Stoch PDE: Anal Comp (2022) 10:1050–1081
https://doi.org/10.1007/s40072-022-00260-y

An application of the splitting-up method for the computation of a neural network representation for the solution for the filtering equations

Dan Crisan[1] · **Alexander Lobbe**[1] · **Salvador Ortiz-Latorre**[2]

This article is dedicated to István Gyöngy on the occasion of his 70th birthday.

Received: 16 November 2021 / Revised: 23 March 2022 / Accepted: 9 May 2022 /
Published online: 9 June 2022

Abstract
The filtering equations govern the evolution of the conditional distribution of a signal process given partial, and possibly noisy, observations arriving sequentially in time. Their numerical approximation plays a central role in many real-life applications, including numerical weather prediction [Llopis et al. (SIAM J Sci Comput 40(3):A1544–A1565, 2018), Galanis et al. (Geophysicae 24(10): 2451–2460, 2006)], finance [Brigo and Hanzon (Insurance Math Econom 22(1):53–64, 1998), Date and Ponomareva (IMA J Manag Math 22(3): 195–211, 2011), Crisan and Rozovskii (The Oxford handbook of nonlinear filtering, 2011)] and engineering [Myötyri et al. (Reliability Eng Syst Saf 91(2):200–208, 2005)]. One of the classical approaches to approximate the solution of the filtering equations is to use a PDE inspired method, called the splitting-up method, initiated by Gyongy, Krylov, LeGland, among other contributors, see e.g., Gyöngy and Krylov (Stochastic inequalities and applications, *Progr. Probab.* 56:301–321, 2003), Le Gland(Stochastic partial differential equations and their applications (Charlotte, NC, 1991), *Lect. Notes Control Inf. Sci.* 176:177–187, 1992). This method, and other PDE based approaches, have particular applicability for solving low-dimensional problems. In this work we combine this method with a neural network representation inspired by [Han et al. (Proc Natl acad Sci 115(34):8505–8510, 2018)]. The new methodology is used to produce an approximation of the unnor-

✉ Dan Crisan
d.crisan@imperial.ac.uk

Alexander Lobbe
alex.lobbe@imperial.ac.uk

Salvador Ortiz-Latorre
salvadoo@math.uio.no

[1] Department of Mathematics, Imperial College London, SW7 2AZ London, UK

[2] Department of Mathematics, University of Oslo, Postboks 1053, 0316 Blindern, Oslo, Norway

malised conditional distribution of the signal process. We further develop a recursive normalisation procedure to recover the normalised conditional distribution of the signal process. The new scheme can be iterated over multiple time steps whilst keeping its asymptotic unbiasedness property intact. We test the neural network approximations with numerical approximation results for the Kalman and Benes filter.

Keywords Stochastic Filtering · Deep Learning · Numerical Approximation of Stochastic PDEs

Mathematics Subject Classification 60H15 · 93E11 · 65C05 · 65C30

1 Introduction

This paper is concerned with the numerical approximation of the solution of the stochastic filtering equations. In addition to its theoretical significance in stochastic analysis and control (see, for example, [3, 10] or [6]), stochastic filtering is an important modelling framework for many domains of application, such as numerical weather prediction [14, 34], finance [8, 11] [10, Part IX] and engineering [35]. Hence, there is a high demand for efficient and accurate numerical methods to approximate the solution of the filtering problem, i.e. the solution of the filtering equations. Here, we are presenting a first study in an ongoing effort to combine a machine learning approach, that has risen in prominence within the numerical community over the past years, with the classical PDE based approach to the numerical resolution of the stochastic filtering problem. In particular, we base our algorithm on the SPDE splitting method that was, among others, developed by Istvan Gyongy, Nikolay Krylov and Francois LeGland [18, 32]. The chosen neural network based machine learning approach for the approximation of the involved deterministic PDE is inspired by [22].

Among all contributors, Istvan Gyongy has made the most fundamental contribution to the development of the splitting-up method as applied to the filtering equation and beyond. In the following we give some brief details of his contribution to the topic. The first of Gyongy's works in this direction was published in 2002 [17] where he presented numerical results for the approximation of stochastic PDEs with a particular focus on the the splitting-up method. Soon after, he published the paper [19] with Nikolay Krylov, in the Annals of Probability. In this work, he investigates the convergence rates of the splitting method for various different classes of stochastic PDEs. Furthermore, in the final part of the paper he explicitly treats the application of these results in the context of stochastic filtering. In another work with Krylov in 2003, Gyongy proved convergence rates in Sobolev norm for the splitting-up method. Notably, this result is proved for the general case of time-dependent coefficients of the considered classes of SPDEs and the rates are even shown to be sharp. A short while later, another work of Gyongy, coauthored by Krylov, appeared in the year 2005 [20]. In this innovative paper, Gyongy devised a theoretical method for the splitting-up approximation of parabolic equations by constructing high order splitting-up methods out of low order ones by means of Richardson extrapolation.

The paper is structured as follows: In Sect. 1.1 we introduce the notation in the paper. Thereafter, in Sect. 1.2, we present the stochastic filtering problem at the level of generality appropriate for the purposes of this work. Notably, Proposition 1 presents the well-known Kallianpur-Striebel formula which establishes the distinction between what we call, respectively, the normalised and unnormalised filter. Subsequently, in Sect. 1.3, we discuss the filtering equations and recall the splitting-up method as we will apply it to the stochastic filtering equations. Based on the SPDE for the unnormalised filter, sometimes referred to as Zakai's equation, we apply the splitting method to decompose the SPDE into the deterministic PDE part and a normalisation, or data-assimilation, step. The first step is commonly solved numerically by using Galerkin methods or similar grid-based approximation schemes. This approach is best applied in low-dimensional settings, due to the computational cost introduced by the discretisation. The second step is to construct the (approximate) likelihood based on the observation and to finally normalise the product of the likelihood function and the PDE solution such that it integrates to unity.

Next, in Sect. 2, we analyse the case when the coefficient functions of the differential operator in the deterministic PDE that arises from the splitting-up method has smooth coefficients. The consequence of this assumption is that the operator can be split into a diffusion operator and a zero-order part. An elementary but crucial part of our argument is then given in Lemma 1 which establishes the fact that the diffusion operator arising from the PDE operator with smooth coefficients generates a stochastic diffusion process, which we will later call *auxiliary diffusion*. Another central ingredient in the derivation of our method is the Feynman-Kac formula, given in Theorem 1 for final-value PDEs. As we are presented with an initial-value problem, we will need the Feynman-Kac formula in a form that applies to such kind of PDEs. This is given in Corollary 1. The significance of the Feynman-Kac formula and the auxiliary diffusion derived in Sect. 2 lies in the fact that the solution to the deterministic PDE problem can then be written as a conditional expectation with respect to the law of the auxiliary diffusion given its initial value. We then give two examples of explicit representations of solutions to the particular filtering problems of the Kalman (linear) filter and the Benes filter in terms of the Feynman-Kac representation. In Sect. 2.3 we prove Proposition 2 based on arguments presented in [22] and thus show that the solution of the PDE over a full hypercube-domain is represented by an infinite-dimensional optimisation of an objective function given by the Feynman-Kac formula.

Section 3 is dedicated to the detailed description of our computational method. In Sect. 3.1 we introduce some terminology on deep learning, and specify how a parametrised neural network representation of the solution of the deterministic PDE is approximated through a Monte-Carlo sampling-based minimisation of the objective function given by the Feynman-Kac formula and the minimisation problem derived before. In practise, the infinite-dimensional function space over which we theoretically minimise is parametrised by the neural network parameters to make it computationally tractable. This enables us to use generic methods for the computational optimisation, provided we are able to sample from the auxiliary diffusion process. Thereafter, in Sect. 3.2 we describe the second part of the splitting method where we rely on the Monte-Carlo approximation of the product of the neural network and the likelihood function to obtain the necessary normalisation constant. Subsequently, in Sect. 3.3

we describe the neural network representation and the chosen optimisation algorithm mathematically which results in a full specification of our method in terms of pseudocode. In particular, the algorithm may be iterated over several time steps whilst remaining asymptotically unbiased.

The numerical results obtained for the Kalman and Benes filters are presented in Sect. 4. In our one-dimensional examples, we observed that the method can successfully be iterated over several time-steps. To the best of our knowledge, we present the first numerical results showing that the sampling based neural network representation of the solution to the Fokker-Planck equation may be iterated while remaining accurate with respect to the exact solution of the filtering problem. In fact, the filtering framework is ideally suited for this kind of study, because of its inherently sequential nature. Moreover, we identify the choice of the domain as a crucial factor for the success of our approximation. Due to the normalisation procedure which uses samples from the likelihood, we need to have a good signal-to-noise ratio in order to obtain a large proportion of samples within our considered domain. If this is not the case, the method diverges. Our study of the nonlinear Benes filter shows that the method is able to handle also nonlinear dynamics.

In conclusion, based on the limited testing performed in this study, we believe that the use of neural network based representations in the numerical approximation of the stochastic filtering problem can be a viable alternative to existing numerical methods. Nevertheless, we emphasize the following two important caveats. First, the mathematical analysis of deep learning algorithms such as the one we employed here is not advanced enough to guarantee explicit convergence rates which might be undesirable in certain settings. Secondly, more numerical studies have to be performed to accurately evaluate the capabilities of neural networks in situations of higher practical relevance than the synthetic study we have performed in this work. We plan to investigate this topic further in future work.

1.1 Notation

Throughout this paper, $\mathbb{N}$ denotes the natural numbers without zero and $\mathbb{N}_0$ is the set of natural numbers including zero. The real numbers are denoted by $\mathbb{R}$ and, given $n \in \mathbb{N}$, $\mathbb{R}^n$ is n-dimensional Euclidean space. For $m, n \in \mathbb{N}$ the set of $m \times n$-matrices with real entries is denoted by $\mathbb{R}^{m\times n}$. For a given matrix $M \in \mathbb{R}^{m\times n}$, M' denotes its transpose and $\mathrm{Tr}(M)$ denotes its trace. For $k \in \mathbb{N}_0 \cup \{\infty\}$ and separable normed $\mathbb{R}$-vector spaces A and B we denote by $C^k(A, B)$ the set of k-times continuously differentiable functions from $A \to B$. Moreover, we use the shorthand $C^k(A)$ whenever $A = B$ and always identify $C^0(A, B) = C(A, B)$. Similarly, the spaces of k-times continuously differentiable functions with compact support are denoted by $C_c^k(A, B)$ and the ones of bounded functions with bounded derivatives of all orders by $C_b^k(A, B)$. For an interval $I \subset \mathbb{R}$ and $d \in \mathbb{N}$ we write $C_b^{1,2}(I \times \mathbb{R}^d, \mathbb{R})$ for the set of bounded functions $f : I \times \mathbb{R}^d \to \mathbb{R}$ that are once continuously differentiable with bounded derivative in the first variable and twice continuously differentiable with bounded derivative in the second variable. For a topological space $(\mathcal{T}, \mathfrak{T})$, $\mathcal{B}(\mathcal{T})$ is the Borel sigma-algebra on $\mathcal{T}$. Further, if $(\mathcal{T}_0, \mathfrak{T}_0)$ is another topological space, then $B(\mathcal{T}, \mathcal{T}_0)$ denotes the set of bounded Borel-measurable functions from $\mathcal{T} \to \mathcal{T}_0$. For a measurable space $(M, \mathfrak{M})$

we write $\mathcal{P}(M)$ for the set of probability measures on $(M, \mathfrak{M})$ and $\mathcal{M}(M)$ for the set of all measures on $(M, \mathfrak{M})$. For $d \in \mathbb{N}$ and $a \in C^1(\mathbb{R}^d, \mathbb{R}^{d\times d})$ we write

$$\overrightarrow{\operatorname{div}}(a) = \left(\sum_{i=1}^{d} \partial_i a_{ij}\right)_{j=1}^{d}.$$

Moreover, when $f \in C^1(\mathbb{R}^d, \mathbb{R})$ we denote the gradient of f by $\operatorname{grad} f$ and the divergence of f by $\operatorname{div} f$. When $g \in C^2(\mathbb{R}^d, \mathbb{R})$ we denote the Hessian of g by $\operatorname{Hess} g$.

1.2 Stochastic filtering problem

In this section we are following Bain and Crisan [3]. Let $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space with a normal filtration $(\mathcal{F}_t)_{t\geq 0}$.[1] Let $d, p \in \mathbb{N}$ and let $X : [0, \infty) \times \Omega \to \mathbb{R}^d$ be a d-dimensional stochastic process satisfying, for all $t \in [0, \infty)$ and $\mathbb{P}$-a.s., that

$$X_t = X_0 + \int_0^t f(X_s)\,\mathrm{d}s + \int_0^t \sigma(X_s)\,\mathrm{d}V_s\,, \tag{1}$$

where $f : \mathbb{R}^d \to \mathbb{R}^d$ and $\sigma : \mathbb{R}^d \to \mathbb{R}^{d\times p}$ are globally Lipschitz continuous functions and $V : [0, \infty) \times \Omega \to \mathbb{R}^p$ is a p-dimensional $(\mathcal{F}_t)_{t\geq 0}$-adapted Brownian motion. Then X admits the infinitesimal generator $A : \mathcal{D}(A) \to B(\mathbb{R}^d)$ given, for all $\varphi \in \mathcal{D}(A)$, by

$$A\varphi = \langle f, \nabla\varphi\rangle + \operatorname{Tr}(a \operatorname{Hess}\varphi), \tag{2}$$

where $\mathcal{D}(A)$ denotes the domain of the differential operator A and where we defined the function $a(\cdot) = \frac{1}{2}\sigma(\cdot)\sigma'(\cdot) : \mathbb{R}^d \to \mathbb{R}^{d\times d}$. We assume from now on that a dense core for the domain $\mathcal{D}(A)$ is $C_c^2(\mathbb{R}^d)$.

In the context of stochastic filtering, X is called the *signal process*. Further, we assume the *observation process* $Y : [0, \infty) \times \Omega \to \mathbb{R}^m$ to be given, for all $t \in [0, \infty)$ and $\mathbb{P}$-a.s., by

$$Y_t = \int_0^t h(X_s)\,\mathrm{d}s + W_t\,, \tag{3}$$

where $W : [0, \infty) \times \Omega \to \mathbb{R}^m$ is an $(\mathcal{F}_t)_{t\geq 0}$-adapted Brownian motion independent of V. The *sensor function* $h : \mathbb{R}^d \to \mathbb{R}^m$ is a globally Lipschitz continuous function

[1] We call the filtration $(\mathcal{F}_t)_{t\geq 0}$ *normal*, if

- $\mathcal{F}_0$ contains all $\mathbb{P}$-nullsets of $\mathcal{F}$, and
- $(\mathcal{F}_t)_{t\geq 0}$ is right-continuous.

with the property that for all $t \in [0, \infty)$, $\mathbb{P}$-a.s.,

$$\mathbb{E}\left[\int_0^t h(X_s)^2 \, \mathrm{d}s\right] < \infty \text{ and } \mathbb{E}\left[\int_0^t Z_s h(X_s)^2 \, \mathrm{d}s\right] < \infty,$$

where the stochastic process $Z : [0, \infty) \times \Omega \to \mathbb{R}$ is defined such that for all $t \in [0, \infty)$,

$$Z_t = \exp\{-\int_0^t h(X_s) \, \mathrm{d}W_s - \frac{1}{2}\int_0^t h(X_s)^2 \, \mathrm{d}s\}.$$

We specify the *observation filtration* for $t \geq 0$ by

$$\mathcal{Y}_t = \sigma(Y_s, s \in [0, t]) \vee \mathcal{N} \quad \text{and write} \quad \mathcal{Y} = \sigma\left(\bigcup_{t\in[0,\infty)} \mathcal{Y}_t\right),$$

where $\mathcal{N}$ is the collection of $\mathbb{P}$-nullsets of $\mathcal{F}$. Then we are interested in the $(\mathcal{Y}_t)_{t\geq 0}$-adapted stochastic process $\pi : [0, \infty) \times \Omega \to \mathcal{P}(\mathbb{R}^d)$ that is defined by the requirement that for all $\varphi \in B(\mathbb{R}^d, \mathbb{R})$ and $t \in [0, \infty)$ it holds $\mathbb{P}$-a.s. that

$$\pi_t \varphi = \mathbb{E}\left[\varphi(X_t) \,|\mathcal{Y}_t\right].$$

The process π is often called the *filter*. Under this model, the stochastic process Z is an $(\mathcal{F}_t)_{t\geq 0}$-martingale and by Novikov's condition we can use Girsanov's theorem to define the change of measure given by $\frac{\mathrm{d}\tilde{\mathbb{P}}^t}{\mathrm{d}\mathbb{P}}\Big|_{\mathcal{F}_t} = Z_t, t \geq 0$. Note that on $\bigcup_{t\in[0,\infty)} \mathcal{F}_t$ we have a consistent measure $\tilde{\mathbb{P}}$ in place of $\tilde{\mathbb{P}}^t$. Moreover, the signal and observation processes X and Y are independent under the new measure and Y is a Brownian motion under $\tilde{\mathbb{P}}$. Furthermore, under $\tilde{\mathbb{P}}$, we can define the stochastic process $\rho : [0, \infty) \times \Omega \to \mathcal{M}(\mathbb{R}^d)$ by the requirement that for all $\varphi \in B(\mathbb{R}^d, \mathbb{R})$ and $t \in [0, \infty)$ it holds $\mathbb{P}$-a.s. that

$$\rho_t \varphi = \mathbb{E}\left[\varphi(X_t) \exp\{\int_0^t h(X_s) \, \mathrm{d}Y_s - \frac{1}{2}\int_0^t h(X_s)^2 \, \mathrm{d}s\}\Big| \mathcal{Y}_t\right]. \tag{4}$$

The following important Proposition 1, known in the literature as the Kallianpur-Striebel formula, justifies the terminology to call ρ the *unnormalised filter*.

Proposition 1 ***(Kallianpur-Striebel formula)*** *For all $t \geq 0$ and $\varphi \in B(\mathbb{R}^d, \mathbb{R})$ it holds $\tilde{\mathbb{P}}$-a.s. that*

$$\pi_t(\varphi) = \frac{\rho_t(\varphi)}{\rho_t(\mathbf{1})} = \frac{\tilde{\mathbb{E}}\left[\varphi(X_t) \exp\{\int_0^t h(X_s) \, \mathrm{d}Y_s - \frac{1}{2}\int_0^t h(X_s)^2 \, \mathrm{d}s\}\Big| \mathcal{Y}\right]}{\tilde{\mathbb{E}}\left[\exp\{\int_0^t h(X_s) \, \mathrm{d}Y_s - \frac{1}{2}\int_0^t h(X_s)^2 \, \mathrm{d}s\}\Big| \mathcal{Y}\right]},$$

where $\mathbf{1}$ *is the constant function* $\mathbb{R}^d \ni x \mapsto 1$.

The proof of Proposition 1 can be found in, e.g., [3].

1.3 Filtering equation and general splitting method

It is well established in the literature (see, e.g., [3]), that the unnormalised filter ρ, defined in (4), satisfies the *filtering equation*, i.e. for all $t \geq 0$ it holds $\tilde{\mathbb{P}}$-a.s. that

$$\rho_t(\varphi) = \pi_0(\varphi) + \int_0^t \rho_s(A\varphi)\,\mathrm{d}s + \int_0^t \rho_s(\varphi h')\,\mathrm{d}Y_s. \tag{5}$$

Moreover, it is known (see, e.g., [3, Theorem 7.8]) that if π_0 is absolutely continuous with respect to Lebesgue measure and such that it has a square-integrable density, and if additionally the sensor function h is uniformly bounded, then ρ_t admits a square-integrable density p_t with respect to the Lebesgue measure on $\mathbb{R}^d$. Then, assuming the necessary regularity for p_t (see, e.g., [3, Theorem 7.12], for the precise condition), the Zakai equation (5) implies that, for all $t \geq 0$ and $\varphi \in C_c^\infty(\mathbb{R}^d, \mathbb{R})$, we have $\tilde{\mathbb{P}}$-a.s. that

$$\rho_t(\varphi) = \int_{\mathbb{R}^d} \varphi(x) p_t(x)\,\mathrm{d}x,$$

The PDE method we will consider is from [9] and seeks to approximate the following stochastic partial differential equation (SPDE) for the density p_t given, for all $t \geq 0$, $x \in \mathbb{R}^d$, and $\mathbb{P}$-a.s. as

$$p_t(x) = p_0(x) + \int_0^t A^* p_s(x)\,\mathrm{d}s + \int_0^t h'(x) p_s(x)\,\mathrm{d}Y_s$$

and relies on the splitting-up algorithm described in [31] and [33]. Here, A^* is the formal adjoint of the infinitesimal generator A of the signal process X, given by the relation

$$\int_{\mathbb{R}^d} A\varphi(x) p_t(x)\,\mathrm{d}x = \int_{\mathbb{R}^d} \varphi(x) A^* p_t(x)\,\mathrm{d}x;\ t \geq 0.$$

Choose a final time $T > 0$ and an integer $N \in \mathbb{N}$ and let $\{t_0 = 0 < \cdots < t_N = T\}$ be a discretisation of the time interval $[0, T]$. Then the first step of the splitting-up approach, also called *prediction* step, is to numerically approximate the *Fokker-Planck equation*

$$\begin{aligned} \frac{\partial q}{\partial t}(t, z) &= A^* q(t, z), \qquad (t, z) \in (0, T] \times \mathbb{R}^d, \\ q(0, z) &= p_0(z), \qquad\qquad z \in \mathbb{R}^d, \end{aligned} \tag{6}$$

over the discretised interval. To this end, note that the first prediction step of the method consists of the numerical approximation of the solution q^1 of the PDE

$$\begin{aligned} \frac{\partial q^1}{\partial t}(t,z) &= A^* q^1(t,z), && (t,z) \in (0,t_1] \times \mathbb{R}^d, \\ q^1(0,z) &= q^0(0,z) := p_0(z), && z \in \mathbb{R}^d. \end{aligned}$$

We denote the numerical approximation of $q^1(t_1,\cdot)$ by $\tilde{p}^1$. Next, we employ the second step of the method, the so-called *correction* step, which consists of the normalisation of the obtained Fokker-Planck approximations using the observation process Y, as given by (3), and the Kallianpur-Striebel formula (see Proposition 1). To illustrate this, the first correction step is calculated as follows. Let

$$z_1 = \frac{1}{t_1 - t_0}(Y_{t_1} - Y_{t_0}),$$

consider the function

$$\mathbb{R}^d \ni z \mapsto \xi_1(z) = \exp\left(-\frac{1}{2}||z_1 - h(z)||^2\right),$$

and define for all $z \in \mathbb{R}^d$,

$$p^1(z) = C_1 \xi_1(z) \tilde{p}^1(z),$$

where C_1 is the normalisation constant such that $\int_{\mathbb{R}^d} p^1(z)\,\mathrm{d}z = 1$.

Therefore, we formulate the splitting-up method below in Note 1.

Note 1 The full method is defined by iterating the above steps with $p^0(\cdot) = p_0(\cdot)$ and such that for all $n \in \{1,\dots,N\}$ we iteratively calculate

1) an approximation $\tilde{p}^n$ of the solution to

$$\begin{aligned} \frac{\partial q^n}{\partial t}(t,z) &= A^* q^n(t,z), && (t,z) \in (t_{n-1},t_n] \times \mathbb{R}^d, \\ q^n(0,z) &= p^{n-1}(z), && z \in \mathbb{R}^d, \end{aligned} \tag{7}$$

   at time t_n and
2) the normalisation based on

$$z_n = \frac{1}{t_n - t_{n-1}}(Y_{t_n} - Y_{t_{n-1}})$$

   and the function

$$\mathbb{R}^d \ni z \mapsto \xi_n(z) = \exp\left(-\frac{t_n - t_{n-1}}{2}||z_n - h(z)||^2\right),$$

so that we can define for all $z \in \mathbb{R}^d$,

$$p^n(z) = \frac{1}{C_n} \xi_n(z) \tilde{p}^n(z),$$

where $C_n = \int_{\mathbb{R}^d} \xi_n(z) \tilde{p}^n(z) \, \mathrm{d}z$.

In this article, we replace the predictor step 1 in Note 1 above by a deep neural network approximation algorithm to avoid an explicit space discretisation which has exponential complexity in the space dimension d. This will be achieved by representing each $\tilde{p}^n(z)$ by a feed-forward neural network and approximating the initial value problem (7) based on its stochastic representation using a sampling procedure.

2 Feynman-Kac representation and auxiliary diffusion

In this section we consider the case when the coefficient functions of the signal and the observation processes are sufficiently smooth and thus allow the expansion of the partial differential operator A^*. Based on this expansion we can rewrite the Fokker-Planck equation (6) as a Kolmogorov equation plus, in general, a zeroth-order term. The reason to do so is that the so obtained representation enables the use of the Feynman-Kac formula (see Theorem 1 below) to rewrite the solution of the PDE problem as an expectation of an appropriately chosen stochastic process. Thus, we can approximate this expectation by Monte-Carlo sampling from the diffusion.

This particular approach follows a recent stream of research into deep learning based approximations of PDEs which is mainly focused on high dimensional problems, see, e.g. [4, 5, 13, 21] and related works within the context of stochastic optimal control [24, 25, 36, 36, 37]. Alternative approaches, typically based on incorporating the PDE directly into the loss function, for the approximation of a neural network representation of solutions of PDEs are also actively developed in the literature, see, for example, [2, 38, 39].

2.1 Fokker-Planck equation

We begin by expanding the differential operator under the assumption that it has smooth coefficient functions. As before, we let d, $p \in \mathbb{N}$, $f = (f_i)_{i=1}^d \in C^1(\mathbb{R}^d, \mathbb{R}^d)$, $\sigma = (\sigma_{ij})_{j=1,\dots,p}^{i=1,\dots,d} \in C^2(\mathbb{R}^d, \mathbb{R}^{d\times p})$, and let $a = (a_{ij})_{i,j=1}^d$ be the function that maps $x \mapsto \frac{1}{2}\sigma(x)\sigma'(x)$. Furthermore, f and σ are assumed to have bounded derivatives and $A^* : C_c^\infty(\mathbb{R}^d, \mathbb{R}) \to C(\mathbb{R}^d, \mathbb{R})$ be the partial differential operator with the property that for all $\varphi \in C_c^\infty(\mathbb{R}^d, \mathbb{R})$,

$$A^*\varphi = -\sum_{i=1}^d \frac{\partial}{\partial x_i} f_i \varphi + \sum_{i,j=1}^d \frac{\partial^2}{\partial x_i \partial x_j} a_{ij} \varphi.$$

Then, for all $\varphi \in C_c^\infty(\mathbb{R}^d, \mathbb{R})$ we have

$$A^*\varphi = \mathrm{Tr}(a\,\mathrm{Hess}\,\varphi) + \langle 2\overrightarrow{\mathrm{div}}(a) - f, \mathrm{grad}\,\varphi\rangle + \mathrm{div}(\overrightarrow{\mathrm{div}}(a) - f)\varphi. \tag{8}$$

Definition 1 Let $d, p \in \mathbb{N}$, $f = (f_i)_{i=1}^d \in C_b^1(\mathbb{R}^d, \mathbb{R}^d)$, let $\sigma = (\sigma_{ij})_{j=1,\dots,p}^{i=1,\dots,d} \in C_b^2(\mathbb{R}^d, \mathbb{R}^{d\times p})$, and let $a = (a_{ij})_{i,j=1}^d \in C_b^2(\mathbb{R}^d, \mathbb{R}^{d\times d})$ be the function that maps $x \mapsto \frac{1}{2}\sigma(x)\sigma'(x)$. Then we define the partial differential operator $\hat{A} : C_c^\infty(\mathbb{R}^d, \mathbb{R}) \to C(\mathbb{R}^d, \mathbb{R})$ such that for all $\varphi \in C_c^\infty(\mathbb{R}^d, \mathbb{R})$,

$$\hat{A}\varphi = \mathrm{Tr}(a\,\mathrm{Hess}\,\varphi) + \langle 2\overrightarrow{\mathrm{div}}(a) - f, \mathrm{grad}\,\varphi\rangle$$

and we define the function $r : \mathbb{R}^d \to \mathbb{R}$ such that for all $x \in \mathbb{R}^d$,

$$r(x) = \mathrm{div}(\overrightarrow{\mathrm{div}}(a) - f)(x).$$

Remark 1 The assumptions on the derivatives of the coefficients f and σ may be relaxed by assuming that they are locally Lipschitz in conjunction with a suitable assumption so that the moments of the diffusion remain bounded.

Lemma 1 *For all $x \in \mathbb{R}^d$ the operator $\hat{A}$ defined in Definition 1 is the infinitesimal generator of the Itô diffusion $\hat{X} : [0,\infty) \times \Omega \to \mathbb{R}^d$ given, for all $t \geq 0$ and $\mathbb{P}$-a.s. by*

$$\hat{X}_t = x + \int_0^t b(\hat{X}_s)\mathrm{d}s + \int_0^t \sigma(\hat{X}_s)\mathrm{d}\hat{W}_s,$$

where $\hat{W} : [0,\infty) \times \Omega \to \mathbb{R}^d$ is a d-dimensional Brownian motion and $b : \mathbb{R}^d \to \mathbb{R}^d$ is the function

$$b = 2\overrightarrow{\mathrm{div}}(a) - f.$$

Proof cf. [26, Chapter IV, Theorem 6.1] □

The next Theorem 1 is the well-known Feynman-Kac formula.

Theorem 1 ***(Feynman-Kac formula)*** *Let $d \in \mathbb{N}$, $T > 0$, $k \in C(\mathbb{R}^d, [0,\infty))$, let $\hat{A}$ be the operator defined in Definition 1, and let $\psi : \mathbb{R}^d \to \mathbb{R}$ be a function[2]. If $v \in C_b^{1,2}([0,T) \times \mathbb{R}^d, \mathbb{R})$ satisfies the Cauchy problem*

$$\begin{aligned} -\frac{\partial v}{\partial t}(t,x) + k(x)v(t,x) &= \hat{A}v(t,x), \qquad (t,x) \in [0,T) \times \mathbb{R}^d, \\ v(T,x) &= \psi(x), \qquad x \in \mathbb{R}^d, \end{aligned} \tag{9}$$

then we have for all $(t,x) \in [0,T) \times \mathbb{R}^d$ that

$$v(t,x) = \mathbb{E}\left[\psi(\hat{X}_T)\exp\left(-\int_t^T k(\hat{X}_\tau)\,\mathrm{d}\tau\right)\middle|\, \hat{X}_t = x\right],$$

where $\hat{X}$ is the diffusion generated by $\hat{A}$ at most polynomially growing function[2].

Proof See [28, Chapter 5, Theorem 7.6]. The assumption that the coefficients of $\hat{A}$ have bounded derivatives ensures that the required conditions are met. □

From Theorem 1 above we can deduce the Corollary 1 below about the initial value problem corresponding to (9).

Corollary 1 *Under the assumptions of the previous Theorem 1, suppose that $u \in C_b^{1,2}((0,T]\times\mathbb{R}^d,\mathbb{R})$ satisfies the Cauchy problem*

$$\begin{aligned} \frac{\partial u}{\partial t}(t,x) + k(x)u(t,x) &= \hat{A}u(t,x), \qquad (t,x)\in(0,T]\times\mathbb{R}^d, \\ u(0,x) &= \psi(x), \qquad x\in\mathbb{R}^d. \end{aligned} \tag{10}$$

Then, for all $(t,x)\in(0,T]\times\mathbb{R}^d$, we have that

$$u(t,x) = \mathbb{E}\left[\psi(\hat{X}_t)\exp\left(-\int_0^t k(\hat{X}_\tau)\,\mathrm{d}\tau\right)\middle|\,\hat{X}_0 = x\right],$$

where $\hat{X}$ is the diffusion generated by $\hat{A}$.

Proof For all $(s,x)\in(0,T]\times\mathbb{R}^d$, set $u(s,x) = v(T-s,x)$, where $v\in C_b^{1,2}([0,T)\times\mathbb{R}^d,\mathbb{R})$ satisfies (9). Then, $u\in C_b^{1,2}((0,T]\times\mathbb{R}^d,\mathbb{R})$ and (9) implies that u satisfies (10), i.e.

$$\begin{aligned} \frac{\partial u}{\partial t}(t,x) + k(x)u(t,x) &= \hat{A}u(t,x), \qquad (t,x)\in(0,T]\times\mathbb{R}^d, \\ u(0,x) &= \psi(x), \qquad x\in\mathbb{R}^d. \end{aligned}$$

Hence, we are in the realm of the claim. Further, since $\hat{X}$ is a time-homogeneous Markov process, we have for all $(s,x)\in(0,T]\times\mathbb{R}^d$,

$$\begin{aligned} u(s,x) = v(T-s,x) &= \mathbb{E}\left[\psi(\hat{X}_T)\exp\left(-\int_{T-s}^T k(\hat{X}_\tau)\,\mathrm{d}\tau\right)\middle|\,\hat{X}_{T-s} = x\right] \\ &= \mathbb{E}\left[\psi(\hat{X}_s)\exp\left(-\int_0^s k(\hat{X}_\tau)\,\mathrm{d}\tau\right)\middle|\,\hat{X}_0 = x\right]. \end{aligned} \tag{11}$$

Therefore, replacing s by t in the above equation (11) proves the assertion. □

In view of our original problem, the Fokker-Planck equation (7) that we want to solve numerically, in this case, reads for all $n\in\{1,\dots,N\}$ as

$$\begin{aligned} \frac{\partial q^n}{\partial t}(t,z) &= \hat{A}q^n(t,z) + r(z)q^n(t,z), \qquad (t,z)\in(t_{n-1},t_n]\times\mathbb{R}^d, \\ q^n(0,z) &= p^{n-1}(z), \qquad z\in\mathbb{R}^d. \end{aligned}$$

[2] I.e. there exist real numbers $\lambda\geq 1$ and $L\geq 0$ such that for all $x\in\mathbb{R}^d$, $|\psi(x)|\leq L(1+\|x\|^{2\lambda})$.

Therefore, considering $k = -r$, and assuming that $-r$ is non-negative in (10), Corollary 1 gives, for all $n \in \{1, \dots, N\}$, $t \in (t_{n-1}, t_n]$, $z \in \mathbb{R}^d$, the representation

$$q^n(t, z) = \mathbb{E}\left[p^{n-1}(\hat{X}_t) \exp\left(\int_{t_{n-1}}^{t} r(\hat{X}_\tau)\, \mathrm{d}\tau \right) \middle| \, \hat{X}_{t_{n-1}} = z \right].$$

To be explicit, in the next subsection we formulate two specific examples of filtering problems and show how they fit into the framework developed thus far by providing the auxiliary diffusion and conditional expectation representations for each of these cases.

2.2 Two simple examples of filtering models

The following are two simple examples for filtering problems, which will be used as benchmarks in the numerical studies. The results from the previous section hold true for these examples, even though the corresponding coefficients do not satisfy the uniform boundedness. The linear filter in Example 1 below is formulated in arbitrary finite dimensions. Additionally, we give in Example 2 the model for the purely one-dimensional, but nonlinear, Benes filter. For more details on the presented examples the reader may consult [3, Chapter 6]

Example 1 **(Linear Filter)** For the Kalman filter we have the signal process given by the coefficient functions

$$f(x) = Mx + \eta \quad \text{and} \quad \sigma(x) = \Sigma$$

and the observation process is determined by the sensor function

$$h(x) = Hx + \gamma.$$

In this case, when X_0 is assumed normally distributed, the solution π_t of the filtering problem is known to be a Gaussian distribution with known mean and covariance, see for example [3, Chapter 6.2]. Then, for the linear filter, we see that the auxiliary diffusion process takes the form

$$\hat{X}_t = \hat{X}_0 - \int_0^t M\hat{X}_s + \eta\, \mathrm{d}s + \int_0^t \Sigma\, \mathrm{d}\hat{W}_s,$$

and is thus the well-known Ornstein-Uhlenbeck process, plus an additional drift represented by η, with explicit representation, in terms of the usual matrix exponential,

$$\hat{X}_t = \exp\{-Mt\}\left(\hat{X}_0 + \int_0^t \exp\{Ms\}\Sigma\, \mathrm{d}\hat{W}_s \right).$$

Moreover, $r(x) = -\operatorname{div} f(x) = -\operatorname{Tr} M$. Then the method for the linear filter is given by the representation

$$q^n(t,z) = \mathbb{E}\Big[\, p^{n-1}(\hat{X}_t) \exp\left(-\operatorname{Tr} M(t-t_{n-1})\right) \Big|\, \hat{X}_{t_{n-1}} = z \Big].$$

Example 2 **(Benes Filter)** For the Benes filter we have one-dimensional signal and observation processes. The signal is given by the coefficient functions

$$f(x) = \alpha\sigma \tanh(\beta + \alpha x/\sigma) \quad \text{and} \quad \sigma(x) \equiv \sigma \in \mathbb{R},$$

where $\alpha, \beta \in \mathbb{R}$ and the observation is given by the affine-linear sensor function

$$h(x) = h_1 x + h_2,$$

with $h_1, h_2 \in \mathbb{R}$. Note that here we have given a special case of the more general class of Benes filters, see [3, Chapter 6.1]. Now, similar to the previous example, we compute the auxiliary diffusion

$$\hat{X}_t = \hat{X}_0 - \int_0^t \alpha\sigma \tanh(\beta + \alpha \hat{X}_s/\sigma)\, \mathrm{d}s + \int_0^t \sigma\, \mathrm{d}\hat{W}_s,$$

and the coefficient

$$r(x) = -\operatorname{div} f(x) = -\alpha^2 \operatorname{sech}^2(\beta + \alpha x/\sigma).$$

This yields the scheme for the Benes case to be derived from the representation

$$q^n(t,z) = \mathbb{E}\left[\, p^{n-1}(\hat{X}_t) \exp\left(-\int_{t_{n-1}}^t \alpha^2 \operatorname{sech}^2(\beta + \alpha \hat{X}_\tau/\sigma)\, \mathrm{d}\tau\right) \Big|\, \hat{X}_{t_{n-1}} = z \right].$$

In the following subsection we introduce the optimisation problem associated with the filtering problem discussed above, and which is based on the simulation of the auxiliary diffusion.

2.3 Optimisation problem for the prior

Above we have found a Feynman-Kac representation for the solution of the Fokker-Planck equation in the case of smooth coefficients of the signal process. In analogy to [5, Proposition 2.7] we formulate the following result about a minimisation property in Proposition 2 below.

Proposition 2 *Let $d \in \mathbb{N}$, $T > 0$, $a < b \in \mathbb{R}$, $k \in C(\mathbb{R}^d, [0,\infty))$, let $\hat{A}$ be the operator defined in Definition 1, and let $\psi : \mathbb{R}^d \to \mathbb{R}$ be an at most polynomially*

growing function. Suppose that $u \in C_b^{1,2}((0,T]\times\mathbb{R}^d,\mathbb{R})$ *satisfies the Cauchy problem*

$$\begin{aligned}\frac{\partial u}{\partial t}(t,x)+k(x)u(t,x)&=\hat{A}u(t,x), \qquad (t,x)\in(0,T]\times\mathbb{R}^d,\\ u(0,x)&=\psi(x), \qquad x\in\mathbb{R}^d,\end{aligned}$$

let $\xi:\Omega\to[a,b]^d$ *be a continuous, uniformly distributed* $\mathcal{F}_0$*-random variable and let* $\hat{\mathbb{X}}$ *be the diffusion generated by* $\hat{A}$ *and with the property that,* $\mathbb{P}$*-a.s.,* $\hat{\mathbb{X}}_0=\xi$*. Then there exists a unique continuous function* $U:[a,b]^d\to\mathbb{R}$ *such that*

$$\begin{aligned}&\mathbb{E}\left[\left|\psi(\hat{\mathbb{X}}_T)\exp\left(-\int_0^T k(\hat{\mathbb{X}}_\tau)\,\mathrm{d}\tau\right)-U(\xi)\right|^2\right]\\ &=\inf_{v\in C([a,b]^d,\mathbb{R})}\mathbb{E}\left[\left|\psi(\hat{\mathbb{X}}_T)\exp\left(-\int_0^T k(\hat{\mathbb{X}}_\tau)\,\mathrm{d}\tau\right)-v(\xi)\right|^2\right]\end{aligned}$$

and for all $x\in[a,b]^d$ *we have* $U(x)=u(T,x)$*.*

Proof Let $T>0$. For all $x\in\mathbb{R}^d$, let $\hat{X}^x$ be the $\hat{A}$-diffusion starting at x. Since, by assumption, k is non-negative and ψ has polynomial growth it follows that there exist real numbers $L>0$ and $\lambda\geq 1$ such that for all $x\in\mathbb{R}^d$,

$$\mathbb{E}\left[\left|\psi(\hat{X}_T^x)\exp\left(-\int_0^T k(\hat{X}_\tau^x)\,\mathrm{d}\tau\right)\right|^2\right]\leq\mathbb{E}\left[\left|L(1+\|\hat{X}_T^x\|^{2\lambda})\right|^2\right]<\infty. \tag{12}$$

Further, because the map

$$\mathbb{R}^d\ni x\mapsto\psi(\hat{X}_T^x)\exp\left(-\int_0^T k(\hat{X}_\tau^x)\,\mathrm{d}\tau\right)$$

is continuous and at most polynomially growing, [5, Lemma 2.6] implies that the function

$$\mathbb{R}^d\ni x\mapsto\mathbb{E}\left[\psi(\hat{X}_T^x)\exp\left(-\int_0^T k(\hat{X}_\tau^x)\,\mathrm{d}\tau\right)\right] \tag{13}$$

is continuous. Note that the function

$$\mathbb{R}^d\times\Omega\ni(x,\omega)\mapsto\psi(\hat{X}_T^x(\omega))\exp\left(-\int_0^T k(\hat{X}_\tau^x(\omega))\,\mathrm{d}\tau\right) \tag{14}$$

is $\mathcal{B}([a,b]^d)\otimes\mathcal{F}/\mathcal{B}(\mathbb{R}^d)$-measurable. Finally, by virtue of (12), (13), (14), and [5, Proposition 2.2], there exists a unique continuous function $U:[a,b]^d\to\mathbb{R}$

such that

$$\int_{[a,b]^d} \mathbb{E}\left[\left|\psi(\hat{X}_T^x)\exp\left(-\int_0^T k(\hat{X}_\tau^x)\,\mathrm{d}\tau\right)-U(x)\right|^2\right]\mathrm{d}x$$
$$=\inf_{v\in C([a,b]^d,\mathbb{R})}\int_{[a,b]^d}\mathbb{E}\left[\left|\psi(\hat{X}_T^x)\exp\left(-\int_0^T k(\hat{X}_\tau^x)\,\mathrm{d}\tau\right)-v(x)\right|^2\right]\mathrm{d}x$$

and such that for all $x\in[a,b]^d$ we have

$$U(x)=\mathbb{E}\left[\psi(\hat{X}_T^x)\exp\left(-\int_0^T k(\hat{X}_\tau^x)\,\mathrm{d}\tau\right)\right].$$

Now, for all $V\in C([a,b]^d,\mathbb{R})$ we have that the map

$$C([0,T],\mathbb{R}^d)\ni\gamma\mapsto\left|\psi(\gamma_T)\exp\left(-\int_0^T k(\gamma_\tau)\,\mathrm{d}\tau\right)-V(\gamma_0)\right|^2\in\mathbb{R}$$

is at most polynomially growing. Thus [5, Lemma 2.6] implies that for all $V\in C([a,b]^d,\mathbb{R})$ we have that

$$\mathbb{E}\left[\left|\psi(\mathbb{X}_T)\exp\left(-\int_0^T k(\mathbb{X}_\tau)\,\mathrm{d}\tau\right)-V(\xi)\right|^2\right]$$
$$=\frac{1}{(b-a)^d}\int_{[a,b]^d}\mathbb{E}\left[\left|\psi(X_T^x)\exp\left(-\int_0^T k(X_\tau^x)\,\mathrm{d}\tau\right)-V(x)\right|^2\right]\mathrm{d}x.$$

Then, for all $V\in C([a,b]^d,\mathbb{R})$ with the property that

$$\mathbb{E}\left[\left|\psi(\mathbb{X}_T)\exp\left(-\int_0^T k(\mathbb{X}_\tau)\,\mathrm{d}\tau\right)-V(\xi)\right|^2\right]$$
$$=\inf_{v\in C([a,b]^d,\mathbb{R})}\mathbb{E}\left[\left|\psi(\mathbb{X}_T)\exp\left(-\int_0^T k(\mathbb{X}_\tau)\,\mathrm{d}\tau\right)-v(\xi)\right|^2\right],$$

a direct calculation shows that

$$\int_{[a,b]^d}\mathbb{E}\left[\left|\psi(X_T^x)\exp\left(-\int_0^T k(X_\tau^x)\,\mathrm{d}\tau\right)-V(x)\right|^2\right]\mathrm{d}x$$
$$=\inf_{v\in C([a,b]^d,\mathbb{R})}\int_{[a,b]^d}\mathbb{E}\left[\left|\psi(X_T^x)\exp\left(-\int_0^T k(X_\tau^x)\,\mathrm{d}\tau\right)-v(\xi)\right|^2\right]\mathrm{d}x.$$

Hence also this minimiser is unique and equals U and finally

$$\mathbb{E}\left[\left|\psi(\mathbb{X}_T)\exp\left(-\int_0^T k(\mathbb{X}_\tau)\,\mathrm{d}\tau\right)-U(\xi)\right|^2\right]$$
$$=\inf_{v\in C([a,b]^d,\mathbb{R})}\mathbb{E}\left[\left|\psi(\mathbb{X}_T)\exp\left(-\int_0^T k(\mathbb{X}_\tau)\,\mathrm{d}\tau\right)-v(\xi)\right|^2\right].$$

This together with the Feynman-Kac formula proves the result. □

Proposition 2 guarantees that we have a feasible minimisation problem to approximate by the learning algorithm.

In the following section we will describe the machine learning algorithm used to approximate the PDE using the above optimisation representation. Furthermore, we derive the Monte-Carlo method used to approximate the normalisation constant in the correction step. We thus specify our full method.

3 Splitting method for the neural network representation of the posterior

Here, we introduce some of the terminology specific to the field of neural networks. For an in-depth discussion on deep learning terminology, algorithms and applications, we refer the reader to the book [16]. Thereafter, we specify explicitly the learning algorithm employed in our method. Subsequently, we derive the Monte-Carlo method used in the correction step of the splitting method and the section ends with a full description of our algorithm in pseudocode.

3.1 Neural network model for prediction step

Definition 2 Given $L\in\mathbb{N}$ and $(l_0,\dots,l_L)\in\mathbb{N}^{L+1}$ and a continuous function $\tau\in C^0(\mathbb{R})$ a (feed-forward fully-connected) neural network $\mathcal{NN}$ is a function $\mathbb{R}^{l_0}\to\mathbb{R}^{l_L}$ given by

$$\mathcal{NN}(x)=\left(\bigcirc_{i=1}^{L}\tau\odot\mathcal{A}_i^{(l_{i-1},l_i)}\right)(x),$$

where the $\mathcal{A}_i^{(l_i-1,l_i)}$ are affine maps $\mathbb{R}^{l_{i-1}}\to\mathbb{R}^{l_i}$ of the form $x\mapsto A_ix+b_i$, $A_i\in\mathbb{R}^{l_{i-1}\times l_i}$, $b_i\in\mathbb{R}^{l_i}$.

The number L is called the *depth* of the network, the function ρ is called the *activation function*, and the matrices and vectors A_i and b_i are called the *weights* and *biases* of the i-th *hidden layer*, respectively. In the experimental part of this work, we consider the activation function $\tau(x)=\tanh(x)$. Other common choices include the ReLu activation function $\mathrm{ReLu}(x)=\max\{0,x\}$ or the sigmoidal function

$\sigma(x) = 1/(1 + \exp(-x))$, among many others. Collectively, the parameters of the function represented by the neural network are denoted

$$\theta = \{\{A_i^{jk}\}_{jk}, \{b_i^j\}_j : i = 1, \dots, L\} \subset \mathbb{R}^{(\sum_{i=2}^{L} l_{i-1} l_i + l_i)}$$

and we sometimes write $\mathcal{NN}_\theta$ to note the dependence explicitly. The symbol $\bigcirc$ denotes function composition and the symbol $\odot$ denotes componentwise function composition, i.e. for any $\mathcal{A}\colon \mathbb{R}^m \to \mathbb{R}^n$ and $x \in \mathbb{R}^m$ we have

$$(\tau \odot \mathcal{A})(x) = (\tau((Ax)_1), \dots, \tau((Ax)_n))' \in \mathbb{R}^n.$$

In general, the weights and biases of a neural network are to be chosen freely and are commonly determined using an optimisation algorithm such as gradient descent, stochastic gradient descent [7] or variants thereof, such as AdaGrad [12], momentum methods [23], or the ADAM optimiser [29]. Our method of choice in this work is the ADAM optimiser. The optimisation procedure based on supplied *training data* is in this context commonly referred to as *learning*. Notice, however, that there is an important distinction between learning and optimisation. While optimisation is concerned with the pure minimisation (or maximisation) of a target function, the goal of learning is to create a model that *generalises* well, i.e. performs well on unseen inputs. Thus, in certain contexts it is undesirable to fit a model too closely to the provided training data, since this can degrade the out-of-sample performance, a phenomenon known as *overfitting*.

Moreover, the initialisation of the parameters is a crucial part of the performance of the optimisation and defines its own branch of research within machine learning. Additionally, the neural network model has various free parameters that are neither given by the original problem nor are they determined by the learning procedure. These include the architecture of the network, i.e. the depth L, the layer widths l_i, or certain parameters in the optimisation algorithm such as the *learning rate* (i.e. the step size of the gradient descent method) or training batch size, and are commonly to be chosen heuristically or from experience. These are commonly called *hyperparameters*.

Additionally, we employ the technique of batch-normalisation [27] in our computations, but refrain here from a detailed discussion. The reader is referred to the original work [27] or the book [16].

In order to use a neural network model for the filtering problem, we employ the splitting-up method to first split the problem into the solution of a deterministic Fokker-Planck PDE and the subsequent inclusion of the observation using the likelihood and normalisation procedure.

The PDE step is where we incorporate the deep learning method to solve the Fokker-Planck equation over a rectangular domain $\Omega_d = [\alpha_1, \beta_1] \times \cdots \times [\alpha_d, \beta_d]$, for the sake of computational feasibility. Its solution is reformulated into the optimisation problem over function space given in Proposition 2. This optimisation problem is approximated

by the optimisation

$$\inf_{\theta\in\mathbb{R}^{\sum_{i=2}^{L} l_{i-1}l_i+l_i}} \mathbb{E}\left[\left|\psi(\hat{\mathbb{X}}_T)\exp\left(-\int_0^T k(\hat{\mathbb{X}}_\tau)\,\mathrm{d}\tau\right)-\mathcal{NN}_\theta(\xi)\right|^2\right]$$

where the solution of the PDE is represented by a neural network and the infinite-dimensional function space has been parametrised by the neural network parameters θ. To this problem we will be able to apply a gradient descent method for the determination of the parameters in the model to minimise the associated *loss function*

$$\begin{aligned}&\mathcal{L}(\theta;\{\xi^i,\{\hat{\mathbb{X}}^i_{\tau_j}\}_{j=0}^J\}_{i=1}^{N_b})\\&=\frac{1}{N_b}\sum_{i=1}^{N_b}\left|\psi(\hat{\mathbb{X}}^i_T)\exp(-\sum_{j=0}^{J-1}k(\hat{\mathbb{X}}^i_{\tau_j})(\tau_{j+1}-\tau_j))-\mathcal{NN}_\theta(\xi^i)\right|^2,\end{aligned}$$

where N_b is the batch size and $\{\xi^i,\{\hat{\mathbb{X}}^i_{\tau_j}\}_{j=0}^J\}_{i=1}^{N_b}$ is a training batch of independent identically distributed realisations ξ^i of $\xi\sim\mathcal{U}(\Omega_d)$ and $\{\hat{\mathbb{X}}^i_{\tau_j}\}_{j=0}^J$ the approximate i.i.d. realisations of sample paths of the auxiliary diffusion started at ξ^i over the time-grid $\tau_0=0<\tau_1<\cdots<\tau_{J-1}<\tau_J=T$. The sample paths are, for example, approximated using the Euler-Maruyama or a similiar SDE simulation method [30]. In practice, since the solution of the Fokker-Planck equation we seek is non-negative, we usually augment the loss $\mathcal{L}$ by an additional term to encourage positivity of the neural network and use

$$\tilde{\mathcal{L}}(\theta;\{\xi^i,\{\hat{\mathbb{X}}^i_{\tau_j}\}_{j=0}^J\}_{i=1}^{N_b})=\mathcal{L}(\theta;\{\xi^i,\{\hat{\mathbb{X}}^i_{\tau_j}\}_{j=0}^J\}_{i=1}^{N_b})+\lambda\sum_{i=1}^{N_b}\max\{0,-\mathcal{NN}_\theta(\xi^i)\}$$

with the hyperparameter λ to be chosen.

Thus, in the notation of Sect. 1.3 we replace the Fokker-Planck solution by a neural network model, i.e. we *postulate* a neural network model

$$\tilde{p}_n(z)=\mathcal{NN}(z),$$

with support on Ω_d. Therefore we require the a priori chosen domain to capture most of the mass of the probability distribution it is approximating.

3.2 Monte-Carlo correction step

We then realise the correction step via Monte-Carlo sampling over the bounded rectangular domain Ω_d to approximate the integral

$$\int_{\mathbb{R}^d}\xi_n(z)\mathcal{NN}(z)\,\mathrm{d}z=\int_{\mathbb{R}^d}\exp\left(-\frac{t_n-t_{n-1}}{2}||z_n-h(z)||^2\right)\mathcal{NN}(z)\,\mathrm{d}z,$$

where, as defined earlier, $z_n = \frac{1}{t_n - t_{n-1}}(Y_{t_n} - Y_{t_{n-1}})$. Now, since the neural network has $\mathrm{supp}(\mathcal{NN}) \subseteq \Omega_d$ this is equal to the integral

$$\int_{\Omega_d} \exp\left(-\frac{t_n - t_{n-1}}{2}||z_n - h(z)||^2\right)\mathcal{NN}(z)\,\mathrm{d}z. \tag{15}$$

In general, to achieve the approximation of the above integral via Monte-Carlo, one needs to be able to sample from an appropriate density. Moreover, see Remark 2 below for possible alternatives.

Remark 2 The usage of the Monte-Carlo method to perform the normalisation is optional in our low-dimensional experimental setup below, where efficient quadrature methods are a good alternative. However, we chose to design our algorithm around the sampling based method, as a large part of the literature devoted to machine learning algorithms for PDEs aims to design grid-free (in space) methods to achieve better performance in high dimensions. In that regard, we specify our algorithm so that it can be tested in higher dimensional, grid-free, settings without major alterations in subsequent studies.

Since, in this work, we are considering mainly affine-linear sensor functions $h(x) = h_1 x + h_2$, we illustrate the Monte-Carlo integration method in this case. Notice that the likelihood function then reads

$$\begin{aligned}
\xi_n(z) &= \exp\left(-\frac{t_n - t_{n-1}}{2}(z_n - h_1 z - h_2)^2\right) \\
&= \exp\left(-\frac{(t_n - t_{n-1})h_1^2}{2}\left(\frac{z_n - h_2}{h_1} - z\right)^2\right) \\
&= \exp\left(-\frac{1}{2}\left(\frac{\frac{z_n - h_2}{h_1} - z}{((t_n - t_{n-1})h_1^2)^{-1/2}}\right)^2\right) \\
&= \frac{\sqrt{2\pi}}{\sqrt{(t_n - t_{n-1})h_1^2}}\mathcal{N}_{\mathrm{pdf}}\left(\frac{z_n - h_2}{h_1}, \frac{1}{(t_n - t_{n-1})h_1^2}\right)(z),
\end{aligned}$$

where $\mathcal{N}_{\mathrm{pdf}}(\mu, \sigma^2)$ denotes the probability density function of a normal distribution with mean μ and variance σ^2. Therefore, we can write the integral (15) as

$$\frac{\sqrt{2\pi}}{\sqrt{(t_n - t_{n-1})h_1^2}}\mathbb{E}_Z[\mathcal{NN}(Z)]; \qquad Z \sim \mathcal{N}\left(\frac{z_n - h_2}{h_1}, \frac{1}{(t_n - t_{n-1})h_1^2}\right).$$

As it is straightforward to numerically sample from a Gaussian distribution, the Monte-Carlo approximation derived above is implementable so that we can compute the normalisation constant C_n numerically. Thus, we can explicitly represent the approx-

Algorithm 1 Splitting method for neural network representation of posterior

Require: Time-grid $0 = t_0, t_1, \ldots, t_N = T$
Require: Initial density p_0
Require: Observations $Y_0, \ldots, Y_N$
Require: Affine-linear sensor function $h(x) = h_1 x + h_2$
function $\text{POSTERIOR}_0(x)$
  return $p_0(x)$
end function
for n from 1 to N **do**
  Initialize $\mathcal{NN}^n_{\theta_{init}}$
  $\mathcal{NN}^n_{\theta_{trained}} \leftarrow \text{TRAINNET}(\mathcal{NN}^n_{\theta_{init}}, \text{POSTERIOR}_{n-1})$
  Compute $z_n = \frac{1}{t_n - t_{n-1}}(Y_{t_n} - Y_{t_{n-1}})$
  Draw $N_{samples}$ samples Z_j from $\mathcal{N}\left(\frac{z_n - h_2}{h_1}, \frac{1}{(t_n - t_{n-1})h_1^2}\right)$
  Compute $C_n = \frac{1}{N_{samples}} \sum_{j=1}^{N_{samples}} \mathcal{NN}^n_{\theta_{trained}}(Z_j)$
  function $\text{POSTERIOR}_n(x)$
    return $\frac{1}{C_n} \exp\left(-\frac{t_n - t_{n-1}}{2}(z_n - h(x))^2\right) \mathcal{NN}^n_{\theta_{trained}}(x)$
  end function
end for

imate posterior density

$$p^n(z) = \frac{1}{C_n} \xi_n(z) \tilde{p}^n(z),$$

and use it as the initial condition for the next time iteration. Therefore, our scheme is fully recursive and can be applied sequentially.

Remark 3 Additional techniques to adjust the support of the approximation are needed when iterating the scheme over a long time duration/many steps as, eventually, in many common filtering setups, it will be the case that the mass of the posterior moves outside the initial domain. The way to mitigate this problem depends, in general, on the specific filtering model under consideration and will be subject of further investigation.

3.3 Algorithm summary

We briefly summarise our full approximation method. In Algorithm 1 we present the pseudocode for the splitting method as we apply it to the filtering equation. The method is designed to be fully grid-free in space, for the reasons outlined above in Remark 2. Furthermore, a main feature of our algorithm is the ability to iterate it over successive time steps so that observations may arrive sequentially, and there is no strict requirement for them to be available beforehand. This is an especially important property in real-world filtering scenarios where observations are typically processed *online*. Therefore, Algorithm 1 is formulated as an iterative procedure over the observation time-grid $0 = t_0, t_1, \ldots, t_N = T$.

Algorithm 1 includes a network training step which we clarify in the pseudocode presented in Algorithm 2. Note that we give here, in the interest of clarity, a simpli-

fied version of the actual gradient-descent method that we employ in the numerical studies in Sect. 4. However, the general rationale behind both methods is the same gradient-descent based process. The important parameters of the learning method are the number of training steps, usually called *epochs*, N_{epochs}, the training batch size N_b as well as the learning rate κ that determines the step size of the gradient descent step to adjust the parameters of the neural network. In our studies below, we chose an adaptive learning rate based on a *learning rate schedule*. That is, we choose a set of integers $0 = K_0 < K_1 < \cdots < K_M < K_{M+1} = \infty$ as cut-off steps, and a set of learning rates $\kappa_0, \ldots, \kappa_M$ and adjust the learning rate during the training procedure according to

$$\kappa(n) = \sum_{i=0}^{M} \kappa_i \mathbf{1}_{[K_i, K_{i+1})}(n), \qquad n = 1, \ldots, N_{epochs}.$$

Since the training method is based on N_b samples of the auxiliary diffusion in each epoch, the full training uses $N_b N_{epochs}$ independent Monte-Carlo samples in total.

Figure 1 illustrates the neural network architecture that we are using in the numerical experiments exhibited in Sect. 4. This architecture is inspired by the one used in previous experiments by other authors, for example in [22].

4 Numerical results for the splitting scheme

We implement Algorithm 1 for Examples 1 and 2 using Tensorflow [1]. For a practical guide on the implementation of deep learning algorithms, the reader may consult [15].

In all examples below, the neural network architecture is a feed-forward fully connected neural network with a one-dimensional input layer, two hidden layers with a layer width of 51 neurons each, and an output layer of dimension one. For the optimisation algorithm we chose the ADAM optimiser and performed the training over 6002 epochs with a batch size of 600 samples. Note that during our testing we found that the batch size had a crucial effect on the performance of our algorithm. If chosen too small, the training procedure we used failed to discover an acceptable set of parameters

Algorithm 2 Network training (simplified)

Require: N_{epochs}, N_b, κ
function TRAINNET($\mathcal{NN}_{\theta_{init}}$, POSTERIOR$_0$)
  $\theta \leftarrow \theta_{init}$
  for n from 1 to N_{epochs} **do**
    Draw N_b samples $\{\xi^i, \{\hat{\mathbb{X}}^i_{\tau_j}\}_{j=0}^J\}$
    Compute $\nabla_\theta \tilde{\mathcal{L}}(\theta; \{\xi^i, \{\hat{\mathbb{X}}^i_{\tau_j}\}_{j=0}^J\}_{i=1}^{N_b})$
    $\theta \leftarrow \theta - \kappa \nabla_\theta \tilde{\mathcal{L}}(\theta; \{\xi^i, \{\hat{\mathbb{X}}^i_{\tau_j}\}_{j=0}^J\}_{i=1}^{N_b})$ ▷ Gradient descent
  end for
  $\theta_{trained} \leftarrow \theta$
  return $\mathcal{NN}_{\theta_{trained}}$
end function

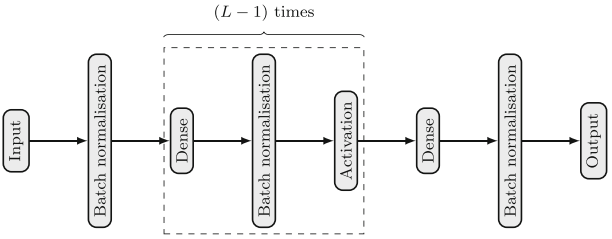

Fig. 1 Neural network architecture used in our experiments. We use the architecture similar to the one employed in [22]. The input is initially transformed by a batch-normalisation layer [27] and then a sequence of a triple (dashed box) consisting of an affine linear (Dense) transformation, a batch normalisation, and a subsequent application of the *tanh*-nonlinearity (Activation) is applied $L-1$ times, where L is the depth of the neural network. Before returning, another affine transformation (Dense) and then a final batch-normalisation are applied

Table 1 Parameters used in the numerical experiment for the one-dimensional linear filter, case 1

x_0	y_0	M	η	Σ	H	γ	Δt
0.0	0.0	−1.0	0.0	0.1	90.0	0.0	0.01

for the neural network. If chosen too large, we observed that the training was slowed down on our hardware.

4.1 One-dimensional linear filter

Here we present the numerical results for the one-dimensional linear filtering setting outlined in Example 1. We first present in Sect. 4.1.1 a filter that does not move outside the domain, based on an Ornstein-Uhlenbeck signal process. Next, we show the results obtained for the linear filter with a signal process that moves toward the domain boundary in Sect. 4.1.2.

4.1.1 Linear filter, case 1: $M = -1$, $\eta = 0$

We are considering a linear filter with an Ornstein-Uhlenbeck signal process using the set of parameters, corresponding to the notation in Example 1, given in Table 1. Moreover, as the initial condition we chose a Gaussian density with mean 0.0 and standard deviation 0.01. We iterate our method over 60 timesteps up to a final time of 0.6.

The results of our approximation method applied to the linear filter with Ornstein-Uhlenbeck signal are visualised in Fig. 2. The full evolution of the estimated posterior is shown in Fig. 2 (a). In particular, we see that the approximated solution stays within the considered spatial domain $[-0.5, 0.5]$. This feature will be important when we discuss the linear filter with drift below. Moreover, note that in correspondence with the theoretical expectations, the variance of the approximated posterior distribution initially increases and then stays constant, with an oscillating mean which is affected by the sequentially arriving observations. In Fig. 2 (b)-(d) we present three snapshots of the numerical solution obtained with our modified splitting scheme. In each of the

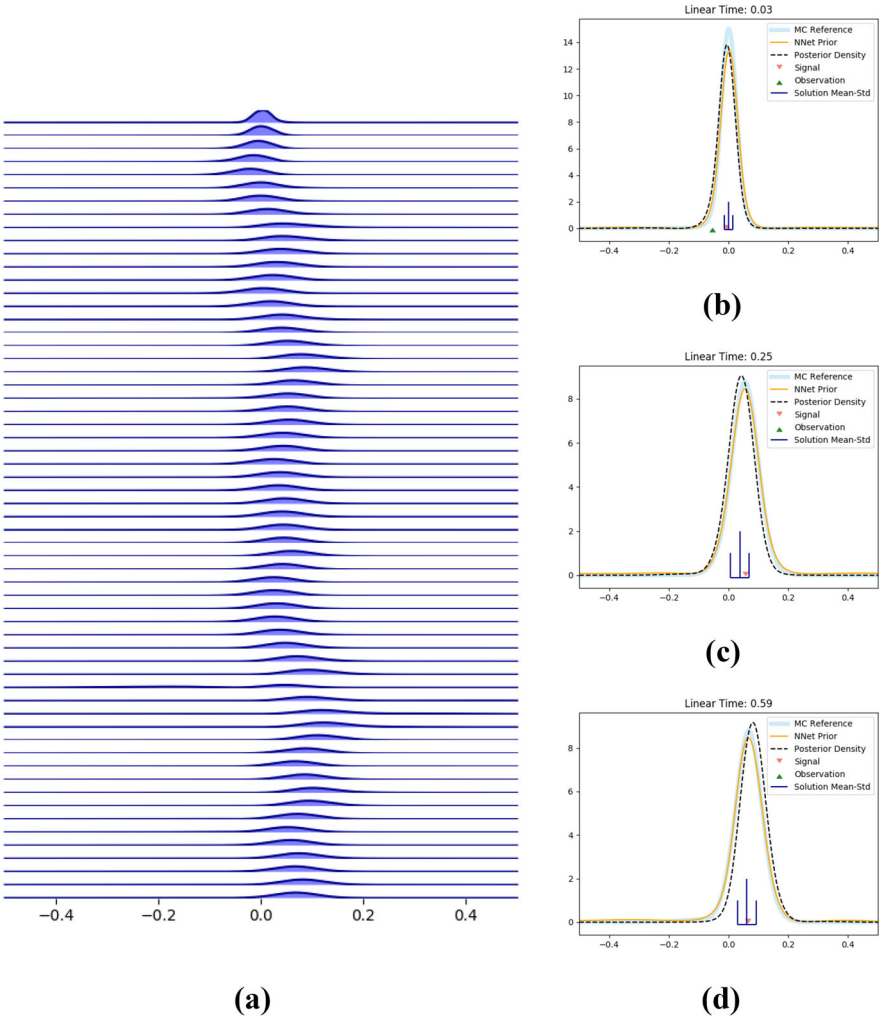

Fig. 2 Results of the combined splitting-up/machine-learning approximation applied iteratively to the linear filtering problem, case 1. (a) The full evolution of the estimated posterior distribution produced by our method, plotted at all intermediate timesteps, from top to bottom. (b)-(d) Snapshots of the approximation at an early time, $t = 0.03$, an intermediate time, $t = 0.25$, and a late time, $t = 0.59$, obtained after 3, 25 and 59 iterations of our method, respectively. The black dotted line in each graph shows the estimated posterior, the yellow line the prior estimate represented by the neural network, and the light-blue shaded line shows the Monte-Carlo reference solution for the Fokker-Planck equation

three graphs, the yellow line shows the plot of the neural network over the observed domain, approximating the solution of the Fokker-Planck PDE with initial condition given by the posterior density obtained from the previous step. The blue-shaded line is a pointwise Monte-Carlo reference solution based on the Sobol sequence over the spatial domain. This is used as a visual guide to judge the quality of the shape the neural network represents. Note that this is *not* the theoretical solution for the filtering

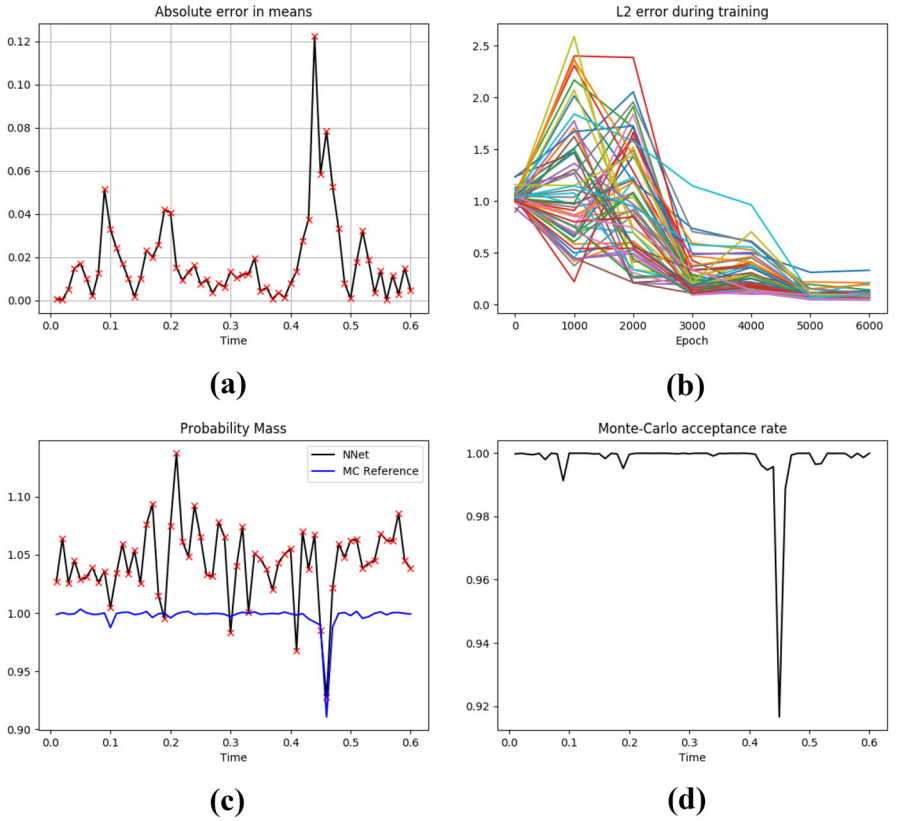

Fig. 3 Error and diagnostics for linear filter, case 1. **a** Absolute error in means between the approximated distribution and the exact solution. **b** L_2 error of the neural network during training with respect to the Monte-Carlo reference solution. (c) Probability mass of the neural network prior. (d) Monte-Carlo acceptance rate

problem, but a reference solution for the Fokker-Planck equation for the prior, based on the initial condition given by the previous estimate. The black dashed line shows the plot of the normalised posterior using the method outlined above. Additionally, we plotted the mean and standard deviation of the exact solution to the considered filtering problem as three blue vertical lines, the higher one representing the theoretical mean and the lower ones the standard deviation. The position of the signal is plotted as a red inverted triangle and the position of the observation as a green triangle. Note that the observation may lie outside the domain and thus may not be present in the graph.

The errors and diagnostics for the linear filter, case 1, are shown in Fig. 3. Here, Fig. 3 (a) is a graph of the absolute value of the difference between the mean of the approximate posterior and the theoretical posterior mean. We see that the error fluctuates about a constant value, which is the desired result. In particular, we do not expect a decreasing error but rather a stable one. This shows that the method is stable when iterated over many time steps. The two peaks at times 0.44 and 0.46 are explained below and due to a statistical outlier in the observation/likelihood. Fig. 3 (b) shows the

Table 2 Parameters used in the numerical experiment for the one-dimensional linear filter, case 2

x_0	y_0	M	η	Σ	H	γ	Δt
0.0	0.0	1.0	−1.0	0.1	90.0	0.0	0.01

training performance of the neural network approximation measured by the L_2-error over the domain between the Monte-Carlo reference solution of the Fokker-Planck PDE and the neural network representation across the training epochs. Each line in the graph represents a separate neural network, one for each timestep. Here we can see that the neural network training consistently converges to the Monte-Carlo solution across all time steps. The probability mass of the neural net and Monte-Carlo reference solutions of the Fokker-Planck equation is plotted against time in Fig. 3 (c), where we conclude that the machine learning approximation tends to slightly overestimate the mass of the solution. Lastly, in Fig. 3 (d) we plot the acceptance rate of the Monte-Carlo integration of the neural network prior with respect to the likelihood as specified in our algorithm. A sample from the density in the likelihood is accepted if it lies within the considered domain, and rejected if it falls outside the domain. This is so because of the assumption that the neural network has support strictly within the domain. Here we can see that the quality of the likelihood is a major factor in the success of the method. The dip in the acceptance rate can be found to negatively affect the mass of the neural network prior (Fig. 3 (c)) and finally results in a spike in the error (Fig. 3 (a)). Furthermore, it is noteworthy that the method seems to recover from this event after the next two time steps which is a further hint at the stability of our method.

4.1.2 Linear filter, case 2: $M = 1$, $\eta = -1$

The second numerical study of this work is based on the Kalman filtering setting with the set of parameters given in Table 2.

As the initial density we chose a Gaussian density with mean 0.0 and standard deviation 0.01. The domain over which we resolve the solution was chosen as the interval $[-0.8, 0.4]$, in anticipation of the drift of the signal. We again iterated our method over 60 time steps up to a final time of 0.6. The results of the simulation are shown in Fig. 4.

As expected, the mean of the posterior moves to the left by approximately 0.01 units of the domain at each time step. Furthermore, the standard deviation also initially increases as time progresses.

In Fig. 5 (a) we show the error between the means of the approximate posterior and the mean of the exact solution of the linear filter. Up to the time of about 0.44, we observe a steady oscillation within a range of 0.00-0.05, except for a few spikes which are classified as outliers. Thereafter, the error increases systematically. This phenomenon coincides with the observation in Fig. 5 (c) where, after the time of about 0.44 the total mass of the network prior becomes unstable. Before this time, the neural network model has slightly overestimated the mass of the solution of the Fokker-Planck equation. Fig. 5 (d) provides the interpretation for the cause of this phenomenon. It shows the Monte-Carlo acceptance rates for the integration method of the neural network prior with respect to the density given by the likelihood. The

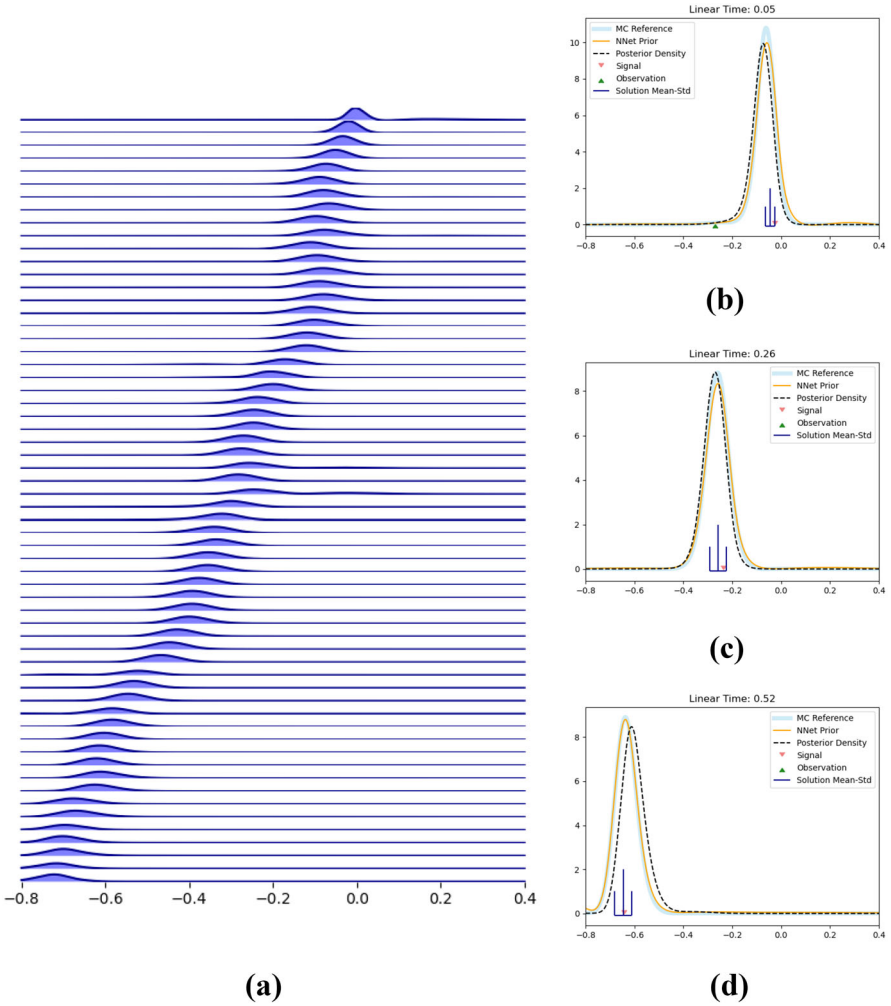

Fig. 4 Results of the combined splitting-up/machine-learning approximation applied iteratively to the linear filtering problem, case 2. (a) The full evolution of the estimated posterior distribution produced by our method, plotted at all intermediate timesteps. (b)-(d) Snapshots of the approximation at an early time, $t = 0.05$, an intermediate time, $t = 0.26$, and a late time, $t = 0.52$, obtained after 5, 26 and 52 iterations of our method, respectively. The black dotted line in each graph shows the estimated posterior, the yellow line the prior estimate represented by the neural network, and the light-blue shaded line shows the Monte-Carlo reference solution for the Fokker-Planck equation

drop in acceptance rate shows that the samples from the likelihood increasingly lie outside the domain of the neural network prior, which depletes the quality of the approximation. Therefore, a strong likelihood within the domain we are considering is an important factor in the performance of our algorithm. This observations is also connected to the so-called signal-to-noise ratio which we need to be high in order to perform an accurate normalisation using the sampling method. Finally, Fig. 5 (b) is an

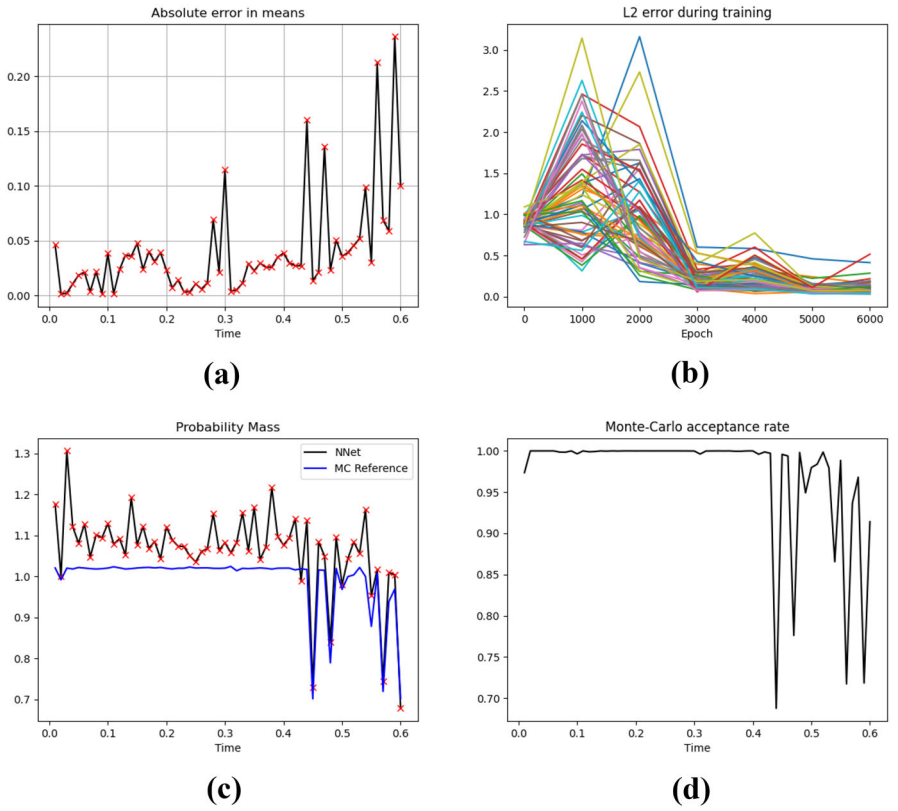

Fig. 5 Error and diagnostics for linear filter, case 2. **a** Absolute error in means between the approximated distribution and the exact solution. **b** L_2 error of the neural network during training with respect to the Monte-Carlo reference solution. **c** Probability mass of the neural network prior. **d** Monte-Carlo acceptance rate

Table 3 Parameters used in the numerical experiment for the one-dimensional Benes filter

x_0	y_0	α	β	σ	h_1	h_2	Δt
0.0	0.0	3.0	0.0	0.5	3.0	0.0	0.1

illustration of the neural network training progress. Each line in the plot corresponds to a timestep, and shows the L_2 error against the training epoch with respect to the Monte-Carlo reference solution of the Fokker-Planck equation.

4.2 One-dimensional Benes filter

The third numerical study of this work is based on the nonlinear Benes filtering setting outlined in Example 2. Here, we are considering the set of parameters, corresponding to the notation in Example 2, given in Table 3.

The initial condition was again chosen to be the Gaussian density with mean 0.0 and standard deviation 0.01. This time, however, we chose a different, larger, time step in order to observe the characteristic bimodality appearing in the solution of the Benes filter. This also necessitated the choice of a larger domain for the neural net, which here was chosen to be the interval $[-4.0, 4.0]$. The results were calculated by iterating our scheme over 12 timesteps for the approximation of the Benes filter and are plotted in Fig. 6. The feature we like to stress in this nonlinear example is the development of the bimodal density that is resolved by our method, in particular in Fig. 6 (c) and (d).

The error and diagnostic plots are shown in Fig. 7. The absolute error in Fig. 7 (a) shows a steady oscillation, and Fig. 7 (b) indicates that the neural network training converges to the Monte-Carlo reference solution across all time steps. Moreover, the probability mass plotted in Fig. 7 (c) oscillates around the correct value 1.0 with a slight tendency to underestimate, also for the Monte-Carlo reference. The initially low mass is explained by the sharp drop of the peak of the initial Gaussian during the first timestep, which is difficult to capture. As observed in the linear cause though, the method seems to be able to recover from occasional inaccuracies. Fig. 7 (d) shows the Monte-Carlo acceptance rate for the correction step. The final drop is still acceptable, as the value of $\sim 93\%$ acceptance rate is still reasonable. These results demonstrate an ability of our algorithm to also track nonlinear problems over several timesteps.

5 Conclusion and outlook

As observed, an important factor in the success of our method lies in accurately determining the domain of resolution *before* beginning the iterative procedure. As the mass of the density begins to move outside our observed window, the results will degrade quickly. A possible solution is to shift the observed window in a suitable manner at regular time intervals to obtain an adaptive method. Moreover, due to the Monte-Carlo sampling based correction step, which relies on samples from the likelihood, we need a high signal-to-noise ratio to maintain an accurate evaluation of the integral in the domain. If the acceptance rate of Monte-Carlo samples from the likelihood drops significantly, the results in our method deteriorate. This can be counteracted by sampling more points from the distribution. However, if the likelihood spread is too large, this will significantly slow down the algorithm.

We do not think that dealing with the domain boundaries is an unsurmountable problem. Future research will focus on investigating approaches to deal with the motion of the posterior outside of the domain.

Note further that, because the density of the optimal filter changes continuously in time, our algorithm is a natural candidate for *transfer learning* the parameters of the neural net instead of retraining them from a random initialisation at every time step. Further details on the area of transfer learning can be found, for example, in [16, Chapter 15.2].

Although we found a similar performance of our method across a range of different hyperparameters such as the batch size, the network architecture, etc. the optimal choice of these for our given problem of filtering remains open.

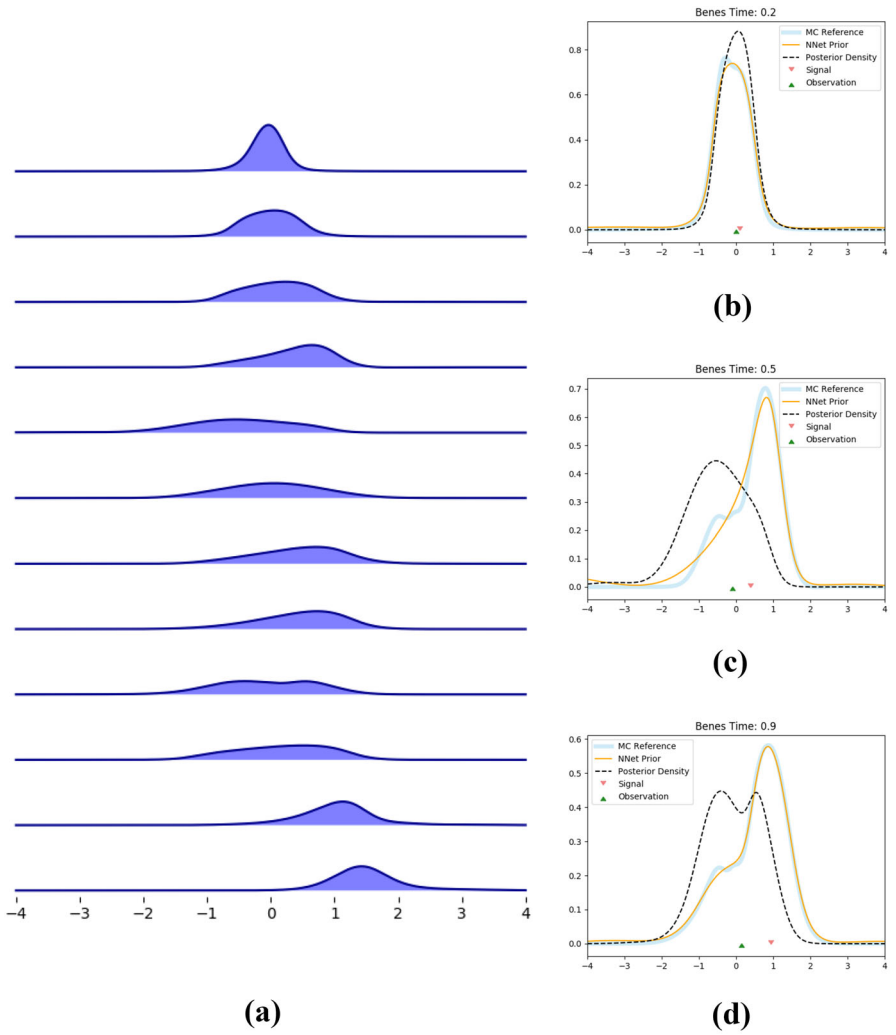

Fig. 6 Results of the combined splitting-up/machine-learning approximation applied iteratively to the Benes filtering problem. (a) The full evolution of the estimated posterior distribution produced by our method, plotted at all intermediate timesteps. (b)–(d) Snapshots of the approximation at an early time, $t = 0.2$, an intermediate time, $t = 0.5$, and a late time, $t = 0.9$, obtained after 2, 5 and 9 iterations of our method, respectively. The black dotted line in each graph shows the estimated posterior, the yellow line the prior estimate represented by the neural network, and the light-blue shaded line shows the Monte-Carlo reference solution for the Fokker-Planck equation

A further direction of future study will be a detailed error analysis of the presented algorithm. This is a complex problem because the approximation performed here introduces inaccuracies at various stages. The first ones are the usual simulations of the signal and observation processes, as well as now also the auxiliary diffusion. Moreover, the machine learning algorithm introduces an error in estimating the Fokker-

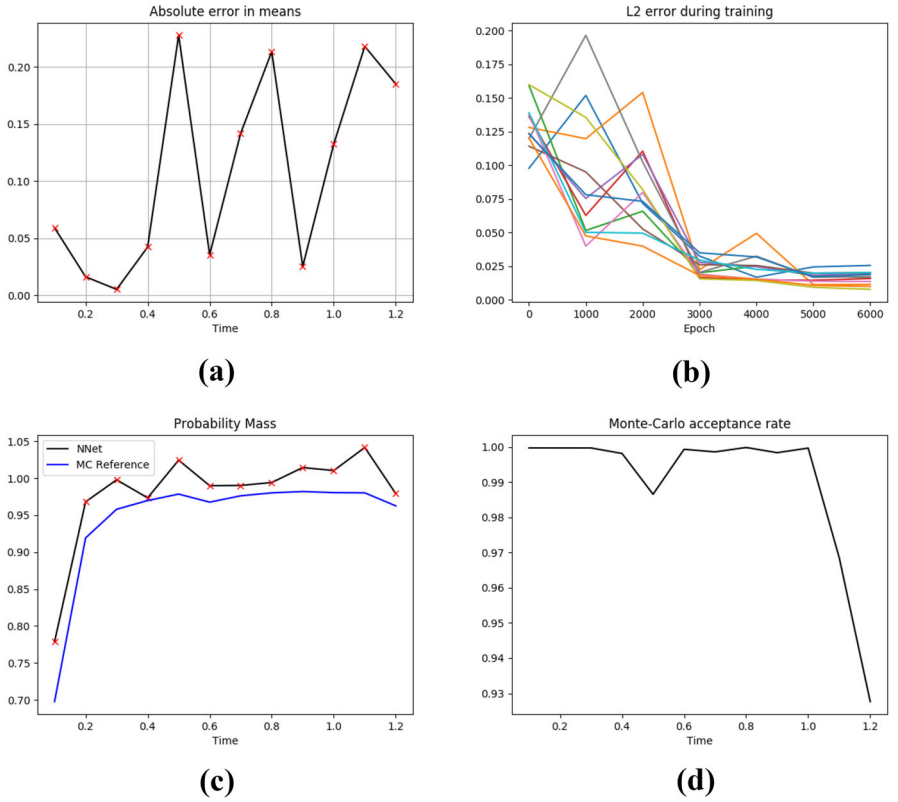

Fig. 7 Error and diagnostics for the Benes filter. **a** Absolute error in means between the approximated distribution and the exact solution. **b** L_2 error of the neural network during training with respect to the Monte-Carlo reference solution. **c** Probability mass of the neural network prior. **d** Monte-Carlo acceptance rate

Planck PDE solution. Finally, the error due to the Monte-Carlo normalisation in the correction step must be analysed.

Acknowledgements Dan Crisan and Salvador Ortiz-Latorre acknowledge the support of the project STORM: Stochastics for Time-Space Risk Models, from the Research Council of Norway (RCN). Project number: 274410. Dan Crisan and Alexander Lobbe were partially supported by the European Research Council (ERC) under the European Union's Horizon 2020 Research and Innovation Programme (ERC, Grant Agreement No 856408, Project Title: Stochastic Transport in the Upper Ocean Dynamics). Alexander Lobbe is grateful for the financial support from Department of Mathematics, University of Oslo, Norway.

Data Availability The document contains no data.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: Large-scale machine learning on heterogeneous systems (2015). https://www.tensorflow.org/. Software available from tensorflow.org
2. Al-Aradi, A., Correia, A., Naiff, D., Jardim, G., Saporito, Y.: Solving nonlinear and high-dimensional partial differential equations via deep learning (2018)
3. Bain, A., Crisan, D.: Fundamentals of Stochastic Filtering. Springer, New York (2008)
4. Beck, C., Becker, S., Cheridito, P., Jentzen, A., Neufeld, A.: Deep splitting method for parabolic pdes (2019)
5. Beck, C., Becker, S., Grohs, P., Jaafari, N., Jentzen, A.: Solving stochastic differential equations and Kolmogorov equations by means of deep learning. arXiv e-prints arXiv:1806.00421 (2018)
6. Bensoussan, A.: Stochastic control of partially observable systems. Cambridge University Press, Cambridge (1992). https://doi.org/10.1017/CBO9780511526503
7. Bottou, L.: Stochastic learning. In: O. Bousquet, U. von Luxburg (eds.) Advanced Lectures on Machine Learning, Lecture Notes in Artificial Intelligence, LNAI 3176, pp. 146–168. Springer Verlag, Berlin (2004). http://leon.bottou.org/papers/bottou-mlss-2004
8. Brigo, D., Hanzon, B.: On some filtering problems arising in mathematical finance. Insurance Math. Econom. **22**(1), 53–64 (1998). https://doi.org/10.1016/S0167-6687(98)00008-0. The interplay between insurance, finance and control (Aarhus, 1997)
9. Cai, Z., Le Gland, F., Zhang, H.: An adaptive local grid refinement method for nonlinear filtering. Ph.D. thesis, INRIA (1995)
10. Crisan, D., Rozovskii, B. (eds.): The Oxford handbook of nonlinear filtering. Oxford University Press, Oxford (2011)
11. Date, P., Ponomareva, K.: Linear and non-linear filtering in mathematical finance: a review. IMA J. Manag. Math. **22**(3), 195–211 (2011). https://doi.org/10.1093/imaman/dpq008
12. Duchi, J., Hazan, E., Singer, Y.: Adaptive subgradient methods for online learning and stochastic optimization. J. Mach. Learn. Res. **12**(null), 2121–2159 (2011)
13. E, W., Han, J., Jentzen, A.: Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. arXiv e-prints arXiv:1706.04702 (2017)
14. Galanis, G., Louka, P., Katsafados, P., Pytharoulis, I., Kallos, G.: Applications of kalman filters based on non-linear functions to numerical weather predictions. Annales Geophysicae **24**(10), 2451–2460 (2006). https://doi.org/10.5194/angeo-24-2451-2006. Cited By 78
15. Géron, A.: Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. O'Reilly Media, Sebastopol (2019)
16. Goodfellow, I., Bengio, Y., Courville, A.: Deep learning. MIT Press, Cambridge (2016)
17. Gyöngy, I.: Approximations of stochastic partial differential equations. In: Stochastic partial differential equations and applications (Trento, 2002), *Lecture Notes in Pure and Appl. Math.*, vol. 227, pp. 287–307. Dekker, New York (2002)
18. Gyöngy, I., Krylov, N.: On the rate of convergence of splitting-up approximations for SPDEs. In: Stochastic inequalities and applications, *Progr. Probab.*, vol. 56, pp. 301–321. Birkhäuser, Basel (2003)
19. Gyöngy, I., Krylov, N.: On the splitting-up method and stochastic partial differential equations. Ann. Probab. **31**(2), 564–591 (2003). https://doi.org/10.1214/aop/1048516528

20. Gyöngy, I., Krylov, N.: An accelerated splitting-up method for parabolic equations. SIAM J. Math. Anal. **37**(4), 1070–1097 (2005). https://doi.org/10.1137/S0036141003437903
21. Han, J., E, W.: Deep Learning Approximation for Stochastic Control Problems. arXiv e-prints arXiv:1611.07422 (2016)
22. Han, J., Jentzen, A., Weinan, E.: Solving high-dimensional partial differential equations using deep learning. Proc. Natl. Acad. Sci. **115**(34), 8505–8510 (2018). https://doi.org/10.1073/pnas.1718942115. (https://www.pnas.org/content/115/34/8505)
23. Hinton, G.E., Rumelhart, D., Williams, R.J.: Learning internal representations by error propagation. MIT Press **8** (1986)
24. Huré, C., Pham, H., Bachouch, A., Langrené, N.: Deep neural networks algorithms for stochastic control problems on finite horizon, part i: convergence analysis (2018)
25. Huré, C., Pham, H., Warin, X.: Some machine learning schemes for high-dimensional nonlinear pdes (2019)
26. Ikeda, N., Watanabe, S.: Stochastic Differential Equations and Diffusion Processes. North-Holland Publishing Company, Amsterdam (1981)
27. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift (2015)
28. Karatzas, I., Shreve, S.E.: Brownian Motion and Stochastic Calculus. Springer, New York (1998)
29. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
30. Kloeden, P.E., Platen, E.: Numerical Solution of Stochastic Differential Equations. Springer, Berlin (1992)
31. Le Gland, F.: Time discretization of nonlinear filtering equations. In: Proceedings of the 28th IEEE Conference on Decision and Control, pp. 2601–2606. IEEE (1989)
32. Le Gland, F.: Splitting-up approximation for SPDEs and SDEs with application to nonlinear filtering. In: Stochastic partial differential equations and their applications (Charlotte, NC, 1991), *Lect. Notes Control Inf. Sci.*, vol. 176, pp. 177–187. Springer, Berlin (1992). https://doi.org/10.1007/BFb0007332
33. LeGland, F.: Splitting-up approximation for spde's and sde's with application to nonlinear filtering. In: Stochastic partial differential equations and their applications, pp. 177–187. Springer (1992)
34. Llopis, F.P., Kantas, N., Beskos, A., Jasra, A.: Particle filtering for stochastic Navier-Stokes signal observed with linear additive noise. SIAM J. Sci. Comput. **40**(3), A1544–A1565 (2018). https://doi.org/10.1137/17M1151900
35. Myötyri, E., Pulkkinen, U., Simola, K.: Application of stochastic filtering for lifetime prediction. Reliability Engineering and System Safety **91**(2), 200–208 (2005). https://doi.org/10.1016/j.ress.2005.01.002. Project code: G2SU00039
36. Pham, H., Warin, X.: Neural networks-based backward scheme for fully nonlinear pdes (2019)
37. Pham, H., Warin, X., Germain, M.: Neural networks-based backward scheme for fully nonlinear pdes (2021)
38. Raissi, M., Perdikaris, P., Karniadakis, G.E.: Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations (2017)
39. Sirignano, J., Spiliopoulos, K.: Dgm: A deep learning algorithm for solving partial differential equations. Journal of Computational Physics **375**, 1339–1364 (2018). https://doi.org/10.1016/j.jcp.2018.08.029