Journal of Scientific Computing (2025) 102:57
https://doi.org/10.1007/s10915-025-02791-7

Adaptive Deep Density Approximation for Stochastic Dynamical Systems

Junjie He[1] · Qifeng Liao[1] · Xiaoliang Wan[2]

Received: 28 June 2024 / Revised: 20 November 2024 / Accepted: 4 January 2025

Abstract
In this paper we consider adaptive deep neural network approximation for stochastic dynamical systems. Based on the continuity equation associated with the stochastic dynamical systems, a new temporal KRnet (tKRnet) is proposed to approximate the probability density functions (PDFs) of the state variables. The tKRnet provides an explicit density model for the solution of the continuity equation, which alleviates the curse of dimensionality issue that limits the application of traditional grid-based numerical methods. To efficiently train the tKRnet, an adaptive procedure is developed to generate collocation points for the corresponding residual loss function, where samples are generated iteratively using the approximate density function at each iteration. A temporal decomposition technique is also employed to improve the long-time integration. Theoretical analysis of our proposed method is provided, and numerical examples are presented to demonstrate its performance.

Keywords Stochastic dynamical systems · Continuity equation · Deep neural networks · Normalizing flows

Mathematics Subject Classification 34F05 · 60H35 · 62M45 · 65C30

1 Introduction

Stochastic dynamical systems naturally emerge in simulations and experiments involving complex systems (e.g., turbulence, semiconductors, and tumor cell growth), where the probability density functions (PDFs) of their states are typically governed by partial differential equations (PDEs) [2, 8, 31, 50]. These include the Fokker–Planck equation [37, 42], which

✉ Qifeng Liao
liaoqf@shanghaitech.edu.cn

Junjie He
hejj1@shanghaitech.edu.cn

Xiaoliang Wan
xlwan@math.lsu.edu

1 School of Information Science and Technology, ShanghaiTech University, Shanghai 201210, China

2 Department of Mathematics and Center for Computation and Technology, Louisiana State University, Baton Rouge, LA 70803, USA

allows for assessing the time evolution of PDFs in Langevin-type stochastic dynamical systems driven by Gaussian white noise, and continuity equations [26, 28, 46, 51] which model the evolution of PDFs in stochastic systems subject to random initial states and input parameters. However, it is challenging to solve these PDEs efficiently due to difficulties such as high-dimensionality of state variables, low regularities, conservation properties, and long-time integration [8]. This paper is devoted to developing new efficient deep learning methods for the continuity equations to address these issues.

The main idea of deep learning methods for solving PDEs is to reformulate a PDE problem as an optimization problem and train deep neural networks to approximate the solution by minimizing the corresponding loss function. Based on this idea, many techniques have been investigated to alleviate the difficulties existing in applying traditional grid methods (e.g., the finite element methods [13]) to complex PDEs. These include, for example, deep Ritz methods [11, 12], physics-informed neural networks (PINNs) [23, 41], deep Galerkin methods [45], Bayesian deep convolutional encoder-decoder networks [61, 62], weak adversarial networks [59] and deep multiscale model learning [55]. Deep neural network methods for complex geometries and interface problems are proposed in [15, 44, 56], and domain decomposition based deep learning methods [10, 17, 21, 24, 29, 30, 58] are studied to further improve the computational efficiency. In addition, residual network based learning strategies for unknown dynamical systems are presented in [5, 57].

As the solution of the continuity equation is a time-dependent probability density function, solving this problem can be considered as a time-dependent density estimation problem. While density estimation is a central topic in unsupervised learning [43], we focus on the normalizing flows [27] in this work. Normalizing flows are a class of generative models that parameterize a family of probability distributions, allowing for both tractable PDF computation and efficient sampling. The idea behind normalizing flows is to construct an invertible mapping from a simple prior distribution (e.g., a standard normal) to a more complex target distribution, which estimates the unknown distribution of interest. The invertible mapping transforms each sample from the unknown distribution to the prior distribution, and the PDF of the underlying unknown distribution can then be computed through the change of variables. Constructing an efficient invertible mapping is crucial for normalizing flows. Real-valued non-volume preserving (real NVP) flows address this by introducing non-volume-preserving affine coupling layers, which have been widely applied in tasks like image density estimation [9]. An extension of real NVP, called Glow, simplifies this architecture through the use of invertible 1×1 convolution [25]. For continuous transformations, neural ordinary differential equations [4] provide continuous normalizing flows, where the invertible mapping is parameterized as a continuous-time flow governed by an ordinary differential equation (ODE). This continuous generalization enables normalizing flows to handle more complex distributions. In our recent work [47], based on the Knothe-Rosenblatt (KR) rearrangement [3] and a modification of affine coupling layers in real NVP [9], a normalizing flow model called KRnet is proposed. Normalizing flows are primarily applied to density estimation, where they compute the exact likelihood of data and are trained by maximizing the log-likelihood. They have also been successfully used in variational inference by approximating posterior distributions [38], and more recently, they have been adapted to solve PDEs whose solutions are probability densities. A systematic procedure to train the KRnet for solving the steady-state Fokker–Planck equation is studied in [48]. In addition, an adaptive learning approach based on temporal normalizing flows is proposed for solving time-dependent Fokker–Planck equations in [14].

In order to efficiently solve the continuity equation, we generalize our KRnet to time-dependent problems and develop an adaptive training procedure. The modified KRnet is referred to as the temporal KRnet (tKRnet) in this paper. The main contributions of this work are as follows. First, the basic layers in KRnet (where the temporal variable is not included) are systematically extended to be time-dependent, and the initial condition of the underlying stochastic dynamical system is encoded in the tKRnet as a prior distribution. Second, an adaptive training procedure for tKRnet is proposed. It is known that choosing proper collocation points is crucial for solving PDEs with deep learning-based methods [48, 49]. To result in effective collocation points, our adaptive procedure has the following two main steps: training a tKRnet to approximate the solution of the continuity equation, and using the trained tKRnet to generate collocation points for the next iteration. Through this procedure, the distribution of the collocation points becomes more consistent with the solution PDF after each iteration. Third, for the challenging problem of long-time integration associated with the continuity equation, a temporal decomposition method is proposed, which provides guidance for applying tKRnet in this challenging problem. Lastly, a theoretical analysis is conducted to build the control of Kullback–Leibler (KL) divergence between the exact solution and the tKRnet approximation.

The rest of the paper is organized as follows. Preliminaries of stochastic dynamical systems and the corresponding continuity equations are presented in Sect. 2. Detailed formulations of tKRnet are given in Sect. 3. In Sect. 4, our adaptive training procedure of tKRnet for solving the continuity equation is presented, the corresponding temporal decomposition is discussed, and the analysis for the KL divergence between the exact solution and the tKRnet approximation is conducted. Numerical results are discussed in Sect. 5. Section 6 concludes the paper.

2 Problem Setup and Preliminaries

Let $\boldsymbol{\xi} \in \mathbb{R}^m$ and $\boldsymbol{y}_0 \in \mathbb{R}^n$ denote random vectors, where m and n are positive integers. We consider the following stochastic dynamical system

$$\frac{\mathrm{d}\boldsymbol{y}(t)}{\mathrm{d}t} = \boldsymbol{g}(\boldsymbol{y}, \boldsymbol{\xi}, t), \quad \boldsymbol{y}(0) = \boldsymbol{y}_0, \tag{1}$$

where $t \in \boldsymbol{I} := (0, T]$ ($T > 0$ is a given final time), and the function $\boldsymbol{g} : \mathbb{R}^n \times \mathbb{R}^m \times \boldsymbol{I} \to \mathbb{R}^n$ is a locally Lipschitz continuous vector function with respect to $\boldsymbol{y}$. The vector $\boldsymbol{y}(t) := [y_1(t), \dots, y_n(t)]^\top \in \mathbb{R}^n$ denotes a multi-dimensional stochastic process with the random vector $\boldsymbol{y}_0$ as the initial condition, and $\boldsymbol{y}(t)$ is also driven by the randomness of the random vector $\boldsymbol{\xi}$. Let $\boldsymbol{x}(t) := [\boldsymbol{y}(t), \boldsymbol{\xi}]^\top \in \mathbb{R}^d$ with $d = n + m$. The system (1) can be reformulated as

$$\frac{\mathrm{d}\boldsymbol{x}(t)}{\mathrm{d}t} = \boldsymbol{f}(\boldsymbol{x}, t), \quad \boldsymbol{x}(0) := [\boldsymbol{y}_0, \boldsymbol{\xi}]^\top \text{ and } \boldsymbol{f}(\boldsymbol{x}, t) := \begin{bmatrix} \boldsymbol{g}(\boldsymbol{x}, t) \\ \boldsymbol{0} \end{bmatrix}. \tag{2}$$

Our objective is to approximate the time-varying PDF of the state $\boldsymbol{x}(t)$ and the associated statistics.

The PDF of $\boldsymbol{x}(t)$, denoted as $p(\boldsymbol{x}, t) : \mathbb{R}^d \times \boldsymbol{I} \to \mathbb{R}_+$, satisfies the following continuity equation

$$\frac{\partial p(\boldsymbol{x}, t)}{\partial t} + \nabla_{\boldsymbol{x}} \cdot (p(\boldsymbol{x}, t)\boldsymbol{f}(\boldsymbol{x}, t)) = 0, \quad p(\boldsymbol{x}, 0) = p_0(\boldsymbol{x}) := p_{\boldsymbol{y}}(\boldsymbol{y}, 0) p_{\boldsymbol{\xi}}(\boldsymbol{\xi}), \tag{3}$$

where $\nabla_{\boldsymbol{x}} \cdot$ denotes the divergence in terms of $\boldsymbol{x}$, $p_{\boldsymbol{y}}(\boldsymbol{y}, 0)$ is the PDF of $\boldsymbol{y}_0$, and $p_{\boldsymbol{\xi}}(\boldsymbol{\xi})$ is the PDF of $\boldsymbol{\xi}$. To improve the numerical stability, the logarithmic continuity equation is proposed in [1, 51], which can be written as

$$\frac{\partial \log p(\boldsymbol{x},t)}{\partial t} + (\nabla_{\boldsymbol{x}} \log p(\boldsymbol{x},t)) \cdot \boldsymbol{f}(\boldsymbol{x},t) + \nabla_{\boldsymbol{x}} \cdot \boldsymbol{f}(\boldsymbol{x},t) = 0. \tag{4}$$

As $p(\boldsymbol{x},t)$ is a PDF for each $t \in \boldsymbol{I}$, it is required that

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

In addition, for any time t, the boundary condition for $p(\boldsymbol{x},t)$ is

$$p(\boldsymbol{x},t) \to 0 \text{ as } \|\boldsymbol{x}\|_2 \to \infty,$$

where $\|\cdot\|_2$ indicates the ℓ_2 norm.

3 Temporal KRnet (tKRnet)

The KRnet [47] is a flow-based generative model for density estimation or approximation, and its adaptive version is developed to solve the steady-state Fokker–Planck equation in [49]. We systematically generalize the KRnet to a time-dependent setting in this section, which is referred to as the temporal KRnet (tKRnet) henceforth, and develop an efficient adaptive training procedure for tKRnet to solve the continuity equation in the next section.

Let $\boldsymbol{X} \in \mathbb{R}^d$ be a random vector, which is associated with a time-dependent PDF $p_{\boldsymbol{X}}(\boldsymbol{x},t)$. In this work, $p_{\boldsymbol{X}}(\boldsymbol{x},t)$ is used to model the solution of (3) (or (4)). Let $\boldsymbol{Z} \in \mathbb{R}^d$ be a random vector associated with a PDF $p_{\boldsymbol{Z}}(\boldsymbol{z})$, where $p_{\boldsymbol{Z}}(\boldsymbol{z})$ is a prior distribution (e.g., a Gaussian distribution). The main idea of time-dependent normalizing flows is to seek a time-dependent invertible mapping $\mathbf{T} : \mathbb{R}^d \times \mathbb{R}_+ \to \mathbb{R}^d$ (i.e., $\boldsymbol{z} = \mathbf{T}(\boldsymbol{x},t)$), and by the change of variables, the PDF $p_{\boldsymbol{X}}(\boldsymbol{x},t)$ is given by

$$p_{\boldsymbol{X}}(\boldsymbol{x},t) = p_{\boldsymbol{Z}}(\boldsymbol{z})|\det \nabla_{\boldsymbol{x}}\mathbf{T}(\boldsymbol{x},t)|, \text{ where } \boldsymbol{z} = \mathbf{T}(\boldsymbol{x},t). \tag{5}$$

Once the prior distribution $p_{\boldsymbol{Z}}$ is specified, the explicit PDF for any random vector $\boldsymbol{X}$ and time t can be obtained through (5). Additionally, exact random samples from $p_{\boldsymbol{X}}(\boldsymbol{x},t)$ can be obtained using the samples of $\boldsymbol{Z}$ (from the prior) and the inverse of $\mathbf{T}$, i.e., $\boldsymbol{X} = \mathbf{T}^{-1}(\boldsymbol{Z},t)$. In the rest of this paper, we let $\mathbf{T}$ denote our tKRnet, which is constructed by a sequence of time-dependent bijections. These include affine coupling layers, scale-bias, squeezing and nonlinear layers, which are defined as follows.

3.1 Time-Dependent Affine Coupling Layers

A major part of tKRnet is affine coupling layers. Let $\boldsymbol{x}_{\text{in}} \in \mathbb{R}^{\tilde{d}}$ denote the input of a time-dependent affine coupling layer. The input $\boldsymbol{x}_{\text{in}}$ is partitioned as $\boldsymbol{x}_{\text{in}} = [\boldsymbol{x}_{\text{in}}^{(1)}, \boldsymbol{x}_{\text{in}}^{(2)}]^\top$, where $\boldsymbol{x}_{\text{in}}^{(1)} \in \mathbb{R}^k$, $\boldsymbol{x}_{\text{in}}^{(2)} \in \mathbb{R}^{\tilde{d}-k}$, and $k < \tilde{d}$ is a positive integer. For $t \in (0,T]$, the output of a time-dependent affine coupling layer $\boldsymbol{x}_{\text{out}} = [\boldsymbol{x}_{\text{out}}^{(1)}, \boldsymbol{x}_{\text{out}}^{(2)}]^\top$ is defined as

$$\begin{aligned} \boldsymbol{x}_{\text{out}}^{(1)} &= \boldsymbol{x}_{\text{in}}^{(1)} \\ \boldsymbol{x}_{\text{out}}^{(2)} &= \boldsymbol{x}_{\text{in}}^{(2)} + \frac{t}{T}\left(\alpha \boldsymbol{x}_{\text{in}}^{(2)} \odot \tanh(\boldsymbol{s}(\boldsymbol{x}_{\text{in}}^{(1)},t)) + e^{\boldsymbol{\beta}} \odot \tanh(\boldsymbol{n}(\boldsymbol{x}_{\text{in}}^{(1)},t))\right), \end{aligned} \tag{6}$$

where $0 < \alpha < 1$ is a fixed hyperparameter, $\boldsymbol{\beta} \in \mathbb{R}^{\tilde{d}-k}$ is a trainable parameter, tanh is the hyperbolic tangent function, and $\odot$ is the Hadamard product or elementwise multiplication.

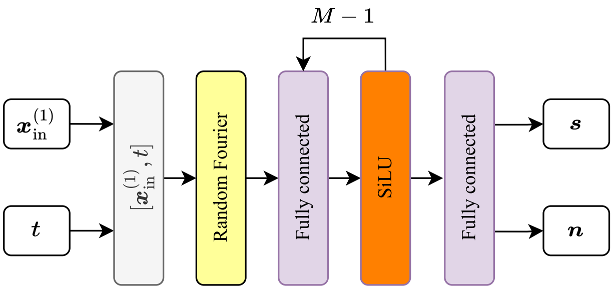


Fig. 1 The structure of the neural network in affine coupling layers. The neural network includes one random Fourier layer and M fully connected layers

We typically set $\alpha = 0.6$ and $k = \lfloor \tilde{d}/2 \rfloor$, where $\lfloor a \rfloor$ ($a \in \mathbb{R}$) is the largest integer that is smaller or equal to a. Similar to the KRnet, we explicitly control the scaling coefficient $1 + \frac{t}{T}\alpha \tanh(\boldsymbol{s}(\boldsymbol{x}_{\text{in}}^{(1)}, t)) \in [1 - \frac{t\alpha}{T}, 1 + \frac{t\alpha}{T}]$ using $\tanh(\cdot)$ to ensure the condition number does not increase too fast with the number of affine coupling layers. In (6), $\boldsymbol{s} : \mathbb{R}^{k+1} \to \mathbb{R}^{\tilde{d}-k}$ and $\boldsymbol{n} : \mathbb{R}^{k+1} \to \mathbb{R}^{\tilde{d}-k}$ stand for scale and translation respectively, which are modeled by a neural network, i.e.,

$$\begin{bmatrix} \boldsymbol{s} \\ \boldsymbol{n} \end{bmatrix} = \mathbf{N}(\boldsymbol{x}_{\text{in}}^{(1)}, t),$$

where $\mathbf{N} : \mathbb{R}^{k+1} \to \mathbb{R}^{2(\tilde{d}-k)}$ is a feedforward neural network.

Inspired by the work [54], we apply a random Fourier feature transformation before fully connected layers to mitigate possible challenges from multiscale structures. The neural network $\mathbf{N}$ is defined as

$$\begin{aligned} \boldsymbol{h}_0 &= [\boldsymbol{x}_{\text{in}}^{(1)}, t]^\top, \\ \boldsymbol{h}_1 &= \begin{bmatrix} \sin\left(\frac{1}{e^\sigma}\mathbf{F}\boldsymbol{h}_0 + \boldsymbol{b}_0\right) \\ \cos\left(\frac{1}{e^\sigma}\mathbf{F}\boldsymbol{h}_0 + \boldsymbol{b}_0\right) \\ \boldsymbol{h}_0 \end{bmatrix}, \\ \boldsymbol{h}_i &= \text{SiLU}(\mathbf{W}_{i-1}\boldsymbol{h}_{i-1} + \boldsymbol{b}_{i-1}), \quad \text{for} \quad i = 2, 3, \ldots, M, \\ \begin{bmatrix} \boldsymbol{s} \\ \boldsymbol{n} \end{bmatrix} &= \mathbf{W}_M \boldsymbol{h}_M + \boldsymbol{b}_M, \end{aligned} \tag{7}$$

where $\boldsymbol{h}_0 \in \mathbb{R}^{k+1}$ is the input vector, $\boldsymbol{h}_1 \in \mathbb{R}^{d_h+k+1}$ is the output of the random Fourier layer, and $\sigma \in \mathbb{R}$ is a learnable scaling parameter. In (7), $\mathbf{F} \in \mathbb{R}^{(d_h/2)\times(k+1)}$ and $\boldsymbol{b}_0 \in \mathbb{R}^{d_h/2}$ are a fixed weight matrix and a fixed bias vector, and the entries of $\mathbf{F}$ and $\boldsymbol{b}_0$ are sampled from the Gaussian distribution $\mathcal{N}(0, 1)$ and the uniform distribution $\mathcal{U}(0, 2\pi)$ respectively. Figure 1 illustrates the structure of the neural network in affine coupling layers, where *Random Fourier* and *Fully connected* are random Fourier layers and fully connected layers respectively, and *SiLU* is the sigmoid linear unit (SiLU) function.

For the fully connected layers, $\mathbf{W}_1 \in \mathbb{R}^{d_h\times(d_h+k+1)}$ and $\boldsymbol{b}_1 \in \mathbb{R}^{d_h}$ are the weight and the bias in the first fully connected layer, and $\mathbf{W}_i \in \mathbb{R}^{d_h\times d_h}$ and $\boldsymbol{b}_i \in \mathbb{R}^{d_h}$ are weight and bias coefficients in the ith fully connected layer, for $i = 2, \ldots, M-1$; $\boldsymbol{h}_i \in \mathbb{R}^{d_h}$ is the hidden feature output of the $(i-1)$th fully connected layer. The output is computed by the final layer with $\mathbf{W}_M \in \mathbb{R}^{2(\tilde{d}-k)\times d_h}$ and $\boldsymbol{b}_M \in \mathbb{R}^{2(\tilde{d}-k)}$. The SiLU function [18] is applied as the nonlinear activation function in the neural network, which is defined as

$$\mathrm{SiLU}(x) = \frac{x}{1 - e^{-x}}.$$

The trainable parameters of the fully connected layer are $\{\mathbf{W}_i, \boldsymbol{b}_i | i = 1, \ldots, M\}$.

The Jacobian of $\boldsymbol{x}_{\text{out}}$ at time t with respect to $\boldsymbol{x}_{\text{in}}$ can be obtained by

$$\nabla_{\boldsymbol{x}_{\text{in}}} \boldsymbol{x}_{\text{out}} = \begin{bmatrix} \frac{\partial \boldsymbol{x}_{\text{out}}^{(1)}}{\partial \boldsymbol{x}_{\text{in}}^{(1)}} & \frac{\partial \boldsymbol{x}_{\text{out}}^{(1)}}{\partial \boldsymbol{x}_{\text{in}}^{(2)}} \\ \frac{\partial \boldsymbol{x}_{\text{out}}^{(2)}}{\partial \boldsymbol{x}_{\text{in}}^{(1)}} & \frac{\partial \boldsymbol{x}_{\text{out}}^{(2)}}{\partial \boldsymbol{x}_{\text{in}}^{(2)}} \end{bmatrix} = \begin{bmatrix} \mathbb{I}_k & \mathbf{0} \\ \frac{\partial \boldsymbol{x}_{\text{out}}^{(2)}}{\partial \boldsymbol{x}_{\text{in}}^{(1)}} & \mathrm{diag}(\mathbf{1}_{\tilde{d}-k} + \alpha \frac{t}{T} \tanh(\boldsymbol{s}(\boldsymbol{x}_{\text{in}}^{(1)}, t))) \end{bmatrix}, \tag{8}$$

where $\mathbb{I}_k \in \mathbb{R}^{k \times k}$ represents a $k \times k$ identity matrix, and $\mathrm{diag}(\cdot)$ is a diagonal matrix. The determinant of $\nabla_{\boldsymbol{x}_{\text{in}}} \boldsymbol{x}_{\text{out}}$ can then be easily computed by multiplying the diagonal entries of $\mathrm{diag}(\mathbf{1}_{\tilde{d}-k} + \alpha \frac{t}{T} \tanh(\boldsymbol{s}(\boldsymbol{x}_{\text{in}}^{(1)}, t)))$.

It is noted that, in the affine coupling layer, only $\boldsymbol{x}_{\text{in}}^{(2)}$ is transformed to $\boldsymbol{x}_{\text{out}}^{(2)}$, while $\boldsymbol{x}_{\text{in}}^{(1)}$ remains unchanged. To completely update all components of $\boldsymbol{x}_{\text{in}}$, the unchanged part $\boldsymbol{x}_{\text{in}}^{(1)}$ is updated in the next affine coupling layer,

$$\begin{aligned} \boldsymbol{x}_{\text{out}}^{(1)} &= \boldsymbol{x}_{\text{in}}^{(1)} + \frac{t}{T} \left(\alpha \boldsymbol{x}_{\text{in}}^{(1)} \odot \tanh(\boldsymbol{s}(\boldsymbol{x}_{\text{in}}^{(2)}, t)) + e^{\boldsymbol{\beta}} \odot \tanh(\boldsymbol{n}(\boldsymbol{x}_{\text{in}}^{(2)}, t)) \right) \\ \boldsymbol{x}_{\text{out}}^{(2)} &= \boldsymbol{x}_{\text{in}}^{(2)}, \end{aligned}$$

where $\boldsymbol{s}$ and $\boldsymbol{n}$ are separate for each layer of the whole transformation.

3.2 Scale-Bias, Squeezing and Nonlinear Layers

We generalize the *scale-bias layer* introduced in [25] to a time-dependent setting. For an input $\boldsymbol{x}_{\text{in}} \in \mathbb{R}^{\tilde{d}}$, the output of the time-dependent scale-bias layer is defined as

$$\boldsymbol{x}_{\text{out}} = e^{\tanh(\boldsymbol{\phi} t) \odot \boldsymbol{a}} \odot \boldsymbol{x}_{\text{in}} + \tanh(\boldsymbol{\phi} t) \odot \boldsymbol{b}, \tag{9}$$

where $\boldsymbol{a} \in \mathbb{R}^{\tilde{d}}$, $\boldsymbol{b} \in \mathbb{R}^{\tilde{d}}$, and $\boldsymbol{\phi} = [\phi_1, \ldots, \phi_{\tilde{d}}]^\top \in \mathbb{R}^{\tilde{d}}$ are trainable parameters, and each ϕ_i (for $i = 1, \ldots, \tilde{d}$) is required to be positive. This can be considered as a simplified version of batch normalization for rescaling.

Squeezing layer L_S is applied to deactivate some dimensions through a mask

$$\mathbf{m} = [\underbrace{1, \ldots, 1}_{k}, \underbrace{0, \ldots, 0}_{\tilde{d}-k}]^\top, \tag{10}$$

where the components $\mathbf{m} \odot \boldsymbol{x}_{\text{in}}$ are updated and the rest components $(1 - \mathbf{m}) \odot \boldsymbol{x}_{\text{in}}$ are fixed from then on.

Time-dependent nonlinear layer L_N extends the nonlinear layers introduced in [47] such that it is consistent with an identity transformation at $t = 0$. For an input $\boldsymbol{x}_{\text{in}}$, where the support of each dimension is $(-\infty, \infty)$, the mapping $\hat{F}(\cdot) : (-\infty, \infty) \to (-\infty, \infty)$ is defined as

$$\hat{F}(x) = \begin{cases} \beta_s(x + a) - a, & x \in (-\infty, -a) \\ 2aF(\frac{x+a}{2a}) - a, & x \in [-a, a] \\ \beta_s(x - a) + a, & x \in (a, \infty), \end{cases} \tag{11}$$

where $\beta_s \in \mathbb{R}_+$ is a scaling factor, $a \in \mathbb{R}_+$ is a fixed hyperparameter, and $F(\cdot) : [0, 1] \to [0, 1]$ represents a nonlinear quadratic function. The inverse and the derivative with respect to x of the nonlinear mapping $\hat{F}$ can be explicitly computed [48].

The nonlinear function applied in (11) is constructed as follows. Let the interval $[0, 1]$ be discretized by a mesh $0 = s_0 < s_1 < \cdots < s_{\hat{m}+1} = 1$ with the element size $h_i = s_{i+1} - s_i$, where $\hat{m}$ is a given positive integer. For each sub-interval $[s_i, s_{i+1}]$, a nonlinear quadratic function defines the nonlinear mapping $F(\cdot) : [0, 1] \to [0, 1]$:

$$F(s) = \frac{w_{i+1} - w_i}{2h_i}(s - s_i)^2 + w_i(s - s_i) + \sum_{k=0}^{i-1} \frac{w_k + w_{k+1}}{2} h_k. \tag{12}$$

In (12), the parameters w_i are normalized as $w_i = \hat{w}_i / c_w$ where

$$\hat{w}_i = e^{\tanh(\varphi_i t)\psi_i}, \quad c_w = \sum_{i=0}^{\hat{m}} \frac{\hat{w}_i + \hat{w}_{i+1}}{2} h_i,$$

and $\varphi_i \in \mathbb{R}_+$, $\psi_i \in \mathbb{R}$ are trainable parameters for $i = 0, \ldots, \hat{m}$.

3.3 The Overall Structure of tKRnet

Our tKRnet is inspired by the KR rearrangement [3], which defines a triangular transport map between two probability measures. Before presenting the tKRnet, we review the KR rearrangement following [3] as follows. Let μ and ν be two absolutely continuous Borel probability density measures on $\mathbb{R}^d$. Denote by μ^1 the first marginal of μ and by $\mu^i_{(x_1,\ldots,x_{i-1})}$ the ith conditional measure of μ given $(x_1, \ldots, x_{i-1})$. Respectively, let ν^1 be the first marginal of ν, and $\nu^i_{(z_1,\ldots,z_{i-1})}$ denote the ith conditional measure of ν given $(z_1, \ldots, z_{i-1})$. Define $\mathbf{T}^{(1)}_{\mathrm{KR}}$ as the one-dimensional monotone nondecreasing transformation from μ^1 to ν^1. Similarly, let $\mathbf{T}^{(2)}_{\mathrm{KR}}(x_1, x_2)$ denote the monotone transformation in x_2 from $\mu^2_{x_1}$ to $\nu^2_{z_1}$. The transformation can be constructed iteratively until the map $\mathbf{T}^{(d)}_{\mathrm{KR}}(x_1, x_2, \ldots, x_d)$ is defined, which transports $\mu^d_{(x_1,\ldots,x_{d-1})}$ to $\nu^d_{(z_1,\ldots,z_{d-1})}$. The KR rearrangement provides a transport map from μ to ν, which is defined as

$$\mathbf{T}_{\mathrm{KR}}(\boldsymbol{x}) = \begin{bmatrix} \mathbf{T}^{(1)}_{\mathrm{KR}}(x_1) \\ \mathbf{T}^{(2)}_{\mathrm{KR}}(x_1, x_2) \\ \vdots \\ \mathbf{T}^{(d)}_{\mathrm{KR}}(x_1, \ldots, x_d) \end{bmatrix}.$$

This map is a lower triangular map and is characterized by a triangular Jacobian matrix with nonnegative diagonal entries. We incorporate the triangular structure of the KR rearrangement to design our tKRnet, ensuring efficient transformation between probability distributions.

The forward computation of the tKRnet is then defined as a composite transformation

$$z = \mathbf{T}(\boldsymbol{x}, t; \Theta) = L_N \circ \mathbf{T}_{[K]} \circ \cdots \circ \mathbf{T}_{[1]}(\boldsymbol{x}, t), \tag{13}$$

where Θ represents the transformation parameters, and $\mathbf{T}_{[i]}$ is defined as

$$\mathbf{T}_{[i]} := \begin{cases} L_S \circ \mathbf{T}_{[i,L]} \circ \cdots \circ \mathbf{T}_{[i,1]}, & i = 1, \ldots, K-1, \\ \mathbf{T}_{[i,L]} \circ \cdots \circ \mathbf{T}_{[i,1]}, & i = K. \end{cases} \tag{14}$$

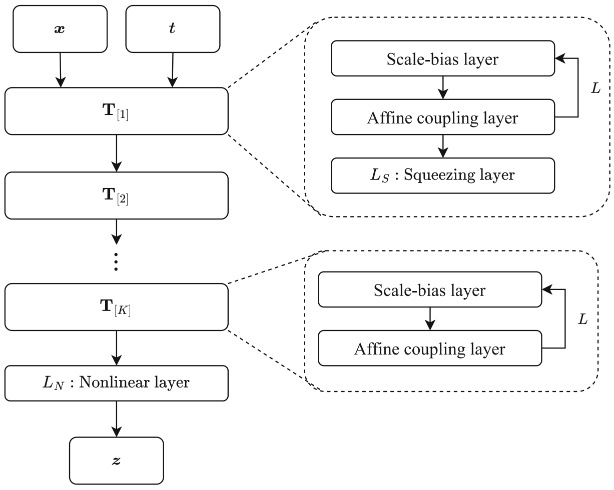


Fig. 2 The overall structure of tKRnet; tKRnet includes K invertible transformation blocks $\mathbf{T}_{[i]}$ (for $i = 1, \ldots, K$) and a nonlinear layer L_N; each $\mathbf{T}_{[i]}$ includes L scale-bias layers, L affine coupling layers and one squeezing layer L_S

In (13)–(14), each $\mathbf{T}_{[i]}$ includes L invertible mappings $\mathbf{T}_{[\cdot,\cdot]}$ for $i = 1, \ldots, K$, and each invertible mapping $\mathbf{T}_{[\cdot,\cdot]}$ is composed by a scale-bias layer (9) and an affine coupling layer (see Sect. 3.1). The terms L_N, L_S denote the nonlinear layer (12) and the squeezing layer (10) respectively.

Let the state variable $\boldsymbol{x}$ be partitioned as $\boldsymbol{x} = [\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(K)}]^\top$, where $\boldsymbol{x}^{(i)} = [\boldsymbol{x}_1^{(i)}, \ldots, \boldsymbol{x}_m^{(i)}]^\top$ with $1 \le K \le d$, $1 \le m \le d$, and $\sum_{i=1}^K \dim(\boldsymbol{x}^{(i)}) = d$. The transformation $\boldsymbol{x}_{[i]} = \mathbf{T}_{[i]}(\boldsymbol{x}_{[i-1]}, t)$ is applied iteratively, starting from $\boldsymbol{x}_{[0]} = \boldsymbol{x}$. The partitions $\boldsymbol{x}_{[i]} = [\boldsymbol{x}_{[i]}^{(1)}, \ldots, \boldsymbol{x}_{[i]}^{(K)}]^\top$ remain consistent for $i = 1, \ldots, K$. At the beginning, $\mathbf{T}_{[1]}$ is applied to the partition $\boldsymbol{x}_{[0]} = [\boldsymbol{x}_{[0]}^{(1:K-1)}, \boldsymbol{x}_{[0]}^{(K)}]^\top$, where $\boldsymbol{x}_{[0]}^{(1:K-1)}$ includes $\{\boldsymbol{x}_{[0]}^{(1)}, \ldots, \boldsymbol{x}_{[0]}^{(K-1)}\}$. Following this, $\boldsymbol{x}_{[1]} = \mathbf{T}_{[1]}(\boldsymbol{x}_{[0]}, t)$ is computed, and after that, the last partition $\boldsymbol{x}_{[1]}^{(K)}$ remains fixed, i.e., $\boldsymbol{x}_{[i]}^{(K)} = \boldsymbol{x}_{[1]}^{(K)}$ for $i > 1$. In general, after applying the transformation $\mathbf{T}_{[i]}$, the $(K - i + 1)$th partition of $\boldsymbol{x}_{[i]}$, i.e., $\boldsymbol{x}_{[i]}^{(K-i+1)}$, is deactivated in addition to the dimensions fixed at previous stages. Finally, the nonlinear layer L_N transforms $\boldsymbol{x}_{[K]}$ to z, where $z = L_N(\boldsymbol{x}_{[K]}, t)$. For brevity, denoting $\tilde{z} = \boldsymbol{x}_{[K]}$, the inverse of the tKRnet corresponds to a blockwise lower-triangular transport map, i.e.,

$$\boldsymbol{x} = \begin{bmatrix} \mathbf{T}_{[K]}^{-1}(\tilde{z}^{(1)}, t) \\ \mathbf{T}_{[K-1]}^{-1}(\tilde{z}^{(1)}, \tilde{z}^{(2)}, t) \\ \vdots \\ \mathbf{T}_{[1]}^{-1}(\tilde{z}^{(1)}, \ldots, \tilde{z}^{(K)}, t) \end{bmatrix}.$$

The overall structure of tKRnet is depicted in Fig. 2.

Based on the tKRnet $\mathbf{T}(\cdot,\cdot;\Theta)$, the PDF $p_\Theta(\boldsymbol{x},t)$ can be obtained by (5) and the chain rule,

$$\begin{aligned} p_\Theta(\boldsymbol{x},t) &= p_Z(\mathbf{T}(\boldsymbol{x},t;\Theta))|\det\nabla_{\boldsymbol{x}}\mathbf{T}(\boldsymbol{x},t;\Theta)| \\ &= p_Z(\mathbf{T}(\boldsymbol{x},t;\Theta))|\det\nabla_{\boldsymbol{x}_{[K]}}L_N(\boldsymbol{x}_{[K]},t)|\prod_{i=1}^{K}|\det\nabla_{\boldsymbol{x}_{[i-1]}}\mathbf{T}_{[i]}(\boldsymbol{x}_{[i-1]},t)|, \end{aligned} \tag{15}$$

where $\boldsymbol{x}_{[0]}=\boldsymbol{x}$, $\boldsymbol{x}_{[i]}=\mathbf{T}_{[i]}(\boldsymbol{x}_{[i-1]},t)$, for $i=1,\dots,K$. To use $p_\Theta(\boldsymbol{x},t)$ as a PDF model for the solution of the continuity equation (3), the prior distribution p_Z is set to the initial PDF p_0. At time $t=0$, all sublayers of tKRnet are identity mappings, and $p_\Theta(\boldsymbol{x},0)=p_0(\mathbf{T}(\boldsymbol{x},0;\Theta))|\det\nabla_{\boldsymbol{x}}\mathbf{T}(\boldsymbol{x},0;\Theta)|=p_0(\boldsymbol{x})$.

4 Adaptive Sampling Based Physics-Informed Training for Density Approximation

In this section, tKRnet is applied to solve the time-dependent continuity equation (3) or its variant (4). We pay particular attention to adaptive sampling and long-time integration.

4.1 Physics-Informed Training and Adaptive Sampling

Adaptivity is crucial for numerically solving PDF equations. The work [7] uses spectral elements on an adaptive non-conforming grid to discretize the spatial domain of the PDF equation and track the time-dependent PDF support. The work [48] shows that adaptive sampling strategy can help normalizing flow models effectively approximate solutions to steady-state Fokker–Planck equations. We propose a physics-informed method consisting of multiple adaptivity iterations. Each adaptivity iteration has two steps: (1) training tKRnets by minimizing the total PDE residual on collocation points in the training set; (2) generating collocation points dynamically through the trained tKRnet to update the training set.

Let $p_\Theta(\boldsymbol{x},t)$ be the approximate PDF induced by tKRnet. The residual given by the continuity equation (3) is defined as

$$r(\boldsymbol{x},t;\Theta):=\frac{\partial p_\Theta(\boldsymbol{x},t)}{\partial t}+\nabla_{\boldsymbol{x}}\cdot(p_\Theta(\boldsymbol{x},t)\boldsymbol{f}(\boldsymbol{x},t)). \tag{16}$$

In high-dimensional problems, the value of $p_\Theta(\boldsymbol{x},t)$ may be too small, which results in underflow such that the residual cannot be effectively minimized. To alleviate this issue, the logarithmic continuity equation (4) is considered, and the corresponding residual is defined as

$$r_{\log}(\boldsymbol{x},t;\Theta):=\frac{\partial\log p_\Theta(\boldsymbol{x},t)}{\partial t}+(\nabla_{\boldsymbol{x}}\log p_\Theta(\boldsymbol{x},t))\cdot\boldsymbol{f}(\boldsymbol{x},t)+\nabla_{\boldsymbol{x}}\cdot\boldsymbol{f}(\boldsymbol{x},t). \tag{17}$$

It is clear that $r(\boldsymbol{x},t;\Theta)=p_\Theta(\boldsymbol{x},t)r_{\log}(\boldsymbol{x},t;\Theta)$. Letting $p_c(\boldsymbol{x},t)$ be a PDF in the space-time domain, the following loss functional is defined,

$$\mathcal{L}(p_\Theta(\boldsymbol{x},t)):=\mathbb{E}_{p_c(\boldsymbol{x},t)}(|r_{\log}(\boldsymbol{x},t;\Theta)|^2), \tag{18}$$

where $\mathbb{E}_{p_c}$ refers to the expectation with respect to $p_c(\boldsymbol{x},t)$. A set of collocation points $\mathcal{S}=\{(\boldsymbol{x}_{\text{res}}^{(i)},t_{\text{res}}^{(i)})\}_{i=1}^{N_r}$ are drawn from $p_c(\boldsymbol{x},t)$, and these points are employed to estimate the loss function as

$$\mathcal{L}(p_\Theta(\boldsymbol{x},t)) \approx \hat{\mathcal{L}}(p_\Theta(\boldsymbol{x},t)) := \frac{1}{N_r}\sum_{i=1}^{N_r} |r_{\log}(\boldsymbol{x}_{\text{res}}^{(i)}, t_{\text{res}}^{(i)}; \Theta)|^2. \tag{19}$$

Optimal parameters for $p_\Theta(\boldsymbol{x},t)$ are chosen by minimizing the approximate loss function, i.e.,

$$\Theta^* = \arg\min_{\Theta} \hat{\mathcal{L}}(p_\Theta(\boldsymbol{x},t)). \tag{20}$$

A variant of the stochastic gradient descent method, AdamW [33], is applied to solve the optimization problem (20). The learning rate scheduler for the optimizer is the cosine annealing method [32]. More specifically, the set $\mathcal{S}$ is divided into N_b mini-batches $\{\mathcal{S}^{(n)}\}_{n=1}^{N_b}$. Let $\Theta_{e,n}$ be the model parameters at the nth step of epoch e with $e = 1, \dots, N_E$ and $n = 1, \dots, N_b$. The model parameters are updated as

$$\Theta_{e,n} = \Theta_{e,n-1} - \tau \nabla_{\Theta_{e,n-1}} \frac{1}{|\mathcal{S}^{(n)}|} \sum_{(\boldsymbol{x}_{\text{res}}, t_{\text{res}}) \in \mathcal{S}^{(n)}} |r_{\log}(\boldsymbol{x}_{\text{res}}, t_{\text{res}}; \Theta_{e,n-1})|^2, \tag{21}$$

where $\tau > 0$ is the learning rate. After each optimization step, the learning rate is decreased by the learning rate scheduler.

To enhance the accuracy of the final approximation, we propose the following adaptive sampling strategy. Let $N_r = MJ$, where M and J are positive integers. On $(0, T]$, $t_{\text{res}}^{(i)}$ (for $i = 1, \dots, N_r$) are set to

$$t_{\text{res}}^{(i)} = \lceil i/M \rceil \Delta t, \tag{22}$$

where $\Delta t = T/(J-1)$ and $\lceil i/M \rceil$ is the smallest integer that is larger or equal to i/M. Letting $\mathcal{S}_{\text{time}} = \{t_{\text{res}}^{(i)}\}_{i=1}^{N_r}$, the training set is initialized as $\mathcal{S}_0 := \{(\boldsymbol{x}_{\text{res}}^{(0,i)}, t_{\text{res}}^{(i)})\}_{i=1}^{N_r}$, where $\{\boldsymbol{x}_{\text{res}}^{(0,i)}\}_{i=1}^{N_r}$ are the initial spatial collocation points (e.g., samples of a uniform distribution). The set $\mathcal{S}_0$ is divided into N_b mini-batches $\{\mathcal{S}_0^{(n)}\}_{n=1}^{N_b}$. The tKRnet (introduced in Sect. 3.3) is initialized as $\mathbf{T}(\boldsymbol{x}, t; \Theta^{(0)})$. In general, the ith ($i = 1, \dots, N_r$) spatial collocation point generated at the kth adaptivity iteration step is denoted by $\boldsymbol{x}_{\text{res}}^{(k,i)}$. The parameters at the kth adaptivity iteration step, the eth epoch and the nth optimization iteration step is denoted by $\Theta_{e,n}^{(k)}$.

Starting with $\Theta_{1,0}^{(1)} = \Theta^{(0)}$, the tKRnet is trained through solving (20) with collocation points $\mathcal{S}_0$, and the trained tKRnet at adaptivity iteration step one is denoted by $\mathbf{T}(\boldsymbol{x}, t; \Theta^{*,(1)})$, where $\Theta^{*,(1)}$ are the parameters of the trained tKRnet at this step. The PDF $p_{\Theta^{*,(1)}} = p_0(\mathbf{T}(\boldsymbol{x}, t; \Theta^{*,(1)}))|\det \nabla_{\boldsymbol{x}} \mathbf{T}(\boldsymbol{x}, t; \Theta^{*,(1)})|$ (see (15)) is then used to generate new collocation points $\mathcal{S}_1 = \{(\boldsymbol{x}_{\text{res}}^{(1,i)}, t_{\text{res}}^{(i)})\}_{i=1}^{N_r}$, where $t_{\text{res}}^{(i)} \in \mathcal{S}_{\text{time}}$ remains unchanged and each $\boldsymbol{x}_{\text{res}}^{(1,i)}$ is drawn from $p_{\Theta^{*,(1)}}(\boldsymbol{x}, t_{\text{res}}^{(i)})$. To generate each $\boldsymbol{x}_{\text{res}}^{(1,i)}$, a sample point $\boldsymbol{z}^{(1,i)}$ is first generated using $p_{\boldsymbol{Z}} = p_0$, and then $\boldsymbol{x}_{\text{res}}^{(1,i)} = \mathbf{T}^{-1}(\boldsymbol{z}^{(1,i)}, t_{\text{res}}^{(i)}; \Theta^{*,(1)})$. Next, starting with $\mathbf{T}(\boldsymbol{x}, t; \Theta_{1,0}^{(2)})$ where $\Theta_{1,0}^{(2)} := \Theta^{*,(1)}$, we continue the training process with the collocation points $\mathcal{S}_1$ to obtain $\mathbf{T}(\boldsymbol{x}, t; \Theta^{*,(2)})$. In general, at adaptivity iteration step k, the collocation points $\mathcal{S}_k$ are generated using $p_{\Theta^{*,(k)}}(\boldsymbol{x}, t)$, and the tKRnet with the initial parameters $\Theta_{1,0}^{(k+1)} := \Theta^{*,(k)}$ is subsequently updated to $\mathbf{T}(\boldsymbol{x}, t; \Theta^{*,(k+1)})$ through solving (20). The adaptivity iteration continues until the maximum number of iterations is achieved, which is denoted by $N_{\text{adaptive}} \in \mathbb{N}_+$. This adaptive procedure is summarized in Algorithm 1.

Algorithm 1 Adaptive sampling based physics-informed training for tKRnet

```
Input: The initial tKRnet $\mathbf{T}(\boldsymbol{x}, t; \Theta^{(0)})$, the number of collocation points $N_r$, the number of epochs $N_E$, the maximum number of adaptivity iterations $N_{\text{adaptive}}$, the learning rate $\tau$, and the number of mini-batches $N_b$.
1: Generate time points $\mathcal{S}_{\text{time}} = \{t_{\text{res}}^{(i)}\}_{i=1}^{N_r}$ (see (22)).
2: Draw initial samples $\{\boldsymbol{x}_{\text{res}}^{(0,i)}\}_{i=1}^{N_r}$ from a given distribution, and let $\mathcal{S}_0 = \{(\boldsymbol{x}_{\text{res}}^{(0,i)}, t_{\text{res}}^{(i)})\}_{i=1}^{N_r}$.
3: Divide $\mathcal{S}_0$ into $N_b$ mini-batches $\{\mathcal{S}_0^{(n)}\}_{n=1}^{N_b}$.
4: for $k = 1,\ldots,N_{\text{adaptive}}$ do
5:   if $k = 1$ then
6:     Let $\Theta_{1,0}^{(k)} = \Theta^{(0)}$.
7:   else
8:     Let $\Theta_{1,0}^{(k)} = \Theta^{*,(k-1)}$.
9:   end if
10:  Initialize the AdamW optimizer and the cosine annealing learning rate scheduler.
11:  for $e = 1,\ldots,N_E$ do
12:    for $n = 1,\ldots,N_b$ do
13:      Compute the loss $\hat{\mathcal{L}}(p_{\Theta_{e,n-1}^{(k)}}(\boldsymbol{x}, t))$ (see (19)) on the mini-batch $\mathcal{S}_{k-1}^{(n)}$.
14:      Update $\Theta_{e,n}^{(k)}$ using the AdamW optimizer with the learning rate $\tau$ (see (21)).
15:      The learning rate scheduler decreases the learning rate.
16:    end for
17:    if $e < N_E$ then
18:      Let $\Theta_{e+1,0}^{(k)} = \Theta_{e,N_b}^{(k)}$.
19:      Shuffle the mini-batches $\{\mathcal{S}_{k-1}^{(n)}\}_{n=1}^{N_b}$ of the set $\mathcal{S}_{k-1}$.
20:    end if
21:  end for
22:  Let $\Theta^{*,(k)} = \Theta_{N_E,N_b}^{(k)}$.
23:  Draw $\boldsymbol{x}_{\text{res}}^{(k,i)}$ from $p_{\Theta^{*,(k)}}(\boldsymbol{x}, t_{\text{res}}^{(i)})$, and let $\mathcal{S}_k = \{(\boldsymbol{x}_{\text{res}}^{(k,i)}, t_{\text{res}}^{(i)})\}_{i=1}^{N_r}$.
24:  Divide $\mathcal{S}_k$ into $N_b$ mini-batches $\{\mathcal{S}_k^{(n)}\}_{n=1}^{N_b}$.
25: end for
26: Let $\Theta^* = \Theta^{*,(N_{\text{adaptive}})}$.
Output: The trained tKRnet $\mathbf{T}(\boldsymbol{x}, t; \Theta^*)$, and the approximate solution $p_{\Theta^*}(\boldsymbol{x}, t)$.
```

4.2 Temporal Decomposition for Long-Time Integration

The performance of PINN may deteriorate for evolution equations when the time domain becomes large [53]. Algorithm 1 suffers a similar issue since it is defined in the framework of PINN. Some remedies [35, 40] have been proposed to alleviate this issue. In this work, we employ a temporal decomposition method when implementing Algorithm 1 on a large time domain. We will demonstrate numerically that coupling temporal decomposition with adaptive sampling yields efficient performance for long-time integration, although other techniques [40, 53] can also be employed for further refinement.

We decompose the time interval $(0, T]$ into N_{sub} sub-intervals $(T_{i-1}, T_i]$ (for $i = 1, \ldots, N_{sub}$), where $0 = T_0 < T_1 < \cdots < T_{N_{sub}} = T$. For the ith sub-interval $(T_{i-1}, T_i]$, assuming that the PDF $p_\Theta(\cdot, T_{i-1})$ is given (e.g., for the interval $(T_0, T_1]$, $p_\Theta(\cdot, T_0)$ is given a priori), our goal is to train tKRnet $\mathbf{T}(\cdot, \cdot; \Theta^{(i,k)})$, where $\Theta^{(i,k)}$ includes the model parameters and k denotes the adaptivity iteration step (see line 4 of Algorithm 1). The trained parameters of the tKRnet for ith sub-interval are denoted by $\Theta^{(i),*}$. The following two choices for temporal decomposition are considered in this work to train the local tKRnets. The first choice is to keep the same tKRnet structure for different sub-intervals (while the local tKRnet parameters are different), and to introduce the cross entropy to maintain the consistency between two adjacent sub-intervals; the second choice is to use $p_{\Theta}(\cdot, T_{i-1})$ as the prior distribution to construct a new tKRnet for each sub-interval $(T_{i-1}, T_i]$.

In the first choice, the solution of (3) or (4) on $(T_{i-1}, T_i]$ is approximated by $p(\boldsymbol{x}, t) \approx p_{\Theta^{(i,k)}}(\boldsymbol{x}, t) := p_0(\boldsymbol{z})|\det \nabla_{\boldsymbol{x}} \mathbf{T}(\boldsymbol{x}, t; \Theta^{(i,k)})|$ where $\boldsymbol{z} = \mathbf{T}(\boldsymbol{x}, t; \Theta^{(i,k)})$ and $\mathbf{T}$ is introduced in Sect. 3. Here, we let $p_{\Theta^{(i,k)}}(\boldsymbol{x}, T_{i-1}) \approx p_{\Theta^{(i-1),*}}(\boldsymbol{x}, T_{i-1})$, where $p_{\Theta^{(i-1),*}}(\boldsymbol{x}, T_{i-1})$ is the given tKRnet solution at $t = T_{i-1}$. The cross entropy between $p_{\Theta^{(i-1),*}}(\boldsymbol{x}, T_{i-1})$ and $p_{\Theta^{(i,k)}}(\boldsymbol{x}, T_{i-1})$ is next introduced to maintain the consistency at time $t = T_{i-1}$. That is, the loss function on $(T_{i-1}, T_i]$ is defined as

$$\begin{aligned}\hat{\mathcal{L}}(p_{\Theta^{(i,k)}}(\boldsymbol{x}, t)) = & \frac{1}{N_r}\sum_{j=1}^{N_r} |r_{\log}(\boldsymbol{x}_{\text{res}}^{(j)}, t_{\text{res}}^{(j)}; \Theta^{(i,k)})|^2 \\ & - \frac{1}{N_r}\sum_{j=1}^{N_r} \log p_{\Theta^{(i,k)}}(\boldsymbol{x}_{\text{interface}}^{(j)}, T_{i-1}),\end{aligned} \tag{23}$$

where $\boldsymbol{x}_{\text{interface}}^{(j)}$ are drawn from $p_{\Theta^{(i-1),*}}(\cdot, T_{i-1})$. Note that the loss function for $\mathbf{T}(\cdot, \cdot; \Theta^{(1,k)})$ (on the sub-interval $(T_0, T_1]$) is

$$\hat{\mathcal{L}}(p_{\Theta^{(1,k)}}(\boldsymbol{x}, t)) = \frac{1}{N_r}\sum_{j=1}^{N_r} |r_{\log}(\boldsymbol{x}_{\text{res}}^{(j)}, t_{\text{res}}^{(j)}; \Theta^{(1,k)})|^2, \tag{24}$$

where only the residual is considered, because the initial distribution at $t = 0$ is used as the prior distribution for the tKRnet. Then, Algorithm 1 is applied with (24) to obtain the local solution $p_{\Theta^{(1),*}}$ for the first sub-interval $(T_0, T_1]$. After that, Algorithm 1 is applied with (23) to train $p_{\Theta^{(2),*}}$, and this procedure is repeated for computing $(T_{i-1}, T_i]$ for $i = 3, \ldots, N_{sub}$.

In the second choice, the local tKRnet for each sub-interval is rebuilt with the prior distribution obtained at the previous sub-interval. The local tKRnet for $(T_{i-1}, T_i]$ is denoted by $\mathbf{T}^{(i)}(\boldsymbol{x}, t; \Theta^{(i,k)})$. The tKRnet solution of (3) (or (4)) at $t = T_{i-1}$ in this choice is defined as,

$$p_{\Theta}(\boldsymbol{x}, T_{i-1}) = p_0(\hat{\boldsymbol{x}}_0)\prod_{j=1}^{i-1} |\det \nabla_{\hat{\boldsymbol{x}}_{i-j}} \mathbf{T}^{(i-j)}(\hat{\boldsymbol{x}}_{i-j}, T_{i-j}; \Theta^{(i-j),*})|, \tag{25}$$

where

$$\hat{\boldsymbol{x}}_{i-1} = \boldsymbol{x}, \quad \hat{\boldsymbol{x}}_{i-j-1} = \mathbf{T}^{(i-j)}(\hat{\boldsymbol{x}}_{i-j}, T_{i-j}; \Theta^{(i-j),*}), \quad j = 1, \ldots, i-1. \tag{26}$$

In (25)–(26), $\{\Theta^{(i-j),*} | j = 1, \ldots, i-1\}$ denotes the set of trained parameters associated with local tKRnets for the previous sub-intervals. With $p_{\Theta}(\cdot, T_{i-1})$ defined in (25), the tKRnet solution for $t \in (T_{i-1}, T_i]$ is obtained through

$$p_{\Theta}(\boldsymbol{x}, t) = p_{\Theta}(\mathbf{T}^{(i)}(\boldsymbol{x}, t; \Theta^{(i,k)}), T_{i-1})|\det \nabla_{\boldsymbol{x}} \mathbf{T}^{(i)}(\boldsymbol{x}, t; \Theta^{(i,k)})|, \tag{27}$$

where $\Theta = \{\Theta^{(i,k)}, \Theta^{(i-j),*} | j = 1, \ldots, i-1\}$. Note that the trainable weights at this stage are $\Theta^{(i,k)}$ while the other parameters remain fixed, and $\mathbf{T}^{(i)}(\cdot, T_{i-1}; \Theta^{(i,k)})$ is an identity mapping, as $p_{\Theta}(\cdot, T_{i-1})$ is the prior distribution. As the tKRnet $\mathbf{T}^{(i)}$ implemented in this method is dependent on T_{i-1} and t, and the affine coupling layer (see (6)) is modified as

$$\begin{aligned}&\boldsymbol{x}_{\text{out}}^{(1)} = \boldsymbol{x}_{\text{in}}^{(1)}\\ &\boldsymbol{x}_{\text{out}}^{(2)} = \boldsymbol{x}_{\text{in}}^{(2)} + \frac{t-T_{i-1}}{T_i - T_{i-1}}\left(\alpha \boldsymbol{x}_{\text{in}}^{(2)} \odot \tanh(\boldsymbol{s}(\boldsymbol{x}_{\text{in}}^{(1)}, t, T_{i-1})) + e^{\boldsymbol{\beta}} \odot \tanh(\boldsymbol{n}(\boldsymbol{x}_{\text{in}}^{(1)}, t, T_{i-1}))\right),\end{aligned} \tag{28}$$

where $[\boldsymbol{s}, \boldsymbol{n}]^\top = \mathbf{N}(\boldsymbol{x}_{\text{in}}^{(1)}, t - T_{i-1}, T_{i-1})$. In scale-bias layers and nonlinear layers (presented in Sect. 3), the temporal variable t is replaced by $t - T_{i-1}$ in this setting. For example, scale-bias layers here are defined as

$$\boldsymbol{x}_{\text{out}} = e^{\tanh(\boldsymbol{\phi}(t-T_{i-1}))\odot \boldsymbol{a}} \odot \boldsymbol{x}_{\text{in}} + \tanh(\boldsymbol{\phi}(t-T_{i-1})) \odot \boldsymbol{b}. \tag{29}$$

After these slight modifications, Algorithm 1 can be implemented to obtain a local solution $p_\Theta(\boldsymbol{x}, t)$ (27) on each sub-interval $(T_{i-1}, T_i]$ for $i = 1, \dots, T_{sub}$.

4.3 Theoretical Properties

In this section, we show that the Kullback-Leibler (KL) divergence between the exact solution $p(\boldsymbol{x}, t)$ of (3) (or (4)) and the tKRnet solution $p_\Theta(\boldsymbol{x}, t)$ (see (16)) can be bounded by the residual $r_{\log}(\boldsymbol{x}, t; \Theta)$ (17).

Theorem 1 (Control of KL divergence) *Assume $p(\boldsymbol{x}, t) \to 0, p_\Theta(\boldsymbol{x}, t) \to 0$ as $\|\boldsymbol{x}\|_2 \to \infty$. Denote $\Omega_C = \{\boldsymbol{x} | \|\boldsymbol{x}\|_2 \le C\}$ with $C > 0$. For any $t \in [0, T]$, assume $\lim_{C\to\infty} \oint_{\partial\Omega_C} \log\left(\frac{p(\boldsymbol{x},t)}{p_\Theta(\boldsymbol{x},t)}\right) p(\boldsymbol{x}, t)\boldsymbol{f}(\boldsymbol{x}, t) \cdot \vec{\boldsymbol{n}}\mathrm{d}s = 0$, where $\vec{\boldsymbol{n}}$ is the outward pointing unit normal. The KL divergence between the exact solution $p(\boldsymbol{x}, t)$ of (3) (or (4)) and the tKRnet solution $p_\Theta(\boldsymbol{x}, t)$ (see (16)) satisfies*

$$\frac{\mathrm{d}}{\mathrm{d}t} D_{KL}(p(\boldsymbol{x}, t) || p_\Theta(\boldsymbol{x}, t)) \le \int_{\mathbb{R}^d} |r_{\log}(\boldsymbol{x}, t; \Theta)| p(\boldsymbol{x}, t)\mathrm{d}\boldsymbol{x}.$$

Proof The detailed proof is in Appendix 7.

Remark 1 In Theorem 1, we bound the KL divergence between $p(\boldsymbol{x}, t)$ and $p_\Theta(\boldsymbol{x}, t)$ using the residual $r_{\log}(\boldsymbol{x}, t; \Theta)$. The residual $r_{\log}(\boldsymbol{x}, t; \Theta)$ apparently depends on the modeling capability of $p_\Theta(\boldsymbol{x}, t)$. Recently some progress on the universal approximation property of invertible mapping has been achieved [20, 60]. For instance, it has been shown in [20] that the normalizing flow model based on real NVP [9] can serve as a universal approximator for an arbitrary PDF in the L_p sense with $p \in [1, \infty)$. Note that our model is a generalization of real NVP. Although tKRnet also has L_p-universality for PDF approximation, more efforts are needed to establish the convergence with respect to Sobolev norms for the approximation of PDEs, which is beyond the scope of this paper.

5 Numerical Experiments

To show the effectiveness of our proposed method presented in Sect. 4, the following five test problems are considered: the double gyre flow problem, the 3-dimensional Kraichnan-Orszag problem, the duffing oscillator, a 40-variable Lorenz-96 system, and a coupled oscillator. In our numerical studies, all trainable weights in affine coupling layers (see (6)) are initialized using the Kaiming uniform initialization [16], and the biases are set to zero. In each scale-bias layer (see (9)), parameters $\boldsymbol{a}$ and $\boldsymbol{b}$ are initialized as zero, and $\boldsymbol{\phi}$ is initially set to one. For the nonlinear layer (see (11)), the coefficients are initialized as $\hat{m} = 32$, $a = 50$, and ψ_i and φ_i are set to zero for $i = 0, \ldots, \hat{m}$ (see (11)). The AdamW optimizer [33] with a learning rate of 0.001 is used to solve (20), and a cosine annealing learning rate scheduler fine-tunes the learning rate. The training is conducted on an NVIDIA GTX 1080Ti GPU.

In order to assess the accuracy of the tKRnet solution $p_\Theta(\boldsymbol{x}, t)$ at time $t \in (0, T]$, we compare it with the reference solution $p(\boldsymbol{x}, t)$ of (3) as follows, which is computed using the method of characteristics. First, $N_v = 10^4$ initial states are sampled from the initial condition $p_0(\boldsymbol{x})$ of (3). For each initial state, the corresponding solution state of (2) at time t is computed using the ODE solver LSODA in SciPy, and the solution states are denoted by $\{\boldsymbol{x}_{\text{val}}^{(i)}\}_{i=1}^{N_v}$. Then, for each $t \in (0, T]$, the following relative error is computed

$$\frac{1}{N_v} \sum_{i=1}^{N_v} \frac{|p(\boldsymbol{x}_{\text{val}}^{(i)}, t) - p_\Theta(\boldsymbol{x}_{\text{val}}^{(i)}, t)|}{|p(\boldsymbol{x}_{\text{val}}^{(i)}, t)|}, \tag{30}$$

and the KL divergence between $p(\boldsymbol{x}, t)$ and $p_\Theta(\boldsymbol{x}, t)$ is estimated as

$$D_{KL}(p(\boldsymbol{x}, t) || p_\Theta(\boldsymbol{x}, t)) \approx \frac{1}{N_v} \sum_{i=1}^{N_v} \log \left(\frac{p(\boldsymbol{x}_{\text{val}}^{(i)}, t)}{p_\Theta(\boldsymbol{x}_{\text{val}}^{(i)}, t)} \right). \tag{31}$$

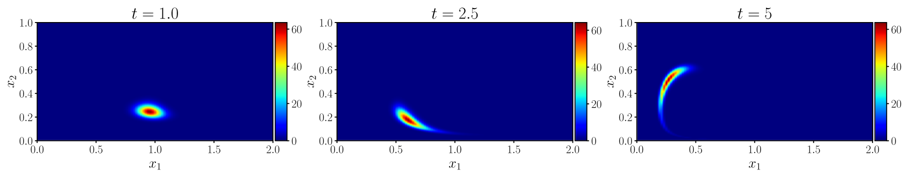
5.1 Double Gyre Flow

We start with the nonlinear time-dependent double gyre flow, which has a significant effect of nonlinearities for long-time integration [34]. The following nonlinear ODE system is considered

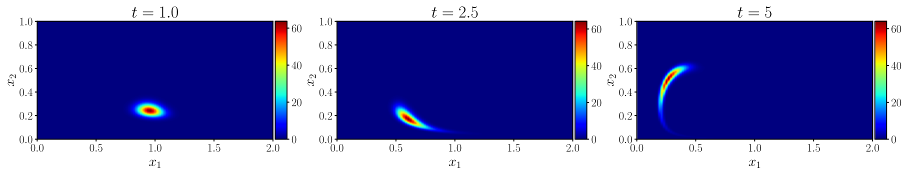
$$\begin{cases} \dfrac{\mathrm{d}x_1}{\mathrm{d}t} = -\pi A \sin(\pi f(x_1, t)) \cos(\pi x_2) \\ \dfrac{\mathrm{d}x_2}{\mathrm{d}t} = \pi A \cos(\pi f(x_1, t)) \sin(\pi x_2) \dfrac{\mathrm{d}f(x_1, t)}{\mathrm{d}x_1}, \end{cases} \tag{32}$$

where $f(x_1, t) = a(t)x_1^2 + b(t)x_1$, $a(t) = \varepsilon \sin(wt)$, $b(t) = 1 - 2\varepsilon \sin(wt)$. In this test problem, the coefficients are set to $A = 0.1$, $w = 2\pi/10$, and $\varepsilon = 0.25$. The time domain is set as $t \in (0, 5]$. The initial condition $p_0(\boldsymbol{x})$ in (3) is set to a Gaussian distribution $\mathcal{N}([1, 0.5]^\top, 0.05^2\mathbb{I})$. The tKRnet (13) has ten time-dependent affine coupling layers, ten scale-bias layers and one nonlinear layer. Each affine coupling layer includes one random Fourier feature layer and two fully connected layers which have thirty two neurons (see (7)). The time domain (0, 5] is discretized with step size $\Delta t = 0.02$, and the number of spatial collocation points is set to $M = 1000$ (see (22)). The parameters in Algorithm 1 are set as $N_r = 251{,}000$, $N_E = 100$, $N_{\text{adaptive}} = 6$, $N_b = 251$, and the initial spatial collocation points are generated through the uniform distribution with range $[0, 2] \times [0, 1]$.

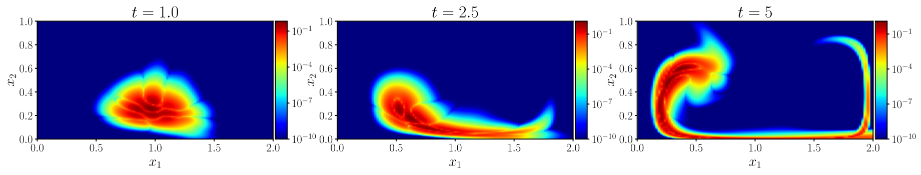
Figure 3a and b show the reference solution obtained using the method of characteristics and our tKRnet solution (obtained using Algorithm 1) at three discrete time steps $t = 1, 2.5, 5$ respectively. Figure 3c shows the point-wise absolute error across the spatial domain between the reference solution and the tKRnet solution. It can be seen that, the tKRnet solution and the reference solution are visually indistinguishable. From Fig. 3c, it is observed that the absolute error is around 1.254 range and the error in the spatial domain is locating at the high density region. The relative errors (defined in (30)) and the values of the KL divergence (defined in (31)) at iteration steps $k = 1, 3, 6$ (see line 4 of Algorithm 1) are shown in Fig. 4, where it is clear that the errors and the values of the KL divergence significantly reduce as



(a) Reference solution

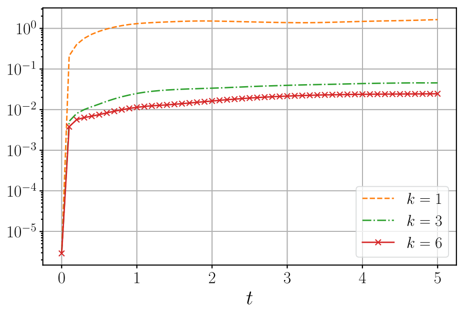


(b) tKRnet solution

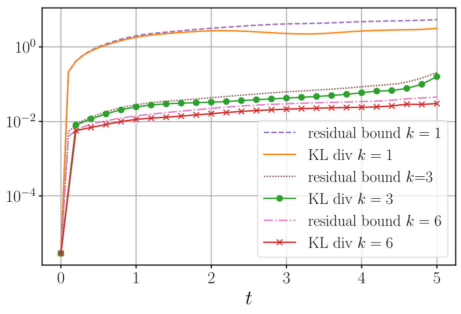


(c) Absolute error

Fig. 3 Double gyre flow problem for $t \in (0, 5]$: the reference solution and the tKRnet solution



(a) Relative absolute error



(b) KL divergence

Fig. 4 Double gyre flow problem for $t \in (0, 5]$: error and KL divergence values of the tKRnet solutions

the number of adaptivity iterations increases. In addition, the absolute values of the residual (17) are shown in Fig. 4b. It can be seen that, for each adaptivity iteration step ($k = 1, 3, 6$), the absolute value of the residual is slightly larger than the value of the KL divergence at each time step, which is consistent with Theorem 1. Training curves including the physics-informed residuals (defined in (19)) and the KL divergence at time $t = 5$ for adaptivity iteration steps $k = 1, 3, 6$ are shown in Fig. 5, where it is clear that the values of the residual and the KL divergence reduce as the number of adaptivity iterations increases.

To show the impact of the numbers of affine coupling layers and neurons on the performance of tKRnet, the following ablation study is conducted. Specifically, we test tKRnets with $L = 4, 10, 20$ affine coupling layers and scale-bias layers, and for each fully connected layer, we consider different numbers of neurons (the number is denoted d_h) with $d_h = 10, 32$

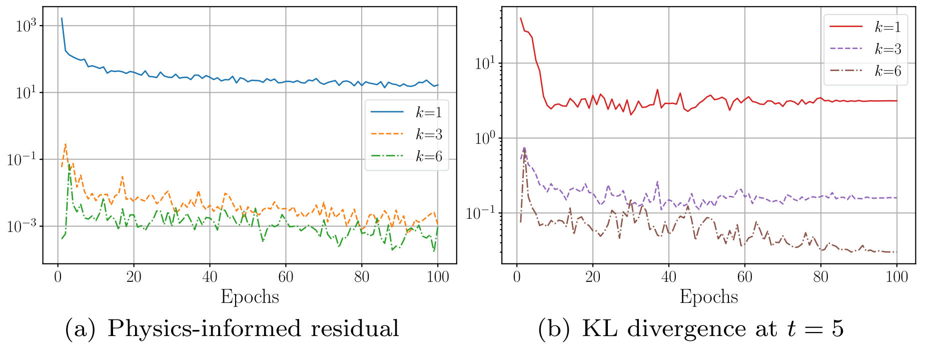


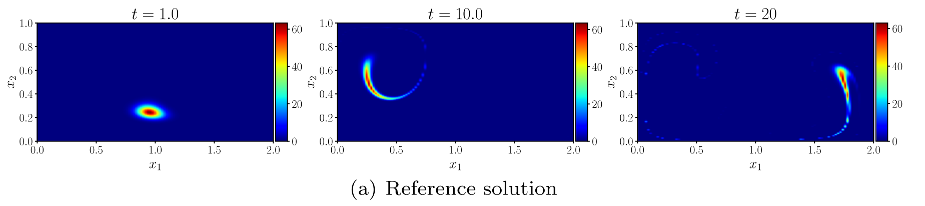
Fig. 5 Double gyre flow problem for $t \in (0, 5]$: physics-informed residuals and KL divergence values in the training procedure

Table 1 Double gyre flow problem for $t \in (0, 5]$: KL divergence values with respect to varying model capacity

	d_h=10	d_h=32	d_h=128
L=4	3.954×10^{-1}	4.049×10^{-1}	9.961×10^{-2}
L=10	7.975×10^{-2}	3.231×10^{-2}	2.216×10^{-2}
L=20	2.812×10^{-2}	2.807×10^{-2}	1.834×10^{-2}

and 128. Table 1 shows the KL divergence values (defined in (31)) at $t = 5$ with respect to tKRnets with different model capacities. It can be seen that the KL divergence reduces as the depth and the width of tKRnet increase.

Next, we keep the other settings of this test problem unchanged but extend the time domain to (0, 20], which results in a long-time integration problem. The reference solution and our tKRnet solution (directly obtained using Algorithm 1) at $t = 1, 10, 20$ are shown in Fig. 6. It can be seen that, directly applying Algorithm 1 to this long-time integration problem gives an inaccurate approximation, which is consistent with the challenges addressed in [53]. To resolve this problem, we apply the temporal decomposition method introduced in Sect. 4.2. Here, the interval (0, 20] is divided into ten equidistant sub-intervals. Each temporal sub-interval is discretized with time step size $\Delta t = 0.02$ (101 time steps), and the number of spatial collocation points is set to $M = 1000$. So, the total number of collocation points N_r for each sub-interval is $101{,}000 = 101 \times 1000$. For the first choice in Sect. 4.2, our tKRnet is trained with the loss function (23), and the prior distribution for the tKRnet is set to $\mathcal{N}([1, 0.5]^\top, 0.05^2\mathbb{I})$. For the second choice in Sect. 4.2 (see (27)), our tKRnet is trained with the loss function (19), where $r_{\log}$ is replaced by r defined (16) to result in an effective training procedure, and the nonlinear layer is not included. Figure 7 shows the results of the two choices for the temporal decomposition. Compared with the reference solution shown in Fig. 6a, both choices give efficient tKRnet approximations for this long-time integration problem.


(a) Reference solution

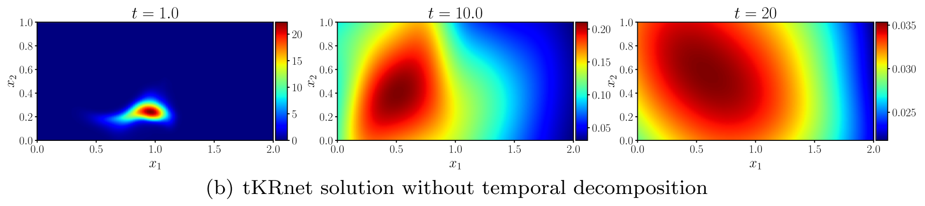
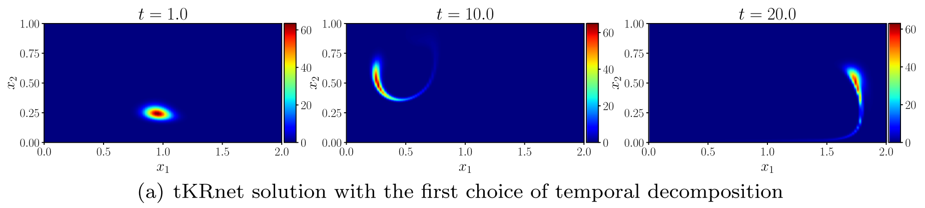

(b) tKRnet solution without temporal decomposition

Fig. 6 Double gyre flow problem for $t \in (0, 20]$: the reference solution and the tKRnet solution without temporal decomposition


(a) tKRnet solution with the first choice of temporal decomposition

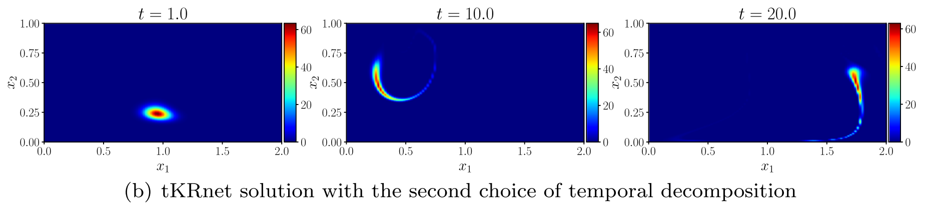

(b) tKRnet solution with the second choice of temporal decomposition

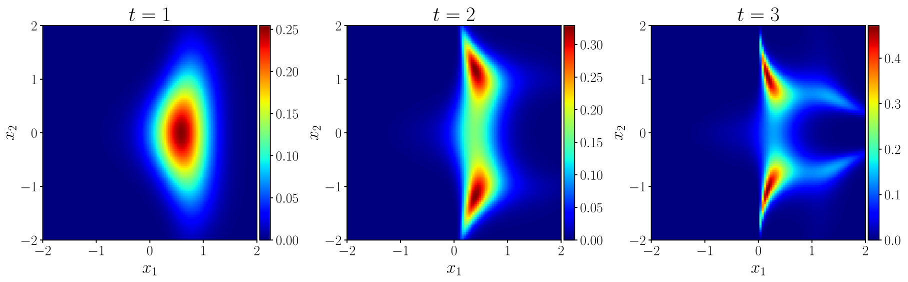
Fig. 7 Double gyre flow problem for $t \in (0, 20]$: tKRnet solutions with temporal decomposition

5.2 Kraichnan-Orszag

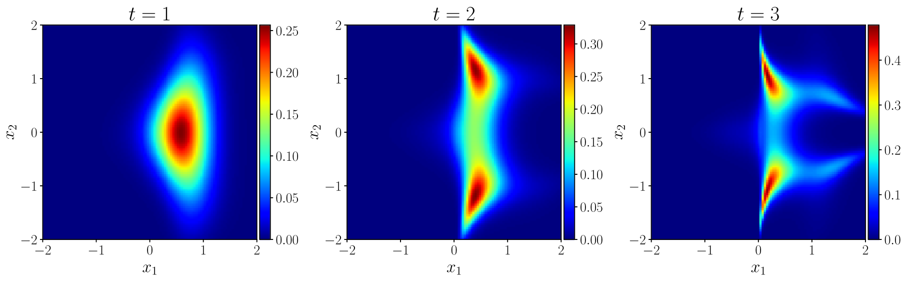
Here we consider the Kraichnan-Orszag problem [52],

$$\begin{cases} \frac{\mathrm{d}x_1}{\mathrm{d}t} = x_1 x_3 \\ \frac{\mathrm{d}x_2}{\mathrm{d}t} = -x_2 x_3 \\ \frac{\mathrm{d}x_3}{\mathrm{d}t} = -x_1^2 + x_2^2, \end{cases} \tag{33}$$

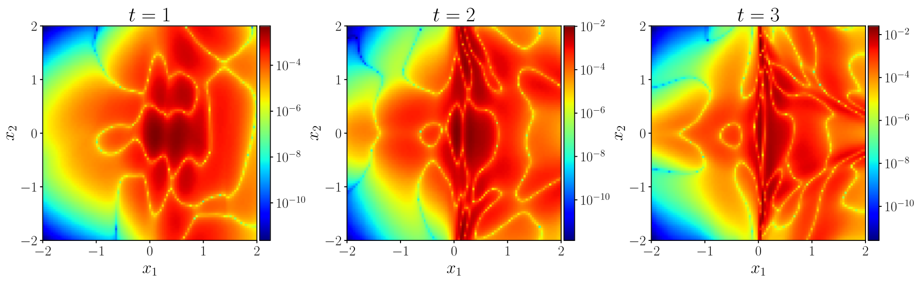
where x_1, x_2, x_3 are state variables. The initial condition $p_0(\boldsymbol{x})$ in (3) is set to a Gaussian distribution $\mathcal{N}([1, 0, 0]^\top, 0.5^2\mathbb{I})$, and the time domain in this test problem is set as $t \in (0, 3]$. The tKRnet (13) consists of $\mathbf{T}_{[1]}$, $\mathbf{T}_{[2]}$ and L_N, where each of $\mathbf{T}_{[1]}, \mathbf{T}_{[2]}$ has eight affine coupling layers and eight scale-bias layers. Each affine coupling layer includes one random Fourier layer and three fully connected layers with thirty two neurons (see (7)).



(a) Reference solution



(b) tKRnet solution



(c) Absolute error

Fig. 8 Kraichnan-Orszag problem: The reference solution and the tKRnet solution

The time domain is discretized with time step size $\Delta t = 0.01$, and the number of spatial collocation points is set to $M = 4000$ (see (22)). The settings in Algorithm 1 are set as $N_r = 1204000$, $N_E = 50$, $N_{\text{adaptive}} = 10$, $N_b = 1204$, and initial spatial collocation points are dawn from the uniform distribution with range $[-5, 5]^3$.

Figure 8 shows the reference solution, the tKRnet solution and the absolute error at $t = 1, 2, 3$, where it can be seen that the tKRnet solution and the reference solution are visually indistinguishable. Figure 8c shows the point-wise absolute error increases as time goes, and the maximum error at $t = 3$ is around 2.537×10^{-2}. The values of the relative error (30) and the KL divergence (31) at three adaptivity iterations $k = 1, 5, 10$ (see line 4 of Algorithm 1) are illustrated in Fig. 9. It is clear that the errors and the values of the KL divergence decrease as the number of adaptivity iteration steps increases. Figure 10 shows training curves including the physics-informed residual (defined in (19)) and the KL divergence at time $t = 3$ for the adaptivity iteration steps $k = 1, 3, 6$, where it is clear that the values of the residual and the KL divergence reduce as the number of adaptivity iterations increases.

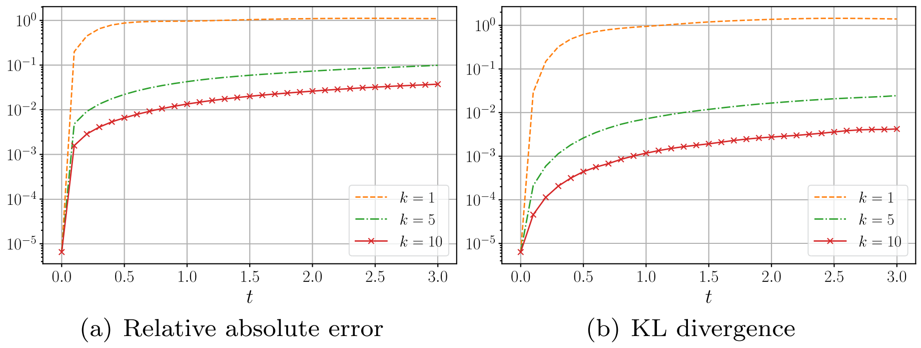


Fig. 9 Kraichnan-Orszag problem: error and KL divergence values of the tKRnet solutions

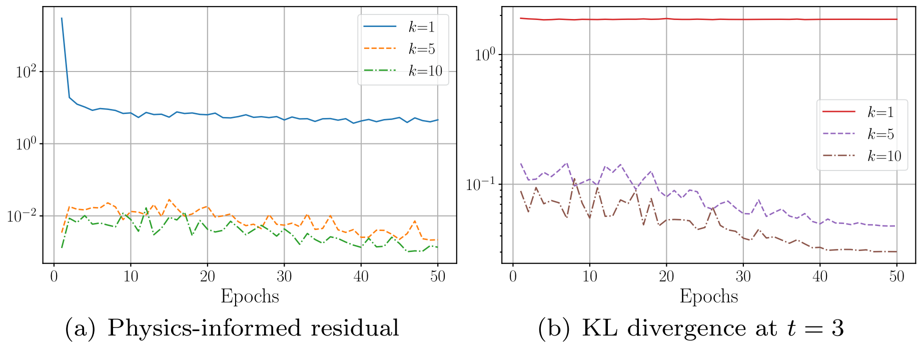


Fig. 10 Kraichnan-Orszag problem: physics-informed residuals and KL divergence values in the training procedure

5.3 Forced Duffing Oscillator

The forced Duffing oscillator system is defined as:

$$\begin{cases} \dfrac{\mathrm{d}y_1}{\mathrm{d}t} = y_2 \\ \dfrac{\mathrm{d}y_2}{\mathrm{d}t} = -\delta y_2 - y_1(\alpha + \beta y_1^2) + \gamma \cos(\omega t), \end{cases} \tag{34}$$

where y_1 and y_2 represent the state variables and $\delta, \alpha, \beta, \gamma$, and ω represent uncertain parameters. The time domain is set to $(0, 2]$. The initial distribution of the state variables $p_{\boldsymbol{y}}(\boldsymbol{y}, 0)$ and the distribution of the uncertain parameters $p_{\boldsymbol{\xi}}(\boldsymbol{\xi})$ in (3) are set as a Gaussian distribution $\mathcal{N}([0, 0]^\top, \mathbb{I})$ and a Gaussian distribution $\mathcal{N}([0.5, -1, 1, 0.5, 1]^\top, 0.25^2\mathbb{I})$ respectively. Letting $\boldsymbol{x} = [\boldsymbol{y}, \boldsymbol{\xi}]^\top$, the initial condition in (3) is constructed as $p_0(\boldsymbol{x}) = p_{\boldsymbol{y}}(\boldsymbol{y}, 0)p_{\boldsymbol{\xi}}(\boldsymbol{\xi})$. The tKRnet (13) consists of $\mathbf{T}_{[1]}, \mathbf{T}_{[2]}, \mathbf{T}_{[3]}$ and L_N, where each of $\mathbf{T}_{[1]}, \mathbf{T}_{[2]}$ and $\mathbf{T}_{[3]}$ has four affine coupling layers and four scale-bias layers. Each affine coupling layer has one random Fourier layer and two fully connected layers with thirty two neurons (see (7)). The coefficients in Algorithm 1 are set as $N_r = 804000$, $N_E = 100$, $N_{\text{adaptive}} = 6$, $N_b = 804$. The time domain is discretized with time step size $\Delta t = 0.01$, and the number of spatial collocation points is $M = 4000$ (see (22)). Initial spatial collocation points are sampled from the uniform distribution with range $[-5, 5]^7$.

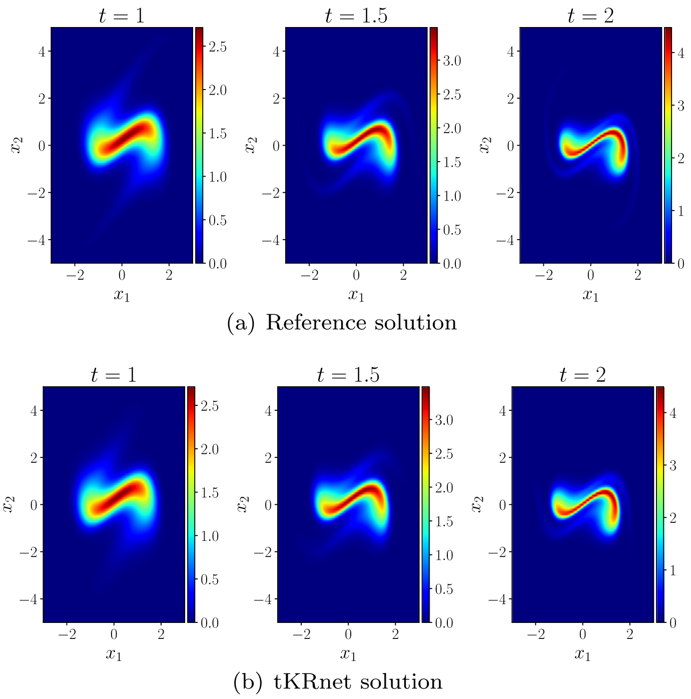


Fig. 11 Duffing oscillator problem: the reference solution and the tKRnet solution

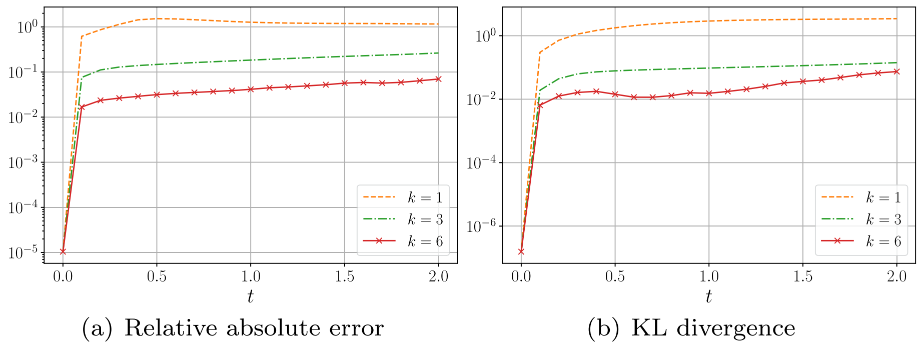


Fig. 12 Duffing oscillator problem: error and KL divergence values of the tKRnet solutions

Figure 11 shows the tKRnet solution and the reference solution at $t = 1, 1.5, 2$. From Fig. 11, it can be seen that the tKRnet solution and the reference solution are visually indistinguishable. The relative absolute errors (30) and the values of KL divergence (31) at three adaptivity iterations $k = 1, 3, 6$ (see line 4 of Algorithm 1) are illustrated in Fig. 12. It is clear that the errors and the values of KL divergence decrease as the number of adaptivity iterations increases.

5.4 Lorenz-96 System

In this test problem, the Lorenz-96 system is considered, which is a model used in numerical weather forecasting [22]. The general form of the Lorenz-96 system is defined as

$$\frac{\mathrm{d}x_i}{\mathrm{d}t} = (x_{i+1} - x_{i-2})x_{i-1} - x_i + F, \quad i = 1, \dots, d, \tag{35}$$

where x_i (for $i = -1, \dots, d+1$) are the state variables, and F represents a constant force. In this test problem, it is assumed that $d \geq 4$, $x_{-1} = x_{d-1}$, $x_0 = x_d$ and $x_{d+1} = x_1$. We set $d = 40$, $F = 1$, and $t \in (0, 1]$. The initial condition $p_0(\boldsymbol{x})$ in (3) is set to the joint Gaussian distribution

$$p_0([x_1, \dots, x_{40}]^\top) = \left(\frac{25}{2\pi}\right)^{20} \prod_{i=1}^{40} \exp\left(-\frac{25}{2}(x_i - (0.5 - |\frac{i}{40} - 0.5|))^2\right).$$

The tKRnet (13) for this problem has a sequence of transformations $\mathbf{T}_{[1]}, \dots, \mathbf{T}_{[5]}$ and one nonlinear layer L_N, where each $\mathbf{T}_{[1]}, \dots, \mathbf{T}_{[5]}$ includes four affine coupling layers and four scale-bias layers. Each affine coupling layer has one random Fourier layer and two fully connected layers with 128 neurons. The time domain is discretized with time step size $\Delta t = 0.01$, and the number of spatial collocation points is set to $M = 2000$ (see (22)). Parameters in Algorithm 1 are set as $N_r = 202{,}000$, $N_E = 50$, $N_{\text{adaptive}} = 10$, $N_b = 202$, and initial spatial collocation points are generated using the uniform distribution with range $[-5, 5]^{40}$.

For this high-dimensional problem, the mean and the variance estimates of the reference solution and the tKRnet solution are compared. For the reference solution, $N_v = 10^4$ initial states are sampled from $p_0(\boldsymbol{x})$, and the states $\{\boldsymbol{x}_{\text{val}}^{(i)}\}_{i=1}^{N_v}$ at time $t \in (0, T = 1]$ are obtained by solving (2) using the LSODA solver. The mean and the variance estimates are computed as

$$\widehat{\mathbb{E}}_{p(\boldsymbol{x},t)}[\boldsymbol{x}; t] := \frac{1}{N_v} \sum_{i=1}^{N_v} \boldsymbol{x}_{\text{val}}^{(i)}, \tag{36}$$

$$\widehat{\text{Var}}_{p(\boldsymbol{x},t)}[\boldsymbol{x}; t] := \frac{N_v}{N_v - 1} \left(\frac{1}{N_v} \sum_{i=1}^{N_v} (\boldsymbol{x}_{\text{val}}^{(i)})^2 - \left(\frac{1}{N_v} \sum_{i=1}^{N_v} \boldsymbol{x}_{\text{val}}^{(i)} \right)^2 \right). \tag{37}$$

For tKRnet approximation solution, for a given time $t \in (0, 1]$, samples of the states are generated by $p_\Theta(\boldsymbol{x}, t)$, and the mean and variance estimates are obtained by putting the samples into (36) and (37), which are denoted by $\widehat{\mathbb{E}}_{p_\Theta(\boldsymbol{x},t)}$ and $\widehat{\text{Var}}_{p_\Theta(\boldsymbol{x},t)}$ respectively. Figure 13 shows the mean and variance estimates of the reference solution and the tKRnet solution, where it can be seen that the results of the reference solution and those of the tKRnet solution are very close. Next, at time $t \in (0, 1]$, the errors in the mean and variance estimates are computed as

$$\left|\widehat{\mathbb{E}}_{p(\boldsymbol{x},t)}[\boldsymbol{x}; t] - \widehat{\mathbb{E}}_{p_\Theta(\boldsymbol{x},t)}[\boldsymbol{x}; t]\right|,$$
$$\left|\widehat{\text{Var}}_{p(\boldsymbol{x},t)}[\boldsymbol{x}; t] - \widehat{\text{Var}}_{p_\Theta(\boldsymbol{x},t)}[\boldsymbol{x}; t]\right|.$$

Figure 14 shows the errors, where it is clear that the errors are small—the maximum of the errors in the mean estimate is around 7.420×10^{-3} and that in the variance estimate is around 2.657×10^{-3}.

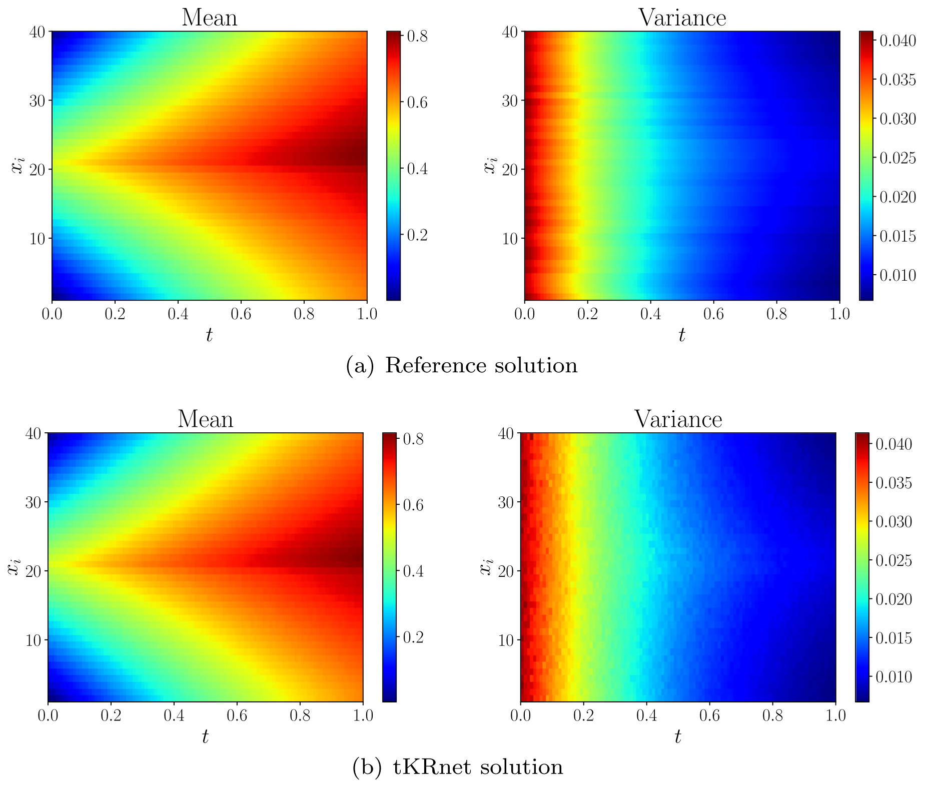


Fig. 13 Lorenz-96 problem: mean and variance of reference and tKRnet solutions

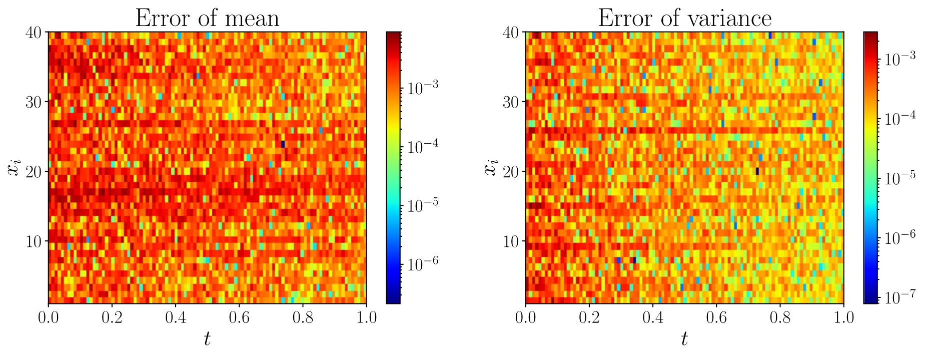


Fig. 14 Lorenz-96 problem: the absolute error to mean and variance in log scale

5.5 Coupled Oscillator

In this test problem, the following coupled oscillator system is considered:

$$\begin{cases} \dfrac{\mathrm{d}y_1}{\mathrm{d}t} = y_2 \\ \dfrac{\mathrm{d}y_2}{\mathrm{d}t} = -k_1 y_1 + k_2(y_3 - y_1) + k_3(y_5 - y_1) + f_1 \cos(t) \\ \dfrac{\mathrm{d}y_3}{\mathrm{d}t} = y_4 \\ \dfrac{\mathrm{d}y_4}{\mathrm{d}t} = -k_4 y_3 + k_2(y_1 - y_3) + k_5(y_5 - y_3) + f_2 \cos(t) \\ \dfrac{\mathrm{d}y_5}{\mathrm{d}t} = y_6 \\ \dfrac{\mathrm{d}y_6}{\mathrm{d}t} = -k_6 y_5 + k_3(y_1 - y_5) + k_5(y_3 - y_5) + f_3 \cos(t) \end{cases} \tag{38}$$

where $y_1, \ldots, y_6$ are the state variables, and $k_1, \ldots, k_6, f_1, f_2, f_3$ represent the uncertain parameters. In this system, k_2, k_3 and k_5 describe the interactions between state variables. The time domain is set to $(0, 2]$. The initial distribution of state variables $p_{\boldsymbol{y}}(\boldsymbol{y}, 0)$ is set as a Gaussian distribution $\mathcal{N}([0, 0, 2, 0, 3, 0]^\top, 0.1^2\mathbb{I})$, and the distribution of the uncertain parameters are set as

$$\mathcal{N}([0.1, 0.1, 0.1, 0.2, 0.1, 0.3, 1, 1, 1]^\top, 0.1^2\mathbb{I}).$$

Denoting $\boldsymbol{x} = [\boldsymbol{y}, \boldsymbol{\xi}]^\top$, the initial condition of (3) is $p_0(\boldsymbol{x}) = p_{\boldsymbol{y}}(\boldsymbol{y}, 0)p_{\boldsymbol{\xi}}(\boldsymbol{\xi})$. The tKRnet (13) consists of $\mathbf{T}_{[1]}$, $\mathbf{T}_{[2]}$ and L_N, where each of $\mathbf{T}_{[1]}$ and $\mathbf{T}_{[2]}$ has eight affine coupling layers and eight scale-bias layers. Each affine coupling layer has one random Fourier layer and two fully connected layers with thirty two neurons (see (7)). The coefficients in Algorithm 1 are set as $N_r = 101{,}000$, $N_E = 100$, $N_{\text{adaptive}} = 6$, $N_b = 101$. The time domain is discretized with time step size $\Delta t = 0.02$, and the number of spatial collocation points is $M = 1000$ (see (22)). Initial spatial collocation points are sampled from $p_0(\boldsymbol{x})$.

The relative absolute errors (30) and the values of KL divergence (31) at three adaptivity iterations $k = 1, 3, 6$ (see line 4 of Algorithm 1) are shown in Fig. 15. Again, it can be seen that the errors and the values of KL divergence decrease as the number of adaptivity iterations increases. Next, the upper bound of the KL divergence in Theorem 1 is computed by applying the trapezoidal rule to integrate the mean absolute value of the residual (17) over time domain $(0, 2]$. Specifically, the time domain $(0, 2]$ is discretized with a step size of 0.01, and 10^4 states at each time step computed using the LSODA solver, given the initial states drawn from $p_0(\boldsymbol{x})$. The mean absolute residual is then estimated using the Monte Carlo method with the computed states at each time step, and the trapezoidal rule is applied to estimate the integral of these mean absolute residual values over the time domain $(0, 2]$. Figure 16 shows the values of the upper bound, which are consistent of Theorem 1.

Kernel density estimation (KDE) [19] is considered for comparison. Specifically, we use the KDE function from the scikit-learn library [39], with a Gaussian kernel and a bandwidth parameter of 0.2, while retaining the default settings for all other parameters within scikit-learn. Here, 10^4 initial states are drawn from $p_0(\boldsymbol{x})$, and then the states at each time step t are computed by solving the coupled oscillator problem using the LSODA solver. Given the states at time $t \in (0, 2]$, KDE is applied to estimate the PDF of the states. The KL divergence values (31) for the PDF estimated by KDE (KDE solution) and the tKRnet solution are

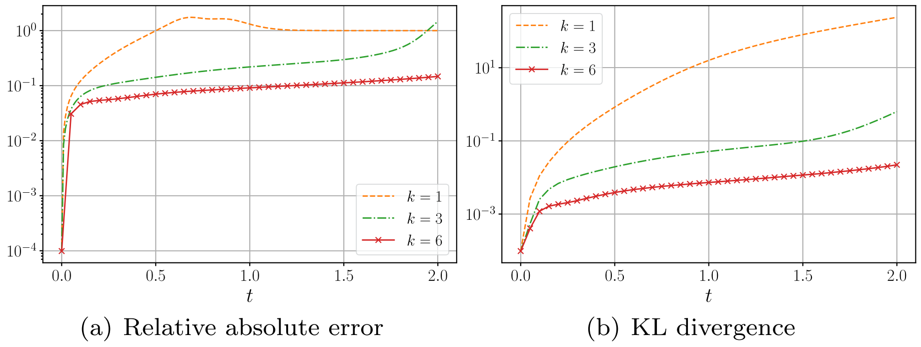


Fig. 15 Coupled oscillator problem: error and KL divergence values of the tKRnet solutions

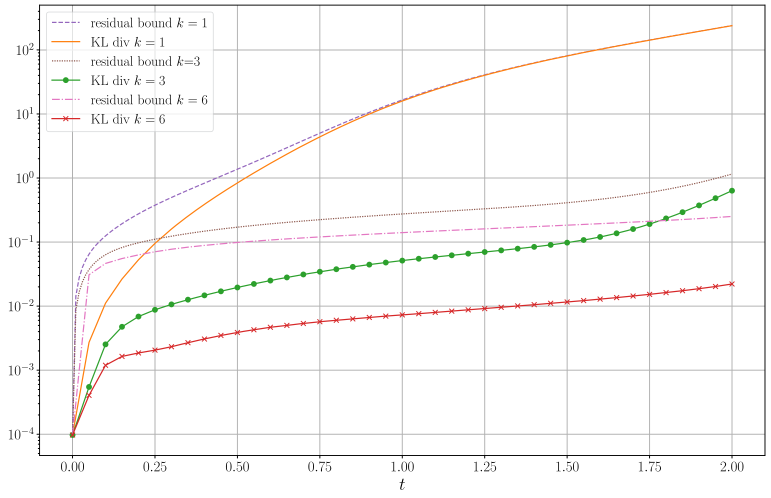


Fig. 16 Coupled oscillator problem: KL divergence values and the estimate bound

shown in Fig. 17, where it is clear that the our tKRnet solution is more accurate than the KDE solution.

6 Conclusions

The uncertainty quantification of stochastic dynamical systems can be addressed by computing the time-dependent PDF of the states. However, the states of the system can be high-dimensional and the support of the PDF may be unbounded. To address these issues, we have proposed a physics-informed adaptive density approximation method based on tKRnets to approximate the continuity equations. The tKRnet provides an explicit family of PDFs via the change of variable rule with a trainable time-dependent invertible transformation. The initial PDF of the continuity equation can be encoded in the tKRnet through the prior distribution. Adaptive sampling plays a crucial role in achieving an accurate PDF approximation, where the training set needs to be updated according to the localized information in the solution. By coupling the adaptive sampling with an efficient temporal decomposition,

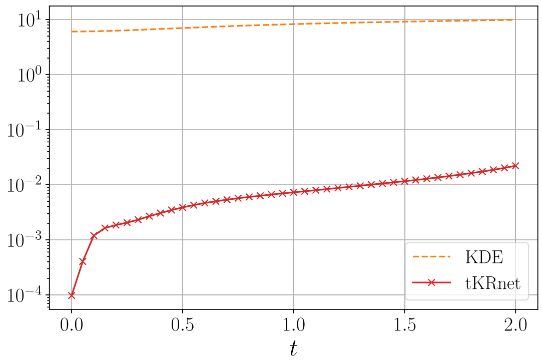


Fig. 17 Coupled oscillator problem: KL divergence values of the tKRnet solution and KDE solution

the long-time integration can be effectively improved. Numerical results have demonstrated the efficiency of our algorithm for high-dimensional stochastic dynamical systems. In this work, we use uniform grids to discretize the time interval without paying much attention to the causality in the time direction. Adaptivity can be introduced into temporal discretization for further refinement. This issue is being investigated and will be reported elsewhere.

7 Proof of Theorem 1

The proof of Theorem 1 is as follows, while a similar theoretical result is provided in [6] for Fokker–Planck equations.

Proof For KL divergence, we have

$$
\begin{aligned}
&\frac{\mathrm{d}}{\mathrm{d}t} D_{KL}(p(\boldsymbol{x},t)||p_{\Theta}(\boldsymbol{x},t)) \\
&\quad= \frac{\mathrm{d}}{\mathrm{d}t}\int_{\mathbb{R}^d} \log\left(\frac{p(\boldsymbol{x},t)}{p_{\Theta}(\boldsymbol{x},t)}\right) p(\boldsymbol{x},t)\mathrm{d}\boldsymbol{x} \\
&\quad= \int_{\mathbb{R}^d} \frac{\partial}{\partial t}\log\left(\frac{p(\boldsymbol{x},t)}{p_{\Theta}(\boldsymbol{x},t)}\right) p(\boldsymbol{x},t)\mathrm{d}\boldsymbol{x} + \int_{\mathbb{R}^d} \log\left(\frac{p(\boldsymbol{x},t)}{p_{\Theta}(\boldsymbol{x},t)}\right)\frac{\partial p(\boldsymbol{x},t)}{\partial t}\mathrm{d}\boldsymbol{x} \\
&\quad= \int_{\mathbb{R}^d} \frac{\partial p(\boldsymbol{x},t)}{\partial t}\mathrm{d}\boldsymbol{x} - \int_{\mathbb{R}^d} \frac{p(\boldsymbol{x},t)}{p_{\Theta}(\boldsymbol{x},t)}\frac{\partial p_{\Theta}(\boldsymbol{x},t)}{\partial t}\mathrm{d}\boldsymbol{x} + \int_{\mathbb{R}^d} \log\left(\frac{p(\boldsymbol{x},t)}{p_{\Theta}(\boldsymbol{x},t)}\right)\frac{\partial p(\boldsymbol{x},t)}{\partial t}\mathrm{d}\boldsymbol{x} \\
&\quad= \frac{\partial \int_{\mathbb{R}^d} p(\boldsymbol{x},t)\mathrm{d}\boldsymbol{x}}{\partial t} - \int_{\mathbb{R}^d}\left(\frac{p(\boldsymbol{x},t)}{p_{\Theta}(\boldsymbol{x},t)}\right)(r(\boldsymbol{x},t;\Theta) - \nabla_{\boldsymbol{x}}\cdot(p_{\Theta}(\boldsymbol{x},t)\boldsymbol{f}(\boldsymbol{x},t)))\mathrm{d}\boldsymbol{x} \\
&\qquad - \int_{\mathbb{R}^d} \log\left(\frac{p(\boldsymbol{x},t)}{p_{\Theta}(\boldsymbol{x},t)}\right)\nabla_{\boldsymbol{x}}\cdot(p(\boldsymbol{x},t)\boldsymbol{f}(\boldsymbol{x},t))\mathrm{d}\boldsymbol{x}.
\end{aligned}
$$

Since $\int_{\mathbb{R}^d} p(\boldsymbol{x},t)\mathrm{d}\boldsymbol{x} = 1$,

$$
\frac{\partial \int_{\mathbb{R}^d} p(\boldsymbol{x},t)\mathrm{d}\boldsymbol{x}}{\partial t} = \frac{\partial 1}{\partial t} = 0.
$$

Then,

$$
\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}t} D_{KL}(p(\boldsymbol{x},t)||p_\Theta(\boldsymbol{x},t)) = & \underbrace{\int_{\mathbb{R}^d} \frac{p(\boldsymbol{x},t)}{p_\Theta(\boldsymbol{x},t)} \nabla_{\boldsymbol{x}} \cdot (p_\Theta(\boldsymbol{x},t)\boldsymbol{f}(\boldsymbol{x},t))\mathrm{d}\boldsymbol{x}}_{I_1} \\
& - \underbrace{\int_{\mathbb{R}^d} \log\left(\frac{p(\boldsymbol{x},t)}{p_\Theta(\boldsymbol{x},t)}\right) \nabla_{\boldsymbol{x}} \cdot (p(\boldsymbol{x},t)\boldsymbol{f}(\boldsymbol{x},t))\mathrm{d}\boldsymbol{x}}_{I_2} \\
& - \int_{\mathbb{R}^d} \frac{r(\boldsymbol{x},t;\Theta)}{p_\Theta(\boldsymbol{x},t)} p(\boldsymbol{x},t)\mathrm{d}\boldsymbol{x}.
\end{aligned}
$$

For I_1, integration by parts yields that

$$
\begin{aligned}
& \int_{\mathbb{R}^d} \frac{p(\boldsymbol{x},t)}{p_\Theta(\boldsymbol{x},t)} \nabla_{\boldsymbol{x}} \cdot (p_\Theta(\boldsymbol{x},t)\boldsymbol{f}(\boldsymbol{x},t))\mathrm{d}\boldsymbol{x} \\
& \quad = \int_{\mathbb{R}^d} \nabla_{\boldsymbol{x}} \cdot (p(\boldsymbol{x},t)\boldsymbol{f}(\boldsymbol{x},t))\mathrm{d}\boldsymbol{x} - \int_{\mathbb{R}^d} \nabla_{\boldsymbol{x}} \left(\frac{p(\boldsymbol{x},t)}{p_\Theta(\boldsymbol{x},t)}\right) \cdot \boldsymbol{f}(\boldsymbol{x},t)p_\Theta(\boldsymbol{x},t)\mathrm{d}\boldsymbol{x} \\
& \quad = - \int_{\mathbb{R}^d} \nabla_{\boldsymbol{x}} \left(\frac{p(\boldsymbol{x},t)}{p_\Theta(\boldsymbol{x},t)}\right) \cdot \boldsymbol{f}(\boldsymbol{x},t)p_\Theta(\boldsymbol{x},t)\mathrm{d}\boldsymbol{x},
\end{aligned}
$$

where the second equality is obtained using

$$
\int_{\mathbb{R}^d} \nabla_{\boldsymbol{x}} \cdot (p(\boldsymbol{x},t)\boldsymbol{f}(\boldsymbol{x},t))\mathrm{d}\boldsymbol{x} = - \int_{\mathbb{R}^d} \frac{\partial p(\boldsymbol{x},t)}{\partial t}\mathrm{d}\boldsymbol{x} = 0.
$$

Similarly, I_2 can be rewritten as

$$
\begin{aligned}
& \int_{\mathbb{R}^d} \log\left(\frac{p(\boldsymbol{x},t)}{p_\Theta(\boldsymbol{x},t)}\right) \nabla_{\boldsymbol{x}} \cdot (p(\boldsymbol{x},t)\boldsymbol{f}(\boldsymbol{x},t))\mathrm{d}\boldsymbol{x} \\
& \quad = \int_{\mathbb{R}^d} \nabla_{\boldsymbol{x}} \cdot \left(\log\left(\frac{p(\boldsymbol{x},t)}{p_\Theta(\boldsymbol{x},t)}\right) p(\boldsymbol{x},t)\boldsymbol{f}(\boldsymbol{x},t)\right)\mathrm{d}\boldsymbol{x} \\
& \qquad - \int_{\mathbb{R}^d} \nabla_{\boldsymbol{x}} \log\left(\frac{p(\boldsymbol{x},t)}{p_\Theta(\boldsymbol{x},t)}\right) \cdot \boldsymbol{f}(\boldsymbol{x},t)p(\boldsymbol{x},t)\mathrm{d}\boldsymbol{x} \\
& \quad = \int_{\mathbb{R}^d} \nabla_{\boldsymbol{x}} \cdot \left(\log\left(\frac{p(\boldsymbol{x},t)}{p_\Theta(\boldsymbol{x},t)}\right) p(\boldsymbol{x},t)\boldsymbol{f}(\boldsymbol{x},t)\right)\mathrm{d}\boldsymbol{x} \\
& \qquad - \int_{\mathbb{R}^d} \nabla_{\boldsymbol{x}} \left(\frac{p(\boldsymbol{x},t)}{p_\Theta(\boldsymbol{x},t)}\right) \cdot \boldsymbol{f}(\boldsymbol{x},t)p_\Theta(\boldsymbol{x},t)\mathrm{d}\boldsymbol{x}.
\end{aligned}
$$

By the divergence theorem,

$$
\begin{aligned}
& \int_{\mathbb{R}^d} \nabla_{\boldsymbol{x}} \cdot \left(\log\left(\frac{p(\boldsymbol{x},t)}{p_\Theta(\boldsymbol{x},t)}\right) p(\boldsymbol{x},t)\boldsymbol{f}(\boldsymbol{x},t)\right)\mathrm{d}\boldsymbol{x} \\
& \quad = \lim_{C\to\infty} \oint_{\partial\Omega_C} \log\left(\frac{p(\boldsymbol{x},t)}{p_\Theta(\boldsymbol{x},t)}\right) p(\boldsymbol{x},t)\boldsymbol{f}(\boldsymbol{x},t) \cdot \vec{\boldsymbol{n}}\mathrm{d}s = 0.
\end{aligned}
$$

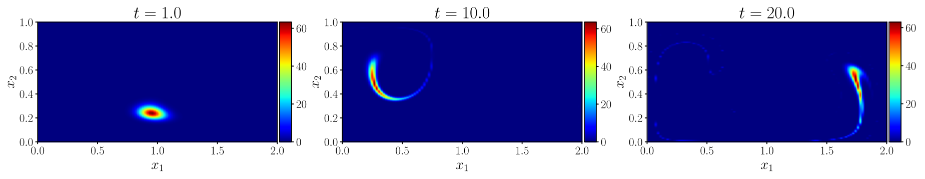


Fig. 18 ODE based approximation

Finally, we get

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t} D_{KL}(p(\boldsymbol{x},t)||p_{\Theta}(\boldsymbol{x},t)) &= -\int_{\mathbb{R}^d} \frac{r(\boldsymbol{x},t;\Theta)}{p_{\Theta}(\boldsymbol{x},t)} p(\boldsymbol{x},t)\mathrm{d}\boldsymbol{x} \\ &= -\int_{\mathbb{R}^d} r_{\log}(\boldsymbol{x},t;\Theta) p(\boldsymbol{x},t)\mathrm{d}\boldsymbol{x} \\ &\le \int_{\mathbb{R}^d} |r_{\log}(\boldsymbol{x},t;\Theta)| p(\boldsymbol{x},t)\mathrm{d}\boldsymbol{x}.\end{aligned}$$

□

8 Additional Training Results with ODE Residual

The PDEs (3) and (4) can be solved using the method of characteristics [36], where the characteristic lines evolve along the solution of the stochastic ODE system (2). Therefore, an alternative approach to learn the solution of (3) (or (4)) is to construct a time-dependent invertible mapping $\mathbf{T}$ (a deep neural network) to approximate the flow map for (2). We define $\boldsymbol{z} = \mathbf{T}(\boldsymbol{x}, t; \Theta)$ and its inverse $\boldsymbol{x} = \mathbf{T}^{-1}(\boldsymbol{z}, t; \Theta)$, where $\boldsymbol{z} \sim p_0$, $\boldsymbol{x}$ is the state variable in (2) and Θ is the parameters of the neural network $\mathbf{T}$. To ensure $\mathbf{T}^{-1}$ learns the solution of (2), the following residual is defined

$$\ell_{ode}(\boldsymbol{z}, t; \Theta) = \left\| \frac{\partial \mathbf{T}^{-1}(\boldsymbol{z}, t; \Theta)}{\partial t} - \boldsymbol{f}(\mathbf{T}^{-1}(\boldsymbol{z}, t; \Theta), t) \right\|_2^2, \text{ where } \boldsymbol{z} = \mathbf{T}(\boldsymbol{x}, t; \Theta);$$

the corresponding loss function is given by

$$\mathcal{L}_{ode}(\boldsymbol{z}, t; \Theta) = \frac{1}{N_r} \sum_{i=1}^{N_r} \ell_{ode}(\boldsymbol{z}_{\mathrm{res}}^{(i)}, t_{\mathrm{res}}^{(i)}; \Theta), \;\text{ where } \boldsymbol{z}_{\mathrm{res}}^{(i)} = \mathbf{T}(\boldsymbol{x}_{\mathrm{res}}^{(i)}, t_{\mathrm{res}}^{(i)}; \Theta), \tag{39}$$

where $\{\boldsymbol{x}_{\mathrm{res}}^{(i)}\}_{i=1}^{N_r}$ are spatial collocation points and $\{t_{\mathrm{res}}^{(i)}\}_{i=1}^{N_r}$ are temporal collocation points (see (22)). Then, the deep neural network $\mathbf{T}$ can be trained using Algorithm 1 with the loss function (39), and the ODE based approximation is constructed as $p_0(\mathbf{T}(\boldsymbol{x}, t; \Theta))|\det \nabla_{\boldsymbol{x}} \mathbf{T}(\boldsymbol{x}, t; \Theta)|$ using the trained neural network $\mathbf{T}(\boldsymbol{x}, t; \Theta)$, while the PDE based approximation is the approximate PDF by minimizing (19).

For the long-time integration problem considered in Sect. 5.1, the ODE based approximation and the PDE based approximation are compared as follows, where the second choice for temporal decomposition (introduced in Sect. 4.2) is applied to both approximations. All settings are the same as those in Sect. 5.1 for $t \in (0, 20]$. Figure 18 shows the ODE based approximation, which approximates the reference solution (Fig. 6(a)) well. Figure 19 shows relative errors (defined in (30)) of ODE and PDE approximations, where it is clear that the

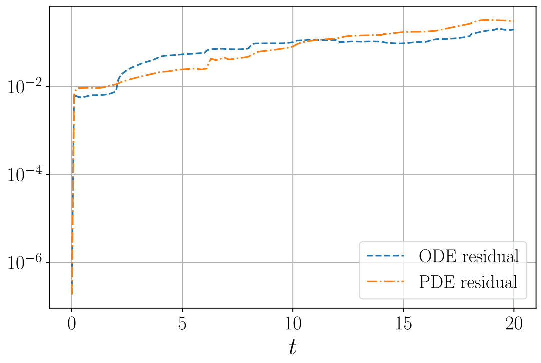


Fig. 19 Relative errors of ODE based approximation and PDE based approximation

relative errors of both approximations are comparable. However, as the training procedure with the loss function (39) requires computing $\partial \mathbf{T}^{-1}(z, t; \Theta)/\partial t$ with backpropagation, the cost for training the ODE based approximation is significantly larger than that for training the PDE based approximation, especially when the state is high-dimensional.

Funding The first two authors are supported by the National Natural Science Foundation of China (No. 12071291) and the Model Reduction Theory and Algorithms for Complex Systems Program of Institute of Mathematical Sciences, ShanghaiTech University (No. 2024X0303-902-01). The third author is supported by NSF grant DMS1913163.

Data Availability Data sharing not applicable to this article as no datasets were generated or analysed during the current study.

Declarations

Conflict of interest The authors have not disclosed any conflict of interest.

References

1. Ben-Hamu, H., Cohen, S., Bose, J., Amos, B., Nickel, M., Grover, A., Chen, R.T.Q., Lipman, Y.: Matching normalizing flows and probability paths on manifolds. In: Proceedings of the 39th International Conference on Machine Learning, pp. 1749–1763 (2022)
2. Brennan, C., Venturi, D.: Data-driven closures for stochastic dynamical systems. J. Comput. Phys. **372**, 281–298 (2018)
3. Carlier, G., Galichon, A., Santambrogio, F.: From Knothe's transport to Brenier's map and a continuation method for optimal transport. SIAM J. Math. Anal. **41**(6), 2554–2576 (2010)
4. Chen, T.Q., Rubanova, Y., Bettencourt, J., Duvenaud, D.: Neural ordinary differential equations. In: Advances in Neural Information Processing Systems 31, NeurIPS 2018, pp. 6572–6583 (2018)
5. Chen, Z., Churchill, V., Wu, K., Xiu, D.: Deep neural network modeling of unknown partial differential equations in nodal space. J. Comput. Phys. **449**, 110782 (2022)
6. Chewi, S.: Log-concave sampling. https://chewisinho.github.io/main.pdf (2024)
7. Cho, H., Venturi, D., Karniadakis, G.E.: Adaptive discontinuous Galerkin method for response-excitation PDF equations. SIAM J. Sci. Comput. **35**(4), B890–B911 (2013)
8. Cho, H., Venturi, D., Karniadakis, G.E.: Numerical methods for high-dimensional probability density function equations. J. Comput. Phys. **305**, 817–837 (2016)
9. Dinh, L., Sohl-Dickstein, J., Bengio, S.: Density estimation using Real NVP. In: 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24–26, 2017, Conference Track Proceedings. OpenReview.net (2017)
10. Dong, S., Li, Z.: Local extreme learning machines and domain decomposition for solving linear and nonlinear partial differential equations. Comput. Methods Appl. Mech. Eng. **387**, 114129 (2021)
11. E, W.: A proposal on machine learning via dynamical systems. Commun. Math. Stat. **1**(5), 1–11 (2017)

12. E, W., Yu, B.: The Deep Ritz Method: a deep learning-based numerical algorithm for solving variational problems. Commun. Math. Stat. **6**(1), 1–12 (2018)
13. Elman, H.C., Silvester, D.J., Wathen, A.J.: Finite Elements and Fast Iterative Solvers: With Applications in Incompressible Fluid Dynamics. Oxford University Press, Oxford (2014)
14. Feng, X., Zeng, L., Zhou, T.: Solving time dependent Fokker-Planck equations via temporal normalizing flow. Commun. Comput. Phys. **32**(2), 401–423 (2022)
15. Gao, H., Sun, L., Wang, J.X.: PhyGeoNet: physics-informed geometry-adaptive convolutional neural networks for solving parameterized steady-state pdes on irregular domain. J. Comput. Phys. **428**, 110079 (2021)
16. He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: surpassing human-level performance on imagenet classification. In: Proceedings of the IEEE international conference on computer vision, pp. 1026–1034 (2015)
17. Heinlein, A., Klawonn, A., Lanser, M., Weber, J.: Combining machine learning and domain decomposition methods for the solution of partial differential equations-a review. GAMM-Mitteilungen **44**(1) (2021)
18. Hendrycks, D., Gimpel, K.: Gaussian error linear units (gelus). arXiv preprint arXiv:1606.08415 (2016)
19. Härdle, W., Werwatz, A., Müller, M., Sperlich, S.: Nonparametric and Semiparametric Models. Springer, Berlin, Heidelberg (2004)
20. Ishikawa, I., Teshima, T., Tojo, K., Oono, K., Ikeda, M., Sugiyama, M.: Universal approximation property of invertible neural networks. J. Mach. Learn. Res. **24**, 1–68 (2023)
21. Jagtap, A.D., Kharazmi, E., Karniadakis, G.E.: Conservative physics-informed neural networks on discrete domains for conservation laws: applications to forward and inverse problems. Comput. Methods Appl. Mech. Eng. **365**, 113028 (2020)
22. Karimi, A., Paul, M.R.: Extensive chaos in the Lorenz-96 model. Chaos: Interdiscip. J. Nonlinear Sci. **20**(4), 043105 (2010)
23. Karniadakis, G.E., Kevrekidis, I.G., Lu, L., Perdikaris, P., Wang, S., Yang, L.: Physics-informed machine learning. Nat. Rev. Phys. **3**(6), 422–440 (2021)
24. Kharazmi, E., Zhang, Z., Karniadakis, G.E.: hp-VPINNs: variational physics-informed neural networks with domain decomposition. Comput. Methods Appl. Mech. Eng. **374**, 113547 (2021)
25. Kingma, D.P., Dhariwal, P.: Glow: Generative flow with invertible 1x1 convolutions. In: Advances in Neural Information Processing Systems 31, NeurIPS 2018, pp. 10236–10245 (2018)
26. Klyatskin, V.: Chapter 3 - indicator function and Liouville equation. In: Klyatskin, V. (ed.) Dynamics of Stochastic Systems, pp. 42–48. Elsevier Science, Amsterdam (2005)
27. Kobyzev, I., Prince, S.J., Brubaker, M.A.: Normalizing flows: an introduction and review of current methods. IEEE Trans. Pattern Anal. Mach. Intell. **43**(11), 3964–3979 (2021)
28. Li, J., Chen, J.: Stochastic Dynamics of Structures. Wiley, Hoboken (2009)
29. Li, K., Tang, K., Wu, T., Liao, Q.: D3M: a deep domain decomposition method for partial differential equations. IEEE Access **8**, 5283–5294 (2020)
30. Li, W., Xiang, X., Xu, Y.: Deep domain decomposition method: Elliptic problems. In: Mathematical and Scientific Machine Learning, pp. 269–286. PMLR (2020)
31. Lord, G.J., Powell, C.E., Shardlow, T.: An Introduction to Computational Stochastic PDEs, vol. 50. Cambridge University Press, Cambridge (2014)
32. Loshchilov, I., Hutter, F.: SGDR: Stochastic gradient descent with warm restarts. In: International Conference on Learning Representations (2017)
33. Loshchilov, I., Hutter, F.: Decoupled weight decay regularization. In: International Conference on Learning Representations (2019)
34. Luchtenburg, D.M., Brunton, S.L., Rowley, C.W.: Long-time uncertainty propagation using generalized polynomial chaos and flow map composition. J. Comput. Phys. **274**, 783–802 (2014)
35. Meng, X., Li, Z., Zhang, D., Karniadakis, G.E.: PPINN: parareal physics-informed neural network for time-dependent PDEs. Comput. Methods Appl. Mech. Eng. **370**, 113250 (2020)
36. Morton, K.W., Mayers, D.F.: Numerical Solution of Partial Differential Equations: An Introduction. Cambridge University Press, Cambridge (2005)
37. Moss, F., McClintock, P.V.: Noise in Nonlinear Dynamical Systems, vol. 1. Cambridge University Press, Cambridge (1989)
38. Papamakarios, G., Nalisnick, E., Rezende, D.J., Mohamed, S., Lakshminarayanan, B.: Normalizing flows for probabilistic modeling and inference. J. Mach. Learn. Res. **22**(57), 1–64 (2021)
39. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: machine learning in python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)

40. Penwarden, M., Jagtap, A.D., Zhe, S., Karniadakis, G.E., Kirby, R.M.: A unified scalable framework for causal sweeping strategies for physics-informed neural networks (PINNs) and their temporal decompositions. J. Comput. Phys. **493**, 112464 (2023)
41. Raissi, M., Perdikaris, P., Karniadakis, G.: Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. J. Comput. Phys. **378**, 686–707 (2019)
42. Risken, H.: The Fokker–Planck Equation: Methods of Solution and Applications, vol. 18. Springer, Berlin (1996)
43. Scott, D.W.: Multivariate Density Estimation: Theory, Practice, and Visualization. Wiley, Hoboken (2015)
44. Sheng, H., Yang, C.: PFNN: a penalty-free neural network method for solving a class of second-order boundary-value problems on complex geometries. J. Comput. Phys. **428**, 110085 (2021)
45. Sirignano, J., Spiliopoulos, K.: DGM: a deep learning algorithm for solving partial differential equations. J. Comput. Phys. **375**, 1339–1364 (2018)
46. Sobczyk, K.: Stochastic Differential Equations: With Applications to Physics and Engineering. Springer, Dordrecht (1991)
47. Tang, K., Wan, X., Liao, Q.: Deep density estimation via invertible block-triangular mapping. Theor. Appl. Mech. Lett. **10**(3), 143–148 (2020)
48. Tang, K., Wan, X., Liao, Q.: Adaptive deep density approximation for Fokker-Planck equations. J. Comput. Phys. **457**, 111080 (2022)
49. Tang, K., Wan, X., Yang, C.: DAS-PINNs: a deep adaptive sampling method for solving high-dimensional partial differential equations. J. Comput. Phys. **476**, 111868 (2023)
50. Tartakovsky, D.M., Gremaud, P.A.: Method of Distributions for Uncertainty Quantification, pp. 763–783. Springer, Cham (2017)
51. Villani, C.: Optimal Transport: Old and New. Springer, Berlin, Heidelberg (2009)
52. Wan, X., Karniadakis, G.E.: An adaptive multi-element generalized polynomial chaos method for stochastic differential equations. J. Comput. Phys. **209**(2), 617–642 (2005)
53. Wang, S., Perdikaris, P.: Long-time integration of parametric evolution equations with physics-informed DeepONets. J. Comput. Phys. **475**, 111855 (2023)
54. Wang, S., Wang, H., Perdikaris, P.: On the eigenvector bias of Fourier feature networks: from regression to solving multi-scale PDEs with physics-informed neural networks. Comput. Methods Appl. Mech. Eng. **384**, 113938 (2021)
55. Wang, Y., Cheung, S.W., Chung, E.T., Efendiev, Y., Wang, M.: Deep multiscale model learning. J. Comput. Phys. **406**, 109071 (2020)
56. Wang, Z., Zhang, Z.: A mesh-free method for interface problems using the deep learning approach. J. Comput. Phys. **400**, 108963 (2020)
57. Wu, K., Xiu, D.: Data-driven deep learning of partial differential equations in modal space. J. Comput. Phys. **408**, 109307 (2020)
58. Xu, Z., Liao, Q., Li, J.: Domain-decomposed Bayesian inversion based on local Karhunen–Loève expansions. J. Comput. Phys. 112856 (2024)
59. Zang, Y., Bao, G., Ye, X., Zhou, H.: Weak adversarial networks for high-dimensional partial differential equations. J. Comput. Phys. **411**, 109409 (2020)
60. Zhu, A., Jin, P., Tang, Y.: Approximation capabilities of measure-preserving neural networks. Neural Netw. **147**, 72–80 (2022)
61. Zhu, Y., Zabaras, N.: Bayesian deep convolutional encoder-decoder networks for surrogate modeling and uncertainty quantification. J. Comput. Phys. **366**, 415–447 (2018)
62. Zhu, Y., Zabaras, N., Koutsourelakis, P.S., Perdikaris, P.: Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. J. Comput. Phys. **394**, 56–81 (2019)