Journal of Machine Learning Research 25 (2024) 1-40 Submitted 7/23; Revised 3/24; Published 6/24

From continuous-time formulations to discretization schemes: tensor trains and robust regression for BSDEs and parabolic PDEs

Lorenz Richter RICHTER@ZIB.DE
Zuse Institute Berlin, Germany
dida Datenschmiede GmbH, Germany

Leon Sallandt LEON.SALLANDT@GMAIL.COM
Technische Universität Berlin, Germany

Nikolas Nüsken NIKOLAS.NUSKEN@KCL.AC.UK
King's College London, UK

Editor: Kilian Weinberger

Abstract

The numerical approximation of partial differential equations (PDEs) poses formidable challenges in high dimensions since classical grid-based methods suffer from the so-called curse of dimensionality. Recent attempts rely on a combination of Monte Carlo methods and variational formulations, using neural networks for function approximation. Extending previous work (Richter et al., 2021), we argue that tensor trains provide an appealing framework for parabolic PDEs: The combination of reformulations in terms of backward stochastic differential equations and regression-type methods holds the promise of leveraging latent low-rank structures, enabling both compression and efficient computation. Emphasizing a continuous-time viewpoint, we develop iterative schemes, which differ in terms of computational efficiency and robustness. We demonstrate both theoretically and numerically that our methods can achieve a favorable trade-off between accuracy and computational efficiency. While previous methods have been either accurate or fast, we have identified a novel numerical strategy that can often combine both of these aspects.

Keywords: High dimensional PDEs, BSDEs, tensor trains, robust loss functionals

1. Introduction

Partial differential equations (PDEs) are present in a wide range of scientific and engineering fields. However, when dealing with high-dimensional situations, their numerical handling becomes difficult due to the *curse of dimensionality* that appears in traditional grid-based approaches such as Galerkin methods, finite differences, and others. Recently, however, randomized sampling techniques in combination with powerful function classes and bespoke optimization routines have shown remarkable empirical success (Han et al., 2018; Karniadakis et al., 2021). Related theoretical results, at least in part confirmatory, are being developed at a fast rate (Beck et al., 2020), although a complete understanding remains elusive. In principle, stochastic representations as well as variational formulations pertaining to the PDE under consideration are central to those novel approaches, enabling the construction of suitable learning objectives.

In this work, we build on our conference paper (Richter et al., 2021), focusing on semilinear parabolic PDEs and the approximation of their solutions via a well known connection to backward stochastic differential equations (BSDEs), see, for instance Pardoux (1998). At the core of this BSDE reformulation is the idea of evaluating (potential) PDE solutions along the paths of a stochastic diffusion process. Therefore, in contrast to more traditional approaches, corresponding numerical methods can be thought of as based on dynamically adaptive random grids, in principle holding the promise of scaling according to dimension-free Monte Carlo approximation rates.

Solving BSDEs requires the choice of both an appropriate function class as well as of an efficient time stepping scheme. For high-dimensional problems (common, for example, in molecular dynamics, finance and optimal control) it is furthermore necessary to use a function class that enables some sort of compression, e.g. sparse coefficients or low-rank formats. As argued in Richter et al. (2021), the tensor train format (Oseledets, 2011) provides an appealing framework that addresses both desiderata. Indeed, tensor train representations ensure good scalability to high-dimensional settings by relying on constrained combinations of functions of a single real variable. In the presence of low-dimensional latent structures (and favorable alignment of those with the chosen tensor train representation), this construction alleviates the curse of dimensionality. At the same time, the specific make-up of tensor trains allows for iterative least-squares updates to efficiently compute the solutions to regression-type problems typical of discrete-time schemes for BSDEs (Bouchard and Touzi, 2004; Gobet et al., 2005).

Conditional expectations and robust regression. The numerical treatment of BSDEs is intimately related to the computation of conditional expectations; many current methods therefore rely on techniques borrowed from statistics, and in particular from regression analysis (Chessari et al., 2023, Section 4). In this introductory paragraph, however, we would like to argue that the settings in statistics and BSDEs differ in a subtle way, and that recognizing and leveraging these differences can lead to improved schemes for BSDEs which will be detailed in the upcoming sections. To convey the main idea in an abstract context, let us assume that the real-valued random variables X and Y have finite second moments and are connected through the relation

$$Y = f^*(X) + \varepsilon, \tag{1}$$

with a noise variable ε that has finite variance and satisfies $\mathbb{E}[\varepsilon|X] = 0$. Under mild conditions on f^*, the task of recovering f^* from K observations $(X_k, Y_k)_{k=1}^K$ (that is, from K realizations of the joint variable (X, Y)) can be approached using the relation

$$f^*(\cdot) = \mathbb{E}[Y|X = \cdot] = \arg\min_f \mathbb{E}\left[(Y - f(X))^2\right], \tag{2}$$

which directly follows from (1) and the characterization of conditional expectations in terms of L^2-projections (Klenke, 2013, Corollary 8.17). Indeed, given a parameterization f_θ of candidate approximations for f^*, equation (2) motivates setting $f^* \approx f_{\theta^*}$ with $\theta^* \in \arg\min \mathcal{L}$, where the loss function $\mathcal{L}$ is given by

$$\mathcal{L}^{(K)}(\theta) = \frac{1}{K}\sum_{k=1}^K (Y_k - f_\theta(X_k))^2 . \tag{3}$$

Notably, the least squares objective (3) can be evaluated without access to the noise realizations $(\varepsilon_k)_{k=1}^K$ and those are indeed typically unavailable in classical statistical settings where (1) could for instance model a phenomenon found in nature. In certain contexts (for example, concerning the numerical treatment of BSDEs, see Section 2 below), the perturbations $(\varepsilon_k)_{k=1}^K$ are generated within the algorithm and may thus be used in the formulation of optimization objectives. In particular, we can modify the loss (3) as follows,

$$\mathcal{L}^{(K)}_{\mathrm{robust}}(\theta) = \frac{1}{K}\sum_{k=1}^{K}\left(Y_k - f_\theta(X_k) - \varepsilon_k\right)^2, \tag{4}$$

directly enforcing the relation (1) on the basis of the samples $(X_k, Y_k, \varepsilon_k)_{k=1}^K$. As alluded to in the notation, we expect the inclusion of $(\varepsilon_k)_{k=1}^K$ to have a variance-reducing effect that can make the numerical procedure more robust. Indeed, it is straightforward to verify that $\partial_\theta \mathcal{L}^{(K)}_{\mathrm{robust}}|_{f_\theta = f^*} = 0$ *almost surely*, that is, the gradient of the objective (4) vanishes at the optimum *notwithstanding the fact that a finite-sample approximation is used.* In contrast, we see that $\partial_\theta \mathcal{L}^{(K)}|_{f_\theta = f^*}$ is random with mean zero, $\mathbb{E}[\partial_\theta \mathcal{L}^{(K)}|_{f=f_\theta}] = 0$, that is, the objective (3) requires the law of large number limit $K \to \infty$ in order to reliably identify the optimizer f^*. As a consequence, procedures based on (3) may become unstable in the regime $f_\theta \approx f^*$ due to a low signal-to-noise ratio.

Contributions. This paper builds on Richter et al. (2021), and we develop the arguments from the previous paragraphs in the context of BSDEs, thus providing a comprehensive analysis on related numerical stability issues (see in particular (23) and (25) below, and compare with (4) and (3), respectively). Whilst our exposition in Section 2 follows in large parts the existing vast literature on the numerical treatment of BSDEs (see, for instance, Chessari et al. (2023) for a survey), we place particular emphasis on formulations in continuous time, turning to time-discretizations at the last step of the derivation, see Section 2.2. This approach allows us to effortlessly construct explicit and implicit time stepping schemes in combination with losses of the form (3) and (4) and bespoke tensor regression schemes, extending the methodology put forward in Richter et al. (2021) and leading in particular to the robust explicit loss in (29), that so far appears to have attracted little attention. We develop a tensor train based scheme for its optimisation (see Section 4.3) and numerically investigate its performance:

Numerical evaluation. Explicit and implicit numerical schemes potentially lead to a trade-off between computational cost and stability (see Chassagneux and Richou (2015) for a related discussion in a slightly different context). In our numerical experiments, however, we do not observe significant improvements in accuracy or stability imparted by the implicit schemes, whereas the numerical overhead is substantial. We thus conjecture that for high dimensional problems, which are the focus of this paper, sampling errors exceed discretization errors.

The BSDE-versions of the robust loss (4) overall significantly outperform methods based on (3) in terms of approximation accuracy. Going beyond the abstract setting from (1), we observe in one experiment that concrete implementations of (3) and (4) in the BSDE setting lead to schemes that tend to shift emphasis from the accuracy of the PDE solution to its gradient (see Remark 4 below and the experiments in Section 5.2). This phenomenon (at the moment supported only by preliminary numerical evidence) might be of particular interest in the context of stochastic optimal control.

Combining the insights from these observations, we would like to advertise the explicit and robust loss defined in (29), which, to the best of our knowledge, has so far not been used in numerical experiments. We believe that a more in-depth analysis of its properties is a promising avenue for future work.

1.1 Setting and notation

Throughout, we denote by $C(\cdot,\cdot)$ the space of continuous functions, where the domain and codomain (or target sets) are specified within the parentheses. Similarly, $C^k(\cdot,\cdot)$ refers to the space of k-times continuously differentiable functions. We focus on semi-linear parabolic PDEs, which have the following form:

$$(\partial_t + L)V(x,t) + h(x,t,V(x,t),(\sigma^\top \nabla V)(x,t)) = 0, \qquad (x,t) \in \mathbb{R}^d \times [0,T), \tag{5a}$$

$$V(x,T) = g(x), \qquad x \in \mathbb{R}^d, \tag{5b}$$

where $h \in C(\mathbb{R}^d \times [0,T] \times \mathbb{R} \times \mathbb{R}^d, \mathbb{R})$ specifies the nonlinearity, $g \in C(\mathbb{R}^d, \mathbb{R})$ is the terminal condition, and

$$L = \frac{1}{2}\sum_{i,j=1}^d (\sigma\sigma^\top)_{ij}(x,t)\partial_{x_i}\partial_{x_j} + \sum_{i=1}^d b_i(x,t)\partial_{x_i} \tag{6}$$

is a second-order (elliptic) differential operator containing the coefficients $b \in C(\mathbb{R}^d \times [0,T], \mathbb{R}^d)$ and $\sigma \in C(\mathbb{R}^d \times [0,T], \mathbb{R}^{d\times d})$. We assume that the matrix $\sigma\sigma^\top(x)$ is nondegenerate for all $x \in \mathbb{R}^d$, and that (5) admits a unique (classical) solution $V : \mathbb{R}^d \times [0,T] \to \mathbb{R}$. Systems of the form (5) generalize the (backwards) heat equation $\partial_t V + \Delta V = 0$ and are widely used in the description of diffusive phenomena, from material science to finance (Evans, 2022). In order to obtain an approximation for the unknown function V, we reformulate (5) as a backwards SDE (see Section 2). From a computational point of view, this change of perspective opens the door for Monte Carlo approaches (potentially mitigating the curse of dimensionality) that can broadly be summarized as follows:

1. For every instance t_n within a time grid $0 = t_0 < t_1 < \cdots < t_N = T$, we produce 'spatial grid points' $(\widehat{X}_n^{(k)})_{k=1}^K$ on the basis of which the solution V shall be approximated. More precisely, we will aim to construct $\widehat{V}_n : \mathbb{R}^d \to \mathbb{R}$ such that

$$\widehat{V}_n(\widehat{X}_n^k) \approx V(\widehat{X}_n^k, t_n), \qquad \text{for all } n = 1,\ldots,N, \quad k = 1,\ldots,K. \tag{7}$$

Crucially, the grid points $(\widehat{X}_n^{(k)})_{k=1}^K$ will be (approximate) samples from the diffusion process

$$\mathrm{d}X_s = b(X_s,s)\,\mathrm{d}s + \sigma(X_s,s)\,\mathrm{d}W_s, \qquad X_0 = x_0, \tag{8}$$

associated to the operator L defined in (6). The connection between (6) and (8) underlying the generation of $(\widehat{X}_n^{(k)})_{k=1}^K$ is at the heart of the BSDE approach, see Section 2. Since $V(\cdot,T) = g$ is known from the terminal condition (5b), we will address (7) iteratively backwards in time using the updates $\widehat{V}_n \rightsquigarrow \widehat{V}_{n-1}$, see Figure 1. This procedure corresponds to the backward process in the BSDE, see Section 2. Note that as (8) does not depend on the unknown function V, we can evolve X forward in time. In what follows, we can therefore assume that trajectory samples from (8) are available.

2. To achieve the approximation (7), we reformulate this equation as a least squares regression problem (utilizing the approximation previously obtained at t_{n+1}), in the spirit of (3) and (4). For $\widehat{V}_n$, we use a tensor train ansatz of the form

$$\widehat{V}_n(x_1,\dots,x_d) = \sum a_{i_1,\dots,i_d}\phi_{i_1}(x_1)\cdot\ldots\cdot\phi_{i_d}(x_d), \tag{9}$$

with fixed ansatz functions $\phi_i : \mathbb{R} \to \mathbb{R}$ that importantly only take real numbers (of dimension one) as inputs, hence could be thought of as one-dimensional building blocks. To ensure scalability to the high-dimensional setting, the tensor train approach places stringent low-rank conditions on the coefficients $a_{i_1,\dots,i_d}$, see Section 4.

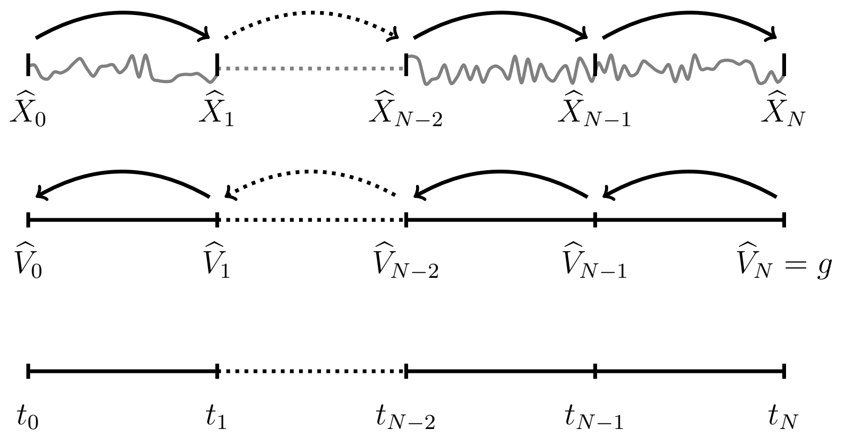


Figure 1: Schematic version of the numerical approximation of the solution to the PDE (5). First, we compute the discretized forward process $\widehat{X}_k$ along a time grid $0 = t_0 < t_1 < \cdots < t_N = T$. Then, we approximate V, starting at the known terminal value $V(\cdot, T) = g$ and gradually iterating backward in time along the time grid and the trajectories of the previously computed forward process $\widehat{X}$.

In summary, step 1 leverages the connection between the PDE and the BSDE (see (12) below) to obtain predictions for the solution V along the trajectories X (see also Figures 8 and 9 in Section 5). Step 2 builds on this data to obtain an approximate solution to V that can be obtained at any point in space, using a least-squares formulation.

Remark 1 (Generalizations)**.** *The discussion in this paper can be generalized in multiple ways beyond the semi-linear parabolic PDE* (5)*. First, the terminal value problem can be converted into an initial value problem by applying the time inversion $t \mapsto T - t$. Furthermore, generalizations to elliptic PDEs as well as to PDEs on bounded domains are in principle possible (cf. Nüsken and Richter (2023)). However, the approaches considered in* *this paper based on backward iterations would need nontrivial adaptations due the change in boundary conditions. Finally, one can also aim at targeting fully nonlinear PDEs, see e.g. Beck et al. (2019) and Pham et al. (2021).*

1.2 Previous work

BSDEs have been introduced in Bismut (1973) and were studied systematically in Pardoux and Peng (1990). We refer the reader to Pardoux (1998) for an overview that highlights the connections of BSDEs to elliptic and parabolic PDEs. A recent survey about numerical methods for BSDEs can be found in Chessari et al. (2023), tracing the development from regression based approaches in finance (Longstaff and Schwartz, 2001) through the works Ma et al. (2002); Bouchard and Touzi (2004); Zhang (2004); Gobet et al. (2005); Gobet and Labart (2007) towards recent developments involving deep learning (Beck et al., 2021; Pham et al., 2021). The present article is an extension of the conference paper Richter et al. (2021), laying particular emphasis on robustness and further analyzing the trade-offs between explicit and implicit discretization schemes. For Richter et al. (2021) and this paper, the following aspects are particularly relevant:

Function classes. Tensor trains were first introduced to the mathematical community in Oseledets (2011), with their foundations based in quantum physics and known as 'matrix product states'. They can be seen as a special case of hierarchical tensor networks, which have been developed in Hackbusch and Kühn (2009). To obtain comprehensive surveys and further information on tensor trains, we recommend referring to the following sources: Hackbusch (2014); Hackbusch and Schneider (2014); Szalay et al. (2015); Bachmayr et al. (2016).

PDE solvers. The numerical treatment of PDEs has seen recent advances via the idea of combining sophisticated function approximation with Monte Carlo sampling. Relying on deep learning techniques, we highlight E et al. (2017), which builds on a variational formulation based on BSDEs, see also Nüsken and Richter (2023). Beck et al. (2021) and Huré et al. (2020) combine deep learning attempts with backwards schemes based on Feynman-Kac and BSDEs, respectively. For recent overviews of deep learning for the approximation of solutions to PDEs we refer to E et al. (2021) and Richter (2021). Tensor trains have been used to approximate PDE solutions as well. For the treatment of parametric PDEs we refer to Dolgov et al. (2015); Eigel et al. (2017); Dektor et al. (2021). The approximation of Hamilton-Jacobi-Bellman PDEs with tensor trains can be found in Horowitz et al. (2014); Stefansson and Leong (2016); Gorodetsky et al. (2018); Dolgov et al. (2021); Oster et al. (2019); Fackeldey et al. (2022); Chen and Lu (2021) and we refer to Khoromskij (2012); Kormann (2015); Lubasch et al. (2018) for further types of PDEs.

Robustness. The idea of incorporating the Itô integral into the objective dates back to Gobet et al. (2005), where fixed point iterations are used for solving BSDEs. Using related robust and variance-reduced objectives for gradient descent based algorithms has been employed in Zhou et al. (2021) for Hamilton-Jacobi-Bellman equations and in Richter and Berner (2022) for linear PDEs. For an extensive analysis of the robustness of losses and their gradients we refer to Nüsken and Richter (2021).

Discretizations. In algorithmic contexts, time-continuous BSDEs have to be discretized. Chassagneux and Crisan (2014) study Runge-Kutta schemes for the approximation of the deterministic integral in order to potentially improve the convergence rate. Linear multistep methods are considered in Chassagneux (2014) and the numerical implications of explicit vs. implicit schemes are investigated in Chassagneux and Richou (2015). For a numerical analysis of the deep BSDE algorithm we refer to Han and Long (2020).

1.3 Outline of the article

The paper is structured as follows. In Section 2 we give an introduction to BSDEs and highlight their interpretation as a stochastic representation of the PDE (5), deriving appropriate loss functionals. In particular, Section 2.1 focuses on continuous-time formulations, while numerical aspects that become apparent in the discretization of BSDEs will be discussed in Section 2.2. Thereafter, Section 3 investigates the identified losses with respect to statistical robustness properties. Section 4 is then devoted to the tensor train format as a valid means for function approximation, encompassing complexity (Section 4.1), optimization (Section 4.2) and gradient-dependent loss functionals (Section 4.3). Finally, in Section 5 we illustrate our theoretical findings on multiple numerical examples, before Section 6 ends with a conclusion and outlook.

2. Solving PDEs via BSDEs

In this section we review the connection between semi-linear PDEs of the type (5) and backward stochastic differential equations (BSDEs), going back to Bismut (1973) and Pardoux and Peng (1990). Roughly, BSDEs can be linked to the Feynman-Kac formula (Oksendal, 2013, Chapter 8), offering stochastic representations of solutions to PDEs (Pardoux (1998)), however also in the nonlinear setting, such as the one stated in (5). As noted in earlier work (Chessari et al., 2023), reformulating the deterministic PDE (5) in stochastic terms is promising from a computational perspective, allowing the transition from grid-based methods to Monte Carlo techniques. The connection between PDEs and BSDEs has been studied extensively in the last decades (see e.g. Bouchard et al. (2009), Fahim et al. (2011) and E et al. (2017)) and we refer to Pham (2009), Touzi (2012) and Zhang (2017) for excellent introductions to the subject.

In a nutshell, BSDEs can be derived from evaluating the solution of the PDE

$$(\partial_t + L)V(x,t) + h(x,t,V(x,t),(\sigma^\top \nabla V)(x,t)) = 0, \qquad (x,t) \in \mathbb{R}^d \times [0,T), \tag{10a}$$

$$V(x,T) = g(x), \qquad x \in \mathbb{R}^d, \tag{10b}$$

defined in (5), here repeated for convenience, along the stochastic process (8),

$$\mathrm{d}X_s = b(X_s,s)\,\mathrm{d}s + \sigma(X_s,s)\,\mathrm{d}W_s, \qquad X_0 = x_0, \tag{11}$$

also restated for convenience, where $b \in C(\mathbb{R}^d \times [0,T], \mathbb{R}^d)$ and $\sigma \in C(\mathbb{R}^d \times [0,T], \mathbb{R}^{d\times d})$ refer to the same functions as in (6). Notice that the initial condition X_0 is deterministic. Conceptually speaking, the SDE (11) provides a (stochastic) grid on which the PDE solution is obtained, not unlike the equidistant (deterministic) grids used in finite-difference schemes (Johnson, 2012). To spell this out, let us assume that $V \in C^{2,1}(\mathbb{R}^d \times [0,T], \mathbb{R})$ is a classical solution to the PDE (5) and define the process

$$Y_s = V(X_s, s), \tag{12}$$

evaluating V along $(X_s)_{0 \le s \le T}$ on the time interval $s \in [0,T]$. An application of Itô's formula shows that

$$V(X_T, T) - V(X_0, 0) = \int_0^T (\partial_t + L) V(X_s, s)\, \mathrm{d}s + \int_0^T (\sigma^\top \nabla V)(X_s, s) \cdot \mathrm{d}W_s, \tag{13}$$

and therefore, using (5),

$$g(X_T) = V(X_0, 0) - \int_0^T h(X_s, s, V(X_s, s), (\sigma^\top \nabla V)(X_s, s))\, \mathrm{d}s + \int_0^T (\sigma^\top \nabla V)(X_s, s) \cdot \mathrm{d}W_s. \tag{14}$$

Further introducing the shorthand notation

$$Z_s = (\sigma^\top \nabla V)(X_s, s), \tag{15}$$

we can equivalently write (14) as

$$\mathrm{d}Y_s = -h(X_s, s, Y_s, Z_s)\, \mathrm{d}s + Z_s \cdot \mathrm{d}W_s, \qquad Y_T = g(X_T). \tag{16}$$

Solutions to the equations (11) and (16) are triplets (X, Y, Z), where X is called the forward process, whilst Y and Z are referred to as backward processes, owing to the terminal condition in (16). However, $(X_s)_{s\in[0,T]}$, $(Y_s)_{s\in[0,T]}$ and $(Z_s)_{s\in[0,T]}$ are adapted to the filtration $(\mathcal{F}_s)_{s\in[0,T]}$ generated by the Brownian motion $(W_s)_{s\in[0,T]}$, and therefore, in a causal sense, should be thought of as evolving forward in time. Together with this adaptedness condition, the equations (11) and (16) provide three constraints for the triplet (X, Y, Z), hence existence and uniqueness under appropriate conditions is plausible. For rigorous results we refer to El Karoui et al. (1997), Touzi (2012, Theorem 10.2), Zhang (2017, Theorem 4.3.1, Theorem 7.3.3) and Kobylanski (2000).

Remark 2 (Controlled forward processes)**.** *Using the same arguments, we can generalize the system of BSDEs specified in* (11) *and* (16) *by adding a control term* $v \in C(\mathbb{R}^d \times [0,T], \mathbb{R}^d)$ *to the forward process in* (11)*. Ensuring* $Y_s^v = V(X_s^v, s)$ *and* $Z_s^v = (\sigma^\top \nabla V)(X_s^v, s)$ *leads to a compensation term in the backward process,*

$$\mathrm{d}X_s^v = (b(X_s^v, s) + \sigma(X_s^v, s) v(X_s^v, s))\, \mathrm{d}s + \sigma(X_s^v, s)\, \mathrm{d}W_s, \qquad X_0^v = x_0, \tag{17a}$$

$$\mathrm{d}Y_s^v = (-h(X_s^v, s, Y_s^v, Z_s^v) + v(X_s^v, s) \cdot Z_s^v)\, \mathrm{d}s + Z_s^v \cdot \mathrm{d}W_s, \qquad Y_T^v = g(X_T^v). \tag{17b}$$

The control v in the forward process may be beneficial from the computational point of view, pushing the process into regions of interest or reducing the variance of Monte Carlo estimators, cf. Hartmann et al. (2019).

2.1 Solving BSDEs: loss functionals in continuous time

The BSDE formulation (16) opens the door for Monte Carlo approaches towards solutions to the PDE (5) by solving for the triplet (X, Y, Z) and using the correspondences (12) and (15). Numerical approaches can broadly be classified into two branches: One can either approximate solutions to the PDE (5) on the entire time interval $[0, T]$ at once, directly incorporating the time dependence (E et al. (2017), Raissi (2018), Nüsken and Richter (2021)). Alternatively, one can divide the problem into multiple subproblems and approach those one after another (Bouchard and Touzi (2004), Gobet et al. (2005), Huré et al. (2020)). In this work we focus on the latter attempt.

In the spirit of the dynamic programming principle from optimal control theory (Fleming and Rishel, 2012), the idea is to solve for the backward process on separate disjoint time intervals defined by the discretization

$$0 = t_0 < t_1 < \cdots < t_N = T, \tag{18}$$

thereby dividing the problem into a sequence of subproblems. The algorithm proceeds backwards in time, starting with the last interval $[t_{N-1}, t_N]$ and using the terminal condition $Y_T = V(X_T, T) = g(X_T)$. Subsequently, the computations on each interval will rely on approximations from previously approached intervals, with the solution at time t_n providing the terminal condition for the subproblem posed on the interval $[t_{n-1}, t_n]$, for $n = 1, \dots, N-1$. As an illustration, we refer to Figure 1.

To approach the subproblem on the time interval $[t_n, t_{n+1}]$, we first state the backward process (16) in an integrated version as

$$Y_{t_{n+1}} - Y_{t_n} = -\int_{t_n}^{t_{n+1}} h(X_s, s, Y_s, Z_s)\,\mathrm{d}s + \int_{t_n}^{t_{n+1}} Z_s \cdot \mathrm{d}W_s, \tag{19}$$

or, equivalently, as

$$\begin{aligned} V(X_{t_{n+1}}, t_{n+1}) - V(X_{t_n}, t_n) = -\int_{t_n}^{t_{n+1}} & h(X_s, s, V(X_s, s), (\sigma^\top \nabla V)(X_s, s))\,\mathrm{d}s \\ & + \int_{t_n}^{t_{n+1}} (\sigma^\top \nabla V)(X_s, s) \cdot \mathrm{d}W_s, \end{aligned} \tag{20}$$

both of which hold true almost surely. According to Pham (2009, Chapter 6) and slightly simplifying, the formulation (20) is equivalent to the PDE (5a) on $[t_n, t_{n+1}]$, in the sense that (20) holds almost surely if and only if V satisfies (5a).

Based on this observation, we may now aim to construct suitable loss functionals

$$\mathcal{L}_n : \mathcal{H} \to \mathbb{R}_{\geq 0} \tag{21}$$

that specify the misfit between an approximating function φ and the solution V on the interval $[t_n, t_{n+1}]$. Here, $\mathcal{H}$ is an appropriate function class to be chosen later on, for now assumed to contain the solution, i.e. $V \in \mathcal{H}$. More precisely, the loss (21) shall be minimal if and only if the approximation is exact, i.e.

$$\varphi^* \in \operatorname*{arg\,min}_{\varphi \in \mathcal{H}} \mathcal{L}_n(\varphi) \qquad \iff \qquad \varphi^*(x, t) = V(x, t) \quad \forall\, (x, t) \in \mathbb{R}^d \times [t_n, t_{n+1}]. \tag{22}$$

In line with the iterative procedure depicted in Figure 1, $\mathcal{L}_n$ will typically be constructed using the approximation φ obtained in the succeeding time interval $[t_{n+1}, t_{n+2}]$; in this situation we require (22) to hold provided that this approximation is exact, i.e. $\varphi = V$ on $[t_{n+1}, t_{n+2}]$.[1]

In the following we review two loss functionals that are based on (20) and satisfy (22). The first loss,

$$\mathcal{L}_{\mathrm{BSDE}}(\varphi) = \mathbb{E}\left[\left(\varphi(X_{t_n}, t_n) - V(X_{t_{n+1}}, t_{n+1}) - \int_{t_n}^{t_{n+1}} h(X_s, s, \varphi(X_s, s), (\sigma^\top \nabla\varphi)(X_s, s))\,\mathrm{d}s \right.\right.$$
$$\left.\left. + \int_{t_n}^{t_{n+1}} (\sigma^\top \nabla\varphi)(X_s, s)\cdot \mathrm{d}W_s\right)^2\right], \tag{23}$$

measures the misfit in the BSDE (20) when the solution V is replaced by the approximating function φ, in mean square sense. The formulation in (23) is in the spirit of Gobet et al. (2005), see equation (4) within in this work. As alluded to above, the loss $\mathcal{L}_{\mathrm{BSDE}}$ assumes knowledge of $V(X_{t_{n+1}}, t_{n+1})$, which can be viewed as a terminal condition for the solution on the time interval $[t_n, t_{n+1}]$. In the context of the iterative scheme from Figure 1, we will set $V(X_{t_{n+1}}, t_{n+1}) \approx \varphi(X_{t_{n+1}}, t_{n+1})$, using the approximation obtained previously in the succeding time interval (cf. Remark 8). Note that here and in the following we omit the index n indicating the time interval in $\mathcal{L}_{\mathrm{BSDE}}$ for notational convenience.

An alternative loss can be derived by applying the conditional expectation with respect to the Brownian filtration $\mathcal{F}_{t_n} = \sigma(W_s :\ 0 \le s \le t_n)$ to (20), yielding

$$V(X_{t_n}, t_n) = \mathbb{E}\left[V(X_{t_{n+1}}, t_{n+1}) + \int_{t_n}^{t_{n+1}} h(X_s, s, V(X_s, s), (\sigma^\top \nabla V)(X_s, s))\,\mathrm{d}s \,\middle|\, \mathcal{F}_{t_n}\right] \tag{24}$$
$$= \arg\min_{\varphi} \mathbb{E}\left[\left(\varphi(X_{t_n}, t_n) - V(X_{t_{n+1}}, t_{n+1}) - \int_{t_n}^{t_{n+1}} h(X_s, s, V(X_s, s), (\sigma^\top \nabla V)(X_s, s))\,\mathrm{d}s\right)^2\right],$$

where in the first line we have used the martingale property of the (Itô) stochastic integral as well as the fact that $V(X_{t_n}, t_n)$ is $\mathcal{F}_{t_n}$-measurable. In the second line, we have reformulated the conditional expectation in terms of a least-squares minimization (or projection), see Klenke (2013, Corollary 8.17). The relation (25) straightforwardly suggests the projection loss

$$\mathcal{L}_{\mathrm{proj}}(\varphi) = \mathbb{E}\left[\left(\varphi(X_{t_n}, t_n) - V(X_{t_{n+1}}, t_{n+1}) \right.\right. \tag{25}$$
$$\left.\left. - \int_{t_n}^{t_{n+1}} h(X_s, s, V(X_s, s), (\sigma^\top \nabla V)(X_s, s))\,\mathrm{d}s\right)^2\right],$$

see Chessari et al. (2023, Section 3.1) and references therein.

1. In fact, as the methods proceed locally in time, we will only require $\varphi(\cdot, t_{n+1}) = V(\cdot, t_{n+1})$.

Comparing $\mathcal{L}_{\mathrm{BSDE}}$ to $\mathcal{L}_{\mathrm{proj}}$, we see that the Itô integral has been removed, essentially by applying the conditional expectation in (20). Conversely, the passage from $\mathcal{L}_{\mathrm{proj}}$ to $\mathcal{L}_{\mathrm{BSDE}}$ has an interpretation in terms of control variates, see Robert and Casella (2004) for a general introduction and e.g. Zhou et al. (2021) or Richter and Berner (2022) for applications of control variates to linear PDEs. Indeed, starting with (24) and using the fact that the stochastic integral in the second line of (23) has (conditional) expectation zero, we may add this term to (24) in the hope of reducing the variance of corresponding Monte Carlo estimators. At this point, we would also like to point out the conceptual similarity between (25) and (1) on the one hand, and (23) and (4) on the other hand.

Further, note that $\mathcal{L}_{\mathrm{proj}}$ contains the solution V in the Riemann integral (in contrast to the corresponding term in $\mathcal{L}_{\mathrm{BSDE}}$, which depends on φ). For discretization schemes, this implies that only an approximation of the integral by its right end point contribution will lead to feasible numerical strategies, making use of $V(X_{t_{n+1}}, t_{n+1}) \approx \varphi(X_{t_{n+1}}, t_{n+1})$ obtained in the previous iteration on the interval $[t_{n+1}, t_{n+2}]$. As we will see in Section 2.2 below, the loss $\mathcal{L}_{\mathrm{BSDE}}$ allows for more flexible discretizations in time.

Remark 3 ((Non-)vanishing losses). *Substituting $\varphi = V$ into $\mathcal{L}_{\mathrm{proj}}$ and $\mathcal{L}_{\mathrm{BSDE}}$ as well as using the relation* (20) *we see that*

$$\min_{\varphi \in \mathcal{H}} \mathcal{L}_{\mathrm{proj}}(\varphi) = \mathbb{E}\left[\int_{t_n}^{t_{n+1}} |(\sigma^\top \nabla V)(X_s, s)|^2 \,\mathrm{d}s\right], \qquad \min_{\varphi \in \mathcal{H}} \mathcal{L}_{\mathrm{BSDE}}(\varphi) = 0. \tag{26}$$

Therefore, the accuracy of an approximation $\varphi \in \mathcal{H}$ may be monitored in an online-fashion in algorithms that are based on $\mathcal{L}_{\mathrm{BSDE}}$. For methods based on $\mathcal{L}_{\mathrm{proj}}$, the same may prove challenging since the expectation in (26) *will rarely be available in closed form.*

2.2 Discretizing the loss functionals

Finally, we can design implementable algorithms by discretizing the involved (stochastic) integrals and replacing the expectations by Monte Carlo estimators. In that context, we content ourselves with approximating the solution at the grid points defined in (18), i.e., we seek functions $\widehat{V}_n \in \widehat{\mathcal{H}}$ such that $\widehat{V}_n(\cdot) \approx V(\cdot, t_n)$, for $n = 0, \dots, N-1$, and an appropriately chosen function class $\widehat{\mathcal{H}}$ (cf. Figure 1). We note that whilst $\mathcal{H}$ in (21) is understood to contain functions of both space and time, $\widehat{\mathcal{H}}$ comprises functions of space only. For the forward process (11) we employ the Euler-Maruyama scheme

$$\widehat{X}_{n+1} = \widehat{X}_n + b(\widehat{X}_n, t_n)\Delta t + \sigma(\widehat{X}_n, t_n)\xi_{n+1}\sqrt{\Delta t}, \tag{27}$$

where $\Delta t = t_{n+1} - t_n$ is the time-step and $\xi_{n+1} \sim \mathcal{N}(0, \mathrm{Id})$ are independent standard normally distributed random variables. It can be shown that $\widehat{X}_n$ approximates $X_{n\Delta t}$ in an appropriate sense as $\Delta t \to 0$, see Kloeden and Platen (1992).

For the losses (23) and (25) we need to discretize the deterministic and stochastic integrals. Here, we only consider one-step approximations (i.e. we use only one term in the 'sum'), but refer to Remark 7 below for generalizations. As commented on before Remark 3, the integral in the first line of (23) can be approximated by either left or right endpoint evaluations of the integrand due to the fact that there is no dependence on $\widehat{V}$. Ultimately, this flexibility allows for the development of both explicit and implicit schemes based on $\mathcal{L}_{\mathrm{BSDE}}$.

For the loss $\mathcal{L}_{\mathrm{proj}}$ in (25), on the other hand, we must rely on the right endpoint, making use of the availability of $V(X_{t_{n+1}}, t_{n+1}) \approx \varphi(X_{t_{n+1}}, t_{n+1})$ from the previous time-step.

To simplify notation, we introduce a shorthand notation as follows:

$$h_n = h(\widehat{X}_n, t_n, \widehat{V}_n(\widehat{X}_n), \sigma^\top(\widehat{X}_n, t_n)\nabla\widehat{V}_n(\widehat{X}_n)). \tag{28}$$

Starting with $\mathcal{L}_{\mathrm{BSDE}}$ from (23), we obtain the losses

$$\widehat{\mathcal{L}}^{\mathrm{exp}}_{\mathrm{BSDE}}(\widehat{V}_n) = \mathbb{E}\left[\left(\widehat{V}_n(\widehat{X}_n) - h_{n+1}\Delta t - \widehat{V}_{n+1}(\widehat{X}_{n+1}) + \sigma^\top(\widehat{X}_n, t_n)\nabla\widehat{V}_n(\widehat{X}_n)\cdot\xi_{n+1}\sqrt{\Delta t}\right)^2\right], \tag{29}$$

and

$$\widehat{\mathcal{L}}^{\mathrm{imp}}_{\mathrm{BSDE}}(\widehat{V}_n) = \mathbb{E}\left[\left(\widehat{V}_n(\widehat{X}_n) - h_{n}\Delta t - \widehat{V}_{n+1}(\widehat{X}_{n+1}) + \sigma^\top(\widehat{X}_n, t_n)\nabla\widehat{V}_n(\widehat{X}_n)\cdot\xi_{n+1}\sqrt{\Delta t}\right)^2\right]. \tag{30}$$

Note that $\widehat{\mathcal{L}}^{\mathrm{exp}}_{\mathrm{BSDE}}$ is explicit in the sense that h_{n+1} only depends on quantities that have already been computed in the previous iteration steps, whereas $\widehat{\mathcal{L}}^{\mathrm{imp}}_{\mathrm{BSDE}}$ is implicit in the sense that the nonlinear function h_n contains the approximating function $\widehat{V}_n$, with respect to which the loss shall be minimized. While implicit numerical schemes may in principle promise improved numerical stability (see e.g. Chassagneux and Richou (2015)), explicit schemes tend to lead to shorter runtimes. We will demonstrate those computational tradeoffs in multiple numerical examples in Section 5.

For $\mathcal{L}_{\mathrm{proj}}$, only the explicit version is available and we obtain

$$\widehat{\mathcal{L}}^{\mathrm{exp}}_{\mathrm{proj}}(\widehat{V}_n) = \mathbb{E}\left[\left(\widehat{V}_n(\widehat{X}_n) - h_{n+1}\Delta t - \widehat{V}_{n+1}(\widehat{X}_{n+1})\right)^2\right], \tag{31}$$

where the approximation $V(X_{t_{n+1}}, t_{n+1}) \approx \varphi(X_{t_{n+1}}, t_{n+1})$ is used at the right end point of the integral in (25).

Remark 4 (Gradient forcing). *The gradient $\nabla\widehat{V}_n$ is not explicitly part of the objective in* (31)*, as $\widehat{V}_{n+1}$ is assumed to be known from the previous iteration. In contrast, both* (29) *and* (30) *contain $\nabla\widehat{V}_n$ (note that the short-hand h_n depends on $\nabla\widehat{V}_n$ as well). These differences are partly rooted in the formulas* (23) *and* (25)*, given that the former is stated in terms of $\nabla\varphi$ rather than ∇V. In challenging (typically high-dimensional) problems where the solution V can only be approximated up to a certain relatively low accuracy, it is plausible that* (29), (30) *and* (31) *encourage differing trade-offs between the accuracy of $\widehat{V}$ and $\nabla\widehat{V}$; a stark example of this phenomenon will be presented in Section 5.2.*

Remark 5 (Previous work). *The loss $\widehat{\mathcal{L}}^{\mathrm{exp}}_{\mathrm{proj}}$ defined in* (31) *has been considered and analyzed extensively in earlier works on the numerical treatment of BSDEs, see Chessari et al. (2023) as well as references therein. A convenient property of this loss is that its explicit nature typically leads to linear regression-type problems that allow for efficient solvers, see Section 4. The loss $\widehat{\mathcal{L}}^{\mathrm{imp}}_{\mathrm{BSDE}}$ defined in* (30) *has been considered in Huré et al. (2020) and Germain et al. (2022) in the context of deep learning. Due to its implicit form, the corresponding algorithms rely on iterative solvers, possibly rendering optimization inefficient. The loss $\widehat{\mathcal{L}}^{\mathrm{exp}}_{\mathrm{BSDE}}$ defined in* (29) *is new to the best of our knowledge. It relies on an explicit numerical discretization and, in comparison with $\widehat{\mathcal{L}}^{\mathrm{exp}}_{\mathrm{proj}}$, enjoys favorable robustness properties when $\varphi \approx V$, see Section 3.*

Remark 6 (Discretization of the stochastic integral)**.** *Whilst the Riemann integral in $\mathcal{L}_{\mathrm{BSDE}}$ can be straightforwardly discretized in multiple different ways, alternative discretizations of the (Itô) stochastic integral incur (Stratonovich-type) correction terms in the limit as $\Delta t \to 0$. For example, we may rewrite* (23) *in terms of backward integrals (denoted by $\mathrm{d}\overleftarrow{W}$, see Kunita (2019, Section 2.7)),*

$$\mathcal{L}_{\mathrm{BSDE}}(\varphi) = \mathbb{E}\left[\left(\varphi(X_{t_n}, t_n) - V(X_{t_{n+1}}, t_{n+1}) - \int_{t_n}^{t_{n+1}} h(X_s, s, \varphi(X_s, s), (\sigma^\top \nabla\varphi)(X_s, s))\,\mathrm{d}s \right.\right.$$
$$\left.\left. + \int_{t_n}^{t_{n+1}} (\sigma^\top \nabla\varphi)(X_s, s)\cdot \mathrm{d}\overleftarrow{W}_s - \int_{t_n}^{t_{n+1}} \mathrm{Tr}\left(\sigma\sigma^\top \mathrm{Hess}\,\varphi\right)(X_s)\,\mathrm{d}s\right)^2\right], \tag{32}$$

where the last term is a correction that has been computed according to equation (2.48) in Kunita (2019). Discretizing (32) *then leads to*

$$\mathcal{L}^{\exp}_{\mathrm{BSDE,back}}(\widehat{V}_n) = \mathbb{E}\left[(\widehat{V}_n(\widehat{X}_n) - \widehat{h}_{n+1}\Delta t - \widehat{V}_{n+1}(\widehat{X}_{n+1}) + \sigma^\top\nabla\widehat{V}_{n+1}(\widehat{X}_{n+1})\cdot\xi_{n+1}\sqrt{\Delta t}\right.$$
$$\left. - \mathrm{Tr}\left(\sigma\sigma^\top \mathrm{Hess}\,\widehat{V}_{n+1}\right)(\widehat{X}_{n+1})\Delta t)^2\right], \tag{33}$$

where now both the Riemann and the stochastic integral are approximated by the right endpoints, yielding a fully explicit numerical scheme. In our numerical experiments, however, we have not observed clear advantages in terms of accuracy or robustness, and reserve a more careful evaluation for future work.

Remark 7 (Losses on the entire time interval)**.** *Dividing $[0, T]$ into multiple smaller time intervals is attractive since the ensuing subproblems might be easier to solve (cf. Figure 1). However, the continuous-time loss $\mathcal{L}_{\mathrm{BSDE}}$ defined in* (23) *(and for linear PDEs also the loss $\mathcal{L}_{\mathrm{proj}}$, see Richter and Berner (2022)) can also be used to approximate the solution on the whole time interval at once, choosing $N = 1$ with $t_0 = 0$ and $t_1 = T$. In order to ensure that the magnitude of the discretization errors remains controlled, one would then need to discretize the integrals on multiple grid points. This, however, automatically leads to implicit schemes and only iterative solvers (or 'shooting methods') seem appropriate, see e.g. E et al. (2017) and Nüsken and Richter (2023).*

Remark 8 (Error propagation)**.** *When dividing the problem into subproblems, on the other hand, we might need to accept a potential propagation of approximation errors, originating from the fact that in implementations the approximation $\widehat{V}_{n+1}(\cdot) \approx V(\cdot, t_{n+1})$ (needed as a 'terminal condition' for deriving the individual losses) will usually not be exact, cf. Gobet (2016, Section 8.3.3)).*

A summary of the algorithms is later given in Algorithm 4.

3. Robustness properties of the loss functionals

In the previous section we have derived numerical schemes for solving parabolic PDEs based on the iterative minimization of appropriate loss functionals. The two losses (23) and (25) are both valid in the sense of (22), i.e. they both yield the solution to the PDE (5) assuming that the expectations involved can be computed exactly and the associated minimization procedure leads to a global minimum. This assumption is not realistic in practice, however. Consequently, this section shall study robustness properties of the losses when the expectations are approximated by a finite sample Monte Carlo estimator. For $\mathcal{L}_{\mathrm{BSDE}}$ and $\mathcal{L}_{\mathrm{proj}}$ as defined in (23) and (25), the natural estimators are given by

$$\mathcal{L}^{(K)}_{\mathrm{proj}}(\varphi) = \frac{1}{K}\sum_{k=1}^{K}\Bigg(\varphi(X^{(k)}_{t_n}, t_n) - V(X^{(k)}_{t_{n+1}}, t_{n+1}) - \int_{t_n}^{t_{n+1}} h(X^{(k)}_s, s, V(X^{(k)}_s, s), (\sigma^\top \nabla V)(X^{(k)}_s, s))\,\mathrm{d}s\Bigg)^2, \tag{34}$$

and

$$\mathcal{L}^{(K)}_{\mathrm{BSDE}}(\varphi) = \frac{1}{K}\sum_{k=1}^{K}\Bigg(\varphi(X^{(k)}_{t_n}, t_n) - V(X^{(k)}_{t_{n+1}}, t_{n+1}) + \int_{t_n}^{t_{n+1}} (\sigma^\top\nabla\varphi)(X^{(k)}_s, s)\cdot \mathrm{d}W^{(k)}_s - \int_{t_n}^{t_{n+1}} h(X^{(k)}_s, s, \varphi(X^{(k)}_s, s), (\sigma^\top \nabla \varphi)(X^{(k)}_s, s))\,\mathrm{d}s\Bigg)^2, \tag{35}$$

where $K \in \mathbb{N}$ is the sample size and $(X^{(k)}_s)_{s\in[0,T],k=1,\ldots,K}$ denote independent and identically distributed copies of the diffusion process (11), driven by the independent Brownian motions $(W^{(k)}_s)_{s\in[0,T]}$.

We now aim to study fluctuations of these Monte Carlo approximations at the solution $\varphi = V$ by way of computing the variance associated to (34) and (35). Our results will shed light on the situation when $\varphi \approx V$, which is the relevant regime towards the end of iterative optimization algorithms (such as those considered in Section 4). The following result is in direct correspondence to the observations made in the introduction concerning the differences between the regression objectives (3) and (4).

Proposition 9 (Expectation and variance of the losses at the solution)**.** *Consider the Monte Carlo estimators* (34) *and* (35)*, evaluated at the solution* $\varphi = V$*. The former has nonzero expectation and variance, namely*

$$\mathbb{E}\left[\mathcal{L}^{(K)}_{\mathrm{proj}}(V)\right] = \mathrm{Var}\left(\int_{t_n}^{t_{n+1}} (\sigma^\top\nabla V)(X_s, s)\cdot \mathrm{d}W_s\right), \tag{36a}$$

$$\mathrm{Var}\left(\mathcal{L}^{(K)}_{\mathrm{proj}}(V)\right) = \frac{1}{K}\,\mathrm{Var}\left(\left(\int_{t_n}^{t_{n+1}} (\sigma^\top\nabla V)(X_s, s)\cdot \mathrm{d}W_s\right)^2\right). \tag{36b}$$

In contrast, the latter is zero almost surely,

$$\mathcal{L}^{(K)}_{\mathrm{BSDE}}(V) = 0, \qquad \text{a.s.}, \tag{37}$$

implying in particular that both its expectation and variance vanish.

Proof Substituting $\varphi = V$ into $\mathcal{L}_{\text{proj}}$ and using the relation (20) we see that

$$\mathcal{L}_{\text{proj}}(V) = \mathbb{E}\left[\left(\int_{t_n}^{t_{n+1}} (\sigma^\top \nabla V)(X_s, s) \cdot \mathrm{d}W_s\right)^2\right] = \operatorname{Var}\left(\int_{t_n}^{t_{n+1}} (\sigma^\top \nabla V)(X_s, s) \cdot \mathrm{d}W_s\right), \tag{38}$$

from which (36a) and (36b) follow directly. For $\mathcal{L}_{\text{BSDE}}(V)$ note that

$$\begin{aligned} V(X_{t_n}^{(k)}, t_n) - V(X_{t_{n+1}}^{(k)}, t_{n+1}) - \int_{t_n}^{t_{n+1}} h(X_s^{(k)}, s, V(X_s^{(k)}, s), (\sigma^\top \nabla V)(X_s^{(k)}, s))\, \mathrm{d}s \\ + \int_{t_n}^{t_{n+1}} (\sigma^\top \nabla V)(X_s^{(k)}, s) \cdot \mathrm{d}W_s^{(k)} = 0 \end{aligned} \tag{39}$$

holds almost surely for every $k \in \{1, \dots, K\}$, see also (20), from which (37) follows immediately and which implies that both the expectation and the variance of the estimator versions are zero. ■

Theorem 9 shows that the Monte Carlo estimator $\mathcal{L}_{\text{proj}}^{(K)}$ exhibits noise even at the solution $\varphi = V$. This is an undesirable property, since stochastic optimization algorithms usually degrade in performance in the face of high variance estimators, necessitating smaller learning rates and longer convergence times (Bottou et al. (2018)). Close to an optimum, the presence of noise is likely to induce instabilities and prevent the algorithm from settling down on an accurate solution.

The estimator $\mathcal{L}_{\text{BSDE}}^{(K)}$ on the other hand possesses variance zero at the optimum, promising statistical advantages in the optimization routines. In particular, once the approximation is close to the solution the optimization algorithm is expected to be robust and stay close to the optimum. We refer to Section 5 for numerical evidence that illustrates those findings.

To extend our analysis to stochastic gradient descent and its variants, we next study the fluctuations of gradient estimators of the losses. For this, we recall the notion of functional derivative (see e.g. Section 5.2 in Siddiqi and Nanda (1986)). For the following definition, we require $\mathcal{L}$ to be defined on a set of functions that is also a vector space. For definiteness, we consider $\mathcal{L} : C_b^1(\mathbb{R}^d \times [0, T]) \to \mathbb{R}$ (that is, implicitly, $\mathcal{H} \subset C_b^1(\mathbb{R}^d \times [0, T])$), but note that the boundedness assumption could be relaxed considerably.

Definition 10 (Gâteaux derivative)**.** *For $\varphi, \psi \in C_b^1(\mathbb{R}^d \times [0, T])$, we say that $\mathcal{L}$ is* Gâteaux differentiable *at φ in direction ψ if the mapping*

$$\varepsilon \mapsto \mathcal{L}(\varphi + \varepsilon\psi) \tag{40}$$

is differentiable at $\varepsilon = 0$. The Gâteaux derivative of $\mathcal{L}$ at φ in direction ψ is then defined as

$$\frac{\delta}{\delta\varphi}\mathcal{L}(\varphi; \psi) := \frac{\mathrm{d}}{\mathrm{d}\varepsilon}\Big|_{\varepsilon=0} \mathcal{L}(\varphi + \varepsilon\psi). \tag{41}$$

We can now investigate the variances of the gradients of the estimated losses. In fact, the following proposition shows that the variance of the gradient of the Monte Carlo estimator of the loss $\mathcal{L}_{\text{BSDE}}$, as defined in (35), vanishes at the solution $\varphi = V$.

Proposition 11 (Variance of the gradients at the solution). *For every direction $\psi \in C_b^1(\mathbb{R}^d \times [0,T])$ it holds*

$$\operatorname{Var}\left(\frac{\delta}{\delta\varphi}\bigg|_{\varphi=V} \mathcal{L}^{(K)}_{\mathrm{BSDE}}(\varphi;\psi) \right) = 0. \tag{42}$$

Proof The proof of the identities (42) and (43) below can be found in Appendix A. ■

Remark 12 (Variance of gradients of the projection loss). *The remarkable property* (42) *does not hold for the Monte Carlo estimator of the projection loss, defined in* (34)*. In fact, one can show that*

$$\operatorname{Var}\left(\frac{\delta}{\delta\varphi}\bigg|_{\varphi=V} \mathcal{L}^{(K)}_{\mathrm{proj}}(\varphi;\psi) \right) = \frac{4}{K}\mathbb{E}\left[(\psi(X_{t_n},t_n))^2 \int_{t_n}^{t_{n+1}} (\sigma^\top \nabla V)(X_s,s))^2 \, \mathrm{d}s \right], \tag{43}$$

see Appendix A for details. Clearly, the right-hand side of (43) *vanishes only in very exceptional cases (for instance, if $\sigma^\top \nabla V \equiv 0$ on $[t_n, t_{n+1}]$).*

4. Tensor trains for solving BSDEs

In this section we explain the tensor train approach (Oseledets, 2011) towards representing the solution V to the PDE (5), following Holtz et al. (2012a) and Sallandt (2022, Chapter 4). We first introduce the model and its representation as a coefficient tensor. We then introduce an optimization procedure used within our methods – the alternating least squares algorithm (ALS) for regression-like problems. Finally, we will see how ALS can be used to minimize loss funtionals developed in the previous sections. We will repeatedly start by recalling the procedure for a simple linear ansatz space and then transfer this knowledge to explain the same procedure for the set of tensor trains.

We start with the classical model for functions of d real variables, $V : \mathbb{R}^d \to \mathbb{R}$, using tensorization of functions of one real variable each, $\phi_i : \mathbb{R} \to \mathbb{R}$. Given an appropriate set of such functions $\Phi = \{\phi_1, \ldots, \phi_m : \mathbb{R} \to \mathbb{R}\}$, functions of d real variables can be constructed as[2]

$$V(x_1, \ldots, x_d) = \sum_{i_1=1}^{m} \cdots \sum_{i_d=1}^{m} c_{i_1,\ldots,i_d} \phi_{i_1}(x_1) \ldots \phi_{i_d}(x_d). \tag{44}$$

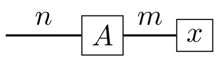
The coefficient tensor $c \in \mathbb{R}^{m \times \cdots \times m} = \mathbb{R}^{m^d}$ is said to be of order d and suffers from the curse of dimensionality: the number of its coefficients grows exponentially, severely limiting the usage of this classical model in higher dimensions. To simplify the presentation we introduce the Python-like notation $c_{i_1,\ldots,i_d} = c[i_1, \ldots, i_d]$. In order to introduce the tensor train model

2. As hinted at by the choice of variable names, we think of (44) as an approximation or representation of the solution to the PDE (5). This correspondence is only vague, as V in (44) does not depend on time. To be more precise, we should think of (44) as one instance of $\widehat{V}_i$ within the iterative scheme depicted in Figure 1, that is, t is fixed.

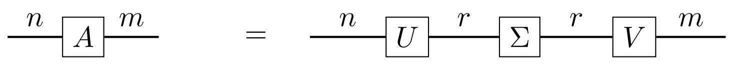
we first define the contraction $\circ$ between the last index of a tensor $w_1 \in \mathbb{R}^{r_1 \times m \times r_2}$ with the first index of another tensor $w_2 \in \mathbb{R}^{r_2 \times m \times r_3}$ as

$$w = w_1 \circ w_2 \in \mathbb{R}^{r_1 \times m \times m \times r_3}, \quad w[j_1, i_1, i_2, j_3] = \sum_{j_2=1}^{r_2} w_1[j_1, i_1, j_2] w_2[j_2, i_2, j_3]. \tag{45}$$

Using this contraction operation, the standard matrix-vector product of $A \in \mathbb{R}^{n \times m}$ and $x \in \mathbb{R}^m$ is given as $A \circ x \in \mathbb{R}^n$. Similarly, the singular value decomposition (SVD) of A can be written as $A = U \circ \Sigma \circ V$. Tensors and their contractions can be efficiently visualized as graphs in which nodes stand for the individual tensors and contractions are denoted as edges between them, cf. Figure 2. We refer to such collections of tensors and contractions between the tensors as *tensor networks*. Using the $\circ$-notation we can define



(a) A tensor network representing the order 1 tensor resulting from the matrix-vector multiplication $Ax = A \circ x \in \mathbb{R}^n$.



(b) Graphical representation of the SVD as a tensor network.

Figure 2: Graphical notation of simple tensors and tensor networks.

the tensor train representation:

Definition 13 (Tensor train)**.** *Let* $c \in \mathbb{R}^{m \times \cdots \times m} = \mathbb{R}^{m^d}$*. A factorization*

$$c = u_1 \circ u_2 \circ \cdots \circ u_d, \tag{46}$$

where $u_1 \in \mathbb{R}^{m \times r_1}$*,* $u_i \in \mathbb{R}^{r_{i-1} \times m \times r_i}$*,* $2 \leq i \leq d-1$*,* $u_d \in \mathbb{R}^{r_{d-1} \times m}$*, is called* tensor train representation (TT-representation) *of* c*. We refer to the individual tensors* u_i *as* component tensors*. The tuple* $(r_1, \ldots, r_{d-1}) \in \mathbb{N}^{d-1}$ *is referred to as the* representation rank *and is a property of the specific representation* (46)*. In contrast, the* tensor train rank (TT-rank) *of* c *is defined as the minimal rank tuple* $r = (r_1, \ldots, r_{d-1})$ *such that there exists a TT-representation of* c *with representation rank* r*. Here, minimality of the rank is defined in terms of the partial order relation on* $\mathbb{N}^{d-1}$ *given by*

$$s \preceq t \iff s_i \leq t_i \qquad \text{for all } i \in \{1, \ldots, d-1\},$$

for $s = (s_1, \ldots, s_{d-1})$*,* $t = (t_1, \ldots, t_{d-1}) \in \mathbb{N}^{d-1}$*.*

The unique minimal rank tuple can for example be computed using a multi-linear SVD, cf. Holtz et al. (2012b). Tensor trains have the appealing property that, assuming that the ranks stay bounded, the number of coefficients grows linearly with the spatial dimension. More precisely, the number of coefficients is at most of order $\mathcal{O}(dmr)$, where $r = \max_i r_i$. The TT-representation is said to be orthogonalized if for some index μ, the component tensors $u_1, \ldots, u_{\mu-1}$ are left-orthogonal and $u_{\mu+1}, \ldots, u_d$ are right-orthogonal, see e.g. Wolf (2019) for a precise definition. The index μ associated to the possibly non-orthogonal component tensor is called the core position. Such an orthogonal representation can be obtained by a sequence of QR-decompositions, and the position of the core can be "moved" efficiently (for example, transforming an orthogonal representation with core index μ into one with core index $\mu + 1$ or $\mu - 1$). We again refer to Wolf (2019) for details.

Using the graphical notation for tensor networks we can represent a TT as in Figure 3.

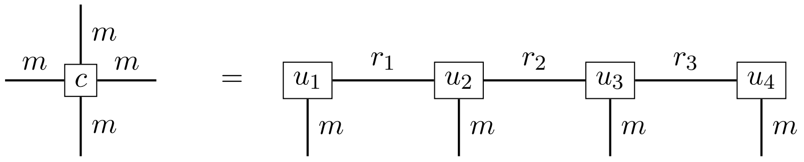


Figure 3: An order-4 tensor with a possible tensor-train representation.

In order to represent functions of several real variables we combine tensor networks with the set of ansatz functions $\Phi = \{\phi_1, \ldots, \phi_m : \mathbb{R} \to \mathbb{R}\}$, which we shall from now on assume to be linearly independent. These ansatz functions have to be predetermined. To simplify the presentation, we overload the notation and introduce $\Phi : \mathbb{R} \to \mathbb{R}^m$, $x \mapsto (\phi_1(x), \ldots, \phi_m(x))^\top$. Now we can contract $\Phi(x)$ with the component tensors as shown in Figure 4 to obtain the function V. We refer to this type of tensor network as a *functional tensor train.*

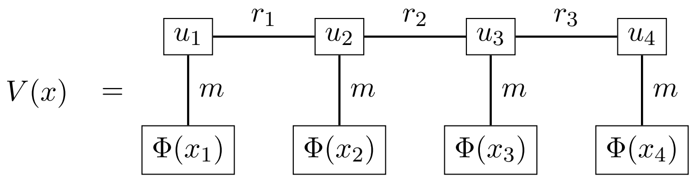


Figure 4: A function of 4 real variables and a TT-representation of its coefficient tensor.

4.1 Complexity estimates for common operations

The functional tensor train (Figure 4) has appealing properties with respect to the computational cost of common operations such as the evaluation of the function value, its gradient as well as its Laplacian. All of these operations can be performed in $\mathcal{O}(dmr^2)$, where $r = \max_i r_i$. Although the same asymptotic complexities can be achieved using reverse-mode automatic differentiation (Griewank and Walther, 2008, Chapter 4), the constructions detailed here allow us to construct efficient regression schemes that incorporate gradient information, see Section 4.3 below. In what follows, we use the notation from Sallandt (2022); Oster et al. (2022).

Evaluation of V: Let us consider the evaluation of V at a point $x = (x_1, \ldots, x_d)^\top \in \mathbb{R}^d$. We first need to compute $\Phi(x_l)$ for $l \in \{1, \ldots, d\}$, which results in md evaluations of the functions $\phi_i : \mathbb{R} \to \mathbb{R}$ in total. We then proceed by performing the contractions in Figure 4. For fixed l, the contraction of $\Phi(x_l)$ with the component tensors is carried out from right to left. Using the $\circ$-notation, this means that

$$V(x) = u_1 \circ \cdots \circ \bigg(\Big(u_{d-1} \circ \big(u_d \circ \Phi(x_d) \big) \Big) \circ \Phi(x_{d-1}) \bigg) \circ \cdots \circ \Phi(x_1). \tag{47}$$

The complexity of the first contraction $v_d = u_d \circ \Phi(x_d)$ is $\mathcal{O}(rm)$. The following contraction $u_{d-1} \circ v_d \circ \Phi(x_{d-1})$ (and also the contractions following thereafter) is of order $\mathcal{O}(r^2 m)$, while the last contraction is again of order $\mathcal{O}(rm)$. This results in a total complexity of order $\mathcal{O}(dr^2 m)$.

Evaluation of ∇V: In order to compute the gradient ∇V, we start with the evaluation of partial derivatives. Defining the shorthand $\Phi'(x) = (\phi_1'(x), \dots, \phi_m'(x))^\top$, the partial derivative $\partial_{x_l} V(x_1, \dots, x_d)$ for the case $d = 4$ and $l = 2$ is given in Figure 5. Again, using

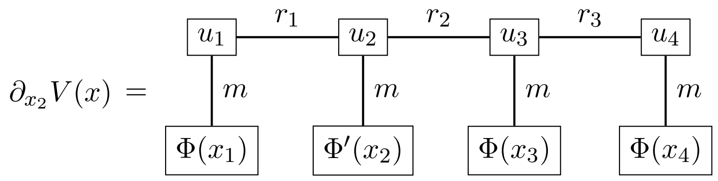


Figure 5: Partial derivative of a 4-dimensional function in TT-representation.

the $\circ$-notation, but leaving out the brackets for simplicity,[3] we obtain

$$\partial_{x_l} V(x) = u_1 \circ \dots \circ u_{l-1} \circ u_l \circ u_{l+1} \circ \dots \circ u_d \circ \\ \circ \Phi(x_d) \circ \dots \circ \Phi(x_{l+1}) \circ \Phi'(x_l) \circ \Phi(x_{l-1}) \circ \dots \circ \Phi(x_1). \quad (48)$$

The complexity of evaluating the partial derivative hence coincides with the complexity of evaluating V, i.e. $\mathcal{O}(r^2 md)$. In order to compute the gradient, d of these contractions have to be performed, meaning that the complexity of this naive gradient evaluation is $\mathcal{O}(r^2 md^2)$. However, we can improve on that by noticing that in $\partial_{x_l} V(x)$ and $\partial_{x_{l+1}} V(x)$ a number of contractions appear repeatedly. Saving these contractions yields an efficient way of computing the gradient.

To make this precise we define the contractions

$$\Psi_l^+(x_{l+1}, \dots, x_d) = u_{l+1} \circ \dots \circ u_d \circ \Phi(x_d) \circ \dots \circ \Phi(x_{l+1}) \in \mathbb{R}^{r_l}, \quad (49)$$

for $l = 1, \dots, d-1$, as well as

$$\Psi_l^-(x_1, \dots, x_{l-1}) = u_1 \circ \dots \circ u_{l-1} \circ \Phi(x_{l-1}) \circ \dots \circ \Phi(x_1) \in \mathbb{R}^{r_{l-1}}, \quad (50)$$

for $l = 2, \dots, d$. In words, Ψ_l^+ is the contraction of every component tensor with index larger than l, while Ψ_l^- is the contraction of every component tensor with index smaller than l. Using (49) and (50), we can rewrite (48) in the form

$$\partial_{x_l} V(x_1, \dots, x_d) = \big((\Psi_l^-(x_1, \dots, x_{l-1}) \circ u_l) \circ \Psi_l^+(x_{l+1}, \dots, x_d)\big) \circ \Phi'(x_l), \quad (51)$$

for $l = 2, \dots, d-1$. The edge cases $l = 1$ and $l = d$ are covered by

$$\partial_{x_1} V(x_1, \dots, x_d) = u_l \circ \Psi_l^+. \quad (52a)$$

3. The order of the contractions is the same as in (47), and can be inferred from the dimensions of the individual tensors.

We furthermore observe the recursive identities

$$\Psi_l^+ = (u_{l+1} \circ \Psi_{l+1}^+) \circ \Phi_{l+1}, \quad \Psi_l^- = \Phi_{l-1} \circ (\Psi_{l-1}^- \circ u_{l-1}), \tag{53}$$

omitting the arguments of Ψ_l^+ and Ψ_l^-. These allow us to reuse already performed contractions when computing partial derivatives with respect to different variables, and, in combination with (51), to arrive at Algorithm 1. Note that every micro-step in the algorithm has complexity $\mathcal{O}(r^2 m)$. Taking into account the for-loops, this amounts to $\mathcal{O}(dr^2 m)$ and compares favorably to $\mathcal{O}(d^2 r^2 m)$ for the naive implementation. Finally, we remark that the Laplacian can be evaluated efficiently in a similar fashion, see Kazeev and Khoromskij (2012).

Algorithm 1: Computing the gradient of a function V in the TT-format.

```
Input  : A function $V$ in TT-format (see equation (47) and Figure 4) and its
         component tensors $u_1, \dots, u_d$, one-dimensional basis functions $\phi_1, \dots, \phi_n$,
         evaluation point $x = (x_1, \dots, x_d)^\top \in \mathbb{R}^d$.
Output: The gradient $\nabla V(x)$.
for $l = 1, \dots, d-1$ do
    Calculate $\Psi_l^-(x_1, \dots x_{l-1})$ using the recursive formula (53).
end
for $l = d, \dots, 1$ do
    Calculate $\Psi_l^+(x_{l+1}, \dots, x_d)$ using the recursive formula (53).
end
for $l = d, \dots, 1$ do
    Calculate $\nabla V(x)[l] = \frac{\partial V}{\partial x_l}(x)$ using (51).
end
```

4.2 Optimization on the TT manifold

Now that we have established how to represent the high-dimensional function V, we next cover the process of finding optimal coefficients in (47), related to the component tensors u_i. We make use of the *alternating least squares* (ALS) algorithm (Holtz et al., 2012a), which replaces the problem of finding all coefficients at once by a sequence of low-dimensional sub-problems, where only one component tensor is optimized in every iteration. This is possible due to the multi-linearity of the tensor train representation.

Digression: Regression problems on linear spaces. In Section 2 we have established that BSDEs can be solved by minimizing appropriate loss functionals. Before explaining how to minimize these loss functionals for the set of tensor trains, we first recall linear regression on simpler linear spaces. Then, we discuss the same problem on the set of tensor trains, before finally extending the method to the loss functionals from Section 2 (involving gradient dependencies and leveraging the construction from Algorithm 1)

We recall the ordinary linear regression problem

$$\min_{\varphi \in U} \mathcal{L}^{(K)}(\varphi), \tag{54}$$

where U is a finite-dimensional linear space of functions on $\mathbb{R}^d$, and the loss functional $\mathcal{L}$ takes the form

$$\mathcal{L}^{(K)}(\varphi) = \frac{1}{K}\sum_{k=1}^{K} |\varphi(x^{(k)}) - y^{(k)}|^2, \tag{55}$$

for given samples $x^{(k)} \in \mathbb{R}^d$ and values (or measurements) $y^{(k)} \in \mathbb{R}$, $k = 1, \ldots, K$. For a basis $b_1, \ldots, b_N$ of U and a function representation $\varphi(x) = \sum_{i=1}^N c_i b_i(x)$, the optimal coefficients[4] $c = (c_1, \ldots, c_N)^\top \in \mathbb{R}^N$ are given by

$$c = (A^\top A)^{-1} A^\top y, \tag{56}$$

where $A = (a_{ij}) \in \mathbb{R}^{K\times N}$ has coefficients $a_{ij} = b_j(x^{(i)})$, and $y = (y^{(1)}, \ldots, y^{(K)})^\top \in \mathbb{R}^K$, assuming a sufficient amount and nondegeneracy of the samples for $A^\top A$ to be invertible (Wasserman, 2004, Chapter 13).

To explain the extension of ordinary least squares achieved by the ALS algorithm, we first introduce some notation. The set of tensor trains with fixed rank $r \in \mathbb{N}^{d-1}$ is denoted by

$$\mathcal{M}_r = \{c = u_1 \circ \cdots \circ u_d \mid \text{TT-rank}(c) = r\}, \tag{57}$$

see Definition 13 for the dimensions of the component tensors u_i. It is important to note that $\mathcal{M}_r$ is not a linear subspace of $\mathbb{R}^{m\times\cdots\times m}$ because of the nonlinearity of the contraction operation $\circ$; it is however a submanifold, sometimes referred to as the TT-manifold of rank r (Holtz et al., 2012b). We will also need the subsets

$$\mathcal{M}_r^l(u_1, \ldots, u_{l-1}, u_{l+1}, \ldots, u_d) := \left\{c = u_1 \circ \ldots \circ u_d : \quad u_l \in \mathbb{R}^{r_{l-1}\times m\times r_l}\right\} \subset \mathcal{M}_r, \tag{58}$$

obtained from $\mathcal{M}_r$ by fixing the component tensors $u_1, \ldots, u_{l-1}, u_{l+1}, \ldots, u_d$ and varying only u_l. In the following, we will write $\mathcal{M}_r^l$ for notational convenience whenever the particular choices of the fixed component tensors are not relevant for the argument. A key observation is that $\mathcal{M}_r^l(u_1, \ldots, u_{l-1}, u_{l+1}, \ldots, u_d)$ is a linear subspace of $\mathbb{R}^{m\times\cdots\times m}$ for any $l \in \{1, \ldots, d\}$ and fixed component tensors, in contrast to $\mathcal{M}_r$. This observation will allow us to efficiently optimize over $\mathcal{M}_r$ using the explicit formula (56) by iteratively restricting to the subspaces $\mathcal{M}_r^l$. Associated to the sets of tensor trains defined in (57) and (58) and a set of basis functions $\Phi = \{\phi_1, \ldots, \phi_m\}$, we obtain the sets of functional tensor trains $(\mathcal{M}_r, \Phi)$ and $(\mathcal{M}_r^l(u_1, \ldots, u_{l-1}, u_{l+1}, \ldots, u_d), \Phi)$ by using the construction principle in (47) and Figure 4. For instance,

$$(\mathcal{M}_r, \Phi) = \Big\{V(x) = u_1 \circ \cdots \circ \Big(\Big(u_{d-1} \circ \big(u_d \circ \Phi(x_d)\big)\Big) \circ \Phi(x_{d-1})\Big) \circ \cdots \circ \Phi(x_1), \\ \text{with} \quad u_1 \circ \ldots \circ u_d \in \mathcal{M}_r\Big\}, \tag{59}$$

and similarly for $(\mathcal{M}_r^l, \Phi)$. Clearly, $(\mathcal{M}_r, \Phi)$ and $(\mathcal{M}_r^l, \Phi)$ inherit key properties from $\mathcal{M}_r$ and $\mathcal{M}_r^l$; in particular, $(\mathcal{M}_r^l, \Phi)$ is a linear function space, while $(\mathcal{M}_r, \Phi)$ is merely a (nonlinear) manifold.

4. Note that we have slightly abused the notation by redefining the quantity c. Previously, it was a coefficient tensor and now it is a coefficient vector. We revisited the basic linear regression in order to draw parallels with the more complex tensor train regression approach. Thus, from now on, c will again be a coefficient tensor.

We are now interested in replacing the linear space U in (54) by the TT-manifold $(\mathcal{M}_r, \Phi)$, i.e.,

$$\min_{\varphi \in (\mathcal{M}_r, \Phi)} \mathcal{L}^{(K)}(\varphi), \tag{60}$$

for a given rank $r \in \mathbb{N}^{d-1}$ and set of basis functions $\Phi = \{\phi_1, \ldots, \phi_m\}$. The ALS algorithm targets (60) by interatively restricting the optimization to $(\mathcal{M}_r^l, \Phi)$, cycling through $l = 1, \ldots, d$, while updating the fixed component tensors using the solutions to the sub-regression problems on $(\mathcal{M}_r^l, \Phi)$. More precisely, we set $l = 1$, start with an arbitrary initialization $u_2, \ldots, u_d$ of the component tensors and consider the *local regression problem*

$$\min_{\varphi \in (\mathcal{M}_r^1(u_2, \ldots, u_d), \Phi)} \mathcal{L}^{(K)}(\varphi). \tag{61}$$

Crucially, since $(M_r^1(u_2, \ldots, u_d), \Phi)$ is a linear function space, we can apply the solution formula (56) to obtain the missing component tensor $\widehat{u}_1$. Proceeding with $l = 2$ and the local regression problem (updated using $\widehat{u}_1$)

$$\min_{\varphi \in (\mathcal{M}_r^2(\widehat{u}_1, u_3, \ldots, u_d), \Phi)} \mathcal{L}^{(K)}(\varphi), \tag{62}$$

we perform the update $u_2 \mapsto \widehat{u}_2$. Following this procedure, we update all the component tensors in turn (proceeding again with $\widehat{u}_1$ after u_d) until an appropriate termination criterion is met (Holtz et al., 2012a). While global convergence to the optimum in (60) is not guaranteed, it is known that the loss $\mathcal{L}(V)$ decreases at every iteration, see Holtz et al. (2012a, Section 3.4).

We next discuss the details of an efficient implementation, making use of the contractions Ψ_l^+ and Ψ_l^- defined in (49) and (50). Starting with the update of the first component tensor u_1 (i.e. $l = 1$ in the discussion above), we observe that Ψ_1^+ can be interpreted as an r_1-dimensional vector of real-valued functions of $d-1$ variables. Taking the tensor product with the ansatz functions in $\Phi = \{\phi_1, \ldots, \phi_m\}$, we obtain a set of $m \cdot r_1$ functions of d variables. We formally denote the linear span of functions by $\Phi \otimes \Psi_1^+$, and note that the corresponding coefficients can be stored in a matrix $c \in \mathbb{R}^{m \times r_1}$ whose dimensions equal those of u_1. As explained above, solving a regression problem of the type (56) then recovers optimal coefficients for the coefficient tensor u_1, under the assumption that $u_2, \ldots, u_d$ are fixed.

Similarly, local regression problems can be formulated to obtain optimal coefficients for the component tensors u_l for $l \in \{1, \ldots, d\}$, the relevant function spaces being $\Psi_l^- \otimes \Phi \otimes \Psi_l^+$. As in Section 4.1 (see in particular Algorithm 1), we can use the identities (53) to compute Ψ_l^+ from Ψ_{l+1}^+ and vice-versa for Ψ_l^-, making the optimization efficient by saving contractions (these saved contractions are commonly referred to as 'stacks' within the ALS algorithm). To summarize, we state the procedure in Algorithm 2, using the convention that $\Psi_1^- = \Psi_d^+ = \operatorname{span}\{1\}$ is the space of constant functions.

To further improve performance, we make two essential modifications of the optimization problem (60). First, we introduce a regularization term in the loss functional, penalizing the Frobenius-norm of the coefficients in the tensor train (multiplied by a small parameter $\delta > 0$). In the case when $\Phi = \{\phi_1, \ldots, \phi_m\}$ consists of $H^2([a, b])$-orthogonal functions, Parseval's identity implies that this regularizing term can be identified with the norm in

Algorithm 2: Alternating Least Squares (ALS) for regression

```
Input  : An order $d$ tensor $u_1 \circ \cdots \circ u_d = A \in \mathbb{R}^{n_1 \times \cdots \times n_d}$, sample points $x^{(k)} \in \mathbb{R}^d$
         and data $y^{(k)} \in \mathbb{R}$.
Output: Optimized component tensors $u_1 \circ \cdots \circ u_d = A \in \mathbb{R}^{n_1 \times \cdots \times n_d}$.
while not converged do
    for $l = d-1$ to $1$ do
        // build stacks
        Move core position to $l$ (using QR-decompositions).
        Compute (and store) $\Psi_l^+(x_{l+1}^{(k)}, \dots, x_d^{(k)})$ for all $k$, using (53).
    end
    for $l = 1$ to $d$ do
        // optimize component tensors
        Move core position to $l$ (using QR-decompositions).
        Compute (and store) $\Psi_l^-(x_1^{(k)}, \dots, x_{l-1}^{(k)})$ for all $k$, using (53).
        Solve the regression problem (61) for the local basis $\Psi_l^- \otimes \Phi \otimes \Psi_l^+$, using
          (56). The solution yields the component tensor $u_l \in \mathbb{R}^{r_{l-1} \times m_l \times r_l}$.
    end
end
```

the Sobolev space with dominating mixed smoothness $H^2_{\mathrm{mix}}([a,b]^d)$, see Sickel and Ullrich (2009), so that the loss functional takes the form

$$\mathcal{L}^{(K)}(\varphi) = \frac{1}{K}\sum_{k=1}^{K} |\varphi(x^{(k)}) - y^{(k)}|^2 + \delta \|\varphi\|^2_{H^2_{\mathrm{mix}}([a,b]^d)}. \tag{63}$$

This regularization term can straightforwardly be incorporated in the computational procedure; we simply add $\delta\,\mathrm{Id}$ to the matrix $A^\top A$ in (56).

The second modification slightly alters the model space $(\mathcal{M}_r, \Phi)$. Oftentimes the terminal condition g cannot be represented exactly (or conveniently) within $(\mathcal{M}_r, \Phi)$ due to the specific tensor train structure. Addressing this, we can simply add g to the model and obtain a representation of V as in Figure 6 (in the same manner, other functions of particular relevance could be added to the model space as well). The modification of the ALS

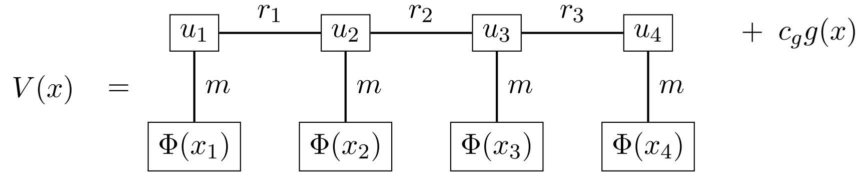


Figure 6: Graphical representation of $V : \mathbb{R}^4 \to \mathbb{R}$, with g included in the model space.

algorithm for this modified model is straightforward, as the local basis $\Psi_l \otimes \Phi \otimes \Psi_l$ can be modified to $(\Psi_l \otimes \Phi \otimes \Psi_l) \oplus g$, thus increasing the dimension of the local regression problem to $r_l m r_{l+1} + 1$.

Remark 14. *In Götte et al. (2021) as well as Trunschke (2021), sparsity promoting modifications of* (60) *have been considered, the latter of which is based on an L^1 rather than an L^2 penalty. These variations could reduce the number of samples needed for accurate estimation, further boosting the performance of the algorithm.*

4.3 Handling gradient-dependent loss functionals

The loss functionals $\widehat{\mathcal{L}}^{\mathrm{exp}}_{\mathrm{BSDE}}$ and $\widehat{\mathcal{L}}^{\mathrm{imp}}_{\mathrm{BSDE}}$ (see equations (29) and (30)) are not of the form (55) because of their dependence on ∇V, and therefore cannot directly be optimized using Algorithm 2. However, their inherently similar structure can still be leveraged to efficiently approximate minimizers in a TT-setting; this is fundamentally due to the similarity between function and gradient evaluations in the tensor train format, cf. equations (47) and (48), and thus we heavily rely on Algorithm 1. More specifically, we can replace (55) by

$$\mathcal{L}^{(K)}(\varphi) = \frac{1}{K}\sum_{k=1}^{K} |\varphi(x^{(k)}) + \nabla\varphi(x^{(k)}) \cdot \Xi^{(k)} - y^{(k)}|^2, \tag{64}$$

with $x^{(k)} \in \mathbb{R}^d$, $\Xi^{(k)} \in \mathbb{R}^d$, and $y^{(k)} \in \mathbb{R}$ for $1 \le k \le K$. In the case of $\widehat{\mathcal{L}}^{\mathrm{exp}}_{\mathrm{BSDE}}$, for instance, we see that setting $y^{(k)} = h^{(k)}_{n+1}\Delta t + \widehat{V}_{n+1}(\widehat{X}_{n+1})$ and $\Xi^{(k)} = \sqrt{\Delta t}\sigma(\widehat{X}^{(k)}_n, t_n)\xi^{(k)}_{n+1}$ recovers the standard Monte Carlo estimator for (29).

As in Section 4.2, we first cover the problem of minimizing (64) over a finite-dimensional linear space of (differentiable) functions on $\mathbb{R}^d$, see the formulation in (54). Inserting a representation $\varphi(x) = \sum_{i=1}^N c_i b_i(x)$ in terms of smooth basis functions, the solution again takes the form (56), $c = (A^\top A)^{-1}A^\top y$, with a modified matrix $A \in \mathbb{R}^{N\times K}$,

$$A_{ik} = b_i(x^{(k)}) + \sum_{j=1}^{d} (\partial_{x_j} b_i(x^{(k)})[j])\Xi^{(k)}[j]. \tag{65}$$

As before we use the Python-like notation to access entries of a vector. In order to obtain an efficient algorithm saving contractions, we set out to find a recursive representation of function (and gradient) evaluations. The first term in (65) can be handled as in Section 4.2, using the operations defined in (49) and (50). To handle the second term, we define

$$\Theta^-_l(x_1,\dots,x_{l-1},\xi) = \sum_{j=1}^{l-1} \partial_{x_j}\Psi^-_l(x_1,\dots,x_{l-1})\xi[j], \tag{66a}$$

$$\Theta^+_l(x_{l+1},\dots,x_d,\xi) = \sum_{j=l+1}^{d} \partial_{x_j}\Psi^+_l(x_{l+1},\dots,x_d)\xi[j] \tag{66b}$$

and notice that

$$\begin{aligned}\Theta^-_l(x_1,\dots,x_{l-1},\xi) = {}& \Theta^-_{l-1}(x_1,\dots,x_{l-2},\xi) \circ u_{l-1} \circ \Phi(x_{l-1}) \\ & + \Psi(x_1,\dots,x_{l-2}) \circ u_{l-1} \circ \Phi(x_{l-1}).\end{aligned} \tag{67}$$

Finally, building on (49)-(50) and (67), the identity

$$\begin{aligned}\sum_{j=1}^{d} \partial_{x_j}\varphi(x)\xi[j] = \Theta_l^-(x_1,\dots,x_{l-1},\xi) \circ \Psi_l^+(x_{l+1},\dots,x_d) \circ \Phi(x_l) \\ + \Psi_l^-(x_1,\dots,x_{l-1}) \circ u_l \circ \Psi_l^+(x_{l+1},\dots,x_d) \circ \Phi'(x_l)\cdot \xi[l] \\ + \Psi_l^-(x_1,\dots,x_{l-1}) \circ u_l \circ \Theta_l^+(x_{l+1},\dots,x_d,\xi) \circ \Phi(x_l) \qquad (68)\end{aligned}$$

yields an efficient way to evaluate $\sum_{j=1}^{d} \partial_{x_j}\varphi(x)\xi[j]$. We summarize the resulting procedure in Algorithm 3.

Algorithm 3: Alternating Least Squares (ALS) for losses involving gradients

```
Input  : An order $d$ tensor $u_1 \circ \cdots \circ u_d = A \in \mathbb{R}^{n_1\times\cdots\times n_d}$, sample points $x^{(k)} \in \mathbb{R}^d$
         and data $y^{(k)} \in \mathbb{R}$.
Output: Optimized component tensors $u_1 \circ \cdots \circ u_d = A \in \mathbb{R}^{n_1\times\cdots\times n_d}$.
while not converged do
    for $l = d-1$ to $1$ do
        // build stacks
        Move core position to $l$ (using QR-decompositions).
        Compute (and store) $\Psi_l^+(x_{l+1}^{(k)},\dots,x_d^{(k)})$ for all $k$, using (53).
        Compute (and store) $\Theta_l^+(x_{l+1}^{(k)},\dots,x_d^{(k)},\xi^{(k)})$ for all $k$, using (67).
    end
    for $l = 1$ to $d$ do
        // optimize component tensors
        Move core position to $l$ (using QR-decompositions).
        Compute (and store) $\Psi_l^-(x_1^{(k)},\dots,x_{l-1}^{(k)})$ for all $k$, using (53).
        Compute (and store) $\Theta_l^-(x_1^{(k)},\dots,x_{l-1}^{(k)},\xi^{(k)})$ for all $k$, using (67).
        Solve the local regression problem for (64) and the local basis $\Psi_l^- \otimes \Phi \otimes \Psi_l^+$,
          using (65). The solution yields the component tensor $u_l \in \mathbb{R}^{r_{l-1}\times m_l \times r_l}$.
    end
end
```

5. Numerical examples

In this section, we extend the numerical examples from Richter et al. (2021) by incorporating the explicit robust loss defined in (29) and comparing it to the other losses from Section 2. As in Richter et al. (2021), we contrast tensor trains with neural networks for function approximation. We refer to Appendix B for implementation details and the definition of the error metrics (RMSE, relative error and PDE loss). Here, we only highlight the fact that RMSE and relative error quantify the accuracy of the approximation in terms of function values, whereas the PDE loss is also sensitive to discrepancies in the derivatives. The code can be found at `https://github.com/lorenzrichter/PDE-backward-solver`. For

Algorithm 4: Approximating the solution to the PDE (5)

```
Input  : Initial parametric choice for the functions $\widehat{V}_n$, for $n \in \{0, \dots, N-1\}$, e.g.
         tensor trains with specified ranks and polynomial degrees.
Output: Approximation of $V(\cdot, t_n) \approx \widehat{V}_n$ along the trajectories for
         $n \in \{0, \dots, N-1\}$.
Simulate $K$ samples of the discretized SDE (27).
Choose $\widehat{V}_N = g$.
for $n = N-1$ to 0 do
    Choose one of the losses (29), (30) or (31).
    Minimize this quantity (explicitly or by iterative schemes, see Algorithms 1-3).
    Set $\widehat{V}_n$ to be the minimizer.
end
```

convenience, we summarize an overview of the general method in Algorithm 4, as already shown in Richter et al. (2021).

5.1 Hamilton-Jacobi-Bellman equation

Associated to (stochastic) optimal control problems, the Hamilton-Jacobi-Bellman (HJB) equation is a PDE for the value function, representing the minimal cost-to-go from which the optimal control policy can be deduced. As suggested by E et al. (2017), we consider the specific form

$$(\partial_t + \Delta) V(x,t) - |\nabla V(x,t)|^2 = 0, \tag{69a}$$

$$V(x,T) = g(x), \tag{69b}$$

with $g(x) = \log\left(\frac{1}{2} + \frac{1}{2}|x|^2\right)$, leading to

$$b = \mathbf{0}, \quad \sigma = \sqrt{2}\,\mathrm{Id}_{d\times d}, \quad h(x,s,y,z) = -\frac{1}{2}|z|^2, \tag{70}$$

using the notation from Section 1.1. A reference solution can be obtained from

$$V(x,t) = -\log \mathbb{E}\left[e^{-g(x+\sqrt{T-t}\sigma\xi)}\right], \tag{71}$$

where $\xi \sim \mathcal{N}(\mathbf{0}, \mathrm{Id}_{d\times d})$ is a normally distributed random variable, see e.g. Appendix D.1 in Richter et al. (2021) for a derivation of this formula.

In our experiments we set $d = 100$, $T = 1$, $\Delta t = 0.01$, $x_0 = (0, \dots, 0)^\top$ and $K = 2000$. Throughout, we complete 100 independent runs of Algorithm 4, each with different realizations of the Brownian increments and with different randomly initialized tensor trains and neural networks. For the tensor train model we first compare different polynomial degrees as well as ranks and observe that choosing the polynomial degree to be 0 and the rank to be 1 yields the best results, in terms of relative error, RMSE and PDE loss. The neural network settings are summarized in Appendix C in Richter et al. (2021). We display the means and standard deviations of the runs in Figure 7 (if no error bars are visible, this means that the standard deviations of the results are very small in comparison to the resolution). The approximations based on tensor trains are obtained significantly faster than those using neural networks, while at the same time being substantially more accurate. Comparing the different losses from Section 2 we see that the BSDE losses lead to a notable increase in performance at the cost of a higher computational budget. The explicit and implicit versions of the BSDE loss perform similarly, in this case with small advantages for the implicit version.

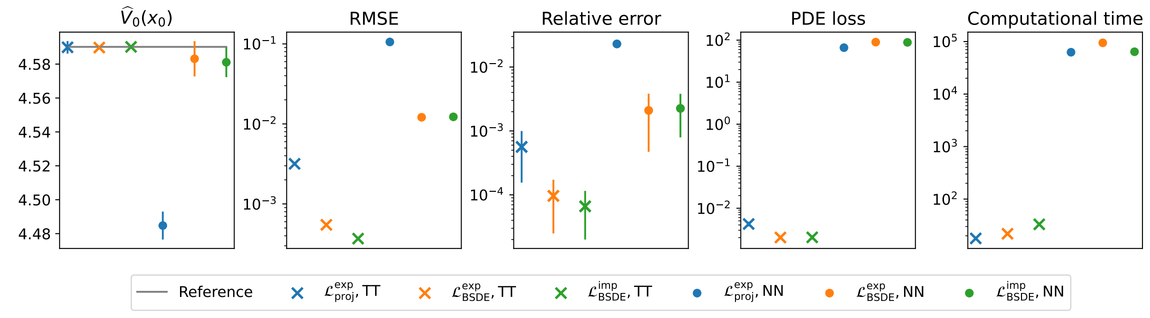


Figure 7: We compare different loss functions for a 100-dimensional HJB example, either relying on tensor trains or on neural networks. For computational details we refer to Appendix B.

To confirm these results, we plot evaluations of the (approximated) value function along two realizations of the forward process in Figures 8 and 9, for dimensions $d = 10$ and $d = 100$, respectively, comparing the tensor train and neural network approximations, both relying on the implicit BSDE loss. We observe that especially in high dimensions the tensor train approximation agrees more with the reference solution than the neural network approximation. Finally, we plot the mean relative error as a function of time in the left-hand panel of Figure 10, which again shows that the tensor train model outperforms the neural network approach.

As observed in Richter et al. (2021), the model simplicity of the tensor train approach is rather surprising (polynomial degree 0, i.e. constant ansatz functions), and therefore the dependence of the polynomial degree for different problem dimensions was studied in more detail. In the right-hand panel of Figure 10 we display relative errors achieved with the tensor train approach using the implicit BSDE loss for varying dimensions and different polynomial degrees. Interestingly, we can see that the problem appears to become easier with growing dimensions and that only for smaller dimensions large polynomial degrees are beneficial. We hypothesize that the observed effect can be attributed to a *blessing of dimensionality*, a phenomenon known from the theory of interacting particle systems ("propagation of chaos", see Sznitman (1991)), where in various scenarios, the joint distribution of a large number of particles tends to approximately factorize as the number of particles increases (that is, as the dimensionality of the joint state space grows large). Similar effects were also reported in Bayer et al. (2023), see Figure 3 and Khoromskij (2012), see Section 1.3, but a theoretical understanding is still lacking. It is plausible that approximate

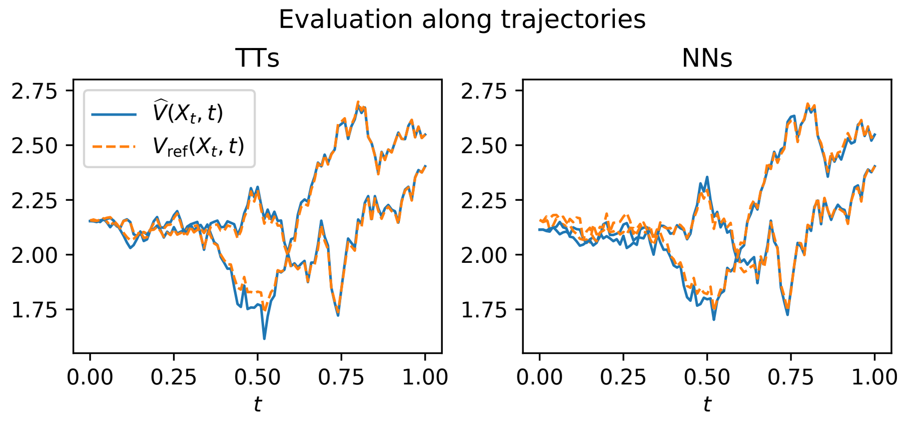


Figure 8: Reference solutions compared with tensor train and neural network approximations using the implicit BSDE loss along two trajectories in $d = 10$.

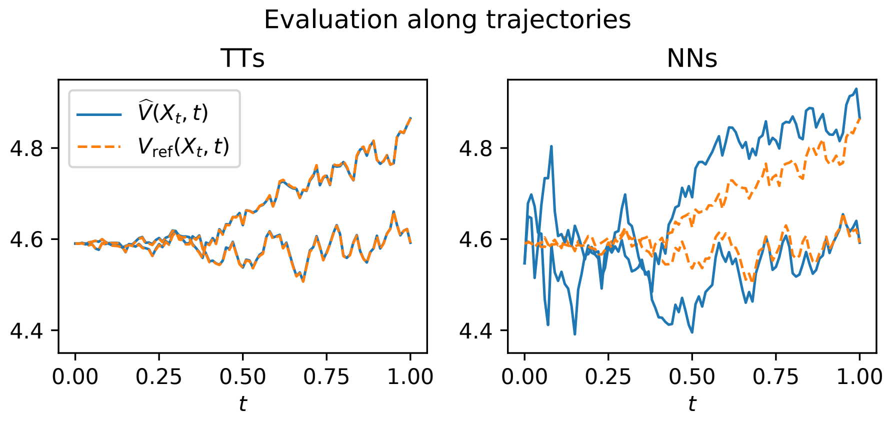


Figure 9: Reference solutions compared with tensor train and neural network approximations using the implicit BSDE loss along two trajectories in $d = 100$.

factorizations are relevant for high-dimensional PDEs and that tensor methods are useful (i) to detect those factorizations and (ii) to exploit them, while the black-box nature of the neural networks does not reveal such properties.

5.2 HJB with double well dynamics

Let us continue with another HJB example, which was proposed by Nüsken and Richter (2021). In this example the forward SDE is nonlinear, which is likely to result in more

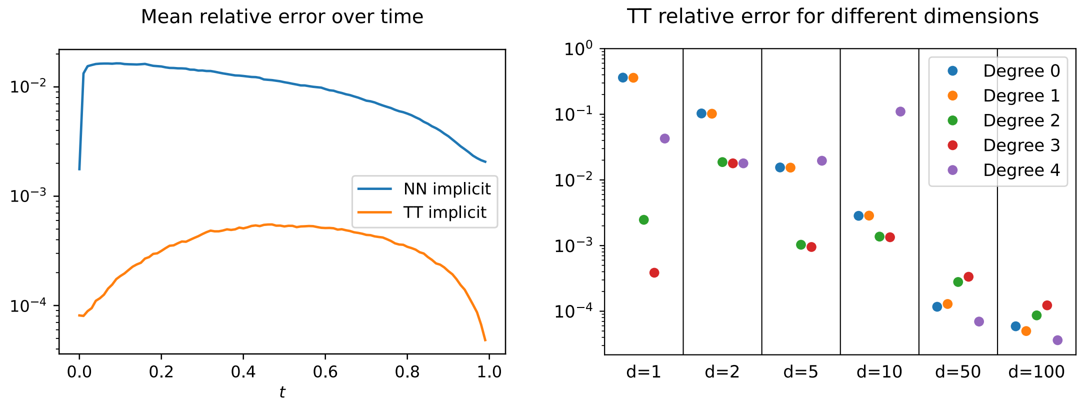


Figure 10: Mean relative error over time for tensor trains and neural networks, relying on the implicit BSDE loss (left). Relative errors achieved with implicit BSDE loss and tensor trains for varying dimensions and different polynomial degrees. The polynomial degree necessary for achieving a good performance seems to decrease with growing state space dimension, which suggests a *blessing of dimensionality* in the setting of Section 5.1 (right).

complicated structures of the PDE solution. We consider

$$(\partial_t + L)\, V(x,t) - \frac{1}{2}|(\sigma^\top \nabla V)(x,t)|^2 = 0, \tag{72a}$$

$$V(x,T) = g(x), \tag{72b}$$

with L as in (6), where now the drift is given as the gradient of a double well potential,

$$b = -\nabla \Psi, \qquad \Psi(x) = \sum_{i,j=1}^{d} C_{ij}(x_i^2 - 1)(x_j^2 - 1), \tag{73}$$

with the matrix $C \in \mathbb{R}^{d\times d}$ assumed to be positive definite. The terminal condition is given by $g(x) = \sum_{i=1}^{d} \nu_i (x_i - 1)^2$ with parameters $\nu_i > 0$.

We first consider a factorized scenario, with a diagonal matrix $C = 0.1\,\mathrm{Id}$, allowing us to estimate the optimal TT-rank to be 2. Further, we set $\sigma = \sqrt{2}\mathrm{Id}$, $T = 0.5$, $d = 50$, $\Delta t = 0.01$, $K = 2000$, $\nu_i = 0.05$ and $x_0 = (-1, \dots, -1) \in \mathbb{R}^d$. For the tensor train approximation we choose the polynomial degree to be 3. Using the fact that the solution factorizes, we can compute a low-dimensional reference solution based on finite differences. We display the results of Algorithm 4 in Figure 11, again comparing tensor trains with neural networks as well as the three main losses from Section 2. As before, the tensor trains are both faster and more accurate than neural networks. We further observe that the explicit BSDE loss yields slightly better results than its implicit counterpart, while being almost one order of magnitude faster.

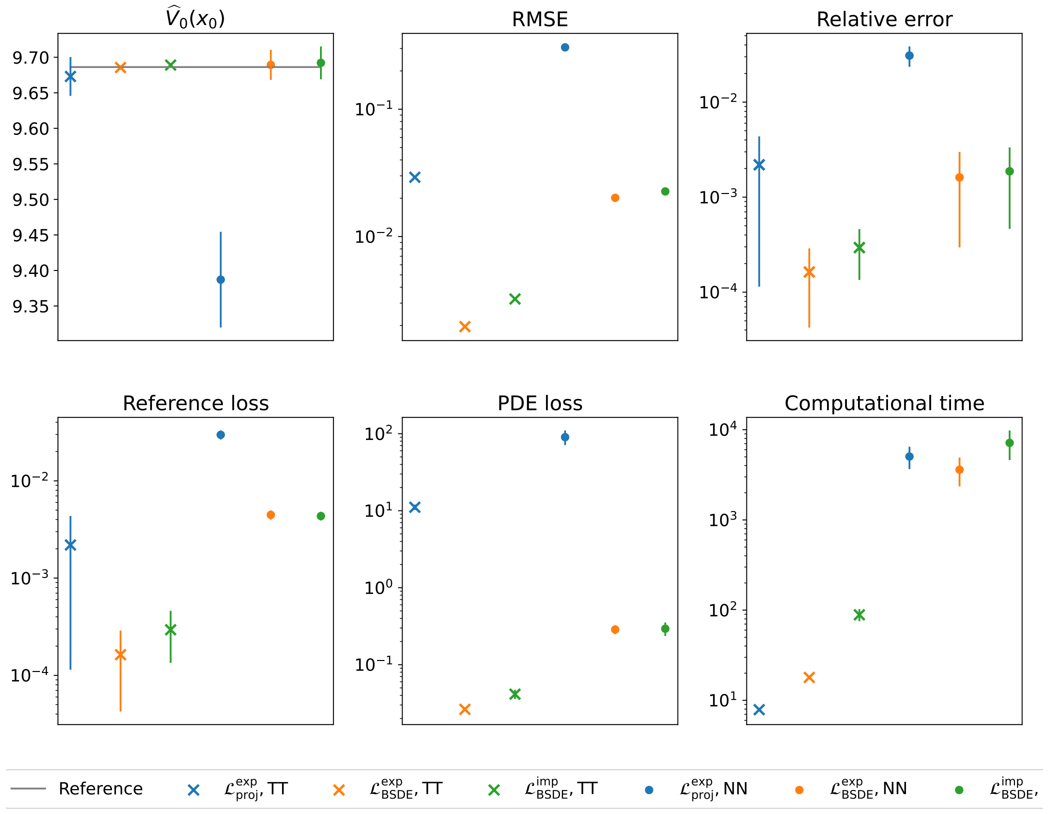


Figure 11: We compare different loss functions for a 50 dimensional HJB example that relates to a nonlinear SDE with the drift given by the negative gradient of a multidimensional extension of a double well potential, either relying on tensor trains or neural networks. For computational details of the evaluation we refer to Appendix B.

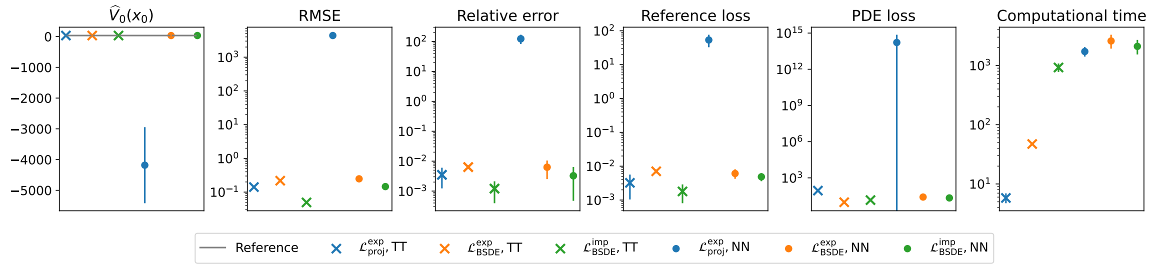
Double well with interacting dimensions

We continue by introducing nondiagonal elements in the matrix C, expecting that higher TT-ranks will be needed to cope with the coupled nature of the problem. We set $d = 20$, $T = 0.3$, $\nu_i = 0.5$ and $C = \mathrm{Id} + \xi_{ij}$, where $\xi_{ij} \sim \mathcal{N}(0, 0.01)$ are sampled once at the beginning of the experiment – all other constants are kept the same as before. We now compute a reference solution by using a Monte Carlo estimate for

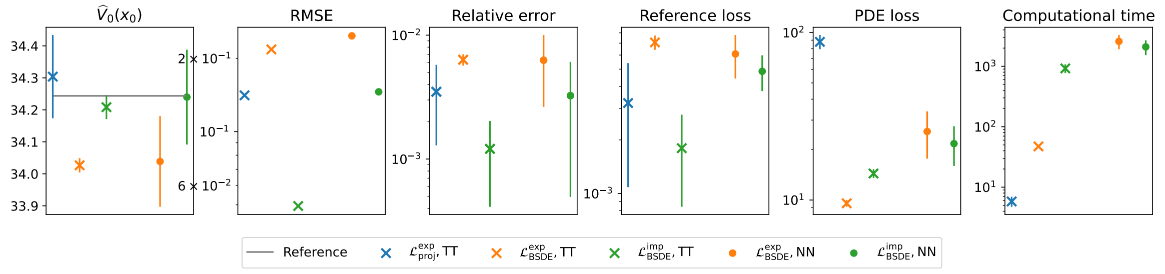
$$V(x,t) = -\log \mathbb{E}\left[e^{-g(X_T)}\middle| X_t = x\right] \tag{74}$$

based on 10^7 samples, obtaining $V(x_0, 0) \approx 34.2687$. For the explicit losses we do not specify the tensor train ranks and instead let the rank-adaptive solver find them. These ranks are mostly between 4 and 6. For the implicit losses the ranks are growing within the iterations if we do not cap them. Motivated by the results for the explicit case we cap the ranks at $r_i \leq 6$. We choose polynomial degree 7 and obtain the results displayed in Figure 12.

In this example, the tensor trains are overall competitive with the neural networks in terms of accuracy, whilst offering shorter computing times. Comparing the results for RMSE and PDE loss reveals an interesting phenomenon: For the regression-based tensor train approaches, losses that explicitly incorporate gradient information ($\mathcal{L}_{\mathrm{BSDE}}^{\mathrm{exp}}$ and $\mathcal{L}_{\mathrm{BSDE}}^{\mathrm{imp}}$) perform better in terms of PDE loss, but worse in terms of RMSE. This observation is in line with Remark 4, and this trade-off might inform the choice of method depending on the application: In optimal control settings, for example, it is often the case ∇V is directly related to the optimal control, and hence of primary interest (rather than V, which might turn out secondary).



(a)



(b)

Figure 12: We compare different loss functions for a 20 dimensional HJB example that relates to a nonlinear SDE with the drift given by the negative gradient of a multidimensional extension of a double well potential with interacting dimensions, either relying on tensor trains or neural networks. Subfigure (a) shows that the implicit BSDE loss does not yield satisfactory results in combination with neural networks. To allow for a more detailed comparison between the remaining approaches, these results are discarded in subfigure (b). For computational details of the evaluation we refer to Appendix B.

5.3 Cox–Ingersoll–Ross model

Finally, we move on to an example from financial mathematics. As proposed by Jiang and Li (2021), we consider the bond price in a multidimensional Cox-Ingersoll-Ross model (CIR), see also Hyndman (2007) and Alfonsi et al. (2015). The PDE is of the form

$$\partial_t V + \frac{1}{2}\sum_{i,j=1}^{d} \sqrt{x_i x_j}\gamma_i\gamma_j \partial_{x_i}\partial_{x_j} V + \sum_{i=1}^{d} a_i(b_i - x_i)\partial_{x_i} V - \left(\max_{1\le i\le d} x_i\right) V = 0, \tag{75}$$

where notationally we omit the space and time dependence of $V = V(x, t)$ for brevity. Here, $a_i, b_i, \gamma_i \in [0, 1]$ are uniformly sampled at the beginning of the experiment and the terminal condition is set to $V(T, x) = 1$. We set the dimension to $d = 100$ and for the tensor trains use polynomial degree 3 and rank 1. Since no reference solution is available we restrict ourselves to the PDE loss as a measure of accuracy. We display the results in Figure 13, showing clear advantages of tensor trains over neural networks as well as the BSDE losses over the projection loss, noting that the explicit BSDE loss is much faster than the implicit one.

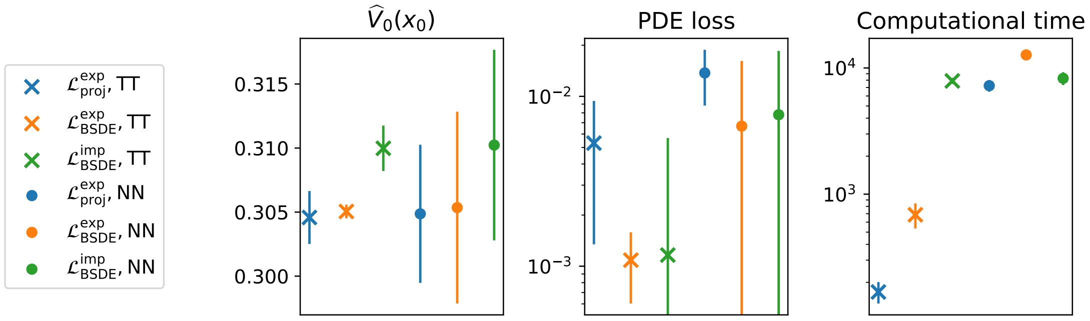


Figure 13: We compare different loss functions for a 100 dimensional CIR example, either relying on tensor trains or neural networks. For computational details of the evaluation we refer to Appendix B.

6. Conclusions and outlook

In this paper, building upon the work of Richter et al. (2021), we have demonstrated the efficacy of tensor trains as a compelling approximation framework for parabolic PDEs. Through a reformulation of the problem in terms of backward stochastic differential equations (BSDEs), we leverage algorithms that draw on backward-in-time iterations to efficiently solve for the PDE along simulated paths of a stochastic process.

Placing emphasis on continuous-time formulations, we have discussed three different loss functionals, one of which is combines statistical robustness with efficient and fast algorithmic computations that has so far attracted very little interest from the community. Future work might consider applying tensor trains to elliptic PDEs and PDEs on bounded domains, and exploring the use of tensor trains with different algorithms such as variational formulations or residual minimization in the spirit of PINNs (Karniadakis et al., 2021).

Acknowledgments

This research has been partially funded by Deutsche Forschungsgemeinschaft (DFG) through the grant CRC 1114 "Scaling Cascades in Complex Systems" (projects A02 and A05, project number 235221301) and the Research Training Group "Differential Equation- and Data-driven Models in Life Sciences and Fluid Dynamics: An Interdisciplinary Research Training Group (DAEDALUS)" (GRK 2433).

Appendix A. Proofs

Proof of Theorem 11 (and Remark 12). Let us define the shorthand

$$h_s^{\varphi,(k)} := h(X_s^{(k)}, s, Y_s^{\varphi,(k)}, Z_s^{\varphi,(k)}) = h(X_s^{(k)}, s, \varphi(X_s^{(k)}, s), \sigma^\top \nabla \varphi(X_s^{(k)}, s)). \tag{76}$$

For $\mathcal{L}_{\mathrm{BSDE}}^{(K)}$ in (42) we compute

$$\begin{aligned}
&\frac{\mathrm{d}}{\mathrm{d}\varepsilon}\Big|_{\varepsilon=0} \mathcal{L}_{\mathrm{BSDE}}^{(K)}(\varphi + \varepsilon\psi) = \\
&\quad \frac{2}{K}\sum_{k=1}^K \left(\varphi(X_{t_n}^{(k)}, t_n) - V(X_{t_{n+1}}^{(k)}, t_{n+1}) - \int_{t_n}^{t_{n+1}} h_s^{\varphi,(k)} \mathrm{d}s + \int_{t_n}^{t_{n+1}} (\sigma^\top \nabla\varphi)(X_s, s)\cdot \mathrm{d}W_s \right) \\
&\qquad \Bigg(\psi(X_{t_n}^{(k)}, t_n) - \int_{t_n}^{t_{n+1}} (\sigma^\top \nabla \psi)(X_s, s) \mathrm{d}s \\
&\qquad\quad - \int_{t_n}^{t_{n+1}} \left(\partial_y h_s^{\varphi,(k)} \psi(X_s^{(k)}, s) + \nabla_z h_s^{\varphi,(k)} \cdot (\sigma^\top \nabla \psi)(X_s^{(k)}, s) \right) \mathrm{d}s \Bigg).
\end{aligned} \tag{77}$$

Setting $\varphi = V$, we see that

$$V(X_{t_n}^{(k)}, t_n) - V(X_{t_{n+1}}^{(k)}, t_{n+1}) - \int_{t_n}^{t_{n+1}} h_s^{\varphi,(k)} \mathrm{d}s + \int_{t_n}^{t_{n+1}} (\sigma^\top \nabla \varphi)(X_s, s) \cdot \mathrm{d}W_s = 0,$$

almost surely, for all $k = 1, \ldots, K$, due to (20), implying (42).

To justify Remark 12, we compute

$$\begin{aligned}
&\frac{\mathrm{d}}{\mathrm{d}\varepsilon}\Big|_{\varepsilon=0} \mathcal{L}_{\mathrm{proj}}^{(K)}(\varphi + \varepsilon\psi) = \\
&\qquad \frac{2}{K}\sum_{k=1}^K \left(\varphi(X_{t_n}^{(k)}, t_n) - V(X_{t_{n+1}}^{(k)}, t_{n+1}) - \int_{t_n}^{t_{n+1}} h_s^{V,(k)} \mathrm{d}s \right) \psi(X_{t_n}^{(k)}, t_n).
\end{aligned} \tag{78}$$

Setting $\varphi = V$ yields

$$\frac{\delta}{\delta\varphi}\Big|_{\varphi=V} \mathcal{L}_{\mathrm{proj}}^{(K)}(\varphi;\psi) = -\frac{2}{K}\sum_{k=1}^K \left(\int_{t_n}^{t_{n+1}} (\sigma^\top \nabla V)(X_s^{(k)}, s)) \cdot \mathrm{d}W_s^{(k)} \right) \psi(X_{t_n}^{(k)}, t_n). \tag{79}$$

According to the law of total variance, the variance of (79) is given by

$$\frac{4}{K^2}\mathbb{E}\left[\mathrm{Var}\left(\sum_{k=1}^K \left(\int_{t_n}^{t_{n+1}} (\sigma^\top \nabla V)(X_s^{(k)}, s)) \cdot \mathrm{d}W_s^{(k)} \right) \psi(X_{t_n}^{(k)}, t_n) \right) \Bigg| \mathcal{F}_n \right] \tag{80a}$$

$$+\frac{4}{K^2}\,\mathrm{Var}\left(\mathbb{E}\left[\sum_{k=1}^K \left(\int_{t_n}^{t_{n+1}} (\sigma^\top \nabla V)(X_s^{(k)}, s)) \cdot \mathrm{d}W_s^{(k)} \right) \psi(X_{t_n}^{(k)}, t_n) \right] \Bigg| \mathcal{F}_n \right). \tag{80b}$$

Since $\psi(X_{t_n}^{(k)}, t_n)$ is $\mathcal{F}_n$-measurable, the contribution in (80b) vanishes according to the martingale property of the Itô stochastic integral. Similarly, the term in (80a) equals (43) by Itô's isometry. ■

Appendix B. Implementation details

In order to evaluate our approximations for solving the PDE in (5) we depend on reference values V_{ref} of V at $(x,t) = (0, x_0)$. This allows us to calculate the *relative error* of $\widehat{V}_0$ using

$$\mathcal{E}_{\mathrm{rel}} = \left| \frac{\widehat{V}_0(x_0) - V_{\mathrm{ref}}(x_0, 0)}{V_{\mathrm{ref}}(x_0, 0)} \right|. \tag{81}$$

Following the computation of different approximations $(\widehat{V}_0^{(m)})_{1 \le m \le M}$ in M runs, we determine the *root mean squared error (RMSE)* by

$$\mathcal{E}_{\mathrm{RMSE}} = \sqrt{\frac{1}{M} \sum_{m=1}^{M} \Big(\widehat{V}_0^{(m)}(x_0) - V_{\mathrm{ref}}(x_0, 0) \Big)^2}. \tag{82}$$

Additionally, we introduce two error metrics that are (at least approximately) zero if and only if the PDE is satisfied along the samples generated by the discrete forward SDE in (27).

First, we define the *PDE loss* (inspired by Raissi et al. (2019)) as

$$\mathcal{E}_{\mathrm{PDE}} = \frac{1}{KN} \sum_{n=1}^{N} \sum_{k=1}^{K} \Big((\partial_t + L) V(\widehat{X}_n^{(k)}, t_n) + h(\widehat{X}_n^{(k)}, t_n, V(\widehat{X}_n^{(k)}, t_n), (\sigma^\top \nabla V)(\widehat{X}_n^{(k)}, t_n)) \Big)^2,$$

where $\widehat{X}_n^{(k)}$ are realizations of (27), the time derivative is approximated with finite differences and the space derivatives are computed analytically (or using automatic differentiation). We exclude the initial time step ($n = 0$) due to the ill-defined regression problem ($X_0^k = x_0$ has the same value for all k) in the explicit and implicit tensor train schemes, but still achieve a good approximation as the regularization term provides a minimum norm solution with the correct point value $V(x_0, 0)$.

Second, we establish the *relative reference loss* as

$$\mathcal{E}_{\mathrm{ref}} = \frac{1}{K(N+1)} \sum_{n=0}^{N} \sum_{k=1}^{K} \left| \frac{V(\widehat{X}_n^{(k)}, t_n) - V_{\mathrm{ref}}(\widehat{X}_n^{(k)}, t_n)}{V_{\mathrm{ref}}(\widehat{X}_n^{(k)}, t_n)} \right|, \tag{83}$$

whenever a reference solution for all x and t is available.

All computation times in the reported tables are measured in seconds.

Our experiments have been performed on a desktop computer containing an AMD Ryzen Threadripper 2990 WX 32x 3.00 GHz mainboard and an NVIDIA Titan RTX GPU, where we note that only the NN optimizations were run on this GPU, since our TT framework does not include GPU support. It is expected that running the TT approximations on a GPU will improve time performances in the future, see Abdelfattah et al. (2016).

All our code is available under `https://github.com/lorenzrichter/PDE-backward-solver`.

For further details, such as tensor train settings and neural network architectures, we refer to Appendix D in Richter et al. (2021).

References

Ahmad Abdelfattah, Marc Baboulin, Veselin Dobrev, Jack Dongarra, Christopher Earl, Joel Falcou, Azzam Haidar, Ian Karlin, Tz Kolev, Ian Masliah, et al. High-performance tensor contractions for GPUs. *Procedia Computer Science*, 80:108–118, 2016.

Aurélien Alfonsi et al. *Affine diffusions and related processes: simulation, theory and applications*, volume 6. Springer, 2015.

Markus Bachmayr, Reinhold Schneider, and André Uschmajew. Tensor networks and hierarchical tensors for the solution of high-dimensional partial differential equations. *Foundations of Computational Mathematics*, 16:1423–1472, 2016.

Christian Bayer, Martin Eigel, Leon Sallandt, and Philipp Trunschke. Pricing high-dimensional Bermudan options with hierarchical tensor formats. *SIAM Journal on Financial Mathematics*, 14(2):383–406, 2023.

Christian Beck, Arnulf Jentzen, et al. Machine learning approximation algorithms for high-dimensional fully nonlinear partial differential equations and second-order backward stochastic differential equations. *Journal of Nonlinear Science*, 29(4):1563–1619, 2019.

Christian Beck, Martin Hutzenthaler, Arnulf Jentzen, and Benno Kuckuck. An overview on deep learning-based approximation methods for partial differential equations. *arXiv preprint arXiv:2012.12348*, 2020.

Christian Beck, Sebastian Becker, Patrick Cheridito, Arnulf Jentzen, and Ariel Neufeld. Deep splitting method for parabolic PDEs. *SIAM Journal on Scientific Computing*, 43 (5):A3135–A3154, 2021.

Jean-Michel Bismut. Conjugate convex functions in optimal stochastic control. *Journal of Mathematical Analysis and Applications*, 44(2):384–404, 1973.

Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *Siam Review*, 60(2):223–311, 2018.

Bruno Bouchard and Nizar Touzi. Discrete-time approximation and Monte-Carlo simulation of backward stochastic differential equations. *Stochastic Processes and their applications*, 111(2):175–206, 2004.

Bruno Bouchard, Romuald Elie, and Nizar Touzi. Discrete-time approximation of BSDEs and probabilistic schemes for fully nonlinear PDEs. *Advanced financial modelling*, 8: 91–124, 2009.

Jean-François Chassagneux. Linear multistep schemes for BSDEs. *SIAM Journal on Numerical Analysis*, 52(6):2815–2836, 2014.

Jean-François Chassagneux and Dan Crisan. Runge–Kutta schemes for backward stochastic differential equations. *The Annals of Applied Probability*, 24(2):679–720, 2014.

Jean-François Chassagneux and Adrien Richou. Numerical stability analysis of the Euler scheme for BSDEs. *SIAM Journal on Numerical Analysis*, 53(2):1172–1193, 2015.

Yidong Chen and Zhonghua Lu. Tensor decomposition and high-performance computing for solving high-dimensional stochastic control system numerically. *Journal of Systems Science and Complexity*, pages 1–14, 2021.

Jared Chessari, Reiichiro Kawai, Yuji Shinozaki, and Toshihiro Yamada. Numerical methods for backward stochastic differential equations: A survey. *Probability Surveys*, 20:486–567, 2023.

Alec Dektor, Abram Rodgers, and Daniele Venturi. Rank-adaptive tensor methods for high-dimensional nonlinear PDEs. *Journal of Scientific Computing*, 88(2):36, 2021.

Sergey Dolgov, Boris N Khoromskij, Alexander Litvinenko, and Hermann G Matthies. Polynomial chaos expansion of random coefficients and the solution of stochastic partial differential equations in the tensor train format. *SIAM/ASA Journal on Uncertainty Quantification*, 3(1):1109–1135, 2015.

Sergey Dolgov, Dante Kalise, and Karl K Kunisch. Tensor decomposition methods for high-dimensional Hamilton–Jacobi–Bellman equations. *SIAM Journal on Scientific Computing*, 43(3):A1625–A1650, 2021.

Weinan E, Jiequn Han, and Arnulf Jentzen. Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. *Communications in Mathematics and Statistics*, 5(4):349–380, 2017.

Weinan E, Jiequn Han, and Arnulf Jentzen. Algorithms for solving high dimensional PDEs: from nonlinear Monte Carlo to machine learning. *Nonlinearity*, 35(1):278, 2021.

Martin Eigel, Max Pfeffer, and Reinhold Schneider. Adaptive stochastic Galerkin FEM with hierarchical tensor representations. *Numerische Mathematik*, 136(3):765–803, 2017.

Nicole El Karoui, Shige Peng, and Marie Claire Quenez. Backward stochastic differential equations in finance. *Mathematical finance*, 7(1):1–71, 1997.

Lawrence C Evans. *Partial differential equations*, volume 19. American Mathematical Society, 2022.

Konstantin Fackeldey, Mathias Oster, Leon Sallandt, and Reinhold Schneider. Approximative policy iteration for exit time feedback control problems driven by stochastic differential equations using tensor train format. *Multiscale Modeling & Simulation*, 20 (1):379–403, 2022.

Arash Fahim, Nizar Touzi, and Xavier Warin. A probabilistic numerical method for fully nonlinear parabolic PDEs. *The Annals of Applied Probability*, 21(4):1322–1364, 2011.

Wendell H Fleming and Raymond W Rishel. *Deterministic and stochastic optimal control*, volume 1. Springer Science & Business Media, 2012.

Maximilien Germain, Huyen Pham, and Xavier Warin. Approximation error analysis of some deep backward schemes for nonlinear PDEs. *SIAM Journal on Scientific Computing*, 44(1):A28–A56, 2022.

Emmanuel Gobet. *Monte-Carlo methods and stochastic processes: from linear to non-linear.* CRC Press, 2016.

Emmanuel Gobet and Céline Labart. Error expansion for the discretization of backward stochastic differential equations. *Stochastic processes and their applications*, 117(7):803–829, 2007.

Emmanuel Gobet, Jean-Philippe Lemor, Xavier Warin, et al. A regression-based Monte Carlo method to solve backward stochastic differential equations. *The Annals of Applied Probability*, 15(3):2172–2202, 2005.

Alex Gorodetsky, Sertac Karaman, and Youssef Marzouk. High-dimensional stochastic optimal control using continuous tensor decompositions. *The International Journal of Robotics Research*, 37(2-3):340–377, 2018.

Michael Götte, Reinhold Schneider, and Philipp Trunschke. A block-sparse tensor train format for sample-efficient high-dimensional polynomial regression. *Frontiers in Applied Mathematics and Statistics*, 7:702486, 2021.

Andreas Griewank and Andrea Walther. *Evaluating derivatives: principles and techniques of algorithmic differentiation.* SIAM, 2008.

Wolfgang Hackbusch. Numerical tensor calculus. *Acta numerica*, 23:651–742, 2014.

Wolfgang Hackbusch and Stefan Kühn. A new scheme for the tensor representation. *Journal of Fourier analysis and applications*, 15(5):706–722, 2009.

Wolfgang Hackbusch and Reinhold Schneider. Tensor spaces and hierarchical tensor representations. *Extraction of quantifiable information from complex systems*, pages 237–261, 2014.

Jiequn Han and Jihao Long. Convergence of the deep BSDE method for coupled FBSDEs. *Probability, Uncertainty and Quantitative Risk*, 5(1):1–33, 2020.

Jiequn Han, Arnulf Jentzen, and Weinan E. Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences*, 115(34): 8505–8510, 2018.

Carsten Hartmann, Omar Kebiri, Lara Neureither, and Lorenz Richter. Variational approach to rare event simulation using least-squares regression. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 29(6):063107, 2019.

Sebastian Holtz, Thorsten Rohwedder, and Reinhold Schneider. The alternating linear scheme for tensor optimization in the tensor train format. *SIAM Journal on Scientific Computing*, 34(2):A683–A713, 2012a.

Sebastian Holtz, Thorsten Rohwedder, and Reinhold Schneider. On manifolds of tensors of fixed TT-rank. *Numerische Mathematik*, 120(4):701–731, 2012b.

Matanya B Horowitz, Anil Damle, and Joel W Burdick. Linear Hamilton Jacobi Bellman equations in high dimensions. In *53rd IEEE Conference on Decision and Control*, pages 5880–5887. IEEE, 2014.

Côme Huré, Huyên Pham, and Xavier Warin. Deep backward schemes for high-dimensional nonlinear PDEs. *Mathematics of Computation*, 89(324):1547–1579, 2020.

Cody Blaine Hyndman. Forward-backward SDEs and the CIR model. *Statistics & probability letters*, 77(17):1676–1682, 2007.

Yifan Jiang and Jinfeng Li. Convergence of the deep BSDE method for FBSDEs with non-Lipschitz coefficients. *arXiv preprint arXiv:2101.01869*, 2021.

Claes Johnson. *Numerical solution of partial differential equations by the finite element method*. Courier Corporation, 2012.

George Em Karniadakis, Ioannis G Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, 2021.

Vladimir A Kazeev and Boris N Khoromskij. Low-rank explicit QTT representation of the laplace operator and its inverse. *SIAM journal on matrix analysis and applications*, 33 (3):742–758, 2012.

Boris N Khoromskij. Tensors-structured numerical methods in scientific computing: Survey on recent advances. *Chemometrics and Intelligent Laboratory Systems*, 110(1):1–19, 2012.

Achim Klenke. *Probability theory: a comprehensive course*. Springer Science & Business Media, 2013.

Peter E Kloeden and Eckhard Platen. Stochastic differential equations. In *Numerical Solution of Stochastic Differential Equations*, pages 103–160. Springer, 1992.

Magdalena Kobylanski. Backward stochastic differential equations and partial differential equations with quadratic growth. *the Annals of Probability*, 28(2):558–602, 2000.

Katharina Kormann. A semi-Lagrangian Vlasov solver in tensor train format. *SIAM Journal on Scientific Computing*, 37(4):B613–B632, 2015.

Hiroshi Kunita. *Stochastic flows and jump-diffusions*. Springer, 2019.

Francis A Longstaff and Eduardo S Schwartz. Valuing American options by simulation: a simple least-squares approach. *The review of financial studies*, 14(1):113–147, 2001.

Michael Lubasch, Pierre Moinier, and Dieter Jaksch. Multigrid renormalization. *Journal of Computational Physics*, 372:587–602, 2018.

Jin Ma, Philip Protter, Jaime San Martín, and Soledad Torres. Numerical method for backward stochastic differential equations. *The Annals of Applied Probability*, 12(1): 302–316, 2002.

Nikolas Nüsken and Lorenz Richter. Solving high-dimensional Hamilton–Jacobi–Bellman PDEs using neural networks: perspectives from the theory of controlled diffusions and measures on path space. *Partial Differential Equations and Applications*, 2(4):1–48, 2021.

Nikolas Nüsken and Lorenz Richter. Interpolating between BSDEs and PINNs: Deep learning for elliptic and parabolic boundary value problems. *Journal of Machine Learning*, 2 (1):31–64, 2023. doi: https://doi.org/10.4208/jml.220416.

Bernt Oksendal. *Stochastic differential equations: an introduction with applications.* Springer Science & Business Media, 2013.

Ivan V Oseledets. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33 (5):2295–2317, 2011.

Mathias Oster, Leon Sallandt, and Reinhold Schneider. Approximating the stationary Hamilton-Jacobi-Bellman equation by hierarchical tensor products. *arXiv preprint arXiv:1911.00279*, 2019.

Mathias Oster, Leon Sallandt, and Reinhold Schneider. Approximating optimal feedback controllers of finite horizon control problems using hierarchical tensor formats. *SIAM Journal on Scientific Computing*, 44(3):B746–B770, 2022.

Étienne Pardoux. Backward stochastic differential equations and viscosity solutions of systems of semilinear parabolic and elliptic PDEs of second order. In *Stochastic Analysis and Related Topics VI*, pages 79–127. Springer, 1998.

Etienne Pardoux and Shige Peng. Adapted solution of a backward stochastic differential equation. *Systems & Control Letters*, 14(1):55–61, 1990.

Huyên Pham. *Continuous-time stochastic control and optimization with financial applications*, volume 61. Springer Science & Business Media, 2009.

Huyen Pham, Xavier Warin, and Maximilien Germain. Neural networks-based backward scheme for fully nonlinear PDEs. *SN Partial Differential Equations and Applications*, 2 (1):1–24, 2021.

Maziar Raissi. Forward-backward stochastic neural networks: Deep learning of high-dimensional partial differential equations. *arXiv preprint arXiv:1804.07010*, 2018.

Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

Lorenz Richter. *Solving high-dimensional PDEs, approximation of path space measures and importance sampling of diffusions.* PhD thesis, BTU Cottbus-Senftenberg, 2021.

Lorenz Richter and Julius Berner. Robust SDE-based variational formulations for solving linear PDEs via deep learning. In *International Conference on Machine Learning*, pages 18649–18666. PMLR, 2022.

Lorenz Richter, Leon Sallandt, and Nikolas Nüsken. Solving high-dimensional parabolic PDEs using the tensor train format. In *International Conference on Machine Learning*, pages 8998–9009. PMLR, 2021.

Christian P Robert and George Casella. *Monte Carlo statistical methods*, volume 2. Springer, 2004.

Leon Jasper Sallandt. *Computing high-dimensional value functions of optimal feedback control problems using the Tensor-train format.* PhD thesis, TU Berlin, 2022.

Winfried Sickel and Tino Ullrich. Tensor products of Sobolev-Besov spaces and applications to approximation from the hyperbolic cross. *Journal of Approximation Theory*, 161(2): 748–786, 2009.

Abul Hasan Siddiqi and Sudarsan Nanda. *Functional analysis with applications.* Springer, 1986.

Elis Stefansson and Yoke Peng Leong. Sequential alternating least squares for solving high dimensional linear Hamilton-Jacobi-Bellman equation. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3757–3764. IEEE, 2016.

Szilárd Szalay, Max Pfeffer, Valentin Murg, Gergely Barcza, Frank Verstraete, Reinhold Schneider, and Örs Legeza. Tensor product methods and entanglement optimization for ab initio quantum chemistry. *International j. of quantum chemistry*, 115(19):1342–1391, 2015. ISSN 1097-461x. doi: 10.1002/qua.24898.

Alain-Sol Sznitman. Topics in propagation of chaos. In *Ecole d'été de probabilités de Saint-Flour XIX—1989*, pages 165–251. Springer, 1991.

Nizar Touzi. *Optimal stochastic control, stochastic target problems, and backward SDE*, volume 29. Springer Science & Business Media, 2012.

Philipp Trunschke. Convergence bounds for nonlinear least squares and applications to tensor recovery. *arXiv preprint arXiv:2108.05237*, 2021.

Larry Wasserman. *All of statistics: a concise course in statistical inference*, volume 26. Springer, 2004.

Alexander Sebastian Johannes Wolf. *Low rank tensor decompositions for high dimensional data approximation, recovery and prediction.* Doctoral thesis, Technische Universität Berlin, Berlin, 2019. URL `http://dx.doi.org/10.14279/depositonce-8109`.

Jianfeng Zhang. A numerical scheme for BSDEs. *The annals of applied probability*, 14(1): 459–488, 2004.

Jianfeng Zhang. *Backward stochastic differential equations.* Springer, 2017.

Mo Zhou, Jiequn Han, and Jianfeng Lu. Actor-critic method for high dimensional static Hamilton–Jacobi–Bellman partial differential equations based on neural networks. *SIAM Journal on Scientific Computing*, 43(6):A4043–A4066, 2021.