
SOLVING TIME DEPENDENT FOKKER-PLANCK EQUATIONS VIA TEMPORAL NORMALIZING FLOW

A PREPRINT

Xiaodong Feng ^{*}

Li Zeng [†]

Tao Zhou [‡]

March 23, 2022

ABSTRACT

In this work, we propose an adaptive learning approach based on temporal normalizing flows for solving time-dependent Fokker-Planck (TFP) equations. It is well known that solutions of such equations are probability density functions, and thus our approach relies on modelling the target solutions with the temporal normalizing flows. The temporal normalizing flow is then trained based on the TFP loss function, without requiring any labeled data. Being a machine learning scheme, the proposed approach is mesh-free and can be easily applied to high dimensional problems. We present a variety of test problems to show the effectiveness of the learning approach.

Keywords Temporal normalizing flow · Fokker-Planck equations · Adaptive density approximation

1 Introduction

The Fokker-Planck (FP) equations, which describe the time evolution of probability density functions (PDFs) of complex stochastic systems, have been widely used in different fields such as physical and biological modelling [31][34]. Solving the FP equations numerically has been an important research topic in the past few decades. Generally, there are two main ways to obtain the PDFs of stochastic dynamics: solving the FP equations directly, or evaluating the transition probability density of the associated stochastic differential equations (SDEs). Traditional numerical methods for doing this include the finite element methods [4], the finite difference methods [19], the path integral methods [37], to name just a few. One of the biggest difficulties of these approaches is that either discretization of a high dimensional (unbounded) physical space is needed, or a large number of sample paths via Monte Carlo method [12] should be used.

In recent years, machine learning techniques have been widely used to solve partial differential equations (PDEs), see e.g. [6, 32, 13] and references therein. Among others, we mention the deep Galerkin method [33], the deep Ritz method [7], and the so-called physics-informed neural networks (PINNs) [28]. These approaches have been widely applied to many realistic problems, such as fluid mechanics [29, 2], high dimensional PDEs (with applications in computational finance) [11, 41], uncertainty quantification [39, 27, 42, 14, 22], to name just a few. Meanwhile, generative models such as generative adversarial networks [9], variational autoencoder [17] and normalizing flow (NF) [24, 30], have also been successfully applied to learn forward and inverse PDEs [3, 43, 40, 20]. For instance, physics-informed generative adversarial model was proposed in [38] to tackle high dimensional stochastic differential equations. In [10], normalizing field flow was developed to build surrogate models for uncertainty quantification problems.

As the solution of the Fokker-Planck equation is a probability density function, solving this problem can also be considered as a density estimation problem. This motivates us to propose in this work an adaptive learning scheme

^{*}LSEC, Institute of Computational Mathematics and Scientific/Engineering Computing, AMSS, Chinese Academy of Sciences, Beijing, China. Email: xdfeng@lsec.cc.ac.cn.

[†]LSEC, Institute of Computational Mathematics and Scientific/Engineering Computing, AMSS, Chinese Academy of Sciences, Beijing, China. Email: zengli@lsec.cc.ac.cn.

[‡]LSEC, Institute of Computational Mathematics and Scientific/Engineering Computing, AMSS, Chinese Academy of Sciences, Beijing, China. Email: tzhou@lsec.cc.ac.cn. This work was partially supported by the national natural science foundation of China (Nos. 11731006, 11831010 and 11871068), the national key R&D program (No. 2020YFA0712000), and the national key basic research program (No. 2018YFA0703903).

based on the normalizing flow. More precisely, our approach relies on modelling the target solutions of the FP equations. Consequently, the temporal normalizing flow is trained based on the TFP loss function (the physics informed residual), without requiring any labeled data. We list in the following the main features and related works of our approach:

- Our approach is an extension of the previous interesting work [36] where only *steady state* FP equations are investigated. To address time dependent problems, we propose an adaptive density approximation scheme based on temporal normalizing flow.
- Being a machine learning scheme, the proposed approach is mesh-free and can be easily applied to high dimensional problems.
- Our approach is based on PDE-loss functions, and does not need sample paths generated from stochastic differential equations. This is different from previous works such as [21] where sample paths of the corresponding stochastic dynamics are used.
- We present a variety of test problems, including FP equations with linear or nonlinear drift terms and high dimensional problems, to show the effectiveness of the learning approach.

The remainder of this paper is structured as follows. In section 2, we provide with some preliminary results. Section 3 provides our adaptive density approximation scheme based on the temporal normalizing flow. In Section 4 we demonstrate the efficiency of our adaptive sampling approach with several numerical experiments. We then give some concluding remarks in Section 5.

2 Problem setup

The main aim of this work is to solve the time dependent FP equations. To this end, we first give a brief introduction to the FP equations.

2.1 Fokker-Planck equations

Consider the state variable \mathbf{X}_t modeled by the following stochastic differential equation

$$d\mathbf{X}_t = \boldsymbol{\mu}(\mathbf{X}_t, t) dt + \boldsymbol{\sigma}(\mathbf{X}_t, t) d\mathbf{W}_t, \quad (2.1)$$

where \mathbf{X}_t and $\boldsymbol{\mu}(\mathbf{X}_t, t)$ are d -dimensional random vectors, $\boldsymbol{\sigma}(\mathbf{X}_t, t)$ is a $d \times M$ matrix and \mathbf{W}_t is an M -dimensional standard Wiener process. The probability density $p(\mathbf{x}, t)$ for \mathbf{X}_t satisfies the so called time dependent Fokker-Planck (TFP) equation:

$$\frac{\partial p(\mathbf{x}, t)}{\partial t} = - \sum_{i=1}^d \frac{\partial}{\partial x_i} [\mu_i(\mathbf{x}, t) p(\mathbf{x}, t)] + \sum_{i=1}^d \sum_{j=1}^d \frac{\partial^2}{\partial x_i \partial x_j} [D_{ij}(\mathbf{x}, t) p(\mathbf{x}, t)], \quad (2.2)$$

where $\mathbf{x} = (x_1, \dots, x_d)$, $\boldsymbol{\mu}$ is the drift vector $\boldsymbol{\mu} = (\mu_1, \dots, \mu_d)$ and \mathbf{D} is the diffusion tensor $\mathbf{D} = \frac{1}{2} \boldsymbol{\sigma} \boldsymbol{\sigma}^T$, i.e.,

$$D_{ij}(\mathbf{x}, t) = \frac{1}{2} \sum_{k=1}^M \sigma_{ik}(\mathbf{x}, t) \sigma_{jk}(\mathbf{x}, t).$$

Sometimes, one may focus on the stationary solution of (2.2), i.e., the invariant measure independent of time,

$$\sum_{i=1}^d \frac{\partial}{\partial x_i} [\mu_i(\mathbf{x}) p(\mathbf{x})] + \sum_{i=1}^d \sum_{j=1}^d \frac{\partial^2}{\partial x_i \partial x_j} [D_{ij}(\mathbf{x}) p(\mathbf{x})] = 0. \quad (2.3)$$

In general, the solutions of the above TFP equations are defined in the unbounded domain with the following boundary condition:

$$p(\mathbf{x}) \rightarrow 0 \quad \text{as} \quad \|\mathbf{x}\|_2 \rightarrow \infty. \quad (2.4)$$

Furthermore, the solution, being a density function, should also satisfy the following extra constraint

$$\int_{\mathbb{R}^d} p(\mathbf{x}, t) d\mathbf{x} \equiv 1, \quad \text{and} \quad p(\mathbf{x}, t) \geq 0. \quad (2.5)$$

2.2 Normalizing flow and RealNVP

Notice that both the constraints (2.4) and (2.5) bring essential difficulties for mesh-dependent numerical schemes when solving the TFP equations. To this end, we introduce in this section the normalizing flow (NF), which serves as a potentially efficient tool for handling TFP equations.

Normalizing flow provides a way for constructing flexible probability distributions over continuous random variables. Specifically, suppose we want to approximate an unknown random variable $\mathbf{x} \in \mathbb{R}^d$ (we define $p_{\mathbf{X}}(\mathbf{x})$ as its density function). The NF model seeks to find an invertible mapping $f : \mathbf{x} \rightarrow \mathbf{z}$, between a simple reference variable $\mathbf{z} \in \mathbb{R}^d$ (with known probability distribution $p_{\mathbf{Z}}$, e.g., Gaussian) and the target variable \mathbf{x} , i.e., $\mathbf{x} = f^{-1}(\mathbf{z})$. Then the distribution of \mathbf{x} is given by

$$p_{\mathbf{X}}(\mathbf{x}) = p_{\mathbf{Z}}(f(\mathbf{x})) \left| \det \nabla_{\mathbf{x}} f(\mathbf{x}) \right|, \quad (2.6)$$

where $\nabla_{\mathbf{x}} f(\mathbf{x})$ is the Jacobian. Given observations of \mathbf{x} , the unknown invertible mapping can be learned via the maximum likelihood estimations.

In the NF model, a complex invertible mapping $f(\cdot)$ can be constructed by stacking a sequence of simple bijections, each of which is a shallow neural network, thus the overall mapping is a deep neural network. Namely, the mapping $f(\cdot)$ can be written in a composite form:

$$\mathbf{z} = f(\mathbf{x}) = f_{[L]} \circ f_{[L-1]} \circ \cdots \circ f_{[1]}(\mathbf{x}). \quad (2.7)$$

Its inverse and Jacobian determinants are given by

$$\mathbf{x} = f^{-1}(\mathbf{z}) = f_{[1]}^{-1} \circ \cdots \circ f_{[L-1]}^{-1} \circ f_{[L]}^{-1}(\mathbf{z}), \quad (2.8)$$

$$|\det \nabla_{\mathbf{x}} f(\cdot)| = \prod_{i=1}^L |\det \nabla_{\mathbf{x}_{[i-1]}} f_{[i]}(\cdot)|, \quad (2.9)$$

where $\mathbf{x}_{[i-1]}$ indicates the immediate variables with $\mathbf{x}_{[0]} = \mathbf{x}$, $\mathbf{x}_{[L]} = \mathbf{z}$. Since computing the Jacobian determinants of large matrices is generally computationally very expensive, the structure of function f should be carefully designed. A successful example is RealNVP [5]. Let $\mathbf{x}_{[i]} = (\mathbf{x}_{[i],1}, \mathbf{x}_{[i],2})$ be a partition with $\mathbf{x}_{[i],1} \in \mathbb{R}^m$ and $\mathbf{x}_{[i],2} \in \mathbb{R}^{d-m}$. RealNVP proposes to use the following affine transformation:

$$\begin{aligned} \mathbf{x}_{[i],1} &= \mathbf{x}_{[i-1],1}, \\ \mathbf{x}_{[i],2} &= \mathbf{x}_{[i-1],2} \odot \exp(\mathbf{s}_i(\mathbf{x}_{[i-1],1})) + \mathbf{b}_i(\mathbf{x}_{[i-1],1}), \end{aligned} \quad (2.10)$$

where $\mathbf{s}_i : \mathbb{R}^m \rightarrow \mathbb{R}^{d-m}$, $\mathbf{b}_i : \mathbb{R}^m \rightarrow \mathbb{R}^{d-m}$ are scaling and translation depending on $\mathbf{x}_{[i-1],1}$, and \odot is Hadamard product. By doing this, the Jacobian matrix $f(\mathbf{x})$ (for the vector case) is lower-triangular whose determinant can be evaluated efficiently. Furthermore, \mathbf{s}_i and \mathbf{b}_i can be modeled by complex neural networks to enhance the expression capacity of the invertible map.

The RealNVP model has been widely adopted recently [18, 16, 26]. Nevertheless, we mention here the interesting work [35] where a so-called KR-net is proposed to enhance the expressive power and improve the numerical stability over RealNVP.

3 Methodology

In this section, we shall propose our learning scheme for the TFP equations.

3.1 Temporal normalizing flow

Note that the NF model discussed in the above section is time-independent. To address time dependent FP equations, we first introduce the so-called temporal normalizing flow (TNF). The TNF was proposed by Both and Kusters [1] aiming at estimating time evolving distributions.

Let $\hat{\mathbf{x}}$ be a $d+1$ dimensional real vector including the temporal variable, namely, $\hat{\mathbf{x}} = (\mathbf{x}, t)$. In addition, $\hat{\mathbf{z}}$ is a $d+1$ dimensional latent variable, $\hat{\mathbf{z}} = (\mathbf{z}, t^*)$, obeying a simple distribution. Then, we consider the following transformation:

$$p_{\hat{\mathbf{X}}}(\hat{\mathbf{x}}) = p_{\hat{\mathbf{Z}}}(\hat{\mathbf{z}}) |\det J|, \quad \hat{\mathbf{z}} = f(\hat{\mathbf{x}}), \quad (3.1)$$

where J is Jacobian of $f(\hat{\mathbf{x}})$. As the conservation law for the temporal axis is not always correct, i.e. $\int p(\mathbf{x}, t) d\mathbf{x} dt \neq 1$, we cannot include it as an additional dimension in eq. (3.1). Notice that the determinant of the Jacobian is

$$\det J = \begin{vmatrix} \frac{\partial z}{\partial \mathbf{x}} & \frac{\partial z}{\partial t} \\ \frac{\partial t^*}{\partial \mathbf{x}} & \frac{\partial t^*}{\partial t} \end{vmatrix}. \quad (3.2)$$

where z is the latent spatial coordinate and t^* the latent temporal coordinate. By letting the latent time t^* be exactly equal to the real time t , one obtains

$$\det J = \begin{vmatrix} \frac{\partial z}{\partial \mathbf{x}} & \frac{\partial z}{\partial t} \\ 0 & 1 \end{vmatrix} = \left| \frac{\partial z(\mathbf{x}, t)}{\partial \mathbf{x}} \right|, \quad (3.3)$$

Consequently, the temporal normalizing flow can be written as

$$p_{\hat{\mathbf{X}}}(\mathbf{x}, t) = p_{\hat{\mathbf{Z}}}(z, t) \left| \frac{\partial z}{\partial \mathbf{x}} \right|, \quad z = f(\mathbf{x}, t). \quad (3.4)$$

Similar to (2.7), $f(\cdot, t)$ can be constructed by stacking a sequence of simple bijections. Namely, given time t , we have

$$z = f(\mathbf{x}, t) = f_{[L]} \circ f_{[L-1]} \circ \dots \circ f_{[1]}(\mathbf{x}, t). \quad (3.5)$$

The inverse of f is then given by

$$\mathbf{x} = f^{-1}(z, t) = f_{[1]}^{-1} \circ f_{[2]}^{-1} \circ \dots \circ f_{[L]}^{-1}(z, t). \quad (3.6)$$

Consequently, the corresponding Jacobian can be obtained by using the chain rule:

$$|\det \nabla_{\mathbf{x}} f(\cdot, t)| = \prod_{i=1}^L |\det \nabla_{\mathbf{x}_{[i-1]}} f_{[i]}(\cdot, t)|, \quad (3.7)$$

where $\mathbf{x}_{[i-1]}$ indicates the immediate variables with $\mathbf{x}_{[0]} = \mathbf{x}$, $\mathbf{x}_{[L]} = z$.

3.2 Architecture

In this section, we present the main architecture of our TNF. Inspired by the construction of KR-net [35], our TNF model can be regarded as a simplified extension of KR-net from spatial domain to temporal-spatial domain. Specifically, each $f_{[i]}$ in our TNF model consists of an Actnorm layer, followed by a modified time-dependent affine mapping [5]. For the last layer, we apply a polynomial spline transformation to increase the modelling power [23]. Figure 1 shows the flow chart of our proposed model, and we shall provide with more details in the following subsections.

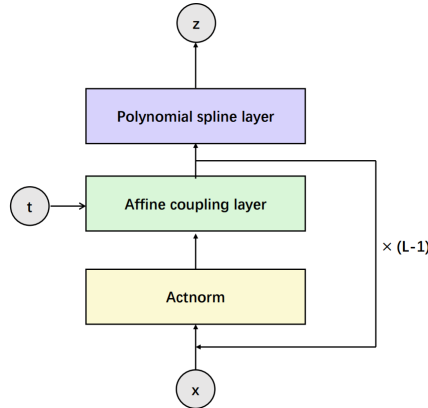


Figure 1: The architecture of our proposed model.

3.2.1 Actnorm: scale and bias layer

We adopt the Actnorm layer with data dependent initialization proposed by Kingma and Dhariwal [16]:

$$\mathbf{y}_{[i]} = \mathbf{a}_i \odot \mathbf{x}_{[i]} + \mathbf{b}_i, \quad (3.8)$$

where \mathbf{a}_i and \mathbf{b}_i are trainable parameters. The parameters can be initialized by the statistic mean and variance related to mini batch data. After initialization, the scale and bias are treated as regular trainable parameters that are independent of the data. The inverse can be easily obtained via

$$\mathbf{x}_{[i]} = (\mathbf{y}_{[i]} - \mathbf{b}_i) / \mathbf{a}_i, \quad (3.9)$$

where division here is operated on each corresponding component.

3.2.2 Affine coupling layer

Let $\mathbf{x}_{[i]} = (\mathbf{x}_{[i],1}, \mathbf{x}_{[i],2})$ be a partition with $\mathbf{x}_{[i],1} \in \mathbb{R}^m$ and $\mathbf{x}_{[i],2} \in \mathbb{R}^{d-m}$. Then we consider a time-dependent affine coupling layer $f_{[i]}(\cdot, t)$ as follows:

$$\begin{aligned} \mathbf{x}_{[i],1} &= \mathbf{x}_{[i-1],1}, \\ \mathbf{x}_{[i],2} &= \mathbf{x}_{[i-1],2} \odot (\mathbf{1}_{d-m} + \beta \tanh(\mathbf{s}_i(\mathbf{x}_{[i-1],1}, t))) + e^{\boldsymbol{\zeta}_i} \odot \tanh(\mathbf{q}_i(\mathbf{x}_{[i-1],1}, t)), \end{aligned} \quad (3.10)$$

where $|\beta| < 1$ is a user-specified parameter (e.g. 0.6), $\mathbf{1}_{d-m}$ denotes a $d - m$ dimensional vector whose components are all 1, $\mathbf{s}_i : \mathbb{R}^{m+1} \rightarrow \mathbb{R}^{d-m}$, and $\mathbf{q}_i : \mathbb{R}^{m+1} \rightarrow \mathbb{R}^{d-m}$ are scaling and translation depending on $\mathbf{x}_{[i-1],1}$ and t . While $\boldsymbol{\zeta}_i \in \mathbb{R}^{d-m}$ is a trainable variable which depends on data directly. Notice that the inverse can be easily computed via:

$$\begin{aligned} \mathbf{x}_{[i-1],1} &= \mathbf{x}_{[i],1}, \\ \mathbf{x}_{[i-1],2} &= (\mathbf{x}_{[i],2} - e^{\boldsymbol{\zeta}_i} \odot \tanh(\mathbf{q}_i(\mathbf{x}_{[i],1}, t))) \odot (\mathbf{1}_{d-m} + \beta \tanh(\mathbf{s}_i(\mathbf{x}_{[i],1}, t)))^{-1}. \end{aligned} \quad (3.11)$$

The Jacobian of $\mathbf{x}_{[i]}(\cdot, t)$ is given by

$$\nabla_{\mathbf{x}_{[i-1]}} \mathbf{x}_{[i]}(\cdot, t) = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \nabla_{\mathbf{x}_{[i-1],1}} \mathbf{x}_{[i],2} & \text{diag}(\mathbf{1}_{d-m} + \alpha \tanh(\mathbf{s}_i(\mathbf{x}_{[i-1],1}, t))) \end{bmatrix}. \quad (3.12)$$

Furthermore, we can model $\mathbf{s}_i, \mathbf{b}_i$ via neural networks

$$(\mathbf{s}_i, \mathbf{q}_i) = \text{NN}_{[i]}(\mathbf{x}_{[i-1]}, t). \quad (3.13)$$

Note that $f_{[i]}(\cdot, t)$ only changes $\mathbf{x}_{[i-1],2}$, hence we can exchange the positions of $\mathbf{x}_{[i],1}$ and $\mathbf{x}_{[i],2}$ to ensure that each component of \mathbf{x} can be updated.

3.2.3 Polynomial spline layer

For the last layer, we use the polynomial spline layer proposed by [36]. Without loss of generality, we present the formula below for the case of $d = 1$. While for higher dimensional cases one simply uses the tensor product arguments. The associated transformation is given by

$$G(x) = \begin{cases} \gamma(x + c) - c, & x \in (-\infty, -c) \\ 2c\hat{G}(\frac{x+c}{2c}) - c, & x \in (-c, c) \\ \gamma(x - c) + c, & x \in (c, \infty) \end{cases} \quad (3.14)$$

where $\gamma > 0$ is a used-specified small parameter (e.g. 10^{-6}), c is a prior constant describing the range of nonlinear transformation, and \hat{G} is a continuous piecewise linear cumulative probability function. Specifically, let $0 = l_0 < l_1 < \dots < l_{m-1} < l_m = 1$ be a given partition in the unit interval and $\{k_j\}_{j=0}^m$ be the corresponding weights satisfying $\sum_j k_j = 1$. A piecewise linear polynomial can be defined as follows:

$$p(x) = \frac{k_{j+1} - k_j}{l_{j+1} - l_j}(x - l_j) + k_j, \quad \forall x \in [l_j, l_{j+1}], \quad \forall t. \quad (3.15)$$

Then the corresponding cumulative probability function \hat{G} admits the following formulation:

$$\hat{G}(x) = \frac{k_{j+1} - k_j}{2(l_{j+1} - l_j)}(x - l_j)^2 + k_j(x - l_j) + \sum_{i=0}^{j-1} \frac{k_i + k_{i+1}}{2}(l_{i+1} - l_i), \quad \forall x \in [l_j, l_{j+1}], \quad \forall t. \quad (3.16)$$

For the sake of the continuity of $p(x)$ at $x = 0$ and $x = 1$, one can set $k_0 = k_m = \gamma$. Furthermore, we can model $\{k_j\}_{j=1}^{m-1}$ as:

$$k_j = \frac{\exp(\tilde{k}_j)}{\sum_{i=0}^m \exp(\tilde{k}_i)}, \quad j = 1, \dots, m-1, \quad (3.17)$$

where $\{\tilde{k}_j\}$ are trainable parameters. Notice that the polynomial spline layer (3.14)-(3.16) yields explicit monotonous expressions, and its inverse can be readily computed.

3.3 Solving time dependent FP equations via temporal normalizing flow

We are now ready to present our scheme for solving time-dependent FP equations (2.1-2.2) with temporal normalizing flow. Specifically, consider the following TFP equations:

$$\begin{cases} p_t + \mathcal{N}_x[p] = 0, & \mathbf{x} \in \Omega, \quad t \in (0, T], \\ p(\mathbf{x}, 0) = p_0(\mathbf{x}), & \mathbf{x} \in \Omega, \\ p(\mathbf{x}, 0) \rightarrow 0 \text{ as } \|\mathbf{x}\| \rightarrow 0, \\ \int_{\Omega} p(\mathbf{x}, t) d\mathbf{x} = 1, \quad p(\mathbf{x}, t) \geq 0, & \mathbf{x} \in \Omega, \quad t \in [0, T], \end{cases} \quad (3.18)$$

where \mathcal{N}_x denotes a differential operator with respect to \mathbf{x} and $\Omega = \mathbb{R}^d$. In addition, $p : \Omega \times [0, T] \rightarrow \mathbb{R}^+ \cup \{0\}$ denotes the unknown latent quantity of interest.

We proceed by approximating $p(\mathbf{x}, t)$ via a temporal normalizing flow $p_{\theta}(\mathbf{x}, t)$, where θ denotes all trainable parameters of the model, and the prior of \mathbf{x} is assumed to be Gaussian. Then we adopt the physical law (i.e., the TFP equation) to yield the following residual

$$\mathbf{r}(\mathbf{x}, t) := \frac{\partial}{\partial t} p_{\theta}(\mathbf{x}, t) + \mathcal{N}_x[p_{\theta}(\mathbf{x}, t)], \quad (3.19)$$

where the partial derivatives with respect to time and space coordinates can be readily computed using automatic differentiation. Notice that, unlike traditional mesh-dependent approaches (such as the finite element/difference methods), the non-negativity and conservation constraints naturally hold for $p_{\theta}(\cdot, t)$. To this end, we propose the following loss function for training the parameter θ :

$$\mathcal{L}_{\text{pde}}(\theta) = \lambda_r \mathcal{L}_r(\theta) + \lambda_{\text{ic}} \mathcal{L}_{\text{ic}}(\theta), \quad (3.20)$$

where \mathcal{L}_r and \mathcal{L}_{ic} are the equation loss and the initial condition loss, respectively. More precisely, we have

$$\mathcal{L}_r(\theta) = \frac{1}{N_r} \sum_{i=1}^{N_r} |\mathbf{r}_{\theta}(\mathbf{x}_r^i, t_r^i)|^2, \quad \mathcal{L}_{\text{ic}}(\theta) = \frac{1}{N_{\text{ic}}} \sum_{i=1}^{N_{\text{ic}}} |p_{\theta}(\mathbf{x}_{\text{ic}}^i, 0) - p_0(\mathbf{x}_{\text{ic}}^i)|^2. \quad (3.21)$$

Here, N_r, N_{ic} denote the batch sizes of training data $\{(\mathbf{x}_r^i, t_r^i)\}_{i=1}^{N_r}, \{\mathbf{x}_{\text{ic}}^i, p_0(\mathbf{x}_{\text{ic}}^i)\}_{i=1}^{N_{\text{ic}}}$, respectively. $\{\lambda_r, \lambda_{\text{ic}}\}$ are weight parameters.

Notice that for each t , $p(\mathbf{x}, t)$ is defined on the whole domain \mathbb{R}^d . However, in practice $p(\mathbf{x}, t)$ is generally concentrated in a small (yet unknown) region. Moreover, the regularity of solution $p(\mathbf{x}, t)$ may also vary in the computation domain. Consequently, how to efficiently choose the training points becomes a core issue. Obviously, a uniform sampling approach in a possibly large domain is not practical. We thus need to develop adaptive sampling strategies. In the TNF framework, this becomes possible, and we propose the following strategy:

1. Initialization: generate an initial training set with samples uniformly distributed in a given domain:

$$C_0 = \{\mathbf{x}_i^r, t_i^r\}_{i=1}^{N_r} \subset D_0.$$

2. Train the temporal normalizing flow by minimizing the loss function (3.20) with training data C_0 to obtain a new probability distribution $p_1(\mathbf{x}, t; \theta)$.
3. Generate samples with $p_1(\cdot, t_i^r; \theta)$ to get a new training set $C_1 = \{\tilde{\mathbf{x}}_i^r, t_i^r\}_{i=1}^{N_r}$, and set $C_0 = C_1$. Notice that for each t_i^r , $\tilde{\mathbf{x}}_i^r$ can be obtained by transforming the prior Gaussian samples via the inverse temporal normalizing flow.
4. Repeat steps 2-3 for N_{adaptive} times to get a convergent approximation.

More details for the sampling procedure are given in Algorithm 1.

Algorithm 1 Temporal normalizing flow for time-dependent FP equations

Input: maximum epoch number N_e , maximum iteration number N_{adaptive} , hyper parameter $\alpha, \lambda_r, \lambda_{\text{ic}}$, initial training data $C_r = \{(\mathbf{x}_r^i, t_r^i)\}_{i=1}^{N_r}$, $C_{\text{ic}} = \{\mathbf{x}_{\text{ic}}^i\}_{i=1}^{N_{\text{ic}}}$, $C_T = \{t_r^i\}_{i=1}^{N_r} \cup \{0\}_{i=1}^{N_{\text{ic}}}$, tolerance ϵ_1, ϵ_2 ; $L_{\text{old}} = 0$;
for $k = 1, \dots, N_{\text{adaptive}}$ **do**
 for $j = 1, \dots, N_e$ **do**
 Divide $C = \{C_r, C_{\text{ic}}\}$ into m batch $\{C^{ib}\}_{ib=1}^m$ randomly;
 for $ib = 1, \dots, m$ **do**
 Compute the loss function (3.20) L_{pde}^{ib} for mini-batch data C^{ib} ;
 Update θ by using Adam optimizer;
 $L_{\text{new}} = \frac{1}{m} \sum_{ib=1}^m L_{\text{pde}}^{ib}$;
 if $L_{\text{new}} < \epsilon_1$ or $|L_{\text{old}} - L_{\text{new}}| < \epsilon_2$ **then**
 Break;
 else
 $L_{\text{old}} = L_{\text{new}}$;
 $N_e = \alpha * N_e$;
 Sample from $p(\cdot, t; \theta)$ for $t \in C_T$ and update training set C ;
 Output: The predicted solution $p(\mathbf{x}, t; \theta)$.

4 Numerical results

In this section, we present a series of comprehensive numerical tests to demonstrate the effectiveness of the proposed algorithm. To quantitatively evaluate the accuracy of the numerical solution p_θ , we shall consider both the relative L^2 error $\|p^* - p_\theta\|_2 / \|p^*\|_2$ and the relative KL divergence given by

$$\frac{D_{\text{KL}}(p^*(t) \| p_\theta(t))}{H(p^*(t))} = \frac{\mathbb{E}_{p^*(t)} \log(p^*(t) / p_\theta(t))}{-\mathbb{E}_{p^*(t)} \log p^*(t)},$$

where \mathbb{E} denotes the expectation. We approximate the above advocated relative KL divergence by Monte Carlo integration, namely,

$$\frac{D_{\text{KL}}(p^*(t) \| p_\theta(t))}{H(p^*(t))} \approx \frac{1}{N_v} \frac{\sum_{i=1}^{N_v} (\log(p^*(\mathbf{x}_i; t) - \log p_\theta(\mathbf{x}_i; t)))}{-\sum_{i=1}^{N_v} \log p^*(\mathbf{x}_i; t)}.$$

Here p^* is the reference/exact solution, and $\mathbf{x}_i(t)$ is drawn from the ground truth $p^*(\mathbf{x}; t)$ and the amount of validation data is set to $N_v = 10^6$ for each t .

We shall employ hyperbolic tangent function (Tanh) as the activation function. For each i , $\text{NN}_{[i]}$ (see 3.13) is a feed forward neural network with two hidden layers and 32 neurons. We use a half-half partition $\mathbf{x}_{[i]} = (\mathbf{x}_{[i],1}, \mathbf{x}_{[i],2})$, $\mathbf{x}_{[i],1} \in \mathbb{R}^{\lfloor d/2 \rfloor}$, $\mathbf{x}_{[i],2} \in \mathbb{R}^{d - \lfloor d/2 \rfloor}$. We initialize all trainable parameters using Glorot initialization [8]. For the training procedure, we use the Adam optimizer [15] with an initial learning rate 0.001. All training data have the shape $N_x \times N_t$, where N_x is the number of spatial sample points and N_t is the temporal sample points. All numerical tests are implemented with Pytorch.

4.1 A toy example

As our first example, we start with a benchmark example to illustrate how the adaptive algorithm works. We consider a 2D TFP equation of the form

$$\begin{aligned} \frac{\partial p}{\partial t} - \frac{1}{2} \Delta p &= 0, & t \in (0, 1], \mathbf{x} \in \mathbb{R}^2, \\ p(\mathbf{x}, 0) &= \frac{1}{2\pi} \exp\left(-\frac{1}{2} \|\mathbf{x} - 4 \cdot \mathbf{1}_2\|^2\right), & \mathbf{x} \in \mathbb{R}^2, \end{aligned} \quad (4.1)$$

and the exact solution yields

$$p(\mathbf{x}, t) = \frac{1}{2\pi(t+1)} \exp\left(-\frac{\|\mathbf{x} - 4 \cdot \mathbf{1}_2\|^2}{2(t+1)}\right). \quad (4.2)$$

The number of epochs is chosen as $N_e = 20$, $\alpha = 2$, batch size is set to 1000, and five adaptivity iterations are conducted for this problem, i.e., $N_{\text{adaptive}} = 5$. We take $L = 6$ affine coupling layers and actnorm layers. In addition,

we turn off the polynomial spline layer. The initial spatial training set is generated via the uniform distributed points in $[-3, 3]^2$ (which only contains partial information of the exact solution), and the sample size is $N_x = 1000$ for each moment. The temporal training set is uniformly sampled in the unit interval $[0, 1]$ with size $N_t = 20$ which results in total $1000 \times 20 = 20,000$ training points for each iteration step for $k = 1, \dots, N_{\text{adaptive}}$.

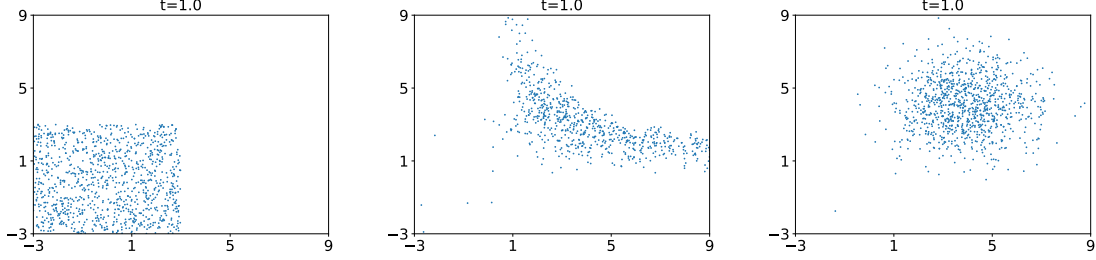


Figure 2: A toy example. Training samples at different adaptive iterations. Left panel: $N_{\text{adaptive}} = 1$. Middle panel: $N_{\text{adaptive}} = 2$. Right panel: $N_{\text{adaptive}} = 4$.

The training points for different adaptive iterations at time $t = 1$ are presented in Figure 2. One can clearly observe that, the training points become increasingly closer to the ground truth as adaptive iterations increase, which shows that the adaptive sampling scheme is rather effective. The predicted solution and the exact solution are presented in Figure 3 and these figures indicate that the predicted solution yields an excellent agreement with the exact solution. The relative L^2 error and relative KL divergence with different adaptive iterations are also provided in Figure 4.

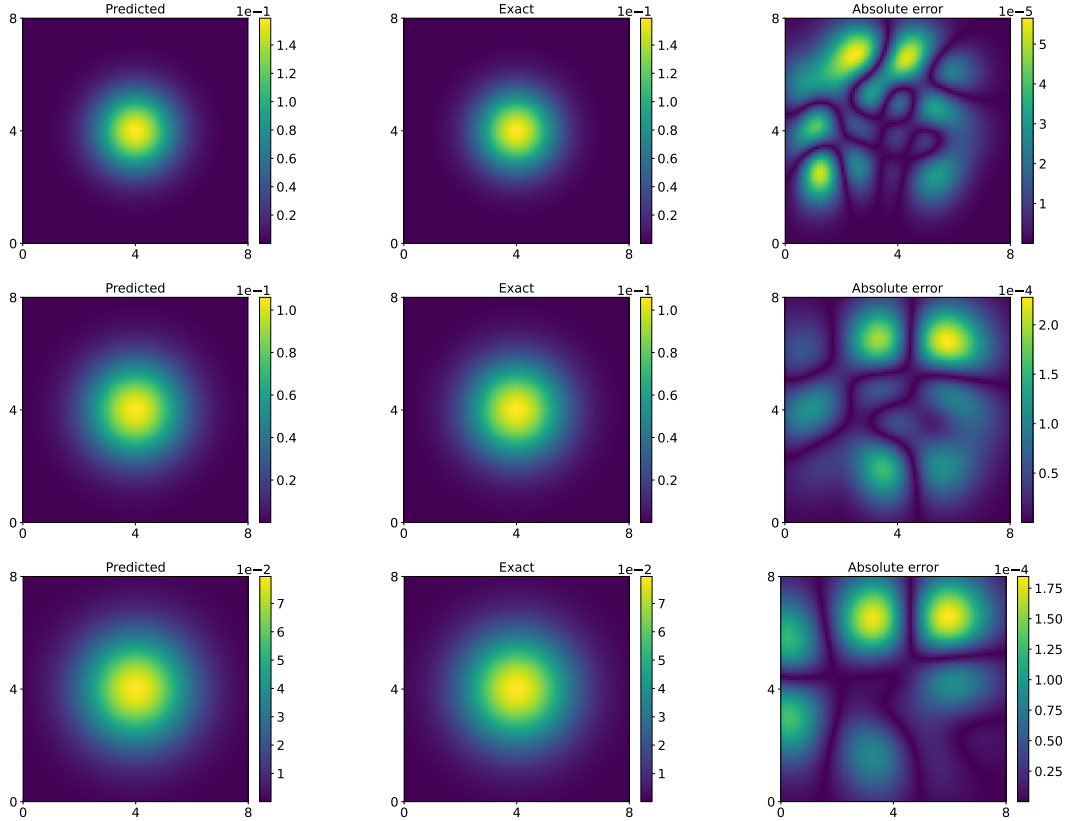


Figure 3: A toy example. Predicted solution versus the reference solution for different time t . Top row: $t = 0$. Middle row: $t = 0.5$. Bottom row: $t = 1$.

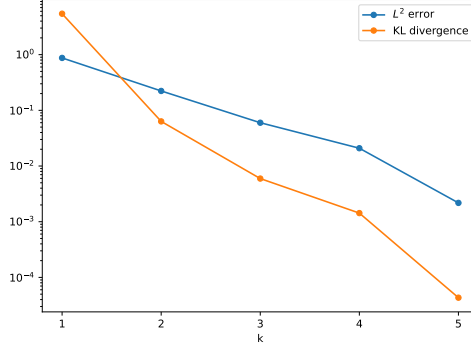


Figure 4: A toy example. Relative L^2 error and relative KL divergence at $t = 1$ for different adaptive iterations k .

4.2 Linear oscillator

We next consider the TFP equation (2.2) with

$$\boldsymbol{\mu} = (x_2, -0.2x_2 - x_1), \quad \mathbf{D} = \text{diag}(0, 0.2). \quad (4.3)$$

The initial condition is given by

$$p(\mathbf{x}, 0) = \mathcal{N}\left((1, 1), \frac{1}{9}\mathbf{I}_2\right), \quad (4.4)$$

where \mathcal{N} denotes Gaussian distribution. We consider that the time interval is $[0, 3]$ and the "exact" solution of this example is computed by the ADI scheme [25] in a truncated spatial domain $[-5, 5]^2$ with mesh size $\delta h = 0.01$ and $\delta t = 0.01$. For our approach, we use $N_e = 60$ and $\alpha = 1$. For now, 1000 batch size and four adaptivity iterations are conducted for this problem, i.e., $N_{\text{adaptive}} = 4$. We take $L = 8$ affine coupling layers and actnorm layers, and turn off the polynomial spline layer. The initial spatial training set is generated through the uniform distribution with a range $[-5, 5]^2$, and the corresponding initial temporal training set is uniformly sampled in the interval $[0, 3]$, which results in total $2000 \times 100 = 200,000$ training points.

The comparison between the numerical solution and the ground truth is presented in Figure 5. We again observe a good agreement between the predicted and the exact solutions. Moreover, Figure 6 shows the behaviors of the relative L^2 error for different adaptive iterations N_{adaptive} , which suggests that a large number of iterations admit great help for improving the convergence.

4.3 Nonlinear oscillator

We now consider the TFP equation with a nonlinear drift term:

$$\boldsymbol{\mu} = (x_2, x_1 - 0.4x_2 - 0.1x_1^3), \quad \mathbf{D} = \text{diag}(0, 0.4). \quad (4.5)$$

The initial distribution is given by

$$p(\mathbf{x}, 0) = \mathcal{N}((0, 5), \mathbf{I}_2). \quad (4.6)$$

We solve this problem in time interval $[0, 3]$, and again we compute the reference solution by the ADI scheme in a truncated spatial domain $[-10, 10]^2$, with mesh size $\delta h = 0.01$ and $\delta t = 0.005$. For our learning scheme, the parameters are chosen as $N_e = 50$, $\alpha = 1.5$. We use five adaptivity iterations, i.e., $N_{\text{adaptive}} = 5$, and set batch size to 10000. We use $L = 4$ affine coupling layers and actnorm layers, and turn on the polynomial spline layer with 50 nodes. The initial spatial training set is generated through a uniform distribution in a range $[-10, 10]^2$ with sample size 5000, and the corresponding initial temporal training set is chosen from a nonuniform partition (with 100 equidistant nodes in the $[0, 1.5]$ and 200 equidistant nodes in the $[1.5, 3]$), which results in total $5000 \times 300 = 1,500,000$ training points. As shown in Figure 7, a good agreement can be achieved between the predictions and the exact solutions.

We also present the relative L^2 errors and relative KL divergences with different adaptive iterations in Figure 8. Again, we can see that more iterations provide great help for improving the convergence. However, similar to the last example, we also observe that the numerical error seems to increase as time evolves. Nevertheless, we shall show in the next examples that this can be improved by increasing the training points and iterations as time evolves.

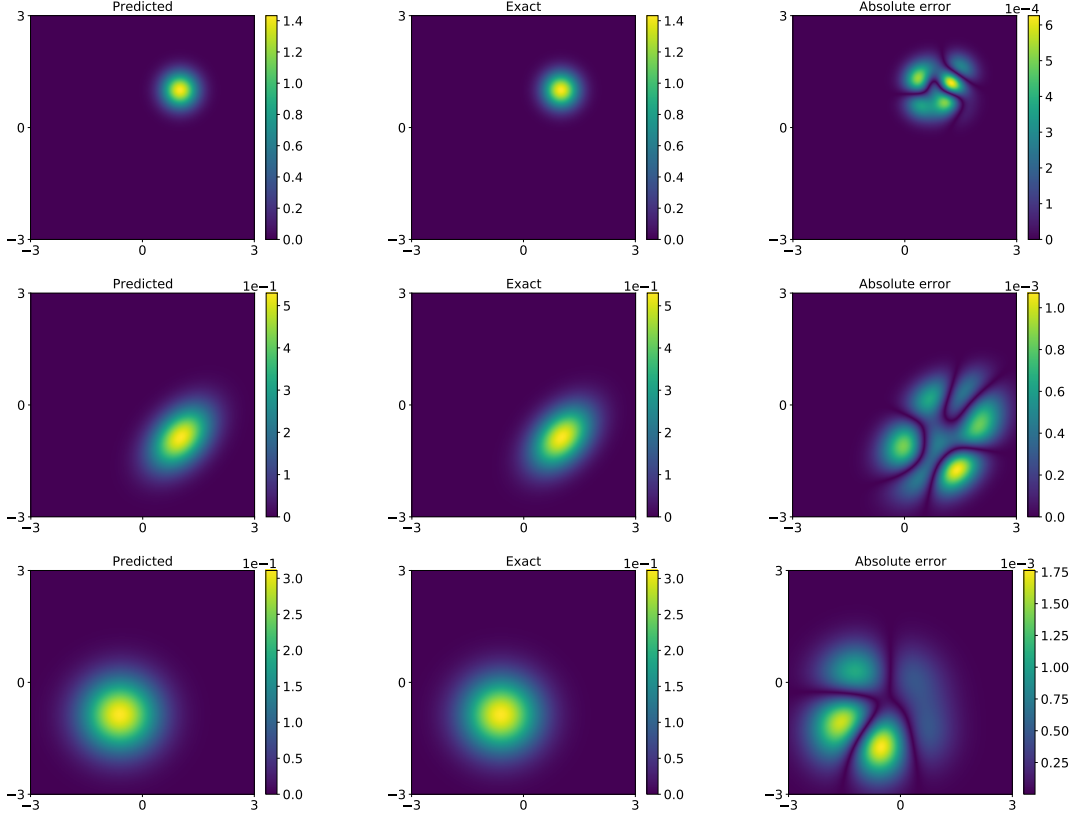


Figure 5: Linear oscillator. Predicted solution versus the reference solution for different time t . Top row: $t = 0$. Middle row: $t = 1.5$. Bottom row: $t = 3$.

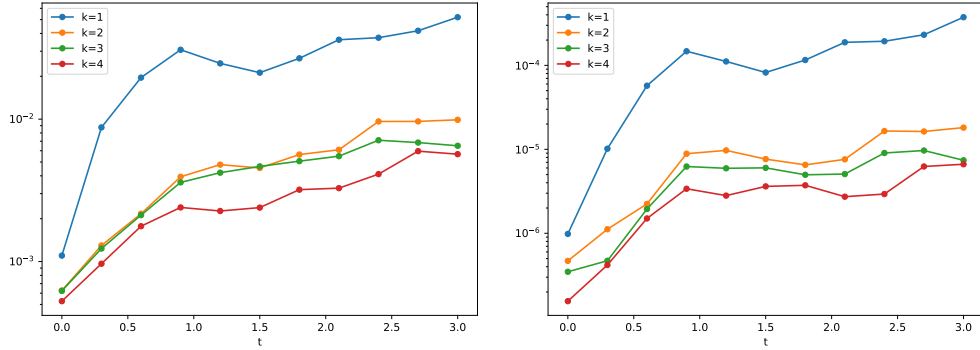


Figure 6: Linear oscillator. Relative L^2 error and relative KL divergence at different adaptive iterations k . Left panel: relative L^2 error. Right panel: relative KL divergence.

4.4 High dimensional TFP equations

Finally, we consider a relatively high dimensional TFP equation:

$$\begin{aligned} \frac{\partial p}{\partial t} - \frac{1}{2} \Delta p + 2\nabla \cdot p &= 0, \\ p(\mathbf{x}, 0) &= (2\pi)^{-d/2} \exp(-\|\mathbf{x}\|^2/2). \end{aligned} \quad (4.7)$$

Here $\mathbf{x} \in \mathbb{R}^d$ and $t \in [0, 1]$. The corresponding exact solution is given by

$$p(\mathbf{x}, t) = \frac{1}{(2\pi(t+1))^{d/2}} \exp\left(-\frac{\|\mathbf{x} - 2t \cdot \mathbf{1}_d\|^2}{2(t+1)}\right). \quad (4.8)$$

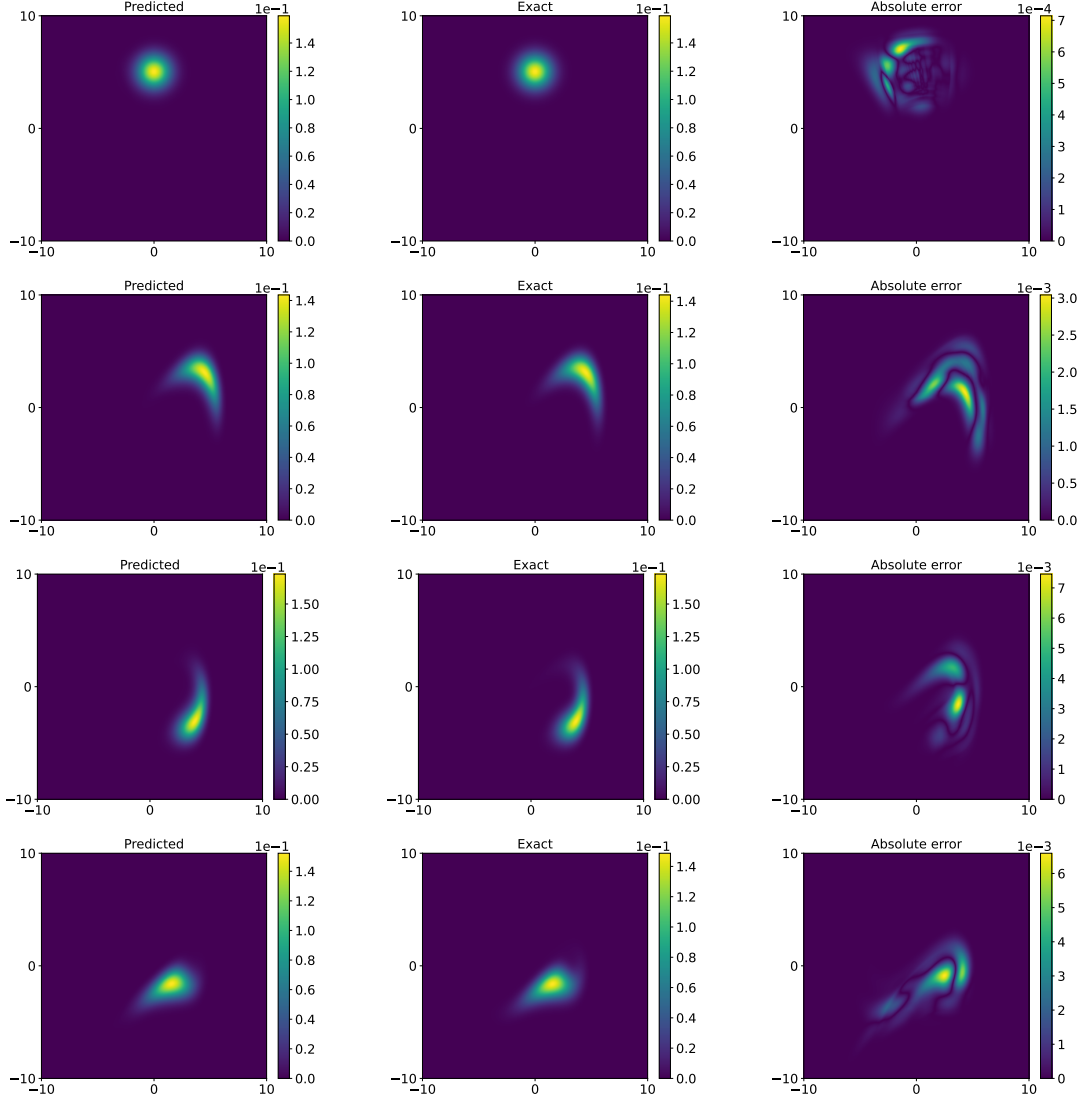


Figure 7: Nonlinear oscillator. Predicted solution versus the reference solution for different time t . Top row: $t = 0$. Second row: $t = 1$. Third row: $t = 2$. Bottom row: $t = 3$.

We shall test this problem for the cases of $d = 4$ and $d = 8$. For $d = 4$, we set $N_e = 100$, $\alpha = 2$, $N_{\text{adaptive}} = 2$, batch size is 10000. We take $L = 8$ affine coupling layers and actnorm layers, and we turn off the polynomial spline layer. The initial spatial training set is generated through a uniform distribution with a range $[-3, 3]^4$, and the corresponding initial temporal training set is uniformly sampled in the unit interval $[0, 1]$, which results in total $10000 \times 50 = 500,000$ training points. The comparison between the prediction and the ground truth is shown in Figure 9. We can observe a good agreement between the predicted and exact solutions.

To test the effectiveness of the adaptivity procedure, we present the relative L^2 errors and relative KL divergences with $k = 1$ and $k = 2$ in Figure 10. Notice that a clear (linear) increase of the numerical error over time is observed. To alleviate this, we next consider a nonuniform sampling procedure for both the time domain and physical domain. More precisely, the time interval $[0, 1]$ is divided into n parts $\{t_1, \dots, t_n\}$:

$$t_i = 1 - \frac{r^{n-i} + 1}{r^n + 1}, \quad i = 1, \dots, n. \quad (4.9)$$

where $r = 1.05$ and $n = 100$. For fixed t , the number of spatial sample points $N_{\mathbf{x}}$ is decided by

$$N_{\mathbf{x}}(t_i) = N_0(1 + \lfloor (i-1)/20 \rfloor) \quad (4.10)$$

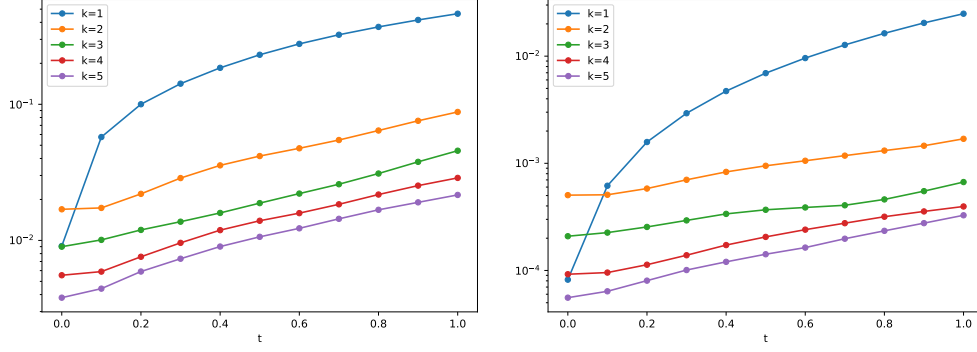


Figure 8: Nonlinear oscillator. Relative L^2 error and relative KL divergence at different adaptive iterations k . Left panel: relative L^2 error. Right panel: relative KL divergence.

where $N_0 = 5000$. Such a sampling procedure results in total $\sum_i^n N_{\mathbf{x}}(t_i) = 1.5 \times 10^6$ training points. Then, we repeat the computation and the numerical result is presented in Figure 11, it is clearly shown that the numerical errors are well controlled over time (compared to Figure 10).

For $d = 8$, we set $N_e = 100$, $\alpha = 2$, $N_{\text{adaptive}} = 3$, and batch size is 10000. We take $L = 10$ affine coupling layers and actnorm layers, and we turn off the polynomial spline layer. The initial spatial training set is generated through a uniform distribution with a range $[-5, 5]^8$, and the corresponding initial temporal training set is uniformly sampled in the unit interval $[0, 1]$, and thus the total sample size is equal to $20000 \times 25 = 500,000$. The predicted solution and the exact solution for $t = 0.2$ and $t = 0.4$ are presented in Figure 12. Furthermore, the relative errors with different adaptive iterations are shown in Figure 13. Again, we may improve the convergence by adding more training points as time evolves; however, for high-dimensional problems this will introduce a huge computational cost.

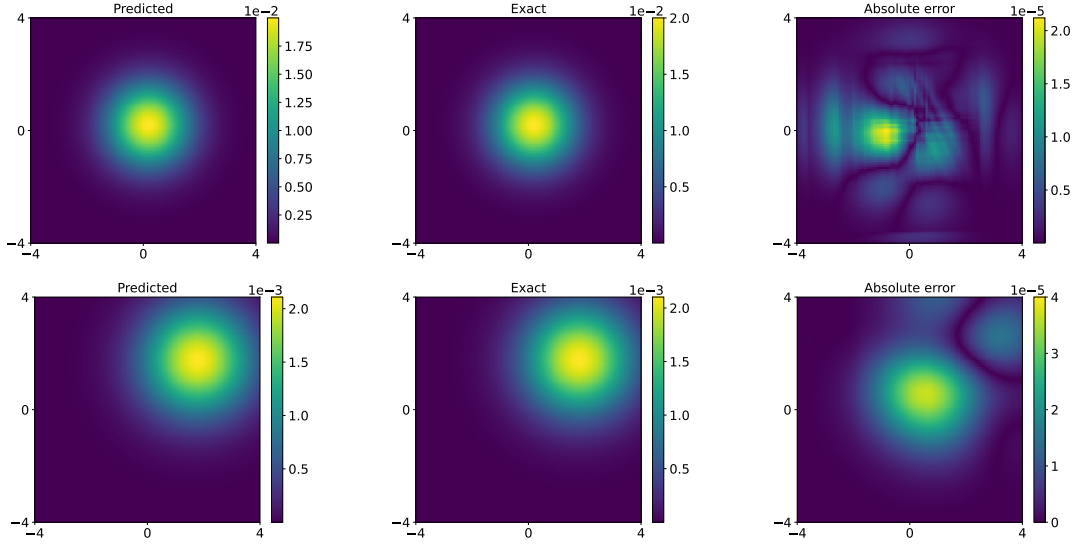


Figure 9: 4D example. Predicted solution versus the reference solution for different time t . Top row: $x_1 = 0.1, x_4 = 0.5, t = 0.1$. Bottom row: $x_1 = 0.1, x_4 = 0.5, t = 0.9$.

5 Conclusions

We have proposed to use the temporal normalizing flows for solving time-dependent Fokker-Planck (TFP) equations. Our approach relies on modelling the target solution by temporal normalizing flows. The temporal normalizing flow is then trained based on the TFP loss function, without requiring any labeled data. Being a machine learning scheme, the

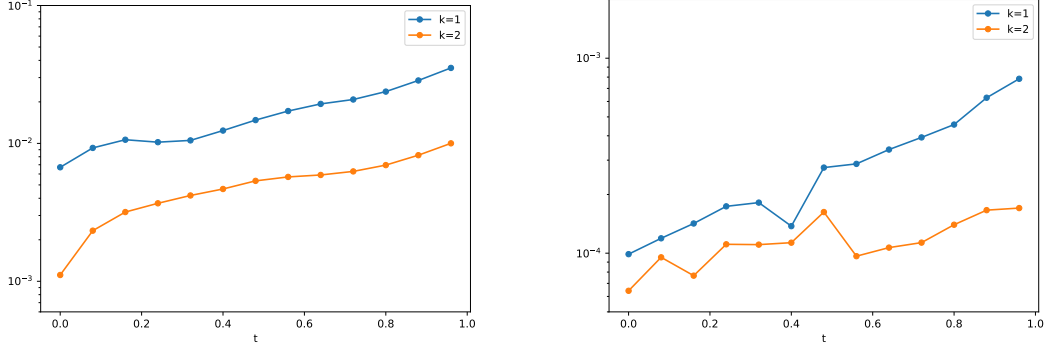


Figure 10: 4D example. Relative L^2 error and relative KL divergence for different adaptive iterations k . Left panel: Relative L^2 error. Right panel: relative KL divergence.

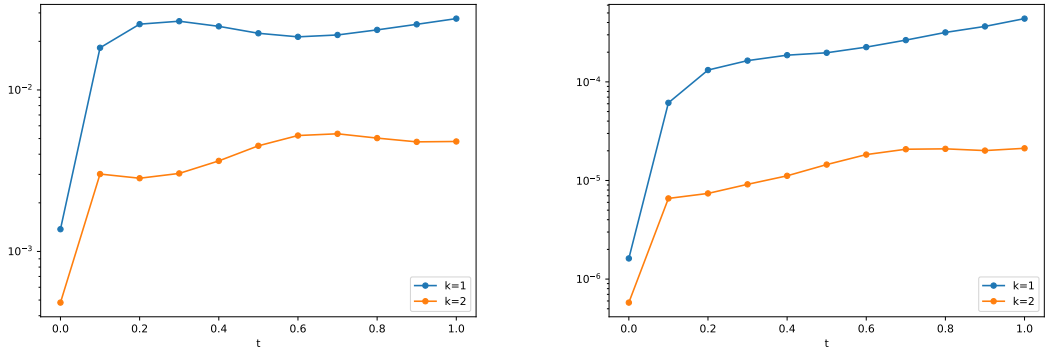


Figure 11: 4D example. Relative L^2 error and relative KL divergence with nonuniform time partition. Left panel: Relative L^2 error. Right panel: relative KL divergence.

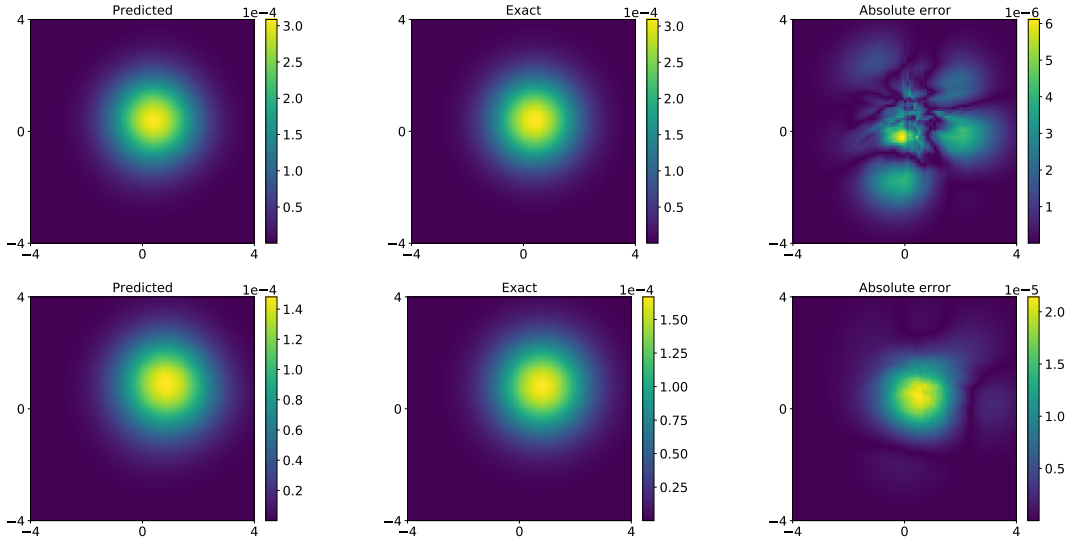


Figure 12: 8D example. Predicted solution versus the reference solution projected for different time t . Top row: $x_2 = \dots = x_7 = 0.4, t = 0.2$. Bottom row: $x_2 = \dots = x_7 = 0.8, t = 0.4$.

proposed approach is mesh-free and can be easily applied to high dimensional problems. We present a variety of test problems to show the effectiveness of the learning approach. There are, however, many important issues need to be addressed. From a theoretical point of view, a rigorous stability and accuracy analysis is still missing. From a practical

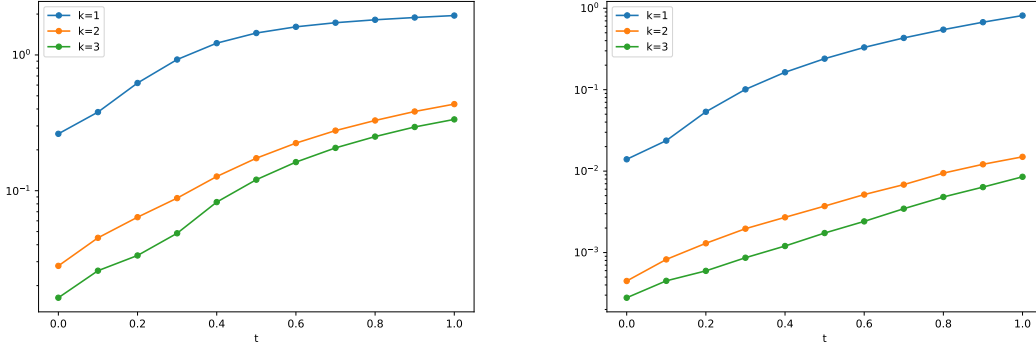


Figure 13: 8D example. Relative L^2 error versus relative KL divergence for different adaptive iterations k . Left panel: Relative L^2 error. Right panel: relative KL divergence.

viewpoint, for high dimensional problems, one needs to carefully design the training sets to balance the computational cost and the approximation error (which seems to increase as time evolves). Another possible way to tackle this issue is to include more temporal information in the reference process or in the network architecture.

References

- [1] Gert-Jan Both and Remy Kusters. Temporal normalizing flows. *arXiv preprint arXiv:1912.09092*, 2019.
- [2] Steven L Brunton, Bernd R Noack, and Petros Koumoutsakos. Machine learning for fluid mechanics. *Annual Review of Fluid Mechanics*, 52:477–508, 2020.
- [3] Xiaoli Chen, Liu Yang, Jinqiao Duan, and George Em Karniadakis. Solving inverse stochastic problems from discrete particle observations using the fokker–planck equation and physics-informed neural networks. *SIAM Journal on Scientific Computing*, 43(3):B811–B830, 2021.
- [4] Weihua Deng. Finite element method for the space and time fractional fokker–planck equation. *SIAM journal on numerical analysis*, 47(1):204–226, 2009.
- [5] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.
- [6] Weinan E. Machine learning and computational mathematics. *arXiv preprint arXiv:2009.14596*, 2020.
- [7] Weinan E and Bing Yu. The deep ritz method: A deep learning-based numerical algorithm for solving variational problems. *Communications in Mathematics and Statistics*, 6(1), 2018.
- [8] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.
- [9] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.
- [10] Ling Guo, Hao Wu, and Tao Zhou. Normalizing field flows: Solving forward and inverse stochastic differential equations using physics-informed flow models. *arXiv preprint arXiv:2108.12956*, 2021.
- [11] 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.
- [12] Eero Hirvijoki, Taina Kurki-Suonio, Simppa Äkäslompolo, Jari Varje, Tuomas Koskela, and Juho Miettunen. Monte carlo method and high performance computing for solving fokker–planck equation of minority plasma particles. *Journal of Plasma Physics*, 81(3), 2015.
- [13] Jianguo Huang, Haoqin Wang, and Tao Zhou. An augmented lagrangian deep learning method for variational problems with essential boundary conditions. *arXiv preprint arXiv:2106.14348*, 2021.
- [14] Raban Iten, Tony Metger, Henrik Wilming, Lidia Del Rio, and Renato Renner. Discovering physical concepts with neural networks. *Physical review letters*, 124(1):010508, 2020.
- [15] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

- [16] Diederik P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. *arXiv preprint arXiv:1807.03039*, 2018.
- [17] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.
- [18] Durk P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved variational inference with inverse autoregressive flow. *Advances in neural information processing systems*, 29:4743–4751, 2016.
- [19] Pankaj Kumar and S Narayanan. Solution of fokker-planck equation by finite element and finite difference methods for nonlinear systems. *Sadhana*, 31(4):445–461, 2006.
- [20] Shu Liu, Wuchen Li, Hongyuan Zha, and Haomin Zhou. Neural parametric fokker-planck equations. *arXiv preprint arXiv:2002.11309*, 2020.
- [21] Yubin Lu, Romit Maulik, Ting Gao, Felix Dietrich, Ioannis G Kevrekidis, and Jinqiao Duan. Learning the temporal evolution of multivariate densities via normalizing flows. *arXiv preprint arXiv:2107.13735*, 2021.
- [22] Xuhui Meng and George Em Karniadakis. A composite neural network that learns from multi-fidelity data: Application to function approximation and inverse pde problems. *Journal of Computational Physics*, 401:109020, 2020.
- [23] Thomas Müller, Brian McWilliams, Fabrice Rousselle, Markus Gross, and Jan Novák. Neural importance sampling. *ACM Transactions on Graphics (TOG)*, 38(5):1–19, 2019.
- [24] George Papamakarios, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. Normalizing flows for probabilistic modeling and inference. *arXiv preprint arXiv:1912.02762*, 2019.
- [25] Lukas Pichler, Arif Masud, and Lawrence A Bergman. Numerical solution of the fokker-planck equation by finite difference and finite element methods—a comparative study. In *Computational Methods in Stochastic Dynamics*, pages 69–85. Springer, 2013.
- [26] Ryan Prenger, Rafael Valle, and Bryan Catanzaro. Waveglow: A flow-based generative network for speech synthesis. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3617–3621. IEEE, 2019.
- [27] Tong Qin, Zhen Chen, John D Jakeman, and Dongbin Xiu. Deep learning of parameterized equations with applications to uncertainty quantification. *International Journal for Uncertainty Quantification*, 11(2), 2021.
- [28] 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.
- [29] Maziar Raissi, Alireza Yazdani, and George Em Karniadakis. Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations. *Science*, 367(6481):1026–1030, 2020.
- [30] Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *International conference on machine learning*, pages 1530–1538. PMLR, 2015.
- [31] Hannes Risken. Fokker-planck equation. In *The Fokker-Planck Equation*, pages 63–95. Springer, 1996.
- [32] Yeonjong Shin, Jerome Darbon, and George Em Karniadakis. On the convergence and generalization of physics informed neural networks. *arXiv e-prints*, pages arXiv–2004, 2020.
- [33] Justin Sirignano and Konstantinos Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. *Journal of computational physics*, 375:1339–1364, 2018.
- [34] Paul Sjöberg, Per Lötstedt, and Johan Elf. Fokker-planck approximation of the master equation in molecular biology. *Computing and Visualization in Science*, 12(1):37–50, 2009.
- [35] Keju Tang, Xiaoliang Wan, and Qifeng Liao. Deep density estimation via invertible block-triangular mapping. *Theoretical and Applied Mechanics Letters*, 10(3):143–148, 2020.
- [36] Kejun Tang, Xiaoliang Wan, and Qifeng Liao. Adaptive deep density approximation for fokker-planck equations. *arXiv preprint arXiv:2103.11181*, 2021.
- [37] M Fo Wehner and WG Wolfer. Numerical evaluation of path-integral solutions to fokker-planck equations. *Physical Review A*, 27(5):2663, 1983.
- [38] Liu Yang, Dongkun Zhang, and George Em Karniadakis. Physics-informed generative adversarial networks for stochastic differential equations. *SIAM Journal on Scientific Computing*, 42(1):A292–A317, 2020.
- [39] Liu Yang, Xuhui Meng, and George Em Karniadakis. B-pinns: Bayesian physics-informed neural networks for forward and inverse pde problems with noisy data. *Journal of Computational Physics*, 425:109913, 2021.

- [40] Yibo Yang and Paris Perdikaris. Adversarial uncertainty quantification in physics-informed neural networks. *Journal of Computational Physics*, 394:136–152, 2019.
- [41] Yaohua Zang, Gang Bao, Xiaojing Ye, and Haomin Zhou. Weak adversarial networks for high-dimensional partial differential equations. *Journal of Computational Physics*, 411:109409, 2020.
- [42] Linfeng Zhang, Jiequn Han, Han Wang, Roberto Car, and Weinan E. Deep potential molecular dynamics: a scalable model with the accuracy of quantum mechanics. *Physical review letters*, 120(14):143001, 2018.
- [43] Yinhao Zhu, Nicholas Zabaras, Phaedon-Stelios Koutsourelakis, and Paris Perdikaris. Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. *Journal of Computational Physics*, 394:56–81, 2019.