that*

$$\begin{aligned}\phi^{n,m}(\theta,\omega) &= \Big[\mathbb{V}_n\big(\theta,\mathcal{Y}^m_{N-n}(\omega)\big) - \mathbb{V}_{n-1}(\Theta^{n-1}_M(\omega),\mathcal{Y}^m_{N-n+1}(\omega)) - (t_n-t_{n-1})\\ &\cdot f\big(\mathcal{Y}^m_{N-n+1}(\omega),\mathbb{V}_{n-1}(\Theta^{n-1}_M(\omega),\mathcal{Y}^m_{N-n+1}(\omega)),(\nabla_x\mathbb{V}_{n-1})(\Theta^{n-1}_M(\omega),\mathcal{Y}^m_{N-n+1}(\omega))\big)\Big]^2,\end{aligned} \tag{60}$$

for every $n\in\{1,2,\ldots,N\}$, $m\in\mathbb{N}_0$ *let* $\Phi^{n,m}\colon\mathbb{R}^\nu\times\Omega\to\mathbb{R}^\nu$ *be the function which satisfies for every* $\theta\in\mathbb{R}^\nu$, $\omega\in\Omega$ *that* $\Phi^{n,m}(\theta,\omega)=(\nabla_\theta\phi^{n,m})(\theta,\omega)$, *and assume for every* $n\in\{1,2,\ldots,N\}$, $m\in\mathbb{N}_0$ *that*

$$\Theta^n_{m+1}=\Theta^n_m-\gamma\cdot\Phi^{n,m}(\Theta^n_m). \tag{61}$$

In the setting of Framework 2.1 we think under suitable hypotheses for sufficiently large $N,M\in\mathbb{N}$, sufficiently small $\gamma\in(0,\infty)$, every $n\in\{0,1,\ldots N\}$, and every $x\in\mathbb{R}^d$ of $\mathbb{V}_n(\Theta^n_M,x)\colon\Omega\to\mathbb{R}$ as a suitable approximation

$$\mathbb{V}_n(\Theta^n_M,x)\approx u(t_n,x) \tag{62}$$

of $u(t_n,x)$ where $u=(u(t,x))_{(t,x)\in[0,T]\times\mathbb{R}^d}\in C^{1,2}([0,T]\times\mathbb{R}^d,\mathbb{R})$ is a function with at most polynomially growing derivatives which satisfies for every $t\in[0,T]$, $x\in\mathbb{R}^d$ that $u(0,x)=\varphi(x)$ and

$$\begin{aligned}(\tfrac{\partial}{\partial t}u)(t,x) &= f\big(x,u(t,x),(\nabla_x u)(t,x)\big)+\big\langle\mu(x),(\nabla_x u)(t,x)\big\rangle_{\mathbb{R}^d}\\ &\quad+\tfrac{1}{2}\operatorname{Trace}\big(\sigma(x)[\sigma(x)]^*(\operatorname{Hess}_x u)(t,x)\big)\end{aligned} \tag{63}$$

(cf. (3), (13), and (36)).

2.8 Description of the algorithm in the general case

In this subsection we present in Framework 2.2 below a general formulation of the deep splitting method which includes the deep splitting method derived in Subsections 2.1–2.7 above as a special case but also enables us to incorporate other minimization algorithms (cf. (68) below and, e.g., E et al. [26, Subsections 3.2, 5.1, and 5.2]) such as the Adam optimizer (cf. Kingma & Ba [62] and (72)–(73)) than just the plain vanilla stochastic gradient descent method (see, e.g., (61) in Framework 2.1 in Subsection 2.7 above). Moreover, Framework 2.2 also enables us to include more advanced machine learning techniques like batch normalization (cf. Ioffe & Szegedy [59] and (67) below). In Section 3 below the general description in Framework 2.2 is illustrated by means of several examples.

Framework 2.2 (General case)**.** *Let* $T\in(0,\infty)$, $N,d,\varrho,\nu,\varsigma\in\mathbb{N}$, $(M_n)_{n\in\{0,1,\ldots,N\}}\subseteq\mathbb{N}$, $(J_m)_{m\in\mathbb{N}_0}\subseteq\mathbb{N}$, $t_0,t_1,\ldots,t_N\in[0,T]$ *satisfy* $0=t_0<t_1<\ldots<t_N=T$, *let* $\tau_0,\tau_1,\ldots,\tau_n\in$

$[0,T]$ *satisfy for every* $n \in \{0,1,\ldots,N\}$ *that* $\tau_n = T - t_{N-n}$, *let* $f\colon \mathbb{R}^d \times \mathbb{R} \times \mathbb{R}^d \to \mathbb{R}$, $\mu\colon \mathbb{R}^d \to \mathbb{R}^d$, $\sigma\colon \mathbb{R}^d \to \mathbb{R}^{d\times d}$, $\varphi\colon \mathbb{R}^d \to \mathbb{R}$ *be functions, let* $(\Omega, \mathcal{F}, \mathbb{P}, (\mathcal{F}_t)_{t\in[0,T]})$ *be a filtered probability space, for every* $n \in \{1,2,\ldots,N\}$ *let* $B^{n,m,j}\colon [0,T]\times\Omega \to \mathbb{R}^d$, $m \in \mathbb{N}_0$, $j \in \mathbb{N}$, *be i.i.d. standard* $(\Omega, \mathcal{F}, \mathbb{P}, (\mathcal{F}_t)_{t\in[0,T]})$*-Brownian motions, for every* $n \in \{1,2,\ldots,N\}$ *let* $\xi^{n,m,j}\colon \Omega \to \mathbb{R}^d$, $m \in \mathbb{N}_0$, $j \in \mathbb{N}$, *be i.i.d.* $\mathcal{F}_0/\mathcal{B}(\mathbb{R}^d)$*-measurable random variables, let* $H\colon [0,T]^2 \times \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^d$ *be a function, let* $\mathbb{V}_n^{j,\mathbf{s}}\colon \mathbb{R}^\nu \times \mathbb{R}^d \to \mathbb{R}$, $(j,\mathbf{s},n) \in \mathbb{N}\times\mathbb{R}^\varsigma\times\{0,1,\ldots,N\}$, *be functions, for every* $n \in \{1,2,\ldots,N\}$, $m \in \mathbb{N}_0$, $j \in \mathbb{N}$ *let* $\mathcal{Y}^{n,m,j}\colon \{0,1,\ldots,N\}\times\Omega \to \mathbb{R}^d$ *be a stochastic process which satisfies for every* $k \in \{0,1,\ldots,N-1\}$ *that* $\mathcal{Y}_0^{n,m,j} = \xi^{n,m,j}$ *and*

$$\mathcal{Y}_{k+1}^{n,m,j} = H(\tau_{k+1}, \tau_k, \mathcal{Y}_k^{n,m,j}, B_{\tau_{k+1}}^{n,m,j} - B_{\tau_k}^{n,m,j}), \tag{64}$$

let $\Theta^n\colon \mathbb{N}_0\times\Omega \to \mathbb{R}^\nu$, $n \in \{0,1,\ldots N\}$, *be stochastic processes, for every* $n \in \{1,2,\ldots,N\}$, $m \in \mathbb{N}_0$, $\mathbf{s} \in \mathbb{R}^\varsigma$ *let* $\phi^{n,m,\mathbf{s}}\colon \mathbb{R}^\nu\times\Omega \to \mathbb{R}$ *be the function which satisfies for every* $\theta \in \mathbb{R}^\nu$, $\omega \in \Omega$ *that*

$$\phi^{n,m,\mathbf{s}}(\theta,\omega) = \frac{1}{J_m}\sum_{j=1}^{J_m}\Bigg[\mathbb{V}_n^{j,\mathbf{s}}\big(\theta, \mathcal{Y}_{N-n}^{n,m,j}(\omega)\big) - \mathbb{V}_{n-1}^{j,\mathbf{s}}\big(\Theta_{M_{n-1}}^{n-1}(\omega), \mathcal{Y}_{N-n+1}^{n,m,j}(\omega)\big) - (t_n - t_{n-1})$$

$$\cdot f\Big(\mathcal{Y}_{N-n+1}^{n,m}(\omega), \mathbb{V}_{n-1}^{j,\mathbf{s}}\big(\Theta_{M_{n-1}}^{n-1}(\omega), \mathcal{Y}_{N-n+1}^{n,m}(\omega)\big), (\nabla_x \mathbb{V}_{n-1}^{j,\mathbf{s}})\big(\Theta_{M_{n-1}}^{n-1}(\omega), \mathcal{Y}_{N-n+1}^{n,m}(\omega)\big)\Big)\Bigg]^2, \tag{65}$$

for every $n \in \{1,2,\ldots,N\}$, $m \in \mathbb{N}_0$, $\mathbf{s} \in \mathbb{R}^\varsigma$ *let* $\Phi^{n,m,\mathbf{s}}\colon \mathbb{R}^\nu\times\Omega \to \mathbb{R}^\nu$ *be a function which satisfies for every* $\omega \in \Omega$, $\theta \in \{\eta \in \mathbb{R}^\nu\colon \phi^{n,m,\mathbf{s}}(\cdot,\omega)\colon \mathbb{R}^\nu \to \mathbb{R}$ *is differentiable at* $\eta\}$ *that*

$$\Phi^{n,m,\mathbf{s}}(\theta,\omega) = (\nabla_\theta \phi^{n,m,\mathbf{s}})(\theta,\omega), \tag{66}$$

let $\mathcal{S}^n\colon \mathbb{R}^\varsigma\times\mathbb{R}^\nu\times(\mathbb{R}^d)^{\{0,1,\ldots,N\}\times\mathbb{N}} \to \mathbb{R}^\varsigma$, $n \in \{1,2,\ldots,N\}$, *be functions, for every* $n \in \{1,2,\ldots,N\}$, $m \in \mathbb{N}_0$ *let* $\psi_m^n\colon \mathbb{R}^\varrho \to \mathbb{R}^\nu$ *and* $\Psi_m^n\colon \mathbb{R}^\varrho\times\mathbb{R}^\nu \to \mathbb{R}^\varrho$ *be functions, for every* $n \in \{1,2,\ldots,N\}$ *let* $\mathbb{S}^n\colon \mathbb{N}_0\times\Omega \to \mathbb{R}^\varsigma$ *and* $\Xi^n\colon \mathbb{N}_0\times\Omega \to \mathbb{R}^\varrho$ *be stochastic processes which satisfy for every* $m \in \mathbb{N}_0$ *that*

$$\mathbb{S}_{m+1}^n = \mathcal{S}^n\big(\mathbb{S}_m^n, \Theta_m^n, (\mathcal{Y}_k^{n,m,i})_{(k,i)\in\{0,1,\ldots,N\}\times\mathbb{N}}\big), \tag{67}$$

$$\Xi_{m+1}^n = \Psi_m^n(\Xi_m^n, \Phi^{n,m,\mathbb{S}_{m+1}^n}(\Theta_m^n)), \qquad \text{and} \qquad \Theta_{m+1}^n = \Theta_m^n - \psi_m^n(\Xi_{m+1}^n). \tag{68}$$

In the setting of Framework 2.2 we think under suitable hypotheses for sufficiently large $N, M \in \mathbb{N}$, every $n \in \{0,1,\ldots N\}$, and every $x \in \mathbb{R}^d$ of $\mathbb{V}_n^{1,\mathbb{S}_M^n}(\Theta_M^n, x)\colon \Omega \to \mathbb{R}$ as a suitable approximation

$$\mathbb{V}_n^{1,\mathbb{S}_M^n}(\Theta_M^n, x) \approx u(t_n, x) \tag{69}$$

of $u(t_n,x)$ where $u = (u(t,x))_{(t,x)\in[0,T]\times\mathbb{R}^d} \in C^{1,2}([0,T]\times\mathbb{R}^d, \mathbb{R})$ is a function with at most polynomially growing derivatives which satisfies for every $t \in [0,T]$, $x \in \mathbb{R}^d$ that

$u(0,x) = \varphi(x)$ and

$$\begin{aligned}(\tfrac{\partial}{\partial t}u)(t,x) = f\big(x, u(t,x), (\nabla_x u)(t,x)\big) + \big\langle \mu(x), (\nabla_x u)(t,x)\big\rangle_{\mathbb{R}^d} \\ + \tfrac{1}{2}\operatorname{Trace}\big(\sigma(x)[\sigma(x)]^*(\operatorname{Hess}_x u)(t,x)\big)\end{aligned} \tag{70}$$

(cf. (3), (13), and (36)). The role of the processes $\mathbb{S}^n\colon \mathbb{N}_0 \times \Omega \to \mathbb{R}^\varsigma$, $n \in \{1,2,\ldots,N\}$, is to describe the variables needed for batch normalization.

3 Examples

In this section we illustrate the performance of the deep splitting method by means of numerical simulations for four concrete example PDEs. In each of these numerical simulations we employ the general approximation method in Subsection 2.8 in conjunction with the Adam optimizer (cf. (72) and (73) in Framework 3.1 below and Kingma & Ba [62]) with mini-batches with 256 samples in each iteration step (see Framework 3.1 for a detailed description).

In our implementation we employ N fully-connected feedforward neural networks to represent $\mathbb{V}_n^{j,\mathbf{s}}(\theta,x)$ for $n \in \{1,2,\ldots,N\}$, $j \in \{1,2,\ldots,256\}$, $\mathbf{s} \in \mathbb{R}^\varsigma$, $\theta \in \mathbb{R}^\nu$, $x \in \mathbb{R}^d$. Each of the neural networks consists of 4 layers (1 input layer, 2 hidden layers, and 1 output layer). In each of the numerical simulations in Subsections 3.1, 3.3, and 3.4 the input layer is d-dimensional, the two hidden layers are $(d+10)$-dimensional, and the output layer is 1-dimensional. In each of the numerical simulations in Subsection 3.2 the input layer is d-dimensional, the the two hidden layers are $(d+10+\mathbb{1}_{[1,100]}(d)\cdot 40)$-dimensional, and the output layer is 1-dimensional. Batch normalization (see Ioffe & Szegedy [59]) is applied just before the first linear transformation, just before each of the two nonlinear activation functions in front of the hidden layers, as well as just before the output layer. As nonlinear activation functions just in front of the two hidden layers we employ the multidimensional version of the rectifier function $\mathbb{R} \ni x \mapsto \max\{x,0\} \in [0,\infty)$. We use Xavier initialisation (see Glorot & Bengio [33]) to initialise all weights in the neural networks. Each of the numerical experiments presented below is performed in Python using TensorFlow on a NVIDIA GeForce GTX 1080 GPU with 1974 MHz core clock and 8 GB GDDR5X memory with 1809.5 MHz clock rate, where the underlying system consists of an Intel Core i7-6800K 3.4 GHz CPU with 64 GB DDR4-2133 memory running TensorFlow 1.5 on Ubuntu 16.04. We also refer to the Python codes in Section 4 below.

Framework 3.1. *Assume Framework 2.2, let* $\nu = (d+10)(d+1)+(d+10)(d+11)+(d+11)$ *(cf. E et al. [26, Remark 4.1] and the second paragraph of this section),* $\varepsilon = 10^{-8}$, $\beta_1 = \frac{9}{10}$, $\beta_2 = \frac{999}{1000}$, $(\gamma_m)_{m\in\mathbb{N}_0} \subseteq (0,\infty)$, *let* $\operatorname{Pow}_r\colon \mathbb{R}^\nu \to \mathbb{R}^\nu$, $r \in (0,\infty)$, *be the functions which satisfy for every* $r \in (0,\infty)$, $x = (x_1,\ldots,x_\nu) \in \mathbb{R}^\nu$ *that* $\operatorname{Pow}_r(x) = (|x_1|^r, |x_2|^r, \ldots, |x_\nu|^r)$, *let* $u = (u(t,x))_{(t,x)\in[0,T]\times\mathbb{R}^d} \in C^{1,2}([0,T]\times\mathbb{R}^d,\mathbb{R})$ *be a function with at most polynomially*

growing derivatives which satisfies for every $t \in [0,T]$, $x \in \mathbb{R}^d$ *that* $u(0,x) = \varphi(x)$ *and*

$$\begin{aligned}(\tfrac{\partial}{\partial t}u)(t,x) = f\big(x, u(t,x), (\nabla_x u)(t,x)\big) + \big\langle \mu(x), (\nabla_x u)(t,x)\big\rangle_{\mathbb{R}^d} \\ + \tfrac{1}{2}\operatorname{Trace}\big(\sigma(x)[\sigma(x)]^*(\operatorname{Hess}_x u)(t,x)\big),\end{aligned} \tag{71}$$

assume for every $m \in \mathbb{N}_0$, $i \in \{0,1,\ldots,N\}$ *that* $J_m = 256$, $t_i = \frac{iT}{N}$, $\varrho = 2\nu$, *and* $d \in \{10, 50, 100, 200, 300, 500, 1000, 5000, 10000\}$, *and assume for every* $n \in \{1,2,\ldots,N\}$, $m \in \mathbb{N}_0$, $x = (x_1, x_2, \ldots, x_\nu)$, $y = (y_1, y_2, \ldots, y_\nu)$, $\eta = (\eta_1, \eta_2, \ldots, \eta_\nu) \in \mathbb{R}^\nu$ *that*

$$\Psi_m^n(x,y,\eta) = (\beta_1 x + (1-\beta_1)\eta, \beta_2 y + (1-\beta_2)\operatorname{Pow}_2(\eta)) \tag{72}$$

and

$$\psi_m^n(x,y) = \left(\left[\sqrt{\tfrac{|y_1|}{1-\beta_2^m}} + \varepsilon\right]^{-1} \frac{\gamma_m x_1}{1-\beta_1^m}, \ldots, \left[\sqrt{\tfrac{|y_\nu|}{1-\beta_2^m}} + \varepsilon\right]^{-1} \frac{\gamma_m x_\nu}{1-\beta_1^m}\right). \tag{73}$$

3.1 Hamilton–Jacobi–Bellman (HJB) equations

In this subsection we use the deep splitting method in Framework 3.1 to approximately calculate the solutions of PDEs of the form

$$(\tfrac{\partial}{\partial t}u)(t,x) = (\Delta_x u)(t,x) - \|(\nabla_x u)(t,x)\|_{\mathbb{R}^d}^2 \tag{74}$$

for $t \in [0,T]$, $x \in \mathbb{R}^d$, $d \in \mathbb{N}$. We refer to (76) below for a more precise description of the PDEs under consideration. The deep splitting method, however, applies to much more general Hamilton–Jacobi–Bellman (HJB) equations than just PDEs of the form (74). A key feature of a PDE of the form (74) is that it can by means of a logarithmic transformation (cf., for example, E et al. [26, Lemma 4.2]) be reduced to a linear heat equation which, in turn, can be approximately solved by a classical Monte Carlo method. This enables us to efficiently compute reference solutions in high dimensions.

Assume Framework 3.1, let $\alpha = 1/2$, assume that $T \in \{1/3, 2/3, 1\}$, $N \in \{8, 16, 24\}$, assume for every $n, m, j \in \mathbb{N}$, $\omega \in \Omega$ that $\xi^{n,m,j}(\omega) = (0,0,\ldots,0) \in \mathbb{R}^d$, assume for every $m \in \mathbb{N}$ that

$$\gamma_m = \begin{cases} 10^{-1}\mathbb{1}_{[0,300]}(m) + 10^{-2}\mathbb{1}_{(300,400]}(m) + 10^{-3}\mathbb{1}_{(400,500]}(m) & : d < 10000 \\ 10^{-1}\mathbb{1}_{[0,400]}(m) + 10^{-2}\mathbb{1}_{(400,500]}(m) + 10^{-3}\mathbb{1}_{(500,600]}(m) & : d = 10000, \end{cases} \tag{75}$$

and assume for every $s,t \in [0,T]$, $b,x,z \in \mathbb{R}^d$, $y \in \mathbb{R}$ that $f(x,y,z) = -\|z\|_{\mathbb{R}^d}^2$, $\mu(x) = (0,0,\ldots,0) \in \mathbb{R}^d$, $\sigma(x) = \sqrt{2}\,\mathrm{Id}_{\mathbb{R}^{d\times d}}$, $\varphi(x) = \|x\|^\alpha$, $H(t,s,x,b) = x + \mu(x)(t-s) + \sigma(x)b$ (cf. (56) and (64)). The solution $u\colon [0,T]\times\mathbb{R}^d \to \mathbb{R}$ of the PDE (71) then satisfies for every $t \in [0,T]$, $x \in \mathbb{R}^d$ that $u(0,x) = \|x\|_{\mathbb{R}^d}^\alpha$ and

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

d	T	N	Mean	Std. dev.	Ref. value	rel. L^1-error	Std. dev. rel. error	avg. runtime
10	1/3	8	1.56645	0.00246699	1.56006	0.00410	0.00158134	18.018
10	2/3	16	1.86402	0.00338646	1.85150	0.00677	0.00182904	37.947
10	1	24	2.07017	0.00634850	2.04629	0.01167	0.00310245	58.200
50	1/3	8	2.39214	0.00151918	2.38654	0.00234	0.00063656	18.033
50	2/3	16	2.84607	0.00140300	2.83647	0.00338	0.00049463	37.922
50	1	24	3.15098	0.00275839	3.13788	0.00417	0.00087906	58.359
100	1/3	8	2.85090	0.00071267	2.84696	0.00138	0.00025033	18.149
100	2/3	16	3.39109	0.00093368	3.38450	0.00195	0.00027587	38.184
100	1	24	3.75329	0.00136920	3.74471	0.00229	0.00036564	58.329
200	1/3	8	3.39423	0.00051028	3.39129	0.00087	0.00015047	18.113
200	2/3	16	4.03680	0.00088215	4.03217	0.00115	0.00021878	38.023
200	1	24	4.46734	0.00079688	4.46172	0.00126	0.00017860	58.159
300	1/3	8	3.75741	0.00063334	3.75530	0.00056	0.00016865	18.271
300	2/3	16	4.46859	0.00049953	4.46514	0.00077	0.00011187	38.534
300	1	24	4.94586	0.00087736	4.94105	0.00097	0.00017756	58.819
500	1/3	8	4.27079	0.00051256	4.26900	0.00042	0.00012007	17.962
500	2/3	16	5.07900	0.00034792	5.07618	0.00056	0.00006854	38.001
500	1	24	5.62126	0.00045092	5.61735	0.00070	0.00008027	57.670
1000	1/3	8	5.07989	0.00022764	5.07876	0.00022	0.00004482	20.649
1000	2/3	16	6.04130	0.00030680	6.03933	0.00033	0.00005080	43.689
1000	1	24	6.68594	0.00040334	6.68335	0.00039	0.00006035	66.546
5000	1/3	8	7.59772	0.00024745	7.59733	0.00005	0.00003257	120.397
5000	2/3	16	9.03721	0.00027322	9.03466	0.00028	0.00003024	256.650
5000	1	24	9.97266	0.00047098	9.99835	0.00257	0.00004711	393.894
10000	1/3	8	9.03574	0.00022994	9.03535	0.00004	0.00002545	519.848
10000	2/3	16	10.74521	0.00026228	10.74478	0.00004	0.00002157	1105.575
10000	1	24	11.87860	0.00022705	11.89099	0.00104	0.00001909	1687.680

Table 1: Numerical simulations of the deep splitting method in Framework 3.1 in the case of the Hamilton–Jacobi–Bellman (HJB) equation in (76).

d	Mean	Stdev	Ref. value	rel. L^1-error	Stdev rel. error	avg. runtime
10	40.6553107	0.1000347132	40.7611353	0.0029624273	0.0019393471	858.3129092
50	37.421057	0.0339765334	37.5217732	0.0026842068	0.0009055151	975.3706101
100	36.3498646	0.027989905	36.4084035	0.0016078403	0.000768776	1481.5484843
200	35.374638	0.035236816	35.4127342	0.0012857962	0.0006625744	951.2294598
300	34.8476466	0.0225350305	34.8747946	0.0008818254	0.0004762554	953.3183895
500	34.2206181	0.0081072294	34.2357988	0.0004701552	0.0001701012	956.0106124
1000	33.4058827	0.0050161752	33.4358163	0.0008952555	0.000150024	1039.5774061
5000	31.7511529	0.0048508218	31.7906594	0.0012427078	0.0001525864	7229.6752827
10000	31.1215014	0.0031131196	31.1569116	0.0011365119	0.00009991746	23593.212019

Table 2: Numerical simulations of the deep splitting method in Framework 3.1 in the case of the nonlinear Black–Scholes equation with default risk in (80).

In Table 1 we use PYTHON code 2 in Subsection 4.2 below to approximately calculate the mean of $\mathbb{V}_N^{1,\mathbb{S}_M^N}(\Theta_M^N, x)$, the standard deviation of $\mathbb{V}_N^{1,\mathbb{S}_M^N}(\Theta_M^N, x)$, the relative L^1-approximation error (relative to the reference value) associated to $\mathbb{V}_N^{1,\mathbb{S}_M^N}(\Theta_M^N, x)$, the uncorrected sample standard deviation of the relative approximation error (relative to the reference value) associated to $\mathbb{V}_N^{1,\mathbb{S}_M^N}(\Theta_M^N, x)$, and the average runtime in seconds needed for calculating one realization of $\mathbb{V}_N^{1,\mathbb{S}_M^N}(\Theta_M^N, x)$ against $M \in \{500, 600\}$ based on 10 independent realizations (10 independent runs of PYTHON code 2 in Subsection 4.2 below). The reference value has been calculated by means of Han et al. [26, Lemma 4.2] (with $d = d$, $T = T$, $\alpha = 1$, $\beta = -1$, $g = \varphi$ in the notation of [26, Lemma 4.2]) and a classical Monte Carlo method.

3.2 Nonlinear Black–Scholes equations

In this subsection we use the deep splitting method in Framework 3.1 to approximately calculate the solutions of high-dimensional nonlinear Black–Scholes equations. In the scientific literature there are a number of models which intend to incorporate nonlinear phenomena such as transaction costs, default risks, or Knightian uncertainty into the classical linear Black–Scholes model and such models typically result in nonlinear Black–Scholes partial differential equations. In this subsection we consider the following nonlinear version of the Black–Scholes PDE which aims to take default risks into account, that is, in this subsection we consider PDEs of the form

$$\begin{aligned}(\tfrac{\partial}{\partial t}u)(t,x) = {} & -u(t,x)\,(1-\delta)\left[\min\Big\{\gamma^h, \max\Big\{\gamma^l, \tfrac{(\gamma^h-\gamma^l)}{(v^h-v^l)}\left(u(t,x)-v^h\right)+\gamma^h\Big\}\Big\}\right] \\ & -Ru(t,x) + \left\langle \bar{\mu}\, x, (\nabla_x u)(t,x)\right\rangle_{\mathbb{R}^d} + \tfrac{\bar{\sigma}^2}{2}\left[\sum_{i=1}^{d} |x_i|^2\,(\tfrac{\partial^2}{\partial x_i^2}u)(t,x)\right]\end{aligned} \tag{77}$$

for $t \in [0,T]$, $x \in \mathbb{R}^d$, $d \in \mathbb{N}$ where δ, R, γ^h, γ^l, v^h, v^l, $\bar{\mu}$, $\bar{\sigma}$ are suitable parameters (see (79)–(80) below for a precise description for the PDEs under consideration). We refer, e.g., to Han et al. [46, Subsection 3.1] and E et al. [28, Subsection 3.1] for explanations regarding the connection of the solution of the PDE in (80) to the problem of pricing financial derivatives without ignoring default risks.

Assume Framework 3.1, assume that $T = 1/3$, $N = 96$, assume for every $n, m, j \in \mathbb{N}$, $\omega \in \Omega$ that $\xi^{n,m,j}(\omega) = (50, \dots, 50) \in \mathbb{R}^d$, assume for every $m \in \mathbb{N}$ that

$$\gamma_m = \begin{cases} 10^{-1}\mathbb{1}_{[0,2500]}(m) + 10^{-2}\mathbb{1}_{(2500,2750]}(m) + 10^{-3}\mathbb{1}_{(2750,3000]}(m) & : d \leq 100 \\ 10^{-1}\mathbb{1}_{[0,1500]}(m) + 10^{-2}\mathbb{1}_{(1500,1750]}(m) + 10^{-3}\mathbb{1}_{(1750,2000]}(m) & : d > 100, \end{cases} \tag{78}$$

let $R = \frac{2}{100}$, $\delta = \frac{2}{3}$, $\bar{\mu} = 0.02$, $\bar{\sigma} = 0.2$, $v^h = 50$, $v^l = 70$, $\gamma^h = 0.2$, $\gamma^l = 0.02$, and assume for every $s, t \in [0,T]$, $b = (b_1, b_2, \dots, b_d), x = (x_1, x_2, \dots, x_d), z = (z_1, z_2, \dots, z_d) \in \mathbb{R}^d$, $y \in \mathbb{R}$ that $\mu(x) = \bar{\mu}x$, $\sigma(x) = \bar{\sigma}x$, $\varphi(x) = \min_{i \in \{1,2,\dots,d\}} x_i$, $H(t,s,x,b) = x + \mu(x)(t-s) + \sigma(x)b$ (cf. (56) and (64)), and

$$f(x,y,z) = -(1-\delta)\min\Big\{\gamma^h, \max\Big\{\gamma^l, \tfrac{(\gamma^h - \gamma^l)}{(v^h - v^l)}\big(y - v^h\big) + \gamma^h\Big\}\Big\} y - Ry \tag{79}$$

The solution $u\colon [0,T] \times \mathbb{R}^d \to \mathbb{R}$ of the PDE (71) then satisfies for every $t \in [0,T]$, $x \in \mathbb{R}^d$ that $u(0,x) = \min_{i \in \{1,2,\dots,d\}} x_i$ and

$$\begin{aligned} (\tfrac{\partial}{\partial t}u)(t,x) = -u(t,x)\,(1-\delta)\left[\min\Big\{\gamma^h, \max\Big\{\gamma^l, \tfrac{(\gamma^h-\gamma^l)}{(v^h-v^l)}\big(u(t,x) - v^h\big) + \gamma^h\Big\}\Big\}\right] \\ -Ru(t,x) + \big\langle \bar{\mu}\, x, (\nabla_x u)(t,x)\big\rangle_{\mathbb{R}^d} + \tfrac{\bar{\sigma}^2}{2}\left[\sum_{i=1}^{d} |x_i|^2 (\tfrac{\partial^2}{\partial x_i^2}u)(t,x)\right] \end{aligned} \tag{80}$$

(cf., for example, Duffie et al. [25], Bender and Schweizer [6], or E et al. [28, Subsection 3.1]). The specific type of PDE in (80), the specific choice of the parameters, and the initial condition have been adopted from E et al. [28, Subsection 3.1] and Han et al. [46, Subsection 3.1]. In the abovenamed references the PDE in (80) is formulated as a terminal value problem as it is usual in this kind of application. Since terminal value problems and initial value problems can easily be transformed into each other we chose the unusual formulation of the PDEs in (80) to keep the PDE formulations consistent across all the examples.

In Table 2 we use PYTHON code 3 in Subsection 4.3 below to approximately calculate the mean of $\mathbb{V}_N^{1,\mathbb{S}_M^N}(\Theta_M^N, x)$, the standard deviation of $\mathbb{V}_N^{1,\mathbb{S}_M^N}(\Theta_M^N, x)$, the relative L^1-approximation error (relative to the reference value which is used as an approximation for the unknown value of the exact solution of (80)) associated to $\mathbb{V}_N^{1,\mathbb{S}_M^N}(\Theta_M^N, x)$, the uncorrected sample standard deviation of the relative approximation error (relative to the reference value which is used as an approximation for the unknown value of the exact solution of (80)) associated to $\mathbb{V}_N^{1,\mathbb{S}_M^N}(\Theta_M^N, x)$, and the average runtime in seconds needed

for calculating one realization of $\mathbb{V}_N^{1,\mathbb{S}_M^N}(\Theta_M^N, x)$ against $M = 500$ based on 10 independent realizations (10 independent runs of PYTHON code 3 in Subsection 4.3 below). The reference value, which is used as an approximation for the unknown value of the exact solution of (80), has been calculated through the deep learning-based numerical method in E et al. [26] (see [26, Subsection 4.4]).

3.3 Allen–Cahn equations

In this subsection we use the deep splitting method in Framework 3.1 to approximately calculate the solutions of high-dimensional Allen–Cahn equations with a cubic nonlinearity, that is, we approximately calculate solutions of PDEs of the form

$$(\tfrac{\partial}{\partial t}u)(t,x) = (\Delta_x u)(t,x) + u(t,x) - [u(t,x)]^3 \tag{81}$$

for $t \in [0,T]$, $x \in \mathbb{R}^d$, $d \in \mathbb{N}$ (cf., for example, Section 4.1 in [3], E et al. [26, Section 4.2], E et al. [28, Section 3.4], and Han et al. [46, Section 3.3] for further numerical tests for equations of the form (81)).

Assume Framework 3.1, assume that $T = 0.3$, $N = 10$, assume for every $n, m, j \in \mathbb{N}$, $\omega \in \Omega$ that $\xi^{n,m,j}(\omega) = (0, 0, \ldots, 0) \in \mathbb{R}^d$, $\gamma_m = 10^{-1}\mathbb{1}_{[0,300]}(m) + 10^{-2}\mathbb{1}_{(300,400]}(m) + 10^{-3}\mathbb{1}_{(400,500]}(m)$, and assume for every $s, t \in [0,T]$, $b, x, z \in \mathbb{R}^d$, $y \in \mathbb{R}$ that $\varphi(x) = \arctan(\max_{i\in\{1,2,\ldots,d\}} x_i)$, $f(x,y,z) = y - y^3$, $\mu(x) = (0,0,\ldots,0) \in \mathbb{R}^d$, $\sigma(x) = \sqrt{2}\,\mathrm{Id}_{\mathbb{R}^{d\times d}}$, $H(t,s,x,b) = x + \mu(x)(t-s) + \sigma(x)b$ (cf. (56) and (64)). The solution $u\colon [0,T]\times\mathbb{R}^d \to \mathbb{R}$ of the PDE (71) then satisfies for every $t \in [0,T]$, $x = (x_1, x_2, \ldots, x_d) \in \mathbb{R}^d$ that $u(0,x) = \arctan(\max_{i\in\{1,2,\ldots,d\}} x_i)$ and

$$(\tfrac{\partial}{\partial t}u)(t,x) = (\Delta_x u)(t,x) + u(t,x) - [u(t,x)]^3. \tag{82}$$

In Table 3 we use PYTHON code 4 in Subsection 3.3 below to approximately calculate the mean of $\mathbb{V}_N^{1,\mathbb{S}_M^N}(\Theta_M^N, x)$, the standard deviation of $\mathbb{V}_N^{1,\mathbb{S}_M^N}(\Theta_M^N, x)$, the relative L^1-approximation error (relative to the reference value which is used as an approximation for the value of the exact solution of (82)) associated to $\mathbb{V}_N^{1,\mathbb{S}_M^N}(\Theta_M^N, x)$, the uncorrected sample standard deviation of the relative approximation error (relative to the reference value which is used as an approximation for the value of the exact solution of (82)) associated to $\mathbb{V}_N^{1,\mathbb{S}_M^N}(\Theta_M^N, x)$, and the average runtime in seconds needed for calculating one realization of $\mathbb{V}_N^{1,\mathbb{S}_M^N}(\Theta_M^N, x)$ against $M = 500$ based on 10 independent realizations (10 independent runs of PYTHON code 4 in Subsection 3.3 below). The reference value, which is used as an approximation for the value of the exact solution of (82), has been calculated through the Multilevel Picard approximation method (see, e.g., [27, 28, 55, 57, 58]).

d	Mean	Std. dev.	Ref. value	rel. L^1-error	Std. dev. rel. error	avg. runtime
10	0.89327	0.00299962	0.89060	0.00364	0.00258004	22.672
50	1.01855	0.00073173	1.01830	0.00063	0.00036976	22.196
100	1.04348	0.00029431	1.04510	0.00156	0.00028161	22.333
200	1.06119	0.00018821	1.06220	0.00096	0.00017719	22.358
300	1.06961	0.00017250	1.07217	0.00239	0.00016089	22.646
500	1.07847	0.00013055	1.08124	0.00256	0.00012074	23.111
1000	1.08842	0.00005689	1.09100	0.00236	0.00005215	25.893
5000	1.10522	0.00005201	1.10691	0.00153	0.00004699	134.552
10000	1.11071	0.00004502	1.11402	0.00296	0.00004041	473.584

Table 3: Numerical simulations of the deep splitting method in Framework 3.1 in the case of the Allen–Cahn equation in (82).

3.4 Semilinear heat equations

In this subsection we use the deep splitting method in Framework 3.1 to approximately calculate the solutions of semilinear heat equations of the form

$$(\tfrac{\partial}{\partial t}u)(t,x) = (\Delta_x u)(t,x) + \frac{1-[u(t,x)]^2}{1+[u(t,x)]^2} \tag{83}$$

for $t \in [0,T]$, $x \in \mathbb{R}^d$, $d \in \mathbb{N}$. We refer to (84) below for a precise description of the PDEs under consideration.

Assume Framework 3.1, assume that $T = 0.3$, $N = 20$, assume for every $n, m, j \in \mathbb{N}$, $\omega \in \Omega$ that $\xi^{n,m,j}(\omega) = (0, 0, \ldots, 0) \in \mathbb{R}^d$, $\gamma_m = 10^{-1}\mathbb{1}_{[0,300]}(m) + 10^{-2}\mathbb{1}_{(300,400]}(m) + 10^{-3}\mathbb{1}_{(400,500]}(m)$, and assume for every $s, t \in [0,T]$, $b, x, z \in \mathbb{R}^d$, $y \in \mathbb{R}$ that $\varphi(x) = {}^{5}\!/_{(10+2\|x\|^2_{\mathbb{R}^d})}$, $f(x,y,z) = \frac{1-y^2}{1+y^2}$, $\mu(x) = (0, 0, \ldots, 0) \in \mathbb{R}^d$, $\sigma(x) = \sqrt{2}\,\mathrm{Id}_{\mathbb{R}^{d\times d}}$, $H(t,s,x,b) = x + \mu(x)(t-s) + \sigma(x)b$ (cf. (56) and (64)). The solution $u\colon [0,T] \times \mathbb{R}^d \to \mathbb{R}$ of the PDE (71) then satisfies for every $t \in [0,T]$, $x = (x_1, x_2, \ldots, x_d) \in \mathbb{R}^d$ that $u(0,x) = {}^{5}\!/_{(10+2\|x\|^2_{\mathbb{R}^d})}$ and

$$(\tfrac{\partial}{\partial t}u)(t,x) = (\Delta_x u)(t,x) + \frac{1-[u(t,x)]^2}{1+[u(t,x)]^2}. \tag{84}$$

In Table 4 we use PYTHON code 5 in Subsection 4.5 below to approximately calculate the mean of $\mathbb{V}_N^{1,\mathbb{S}_M^N}(\Theta_M^N, x)$, the standard deviation of $\mathbb{V}_N^{1,\mathbb{S}_M^N}(\Theta_M^N, x)$, the relative L^1-approximation error (relative to the reference value which is used as an approximation for the value of the exact solution of (84)) associated to $\mathbb{V}_N^{1,\mathbb{S}_M^N}(\Theta_M^N, x)$, the uncorrected sample standard deviation of the relative approximation error (relative to the reference value which is used as an approximation for the value of the exact solution of (84)) associated

d	Mean	Std. dev.	Ref. value	rel. L^1-error	Std. dev. rel. error	avg. runtime
10	0.47138	0.00035606	0.47006	0.00282	0.00075749	46.712
50	0.34584	0.00018791	0.34425	0.00462	0.00054586	46.741
100	0.31783	0.00008298	0.31674	0.00343	0.00026198	47.375
200	0.30210	0.00002238	0.30091	0.00394	0.00007436	48.065
300	0.29654	0.00001499	0.29534	0.00406	0.00005075	48.282
500	0.29200	0.00000611	0.29095	0.00361	0.00002099	48.528
1000	0.28852	0.00000267	0.28753	0.00344	0.00000930	54.080
5000	0.28569	0.00000042	0.28469	0.00352	0.00000148	286.306
10000	0.28533	0.00000048	0.28433	0.00353	0.00000170	1013.008

Table 4: Numerical simulations of the deep splitting method in Framework 3.1 in the case of the semilinear heat equation in (84).

to $\mathbb{V}_N^{1,\mathbb{S}_M^N}(\Theta_M^N, x)$, and the average runtime in seconds needed for calculating one realization of $\mathbb{V}_N^{1,\mathbb{S}_M^N}(\Theta_M^N, x)$ against $M = 500$ based on 10 independent realizations (10 independent runs of PYTHON code 5 in Subsection 4.5 below). The reference value, which is used as an approximation for the value of the exact solution of (84), has been calculated through the Multilevel Picard approximation method (see, e.g., [27, 28, 55, 57, 58]).

4 Source codes

4.1 PYTHON source code for the algorithm

In Subsections 4.2–4.5 below we present PYTHON source codes associated to the numerical simulations in Subsections 3.1–3.4 above. The following PYTHON source code, PYTHON code 1 below, is employed in the case of each of the PYTHON source codes in Subsections 4.2–4.5 below.

```
import tensorflow as tf
import os
from tensorflow.contrib.layers.python.layers import initializers
from tensorflow.python.training.moving_averages \
    import assign_moving_average
from tensorflow.contrib.layers.python.layers import utils


def neural_net(y, neurons, name, is_training,
               reuse=None, decay=0.9, dtype=tf.float32):

    def batch_normalization(x):
        beta = tf.get_variable(
            'beta', [x.get_shape()[-1]], dtype,
            tf.zeros_initializer())
        gamma = tf.get_variable(
            'gamma', [x.get_shape()[-1]], dtype,
            tf.ones_initializer())
        mv_mean = tf.get_variable(
            'mv_mean', [x.get_shape()[-1]], dtype=dtype,
            initializer=tf.zeros_initializer(), trainable=False)
        mv_var = tf.get_variable(
            'mv_var', [x.get_shape()[-1]], dtype=dtype,
            initializer=tf.ones_initializer(), trainable=False)
        mean, variance = tf.nn.moments(x, [0], name='moments')
        tf.add_to_collection(
            tf.GraphKeys.UPDATE_OPS,
            assign_moving_average(mv_mean, mean, decay,
                                  zero_debias=True))
        tf.add_to_collection(
            tf.GraphKeys.UPDATE_OPS,
            assign_moving_average(mv_var, variance, decay,
                                  zero_debias=False))
        mean, variance = utils.smart_cond(is_training,
                                          lambda: (mean, variance),
                                          lambda: (mv_mean, mv_var))
        return tf.nn.batch_normalization(x, mean, variance,
                                         beta, gamma, 1e-6)

```

```
    def layer(x, out_size, activation):
        w = tf.get_variable(
            'weights', [x.get_shape().as_list()[-1], out_size],
            dtype, initializers.xavier_initializer())
        return activation(batch_normalization(tf.matmul(x, w)))

    with tf.variable_scope(name, reuse=reuse):
        y = batch_normalization(y)
        for i in range(len(neurons) - 1):
            with tf.variable_scope('layer_%i_' % (i + 1)):
                y = layer(y, neurons[i], tf.nn.relu)
        with tf.variable_scope('layer_%i_' % len(neurons)):
            return layer(y, neurons[-1], tf.identity)


def splitting_model(y, t, n, phi, f, net,
                    neurons, batch_size, dtype=tf.float32):

    v_n = None

    _y = y[:, :, 1]
    if net == 0:
        v_i = phi(_y)
    else:
        v_i = neural_net(_y, neurons, 'v_%i_' % net,
                         False, dtype=dtype)
    grad_v = tf.gradients(v_i, _y)

    if net == n - 1:
        v_n = tf.get_variable('v_%i_' % (net + 1), [], dtype,
                              tf.random_uniform_initializer())
        v_j = tf.ones([batch_size, 1], dtype) * v_n
    else:
        v_j = neural_net(y[:, :, 0], neurons, 'v_%i_' % (net + 1),
                         True, dtype=dtype)

    loss = (v_j - tf.stop_gradient(v_i
            + t / n * f(_y, v_i, grad_v[0]))) ** 2

    return tf.reduce_mean(loss), v_n


def simulate(t, n, d, sde, phi, f, neurons, train_steps, batch_size,
             lr_boundaries, lr_values, path, epsilon=1e-8):

    for i in range(n):

        tf.reset_default_graph()

```

```
        y = sde(d, n - i - 1)
        loss, v_n = splitting_model(y, t, n, phi, f, i,
                                    neurons, batch_size)

        global_step = tf.get_variable(
            'global_step_%i_' % (i + 1), [], tf.int32,
            tf.zeros_initializer(), trainable=False)
        learning_rate = tf.train.piecewise_constant(
            global_step, lr_boundaries, lr_values)
        update_ops = tf.get_collection(
            tf.GraphKeys.UPDATE_OPS, 'v_%i_' % (i + 1))
        with tf.control_dependencies(update_ops):
            train_op = tf.train.AdamOptimizer(
                learning_rate, epsilon=epsilon).minimize(
                loss, global_step=global_step)

        with tf.Session() as sess:

            sess.run(tf.global_variables_initializer())
            var_list_n = tf.get_collection(
                tf.GraphKeys.GLOBAL_VARIABLES, 'v_%i_' % (i + 1))
            saver_n = tf.train.Saver(var_list=var_list_n)

            if i > 0:
                saver_p = tf.train.Saver(
                    var_list=tf.get_collection(
                        tf.GraphKeys.GLOBAL_VARIABLES, 'v_%i_' % i))
                saver_p.restore(
                    sess, os.path.join(path, 'model_%i_' % i))

            for _ in range(train_steps):
                sess.run(train_op)

            saver_n.save(
                sess, os.path.join(path, 'model_%i_' % (i + 1)))

            if i == n - 1:
                return sess.run(v_n)
```

PYTHON code 1: *SplittingModel.py*

4.2 A PYTHON source code associated to the numerical simulations in Subsection 3.1

```
import tensorflow as tf
import numpy as np
```

```
import os
import time
import shutil
from SplittingModel import simulate


def phi(y):
    return tf.reduce_sum(y ** 2, axis=1, keepdims=True) ** 0.25


def f(x, y, z):
    return -tf.reduce_sum(z ** 2, axis=1, keepdims=True)


def sde(_d, n):
    x = [tf.random_normal([batch_size, _d, 1],
                          stddev=np.sqrt(2. * n * T / N)),
         tf.random_normal([batch_size, _d, 1],
                          stddev=np.sqrt(2. * T / N))]
    return tf.cumsum(tf.concat(x, axis=2), axis=2)


batch_size = 256
train_steps = 600
lr_boundaries = [400, 500]
lr_values = [0.1, 0.01, 0.001]

_file = open('HJB.csv', 'w')
_file.write('d, T, N, run, value, time\n')

for d in [10, 50, 100, 200, 300, 500, 1000, 5000, 10000]:

    neurons = [d + 10, d + 10, 1]

    for N in [8, 16, 24]:

        T = N / 24.

        for run in range(10):

            path = '/tmp/hjb'
            if os.path.exists(path):
                shutil.rmtree(path)
            os.mkdir(path)

            t_0 = time.time()
            v_n = simulate(T, N, d, sde, phi, f, neurons,
                           train_steps, batch_size,
                           lr_boundaries, lr_values, path)
```

```
            t_1 = time.time()

            _file.write('%i, %f, %i, %i, %f, %f\n'
                        % (d, T, N, run, v_n, t_1 - t_0))
            _file.flush()
            print d, T, N, run, v_n, t_1 - t_0

_file.close()
```

PYTHON code 2: *HamiltonJacobiBellman.py*

4.3 A PYTHON source code associated to the numerical simulations in Subsection 3.2

```
import tensorflow as tf
import numpy as np
import os
import time
import shutil
from SplittingModel import simulate


def phi(y):
    return tf.reduce_min(y, axis=1, keepdims=True)


def f(x, y, z):
    return -(1. - delta) * tf.minimum(
        tf.maximum((y - v_h) * (gamma_h - gamma_l) / (v_h - v_l)
                   + gamma_h, gamma_l), gamma_h) * y - R * y


def sde(_d, n):
    y = [tf.ones((batch_size, _d)) * 50.]
    for _n in range(n + 1):
        y.append(y[-1] * (1. + mu_bar * T / N
          + sigma_bar * tf.random_normal((batch_size, _d),
                                         stddev=np.sqrt(T / N))))
    return tf.stack(y[n:n + 2], axis=2)


def sde_loop(_d, n):
    xi = tf.ones((batch_size, _d)) * 50.

    def loop(_n, _x0, _x1):
        _x0 = _x1
```

```
        _x1 = _x1 * (1. + mu_bar * T / N
          + sigma_bar * tf.random_normal((batch_size, _d),
                                         stddev=np.sqrt(T / N)))
        return _n + 1, _x0, _x1

    _, x0, x1 = tf.while_loop(lambda _n, _x0, _x1: _n <= n,
                              loop,
                              (tf.constant(0), xi, xi))

    return tf.stack([x0, x1], axis=2)


N, T = 96, 1. / 3.
delta, R = 2. / 3., 0.02
mu_bar, sigma_bar = 0.02, 0.2
v_h, v_l = 50., 70.
gamma_h, gamma_l = 0.2, 0.02
lr_values = [0.1, 0.01, 0.001]
_file = open('nonlinear_BS.csv', 'w')
_file.write('d, T, N, run, value, time\n')

for d in [10, 50, 100, 200, 300, 500, 1000, 5000, 10000]:

    neurons = [d + 10, d + 10, 1] if d > 100 \
        else [d + 50, d + 50, 1]
    batch_size = 256 if d > 100 else 4096
    train_steps = 2000 if d > 100 else 3000
    lr_boundaries = [1500, 1750] if d > 100 \
        else [2500, 2750]

    for run in range(1):

        path = '/tmp/bs'
        if os.path.exists(path):
            shutil.rmtree(path)
        os.mkdir(path)

        t_0 = time.time()
        v_n = simulate(T, N, d,
                       sde_loop if d > 100 else sde,
                       phi, f, neurons, train_steps,
                       batch_size, lr_boundaries,
                       lr_values, path)
        t_1 = time.time()

        _file.write('%i, %f, %i, %i, %f, %f\n'
                    % (d, T, N, run, v_n, t_1 - t_0))
        _file.flush()

```

```
82 _file.close()
```

PYTHON code 3: *nonlinearBlackScholes.py*

4.4 A PYTHON source code associated to the numerical simulations in Subsection 3.3

```
1  import tensorflow as tf
2  import numpy as np
3  import os
4  import time
5  import shutil
6  from SplittingModel import simulate
7
8
9  def phi(y):
10     return tf.atan(tf.reduce_max(y, axis=1, keepdims=True))
11
12
13 def f(x, y, z):
14     return y - y ** 3
15
16
17 def sde(_d, n):
18     x = [tf.random_normal([batch_size, _d, 1],
19                           stddev=np.sqrt(2. * n * T / N)),
20          tf.random_normal([batch_size, _d, 1],
21                           stddev=np.sqrt(2. * T / N))]
22     return tf.cumsum(tf.concat(x, axis=2), axis=2)
23
24
25 N, T = 10, 3. / 10.
26 batch_size = 256
27 train_steps = 500
28 lr_boundaries = [300, 400]
29 lr_values = [0.1, 0.01, 0.001]
30
31 _file = open('AllenCahn.csv', 'w')
32 _file.write('d, T, N, run, value, time\n')
33
34 for d in [10, 50, 100, 200, 300, 500, 1000, 5000, 10000]:
35
36     neurons = [d + 10, d + 10, 1]
37
38     for run in range(10):
39
```

```
        path = '/tmp/allencahn'
        if os.path.exists(path):
            shutil.rmtree(path)
        os.mkdir(path)

        t_0 = time.time()
        v_n = simulate(T, N, d, sde, phi, f, neurons,
                       train_steps, batch_size,
                       lr_boundaries, lr_values, path)
        t_1 = time.time()

        _file.write('%i, %f, %i, %i, %f, %f\n'
                    % (d, T, N, run, v_n, t_1 - t_0))
        print (d, T, N, run, v_n, t_1 - t_0)

_file.close()
```

PYTHON code 4: *AllenCahn.py*

4.5 A PYTHON source code associated to the numerical simulations in Subsection 3.4

```
import tensorflow as tf
import numpy as np
import os
import time
import shutil
from SplittingModel import simulate


def phi(y):
    return 1. / (2. + 2. / 5. * tf.reduce_sum(y ** 2, axis=1))


def f(x, y, z):
    return (1. - y ** 2) / (1. + y ** 2)


def sde(_d, n):
    x = [tf.random_normal([batch_size, _d, 1],
                          stddev=np.sqrt(2. * n * T / N)),
         tf.random_normal([batch_size, _d, 1],
                          stddev=np.sqrt(2. * T / N))]
    return tf.cumsum(tf.concat(x, axis=2), axis=2)


```

```
batch_size = 256
train_steps = 500
lr_boundaries = [300, 400]
lr_values = [0.1, 0.01, 0.001]

_file = open('heat_equation.csv', 'w')
_file.write('d, T, N, run, value, time\n')

for d in [10, 50, 100, 200, 300, 500, 1000, 5000, 10000]:

    neurons = [d + 10, d + 10, 1]

    for N in [20]:

        T = 3./10.

        for run in range(10):

            path = '/tmp/heateq'
            if os.path.exists(path):
                shutil.rmtree(path)
            os.mkdir(path)

            t_0 = time.time()
            v_n = simulate(T, N, d, sde, phi, f, neurons,
                           train_steps, batch_size,
                           lr_boundaries, lr_values, path)
            t_1 = time.time()

            _file.write('%i, %f, %i, %i, %f, %f\n'
                        % (d, T, N, run, v_n, t_1 - t_0))
            print (d, T, N, run, v_n, t_1 - t_0)

_file.close()
```

PYTHON code 5: *HeatEquation.py*

Acknowledgments

Special thanks are due to Adam Andersson for fruitful discussions at the beginning of this project, in particular, for suggesting an approximation scheme which inspired us to propose the deep splitting scheme in this article. This project has been partially supported through the Swiss National Science Foundation Research project 200020_175699 "Higher order numerical approximation methods for stochastic partial differential equations" and through the Nanyang Assistant Professor Grant "Machine Learning based Algorithms in Finance and Insurance".

References

[1] Bally, V., and Pagès, G. A quantization algorithm for solving multi-dimensional discrete-time optimal stopping problems. *Bernoulli 9*, 6 (2003), 1003–1049.

[2] Beck, C., Becker, S., Grohs, P., Jaafari, N., and Jentzen, A. Solving stochastic differential equations and Kolmogorov equations by means of deep learning. *arXiv:1806.00421* (2018), 56 pages.

[3] Beck, C., E, W., and Jentzen, A. Machine learning approximation algorithms for high-dimensional fully nonlinear partial differential equations and second-order backward stochastic differential equations. *Journal of Nonlinear Science* (2017), 1–57.

[4] Becker, S., Cheridito, P., and Jentzen, A. Deep optimal stopping. *The Journal of Machine Learning Research 20*, 74 (2015), 1–25.

[5] Bender, C., and Denk, R. A forward scheme for backward SDEs. *Stochastic Processes and their Applications 117*, 12 (2007), 1793–1812.

[6] Bender, C., Schweizer, N., and Zhuo, J. A primal-dual algorithm for BSDEs. *Mathematical Finance 27*, 3 (2017), 866–901.

[7] Bengio, Y. Learning deep architectures for AI. *Foundations and Trends in Machine Learning 2*, 1 (2009), 1–127.

[8] Berg, J., and Nyström, K. A unified deep artificial neural network approach to partial differential equations in complex geometries. *Neurocomputing 317* (2018), 28–41.

[9] Berner, J., Grohs, P., and Jentzen, A. Analysis of the generalization error: Empirical risk minimization over deep artificial neural networks overcomes the curse of dimensionality in the numerical approximation of Black-Scholes partial differential equations. *arXiv:1809.03062* (2018), 35 pages.

[10] Bouchard, B., and Touzi, N. Discrete-time approximation and Monte-Carlo simulation of backward stochastic differential equations. *Stochastic Processes and their Applications 111*, 2 (2004), 175–206.

[11] Braess, D. *Finite elements*, third ed. Cambridge University Press, Cambridge, 2007. Theory, fast solvers, and applications in elasticity theory, Translated from the German by Larry L. Schumaker.

[12] Chan-Wai-Nam, Q., Mikael, J., and Warin, X. Machine learning for semi linear PDEs. *arXiv:1809.07609* (2018), 38 pages.

[13] Chassagneux, J.-F. Linear multistep schemes for BSDEs. *SIAM Journal on Numerical Analysis 52*, 6 (2014), 2815–2836.

[14] Chassagneux, J.-F., and Crisan, D. Runge-Kutta schemes for backward stochastic differential equations. *The Annals of Applied Probability 24*, 2 (2014), 679–720.

[15] Chassagneux, J.-F., and Richou, A. Numerical stability analysis of the Euler scheme for BSDEs. *SIAM Journal on Numerical Analysis 53*, 2 (2015), 1172–1193.

[16] Chassagneux, J.-F., and Richou, A. Numerical simulation of quadratic BSDEs. *The Annals of Applied Probability 26*, 1 (2016), 262–304.

[17] Crisan, D., and Manolarakis, K. Probabilistic methods for semilinear partial differential equations. Applications to finance. *M2AN Mathematical Modelling and Numerical Analysis 44*, 5 (2010), 1107–1133.

[18] 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.

[19] Crisan, D., and Manolarakis, K. Second order discretization of backward SDEs and simulation with the cubature method. *The Annals of Applied Probability 24*, 2 (2014), 652–678.

[20] Crisan, D., Manolarakis, K., and Touzi, N. On the Monte Carlo simulation of BSDEs: an improvement on the Malliavin weights. *Stochastic Processes and their Applications 120*, 7 (2010), 1133–1158.

[21] Deck, T., and Kruse, S. Parabolic differential equations with unbounded coefficients—a generalization of the parametrix method. *Acta Applicandae Mathematicae 74*, 1 (2002), 71–91.

[22] Delarue, F., and Menozzi, S. A forward-backward stochastic algorithm for quasi-linear PDEs. *The Annals of Applied Probability 16*, 1 (2006), 140–184.

[23] Dörsek, P. Semigroup splitting and cubature approximations for the stochastic Navier-Stokes equations. *SIAM Journal on Numerical Analysis 50*, 2 (2012), 729–746.

[24] Douglas, Jr., J., Ma, J., and Protter, P. Numerical methods for forward-backward stochastic differential equations. *The Annals of Applied Probability 6*, 3 (1996), 940–968.

[25] Duffie, D., Schroder, M., and Skiadas, C. Recursive valuation of defaultable securities and the timing of resolution of uncertainty. *Ann. Appl. Probab. 6*, 4 (1996), 1075–1090.

[26] E, W., Han, J., and Jentzen, A. Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. *Communications in Mathematics and Statistics 5* (2017), 349–380.

[27] E, W., Hutzenthaler, M., Jentzen, A., and Kruse, T. Multilevel Picard iterations for solving smooth semilinear parabolic heat equations. *arXiv:1607.03295* (2017), 18 pages.

[28] E, W., Hutzenthaler, M., Jentzen, A., and Kruse, T. On multilevel Picard numerical approximations for high-dimensional nonlinear parabolic partial differential equations and high-dimensional nonlinear backward stochastic differential equations. *Journal of Scientific Computing* (2019), 1–38.

[29] E, W., and Yu, B. The deep Ritz method: A deep learning-based numerical algorithm for solving variational problems. *arXiv:1710.00211* (2017), 14 pages.

[30] Elbrächter, D., Grohs, P., Jentzen, A., and Schwab, C. DNN expression rate analysis of high-dimensional PDEs: Application to option pricing. *arXiv:1809.07669* (2018), 50 pages.

[31] Farahmand, A.-m., Nabi, S., and Nikovski, D. N. Deep reinforcement learning for partial differential equation control. *2017 American Control Conference (ACC)* (2017), 3120–3127.

[32] Fujii, M., Takahashi, A., and Takahashi, M. Asymptotic expansion as prior knowledge in deep learning method for high dimensional BSDEs. *arXiv:1710.07030* (2017), 16 pages.

[33] Glorot, X., and Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. *Proceedings of the thirteenth international conference on artificial intelligence and statistics* (2010), 249–256.

[34] Gobet, E., and Labart, C. Solving BSDE with adaptive control variate. *SIAM Journal on Numerical Analysis 48*, 1 (2010), 257–277.

[35] Gobet, E., and Lemor, J.-P. Numerical simulation of BSDEs using empirical regression methods: theory and practice. *arXiv:0806.4447* (2008), 17 pages.

[36] Gobet, E., Lemor, J.-P., and Warin, X. A regression-based Monte Carlo method to solve backward stochastic differential equations. *The Annals of Applied Probability 15*, 3 (2005), 2172–2202.

[37] Gobet, E., López-Salas, J. G., Turkedjiev, P., and Vázquez, C. Stratified regression Monte-Carlo scheme for semilinear PDEs and BSDEs with large scale parallelization on GPUs. *SIAM Journal on Scientific Computing 38*, 6 (2016), C652–C677.

[38] Gobet, E., and Turkedjiev, P. Approximation of backward stochastic differential equations using Malliavin weights and least-squares regression. *Bernoulli 22*, 1 (2016), 530–562.

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

[40] Goudenege, L., Molent, A., and Zanette, A. Machine learning for pricing american options in high dimension. *arXiv:1903.11275* (2019), 11 pages.

[41] Grecksch, W., and Lisei, H. Approximation of stochastic nonlinear equations of Schrödinger type by the splitting method. *Stochastic Analysis and Applications 31*, 2 (2013), 314–335.

[42] Grohs, P., Hornung, F., Jentzen, A., and von Wurstemberger, P. A proof that artificial neural networks overcome the curse of dimensionality in the numerical approximation of Black-Scholes partial differential equations. *arXiv:1809.02362* (2018), 124 pages.

[43] Gyöngy, I., and Krylov, N. On the rate of convergence of splitting-up approximations for SPDEs. *Stochastic inequalities and applications* (2003), 301–321.

[44] Gyöngy, I., and Krylov, N. On the splitting-up method and stochastic partial differential equations. *The Annals of Probability 31*, 2 (2003), 564–591.

[45] Hairer, M., Hutzenthaler, M., and Jentzen, A. Loss of regularity for Kolmogorov equations. *The Annals of Probability 43*, 2 (2015), 468–527.

[46] Han, J., Jentzen, A., and E, W. Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences 115*, 34 (2018), 8505–8510.

[47] Han, J., and Long, J. Convergence of the deep BSDE method for coupled FBSDEs. *arXiv:1811.01165* (2018), 26 pages.

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

[49] Henry-Labordere, P. Deep primal-dual algorithm for BSDEs: Applications of machine learning to CVA and IM. *Preprint, SSRN–id3071506* (2017), 16 pages.

[50] Henry-Labordère, P., Oudjane, N., Tan, X., Touzi, N., and Warin, X. Branching diffusion representation of semilinear PDEs and Monte Carlo approximation. *To appear in Annales de l'Institut Henri Poincaré (B) Probabilits et Statistiques, arXiv:1603.01727* (2016), 30 pages.

[51] Henry-Labordère, P., Tan, X., and Touzi, N. A numerical algorithm for a class of BSDEs via the branching process. *Stochastic Processes and their Applications 124*, 2 (2014), 1112–1140.

[52] Huijskens, T. P., Ruijter, M. J., and Oosterlee, C. W. Efficient numerical Fourier methods for coupled forward-backward SDEs. *Journal of Computational and Applied Mathematics 296* (2016), 593–612.

[53] Huré, C., Pham, H., and Warin, X. Some machine learning schemes for high-dimensional nonlinear PDEs. *arXiv:1902.01599* (2019), 33 pages.

[54] Hutzenthaler, M., Jentzen, A., Kruse, T., and Nguyen, T. A. A proof that rectified deep neural networks overcome the curse of dimensionality in the numerical approximation of semilinear heat equations. *arXiv:1901.10854* (2019), 24 pages.

[55] Hutzenthaler, M., Jentzen, A., Kruse, T., Nguyen, T. A., and von Wurstemberger, P. Overcoming the curse of dimensionality in the numerical approximation of semilinear parabolic partial differential equations. *arXiv:1807.01212* (2018), 27 pages.

[56] Hutzenthaler, M., Jentzen, A., and Salimova, D. Strong convergence of full-discrete nonlinearity-truncated accelerated exponential Euler-type approximations for stochastic Kuramoto-Sivashinsky equations. *Communications in Mathematical Sciences 16* (2018), 1489–1529.

[57] Hutzenthaler, M., Jentzen, A., and von Wurstemberger, P. Overcoming the curse of dimensionality in the approximative pricing of financial derivatives with default risks. *arXiv:1903.05985* (2019), 71 pages.

[58] Hutzenthaler, M., and Kruse, T. Multi-level Picard approximations of high-dimensional semilinear parabolic differential equations with gradient-dependent nonlinearities. *arXiv:1711.01080* (2017), 19 pages.

[59] Ioffe, S., and Szegedy, C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv:1502.03167* (2015), 11 pages.

[60] Jacquier, A., and Oumgari, M. Deep PPDEs for rough local stochastic volatility. *arXiv:1906.02551* (2019), 21 pages.

[61] Jentzen, A., Salimova, D., and Welti, T. A proof that deep artificial neural networks overcome the curse of dimensionality in the numerical approximation of Kolmogorov partial differential equations with constant diffusion and nonlinear drift coefficients. *arXiv:1809.07321* (2018), 48 pages.

[62] Kingma, D., and Ba, J. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980* (2014), 15 pages.

[63] Klenke, A. *Probability theory. A comprehensive course*, second ed. Universitext. Springer, London, 2014.

[64] Kloeden, P. E., and Platen, E. *Numerical solution of stochastic differential equations*, vol. 23 of *Applications of Mathematics (New York)*. Springer-Verlag, Berlin, 1992.

[65] Krylov, N. V. *Lectures on elliptic and parabolic equations in Hölder spaces*, vol. 12 of *Graduate Studies in Mathematics*. American Mathematical Society, Providence, RI, 1996.

[66] Krylov, N. V. On Kolmogorov's equations for finite-dimensional diffusions. *Stochastic PDE's and Kolmogorov equations in infinite dimensions (Cetraro, 1998) 1715* (1999), 1–63.

[67] Kutyniok, G., Petersen, P., Raslan, M., and Schneider, R. A theoretical analysis of deep neural networks and parametric PDEs. *arXiv:1904.00377* (2019), 42 pages.

[68] Labart, C., and Lelong, J. A parallel algorithm for solving BSDEs. *Monte Carlo Methods and Applications 19*, 1 (2013), 11–39.

[69] Larsson, S., and Thomée, V. *Partial differential equations with numerical methods*, vol. 45 of *Texts in Applied Mathematics*. Springer-Verlag, Berlin, 2003.

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

[71] Lemor, J.-P., Gobet, E., and Warin, X. Rate of convergence of an empirical regression method for solving generalized backward stochastic differential equations. *Bernoulli 12*, 5 (2006), 889–916.

[72] Lionnet, A., dos Reis, G., and Szpruch, L. Time discretization of FBSDE with polynomial growth drivers and reaction-diffusion PDEs. *The Annals of Applied Probability 25*, 5 (2015), 2563–2625.

[73] Long, Z., Lu, Y., Ma, X., and Dong, B. PDE-Net: Learning PDEs from Data. *arXiv:1710.09668* (2017), 15 pages.

[74] Lye, K. O., Mishra, S., and Ray, D. Deep learning observables in computational fluid dynamics. *arXiv:1903.03040* (2019), 57 pages.

[75] Ma, J., Protter, P., San Martín, J., and Torres, S. Numerical method for backward stochastic differential equations. *The Annals of Applied Probability 12*, 1 (2002), 302–316.

[76] Ma, J., Protter, P., and Yong, J. M. Solving forward-backward stochastic differential equations explicitly—a four step scheme. *Probability Theory and Related Fields 98*, 3 (1994), 339–359.

[77] Ma, J., and Yong, J. *Forward-backward stochastic differential equations and their applications*, vol. 1702 of *Lecture Notes in Mathematics*. Springer-Verlag, Berlin, 1999.

[78] Magill, M., Qureshi, F., and de Haan, H. Neural networks trained to solve differential equations learn general representations. *Advances in Neural Information Processing Systems* (2018), 4075–4085.

[79] Maruyama, G. Continuous Markov processes and stochastic equations. *Rendiconti del Circolo Matematico di Palermo. Serie II 4* (1955), 48–90.

[80] McKean, H. P. Application of Brownian motion to the equation of Kolmogorov-Petrovskii-Piskunov. *Communications on Pure and Applied Mathematics 28*, 3 (1975), 323–331.

[81] Milstein, G. N. Approximate integration of stochastic differential equations. *Theory of Probability & Its Applications 19*, 3 (1975), 557–562.

[82] Milstein, G. N., and Tretyakov, M. V. Numerical algorithms for forward-backward stochastic differential equations. *SIAM Journal on Scientific Computing 28*, 2 (2006), 561–582.

[83] Milstein, G. N., and Tretyakov, M. V. Discretization of forward-backward stochastic differential equations and related quasi-linear parabolic equations. *IMA Journal of Numerical Analysis 27*, 1 (2007), 24–44.

[84] Milstein, G. N., and Tretyakov, M. V. Solving parabolic stochastic partial differential equations via averaging over characteristics. *Mathematics of Computation 78*, 268 (2009), 2075–2106.

[85] Pham, H. Feynman-Kac representation of fully nonlinear PDEs and applications. *Acta Mathematica Vietnamica 40*, 2 (2015), 255–269.

[86] Raissi, M. Deep hidden physics models: Deep learning of nonlinear partial differential equations. *arXiv:1801.06637* (2018), 26 pages.

[87] Ruijter, M. J., and Oosterlee, C. W. A Fourier cosine method for an efficient computation of solutions to BSDEs. *SIAM Journal on Scientific Computing 37*, 2 (2015), A859–A889.

[88] Ruijter, M. J., and Oosterlee, C. W. Numerical Fourier method and second-order Taylor scheme for backward SDEs in finance. *Applied Numerical Mathematics 103* (2016), 1–26.

[89] Ruszczynski, A., and Yao, J. A dual method for backward stochastic differential equations with application to risk valuation. *arXiv:1701.06234* (2017), 22 pages.

[90] Sirignano, J., and Spiliopoulos, K. DGM: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics 375* (2018), 1339–1364.

[91] Skorokhod, A. V. Branching diffusion processes. *Theory of Probability & Its Applications 9*, 3 (1964), 445–449.

[92] Stroock, D. W. *Lectures on topics in stochastic differential equations*, vol. 68 of *Tata Institute of Fundamental Research Lectures on Mathematics and Physics*. Tata Institute of Fundamental Research, Bombay; by Springer-Verlag, Berlin-New York, 1982. With notes by Satyajit Karmakar.

[93] Thomée, V. *Galerkin finite element methods for parabolic problems*, vol. 25 of *Springer Series in Computational Mathematics*. Springer-Verlag, Berlin, 1997.

[94] Turkedjiev, P. Two algorithms for the discrete time approximation of Markovian backward stochastic differential equations under local conditions. *Electronic Journal of Probability 20* (2015), no. 50, 49.

[95] Watanabe, S. On the branching process for Brownian particles with an absorbing boundary. *Journal of Mathematics of Kyoto University 4* (1965), 385–398.

[96] Zhang, J. A numerical scheme for BSDEs. *The Annals of Applied Probability 14*, 1 (2004), 459–488.