Journal of Scientific Computing (2023) 97:68
https://doi.org/10.1007/s10915-023-02379-z

Adaptive Deep Density Approximation for Fractional Fokker–Planck Equations

Li Zeng[1,2] · **Xiaoliang Wan**[3] · **Tao Zhou**[1]

Received: 11 November 2022 / Revised: 21 September 2023 / Accepted: 11 October 2023

Abstract
In this work, we propose adaptive deep learning approaches based on normalizing flows for solving fractional Fokker–Planck equations (FPEs). The solution of a FPE is a probability density function (PDF). Traditional mesh-based methods are ineffective because of a unbounded computation domain, a large number of dimensions and a nonlocal fractional operator. To this end, we represent the solution with an explicit PDF model induced by a flow-based deep generative model, which constructs a transport map from a simple distribution to the target distribution. We consider two methods to approximate the fractional Laplacian. One method is the Monte Carlo approximation. The other method is to construct an auxiliary model with Gaussian radial basis functions (GRBFs) to approximate the solution such that we may take advantage of the fact that the fractional Laplacian of a Gaussian is known analytically. Based on these two different ways for the approximation of the fractional Laplacian, we propose two models to approximate stationary FPEs and one model to approximate time-dependent FPEs. To further improve the accuracy, we refine the training set and the approximate solution alternately. A variety of numerical examples is presented to demonstrate the effectiveness of our adaptive deep density approaches.

Keywords Fractional Fokker–Planck equation · Normalizing flow · Adaptive density approximation · Monte Carlo sampling · Gaussian radial basis functions

Mathematics Subject Classification 65M75 · 65C30 · 68T07

✉ Li Zeng
zengli@lsec.cc.ac.cn

Xiaoliang Wan
xlwan@lsu.edu

Tao Zhou
tzhou@lsec.cc.ac.cn

[1] LSEC, Institute of Computational Mathematics and Scientific/Engineering Computing, AMSS, Chinese Academy of Sciences, Beijing, China

[2] School of Mathematics and Statistics, Fuzhou University, Fuzhou, China

[3] Department of Mathematics and Center for Computation and Technology, Louisiana State University, Baton Rouge 70803, USA

Published online: 03 November 2023

1 Introduction

The fractional Fokker–Planck equations (FPEs) describe the time evolution of the probability density function of particles driven by Lévy noise as well as Gaussian noise. Compared to integer-order FPEs whose associated stochastic differential equations (SDEs) are only driven by Gaussian noise, the fractional FPEs have a much wider range of applications in physics, biology, and other fields [9, 10, 32] since more than one kind of noise is often needed to simulate complex systems in practice. However, it is very challenging to approximate the fractional FPEs due to the following four obstacles:

(i) The solution is a probability density function requiring vanishing boundary condition, normality, and non-negativity.
(ii) The computational domain may be unbounded.
(iii) The fractional Laplacian operator is nonlocal.
(iv) The problem may have a large number of dimensions.

Traditional methods such as finite difference method, finite element method, spectral method as well as path integral method [1, 6, 12, 40, 46] have been applied to approximate fractional FPEs. Most of these methods are limited to problems of dimension one or two because the mesh-based discretization of high-dimensional problems induces unaffordable computational cost. On the other hand, simulating the SDEs associated with the FPEs [44] needs a large number of sample paths. Thus more efficient methods are still needed to approximate the fractional FPEs.

Recently, deep learning techniques have shown strong vitality in solving PDEs, e.g. deep Galerkin method [33], deep Ritz method [39] and physics-informed neural networks (PINNs) [27]. These techniques have gained encouraging performance in many applications [2, 17, 18, 23, 26, 28, 41, 45, 48]. Meanwhile, many deep generative models such as generative adversarial networks (GANs) [14], variational autoencoder (VAE) [21] and normalizing flow (NF) [25, 29] have been successfully applied to learn forward and inverse SDEs [5, 22, 43, 49]. For instance, a physics-informed generative adversarial model was proposed in [42] to tackle high-dimensional SDEs. In [16], a normalizing field flow was developed to build surrogate models for uncertainty quantification problems. The key issue of these methods is to convert the PDE problem into an optimization problem constrained to physical laws where the loss function is discretized by random training points. The training points here refer to space-time collocation points where the equations are enforced through optimization. The choice of training points will significantly affect the final numerical accuracy especially for unbounded problems. An adaptive sampling procedure was proposed in [11, 36] to solve integer-order FPEs, where the training set is updated by the current approximate solution which will be subsequently improved by the new training set. We will employ a similar adaptive procedure to deal with fractional FPEs.

To alleviate the difficulties induced by the constraints of a probability density function (PDF), we consider an explicit PDF model given by the normalizing flow. A normalizing flow constructs an invertible mapping from a simple distribution to the target distribution and results in an explicit PDF through the change of variable. We represent the solution of the FPE via a normalizing flow. In particular, we employ KRnet [35], which has been successfully applied to estimate high-dimensional density function and to approximate integer-order FPEs [11, 36].

Since KRnet yields a PDF explicitly, the first difficulty is avoided naturally. What's more, as a generative model, KRnet can generate exact random samples efficiently, which resolves the second obstacle because the commonly used uniform samples cannot be applied to an

unbounded domain and are not effective for a large truncated domain. Using KRnet, we may update the training points by new samples from the current KRnet which automatically generates more samples in the region of high density. It is well known that automatic differentiation brings great convenience to the approximation of PDEs. However, it only works for the computation of integer-order derivatives. An effective method is needed to tackle the fractional derivatives. Several approaches have been developed to discretize the fractional derivatives when the PDE solution is modeled by neural networks. For example, a finite difference method is applied in [24], and a direct Monte Carlo sampling approach was proposed in [15]. In [3], Gaussian radial basis functions (GRBFs) were used to represent the solution of fractional PDEs based on the fact that the fractional Laplacian of GRBFs can be derived analytically. In this work, we will employ either the Monte Carlo sampling approach or auxiliary GRBFs to address the fractional Laplacian operator in nonlocal FPEs.

Integrating the PDF model from KRnet, automatic differentiation for integer-order derivatives and Monte Carlo sampling/GRBFs approach for fractional Laplacian, we have developed two effective deep learning techniques to address the approximation of nonlocal FPEs without requiring any labeled data. Following are the main features of our approaches:

- Our approaches are based on the explicit PDF model given by KRnet, which satisfies naturally all the constraints of a PDF. They are different from work [47] which handles the constraints via adding penalty terms to the loss function.
- Our approaches are extensions to the previous work [11, 36] where only FPEs with integer-order derivatives are investigated. We have paid particular attention to how to improve both the accuracy and efficiency when the fractional derivatives are involved.
- Being machine learning schemes, the proposed approaches are mesh-free and can be easily applied to high dimensional problems.

The remainder of this paper is structured as follows. In Sect. 2, we present a brief description of the fractional FPEs. Section 3 provides an adaptive density approximation scheme for stationary fractional FPEs. In Sect. 4, we generalize the approach to deal with time-dependent fractional FPEs. We demonstrate the effectiveness and efficiency of our adaptive sampling approaches with several numerical experiments in Sect. 5 followed by some concluding remarks in Sect. 6.

2 Problem Setup

The main aim of this work is to solve the fractional FPEs. We first give a brief introduction to the fractional FPEs.

2.1 Fractional Fokker–Planck Equations

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

$$\mathrm{d}\boldsymbol{X}_t = \boldsymbol{\mu}(\boldsymbol{X}_t, t)\,\mathrm{d}t + \boldsymbol{\sigma}(\boldsymbol{X}_t, t)\,\mathrm{d}\boldsymbol{W}_t + \mathrm{d}\boldsymbol{L}_t^{\alpha}, \tag{2.1}$$

where $\boldsymbol{X}_t$ and $\boldsymbol{\mu}(\boldsymbol{X}_t, t)$ are d-dimensional random vectors, $\boldsymbol{\sigma}(\boldsymbol{X}_t, t)$ is a $d \times M$ matrix, $\boldsymbol{W}_t$ is an M-dimensional standard Wiener process and $\boldsymbol{L}_t^{\alpha}$ is a α-stable Levy motion with $\alpha \in (0, 2)$. The probability density function (PDF) $p(\boldsymbol{x}, t)$ for $\boldsymbol{X}_t$ satisfies the time-dependent FPE:

$$\frac{\partial p}{\partial t} = \mathcal{L}p - (-\Delta)^{\alpha/2} p, \tag{2.2}$$

where

$$\mathcal{L}p = -\nabla \cdot (p\boldsymbol{\mu}) + \frac{1}{2}\nabla \cdot \nabla \cdot (\boldsymbol{\sigma}\boldsymbol{\sigma}^{\mathrm{T}} p), \tag{2.3}$$

is induced by the drift and the diffusion, and the following nonlocal Laplacian operator

$$(-\Delta)^{\alpha/2} p = C_{d,\alpha}\ \mathrm{P.V.} \int_{\mathbb{R}^d} \frac{p(\boldsymbol{x}) - p(\boldsymbol{y})}{|\boldsymbol{x} - \boldsymbol{y}|_2^{d+\alpha}} \mathrm{d}\boldsymbol{y}, \tag{2.4}$$

is induced by the Levy motion, where $|\cdot|_2$ indicates the ℓ_2 norm of a vector and P.V. denotes the principle value of the integral and $C_{d,\alpha}$ is a constant given by

$$C_{d,\alpha} = \frac{2^{\alpha-1}\alpha\Gamma(\frac{\alpha+d}{2})}{\pi^{d/2}\Gamma(1-\alpha/2)}, \tag{2.5}$$

with $\Gamma(\cdot)$ being the gamma function.

In general, Eq. (2.2) is defined on $\mathbb{R}^d$ with the following boundary condition

$$p(\boldsymbol{x}) \to 0 \quad \text{as} \quad |\boldsymbol{x}|_2 \to \infty. \tag{2.6}$$

Furthermore, the solution as a probability density function should be conservative and non-negative, i.e.,

$$\int_{\mathbb{R}^d} p(\boldsymbol{x}, t)\mathrm{d}\boldsymbol{x} \equiv 1, \quad \text{and} \quad p(\boldsymbol{x}, t) \geq 0. \tag{2.7}$$

In this work, we first address the numerical approximation of Eq. (2.2) when $\partial_t p = 0$, i.e.,

$$(\mathcal{L} - (-\Delta)^{\alpha/2})p = 0, \tag{2.8}$$

and then consider the time-dependent FPE, i.e., $\partial_t p \neq 0$.

3 Stationary Fractional FPE

3.1 A Bird's-Eye View of Proposed Approaches

As mentioned in the introduction, we resort to deep generative modeling to construct an explicit PDF model on $\mathbb{R}^d$ to remove all the constraints of a PDF, which also alleviates the curse of dimensionality. Depending on how to approximate the fractional Laplacian operator, we will develop two approaches to solve the fractional FPE (see Table 1). In MCNF, we approximate the fractional Laplacian by the Monte Carlo method. While in GRBNF, we introduce an auxiliary model to represent the approximate solution with Gaussian radial basis functions such that we may take advantage of the fact that the fractional Laplacian of a Gaussian is known explicitly. As for the time-dependent fractional FPEs, temporal KRnet is considered as in [11], see Sect. 4 for the definition of MCTNF.

3.1.1 MCNF

Assume that the unknown PDF $p(\boldsymbol{x})$ is modeled by KRnet as $p_{\mathrm{KRnet},\boldsymbol{\theta}}$ which will be specified in Sect. 3.2. We adopt the idea of physics-informed neural network to handle Eq. (2.8), where the overall residuals of Eq. (2.8) at some prescribed collocation points within the computation

Table 1 GRBF and MC indicate how to deal with the fractional Laplacian operator. NF indicates how to obtain a solution model

Notations	Methods
GRBFNF	Gaussian radial basis function (GRBF) + Normalizing flow (NF)
MCNF	Monte Carlo (MC) sampling + Normalizing flow (NF)
MCTNF	Monte Carlo (MC) sampling + Temporal normalizing flow (TNF)

domain will be minimized. For the given training data $S = \{\boldsymbol{x}^i\}_{i=1}^{N_S}$, we define the following loss function,

$$L(p_{\mathrm{KRnet},\boldsymbol{\theta}}) := \frac{1}{N_S}\sum_{i=1}^{N_S} |R_{\boldsymbol{\theta}}(\boldsymbol{x}^i)|^2, \tag{3.1}$$

where $R_{\boldsymbol{\theta}}(\boldsymbol{x})$ is the residual

$$R_{\boldsymbol{\theta}}(\boldsymbol{x}) := (\mathcal{L} - (-\Delta)^{\alpha/2}) p_{\mathrm{KRnet},\boldsymbol{\theta}}(\boldsymbol{x}). \tag{3.2}$$

The optimal parameters $\boldsymbol{\theta}^*$ is given by the following optimization problem

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} L(p_{\mathrm{KRnet},\boldsymbol{\theta}}). \tag{3.3}$$

The stochastic approximation proposed in [15] is used to compute the fractional Laplacian of $p_{\mathrm{KRnet},\boldsymbol{\theta}}$, which will be specified in Sect. 3.3. Another key component of our approach is the adaptive improvement of $p_{\mathrm{KRnet},\boldsymbol{\theta}}$ (see Sect. 3.5), wherein the training set S is updated with samples from the current optimal model $p_{\mathrm{KRnet},\boldsymbol{\theta}^*}$ that will be subsequently improved by the new training set. When the convergence is reached, we expect that the samples in S will be distributed in terms of the exact solution $p(\boldsymbol{x})$.

3.1.2 GRBFNF

We rewrite Eq. (2.8) as

$$\begin{cases} \mathcal{L} p_{\mathrm{KRnet},\boldsymbol{\theta}}(\boldsymbol{x}) = (-\Delta)^{\alpha/2} p_{\mathrm{GRBF},\tilde{\boldsymbol{\theta}}}(\boldsymbol{x}), \\ p_{\mathrm{KRnet},\boldsymbol{\theta}}(\boldsymbol{x}) = p_{\mathrm{GRBF},\tilde{\boldsymbol{\theta}}}(\boldsymbol{x}), \end{cases} \tag{3.4}$$

where $p_{\mathrm{KRnet},\boldsymbol{\theta}}(\boldsymbol{x})$ is the same as the model used for MCNF and $p_{\mathrm{GRBF},\tilde{\boldsymbol{\theta}}}(\boldsymbol{x})$ is an auxiliary model for $p(\boldsymbol{x})$ (see Sect. 3.4). In other words,

$$p(\boldsymbol{x}) \approx p_{\mathrm{KRnet}\boldsymbol{\theta}}(\boldsymbol{x}), \quad p(\boldsymbol{x}) \approx p_{\mathrm{GRBF},\tilde{\boldsymbol{\theta}}}(\boldsymbol{x}).$$

For a set $S = \{\boldsymbol{x}^i\}_{i=1}^{N_S}$ of collocations points within the computation domain, we consider the following optimization problem:

$$(\boldsymbol{\theta}^*, \tilde{\boldsymbol{\theta}}^*) = \arg\min_{\boldsymbol{\theta},\tilde{\boldsymbol{\theta}}} \tilde{L}(p_{\mathrm{KRnet},\boldsymbol{\theta}}, p_{\mathrm{GRBF},\tilde{\boldsymbol{\theta}}}), \tag{3.5}$$

where the tuple $(\boldsymbol{\theta}^*, \tilde{\boldsymbol{\theta}}^*)$ is the minimizer of the loss function defined as

$$\begin{aligned}\tilde{L}(p_{\mathrm{KRnet},\boldsymbol{\theta}}, p_{\mathrm{GRBF},\tilde{\boldsymbol{\theta}}}) = &\frac{1}{N_S}\sum_{i=1}^{N_S}\left(\mathcal{L}p_{\mathrm{KRnet},\boldsymbol{\theta}}(\boldsymbol{x}^i) - (-\Delta)^{\alpha/2}p_{\mathrm{GRBF},\tilde{\boldsymbol{\theta}}}(\boldsymbol{x}^i)\right)^2 \\ &+ \frac{\beta_m}{N_S}\sum_{i=1}^{N_S}\left(p_{\mathrm{KRnet},\boldsymbol{\theta}}(\boldsymbol{x}^i) - p_{\mathrm{GRBF},\tilde{\boldsymbol{\theta}}}(\boldsymbol{x}^i)\right)^2,\end{aligned} \tag{3.6}$$

with $0 < \beta_m < \infty$ being a penalty parameter. The main difference of GRBFNF from MCNF is the introduction of the auxiliary model $p_{\mathrm{GRBF},\tilde{\boldsymbol{\theta}}}(\boldsymbol{x})$, which will be mainly used to simplify the computation of the fractional Laplacian. More specifically, $p_{\mathrm{GRBF},\tilde{\boldsymbol{\theta}}}(\boldsymbol{x})$ is a linear combination of the Gaussian radial basis functions with centers $\tilde{\boldsymbol{x}}_i \in S_{\mathrm{center}}$, which corresponds to a neural network with one hidden layer. The fractional Laplacian of $p_{\mathrm{GRBF},\tilde{\boldsymbol{\theta}}}(\boldsymbol{x})$ can be computed efficiently because the fractional Laplacian of a standard Gaussian is known analytically.

3.2 The Density Model $p_{\mathrm{KRnet},\boldsymbol{\theta}}$

The constraints specified in Eqs. (2.6) and (2.7) on $p(\boldsymbol{x})$ bring essential difficulties to mesh-based numerical schemes for the approximation of the fractional FPEs. To this end, we employ KRnet, a certain type of normalizing flow, to build an effective approximator for FPE [11, 36].

A normalizing flow seeks an invertible mapping that corresponds to a transport map between a specified distribution and an arbitrary one. Let $\boldsymbol{Z} \in \mathbb{R}^d$ be a simple reference random variable with a known PDF $p_{\boldsymbol{Z}}$, e.g., Gaussian. Let $f : \boldsymbol{x} \to \boldsymbol{z}$ be an invertible mapping defined by a normalizing flow. Then the PDF of $\boldsymbol{X} = f^{-1}(\boldsymbol{Z})$ is given by the change of variables, i.e.,

$$p_{\boldsymbol{X}}(\boldsymbol{x}) = p_{\boldsymbol{Z}}(f(\boldsymbol{x}))\left|\det \nabla_{\boldsymbol{x}} f(\boldsymbol{x})\right|, \tag{3.7}$$

where $\nabla_{\boldsymbol{x}} f(\boldsymbol{x})$ is the Jacobian matrix. Given observations of $\boldsymbol{X}$, the unknown invertible mapping can be learned through the maximum likelihood estimation.

To construct a complex bijection f, a general idea is to stack a sequence of simple bijections, each of which is a shallow neural network, in other words, the overall mapping is a deep neural network. Namely, the mapping $f(\cdot)$ can be written in a composite form:

$$\boldsymbol{z} = f(\boldsymbol{x}) = f_{[L]} \circ f_{[L-1]} \circ \cdots \circ f_{[1]}(\boldsymbol{x}). \tag{3.8}$$

Its inverse and Jacobian determinants are given as

$$\boldsymbol{x} = f^{-1}(\boldsymbol{z}) = f_{[1]}^{-1} \circ \cdots \circ f_{[L-1]}^{-1} \circ f_{[L]}^{-1}(\boldsymbol{z}), \tag{3.9}$$

$$|\det \nabla_{\boldsymbol{x}} f(\cdot)| = \prod_{i=1}^{L} |\det \nabla_{\boldsymbol{x}_{[i-1]}} f_{[i]}(\cdot)|, \tag{3.10}$$

where $\boldsymbol{x}_{[i-1]}$ indicates the immediate variables with $\boldsymbol{x}_{[0]} = \boldsymbol{x}, \boldsymbol{x}_{[L]} = \boldsymbol{z}$. Many variants of f have been proposed to enhance the expressive power and alleviate the computational cost of Jacobian determinants at the same time [7, 8, 20]. Among them, a successful example is KRnet [8]. We here employ a simplified KRnet, which includes affine coupling layers with an invertible block-triangle structure and actnorm layers.

3.2.1 Actnorm Layer: Scale and Bias Layer

We adopt the Actnorm layer $L_{\text{Actn},[i]}$ with data-dependent initialization proposed by Kingma and Dhariwal [20]:

$$\boldsymbol{y}_{[i]} = \boldsymbol{a}_i \odot \boldsymbol{x}_{[i]} + \boldsymbol{b}_i, \tag{3.11}$$

where $\boldsymbol{a}_i$ and $\boldsymbol{b}_i$ are trainable parameters. When data are available, the parameters $\boldsymbol{b}_i$ and $\boldsymbol{a}_i$ can be initialized by the statistical mean and standard deviation from the data respectively. Otherwise, we may simply initialize $\boldsymbol{b}_i$ and $\boldsymbol{a}_i$ as $\boldsymbol{b}_i = \boldsymbol{0}$ and $\boldsymbol{a}_i = \boldsymbol{1}_d$, where $\boldsymbol{1}_d$ denotes a d-dimensional vector whose components are all 1. After initialization, the scale and bias are treated as regular trainable parameters that are independent of the data. The inverse can be easily obtained via

$$\boldsymbol{x}_{[i]} = (\boldsymbol{y}_{[i]} - \boldsymbol{b}_i)/\boldsymbol{a}_i, \tag{3.12}$$

where the division here is applied to each corresponding component.

3.2.2 Affine Coupling Layer

Let $\boldsymbol{x}_{[i]} = (\boldsymbol{x}_{[i],1}, \boldsymbol{x}_{[i],2})$ be a partition with $\boldsymbol{x}_{[i],1} \in \mathbb{R}^m$ and $\boldsymbol{x}_{[i],2} \in \mathbb{R}^{d-m}$. An affine coupling layer $L_{\text{Aff},[i]}(\cdot)$ is defined as

$$\begin{aligned} \boldsymbol{x}_{[i],1} &= \boldsymbol{x}_{[i-1],1}, \\ \boldsymbol{x}_{[i],2} &= \boldsymbol{x}_{[i-1],2} \odot \left(\boldsymbol{1}_{d-m} + \beta \tanh(\boldsymbol{s}_i(\boldsymbol{x}_{[i-1],1}))\right) + e^{\boldsymbol{\zeta}_i} \odot \tanh(\boldsymbol{q}_i(\boldsymbol{x}_{[i-1],1})), \end{aligned} \tag{3.13}$$

where $|\beta| < 1$ is a user-specified parameter (a commonly used choice is $\beta = 0.6$), $\boldsymbol{s}_i, \boldsymbol{q}_i : \mathbb{R}^m \to \mathbb{R}^{d-m}$ are scaling and translation depending only on $\boldsymbol{x}_{[i-1],1}$, and $\boldsymbol{\zeta}_i \in \mathbb{R}^{d-m}$ is a trainable variable. Notice that the inverse can be easily computed via:

$$\begin{aligned} \boldsymbol{x}_{[i-1],1} &= \boldsymbol{x}_{[i],1}, \\ \boldsymbol{x}_{[i-1],2} &= (\boldsymbol{x}_{[i],2} - e^{\boldsymbol{\zeta}_i} \odot \tanh(\boldsymbol{q}_i(\boldsymbol{x}_{[i],1}))) \odot \left(\boldsymbol{1}_{d-m} + \beta \tanh(\boldsymbol{s}_i(\boldsymbol{x}_{[i],1}))\right)^{-1}. \end{aligned} \tag{3.14}$$

The Jacobian of Eq. (3.13) can be easily computed due to the lower triangular form of Jacob matrix. Furthermore, we can model $\boldsymbol{s}_i, \boldsymbol{q}_i$ via a neural network

$$(\boldsymbol{s}_i, \boldsymbol{q}_i) = \text{NN}_{[i]}(\boldsymbol{x}_{[i-1],1}). \tag{3.15}$$

Note that $L_{\text{Aff},[i]}(\cdot)$ only changes $\boldsymbol{x}_{[i-1],2}$, implying that in the next affine coupling layer we should exchange the positions of $\boldsymbol{x}_{[i],1}$ and $\boldsymbol{x}_{[i],2}$ to ensure that each component of $\boldsymbol{x}_{[i]}$ will be updated.

Based on the actnorm layer and affine coupling layer, our simplified KRnet can be represented by

$$\boldsymbol{z} = f_{\text{KRnet}}(\boldsymbol{x}) = f_{[L]} \circ f_{[L-1]} \circ \cdots \circ f_{[1]}(\boldsymbol{x}), \tag{3.16}$$

$$f_{[i]} = L_{\text{Aff},[i]} \circ L_{\text{Actn},[i]}, \quad i = 1, \ldots, L, \tag{3.17}$$

where $L_{\text{Aff},[i]}$ is an affine coupling layer defined by (3.13) and $L_{\text{Actn},[i]}$ is an Actnorm layer defined by (3.11).

3.3 Stochastic Approximation of the Fractional Operator

To compute the fractional Laplacian of the $p_{\mathrm{KRnet},\boldsymbol{\theta}}(x)$ with $\alpha \in (0,2)$, we apply the stochastic approximation method proposed in [15].

Lemma 1 [15] *Given a function u, its fractional Laplacian can be decomposed into terms over a neighborhood $B_{r_0}(\boldsymbol{x}) = \{\boldsymbol{y} \mid |\boldsymbol{y}-\boldsymbol{x}|_2 \le r_0\}$ around $\boldsymbol{x}$ and its complement as*

$$(-\Delta)^{\alpha/2}u(\boldsymbol{x}) = C_{d,\alpha}\left(\int_{\boldsymbol{y}\in B_{r_0}(\boldsymbol{x})} \frac{u(\boldsymbol{x})-u(\boldsymbol{y})}{|\boldsymbol{x}-\boldsymbol{y}|_2^{d+\alpha}}\mathrm{d}\boldsymbol{y} + \int_{\boldsymbol{y}\notin B_{r_0}(\boldsymbol{x})} \frac{u(\boldsymbol{x})-u(\boldsymbol{y})}{|\boldsymbol{x}-\boldsymbol{y}|_2^{d+\alpha}}\mathrm{d}\boldsymbol{y}\right), \tag{3.18}$$

which, if exists, takes the form

$$\begin{aligned}(-\Delta)^{\alpha/2}u(\boldsymbol{x}) = {} & C_{d,\alpha}\frac{|S^{d-1}|\,r_0^{2-\alpha}}{2(2-\alpha)}\mathbb{E}_{\boldsymbol{\xi}\sim \mathrm{U}(\mathrm{S}^{\mathrm{d}-1}),\mathrm{r}_1\sim \mathrm{f}_{\mathrm{I}}(\mathrm{r})} \\ & \left[\frac{2u(\boldsymbol{x})-u(\boldsymbol{x}-r_1\boldsymbol{\xi})-u(\boldsymbol{x}+r_1\boldsymbol{\xi})}{r_1^2}\right] \\ & + C_{d,\alpha}\frac{|S^{d-1}|\,r_0^{-\alpha}}{2\alpha}\mathbb{E}_{\boldsymbol{\eta}\sim \mathrm{U}(\mathrm{S}^{\mathrm{d}-1}),\mathrm{r}_2\sim \mathrm{f}_{\mathrm{O}}(\mathrm{r})}\big[2u(\boldsymbol{x})-u(\boldsymbol{x}-r_2\boldsymbol{\eta})-u(\boldsymbol{x}+r_2\boldsymbol{\eta})\big],\end{aligned} \tag{3.19}$$

where $\boldsymbol{\xi}$ and $\boldsymbol{\eta}$ are uniformly distributed on the the $(d-1)$-dimensional unit sphere S^{d-1}, $|S^{d-1}|$ denotes the surface area of S^{d-1},

$$f_{\mathrm{I}}(r) = \frac{2-\alpha}{r_0^{2-\alpha}} r^{1-\alpha}\cdot 1_{r\in[0,r_0]}, \quad f_{\mathrm{O}}(r) = \alpha r_0^{\alpha} r^{-1-\alpha} 1_{r\in[r_0,\infty)},$$

1_{Ω} is a characteristic function and r_1 and r_2 can be sampled as

$$r_1/r_0 \sim \mathrm{Beta}(2-\alpha,1), \quad r_0/r_2 \sim \mathrm{Beta}(\alpha,1). \tag{3.20}$$

Notice that the first expectation in Eq. (3.19) may suffer from the round-off error and give rise to numerical instability for an extremely small r. Therefore, the following approximation is considered in practice

$$\begin{aligned}&\mathbb{E}_{\boldsymbol{\xi},r_1}\left[\frac{2u(\boldsymbol{x})-u(\boldsymbol{x}-r_1\boldsymbol{\xi})-u(\boldsymbol{x}+r_1\boldsymbol{\xi})}{r_1^2}\right] \\ &\quad\approx \mathbb{E}_{\boldsymbol{\xi},r_1}\left[\frac{2u(\boldsymbol{x})-u(\boldsymbol{x}-r_\epsilon\boldsymbol{\xi})-u(\boldsymbol{x}+r_\epsilon\boldsymbol{\xi})}{r_\epsilon^2}\right],\end{aligned} \tag{3.21}$$

where $\boldsymbol{\xi} \sim \mathrm{U}(\mathrm{S}^{\mathrm{d}-1})$, $\mathrm{r}_1 \sim \mathrm{f}_{\mathrm{I}}(\mathrm{r})$ and $r_\epsilon = \max\{\epsilon, r_1\}$, $\epsilon > 0$ is a small positive number.

By combining the stochastic approximation for the fractional Laplacian operator and the physics-informed neural network (3.1), along with automatic differentiation for the integer-order derivative, we obtain the final approximation $\hat{L}(p_{\mathrm{KRnet},\boldsymbol{\theta}}; r_\epsilon, r_0)$ for $L(p_{\mathrm{KRnet},\boldsymbol{\theta}})$ as

follows

$$\begin{aligned}
&\hat{L}(p_{\mathrm{KRnet},\boldsymbol{\theta}}; r_\epsilon, r_0)\\
&= \frac{1}{N_S}\sum_{i=1}^{N_S}\Big|-\nabla\cdot(\boldsymbol{\mu}\, p_{\mathrm{KRnet},\boldsymbol{\theta}})(\boldsymbol{x}^i) + \frac{1}{2}\nabla\cdot\nabla\cdot(\boldsymbol{\sigma}\boldsymbol{\sigma}^{\mathrm{T}} p_{\mathrm{KRnet},\boldsymbol{\theta}})(\boldsymbol{x}^i)\\
&\quad - C_{d,\alpha}\frac{\left|S^{d-1}\right| r_0^{2-\alpha}}{2\,(2-\alpha)}\mathbb{E}_{\boldsymbol{\xi}\sim \mathrm{U}(\mathrm{S}^{\mathrm{d}-1}),\mathrm{r}_1\sim \mathrm{f}_{\mathrm{I}}(\mathrm{r})}\\
&\quad \left[\frac{2p_{\mathrm{KRnet},\boldsymbol{\theta}}(\boldsymbol{x}^i) - p_{\mathrm{KRnet},\boldsymbol{\theta}}(\boldsymbol{x}^i - r_\epsilon\boldsymbol{\xi}) - p_{\mathrm{KRnet},\boldsymbol{\theta}}(\boldsymbol{x}^i + r_\epsilon\boldsymbol{\xi})}{r_\epsilon^2}\right]\\
&\quad - C_{d,\alpha}\frac{\left|S^{d-1}\right| r_0^{-\alpha}}{2\alpha}\mathbb{E}_{\boldsymbol{\eta}\sim \mathrm{U}(\mathrm{S}^{\mathrm{d}-1}),\mathrm{r}_2\sim \mathrm{f}_{\mathrm{O}}(\mathrm{r})}\\
&\quad \left[2p_{\mathrm{KRnet},\boldsymbol{\theta}}(\boldsymbol{x}^i) - p_{\mathrm{KRnet},\boldsymbol{\theta}}(\boldsymbol{x}^i - r_2\boldsymbol{\eta}) - p_{\mathrm{KRnet},\boldsymbol{\theta}}(\boldsymbol{x}^i + r_2\boldsymbol{\eta})\right]\Big|^2.
\end{aligned} \tag{3.22}$$

3.4 The Auxiliary Density Model $p_{\mathrm{GRBF},\tilde{\boldsymbol{\theta}}}$

The definition of the auxiliary density model $p_{\mathrm{GRBF},\tilde{\boldsymbol{\theta}}}$ is based on the following lemma [3]:

Lemma 2 *Let u be a Gaussian function of the form $u(\boldsymbol{x}) = exp(-\sigma^{-2}|\boldsymbol{x}-\boldsymbol{x_0}|_2^2)$ for $\boldsymbol{x}, \boldsymbol{x}_0 \in \mathbb{R}^d$. Then the fractional Laplacian of u is analytically given as*

$$(-\Delta)^{\frac{\alpha}{2}}u(\boldsymbol{x}) = c_{\alpha,d}|\sigma|^{-\alpha}{}_1F_1\left(\frac{d+\alpha}{2}; \frac{d}{2}; -\sigma^{-2}|\boldsymbol{x}-\boldsymbol{x_0}|_2^2\right) \text{ for } \boldsymbol{x}\in\mathbb{R}^d, \alpha \geq 0, \tag{3.23}$$

where ${}_1F_1$ denotes the confluent hypergeometric function, and

$$c_{\alpha,d} = \frac{2^\alpha \Gamma\left(\frac{d+\alpha}{2}\right)}{\Gamma\left(\frac{d}{2}\right)}.$$

For a set $S_{\mathrm{center}} = \{\tilde{\boldsymbol{x}}_i\}_{i=1}^M$, we let

$$p_{\mathrm{GRBF},\tilde{\boldsymbol{\theta}}}(\boldsymbol{x}) = \sum_{i=1}^M w_i \mathcal{N}(\tilde{\boldsymbol{x}}_i, \sigma_i^2\mathbf{I})(\boldsymbol{x}), \tag{3.24}$$

where $0 \leq \omega_i \leq 1$ such that $\sum_{i=1}^M \omega_i = 1$, σ_i is the bandwidth at $\tilde{\boldsymbol{x}}_i$, and $\mathcal{N}$ denotes the Normal distribution,

$$\mathcal{N}(\tilde{\boldsymbol{x}}_i, \sigma_i^2\mathbf{I})(\boldsymbol{x}) = (2\pi)^{-d/2}\sigma_i^{-d}\exp\left(-\frac{|\boldsymbol{x}-\tilde{\boldsymbol{x}}_i|_2^2}{2\sigma_i^2}\right). \tag{3.25}$$

Here both ω_i and σ_i can be trainable parameters, which are included in $\tilde{\boldsymbol{\theta}}$. Using Lemma 2, we obtain that

$$
\begin{aligned}
(-\Delta)^{\alpha/2} p_{\mathrm{GRBF},\tilde{\boldsymbol{\theta}}}(\boldsymbol{x}) &= \sum_{i=1}^{M} w_i (-\Delta)^{\alpha/2} \mathcal{N}(\tilde{\boldsymbol{x}}_i, \sigma_i^2 \mathbf{I})(\boldsymbol{x}) \\
&= c_{\alpha,d} \pi^{-d/2} 2^{-\frac{d+\alpha}{2}} \sum_{i=1}^{M} w_i |\sigma_i|^{-(d+\alpha)} {}_1F_1\left(\frac{d+\alpha}{2}; \frac{d}{2}; -\frac{|\boldsymbol{x}-\tilde{\boldsymbol{x}}_i|_2^2}{2{\sigma_i}^2}\right).
\end{aligned} \tag{3.26}
$$

Consequently, the loss function (3.6) can be rewritten by

$$
\begin{aligned}
&\tilde{L}(p_{\mathrm{KRnet},\boldsymbol{\theta}}, p_{\mathrm{GRBF},\tilde{\boldsymbol{\theta}}}) \\
&= \frac{1}{N} \sum_{i=1}^{N} \Bigg(\mathcal{L} p_{\mathrm{KRnet},\boldsymbol{\theta}}(\boldsymbol{x}^i) - c_{\alpha,d} \pi^{-\frac{d}{2}} 2^{-\frac{d+\alpha}{2}} \\
&\quad \sum_{j=1}^{M} w_j |\sigma_j|^{-(d+\alpha)} {}_1F_1\left(\frac{d+\alpha}{2}; \frac{d}{2}; -\frac{|\boldsymbol{x}^i-\tilde{\boldsymbol{x}}_j|_2^2}{2{\sigma_j}^2}\right)\Bigg)^2 \\
&\quad + \frac{\beta_m}{N} \sum_{i=1}^{N} \left(p_{\mathrm{KRnet},\boldsymbol{\theta}}(\boldsymbol{x}^i) - \sum_{j=1}^{M} w_j \mathcal{N}(\tilde{\boldsymbol{x}}_j, \sigma_j^2 \mathbf{I})(\boldsymbol{x}^i) \right)^2,
\end{aligned} \tag{3.27}
$$

where the integer-order derivatives in the operator $\mathcal{L}$ can be conducted via automatic differentiation.

It is seen that the factional Laplacian of $p_{\mathrm{GRBF},\tilde{\boldsymbol{\theta}}}$ is determined by the confluent hypergeometric function ${}_1F_1(\cdot)$. If we allow σ_i to be a trainable parameter, we need the derivative of ${}_1F_1$ which is

$$
\frac{\mathrm{d}}{\mathrm{d}x} {}_1F_1\left(\frac{d+\alpha}{2}; \frac{d}{2}; x\right) = \frac{d+\alpha}{d} {}_1F_1\left(\frac{d+\alpha}{2}+1; \frac{d}{2}+1; x\right). \tag{3.28}
$$

In general, it is computationally expensive to evaluate the confluent hypergeometric function. Fortunately, only the one-dimensional hypergeometric function is needed. We then use piecewise Chebyshev polynomials to approximate the one-dimensional confluent hypergeometric function up to a desired accuracy, which can be done once and for all at the preprocessing stage.

3.5 An Adaptive Strategy for the Training Process

3.5.1 Where Do We Need Adaptivity

We pay particular attention to two components of the algorithm that are closely related to adaptivity: one is the training set S and the other one is the auxiliary model $p_{\mathrm{GRBF},\tilde{\boldsymbol{\theta}}}$. In MCNF, we only consider adaptivity for the training set S while in GRBFNF we address the adaptivity for both S and the model $p_{\mathrm{GRBF},\tilde{\boldsymbol{\theta}}}$.

If the modeling capability of $p_{\mathrm{KRnet},\boldsymbol{\theta}}$ is sufficient, the training set S determines the accuracy of $p_{\mathrm{KRnet},\boldsymbol{\theta}^*}$ because it defines the loss function for both MCNF and GRBFNF. For example, the loss function (3.1) can be regarded as

$$
L(p_{\mathrm{KRnet},\boldsymbol{\theta}}) := \frac{1}{N_S} \sum_{i=1}^{N_S} |R_{\boldsymbol{\theta}}(\boldsymbol{x}^i)|^2 \approx \int_{\mathbb{R}^d} R_{\boldsymbol{\theta}}^2(\boldsymbol{x}) \rho(\boldsymbol{x}) d\boldsymbol{x}, \tag{3.29}
$$

where $\boldsymbol{x}^i$ are samples from a PDF $\rho(\boldsymbol{x})$ with $\rho(\boldsymbol{x}) > 0$ for any $\boldsymbol{x} \in \mathbb{R}^d$. We regard the loss function (3.1) of MCNF as a Monte Carlo approximation of the expectation $\mathbb{E}_\rho[R_{\boldsymbol{\theta}}]$ with respect to an underlying PDF $\rho(\boldsymbol{x})$, in other words, the loss in the continuous form corresponds to a weighted L_2 norm of $R_{\boldsymbol{\theta}}(\boldsymbol{x})$ with the weight function $\rho(\boldsymbol{x})$. The importance of $\rho(\boldsymbol{x})$ is twofold: First, calculating the loss function (3.29) presents a significant challenge due to its integration over an infinite region. It is obvious that $\rho(\boldsymbol{x})$ cannot be uniform on $\mathbb{R}^d$. One commonly used strategy in practice is to employ uniform samples on a truncated computation domain. Second, for a fixed sample size, the Monte Carlo approximation of $\mathbb{E}_\rho[R_{\boldsymbol{\theta}}]$ has a smaller error when $R_{\boldsymbol{\theta}}$ has a smaller variance in terms of $\rho(\boldsymbol{x})$.

An evident option for $\rho(\boldsymbol{x})$ is the solution $p(\boldsymbol{x})$. This choice is driven by the intuition that the residual in the region of higher probability holds greater significance compared to those with lower probability density. In application problems, the solution $p(\boldsymbol{x})$ is often smooth but localized. The simplest way to incorporate the properties of $p(\boldsymbol{x})$ into the training process is to use more samples in the region of high density and fewer samples in the region of low density. Using the nonuniform samples that are consistent with $p(\boldsymbol{x})$ has a similar effect to the strategy that associates each uniform sample with a weight. Apparently, for a better approximation the weight for the sample from the region of high density should be larger. We also note that the nonuniform samples help smooth the profile of the residual $R_{\boldsymbol{\theta}}$, which reduces the variance of $R_{\boldsymbol{\theta}}$, implying that the Monte Carlo approximation of $\mathbb{E}_\rho[R_{\boldsymbol{\theta}}]$ is improved through variance reduction. A better approximation of $\mathbb{E}_\rho[R_{\boldsymbol{\theta}}]$ will eventually reduce the statistical or estimator error of the approximate solution $p_{\text{KRnet},\boldsymbol{\theta}^*}$ [37]. Since $p(\boldsymbol{x})$ is unknown, we may sample its approximation $p_{\text{KRnet},\boldsymbol{\theta}^*}$ to form a new training set S. This suggests an adaptive solver for $p_{\text{KRnet},\boldsymbol{\theta}}$, where we update S and $p_{\text{KRnet},\boldsymbol{\theta}^*}$ alternately. Depending on the assumption on $p(\boldsymbol{x})$, more sophisticated adaptive sampling strategies may be needed. For example, if $p(\boldsymbol{x})$ is discontinuous, sampling $p(\boldsymbol{x})$ may not be sufficient since it does not take into account the discontinuity. We leave this for future study.

The auxiliary model $p_{\text{GRBF},\tilde{\boldsymbol{\theta}}}$ as an alternative representation of $p_{\text{KRnet},\boldsymbol{\theta}}$ can be regarded as a kernel density estimator (KDE) since $p_{\text{KRnet},\boldsymbol{\theta}}$ is a PDF. Given a set of samples $\{\boldsymbol{x}_i\}$, a general adaptive multivariate KDE takes the form [38],

$$\hat{p}(\boldsymbol{x}) = \frac{1}{N}\sum_{i=1}^{N} K_{\boldsymbol{H}_i}(\boldsymbol{x} - \boldsymbol{x}_i), \tag{3.30}$$

where $\boldsymbol{H}_i$ is the bandwidth matrix and $K_{\boldsymbol{H}_i} = |\boldsymbol{H}_i|^{-1} K(\boldsymbol{H}_i^{-1}\boldsymbol{x})$ rescales a kernel function $K(\boldsymbol{x})$. Due to Lemma 2, we choose $\boldsymbol{K}(\boldsymbol{x})$ as a standard multivariate Gaussian and $\boldsymbol{H}_i = h_i \boldsymbol{I}$ with h_i being the bandwidth shared by all dimensions. The main difference between $p_{\text{GRBF},\tilde{\boldsymbol{\theta}}}$ and a kernel density estimator is that the points in S_{center} may not be samples from the probability density function to be approximated. This is why $p_{\text{GRBF},\tilde{\boldsymbol{\theta}}}$ in Eq. (3.24) has variable coefficients w_i while the KDE in Eq. (3.30) has a constant coefficient $\frac{1}{N}$. Since $p_{\text{GRBF},\tilde{\boldsymbol{\theta}}^*} \approx p_{\text{KRnet},\boldsymbol{\theta}^*}$, we expect that S_{center} has a data distribution that is consistent with $p_{\text{KRnet},\boldsymbol{\theta}^*}$. When $p_{\text{KRnet},\boldsymbol{\theta}^*}$ is updated adaptively, the set S_{center} should be updated accordingly for a more effective representation of $p_{\text{GRBF},\tilde{\boldsymbol{\theta}}^*}$. As $N \to \infty$, the KDE is simply the Monte Carlo simulation. However, for a GRBF approximation with a relatively small number of basis functions, varying w_i rather than the constant $\frac{1}{N}$ yields better performance. Once a new S_{center} is specified, a straightforward idea to update the parameters of GRBFs is to project $p_{\text{KRnet},\boldsymbol{\theta}^*}$ onto the new space spanned by the Gaussian radial basis functions with updated centers.

3.5.2 Adaptivity of MCNF

We propose the following adaptive sampling strategy to update the training set S. The initial collocation points in S are drawn from a uniform distribution in an area determined by our prior knowledge of $p(\boldsymbol{x})$. Then we solve the optimization problem (3.24) via the Adam optimizer to obtain the optimal $\boldsymbol{\theta}^*$, which corresponds to an NF mapping $f_{\boldsymbol{\theta}^*,0}$ and a PDF $p_{\mathrm{KRnet},\boldsymbol{\theta}^*,0}(\boldsymbol{x})$. We subsequently update S using samples from $p_{\mathrm{KRnet},\boldsymbol{\theta}^*,0}(\boldsymbol{x})$. To be precise, we sample the latent Gaussian random variable $\boldsymbol{Z}$, and use the samples of $\boldsymbol{X} = \left(f_{\boldsymbol{\theta}^*,0}\right)^{-1}(\boldsymbol{Z})$ to form the new training set S_1. With S_1, we start a new round of training to update $p_{\mathrm{KRnet},\boldsymbol{\theta}^*,0}(\boldsymbol{x})$. We repeat this procedure until the maximum iteration number is reached. This strategy can be concluded as follows.

1. Generate an initial training set with samples uniformly distributed in $\Omega_0 \subset \mathbb{R}^d$:

$$S_0 = \{\boldsymbol{x}^{i,0}\}_{i=1}^{N_S} \subset \Omega_0, \quad \boldsymbol{x}^{i,0} \sim \text{Uniform } \Omega_0.$$

2. Train the KRnet by minimizing the loss function (3.22) with training data S_0 and hyper-parameters r_ϵ and r_0 to obtain $\boldsymbol{\theta}^{*,0}$.

$$\boldsymbol{\theta}^{*,0} = \arg\min_{\boldsymbol{\theta}} \hat{L}(p_{\mathrm{KRnet},\boldsymbol{\theta}}; r_\epsilon, r_0).$$

3. Generate samples from $p_{\mathrm{KRnet},\boldsymbol{\theta}^*,0}(\cdot)$ to get a new training set $S_1 = \{\boldsymbol{x}^{i,1}\}_{i=1}^{N_S}$, and set $S_0 = S_1$. Notice that $\boldsymbol{x}^{i,1}$ can be obtained by transforming the prior Gaussian samples via the inverse temporal normalizing flow,

$$\boldsymbol{z}^{i,1} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}), \quad \boldsymbol{x}^{i,1} = \left(f_{\boldsymbol{\theta}^*,0}\right)^{-1}(\boldsymbol{z}^{i,1}).$$

4. Repeat steps 2-3 for N_{adaptive} times to get a convergent approximation.

The algorithm for MCNF is presented in Algorithm 1 and the corresponding flow chart is depicted in Fig. 1. Mini batches are used to accelerate the training process. Since the initial training points are uniformly distributed, we only expect that $p_{\mathrm{KRnet},\boldsymbol{\theta}^*,0}(\boldsymbol{x})$ could capture the main behavior of the exact solution $p(\boldsymbol{x})$, which implies that a relatively small number of epochs is enough. As the convergence is being established by the adaptive procedure, we expect that $p_{\mathrm{KRnet},\boldsymbol{\theta}^*,i}(\boldsymbol{x})$ could capture more details of $p(\boldsymbol{x})$ as the iteration number i increases, which implies that the number of epochs may increase accordingly. We introduce a hyper-parameter γ in Algorithm 1 to represent the growth rate of epoch number for each adaptivity iteration.

3.5.3 Adaptivity of GRBFNF

Compared to MCNF, we need to address the adaptivity for both S and the auxiliary model $p_{\mathrm{GRBF},\tilde{\boldsymbol{\theta}}}$. The training set S follows the same adaptive procedure as in MCNF. We here focus on the adaptivity for the auxiliary model. Depending on the prior knowledge, the initial center set S_{center} will be formed by uniform samples in a certain area. After $p_{\mathrm{KRnet},\boldsymbol{\theta}^*,0}$ is obtained, S_{center} will be updated by samples from $p_{\mathrm{KRnet},\boldsymbol{\theta}^*,0}$. To continue the training process with the updated S and S_{center}, we need to reinitialize the weights $\{w_i\}$ and the bandwidths $\{\sigma_i\}$ for the GRBFs of the auxiliary model, which will be done by solving a least-square problem.

Reinitialization of the GRBFs. Once $p_{\mathrm{KRnet},\boldsymbol{\theta}^*,k}$ is obtained for the k-th adaptivity iteration, we sample it to update the training set from S_k to S_{k+1} and GBRF centers from $S_{\text{center},k}$ to

Algorithm 1 MCNF

Input: maximum epoch number N_e, maximum iteration number N_{adaptive}, fractional order α, hyper-parameter r_ϵ, r_0, γ, initial training data $S = \{\boldsymbol{x}^i\}_{i=1}^{N_S}$, tolerance ϵ_1, ϵ_2;
$L_{old} = 0$;
for $k = 0, \cdots, N_{\text{adaptive}}$ **do**
  for $j = 1, \cdots, N_e$ **do**
    Divide S into m batch $\{S^{ib}\}_{ib=1}^{m}$ randomly;
    for $ib = 1, \cdots, m$ **do**
      Compute the loss function (3.22) $\hat{L}^{ib}(p_{\text{KRnet},\boldsymbol{\theta}}; r_\epsilon, r_0)$ for mini-batch data S^{ib} and order α;
      Update $\boldsymbol{\theta}$ by using the Adam optimizer;
    $L_{new} = \frac{1}{m}\sum_{ib=1}^{m} \hat{L}^{ib}(p_{\text{KRnet},\boldsymbol{\theta}}; r_\epsilon, r_0)$;
    if $L_{new} < \epsilon_1$ or $|L_{old} - L_{new}| < \epsilon_2$ **then**
      Break;
    else
      $L_{old} = L_{new}$;
  $N_e = \gamma * N_e$;
  Sample from $p_{\text{KRnet},\boldsymbol{\theta}}(\cdot)$ and update training set S;
Output: The predicted solution $p_{\text{KRnet},\boldsymbol{\theta}}(\boldsymbol{x})$.

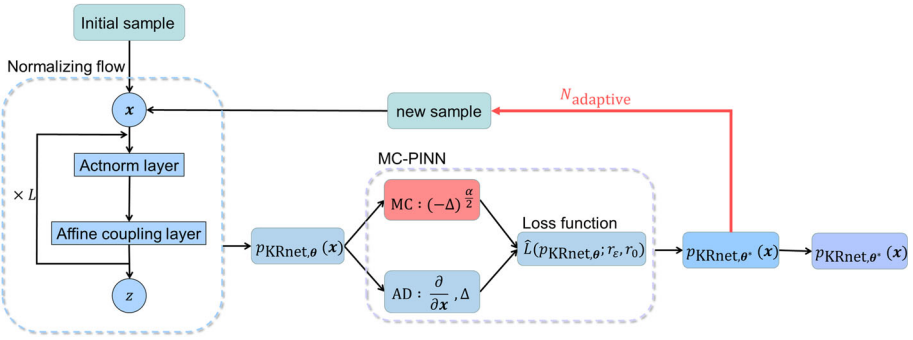


Fig. 1 Flow chart of MCNF

$S_{\text{center},k+1}$. The new auxiliary model is defined as

$$p_{\text{GRBF},\tilde{\boldsymbol{\theta}}^{k+1}}(\boldsymbol{x}) = \sum_{i=1}^{M} w_i N(\tilde{\boldsymbol{x}}_i^{k+1}, \sigma_i^2 \boldsymbol{I}_d)(\boldsymbol{x}), \quad \tilde{\boldsymbol{x}}_i^{k+1} \in S_{\text{center},k+1}. \tag{3.31}$$

and initialized as

$$\begin{aligned} \{w_{\text{new},i}, \sigma_{\text{new},i}\}_{i=1}^{M} &= \arg\min_{w_i,\sigma_i} Loss \\ &= \arg\min_{w_i,\sigma_i} \frac{1}{N_{S_{k+1}}} \sum_{j=1}^{N_{S_{k+1}}} (p_{\text{GRBF},\tilde{\boldsymbol{\theta}}^{k+1}}(\boldsymbol{x}^j) - p_{\text{KRnet},\boldsymbol{\theta}^{*,k}}(\boldsymbol{x}^j))^2, \end{aligned} \tag{3.32}$$

where the Adam optimizer is used to solve the above optimization problem. After initialization, both the weights $\{w_i\}$ and the bandwidths $\{\sigma_i\}$ are trainable.

This strategy can be concluded as follows.

1. Generate initial training sets S and S_{center} with samples uniformly distributed in a certain physical domain:

$$S_0 = \{\boldsymbol{x}^{i,0}\}_{i=1}^{N_S} \subset \Omega_0, \quad \boldsymbol{x}^{i,0} \sim \text{Uniform } \Omega_0,$$
$$S_{\text{center},0} = \{\tilde{\boldsymbol{x}}_i^0\}_{i=1}^{M} \subset \Omega_0, \quad \tilde{\boldsymbol{x}}_i^0 \sim \text{Uniform } \Omega_0.$$

2. Train the KRnet by minimizing the loss function (3.27) with training data S_0 to obtain $\boldsymbol{\theta}^{*,0}$ and $\tilde{\boldsymbol{\theta}}^{*,0}$, i.e.,

$$\{\boldsymbol{\theta}^{*,0}, \tilde{\boldsymbol{\theta}}^{*,0}\} = \arg\min_{\boldsymbol{\theta},\tilde{\boldsymbol{\theta}}} \tilde{L}(p_{\text{KRnet},\boldsymbol{\theta}}, p_{\text{GRBF},\tilde{\boldsymbol{\theta}}}).$$

3. Generate samples with $p_{\text{KRnet},\boldsymbol{\theta}^{*,0}}$ to get a new training set $S_1 = \{\boldsymbol{x}^{i,1}\}_{i=1}^{N_r}$, and a new center set $S_{\text{center},1} = \{\tilde{\boldsymbol{x}}_i^1\}$. Notice that $\boldsymbol{x}^{i,1}$ and $\tilde{\boldsymbol{x}}_j^1$ can be obtained by transforming the prior Gaussian samples via the inverse normalizing flow.

$$\boldsymbol{z}^{i,1} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}), \quad \boldsymbol{x}^{i,1} = \left(f_{\boldsymbol{\theta}^{*,0}}\right)^{-1}(\boldsymbol{z}^{i,1}),$$
$$\tilde{\boldsymbol{z}}_j^1 \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}), \quad \tilde{\boldsymbol{x}}_j^1 = \left(f_{\boldsymbol{\theta}^{*,0}}\right)^{-1}(\tilde{\boldsymbol{z}}_j^1).$$

4. Project $p_{\text{KRnet},\boldsymbol{\theta}^{*,0}}$ onto the new GRBF space by solving the problem (3.32). Set $S_0 = S_1$, $S_{\text{center},0} = S_{\text{center},1}$.
5. Repeat steps 2-4 for N_{adaptive} times to get a convergent approximation.

The algorithm for GRBFNF is summarized in Algorithm 2, and a flow chart is given in Fig. 2.

Algorithm 2 GRBFNF

Input: maximum epoch number N_e, maximum iteration number N_{adaptive}, fractional order α, hyper parameter γ, initial training data $S = \{\boldsymbol{x}^i\}_{i=1}^N$, center set $S_{\text{center}} = \{\tilde{\boldsymbol{x}}_i\}_{i=1}^M$, tolerance ϵ_1, ϵ_2;
$L_{old} = 0$;
for $k = 0, \cdots, N_{\text{adaptive}}$ **do**
  for $j = 1, \cdots, N_e$ **do**
    Divide S into m batch $\{S^{ib}\}_{ib=1}^m$ randomly;
    for $ib = 1, \cdots, m$ **do**
      Compute the loss function (3.27) $\tilde{L}^{ib}(p_{\text{KRnet},\boldsymbol{\theta}}, p_{\text{GRBF},\tilde{\boldsymbol{\theta}}})$ for mini-batch data S^{ib} and fractional order α;
      Update $\boldsymbol{\theta}, \tilde{\boldsymbol{\theta}}$ using the Adam optimizer;
    $L_{new} = \frac{1}{m}\sum_{ib=1}^m \tilde{L}^{ib}(p_{\text{KRnet},\boldsymbol{\theta}}, p_{\text{GRBF},\tilde{\boldsymbol{\theta}}})$;
    if $L_{new} < \epsilon_1$ or $|L_{old} - L_{new}| < \epsilon_2$ **then**
      Break;
    else
      $L_{old} = L_{new}$;
  $N_e = \gamma * N_e$;
  Sample from $p_{\text{KRnet},\boldsymbol{\theta}}(\cdot)$ and update the training set S, S_{center};
  Update $p_{\text{GRBF},\tilde{\boldsymbol{\theta}}}$ by solving the optimization problem (3.32).
Output: The predicted solution $p_{\text{KRnet},\boldsymbol{\theta}}(\boldsymbol{x})$.

4 Time-Dependent Fractional FPEs

The procedure is overall similar to the stationary case if we can address the time-dependent problems on a space-time domain. Considering that the update of GRBF centers cannot

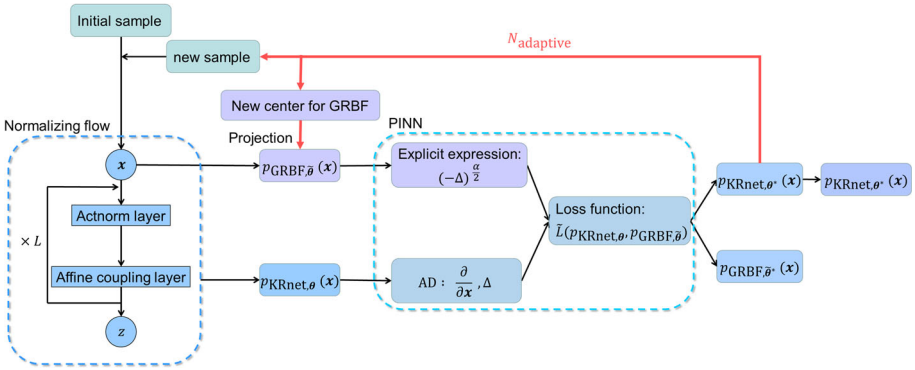


Fig. 2 Flow chart of GRBFNF

be straightforwardly generalized to the space-time domain, we only generalize MCNF for time-dependent FPEs in this work.

Given training sets $S_t = \{(x^i, t^i)\}_{i=1}^{N_t}$ and $S_{\text{ic}} = \{(\boldsymbol{x}_{\text{ic}}^i, p_0(\boldsymbol{x}_{\text{ic}}^i))\}_{i=1}^{N_{\text{ic}}}$ with $(\boldsymbol{x}, t) \in \mathbb{R}^d \times [0, T]$, we define the following loss function

$$L_T(p_{\text{KRnet},\boldsymbol{\theta}}(\boldsymbol{x}, t)) := \frac{1}{N_t}\sum_{i=1}^{N_t}\left|R_{\boldsymbol{\theta}}(\boldsymbol{x}^i, t^i)\right|^2 + \frac{\beta_D}{N_{\text{ic}}}\sum_{i=1}^{N_{\text{ic}}}\left|p_{\text{KRnet},\boldsymbol{\theta}}(\boldsymbol{x}_{\text{ic}}^i, 0) - p_0(\boldsymbol{x}_{\text{ic}}^i)\right|^2, \quad (4.1)$$

where $p_0(\cdot)$ is an initial distribution, β_D is a weight parameter to balance the governing equation loss and the initial condition loss, and the residual $R_{\boldsymbol{\theta}}(\boldsymbol{x}, t)$ is defined as

$$R_{\boldsymbol{\theta}}(\boldsymbol{x}, t) := (\partial_t - \mathcal{L} + (-\Delta)^{\alpha/2})\, p_{\text{KRnet},\boldsymbol{\theta}}(\boldsymbol{x}, t). \quad (4.2)$$

The optimal parameter $\boldsymbol{\theta}^*$ can be obtained via solving the following optimization problem:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} L_T(p_{\text{KRnet},\boldsymbol{\theta}}). \quad (4.3)$$

Note that $p_{\text{KRnet},\boldsymbol{\theta}}(\boldsymbol{x}, t)$ depends on both $\boldsymbol{x}$ and t, meaning that the corresponding KRnet is a time-independent normalizing flow.

4.1 Time-Dependent Density Model

The time-dependent PDF $p_{\text{KRnet},\theta}(\boldsymbol{x}, t)$ can be regarded as a conditional PDF $p_{\text{KRnet},\theta}(\boldsymbol{x}|t)$, which can be achieved by making the affine coupling layer time dependent. Let $\boldsymbol{x}_{[i]} = (\boldsymbol{x}_{[i],1}, \boldsymbol{x}_{[i],2})$ be a partition with $\boldsymbol{x}_{[i],1} \in \mathbb{R}^m$ and $\boldsymbol{x}_{[i],2} \in \mathbb{R}^{d-m}$. We define a time-dependent coupling layer $T_{\text{Aff},[i]}(\cdot, t)$ as follows:

$$\begin{aligned} \boldsymbol{x}_{[i],1} &= \boldsymbol{x}_{[i-1],1}, \\ \boldsymbol{x}_{[i],2} &= \boldsymbol{x}_{[i-1],2} \odot \left(\mathbf{1}_{d-m} + \beta \tanh(\boldsymbol{s}_{i,t}(\boldsymbol{x}_{[i-1],1}, t))\right) + e^{\boldsymbol{\zeta}_i} \odot \tanh(\boldsymbol{q}_{i,t}(\boldsymbol{x}_{[i-1],1}, t)), \end{aligned} \quad (4.4)$$

where the only difference from the affine coupling layer defined in Sect. 3.2.2 is that $\boldsymbol{s}_{i,t}$ and $\boldsymbol{q}_{i,t}$ include t as their inputs such that

$$(\boldsymbol{s}_{i,t}, \boldsymbol{q}_{i,t}) = \text{NN}_{[i],t}(\boldsymbol{x}_{[i-1],1}, t). \quad (4.5)$$

Based on the Actnorm layer and time-dependent affine coupling layer, our simplified time-dependent KRnet can be represented by

$$z = f_{\text{KRnet},\boldsymbol{\theta}}(\boldsymbol{x}, t) = f_{[L]} \circ f_{[L-1]} \circ \cdots \circ f_{[1]}(\boldsymbol{x}, t), \tag{4.6}$$

$$f_{[i]} = T_{\text{Aff},[i]} \circ L_{\text{Actn},[i]}, \quad i = 1, \ldots, L, \tag{4.7}$$

where $T_{\text{Aff},[i]}$ is a time-dependent affine coupling layer defined by Eq. (4.4) and $L_{\text{Actn},[i]}$ is an Actnorm layer defined by Eq. (3.11). For any t, we obtain an explicit condition PDF from Eq. (4.6):

$$p_{\text{KRnet},\theta}(\boldsymbol{x}, t) = p_{\text{KRnet},\theta}(\boldsymbol{x}|t) = p_{\boldsymbol{Z}}(f_{\text{KRnet},\boldsymbol{\theta}}(\boldsymbol{x}, t)) \left|\nabla_{\boldsymbol{x}} f_{\text{KRnet},\boldsymbol{\theta}}(\boldsymbol{x}, t)\right|. \tag{4.8}$$

Also note that for any t, $(-\Delta)^{\alpha/2} p_{\text{KRnet},\theta}(\boldsymbol{x}, t)$ can be approximated using the same procedure given in Sect. 3.3.

One commonly used strategy to enhance the effectiveness and robustness of the algorithm is to integrate some physical constraints explicitly into the algorithm [4, 30, 31, 34]. We here propose a simple modification for the affine coupling layer such that $p_{\text{KRnet},\theta}(\boldsymbol{x}, t)$ may satisfy the initial condition exactly without introducing a penalty term in the loss function.

Modified affine coupling layer. To include the initial condition, we consider a modified affine coupling layer $T_{\text{Aff}',[i]}(\cdot, t)$ as follows

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

where $\boldsymbol{s}_{i,t}, \boldsymbol{q}_{i,t}$ are modeled by neural network (4.5) and the only modification is the scaling of $\boldsymbol{s}_{i,t}$ and $\boldsymbol{q}_{i,t}$ with time t. Therefore $T_{\text{Aff}',[i]}$ is an identity when $t = 0$. Replacing $f_{[i]}$ with $T_{\text{Aff}',[i]}$ in Eqs. (4.6) and (4.8) we obtain the following expression for $t = 0$,

$$z = f_{\text{KRnet},\boldsymbol{\theta}}(\boldsymbol{x}, 0) = \boldsymbol{x} \quad \text{or} \quad p_{\text{KRnet},\boldsymbol{\theta}}(\boldsymbol{x}, 0) = p_{\boldsymbol{Z}}(\boldsymbol{x}). \tag{4.9}$$

If we choose the prior $p_{\boldsymbol{Z}}(z)$ the same as the initial distribution $p_0(z)$, the initial condition is satisfied exactly.

4.2 Adaptive Procedure of MCTNF

We initialize $S_{t,0} = \{(\boldsymbol{x}^{i,0}, t^{i,0})\}$ using uniform samples from a space-time domain $\Omega_0 \times [0, T]$, where Ω_0 is a finite volume, and specify $S_{\text{ic},0} = \{(\boldsymbol{x}_{\text{ic}}^{i,0}, p_0(\boldsymbol{x}_{\text{ic}}^{i,0}))\}$. Then we solve the optimization problem (4.3) to obtain the optimal $\boldsymbol{\theta}^{*,0}$. After that we update the training sets $S_{t,0}$ and $S_{\text{ic},0}$ using samples from $p_{\text{KRnet},\boldsymbol{\theta}^{*,0}}(\boldsymbol{x}, t)$. To be precise, we sample temporal points $\{t^{i,1}\}$ from a uniform distribution on $(0, T]$. For each $t^{i,1}$, we sample a latent normal random variable $\boldsymbol{Z}$ to obtain a sample $\boldsymbol{x}^{i,1}$ of $\boldsymbol{X} = f^{-1}_{\text{KRnet},\boldsymbol{\theta}^{*,0}}(\boldsymbol{Z}, t^{i,1})$. We then form $S_{t,1} = \{(\boldsymbol{x}^{i,1}, t^{i,1})\}$. $S_{\text{ic},1} = \{\boldsymbol{x}_{\text{ic}}^{i,1}\}$ can be obtained via the same procedure by letting $t = 0$, i.e. $\boldsymbol{x}_{\text{ic}}^{i,1} = f^{-1}_{\text{KRnet},\boldsymbol{\theta}^{*,0}}(z^i, 0)$. We then continue the training process with $S_{t,1}$ and $S_{\text{ic},1}$. The procedure is repeated after the second training is done. Such a strategy can be concluded as follows.

1. Generate initial training sets using uniform samples on $\Omega_0 \times (0, T]$ where $\Omega_0 \in \mathbb{R}^d$ and $|\Omega_0| < \infty$:

$$\begin{aligned}
&S_{t,0} = \{(\boldsymbol{x}^{i,0}, t^{i,0})\}_{i=1}^{N_t}, \quad t^{i,0} \sim \text{Uniform}(0, T], \quad \boldsymbol{x}^{i,0} \sim \text{Uniform}\ \ \Omega_0, \\
&S_{\text{ic},0} = \{(\boldsymbol{x}_{\text{ic}}^{i,0}, p_0(\boldsymbol{x}_{\text{ic}}^{i,0}))\}, \quad \boldsymbol{x}_{\text{ic}}^{i,0} \sim \text{Uniform}\ \ \Omega_0.
\end{aligned}$$

2. Train the temporal KRnet by solving the optimization problem (4.3) with training data $S_{t,0}$, $S_{\mathrm{ic},0}$ to obtain the optimal parameters $\boldsymbol{\theta}^{*,0}$:

$$\boldsymbol{\theta}^{*,0} = \arg\min_{\boldsymbol{\theta}} L_T(p_{\mathrm{KRnet},\boldsymbol{\theta}}(\boldsymbol{x}, t)).$$

3. Generate temporal samples from a uniform distribution on $(0, T]$ and spatial samples by conditioning on t using the distribution $p_{\mathrm{KRnet},\boldsymbol{\theta}^{*,0}}(\boldsymbol{x}|t)$ to obtain $S_{t,1}$, $S_{\mathrm{ic},1}$.

$$S_{t,1} = \{(\boldsymbol{x}^{i,1}, t^{i,1})\}_{i=1}^{N_t}, \quad t^{i,1} \sim \mathrm{Uniform}(0, T], \quad \boldsymbol{x}^{i,1} \sim p_{\mathrm{KRnet},\boldsymbol{\theta}^{*,0}}(\boldsymbol{x}|t = t^{i,1}),$$
$$S_{\mathrm{ic},1} = \{(\boldsymbol{x}_{\mathrm{ic}}^{i,1}, p_0(\boldsymbol{x}_{\mathrm{ic}}^{i,1}))\}, \quad \boldsymbol{x}_{\mathrm{ic}}^{i,1} \sim p_{\mathrm{KRnet},\boldsymbol{\theta}^{*,0}}(\boldsymbol{x}|t = 0).$$

   Set $S_{t,0} = S_{t,1}$, $S_{\mathrm{ic},0} = S_{\mathrm{ic},1}$.
4. Repeat steps 2-3 for N_{adaptive} times to get a convergent approximation.

Our algorithm for solving time-dependent fractional FPEs is given in Algorithm 3.

Algorithm 3 MCTNF

Input: maximum epoch number N_e, maximum iteration number N_{adaptive}, fractional order α, hyper-parameter r_ϵ, r_0, β_D, initial training data $S_t = \{(\boldsymbol{x}^i, t^i)\}_{i=1}^{N_t}$, $S_{\mathrm{ic}} = \{(\boldsymbol{x}_{\mathrm{ic}}^i, p_0(\boldsymbol{x}_{\mathrm{ic}}^i))\}_{i=1}^{N_{\mathrm{ic}}}$, $C_T = \{t_r^i\}_{i=1}^{N_r} \cup \{0\}_{i=1}^{N_{\mathrm{ic}}}$, tolerance ϵ_1, ϵ_2;
$L_{old} = 0$;
for $k = 0, \cdots, N_{\mathrm{adaptive}}$ **do**
  for $j = 1, \cdots, N_e$ **do**
    Divide S_t, S_{ic} into m batches $\{S_t^{\mathrm{ib}}\}_{\mathrm{ib}=1}^m$, $\{S_{\mathrm{ic}}^{\mathrm{ib}}\}_{\mathrm{ib}=1}^m$ randomly;
    for $ib = 1, \cdots, m$ **do**
      Compute the loss function $L_T(p_{\mathrm{KRnet},\boldsymbol{\theta}})$ for mini-batch data S_t^{ib}, $S_{\mathrm{ic}}^{\mathrm{ic}}$ and fractional order α;
      Update $\boldsymbol{\theta}$ using the Adam optimizer;
    $L_{new} = \frac{1}{m} \sum_{ib=1}^{m} L_T(p_{\mathrm{KRnet},\boldsymbol{\theta}})$;
    if $L_{new} < \epsilon_1$ or $|L_{old} - L_{new}| < \epsilon_2$ **then**
      Break;
    else
      $L_{old} = L_{new}$;
  $N_e = \gamma * N_e$;
  Sample from $t \sim \mathrm{Uniform}([0, T])$ and $p_{\mathrm{KRnet},\boldsymbol{\theta}}(\boldsymbol{x}|t)$ to update the training sets S_t, S_{ic};
Output: The predicted solution $p_{\mathrm{KRnet},\boldsymbol{\theta}}(\boldsymbol{x}, t)$.

5 Numerical Experiments

In this section, we present a series of comprehensive numerical tests to demonstrate the effectiveness of the proposed algorithms. To quantitatively evaluate the accuracy of the numerical solution $p_{\mathrm{KRnet},\boldsymbol{\theta}}$, we shall consider both the relative L_2 error $\|p^* - p_{\mathrm{KRnet},\boldsymbol{\theta}}\|_2 / \|p^*\|_2$ and the relative Kullback–Leibler (KL) divergence given by

$$\frac{D_{\mathrm{KL}}(p^* || p_{\mathrm{KRnet},\boldsymbol{\theta}})}{H(p^*)} = \frac{\mathbb{E}_{p^*}[\log(p^*/p_{\mathrm{KRnet},\boldsymbol{\theta}})]}{-\mathbb{E}_{p^*}[\log p^*]},$$

where $\mathbb{E}$ denotes the expectation and p^* the ground truth. We approximate the above relative L_2 error as follows,

$$\frac{\left\| p^* - p_{\mathrm{KRnet},\boldsymbol{\theta}} \right\|_2}{\left\| p^* \right\|_2} \approx \frac{\left(\int_{\tilde{\Omega}} (p^*(\boldsymbol{x}) - p_{\mathrm{KRnet},\boldsymbol{\theta}}(\boldsymbol{x}))^2 \mathrm{d}\boldsymbol{x}\right)^{1/2}}{\left(\int_{\tilde{\Omega}} (p^*(\boldsymbol{x}))^2 \mathrm{d}\boldsymbol{x}\right)^{1/2}} \approx \frac{\left(\sum_{i=1}^N (p^*(\boldsymbol{x}_i) - p_{\mathrm{KRnet},\boldsymbol{\theta}}(\boldsymbol{x}_i))^2\right)^{1/2}}{\left(\sum_{i=1}^N p^*(\boldsymbol{x}_i)^2\right)^{1/2}}.$$

Note that the computational domain is unbounded. To numerically compute the above error, we truncate the unbounded domain to $\tilde{\Omega}$ such that $\int_{\tilde{\Omega}} p^*(\boldsymbol{x})\mathrm{d}\boldsymbol{x} \geq 0.9$. And $\{\boldsymbol{x}_i\}_{i=1}^N$ are obtained from a uniform grid of nodes when the dimension, d, is equal to 2. In this case, we set N to a value such that the mesh size along each spatial dimension is 0.04. For higher dimensions, $d > 2$, $\{\boldsymbol{x}_i\}_{i=1}^N$ are chosen from a uniform distribution and N is set to 10^6. Similarly, we also approximate the above relative KL divergence by Monte Carlo integration, i.e.,

$$\frac{D_{\mathrm{KL}}(p^* || p_{\mathrm{KRnet},\boldsymbol{\theta}})}{H(p^*)} \approx \frac{\sum_{i=1}^{N_v} \left(\log p^*(\boldsymbol{x}_i) - \log p_{\mathrm{KRnet},\boldsymbol{\theta}}(\boldsymbol{x}_i)\right)}{-\sum_{i=1}^{N_v} \log p^*(\boldsymbol{x}_i)}.$$

Here $\boldsymbol{x}_i$ is drawn from the ground truth $p^*(\boldsymbol{x})$, and the amount of validation data is set to $N_v = 10^6$. For time-dependent problems, we obtain the relative L_2 error and the relative KL divergence according to the aforementioned formulas for each given t.

We shall employ hyperbolic tangent function (tanh) as the activation function. For each i, $\mathrm{NN}_{[i]}$ (see 3.15) is a feed forward neural network with two hidden layers. We use a half-half partition $\boldsymbol{x}_{[i]} = (\boldsymbol{x}_{[i],1}, \boldsymbol{x}_{[i],2})$, where $\boldsymbol{x}_{[i],1} \in \mathbb{R}^{\lfloor d/2 \rfloor}$ and $\boldsymbol{x}_{[i],2} \in \mathbb{R}^{d-\lfloor d/2 \rfloor}$ unless specified. We initialize all trainable parameters using Glorot initialization [13]. For the training procedure, we use the Adam optimizer [19]. All numerical tests are implemented with Pytorch.

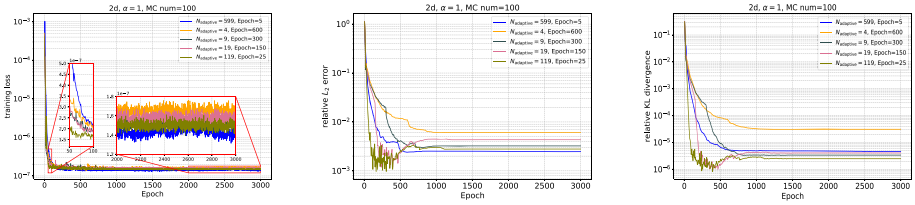
5.1 FPE with Only Fractional Laplacian

We start with a toy example with only the fractional Laplacian term. Consider the following 2D equation

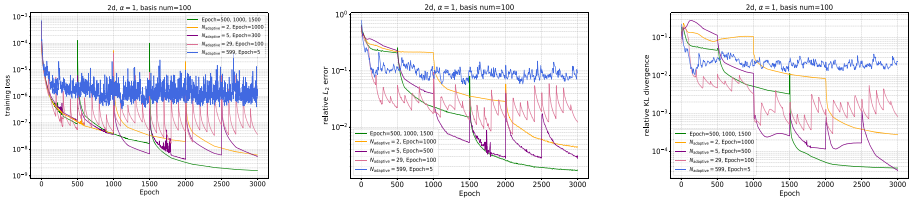
$$\begin{cases} (-\Delta)^{\alpha/2} p(\boldsymbol{x}) = f(\boldsymbol{x}), \quad \boldsymbol{x} \in \mathbb{R}^2, \\ \displaystyle\int_{\mathbb{R}^2} p(\boldsymbol{x})\mathrm{d}\boldsymbol{x} = 1, \quad p(\boldsymbol{x}) \geq 0, \end{cases} \tag{5.1}$$

where $f(\boldsymbol{x}) = -\frac{1}{2\pi} B(2,\alpha) 2^{-\frac{\alpha}{2}} \sigma^{-(2+\alpha)} {}_1F_1(\frac{2+\alpha}{2}; 1; -\frac{\|x-\mu\|_2^2}{2\sigma^2})$, $B(d,\alpha) = \frac{2^\alpha \Gamma((\alpha+d)/2)}{\Gamma(d/2)}$. The true solution is $p(\boldsymbol{x}) = \frac{1}{2\pi\sigma^2} \exp(-\frac{\|\boldsymbol{x}-\boldsymbol{\mu}\|_2^2}{2\sigma^2})$. We take $\alpha = 1$, $\boldsymbol{\mu} = (1,1)$, $\sigma = 2$.

For the NF, we take 8 affine coupling layers each with 32 hidden neurons. The initial training set is generated via the uniformly distributed points in the range $[0,6]^2$. Note that $\mathbb{E}_p[1_{[0,6]^2}] \approx 0.5$, meaning that we have only used about 50% information about the effective domain of the target distribution, where $\mathbb{E}_p$ indicates the expectation with respect to $p(\boldsymbol{x})$ and 1_Ω is an indicator function for $\Omega \subset \mathbb{R}^2$. The sample size is 5000, and the batch size is 1024. Both MCNF and GRBFNF methods are applied. For MCNF, the number of Monte Carlo samples used to approximate the fractional Laplacian is 100, with $r_0 = 4$ and $r_\epsilon = 0.01$. The initial learning rate is 0.001 with halving decay every 100 steps. For GRBFNF, the number of



(a) Different adaptive frequencies for MCNF.



(b) Different adaptive frequencies for GRBFNF.

Fig. 3 FPE with only fractional Laplacian. Left: training loss. Middle: the relative L_2 error. Right: the relative KL divergence

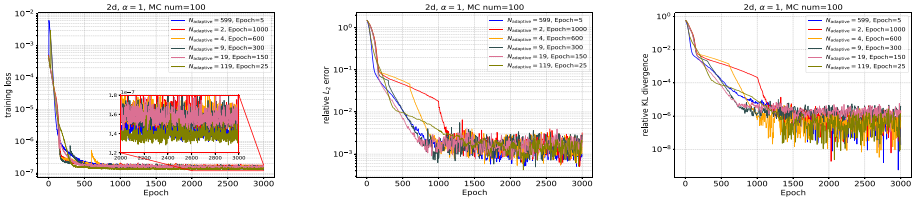


Fig. 4 Different adaptive frequencies for MCNF. The initial learning rate is 0.0001 with halving decay after 1500 epochs. Left: training loss. Middle: the relative L_2 error. Right: the relative KL divergence

basis functions is 100 and the initial center points of the basis functions are generated from a uniform distribution on $[0, 6]^2$. The learning rate is 0.01 with halving decay every 300 steps.

We first discuss the training strategy of MCNF and GRBFNF by adjusting the adaptive frequency of training. We present the training loss, relative L_2 error and relative KL divergence for different adaptive frequencies in Fig. 3. For MCNF method, following the current learning rate decay strategy, increasing the adaptive frequency leads to improved outcomes. However, for GRBFNF method, the adaptivity should not be activated until the current models is well trained, otherwise, the loss may be stuck in the transition period induced by the re-initialization of current models. On the other hand, in the case of MCNF, if we reduce the decay rate of the learning rate and set the initial learning rate to 0.0001 with halving decay after 1500 steps, our observation from Fig. 4 reveals that increasing the adaptive frequency appropriately can lead to expedited convergence. Moreover, it is noteworthy that there is minimal discernible difference in final accuracy across different adaptive frequencies.

Next, we focus on two experiments to investigate how adaptivity works. Specifically, for MCNF, we consider 599 adaptivity iterations, with 5 epochs per iteration. And for GRBFNF, we choose 2 adaptivity iterations with increasing epochs, 500 epochs in the first iteration, 1000 epochs in the first adaptivity iteration and 2000 epochs in the second adaptivity iteration. The time costs of MCNF and GRBFNF are 64 min, and 46 min respectively. The training points as well as the center points of the basis functions for different adaptivity iteration

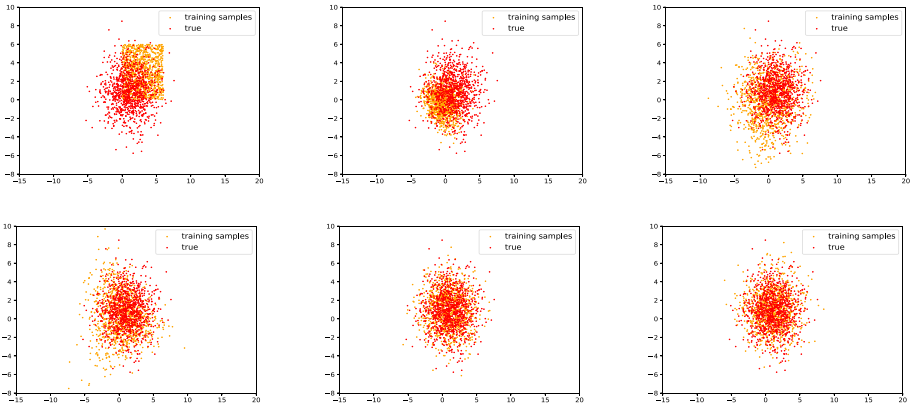
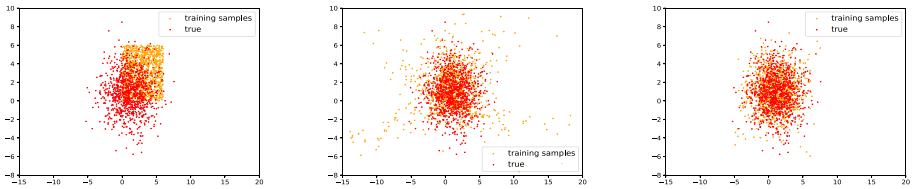
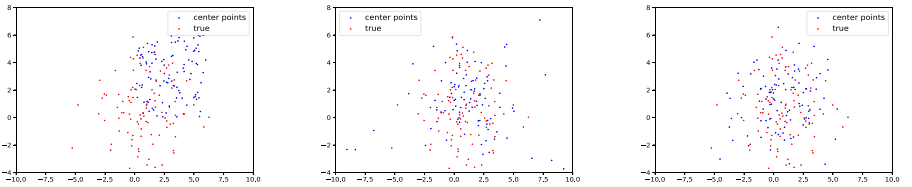


Fig. 5 Distribution of training samples at different adaptivity iteration numbers in MCNF. From left to right and from top to bottom, $k = 0, 1, 2, 4, 29, 249$



(a) Training samples at different adaptive iterations. From left to right, $k = 0, 1, 2$.

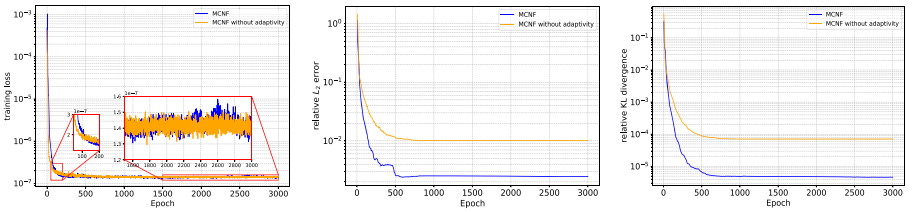


(b) Center points of GRBFs at different adaptive iterations.From left to right, $k = 0, 1, 2$.

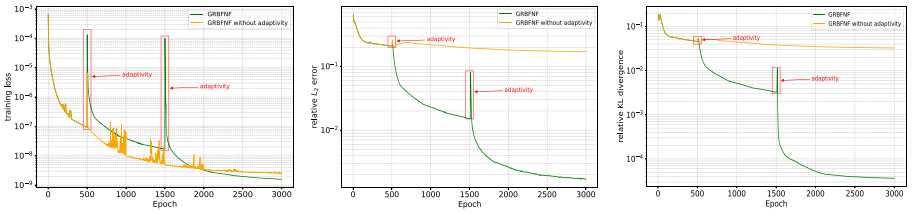
Fig. 6 Adaptivity of GRBFNF for the training set and the centers of GRBF basis functions

numbers are presented in Figs. 5 and 6. One can clearly observe that the training points and center points of the basis functions become increasingly closer to the ground truth as the iteration number increases, showing the effectiveness of the adaptive sampling scheme.

What's more, we compare our adaptive methods with non-adaptive methods in Fig. 7. It can be seen that the accuracy of adaptive algorithm is higher than that of the non-adaptive algorithm, especially for GRBFNF. The computational area of the non-adaptive method is always $[0, 6]^2$, which certainly affects the performance outside this area. That is to say, the numerical solution can approximate the ground truth well inside a predetermined area while failing to capture the information outside this area especially when the prior knowledge is not enough to design a suitable computational area. We drawn the ground truth in Fig. 8. The comparison between the predicted solution and the exact solution is presented in Fig. 9, from which we can clearly observe that the non-adaptive methods show larger errors outside the computational area $[0, 6]^2$. On the other hand, our methods update the training points adaptively, which can effectively alleviate the limitation of a fixed computational area. Both



(a) MCNF, $\alpha = 1$.



(b) GRBFNF, $\alpha = 1$

Fig. 7 Comparison between adaptive and non-adaptive methods. Top row: MCNF. Bottom row: GRBFNF. Left: training loss. Middle: relative L_2 error. Right: relative KL divergence

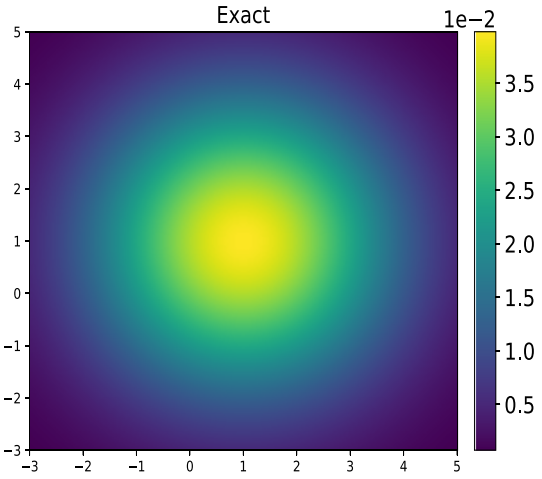
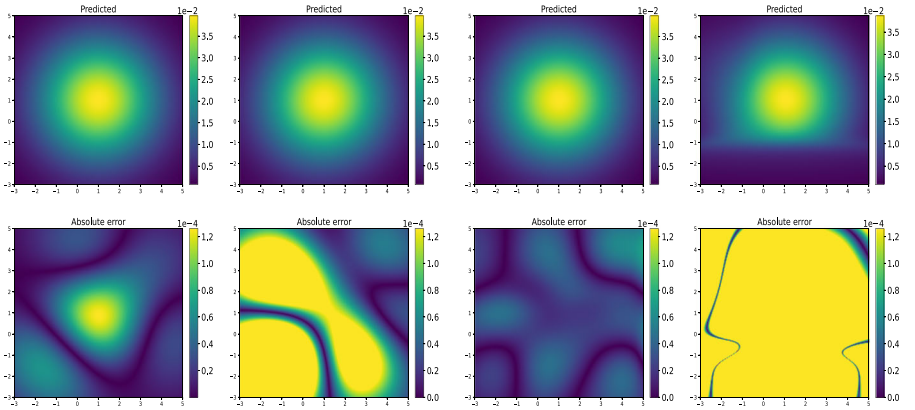


Fig. 8 The reference solution of FFP with only the fractional Laplacian term

the solutions of MCNF and GRBFNF yield excellent agreement with the exact solution. The relative L_2 error and the relative KL divergence with different adaptivity iteration numbers are also provided in Fig. 10. The relative L_2 error of GRBFNF is smaller than that of MCNF while the relative KL divergence of GRBFNF is larger than that of MCNF.

Finally, we take $[-3, 3]^2$ to replace the above initial sampling area $[0, 6]^2$ and repeat the experiments to test the performance of MCNF and GRBFNF for fractional FPEs with different fractional orders α. A total of 199 adaptivity iterations are carried out, with each iteration consisting of 5 epochs. The results are presented in Fig. 11, where we also display the accuracy of the Monte Carlo sampling method to compute the associated fractional Laplacian. The numerical error of approximating the fractional Laplacian is defined by $\frac{\sum_i |(-\Delta)^{\frac{\alpha}{2}}[p](\boldsymbol{x}_i) - \mathcal{M}[p](\boldsymbol{x}_i)|^2}{\sum_i |(-\Delta)^{\frac{\alpha}{2}}[p](\boldsymbol{x}_i)|^2}$ where $\mathcal{M}[p]$ denotes the numerical approximation. Both MCNF and GRBFNF achieve good agreement with the ground truth for $\alpha = 0.5, 1, 1.5, 1.8$.



(a) MCNF (b) MCNF without adaptivity (c) GRBFNF (d) GRBFNF without adaptivity

Fig. 9 Comparison between the predicted solutions and the reference solutions. Top row: numerical solution. Bottom row: Absolute error between the numerical solution and the exact solution

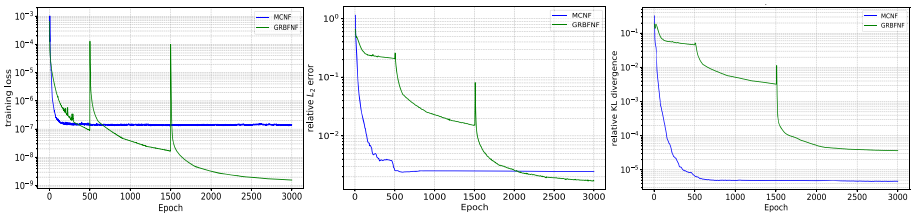


Fig. 10 Convergence behavior of MCNF and GRBFNF. Left: training loss. Middle: relative L_2 error. Right: relative KL divergence

5.2 Bimodal Distribution

To test the performance of MCNF and GRBFNF in handling a bimodal distribution, we consider

$$\begin{cases} \nabla \cdot (\boldsymbol{g}(\boldsymbol{x})p(\boldsymbol{x})) + 0.05\Delta p(\boldsymbol{x}) - (-\Delta)^{\alpha/2}p(\boldsymbol{x}) = f(\boldsymbol{x}), \quad \boldsymbol{x} \in \mathbb{R}^2, \\ \displaystyle\int_{\mathbb{R}^2} p(\boldsymbol{x})\mathrm{d}\boldsymbol{x} = 1, \quad p(\boldsymbol{x}) \geq 0, \end{cases} \tag{5.2}$$

where $\boldsymbol{g}(x) = 0.2\boldsymbol{x}$,

$$f(\boldsymbol{x}) = \frac{1}{5\pi}\nabla \cdot \left(\exp(-2\|\boldsymbol{x} - \mathbf{1}_2\|_2^2)\right) - \frac{1}{\pi}B(2,\alpha)2^{\frac{\alpha}{2}}\left({}_1F_1\left(\frac{2+\alpha}{2}; 1; -2\|\boldsymbol{x}\|_2^2\right) + {}_1F_1\left(\frac{2+\alpha}{2}; 1; -2\|\boldsymbol{x} - \mathbf{1}_2\|_2^2\right)\right).$$

The true solution is

$$p(\boldsymbol{x}) = \frac{1}{\pi}\Big(\exp\big(-2\|\boldsymbol{x}\|_2^2\big) + \exp\big(-2\|\boldsymbol{x} - \mathbf{1}_2\|_2^2\big)\Big).$$

For the NF, we take $L = 8$ affine coupling layers each with 32 hidden neurons. The initial training set is generated via uniformly distributed points in $[-3, 3]^2$. The sample size is 5000

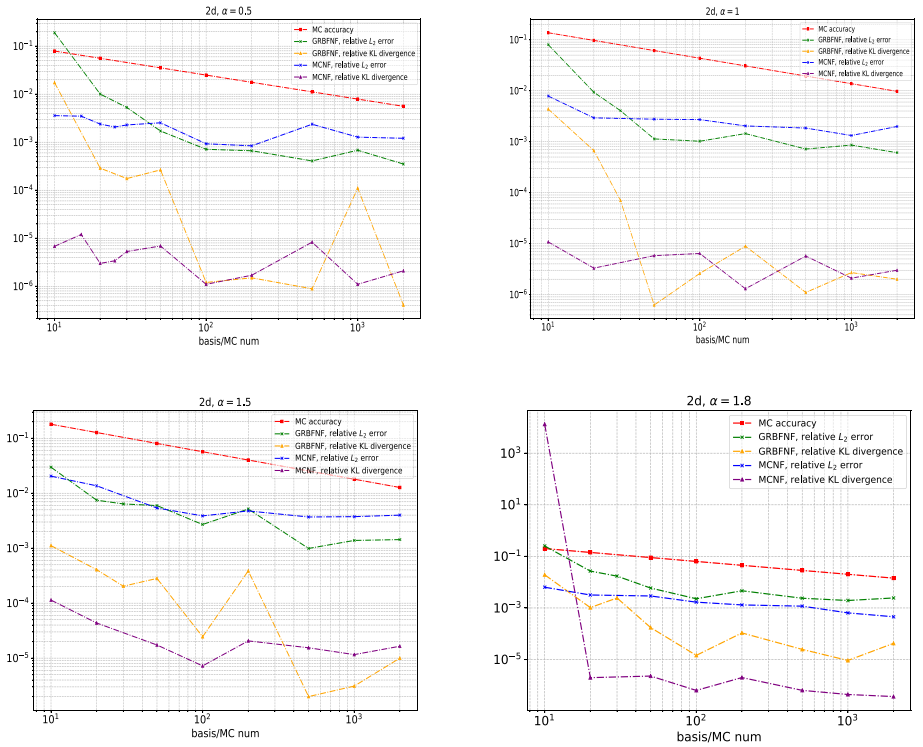
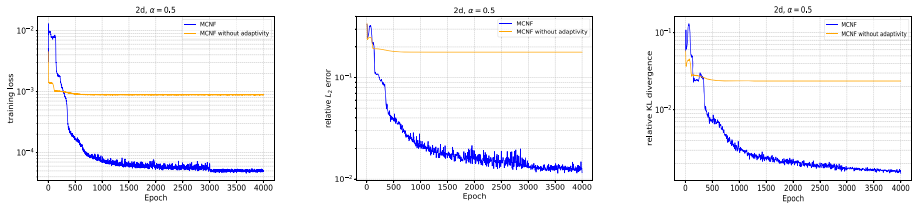


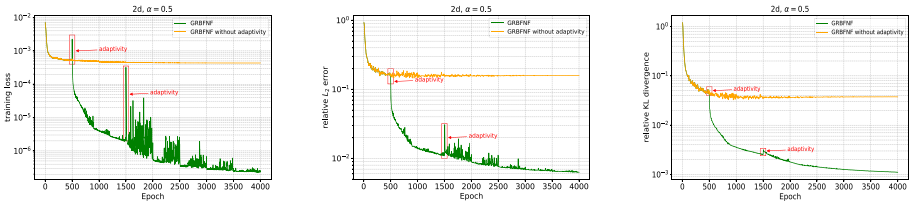
Fig. 11 Error decay of MCNF and GRBFNF for different α in terms of the number of MC samples and GRBF basis functions

and the batch size is set to 1024. Both MCNF and GRBFNF are applied. For MCNF, the number of the samples used to approximate fractional Laplacian is 100, $r_0 = 0.3$, $r_\epsilon = 0.0001$. 799 adaptivity iterations with 5 epochs for each adaptivity iteration are conducted. The initial learning rate is 0.001 with 80% decay every 3000 steps. For GRBFNF, the number of the basis functions is 100 and the initial center points of the basis functions are generated from a uniform distribution in the area $[-3, 3]^2$. 2 adaptivity iterations with increasing epochs are conducted for this problem, i.e. 500 epochs for the initial iteration, 1000 epochs for the first adaptivity iteration, and 2500 epochs for the second iteration. The learning rate is 0.01 with halving decay every 300 steps, and it is reset to 0.005 after each adaptivity step.

We also apply MCNF and GRBFNF without adaptivity to solve this problem. The relative L_2 error and the relative KL divergence are provided in Fig. 12, which once again verifies the strength of the adaptive methods. Although in this example, the unknown PDF is mainly concentrated in the initial sampling area which is different from the previous example, uniform samples used by non-adaptive methods fail to yield an accurate approximation, and the adaptive sampling may improve the results by at least one order of magnitude. The exact solution is presented in Fig. 13. The comparison between the predicted solution and the exact solution is presented in Fig. 14. Both MCNF and GRBFNF can approximate the exact solution well. The training loss, the relative L_2 error, and the relative KL divergence are presented in Fig. 15. GRBFNF shows better performance than MCNF in this example.



(a) MCNF, $\alpha = 0.5$.



(b) GRBFNF, $\alpha = 0.5$

Fig. 12 Convergence behavior of MCNF and GRBFNF with and without adaptive sampling. Left: training loss. Middle: relative L_2 error. Right: relative KL divergence

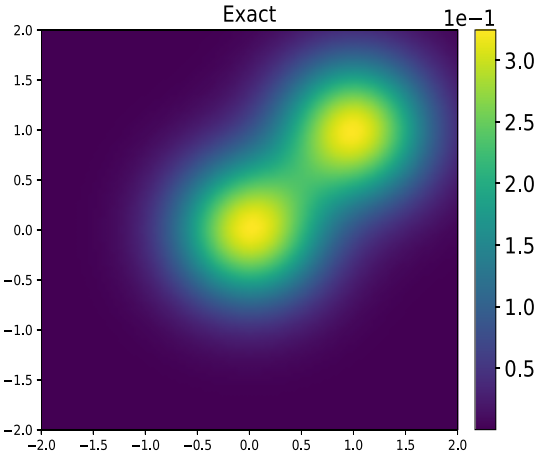


Fig. 13 The ground truth of bimodal distribution

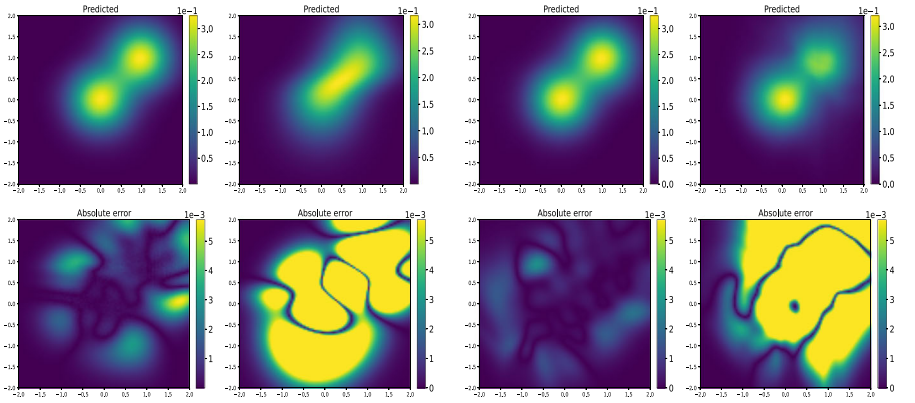
5.3 High Dimensional Fractional Fokker–Planck Equations

In this part, we consider a high-dimensional FPE.

$$\begin{cases} \nabla \cdot (\boldsymbol{g}(\boldsymbol{x})p(\boldsymbol{x})) + \Delta p(\boldsymbol{x}) - (-\Delta)^{\alpha/2} p(\boldsymbol{x}) = f(\boldsymbol{x}), \quad \boldsymbol{x} \in \mathbb{R}^d, \\ \int_{\mathbb{R}^2} p(\boldsymbol{x}) \mathrm{d}\boldsymbol{x} = 1, \quad p(\boldsymbol{x}) \geq 0. \end{cases} \tag{5.3}$$

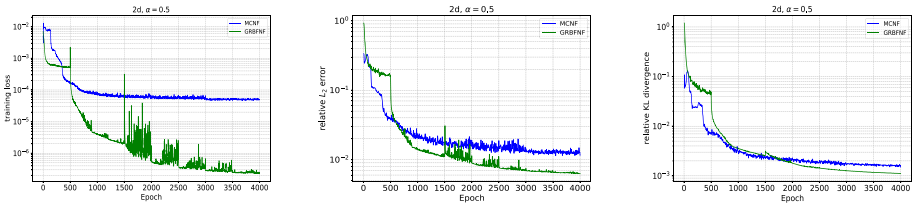
where $\boldsymbol{g}(\boldsymbol{x}) = \frac{\boldsymbol{x}-\boldsymbol{\mu}}{\sigma^2}$, the corresponding analytic solution is

$$p(\boldsymbol{x}) = \mathcal{N}(\boldsymbol{\mu}, \Sigma) = \frac{1}{(2\pi)^{d/2}\sigma^d} \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^{\mathrm{T}} \Sigma^{-1} (\boldsymbol{x}-\boldsymbol{\mu})\right). \tag{5.4}$$



(a) MCNF (b) MCNF without adaptivity (c) GRBFNF (d) GRBFNF without adaptivity

Fig. 14 Comparison between the predicted solutions and the reference solutions. Top row: numerical solution. Bottom row: Absolute error between the numerical solution and the exact solution



(a) $\alpha = 0.5$

Fig. 15 Convergence behavior of MCNF and GRBFNF for the bimodal distribution case. Left: training loss. Middle: relative L_2 error. Right: relative KL divergence. The unit of time is second

We take $d = 4, 6, 8$, $\boldsymbol{\mu} = \mathbf{1}_d$ and $\Sigma = \sigma^2 \boldsymbol{I}_d$, where $\boldsymbol{I}_d$ is a d-dimensional identity matrix and $\sigma = 2$.

For high-dimensional problems, the PDF and the associated loss function may be too small, which results in numerical underflow issues. For the sake of numerical stability, we magnify the solution by multiplying a large enough constant C. Thus Cp satisfies

$$\frac{\partial (Cp)}{\partial t} = \mathcal{L}(Cp) - (-\Delta)^{\alpha/2}(Cp). \tag{5.5}$$

Actually, the values of C used here are 1 for $d = 4$, 10 for $d = 6$ and 200 for $d = 8$.

For the NF, we take $L = 8$ affine coupling layers with 64 hidden neurons. The initial training set is generated via uniformly distributed points in $[-3, 5]^d$. The sample size is 50000. The batch size is set to 4096. We employ MCNF for solving this problem. GRBFNF is harder to train in high-dimensional cases since its structure is more complex. The number of samples used to approximate the fractional Laplacian is 200, with $r_0 = 0.3$ and $\epsilon = 0.0001$. We take a half-half partition here. For $d = 4, 6, 99$ adaptivity iterations with 20 epochs for each adaptivity iteration are conducted. The learning rate is 0.001 with halving decay every 300 steps. For $d = 8, 19$ adaptivity iterations with 200 epochs for each adaptivity iteration are conducted. The learning rate is 0.001 with halving decay every 1000 steps. The comparisons between the MCNF solutions and the true solutions are presented in Figs. 16, 17, 18, all of

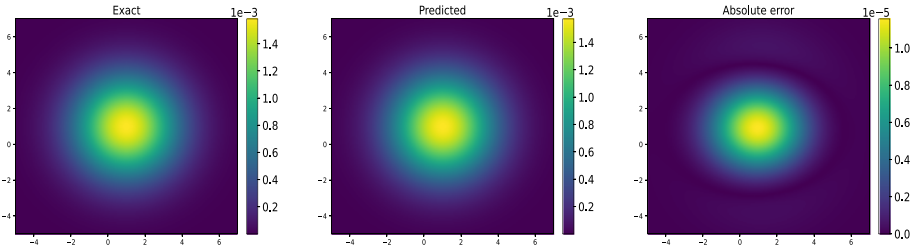


Fig. 16 MCNF for a 4-dimensional problem, where the first two dimensions are plotted with $x_3 = x_4 = 1$. Predicted solution versus the reference solution. Left: exact solution. Middle: prediction. Right: absolute error

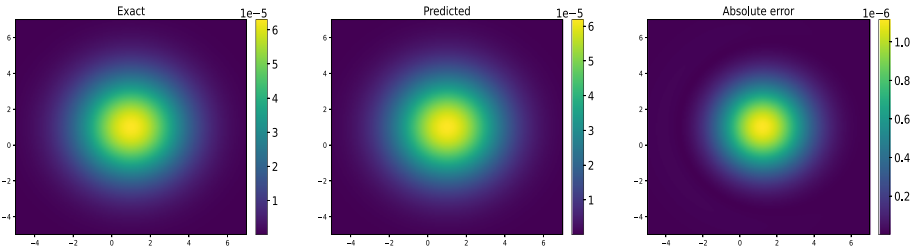


Fig. 17 MCNF for a 6-dimensional problem, where the first two dimensions are plotted with $x_3 = x_4 = x_5 = x_6 = 1$. Predicted solution versus the reference solution. Left: exact solution. Middle: prediction. Right: absolute error

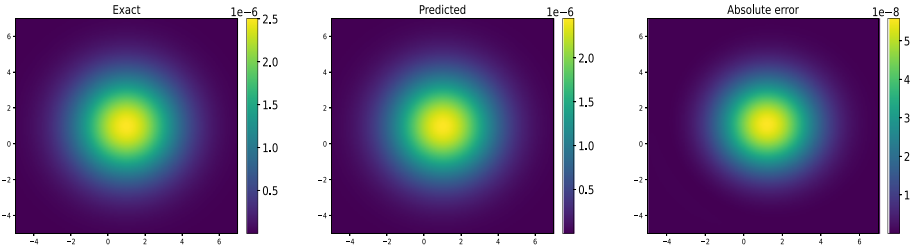


Fig. 18 MCNF for the 8-dimensional problem, where the first two dimensions are plotted at $x_3 = x_4 = \cdots = x_8 = 1$. Predicted solution versus the reference solution. Left: exact solution. Middle: prediction. Right: absolute error

which show great performance of our approach. We also present the relative L_2 error and the relative KL divergence in Fig. 19.

5.4 Time-Dependent Fractional FPE: Cauchy Distribution

We consider the following stochastic process,

$$\mathrm{d}\mathbf{X}_t = \mathrm{d}\mathbf{L}_t^{\alpha}, \quad \alpha = 1. \tag{5.6}$$

For $d = 2$, the corresponding fractional Fokker–Planck equation is

$$\begin{cases} \dfrac{\partial p}{\partial t} = -(-\Delta)^{\alpha/2} p, \quad \alpha = 1, \\ p(\boldsymbol{x}, 0) = p_0(\boldsymbol{x}). \end{cases} \tag{5.7}$$

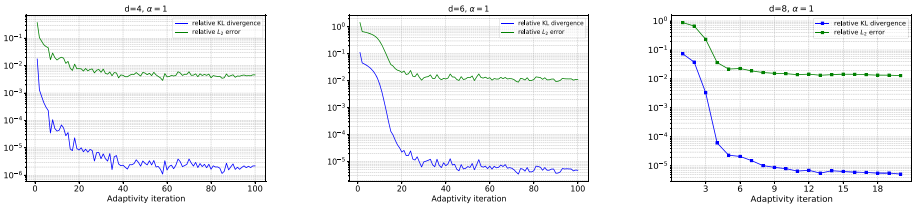


Fig. 19 Convergence behavior of MCNF for high-dimensional FPEs. Left: $d = 4$. Middle: $d = 6$. Right: $d = 8$

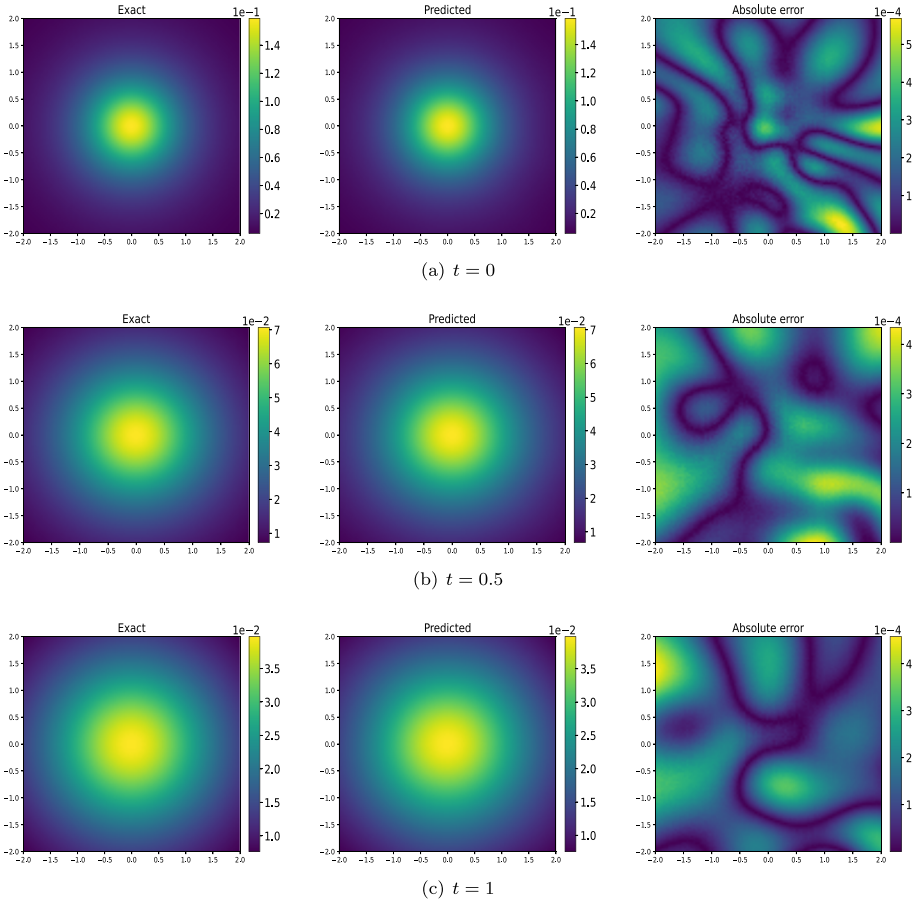


Fig. 20 The predicted solutions versus the reference solutions for MCTNF at $t = 0, 0.5, 1$

For the initial condition $p_0(\boldsymbol{x}) = \frac{1}{2\pi(1+\|\boldsymbol{x}\|_2^2)^{3/2}}$, the solution of Eq. (5.7) is $p(\boldsymbol{x}, t) = \frac{t+1}{2\pi\left((t+1)^2+\|\boldsymbol{x}\|_2^2\right)^{3/2}}$, where $\boldsymbol{x} \in \mathbb{R}^2$ and $t \in [0, 1]$.

For the NF, we take $L = 8$ affine coupling layers with 32 hidden neurons. The initial spatial samples are drawn from a uniform distribution in $[-3, 3]^2$, and temporal samples are generated from a uniform distribution in $[0, 1]$. The sample size is 100000, and the batch

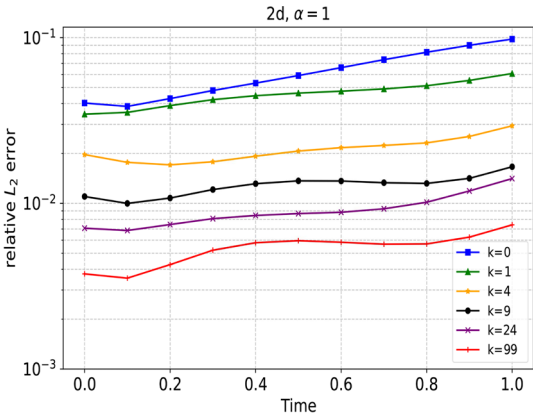


Fig. 21 The relative L_2 errors of MCTNF

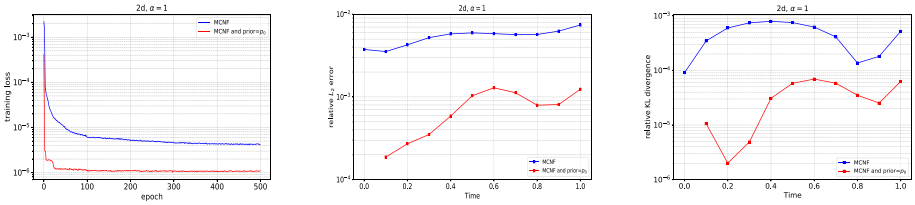


Fig. 22 Comparison between the original MCTNF and the modified MCTNF. Left: training loss. Middle: realtive L_2 error. Right panel: relative KL divergence

size is set to be 4096. For MCNF, the number of samples used to approximate the fractional Laplacian is 100 with $r_0 = 1$ and $r_\epsilon = 0.01$. 99 adaptivity iterations with 5 epochs for each adaptivity iteration are conducted. The initial learning rate is 0.001 with halving decay every 100 steps. One can observe a good agreement between the predicted solutions and the ground truth from the Fig. 20. The relative L_2 error and the relative KL divergence against time t for different adaptive iterations are also provided in Fig. 21, which indicates the effectiveness of adaptivity. We present the comparison of the relative error between the original MCTNF and modified MCTNF in Fig. 22. The modified MCTNF indeed improves the approximation. It is worth mentioning that the numerical error seems to increase as time progresses. We will investigate this issue in the subsequent work.

6 Conclusions

We have proposed flow-based adaptive algorithms for solving fractional FPEs. The core idea is to model the unknown PDF by a normalizing flow which yields an explicit PDF model as well as the corresponding exact random samples. For stationary FPEs, we proposed two methods: MCNF and GRBFNF. It is usually hard to choose a suitable computational area for unbounded problems. Our methods alleviate this difficulty by adaptively updating the training points. We train the MCNF model or GRBFNF model with current training points, and generate new training points using the current approximate solution. Then the training sets and the solution approximation are updated alternately. For time-dependent FPEs, we proposed MCTNF, where we modified the affine coupling layer to satisfy the initial condition exactly to improve the accuracy. Our approaches are validated by numerical experiments for

both stationary and time-dependent FPEs. Compared to non-adaptive methods, both MCNF and GRBFNF may improve the accuracy by at least one order of magnitude. From the numerical results, GRBFNF appeals to be more suitable for low-dimensional problems, while MCNF demonstrates greater flexibility for high-dimensional problems. The main difference between MCNF and GRBFNF is how the fractional Laplacian is approximated. MCNF uses the Monte Carlo approximation, whereas GRBFNF relies on the GRBF approximation of the solution. GRBFNF is more effective for low-dimensional problems since the GRBF approximation is a linear model. MCNF performs better for high-dimensional problems because of the weak dependence of the Monte Carlo method on dimensionality. However, to further reduce the statistical error of the MC approximation of the fractional Laplacian, we may consider variance reduction techniques, which will be left for future study.

Acknowledgements This work is supported by the NSF of China (No. 12288201), the Strategic Priority Research Program of Chinese Academy of Sciences (Grant No. XDA25010404), the National Key R&D Program of China (2020YFA0712000), the youth innovation promotion association (CAS), and Henan Academy of Sciences. The second author is supported by NSF Grant DMS-1913163.

Data Availibility The datasets generated during and/or analysed during the current study are available from the corresponding author on reasonable request.

Declarations

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

References

1. Ayi, N., Herda, M., Hivert, H., Tristani, I.: On a structure-preserving numerical method for fractional Fokker–Planck equations. arXiv preprint arXiv:2107.13416 (2021)
2. Brunton, S.L., Noack, B.R., Koumoutsakos, P.: Machine learning for fluid mechanics. Annu. Rev. Fluid Mech. **52**, 477–508 (2020)
3. Burkardt, J., Wu, Y., Zhang, Y.: A unified meshfree pseudospectral method for solving both classical and fractional PDEs. SIAM J. Sci. Comput. **43**(2), A1389–A1411 (2021)
4. Chen, J., Du, R., Wu, K.: A Comparison Study of Deep Galerkin Method and Deep Ritz Method for Elliptic Problems with Different Boundary Conditions. Commun. Math. Res. **36**, 354–376 (2020)
5. Chen, X., Yang, L., Duan, J., Karniadakis, G.E.: Solving inverse stochastic problems from discrete particle observations using the Fokker-Planck equation and physics-informed neural networks. SIAM J. Sci. Comput. **43**(3), B811–B830 (2021)
6. Deng, W.: Finite element method for the space and time fractional Fokker–Planck equation. SIAM J. Numer. Anal. **47**(1), 204–226 (2009)
7. Dinh, L., Krueger, D., Bengio, Y.: NICE: non-linear independent components estimation. arXiv preprint arXiv:1410.8516 (2014)
8. Dinh, L., Sohl-Dickstein, J., Bengio, S.: Density estimation using real NVP. arXiv preprint arXiv:1605.08803 (2016)
9. Ditlevsen, P.D.: Observation of α-stable noise induced millennial climate changes from an ice-core record. Geophys. Res. Lett. **26**(10), 1441–1444 (1999)
10. Elowitz, M.B., Levine, A.J., Siggia, E.D., Swain, P.S.: Stochastic gene expression in a single cell. Science **297**(5584), 1183–1186 (2002)
11. Feng, X., Zeng, L., Zhou, T.: Solving time dependent Fokker–Planck equations via temporal normalizing flow. Commun. Comput. Phys. **32**, 401–423 (2022)
12. Gao, T., Duan, J., Li, X.: Fokker–Planck equations for stochastic dynamical systems with symmetric lévy motions. Appl. Math. Comput. **278**, 1–20 (2016)
13. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the 13th International Conference on Artificial Intelligence and Statistics, pp. 249–256. JMLR Workshop and Conference Proceedings (2010)

14. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. In: Advances in Neural Information Processing Systems, vol. 27 (2014)
15. Guo, L., Wu, H., Yu, X., Zhou, T.: Monte Carlo fPINNs: Deep learning method for forward and inverse problems involving high dimensional fractional partial differential equations. Comput. Methods Appl. Mech. Eng. (2022)
16. Guo, L., Wu, H., Zhou, T.: Normalizing field flows: solving forward and inverse stochastic differential equations using physics-informed flow models. J. Comput. Phys. **461**, 111202 (2022)
17. Han, J., Jentzen, A., Weinan, E.: Solving high-dimensional partial differential equations using deep learning. Proc. Nat. Acad. Sci. **115**(34), 8505–8510 (2018)
18. Iten, R., Metger, T., Wilming, H., Del Rio, L., Renner, R.: Discovering physical concepts with neural networks. Phys. Rev. Lett. **124**(1), 010508 (2020)
19. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
20. Kingma, D.P., Dhariwal, P.: Glow: generative flow with invertible 1x1 convolutions. arXiv preprint arXiv:1807.03039 (2018)
21. Kingma, D.P., Welling, M.: Auto-encoding variational bayes. arXiv preprint arXiv:1312.6114 (2013)
22. Liu, S., Li, W., Zha, H., Zhou, H.: Neural parametric Fokker–Planck equations. SIAM J. Sci. Comput. **60**(3), 1385–1449 (2022)
23. Meng, X., Karniadakis, G.E.: A composite neural network that learns from multi-fidelity data: application to function approximation and inverse PDE problems. J. Comput. Phys. **401**, 109020 (2020)
24. Pang, G., Lu, L., Karniadakis, G.: fPINNs: fractional physics-informed neural networks. SIAM J. Sci. Comput. **41**, A2603–A2626 (2019)
25. Papamakarios, G., Nalisnick, E., Rezende, D.J., Mohamed, S., Lakshminarayanan, B.: Normalizing flows for probabilistic modeling and inference. J. Mach. Learn. Res. **22**, 1–64 (2021)
26. Qin, T., Chen, Z., Jakeman, J.D., Xiu, D.: Deep learning of parameterized equations with applications to uncertainty quantification. Int. J. Uncertain. Quantif. **11**(2) (2021)
27. Raissi, M., Perdikaris, P., Karniadakis, G.E.: 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)
28. Raissi, M., Yazdani, A., Karniadakis, G.E.: Hidden fluid mechanics: learning velocity and pressure fields from flow visualizations. Science **367**(6481), 1026–1030 (2020)
29. Rezende, D., Mohamed, S.: Variational inference with normalizing flows. In: International Conference on Machine Learning, pp. 1530–1538. PMLR (2015)
30. 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)
31. Sheng, H., Yang, C.: PFNN-2: A domain decomposed penalty-free neural network method for solving partial differential equations. arXiv preprint arXiv:2205.00593 (2022)
32. Shlesinger, M., Zaslavsky, G., Frisch, U.: Lévy Flights and Related Topics in Physics. Lecture Notes in Physics, Springer, Berlin (1995)
33. Sirignano, J., Spiliopoulos, K.: DGM: A deep learning algorithm for solving partial differential equations. J. Comput. Phys. **375**, 1339–1364 (2018)
34. Sukumar, N., Srivastava, A.: Exact imposition of boundary conditions with distance functions in physics-informed deep neural networks. Comput. Methods Appl. Mech. Eng. **389**, 114333 (2022)
35. Tang, K., Wan, X., Liao, Q.: Deep density estimation via invertible block-triangular mapping. Theor. Appl. Mech. Lett. **10**(3), 143–148 (2020)
36. Tang, K., Wan, X., Liao, Q.: Adaptive deep density approximation for Fokker–Planck equations. J. Comput. Phys. **457**, 111080 (2022)
37. 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)
38. Terrell, G.R., Scott, D.W.: Variable kernel density estimation. Ann. Stat. 1236–1265 (1992)
39. Weinan, E., Yu, B.: The deep Ritz method: a deep learning-based numerical algorithm for solving variational problems. Commun. Math. Stat. **6**(1), 1–12 (2018)
40. Xu, Y., Zan, W., Jia, W., Kurths, J.: Path integral solutions of the governing equation of SDEs excited by Lévy white noise. J. Comput. Phys. **394**, 41–55 (2019)
41. Yang, L., Meng, X., Karniadakis, G.E.: B-PINNs: Bayesian physics-informed neural networks for forward and inverse PDE problems with noisy data. J. Comput. Phys. **425**, 109913 (2021)
42. Yang, L., Zhang, D., Karniadakis, G.E.: Physics-informed generative adversarial networks for stochastic differential equations. SIAM J. Sci. Comput. **42**(1), A292–A317 (2020)
43. Yang, Y., Perdikaris, P.: Adversarial uncertainty quantification in physics-informed neural networks. J. Comput. Phys. **394**, 136–152 (2019)

44. Zan, W., Xu, Y., Kurths, J., Chechkin, A., Metzler, R.: Stochastic dynamics driven by combined Lévy-Gaussian noise: fractional Fokker–Planck–Kolmogorov equation and solution. J. Phys. A Math. Theor. **53** (2020)
45. Zang, Y., Bao, G., Ye, X., Zhou, H.: Weak adversarial networks for high-dimensional partial differential equations. J. Comput. Phys. **411**, 109409 (2020)
46. Zhang, H., Jiang, X., Yang, X.: A time-space spectral method for the time-space fractional Fokker–Planck equation and its inverse problem. Appl. Math. and Comput. **320**, 302–318 (2018)
47. Zhang, H., Xu, Y., Li, Y., Kurths, J.: Statistical solution to SDEs with α-stable Lévy noise via deep neural network. Int. J. Dyn. Control **8**(4), 1129–1140 (2020)
48. Zhang, L., Han, J., Wang, H., Car, R., Weinan, E.: Deep potential molecular dynamics: a scalable model with the accuracy of quantum mechanics. Phys. Rev. Lett. **120**(14), 143001 (2018)
49. 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)