

RESEARCH

# High-dimensional density estimation with tensorizing flow



Yinuo Ren<sup>1\*</sup> , Hongli Zhao<sup>2</sup>, Yuehaw Khoo<sup>2</sup> and Lexing Ying<sup>3</sup>

\*Correspondence:

yinuoren@stanford.edu

<sup>1</sup>Institute for Computational and  
Mathematical Engineering  
(ICME), Stanford University,  
Stanford, CA 94305, USA

Full list of author information is  
available at the end of the article

## Abstract

We propose the tensorizing flow method for estimating high-dimensional probability density functions from observed data. Our method combines the optimization-less feature of the tensor-train with the flexibility of flow-based generative models, providing an accurate and efficient approach for density estimation. Specifically, our method first constructs an approximate density in the tensor-train form by efficiently solving the tensor cores from a linear system based on kernel density estimators of low-dimensional marginals. Subsequently, a continuous-time flow model is trained from this tensor-train density to the observed empirical distribution using maximum likelihood estimation. Numerical results are presented to demonstrate the performance of our method.

**Keywords:** Maximum likelihood estimation, Density estimation, Tensor-train, Flow-based generative modeling

## 1 Introduction

Density estimation is fundamental in statistical inference, machine learning, and data analysis. It aims to infer the underlying probability density function directly from observed data. However, estimating high-dimensional probability distributions remains a major theoretical and computational challenge.

In recent years, deep generative modeling has emerged as a powerful method for approximating high-dimensional densities from many samples [8]. Among them, flow-based generative models [19, 57] have shown promising results by constructing a parameterized flow from a normal distribution to the target distribution. However, the limitation of confining the source distribution to a normal distribution (or a mixture of Gaussians) can be restrictive, especially when dealing with singular or multimodal distributions.

The tensor-train network [52], also known as the matrix product state (MPS) [55] in the physics literature, is a class of tensor network structures widely used to model high-dimensional functions in physics, such as the wavefunction of many-body quantum states under certain correlation decay assumptions [11], and more recently in many other contexts [1, 32, 45]. Various methods have been proposed to efficiently obtain the tensor-train representation of closed-form high-dimensional functions by utilizing the techniques of linear algebra [51], optimization [60] and parallel computing [62]. More recently, [38]

proposed an optimization-less linear algebra framework for recovering the tensor-train representation of a density directly from its empirical distribution. However, tensor-trains may be inflexible when it comes to machine learning applications. One challenge of working with tensor-trains is the need to pre-determine the ordering of the variables based on their correlations, which can be difficult in practice. In addition, a sub-optimal ordering may lead to larger tensor-train ranks, increased storage, and higher computational complexity.

### 1.1 Contributions

Motivated by the strengths and limitations of flow-based generative models and the tensor-train representation, we propose a new framework that combines the benefits of both approaches for density estimation.

- Given a set of samples, we first construct a low-rank tensor-train representation as an approximation to the base distribution. To cope with data sparsity in high-dimensional settings, we estimate the required low-order marginals using kernel density estimation during the construction.
- We then adopt an ODE-based continuous-time flow model that maps the tensor-train base distribution to the target distribution. The flow model is parameterized by a neural network and trained using maximum likelihood estimation with both forward and inverse maps computed efficiently.

Following [40], we refer to this two-stage method as the *tensorizing flow* approach for density estimation, i.e., applying a neural network transformation to a base distribution represented by a tensor-train.

### 1.2 Related works

#### *Deep generative modeling*

Deep generative models have recently gained a lot of attention for approximating high-dimensional probability distributions. The Boltzmann machine [34,35], one of the earliest deep generative models, is based on a particular energy function form of the probability distribution. Variational autoencoders (VAEs) [43,58] encode observations in a regularized latent space, while generative adversarial networks (GANs) [29] involve a generator and a discriminator trained jointly as a minimax game [61]. Autoregressive likelihood models [5,27,47], on the other hand, are based on the chain rule of probability.

Another popular approach is flow-based generative modeling [19,57], which is based on a sequence of diffeomorphisms between a known base distribution and a target distribution of interest. Unlike the autoregressive models and VAEs, the transformations performed in flow-based models must be invertible, and the determinant of its Jacobian should be computed efficiently [44]. Several widely adopted architectures include planar flow [57], coupling flow [19,20], autoregressive flow [42,53],  $1 \times 1$  convolution [41], and spline flow [24,25]. Residual networks use residual connections to build a reversible network, such as RevNets [28], iRevNets [39], and iResNet [4].

The idea of residual connections can be extended to continuous-time or infinitesimal flow models. One type of continuous-time flow models is formulated by the theory of ordinary differential equations (ODEs) [14,23,31], among which [74] proposes a continuous-

time gradient flow model from the perspective of optimal transport and fluid dynamics. The other type is based on diffusion processes and formulated by stochastic differential equations (SDEs) [13,66,69].

### ***Tensor-train representation***

The tensor-train representation originates in the density-matrix renormalization group (DMRG) [72] from physics and has proven useful in computational mathematics [17,18,30]. It has been successfully applied to high-dimensional scientific computing problems [1,22,45], quantum chemistry and molecular physics [2,12], and signal and image processing [16,71]. Numerous methods have been proposed for constructing the low-rank tensor-train representation of high-dimensional functions in scenarios where the function can be evaluated at arbitrary points or with a limited number of evaluations, such as TT-cross [51] and DMRG-cross [60], TT completion [64], and STTA [46].

In addition to its applications in function approximation, tensor-train representations are widely used for approximating high-dimensional probability densities, closely related to probabilistic graphical models. [59] proves a duality between the tensor networks and graphical models, and most notably, the hidden Markov model (HMM) can be considered as a special version of the tensor-train representation [9]. This restricted version of the tensor-train representation with all matrices only containing non-negative entries has been explored by [68].

The tensor-train representation has also been recently applied to generative modeling, where one constructs the tensor-train model directly from samples without access to the values of the density function. Several early attempts are based on the optimization-based DMRG scheme [10,32] and Riemannian optimization [50]. In the work by [38], an optimization-less method is proposed for constructing the tensor-train representation directly from samples using a sketching technique. Other works have also taken advantage of other structures of tensor networks, including the tree tensor network [15,67], and the projected entangled-pair state (PEPS) [70].

### ***Variational inference with tensorizing flow***

Density estimation using maximum likelihood estimation is closely connected to the variational inference (VI) problem. VI seeks to approximate an unnormalized density with a low-complexity ansatz by optimizing over variational parameters [7]. Early approaches to VI include mean-field VI, coordinate ascend VI [6], stochastic VI [36], and black box VI [56]. Deep neural networks have recently been actively applied in this field [48,49].

A concurrent work [40] proposes the combination of a tensor-based distribution and a neural network for variational inference problems, where an unnormalized analytic form of the density is given for constructing the approximate tensor-train representation. In contrast, our current density estimation task involves constructing the approximate density solely based on limited given samples, requiring different techniques to deal with the challenges.

## **1.3 Organization**

The paper is organized as follows. In Sect. 2, we provide an introduction to the necessary background and preliminary concepts. Our proposed method is detailed in Sect. 3.

In Sect. 4, we demonstrate the advantages of our proposed method through numerical experiments. Finally, we conclude in Sect. 5 with a discussion of our method.

## 2 Problem and background

In this section, we introduce the problem setting and the commonly used notations in Sect. 2.1, the tensor-train representation for both tensors and general functions in Sect. 2.2, and the continuous-time flow model in Sect. 2.3.

### 2.1 Problem setting and notations

We consider the problem of approximating an unknown probability density function  $p^*(\mathbf{x})$  on  $\mathbb{R}^d$ , where  $\mathbf{x} = (x_1, \dots, x_d)$  and  $x_i$ 's are individual coordinates. Suppose we are given  $N$  independent  $d$ -dimensional samples  $\mathbf{x}^{(i)} = (x_1^{(i)}, \dots, x_d^{(i)})_{1 \leq i \leq N}$  drawn from  $p^*(\mathbf{x})$ , the goal is to construct another probability density function  $p_\theta(\mathbf{x})$  with parameter  $\theta$  that can approximate  $p^*(\mathbf{x})$ . The approximation  $p_\theta(\mathbf{x})$  is required to be normalized and easy to sample.

Let  $p^E(\mathbf{x})$  be the empirical distribution of the samples, i.e.,

$$p^E(\mathbf{x}) = \frac{1}{N} \sum_{i=1}^N \delta(\mathbf{x} - \mathbf{x}^{(i)}). \quad (2.1)$$

This task is typically formulated using maximum likelihood estimation, where the parameter  $\theta$  is obtained by

$$\begin{aligned} \theta &= \operatorname{argmin}_\theta \operatorname{D}_{\text{KL}}(p^*(\cdot) \| p_\theta(\cdot)) = \operatorname{argmin}_\theta \mathbb{E}_{\mathbf{x} \sim p^*} [-\log p_\theta(\mathbf{x})] \\ &\approx \operatorname{argmin}_\theta \mathbb{E}_{\mathbf{x} \sim p^E} [-\log p_\theta(\mathbf{x})]. \end{aligned} \quad (2.2)$$

In the following, we use MATLAB notation to simplify the notation. For example, we use  $m:n$  to represent  $m, \dots, n$ . For a 3-tensor  $\mathbf{A}$ ,  $\mathbf{A}(:, i, :)$  denotes the  $i$ -th slice of the 3-dimensional tensor  $\mathbf{A}$  along its second dimension. We also denote  $1, \dots, n$  as  $[n]$ , variables  $x_m, \dots, x_n$  as  $x_{m:n}$ , and the corresponding infinitesimal volume  $dx_m \cdots dx_n$  as  $dx_{m:n}$ . The marginal distribution of variables  $x_{m:n}$  for a distribution  $p(\mathbf{x})$  is denoted as  $p(x_{m:n})$ . In particular, the marginal distributions of variables  $x_{1:2}, x_{1:3}, \dots, x_{d-2:d}, x_{d-1:d}$  are denoted as

$$p_1(x_{1:2}), p_2(x_{1:3}), \dots, p_{d-1}(x_{d-2:d}), p_d(x_{d-1:d}), \quad (2.3)$$

among which  $p_1$  and  $p_d$  are 2-marginals and the rest are 3-marginals.

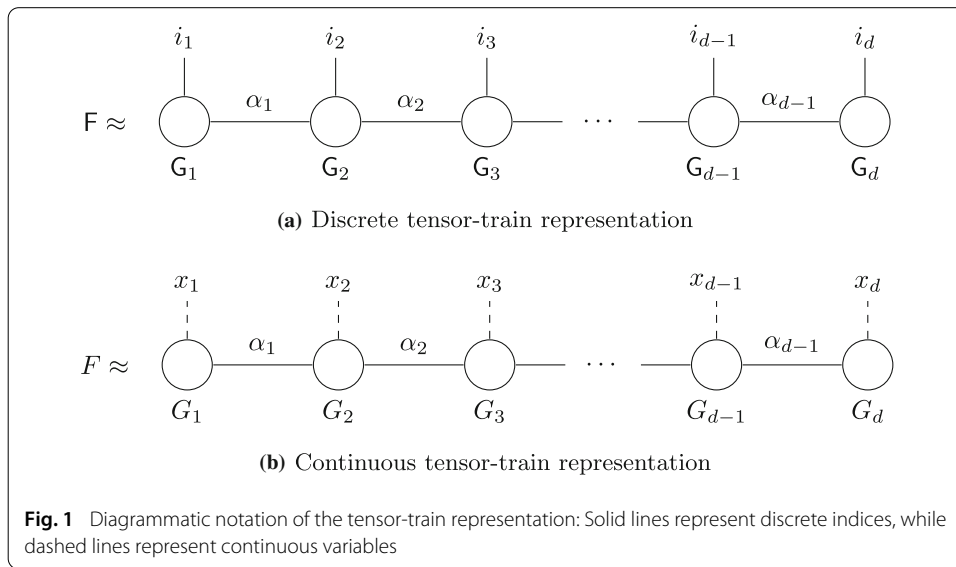
For simplicity, we assume  $p(\mathbf{x})$  is sufficiently smooth and  $\operatorname{supp}(p) \subset I^d$  for an interval  $I \subset \mathbb{R}$ . Throughout Sects. 2 and 3, we further assume  $I = [-1, 1]$ , while general cases where  $I = [a, b]$  or  $I = \mathbb{R}$  can be handled in a similar manner through appropriate translation and rescaling.

### 2.2 Tensor-train representation

Data are often represented as tensors in modern machine learning and scientific computing. A  $d$ -dimensional tensor  $\mathbf{F}$  in  $\mathbb{R}^{n_1 \times \cdots \times n_d}$  is a collection of numbers denoted by  $F(i_1, \dots, i_d)$  with  $1 \leq i_1, \dots, i_d \leq n$ . With  $n^d$  elements, the computational cost increases exponentially as the dimension  $d$  grows.

One way to represent or approximate high-dimensional tensors is to use the *tensor-train (TT) representation*, i.e.,

$$F(i_1, \dots, i_d) \approx G_1(i_1, :) G_2(:, i_2, :) \cdots G_d(:, i_d), \quad (2.4)$$



where  $G_1 \in \mathbb{R}^{n \times r_1}$ ,  $G_2 \in \mathbb{R}^{r_1 \times n \times r_2}$ ,  $\dots$ ,  $G_d \in \mathbb{R}^{r_{d-1} \times n}$  are the *cores*, and  $r_i$  for  $1 \leq i \leq d-1$  are the *ranks* of the TT representation.

The tensor  $F$  is then represented by the product of a sequence of corresponding slices of the cores, which is often described in the *diagrammatic notation* as shown in Fig. 1. We refer the readers to the discussions by [54] for interpreting this kind of notation. When the ranks  $\{r_i\}_{1 \leq i \leq d-1}$  are bounded, TT format features linear cost in  $n$  and  $d$ .

The idea of tensor-train can also be generalized to obtain the low-rank approximation of high-dimensional functions. The TT representation of a general  $d$ -dimensional function  $F(\mathbf{x}) : I^d \rightarrow \mathbb{R}$  consists of a sequence of  $d$  functions  $G_1 : I \times [r_1] \rightarrow \mathbb{R}$ ,  $G_2 : [r_1] \times I \times [r_2] \rightarrow \mathbb{R}$ ,  $\dots$ ,  $G_d : [r_{d-1}] \times I \rightarrow \mathbb{R}$ , as

$$F(x_{1:d}) \approx \sum_{\alpha_1=1}^{r_1} \sum_{\alpha_2=1}^{r_2} \cdots \sum_{\alpha_{d-1}=1}^{r_{d-1}} G_1(x_1, \alpha_1) G_2(\alpha_1, x_2, \alpha_2) \cdots G_d(\alpha_{d-1}, x_d), \quad (2.5)$$

or more compactly

$$F(x_{1:d}) \approx G_1(x_1, :) G_2(:, x_2, :) \cdots G_d(:, x_d).$$

The diagrammatic notation of this continuous tensor-train is shown in Fig. 1.

### 2.3 Continuous-time flow model

Flow-based generative models typically aim to design a pushforward  $f : \mathbb{R}^d \rightarrow \mathbb{R}^d$  that maps a latent, easy-to-sample probability density  $q_0(\mathbf{x})$  to a challenging target probability density  $q_1(\mathbf{x})$  that satisfies

$$q_1(\mathbf{x}) = q_0(f^{-1}(\mathbf{x})) \left| \det \left( \frac{\partial f^{-1}}{\partial \mathbf{x}} \right) \right|. \quad (2.6)$$

A *continuous-time flow model* is based on the perspective that regards  $f$  as the result of a flow that pushes the density  $q(\mathbf{x}, t)$ , initialized as  $q(\mathbf{x}, 0) = q_0(\mathbf{x})$ , over time  $t$  while conserving total probability mass. The evolution of the density  $q(\mathbf{x}, t)$  is characterized by the *continuity equation* from fluid mechanics:

$$\frac{\partial q(\mathbf{x}, t)}{\partial t} + \nabla \cdot [q(\mathbf{x}, t) \mathbf{v}(\mathbf{x})] = 0, \quad (2.7)$$

where  $\mathbf{v}(\mathbf{x})$  is the velocity field of the flow. Motivated by the linearized optimal transport, [74] assumes that the flow is irrotational, so  $\nabla \times \mathbf{v}(\mathbf{x}) = 0$ . Consequently,  $\mathbf{v}(\mathbf{x})$  can be expressed as the gradient of a potential function  $\phi(\mathbf{x})$ , i.e.,  $\mathbf{v}(\mathbf{x}) = \nabla \phi(\mathbf{x})$  [3].

An alternative description concerns the trajectory  $\mathbf{x}(t)$  that follows the velocity field  $\nabla \phi(\mathbf{x})$ , along which the following two ODEs hold:

$$\frac{d\mathbf{x}(t)}{dt} = \nabla \phi(\mathbf{x}(t)), \quad (2.8a)$$

$$\frac{dq(\mathbf{x}(t), t)}{dt} = -q(\mathbf{x}(t), t) \nabla^2 \phi(\mathbf{x}(t)) \quad (2.8b)$$

where the second equation directly follows from the continuity Eq. (2.7) and the formula of total derivative  $d/dt = \partial/\partial t + d\mathbf{x}(t)/dt \cdot \nabla$  [3]. This formulation provides a more intuitive way to understand the forward map  $f$  as the map from  $\mathbf{x}(0)$  to  $\mathbf{x}(T)$  and the inverse map  $f^{-1}$  as the map from  $\mathbf{x}(T)$  to  $\mathbf{x}(0)$ . During the evaluation of  $q(\mathbf{y}, T)$  for an arbitrary  $\mathbf{y}$ , we first compute the inverse map  $f^{-1}(\mathbf{y})$  by solving (2.8a) from  $t = T$  to 0 with  $\mathbf{x}(T) = \mathbf{y}$ , and then solve (2.8b) from  $t = 0$  to  $T$  with  $q(\mathbf{x}(0), 0) = p_0(f^{-1}(\mathbf{y}))$ . During sampling, we first draw a sample  $\mathbf{z}$  from the initial distribution  $p_0$  and then output  $f(\mathbf{z})$  by solving (2.8a) from  $t = 0$  to  $T$  with  $\mathbf{x}(0) = \mathbf{z}$ .

For a predetermined time horizon  $T$ , the flow determined by different potential functions  $\phi(\mathbf{x})$  may evolve the initial density  $q(\mathbf{x}, 0) = q_0(\mathbf{x})$  into a variety of densities  $q(\mathbf{x}, T)$  at time  $T$ . From the perspective of optimal control theory, the optimal potential function  $\phi(\mathbf{x})$  for approximating the target density  $q_1(\mathbf{x})$  should be the solution to the following optimization problem:

$$\min_{\phi: \mathbb{R}^d \rightarrow \mathbb{R}} D(q_1(\cdot), q(\cdot, T)), \quad (2.9)$$

where  $D(\cdot, \cdot)$  is a proper metric or divergence for probability measures.

The potential function  $\phi(\mathbf{x})$  is represented by a neural network denoted as  $\phi_\theta(\mathbf{x})$  with parameters  $\theta$ . In what follows, the resulting pushforward  $f$  and density  $q(\mathbf{x}, T)$  obtained from this continuous-time flow model are denoted as  $f_\theta$  and  $q_\theta(\mathbf{x})$ . Using the Kullback–Leibler (KL) divergence as the metric  $D(\cdot, \cdot)$ , (2.9) reduces to an MLE problem as in (2.2). Thus, the parameter  $\theta$  of the neural network is trained by minimizing the negative log-likelihood

$$\theta = \operatorname{argmin}_\theta \mathbb{E}_{\mathbf{x} \sim q_1} \left[ \log \frac{q_1(\mathbf{x})}{q_\theta(\mathbf{x})} \right] = \operatorname{argmin}_\theta \mathbb{E}_{\mathbf{x} \sim q_1} [-\log q_\theta(\mathbf{x})].$$

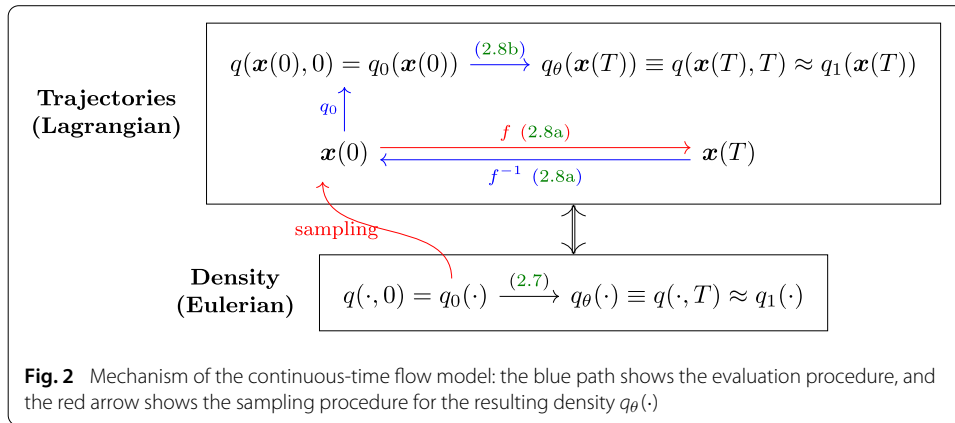
The mechanism of this continuous-time flow model is shown in Fig. 2.

### 3 Tensorizing flow

This section presents our tensorizing flow algorithm for high-dimensional density estimation. While the tensor-train representation is accurate only for a limited family of probability densities, it offers a more precise initial estimate than the normal distribution used in most flow-based models. As a result, it is reasonable to construct a flow-based model that maps this tensor-train density to the desired target distribution. This hybrid approach can potentially address well-known problems of normalizing flows, including mode collapse, limited expressivity, and difficulty in inverting a flow model with a high condition number.

As an overview, our algorithm consists of two main stages:

$$p^E(\cdot) = \frac{1}{N} \sum_{i=1}^N \delta(\cdot - \mathbf{x}^{(i)}) \xrightarrow{1} p^\Pi(\cdot) \xrightarrow{2} p^{\text{TF}}(\cdot)$$



1. Construct an approximate tensor-train representation  $p^\text{TT}$  from the samples  $\{\mathbf{x}^{(i)}\}_{1 \leq i \leq N}$  by combining sketching techniques with kernel density estimation (Sect. 3.1);
2. Apply the continuous-time flow model to drive  $p^\text{TT}$  toward  $\{\mathbf{x}^{(i)}\}_{1 \leq i \leq N}$ , resulting in the distribution  $p^\text{TF}$  (Sect. 3.2).

### 3.1 Stage 1: Construction of $p^\text{TT}$

We start with a conceptual algorithm and follow with a more practical one.

#### 3.1.1 Ideal case

Let us motivate the construction of an approximate TT representation by considering an ideal case where the underlying density  $p$  has a *finite-rank* structure and is also *Markovian*, which are explained as follows.

We assume that each reshaped version  $p(x_{1:k}; x_{k+1:d})$  of  $p(\mathbf{x})$  for  $1 \leq k \leq d-1$  is a Hilbert–Schmidt kernel [63] so that we can apply singular value decomposition (SVD) [73] (also called Schmidt decomposition) to them. For a Hilbert–Schmidt kernel  $K$ , we define its *column space* by its range and its *row space* by the range of its adjoint.

**Definition 3.1** (Finite-rank) A probability density function  $p(\mathbf{x})$  is *finite-rank* if for any  $1 \leq k \leq d-1$ , the reshaped version  $p(x_{1:k}; x_{k+1:d})$  of  $p(\mathbf{x})$  as a Hilbert–Schmidt kernel is finite rank, i.e., of finite-dimensional column space.

We also assume throughout that all the marginal distributions of  $p(\mathbf{x})$ , especially the 2- or 3-marginals  $p_k$  (2.3), belong to the class of Hilbert–Schmidt kernels so we can also perform SVD on them when necessary.

**Definition 3.2** (Markovian) A probability density function  $p(\mathbf{x})$  is *Markovian* if it can be written as

$$p(\mathbf{x}) = p(x_1)p(x_2|x_1) \cdots p(x_d|x_{d-1}).$$

### Finite-rank structure

Under the finite-rank assumption, the cores of the TT representation of  $p(\mathbf{x})$  can be simply obtained using the following proposition:

**Proposition 3.3** (Core determining equation) *Suppose that the probability density  $p$  is finite-rank. For  $1 \leq k \leq d-1$ , denote the rank of its reshaped version  $p(x_{1:k}; x_{k+1:d})$  by  $r_k$  and let  $\{\Phi_k(x_{1:k}; \alpha_k)\}_{1 \leq \alpha_k \leq r_k}$  be the first  $r_k$  left singular vectors of  $p(x_{1:k}; x_{k+1:d})$ . Then, there exists a unique solution  $G_1 : I \times [r_1] \rightarrow \mathbb{R}$ ,  $G_2 : [r_1] \times I \times [r_2] \rightarrow \mathbb{R}$ ,  $\dots$ ,  $G_d : [r_{d-1}] \times I \rightarrow \mathbb{R}$  to the following system of core determining equations (CDEs):*

$$\begin{aligned} G_1(x_1; \alpha_1) &= \Phi_1(x_1; \alpha_1), \\ \sum_{\alpha_{k-1}=1}^{r_{k-1}} \Phi_{k-1}(x_{1:k-1}; \alpha_{k-1}) G_k(\alpha_{k-1}; x_k, \alpha_k) &= \Phi_k(x_{1:k-1}; x_k, \alpha_k), \quad 2 \leq k \leq d-1, \\ \sum_{\alpha_{d-1}=1}^{r_{d-1}} \Phi_{d-1}(x_{1:d-1}; \alpha_{d-1}) G_d(\alpha_{d-1}; x_d) &= p(x_{1:d-1}; x_d), \end{aligned} \quad (3.1)$$

where the cores  $G_k$  give an exact TT representation of  $p(\mathbf{x})$ :

$$p(\mathbf{x}) = G_1(x_1, :) G_2(:, x_2, :) \cdots G_d(:, x_d). \quad (3.2)$$

We refer to Appendix 6 for the proof of this proposition.

### Left-sketching technique

Unfortunately, the size of equations in (3.1) grows exponentially with the dimension  $d$ , even after discretization. As a result, it is impossible to estimate all the coefficients  $\Phi_k$  from finite samples. However, a key observation is that this linear system is significantly over-determined, allowing for efficient reduction by a sketching technique.

In order to implement the sketching, we select suitable left-sketching functions  $S_{k-1}(y_{k-1}; x_{1:k-1})$  for  $2 \leq k \leq d$ , where  $y_{k-1} \in \mathcal{Y}_{k-1}$  with  $\mathcal{Y}_{k-1}$  being an appropriate set specified by the model. By contracting them with the left-hand sides of (3.1), we obtain the following reduced system of CDEs:

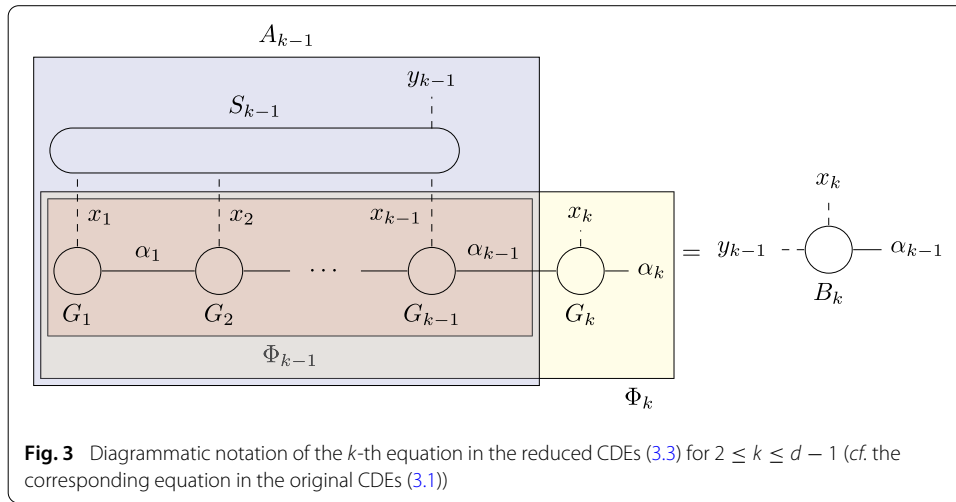
$$\begin{aligned} G_1(x_1; \alpha_1) &= B_1(x_1; \alpha_1), \\ \sum_{\alpha_{k-1}=1}^{r_{k-1}} A_{k-1}(y_{k-1}; \alpha_{k-1}) G_k(\alpha_{k-1}; x_k, \alpha_k) &= B_k(y_{k-1}; x_k, \alpha_k), \quad 2 \leq k \leq d-1, \\ \sum_{\alpha_{d-1}=1}^{r_{d-1}} A_{d-1}(y_{d-1}; \alpha_{d-1}) G_d(\alpha_{d-1}; x_d) &= B_d(y_{d-1}; x_d), \end{aligned} \quad (3.3)$$

where the coefficients  $B_k$  and  $A_k$  are given by

$$\begin{aligned} B_1(x_1; \alpha_1) &= \Phi_1(x_1; \alpha_1), \\ B_k(y_{k-1}; x_k, \alpha_k) &= \int_{I^{k-1}} S_{k-1}(y_{k-1}; x_{1:k-1}) \Phi_k(x_{1:k-1}; x_k, \alpha_k) dx_{1:k-1}, \quad 2 \leq k \leq d-1, \\ B_d(y_{d-1}; x_d) &= \int_{I^{d-1}} S_{d-1}(y_{d-1}; x_{1:d-1}) p(x_{1:d-1}; x_d) dx_{1:d-1}, \\ A_{k-1}(y_{k-1}; \alpha_{k-1}) &= \int_{I^{k-1}} S_{k-1}(y_{k-1}; x_{1:k-1}) \Phi_{k-1}(x_{1:k-1}; \alpha_{k-1}) dx_{1:k-1}, \quad 2 \leq k \leq d. \end{aligned} \quad (3.4)$$

Generally, the left-sketching functions  $S_{k-1}(y_{k-1}; x_{1:k-1})$  need to be chosen such that the row space of  $\Phi_k(x_{1:k-1}; x_k, \alpha_k)$  is retained, and the variance of the coefficient matrices





is reduced as much as possible [38]. For an illustration of the sketching technique, we refer readers to Fig. 3 for the diagrammatic notation of the  $k$ -th equation in the reduced CDEs (3.3) for  $2 \leq k \leq d-1$  (cf. the corresponding equation in the original CDEs (3.1)).

#### Markovian structure

In general, it is unclear which  $S_k$  to choose to get  $B_k$  and  $A_k$  in (3.4), and it is impossible to compute or estimate the singular vectors  $\Phi_k$  in practice. However, under the extra Markovian assumption, the computation of  $B_k$  and  $A_k$  can be simplified due to the following lemma:

**Lemma 3.4** ([38, Lemma 5]) Suppose  $p(\mathbf{x})$  is Markovian, then for any  $i \leq k \leq j-1$ ,

1.  $p(x_{i:k}; x_{k+1:j})$  and  $p(x_{i:k}; x_{k+1})$  have the same column space;
2.  $p(x_{i:k}; x_{k+1:j})$  and  $p(x_k; x_{k+1:j})$  have the same row space.

Lemma 3.4 essentially tells us that for  $p(x_{1:k}; x_{k+1:d})$ , marginalizing out  $x_{k+2:d}$  or  $x_{1:k-1}$  does not affect the corresponding column or row space. Motivated by this lemma, we can make the following two simplifications while computing the coefficients  $B_k$  and  $A_k$  in (3.1):

1. Obtain  $\Phi_k(x_{1:k}; \alpha_k)$  for  $1 \leq k \leq d-1$  by only considering the column space of the  $(k+1)$ -dimensional marginal distribution  $p(x_{1:k}; x_{k+1})$  instead of the full distribution  $p(x_{1:k}; x_{k+1:d})$ ;
2. Simply take  $\mathcal{Y}_{k-1} = I$  and  $S_{k-1}(y_{k-1}; x_{1:k-1}) = \delta(y_{k-1} - x_{k-1})$ , i.e., the Schwartz distribution that marginalizes out the first  $k-2$  dimensions, for  $2 \leq k \leq d$  as suggested by [38].

For  $k=1$ , these simplifications indicate that  $\Phi_1(x_1; \alpha_1)$  can be obtained directly by applying SVD to the 2-marginal  $p_1(x_1; x_2)$ , and subsequently

$$A_1(y_1; \alpha_1) = \int_I \delta(y_1 - x_1) \Phi_1(x_1; \alpha_1) dx_1 = \Phi_1(y_1; \alpha_1) = B_1(y_1; \alpha_1),$$

where the last equality is by definition (3.4). Similarly, for  $k=d$ ,  $B_d(y_{d-1}; x_d) = p_d(y_{d-1}; x_d)$ .

For  $2 \leq k \leq d-1$ , the simplifications yield

$$\begin{aligned} B_k(y_{k-1}; x_k, \alpha_k) &= \int_{I^{k-1}} \delta(y_{k-1} - x_{k-1}) \Phi_k(x_{1:k-1}; x_k, \alpha_k) dx_{1:k-1} \\ &= \int_{I^{k-2}} \Phi_k(x_{1:k-2}; y_{k-1}, x_k, \alpha_k) dx_{1:k-2}. \end{aligned} \quad (3.5)$$

A natural way to build  $B_k$  is to first calculate  $\Phi_k(x_{1:k}; \alpha_k)$  by performing SVD directly on  $p(x_{1:k}; x_{k+1})$ , and then apply left-sketching by marginalizing out  $x_{1:k-2}$  from  $\Phi_k(x_{1:k}; \alpha_k)$  as in (3.5). However, this approach is not practical because  $p(x_{1:k}; x_{k+1})$  grows exponentially with  $d$ , and its range  $\Phi_k(x_{1:k}; \alpha_k)$  cannot be estimated accurately from a limited collection of samples. Thus, instead of calculating  $\Phi_k$  by SVD and obtaining  $B_k$  through (3.5) directly, we propose the following method to obtain an approximate  $B_k$  implicitly: we first apply the left-sketching functions  $S_{k-1}$  to  $p(x_{1:k}; x_{k+1})$ , i.e., marginalize out  $x_{1:k-2}$  from  $p(x_{1:k}; x_{k+1})$  to obtain the 3-marginal  $p_k(x_{k-1}, x_k; x_{k+1})$ , and then perform SVD on  $p_k(x_{k-1}, x_k; x_{k+1})$ . Then,  $B_k(x_{k-1}, x_k; \alpha_k)$  is formed by the first  $r_k$  left singular vectors of  $p_k(x_{k-1}, x_k; x_{k+1})$ . Moreover, since

$$\begin{aligned} A_k(y_k; \alpha_k) &= \int_{I^k} \delta(y_k - x_k) \Phi_k(x_{1:k}; \alpha_k) dx_{1:k} = \int_{I^{k-1}} \Phi_k(x_{1:k-1}; y_k, \alpha_k) dx_{1:k-1} \\ &= \int_I \int_{I^{k-2}} \Phi_k(x_{1:k-2}; y_{k-1}, y_k, \alpha_k) dx_{1:k-2} dy_{k-1} = \int_I B_k(y_{k-1}; y_k, \alpha_k) dy_{k-1}, \end{aligned}$$

$A_k$  is obtained by marginalizing out the first dimension of  $B_k$ .

In conclusion, we have shown that for finite-rank and Markovian distributions  $p(\mathbf{x})$ , an exact TT representation in the form of (3.2) can be obtained by first forming the coefficients  $B_k$  and  $A_k$  and then solving the reduced system of CDEs (3.3).

### 3.1.2 General case

In Sect. 3.1.1, we make the assumption that the distribution  $p$  is finite-rank and Markovian. We also assume the access to the marginals  $p_k$  (2.3), and the singular value decomposition for Hilbert–Schmidt kernels (rather than finite-dimensional matrices). However, in practical applications, we often face unknown underlying distributions  $p^*$ , which may not necessarily be low-rank or Markovian. Moreover, instead of having analytic formulae for the marginals, we only have access to a limited set of samples  $\{\mathbf{x}^{(i)}\}_{1 \leq i \leq N}$  drawn from  $p^*$ .

To bridge this gap between the ideal case and the practical scenario, we adapt the method described in Sect. 3.1.1. The resulting TT  $p^\Pi$  reasonably approximates the unknown underlying distribution  $p^*$ .

#### Step 1

Construct kernel density estimators  $p_k^S$  of the marginals  $p_k^*$  (2.3) from samples  $\{\mathbf{x}^{(i)}\}_{1 \leq i \leq N}$  for  $1 \leq k \leq d$ .

When evaluating the coefficients  $B_k$  and  $A_k$ , one direct approach is to estimate the marginals  $p_k^*$  by interpolating the marginal distribution  $p_k^E$  of the empirical distribution  $p^E$  (2.1) with polynomials, as done in [38]. Instead, we estimate  $p_k^*$  by applying kernel density estimation (KDE) to the corresponding slices of samples. For example, for  $2 \leq k \leq d-1$ ,  $p_k^*$  is estimated by kernel density estimators

$$p_k^S(x_{k-1:k+1}) := \frac{1}{Nh} \sum_{i=1}^N K\left(\frac{x_{k-1:k+1} - x_{k-1:k+1}^{(i)}}{h}\right), \quad (3.6)$$

where  $K(\cdot)$  is the Gaussian kernel  $(2\pi)^{-3/2} \exp(-\|\cdot\|^2/2)$  and  $h$  is the bandwidth.

**Remark 3.5** The use of KDE on marginal distributions is a crucial step in constructing the TT representation. This is because performing SVD directly on the sparse marginal empirical distributions  $p_k^E$  may lead to severe Gibbs phenomenon, posing a major obstacle in implementing tensorizing flow. Therefore, we first smooth  $p_k^E$  using KDE and perform SVD on the resulting kernel density estimators  $p_k^S$  instead of  $p_k^E$ . It is important to note that we only use KDE to estimate the 2- or 3-marginals  $p_k^*$  but *not* the full distribution  $p^*$ , as applying KDE directly to  $p^*$  would result in poor performance due to the curse of dimensionality.

In general, choosing the bandwidth parameter  $h$  involves a bias-variance trade-off. As  $h \rightarrow 0$ , the kernel density estimator  $\hat{p}_k$  approaches the empirical distribution  $p_k^E$ , an unbiased estimator of the true distribution  $p_k$ . On the other hand, as  $h$  grows,  $\hat{p}_k$  becomes smoother with a certain bias. When the bandwidth  $h$  is sufficiently large,  $\hat{p}_k$  is smooth enough to be well-approximated by polynomials.

### Step 2

Estimate the coefficients  $B_k$  for  $1 \leq k \leq d$  and  $A_k$  for  $1 \leq k \leq d-1$  from kernel density estimators  $p_k^S$ .

Ideally,  $B_d = p_d^S$ , and for  $1 \leq k \leq d-1$ ,  $B_k$  is formed by the first  $r_k$  left singular vectors by performing SVD on the  $d-1$  kernel density estimators

$$p_1^S(x_1; x_2), p_2^S(x_1, x_2; x_3), \dots, p_{d-1}^S(x_{d-2}, x_{d-1}; x_d).$$

Afterward,  $A_1 = B_1$ , and  $A_k$  is obtained by marginalizing out the first variable of  $B_k$  for  $2 \leq k \leq d-1$ .

However, since  $x_i$  takes value in  $I = [-1, 1]$  and all marginals are continuous functions, a numerical approximation is necessary for SVD. To this end, we introduce the normalized Legendre polynomials  $\{L_i(x)\}_{i \geq 1}$  with  $\deg(L_i) = i-1$ , which form an orthonormal basis of  $L^2(I)$ . For example, when evaluating  $B_k$  for  $2 \leq k \leq d-1$ , we take tensor-product normalized Legendre polynomials  $\{L_{i_{k-1}}(x_{k-1})L_{i_k}(x_k)\}_{1 \leq i_{k-1}, i_k \leq M}$  as the expansion basis for variables  $(x_{k-1}, x_k)$  and  $\{L_{i_{k+1}}(x_{k+1})\}_{1 \leq i_{k+1} \leq M}$  for  $x_{k+1}$ . Here,  $M$  is a constant that controls the accuracy of the polynomial approximation. Projecting  $p_k^S(x_{k-1}, x_k; x_{k+1})$  orthogonally onto these two sets of basis functions gives the following  $M^2 \times M$  coefficient matrix with entry

$$P_k^S(i_{k-1}, i_k; i_{k+1}) = \int_{I \times I} \int_I (L_{i_{k-1}}(x_{k-1})L_{i_k}(x_k)) p_k^S(x_{k-1}, x_k; x_{k+1}) L_{i_{k+1}}(x_{k+1}) dx_{k-1} dx_k dx_{k+1}.$$

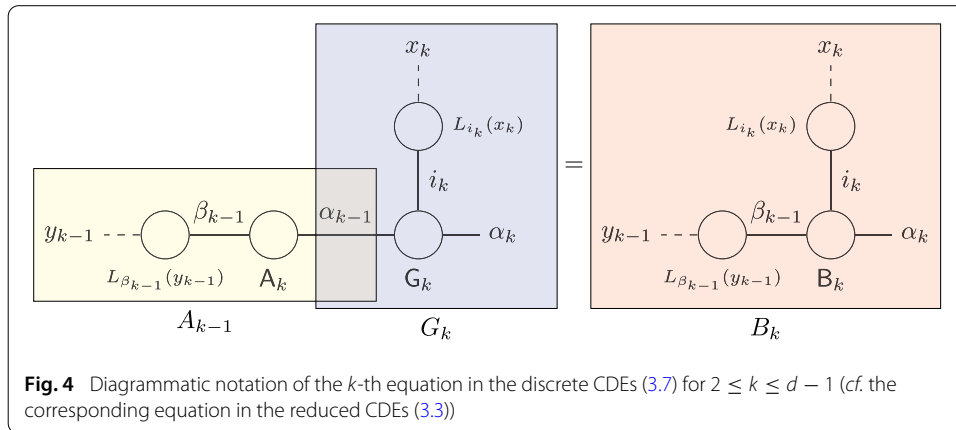
Next, we compute the truncated SVD for  $P_k^S(i_{k-1}, i_k; i_{k+1})$  and group the first  $r_k$  singular vectors into a matrix  $B_k(i_{k-1}, i_k; \alpha_k)$  of size  $M^2 \times r_k$ , where  $r_k$  is the numerical rank. Finally,  $B_k(x_{k-1}, x_k; \alpha_k)$  is obtained by

$$B_k(x_{k-1}, x_k; \alpha_k) := \sum_{i_{k-1}=1}^M \sum_{i_k=1}^M B_k(i_{k-1}, i_k; \alpha_k) L_{i_{k-1}}(x_{k-1}) L_{i_k}(x_k),$$

and by contracting  $L_1(x_{k-1}) \equiv 1/\sqrt{2}$  to both sides,  $A_k$  is obtained subsequently by

$$A_k(x_k; \alpha_k) := \sqrt{2} \sum_{i_k=1}^M B_k(1, i_k; \alpha_k) L_{i_k}(x_k).$$

The cases for  $k=1$  and  $d$  are handled similarly.



### Step 3

Solve (3.3) using least squares for the cores  $G_1, \dots, G_d$ .

Similar to the previous step, a numerical approximation is needed since (3.3) is formulated in terms of functions. We again resort to polynomial approximation and expand  $G_k$ ,  $B_k$ , and  $A_k$  w.r.t. the first  $M$  normalized Legendre polynomials. For example, for  $2 \leq k \leq d-1$ , the corresponding coefficient matrices  $G_k$ ,  $B_k$ , and  $A_k$  are given by

$$\begin{aligned} G_k(\alpha_{k-1}; i_k, \alpha_k) &= \int_I G_k(\alpha_{k-1}; x_k, \alpha_k) L_{i_k}(x_k) dx_k, \\ B_k(\beta_{k-1}; i_k, \alpha_k) &= \int_{I \times I} B_k(y_{k-1}; x_k, \alpha_k) L_{\beta_{k-1}}(y_{k-1}) L_{i_k}(x_k) dy_{k-1} dx_k \\ A_{k-1}(\beta_{k-1}; \alpha_{k-1}) &= \int_I A_{k-1}(y_{k-1}; \alpha_{k-1}) L_{\beta_{k-1}}(y_{k-1}) dy_{k-1}. \end{aligned}$$

As interpreted by diagrammatic notation in Fig. 4, the projected version of the system (3.3) is

$$\begin{aligned} G_1(i_1; \alpha_1) &= B_1(i_1; \alpha_1), \\ \sum_{\alpha_{k-1}=1}^{r_{k-1}} A_{k-1}(\beta_{k-1}; \alpha_{k-1}) G_k(\alpha_{k-1}; i_k, \alpha_k) &= B_k(\beta_{k-1}; i_k, \alpha_k), \quad 2 \leq k \leq d-1, \\ \sum_{\alpha_{d-1}=1}^{r_{d-1}} A_{d-1}(\beta_{d-1}; \alpha_{d-1}) G_d(\alpha_{d-1}; i_d) &= B_d(\beta_{d-1}; i_d). \end{aligned} \quad (3.7)$$

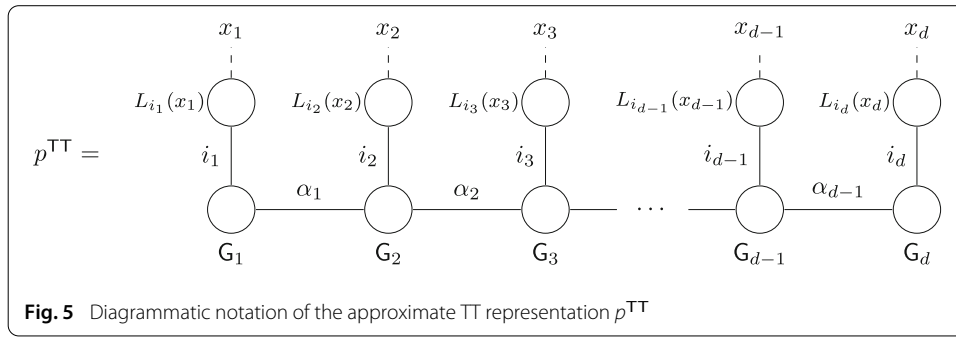
The discrete cores  $G_k$  can be efficiently obtained by solving these equations using least squares. Once  $G_k$ 's are solved, they are combined with the normalized Legendre polynomials to produce the continuous cores

$$G_k(\alpha_{k-1}; x_k, \alpha_k) \approx \sum_{i_k=1}^M G_k(\alpha_{k-1}; i_k, \alpha_k) L_{i_k}(x_k).$$

### Step 4

With the cores  $G_k$  ready, the approximate TT representation  $p^\Pi(\mathbf{x})$  of  $p^E(\mathbf{x})$  can be set to  $G_1(x_1, : ) G_2(x_2, : ) \cdots G_d(x_d, : )$  as in (3.2). However, there are two extra issues to be addressed.

First,  $G_1(x_1, : ) G_2(x_2, : ) \cdots G_d(x_d, : )$  does not necessarily integrate to unity. The normalization can be achieved by contracting  $G_1(x_1, : ) G_2(x_2, : ) \cdots G_d(x_d, : )$  with the all-one



function and absorbing the resulting constant into any of  $G_k$  so that  $p^{\text{TT}}$  retains the form

$$p^{\text{TT}}(x_{1:d}) := G_1(x_1, :) G_2(:, x_2, :) \cdots G_d(:, x_d), \quad (3.8)$$

the diagrammatic notation is shown in Fig. 5.

The second issue is that  $G_1(x_1, :) G_2(:, x_2, :) \cdots G_d(:, x_d)$  is not necessarily non-negative. To ensure the non-negativity, we may adopt the following post-processing by following the approach of *Born machine* [32], i.e., one solves

$$\begin{aligned} \min_{q(\mathbf{x})} & \|p^{\text{TT}}(\mathbf{x}) - r(\mathbf{x})^2\|_{L^2(I^d)} \\ \text{s.t. } r(\mathbf{x}) &= \sum_{i_1=1}^M \cdots \sum_{i_d=1}^M R(i_{1:d}) L_{i_1}(x_1) \cdots L_{i_d}(x_d), \end{aligned}$$

where  $R(i_{1:d}) = H_1(i_1, :) H_2(:, i_2, :) \cdots H_d(:, i_d)$  is a discrete tensor-train with discrete cores  $H_i$ . Noticing that

$$\int_{I^d} p^{\text{TT}}(\mathbf{x}) d\mathbf{x} = \int_{I^d} r(\mathbf{x})^2 d\mathbf{x} = \|R\|_F^2$$

by the orthogonality of Legendre polynomials, then

$$p^{\text{TT}}(\mathbf{x}) := r(\mathbf{x})^2 \quad (3.9)$$

is guaranteed to be non-negative and integrated into one by normalizing the Frobenius norm of the discrete tensor-train  $R$ . Strictly speaking, this is not a TT representation but the point-wise square of a TT representation.

**Remark 3.6** It is worth noting that the coefficients of the Legendre expansion decay super-algebraically for  $C^\infty$  functions [65, Theorem 9.1.1]. As a result, a moderately large value of  $M$  can lead to an accurate approximation. The numerical ranks  $r_k$  for a Markovian model are at most  $M$  and can be determined by truncating the singular values using an appropriate threshold. Empirically, these parameters can be selected by performing cross-validations on marginals. Additionally, as this tensor-train approximation is used as an initial guess in the next stage, these parameters can be kept quite small.

### 3.2 Stage 2: Construction of $p^{\text{TF}}$

In the second stage of our method, we depart from the typical approach of using the normal distribution as the base distribution. Instead, we begin with the approximate TT representation  $p^{\text{TT}}(\mathbf{x})$  obtained in (3.8) or (3.9) and use the continuous-time flow model in Sect. 2.3 to improve this approximation.

Following the steps outlined in Sect. 2.3, we initialize the distribution  $q(\mathbf{x}, 0)$  as  $p^{\text{TT}}(\mathbf{x})$ , and then choose appropriate values for the time horizon  $T$  and step-size  $\tau$ . This yields a new density approximation  $q_\theta(\mathbf{x}) \equiv q(\mathbf{x}, T)$ , where the subscript  $\theta$  denotes the neural network used to parameterize the potential function  $\phi_\theta(\mathbf{x})$  that guides the flow described in (2.8).

As in the MLE setup (2.2), the loss function for training is chosen as the negative log-likelihood:

$$\mathcal{L}(\theta) := -\mathbb{E}_{\mathbf{x} \sim p^\mathbb{E}} \log q_\theta(\mathbf{x}). \quad (3.10)$$

In the actual implementation, the neural network is trained on batches, which are randomly selected from the full sample set  $\{\mathbf{x}^{(i)}\}_{1 \leq i \leq N}$ . In each training step, the loss function is approximated by  $-1/N_{\text{batch}} \sum_{j=1}^{N_{\text{batch}}} \log q_\theta(\mathbf{x}^{(j)})$ , where  $N_{\text{batch}}$  is the batch size. Each likelihood  $q_\theta(\mathbf{x}^{(j)})$  is calculated by solving the dynamic system (2.8) by the fourth-order Runge–Kutta scheme.

Once  $q_\theta(\mathbf{x})$  is learned, we use it to define the final product

$$p^{\text{TF}}(\mathbf{x}) := q_\theta(\mathbf{x})$$

that approximate the unknown underlying distribution  $p^*(\mathbf{x})$ . To sample from  $p^{\text{TF}}(\mathbf{x}) = q_\theta(\mathbf{x})$ , we first sample from the approximate TT representation  $p^{\text{TT}}(\mathbf{x})^1$  and then applying the pushforward  $f$  again by numerically integrating (2.8a). Readers may refer to Fig. 2 for the evaluation and sampling procedures for  $q_\theta$ .

*Remark 3.7* In the case of normalizing flow, the base distribution is typically chosen as a normal distribution, which does not contain any information about the target distribution  $p^*$ . Consequently, the neural network in the flow model must be sufficiently large to learn the complicated pushforward map from the normal distribution to  $p^*$ . In contrast, our tensorizing flow approach uses a base distribution  $p^{\text{TT}}$  already close to  $p^*$ , so the pushforward is close to the identity map. As a result, the neural network in our flow model can be quite simple, making training easy and generalization robust. Furthermore, when we initialize the potential function  $\phi_\theta$  in the continuous-time flow model as a constant function, both the forward and inverse maps are initially the identity. This means we can exploit  $p^{\text{TT}}$  as prior knowledge, and the density approximation  $p^{\text{TF}}$  is guaranteed to be better than  $p^{\text{TT}}$  through training.

For this near-identity flow, one may also consider using residual flows [33]. Several related works [28, 39] introduce extra variables to create a reversible network architecture based on residual connections. However, compared with classical flows such as NICE [19], Real NVP [20], MAF [53], and Glow [41], these networks cannot be inverted analytically, which significantly affects their efficiency and feasibility. Moreover, due to the use of convolutional layers in these networks, evaluating the Jacobian determinant in (2.6) is expensive and often requires a biased, yet still costly, estimate of the log-Jacobian, given by the power series for the trace of the matrix logarithm  $\log(\det(I + F)) = \text{tr}(\log(I + F)) = \sum_{k=1}^{\infty} (-1)^{k-1} \text{tr}(F^k)/k$ . Since we use an ODE-based continuous-time flow model, we can obtain both the path and the log-Jacobian by numerical integrating (2.8), circumventing the inefficiency aforementioned.

Before ending the algorithmic discussion, we summarize the method in Algorithm 1.

<sup>1</sup>Efficient sampling from a given TT representation can be achieved using algorithms presented by [21] and [50].

**Algorithm 1** Tensorizing flow

**Require:** A collection of samples  $\{\mathbf{x}^{(i)}\}_{1 \leq i \leq N}$  independently drawn from an underlying distribution  $p^*(\mathbf{x}) : I^d \rightarrow \mathbb{R}$ ;

1. Construct the approximate TT representation  $p^{\text{TT}}(\mathbf{x})$  from the samples  $\{\mathbf{x}^{(i)}\}_{1 \leq i \leq N}$  following the routine outlined in Section 3.1.2.
2. Construct a potential function  $\phi_\theta(\mathbf{x})$  parameterized by the neural network  $\theta$ , set  $q(\mathbf{x}, 0) = p^{\text{TT}}(\mathbf{x})$ , and construct the density estimation  $q_\theta(\mathbf{x}) = q(\mathbf{x}, T)$  by applying Runge-Kutta scheme to (2.8) with step-size  $\tau$  for  $\lfloor T/\tau \rfloor$  steps;
3. Train the neural network on the sample set  $\{\mathbf{x}^{(i)}\}_{1 \leq i \leq N}$  w.r.t. loss function (3.10) and output  $q_\theta(\mathbf{x})$  as the final estimation  $p^{\text{TF}}(\mathbf{x})$  for  $p^*(\mathbf{x})$ .

*Remark 3.8* Here, we would like to emphasize the distinct nature of our methodology compared to [40]. The latter focuses on performing variational inference on a density of the form  $p(\mathbf{x}) \propto \exp(-U(\mathbf{x}))$ . The explicit expression of  $U(\mathbf{x})$  equips [40] with the ability to evaluate the unnormalized density  $\exp(-U(\mathbf{x}))$  at any selected point across a multivariate grid. Then, they utilize the TT-cross algorithm [51] to directly create the TT representation of the unnormalized density, which is the foundation for subsequent procedures. In contrast, our method caters to scenarios where the potential function is undisclosed and the density can only be assessed via samples. In these instances, we propose to build each core of the TT representation  $p^{\text{TT}}$  of the density with the marginal samples (cf. Sect. 3.1.2), followed by enhancing  $p^{\text{TT}}$  using the subsequent flow model.

#### 4 Experimental results

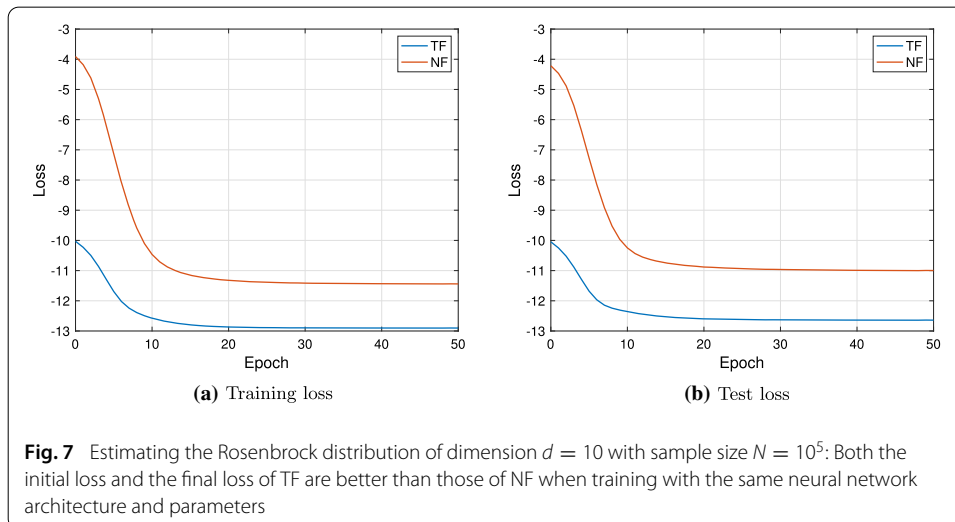
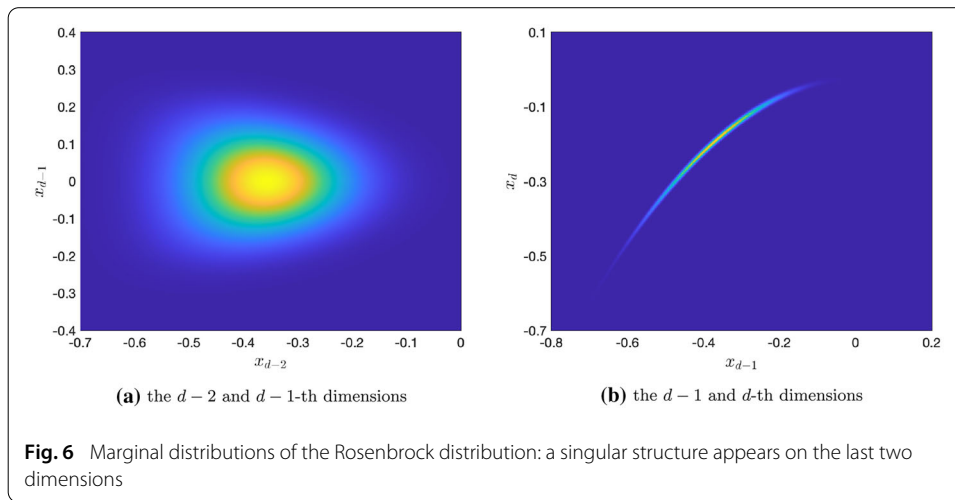
In this section, we present several experimental results that demonstrate the performance of our algorithm. Our algorithm is implemented by properly transforming and scaling the Legendre polynomials for an arbitrary interval  $I$  other than  $[-1, 1]$ , under the assumption that  $\text{supp}(p) \subset I^d$ . We will specify the choice of  $I$  for each example presented below. Gauss–Legendre quadrature is adopted for all numerical integration in constructing the TT representation with  $l$  quadrature points along each dimension.

We adopt a multi-layer perceptron (MLP) structure with an input layer, two hidden layers of width  $D$ , and an output layer, for the neural network used to parameterize the potential function  $\phi_\theta : \mathbb{R}^d \rightarrow \mathbb{R}$  in the flow model. To provide sufficient smoothness for  $\phi_\theta$  and  $q_\theta$ , the activation functions are chosen as log cosh and the soft-plus function for the first and second hidden layers, respectively. We use the Adam optimizer for the neural network training with two parameters: the learning rate (LR) and weight decay (WD). The learning rate is scaled by a multiplicative factor  $\gamma$  after each epoch. The choices of all parameters are organized in Table 1 in Appendix 7. All the experiments are implemented using PyTorch deep learning framework and conducted on a Tesla V100 GPU.

Throughout all the examples, we provide the experimental results of our tensorizing flow (TF) algorithm (Algorithm 1) in comparison with the normalizing flow (NF) approach, which also allows for explicit likelihood evaluation. To ensure a fair and direct comparison, we utilize the continuous-time flow model of the same neural network architecture and parameters (see Table 1) for the normalizing flow.

We would also like to emphasize that the loss (3.10) satisfies

$$\mathcal{L}(\theta) = -\mathbb{E}_{\mathbf{x} \sim p^E} \log q_\theta(\mathbf{x}) = \text{D}_{\text{KL}}(p^E(\cdot) \| q_\theta(\cdot)) - \mathbb{E}_{\mathbf{x} \sim p^E} \log p^E(\mathbf{x}).$$



Hence, it should *not* be expected for the loss to approach zero during the training process.

#### 4.1 Rosenbrock distribution

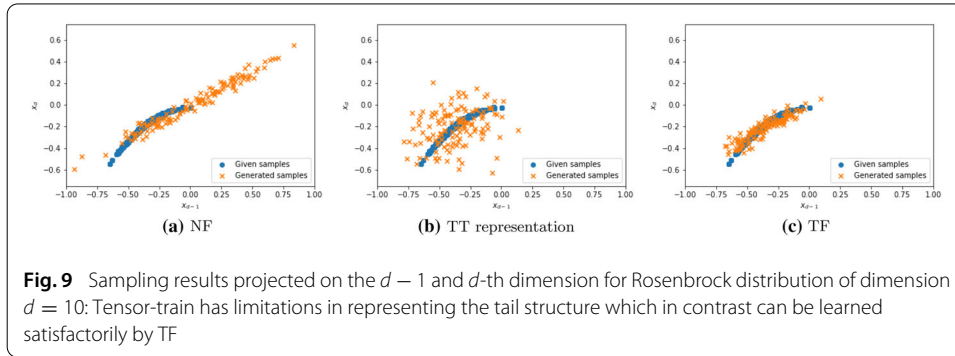
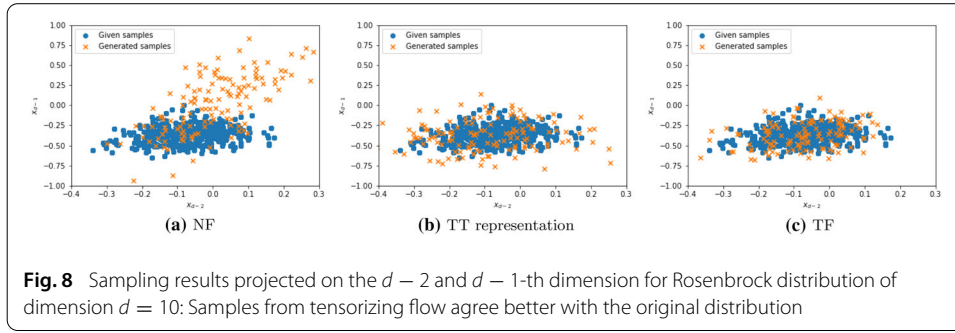
In this example, we consider the distribution induced by the Rosenbrock function  $v(\mathbf{x})$ , i.e.,  $p^*(\mathbf{x}) \propto \exp(-v(\mathbf{x})/2)$ , where

$$v(\mathbf{x}) = \sum_{i=1}^{d-1} \left[ c_i^2 x_i^2 + (c_{i+1} x_{i+1} + 5(c_i^2 x_i^2 + 1))^2 \right].$$

We set the dimension  $d = 10$  and restrict all  $x_i$  to the finite interval  $I = [-1, 1]$ . Following the example of [21], we select the scaling factor  $c_i = 2$  for  $1 \leq i \leq d-2$ ,  $c_{d-1} = 7$ , and  $c_d = 200$ . As shown in Fig. 6, the Rosenbrock distribution is relatively isotropic in the first  $d-2$  variables but is concentrated along a curve on the last two dimensions.

The experiment results are shown in Fig. 7. Our algorithm starts with a significantly lower loss when compared to normalizing flow. This observation aligns with our expectation that the approximate TT representation  $p^\Pi(\mathbf{x})$  serves as a much better base distribution  $q(\mathbf{x}, 0)$  than the normal distribution in the flow model, in terms of both initial and final losses.





Figures 8 and 9 display samples generated by our method and the given samples. It is clear that using the approximate TT representation, in contrast to a normal distribution as the base distribution, better captures the rough structure of the target distribution. In [21], extra fine grids on the last two dimensions are adopted (4 and 32 times finer than the first  $d - 2$  dimensions, respectively) to address the singular tail structure. However, since the empirical distribution is smoothed by kernel density estimation in our approach, no additional specific measures are required for constructing the approximate TT representation, and the subsequent neural network-based flow automatically corrects the estimation.

#### 4.2 Ginzburg–Landau distribution

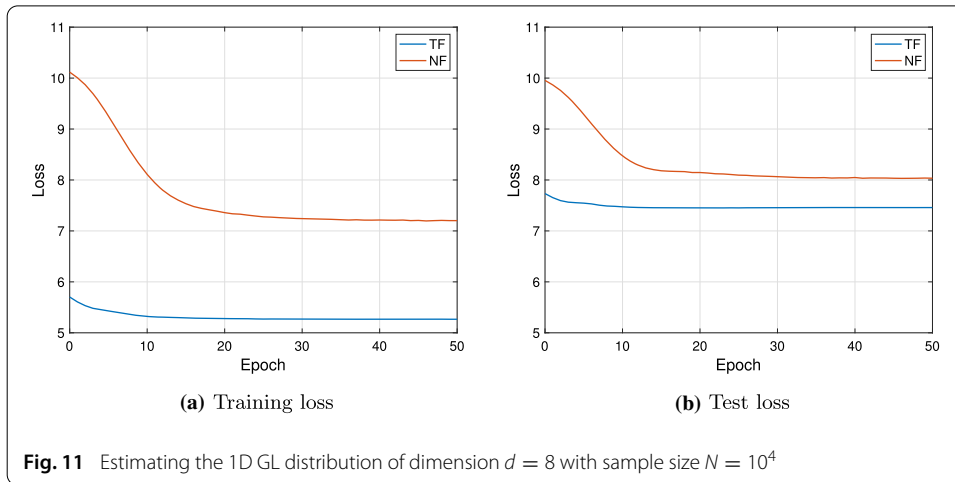
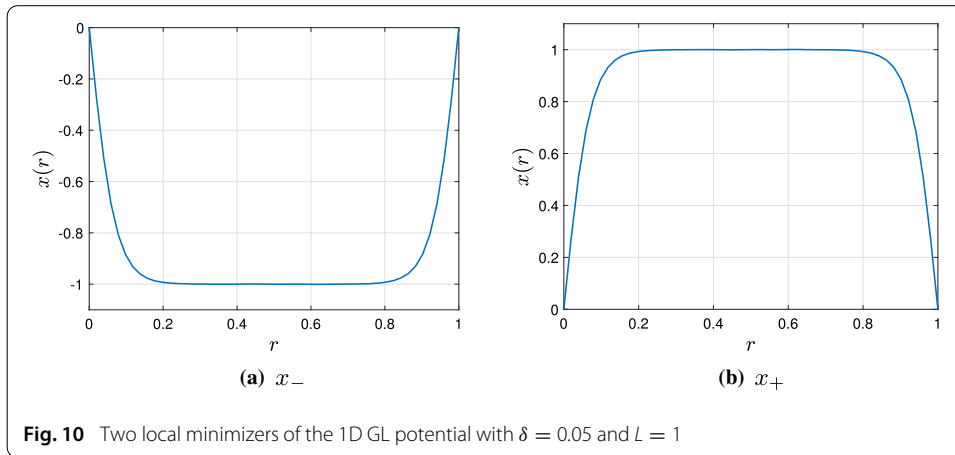
The Ginzburg–Landau (GL) theory is a widely used model for studying phase transition in statistical mechanics [37]. Let  $\Omega \subset \mathbb{R}^k$  be some domain with appropriate boundary conditions. The general Ginzburg–Landau potential, defined for a sufficiently smooth function  $x(\mathbf{r}) : \Omega \rightarrow \mathbb{R}$ , is given by

$$\mathcal{E}[x(\cdot)] = \int_{\Omega} \left[ \frac{\delta}{2} |\nabla_{\mathbf{r}} x(\mathbf{r})|^2 + \frac{1}{\delta} V(x(\mathbf{r})) \right] d\mathbf{r}, \quad (4.1)$$

where the potential  $V(x) = (1 - x^2)^2 / 4$ .

##### 4.2.1 1D Ginzburg–Landau distribution

We consider the 1-dimensional physical domain  $\Omega = [0, L]$ . The function  $x(\mathbf{r})$  is discretized using the vector  $\mathbf{x} \equiv (x_0, \dots, x_{d+1})$ , which contains the values of  $x(\mathbf{r})$  on the uniform grid  $(ih)_{i=0}^{d+1}$ , with Dirichlet boundary condition  $x_0 = x_{d+1} = 0$  and grid size  $h = L/(d + 1)$ . By employing the first-order finite difference scheme and estimating the integral in (4.1) with the Riemann sum, the 1D Ginzburg–Landau potential can be



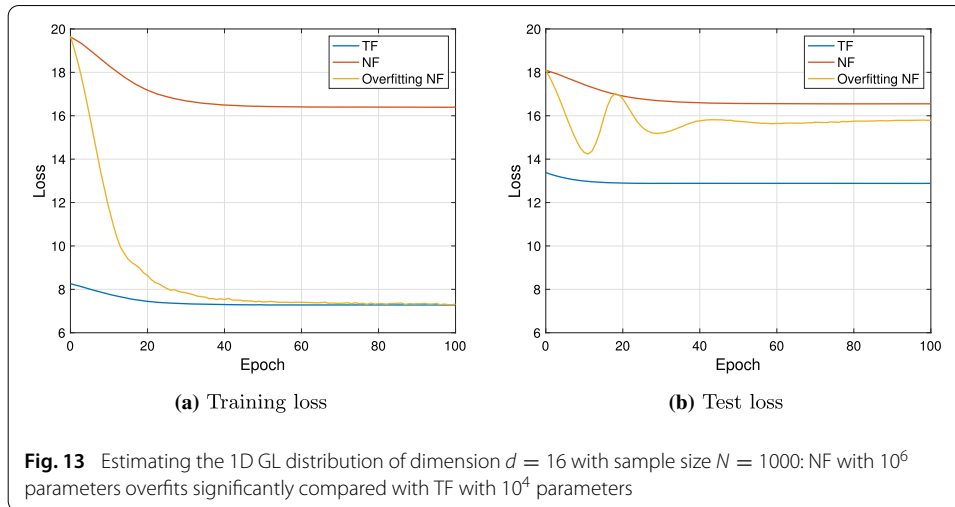
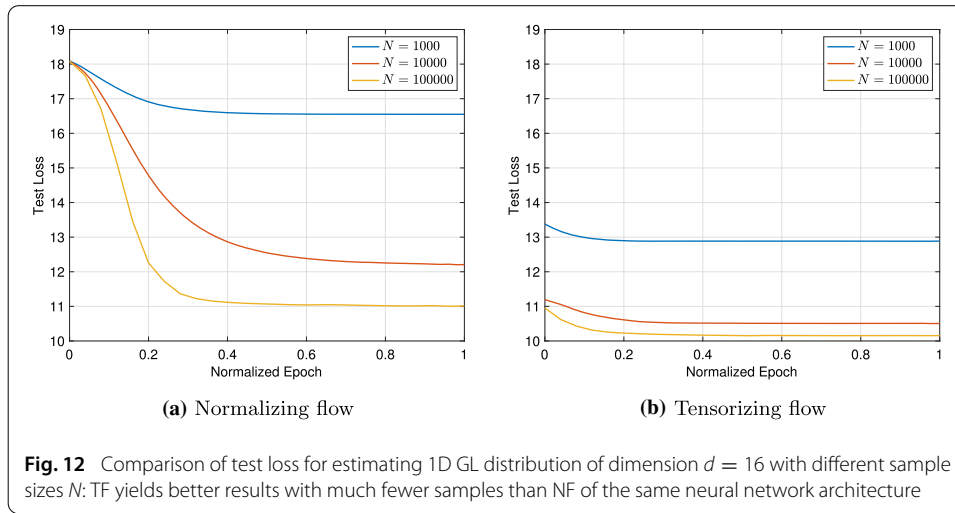
approximated by

$$E(\mathbf{x}) = \sum_{i=1}^{d+1} \left[ \frac{\delta}{2} \left( \frac{x_i - x_{i-1}}{h} \right)^2 + \frac{1}{4\delta} (1 - x_i^2)^2 \right] \quad (4.2)$$

and its associated Boltzmann distribution satisfies  $p^*(\mathbf{x}) \propto \exp(-\beta E(\mathbf{x}))$ , where  $\beta$  is the inverse temperature. As mentioned by [26], the majority of the states  $\mathbf{x}$  of interest lie within the range between  $\mathbf{x}_-$  and  $\mathbf{x}_+$ , which are the two minimizers of the 1D Ginzburg–Landau potential (4.2), shown in Fig. 10. Therefore, we choose  $I = [-3, 3]$  as the range for each  $x_i$  in the discretization  $\mathbf{x}$ .

The results for the case where  $d = 8$ ,  $\beta = 3$ ,  $\delta = 0.5$ , and  $h = 1$  are shown in Fig. 11. Similar to the previous example, the tensorizing flow algorithm achieves a lower initial loss and, eventually, a lower final loss than the normalizing flow algorithm.

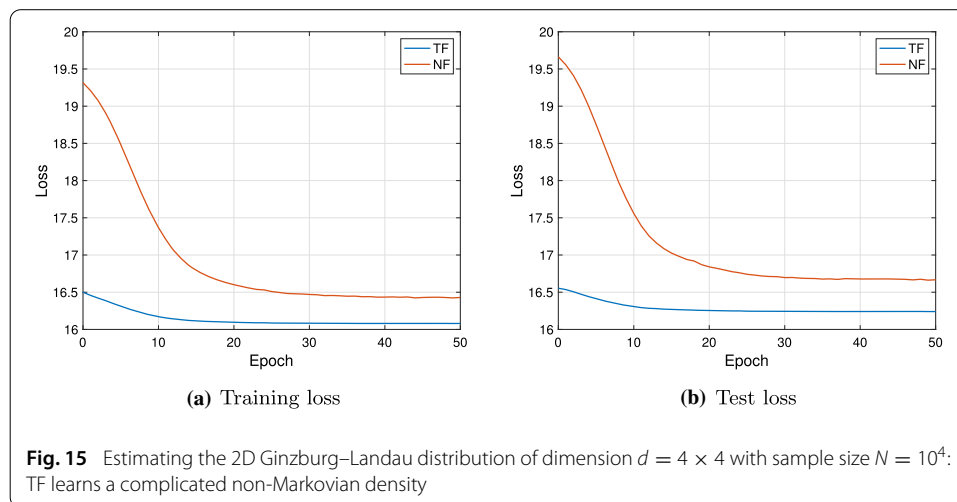
To demonstrate the impact of sample size on our approach, we perform additional experiments for the setting  $d = 16$ ,  $\beta = 3$ ,  $\delta = 1$ , and  $h = 1$  with sample sizes  $10^3$ ,  $10^4$ , and  $10^5$ . We adjust the training parameters, including the number of epochs, learning rate, etc., proportionally to ensure a fair comparison between these experiments (see Table 1). The results, presented in Fig. 12, depict the normalized epoch on the horizontal axis, i.e., epoch divided by the total number of epochs, and the test loss computed using a common test set of size  $5 \times 10^3$  on the vertical axis. As the performance of the normalizing flow improves



with an increase in sample size, a larger sample set yields a better TT representation  $p^{\text{TT}}$  to begin with, which ultimately results in a better density estimation  $p^{\text{TF}}$  after training. Furthermore, Fig. 12 reveals that our method produces a better density estimation with  $10^4$  samples than normalizing flow with  $10^5$  samples, demonstrating the efficiency of our method in terms of samples required to reach a certain accuracy in density estimation.

In Fig. 13, we present another example with  $d = 16$ ,  $\beta = 3$ ,  $\delta = 1$ ,  $h = 1$ , and  $N = 10^3$ , aiming to understand why normalizing flow cannot achieve the same test loss as tensorizing flow. Normalizing flow has higher training and test losses using the same architecture than tensorizing flow (see the red curves in Fig. 13). To improve the training loss of normalizing flow, we use a relatively over-parameterized neural network for normalizing flow (see the corresponding parameters in Table 1) so that its training loss matches that of tensorizing flow (see the yellow curve in Fig. 13). With a matching training loss, normalizing flow significantly overfits (see the yellow curve in Fig. 13). This demonstrates that tensorizing flow provides much better generalization and is not prone to overfitting, as it uses a relatively small and less expressive neural network.





parameterized by a neural network and fast calculation of both the forward and inverse map by the Runge–Kutta scheme.

Our experiments demonstrate that our method outperforms normalizing flow with similar architectures and can deal with distributions of certain singularities and non-Markovian models, for which traditional tensor-train methods may encounter difficulties. The near-identity nature of the tensorizing flow means that a relatively simple neural network is sufficient for the flow model, which is easier to train and less prone to overfitting than normalizing flow.

Our method makes several algorithmic choices. For example, although we use Legendre polynomials as the expansion basis in this work, our method is open to other expansion bases, such as the Chebyshev polynomials and Fourier bases. Furthermore, in the second stage of the algorithm, it is possible to replace the continuous-time flow model adopted here with many other flow-based models. A performance comparison of these models will be useful.

One limitation is the presumed Markovian structure of the distributions in the first stage of our method. Future research may focus on designing a more adaptive scheme for non-Markovian models with more sophisticated graph structures. While our preliminary experimental results are promising, the potential of tensorizing flow has yet to be explored and compared, especially for large-scale real-world datasets.

#### FUNDING

Lexing Ying is partially supported by National Science Foundation under Award No. DMS-2011699. Yuehaw Khoo is partially supported by National Science Foundation under Award No. DMS-2111563 and U.S. Department of Energy, Office of Science under Award No. DE-SC0022232.

#### Data Availability

The datasets used and/or analyzed during the current study are available from the corresponding author on reasonable request.

#### Author details

<sup>1</sup>Institute for Computational and Mathematical Engineering (ICME), Stanford University, Stanford, CA 94305, USA,

<sup>2</sup>Department of Statistics, University of Chicago, Chicago, IL 60637, USA, <sup>3</sup>Department of Mathematics and ICME, Stanford University, Stanford, CA 94305, USA.

## 6 Appendix A. Proof of Proposition 3.3

In this appendix, we prove Proposition 3.3 from Sect. 3.1:

*Proof of Proposition 3.3* For  $2 \leq k \leq d$ , it suffices for us to consider the  $k$ -th equation in (3.1):

$$\sum_{\alpha_{k-1}=1}^{r_{k-1}} \Phi_{k-1}(x_{1:k-1}; \alpha_{k-1}) G_k(\alpha_{k-1}; x_k, \alpha_k) = \Phi_k(x_{1:k-1}; x_k, \alpha_k). \quad (\text{A.1})$$

By Definition 3.1, there exist orthonormal right singular vectors

$$\{\Psi_{k-1}(\alpha_{k-1}; x_{k:d})\}_{1 \leq \alpha_{k-1} \leq r_{k-1}} \subset L^2(I^{d-k+1})$$

of  $p(x_{1:k-1}; x_{k:d})$  and

$$\{\Psi_k(\alpha_k; x_{k+1:d})\}_{1 \leq \alpha_k \leq r_k} \subset L^2(I^{d-k})$$

of  $p(x_{1:k}; x_{k+1:d})$ , and corresponding singular values  $\sigma_{k-1}(1) \geq \cdots \geq \sigma_{k-1}(r_{k-1})$  and  $\sigma_k(1) \geq \cdots \geq \sigma_k(r_k)$ , satisfying

$$p(x_{1:k-1}; x_{k:d}) = \sum_{\alpha_{k-1}=1}^{r_{k-1}} \sigma_{k-1}(\alpha_{k-1}) \Phi_{k-1}(x_{1:k-1}; \alpha_{k-1}) \Psi_{k-1}(\alpha_{k-1}; x_{k:d}), \quad (\text{A.2})$$

and

$$p(x_{1:k}; x_{k+1:d}) = \sum_{\alpha_k=1}^{r_k} \sigma_k(\alpha_k) \Phi_k(x_{1:k}; \alpha_k) \Psi_k(\alpha_k; x_{k+1:d}).$$

Define  $\Xi_k(x_{k+1:d}; \alpha_k) = \sigma_k(\alpha_k)^{-1} \Psi_k(\alpha_k; x_{k+1:d})$ . It is easy to check that

$$\begin{aligned} & \int_{I^{d-k}} p(x_{1:k}; x_{k+1:d}) \Xi_k(x_{k+1:d}; \alpha_k) dx_{k+1:d} \\ &= \int_{I^{d-k}} \sum_{\alpha'_k=1}^{r_k} \sigma_k(\alpha'_k) \sigma_k(\alpha_k)^{-1} \Phi_k(x_{1:k}; \alpha'_k) \Psi_k(\alpha'_k; x_{k+1:d}) \Psi_k(\alpha_k; x_{k+1:d}) dx_{k+1:d} \\ &= \Phi_k(x_{1:k}; \alpha_k). \end{aligned}$$

Therefore, by contracting  $\Xi_k(x_{k+1:d}; \alpha_k)$  to both sides of (A.2), we have

$$\begin{aligned} \Phi_k(x_{1:k}; \alpha_k) &= \int_{I^{d-k}} p(x_{1:k-1}; x_{k:d}) \Xi_k(x_{k+1:d}; \alpha_k) dx_{k+1:d} \\ &= \sum_{\alpha_{k-1}=1}^{r_{k-1}} \sigma_{k-1}(\alpha_{k-1}) \Phi_{k-1}(x_{1:k-1}; \alpha_{k-1}) \int_{I^{d-k}} \Psi_{k-1}(\alpha_{k-1}; x_{k:d}) \Xi_k(x_{k+1:d}; \alpha_k) dx_{k+1:d}, \end{aligned}$$

and consequently,

$$G_k(\alpha_{k-1}; x_k, \alpha_k) = \sigma_{k-1}(\alpha_{k-1}) \int_{I^{d-k}} \Psi_{k-1}(\alpha_{k-1}; x_{k:d}) \Xi_k(x_{k+1:d}; \alpha_k) dx_{k+1:d}$$

solves equation (A.1).

The uniqueness of the solution is guaranteed by the orthogonality of the functions  $\{\Psi_{k-1}(\alpha_{k-1}; x_{k:d})\}_{1 \leq \alpha_{k-1} \leq r_{k-1}}$  by definition. Once  $G_k$  is ready, it is easy to check the validity of (3.2) by plugging the CDE in (3.1) one into the next successively.

**Table 1** Hyperparameters used in the examples

| Example              | Instance       | $N$  | $M$ | $N_{\text{batch}}$ | $D$  | LR   | WD   | $\gamma$ |
|----------------------|----------------|------|-----|--------------------|------|------|------|----------|
| Rosenbrock(Figure 7) | TF/NF          | 1e+5 | 30  | 5e+3               | 64   | 5e-4 | 2e-3 | 0.9      |
| 1D GL(Figure 11)     | TF/NF          | 1e+4 | 25  | 5e+3               | 128  | 5e-3 | 1e-3 | 0.9      |
| 1D GL(Figure 12)     | TF/NF          | 1e+3 | 25  | 1e+3               | 128  | 5e-3 | 1e-3 | 0.9      |
|                      |                | 1e+4 | 25  | 5e+3               | 128  | 5e-3 | 1e-3 | 0.9      |
|                      |                | 1e+5 | 25  | 5e+3               | 128  | 2e-3 | 1e-3 | 0.85     |
|                      |                | 1e+3 | 25  | 1e+3               | 128  | 5e-3 | 1e-3 | 0.9      |
| 1D GL(Figure 13)     | TF/NF          | 1e+3 | 25  | 1e+3               | 128  | 5e-3 | 1e-3 | 0.9      |
|                      | Overfitting NF | 1e+3 | 25  | 1e+3               | 1024 | 2e-3 | 1e-3 | 0.9      |
| 2D GL(Figure 15)     | TF/NF          | 1e+4 | 25  | 5e+3               | 128  | 5e-3 | 1e-3 | 0.9      |

## 7 Appendix B. Hyperparameters

This section presents the hyperparameters of our tensorizing flow algorithm used for each example in Sect. 4. For simplicity, we choose the internal ranks  $r_k = 2$  for  $1 \leq k \leq d - 1$ , and the number of quadrature points  $l = 20$  for all numerical integrations involved. We set the time horizon  $T = 0.2$  with step-size  $\tau = 0.01$  in the flow model. We choose the bandwidth parameter  $h$  in (3.6) to be 5% of the range of the data. We generate  $N/2$  samples separately from the training samples as the test samples. The rest of the hyperparameters are organized in Table 1.

Received: 24 February 2023 Accepted: 9 June 2023 Published online: 21 June 2023

## References

- Bachmayr, M., Schneider, R., Uschmajew, A.: Tensor networks and hierarchical tensors for the solution of high-dimensional partial differential equations. *Found. Comput. Math.* **16**, 1423–1472 (2016)
- Baiardi, A., Reiher, M.: The density matrix renormalization group in chemistry and molecular physics: recent developments and new challenges. *J. Chem. Phys.* **152**, 040903 (2020)
- Batchelor, G.K.: *An Introduction to Fluid Dynamics*. Cambridge University Press, Cambridge (2000)
- Behrmann, J., Grathwohl, W., Chen, R.T., Duvenaud, D., Jacobsen, J.-H.: Invertible residual networks. In: *International Conference on Machine Learning*, PMLR, pp. 573–582 (2019)
- Bengio, Y., Ducharme, R., Vincent, P.: A neural probabilistic language model. *Adv Neural Inf. Process. Syst.* **13** (2000)
- Bishop, C.M., Nasrabadi, N.M.: *Pattern Recognition and Machine Learning*, vol. 4. Springer, Cham (2006)
- Blei, D.M., Kucukelbir, A., McAuliffe, J.D.: Variational inference: a review for statisticians. *J. Am. Stat. Assoc.* **112**, 859–877 (2017)
- Bond-Taylor, S., Leach, A., Long, Y., Willcocks, C.G.: Deep generative modelling: A comparative review of vaes, gans, normalizing flows, energy-based and autoregressive models, arXiv preprint [arXiv:2103.04922](https://arxiv.org/abs/2103.04922) (2021)
- Bonnevie, R., Schmidt, M.N.: Matrix product states for inference in discrete probabilistic models. *The. J. Mach. Learn. Res.* **22**, 8396–8443 (2021)
- Bradley, T.-D., Stoudenmire, E.M., Terilla, J.: Modeling sequences with quantum states: a look under the hood. *Mach. Learn. Sci. Technol.* **1**, 035008 (2020)
- Brandao, F.G., Horodecki, M.: Exponential decay of correlations implies area law. *Commun. Math. Phys.* **333**, 761–798 (2015)
- Chan, G.K.-L., Sharma, S.: The density matrix renormalization group in quantum chemistry. *Annu. Rev. Phys. Chem.* **62**, 465–481 (2011)
- Chen, C., Li, C., Chen, L., Wang, W., Pu, Y., Duke, L.C.: Continuous-time flows for efficient inference and density estimation. In: *International Conference on Machine Learning*, PMLR, pp. 824–833 (2018)
- Chen, R.T., Rubanova, Y., Bettencourt, J., Duvenaud, D.K.: Neural ordinary differential equations. *Adv. Neural Inf. Process. Syst.* **31** (2018)
- Cheng, S., Wang, L., Xiang, T., Zhang, P.: Tree tensor networks for generative modeling. *Phys. Rev. B* **99**, 155131 (2019)
- Cichocki, A., Zdunek, R., Phan, A.H., Amari, S.-I.: *Nonnegative Matrix and Tensor Factorizations: Applications to Exploratory Multi-way Data Analysis and Blind Source Separation*. Wiley, London (2009)
- De Lathauwer, L., De Moor, B., Vandewalle, J.: A multilinear singular value decomposition. *SIAM J. Matrix Anal. Appl.* **21**, 1253–1278 (2000)
- De Lathauwer, L., De Moor, B., Vandewalle, J.: On the best rank-1 and rank- $(r_1, r_2, \dots, r_n)$  approximation of higher-order tensors. *SIAM J. Matrix Anal. Appl.* **21**, 1324–1342 (2000)
- Dinh, L., Krueger, D., Bengio, Y.: Nice: non-linear independent components estimation. arXiv preprint [arXiv:1410.8516](https://arxiv.org/abs/1410.8516) (2014)
- Dinh, L., Sohl-Dickstein, J., Bengio, S.: Density estimation using real NVP. arXiv preprint [arXiv:1605.08803](https://arxiv.org/abs/1605.08803) (2016)

21. Dolgov, S., Anaya-Izquierdo, K., Fox, C., Scheichl, R.: Approximation and sampling of multivariate probability distributions in the tensor train decomposition. *Stat. Comput.* **30**, 603–625 (2020)
22. Dolgov, S.V., Khoromskij, B.N., Oseledets, I.V., Savostyanov, D.V.: Computation of extreme eigenvalues in higher dimensions using block tensor train format. *Comput. Phys. Commun.* **185**, 1207–1216 (2014)
23. Dupont, E., Doucet, A., Teh, Y.W.: Augmented neural odes. *Adv. Neural Inf. Process. Syst.* **32** (2019)
24. Durkan, C., Bekasov, A., Murray, I., Papamakarios, G.: Cubic-spline flows, arXiv preprint [arXiv:1906.02145](https://arxiv.org/abs/1906.02145) (2019)
25. Durkan, C., Bekasov, A., Murray, I., Papamakarios, G.: Neural spline flows. *Adv. Neural Inf. Process. Syst.* **32** (2019)
26. Ren, W., Vanden-Eijnden, E.: Minimum action method for the study of rare events. *Commun. Pure Appl. Math.* **57**, 637–656 (2004)
27. Germain, M., Gregor, K., Murray, I., Larochelle, H.: Made: masked autoencoder for distribution estimation. In: *International Conference on Machine Learning*, PMLR, pp. 881–889 (2015)
28. Gomez, A.N., Ren, M., Urtasun, R., Grosse, R.B.: The reversible residual network: backpropagation without storing activations. *Adv. Neural Inf. Process. Syst.* **30** (2017)
29. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. *Adv. Neural Inf. Process. Syst.* **27** (2014)
30. Grasedyck, L.: Hierarchical singular value decomposition of tensors. *SIAM J. Matrix Anal. Appl.* **31**, 2029–2054 (2010)
31. Grathwohl, W., Chen, R.T., Bettencourt, J., Sutskever, I., Duvenaud, D.: Fjord: free-form continuous dynamics for scalable reversible generative models. arXiv preprint [arXiv:1810.01367](https://arxiv.org/abs/1810.01367) (2018)
32. Han, Z.-Y., Wang, J., Fan, H., Wang, L., Zhang, P.: Unsupervised generative modeling using matrix product states. *Phys. Rev. X* **8**, 031012 (2018)
33. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778 (2016)
34. Hinton, G.E.: Training products of experts by minimizing contrastive divergence. *Neural Comput.* **14**, 1771–1800 (2002)
35. Hinton, G.E., Sejnowski, T.J.: Optimal perceptual inference. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, vol. 448, Citeseer, pp. 448–453 (1983)
36. Hoffman, M.D., Blei, D.M., Wang, C., Paisley, J.: Stochastic variational inference. *J. Mach. Learn. Res.* (2013)
37. Hohenberg, P., Krekhov, A.: An introduction to the Ginzburg-Landau theory of phase transitions and nonequilibrium patterns. *Phys. Rep.* **572**, 1–42 (2015)
38. Hur, Y., Hoskins, J.G., Lindsey, M., Stoudenmire, E., Khoo, Y.: Generative modeling via tensor train sketching. arXiv preprint [arXiv:2202.11788](https://arxiv.org/abs/2202.11788) (2022)
39. Jacobsen, J.-H., Smeulders, A., Oyallon, E.: i-revnet: deep invertible networks. arXiv preprint [arXiv:1802.07088](https://arxiv.org/abs/1802.07088) (2018)
40. Khoo, Y., Lindsey, M., Zhao, H.: Tensorizing flows: a tool for variational inference. arXiv preprint [arXiv:2305.02460](https://arxiv.org/abs/2305.02460) (2023)
41. Kingma, D.P., Dhariwal, P.: Glow: generative flow with invertible 1x1 convolutions. *Adv. Neural Inf. Process. Syst.* **31** (2018)
42. Kingma, D.P., Salimans, T., Jozefowicz, R., Chen, X., Sutskever, I., Welling, M.: Improved variational inference with inverse autoregressive flow. *Adv. Neural Inf. Process. Syst.* **29** (2016)
43. Kingma, D.P., Welling, M.: Auto-encoding variational bayes. arXiv preprint [arXiv:1312.6114](https://arxiv.org/abs/1312.6114) (2013)
44. Kobyzev, I., Prince, S.J., Brubaker, M.A.: Normalizing flows: an introduction and review of current methods. *IEEE Trans. Pattern Anal. Mach. Intell.* **43**, 3964–3979 (2020)
45. Kressner, D., Uschmajew, A.: On low-rank approximability of solutions to high-dimensional operator equations and eigenvalue problems. *Linear Algebra Appl.* **493**, 556–572 (2016)
46. Kressner, D., Vandereycken, B., Voorhaar, R.: Streaming tensor train approximation. arXiv preprint [arXiv:2208.02600](https://arxiv.org/abs/2208.02600) (2022)
47. Larochelle, H., Murray, I.: The neural autoregressive distribution estimator. In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, JMLR Workshop and Conference Proceedings*, pp. 29–37 (2011)
48. Miao, Y., Yu, L., Blunsom, P.: Neural variational inference for text processing. In: *International Conference on Machine Learning*, PMLR, pp. 1727–1736 (2016)
49. Mnih, A., Gregor, K.: Neural variational inference and learning in belief networks. In: *International Conference on Machine Learning*, PMLR, pp. 1791–1799 (2014)
50. Novikov, G.S., Panov, M.E., Oseledets, I.V.: Tensor-train density estimation. In: *Uncertainty in Artificial Intelligence*, PMLR, pp. 1321–1331 (2021)
51. Oseledets, I., Tyrtysnikov, E.: Tt-cross approximation for multidimensional arrays. *Linear Algebra Appl.* **432**, 70–88 (2010)
52. Oseledets, I.V.: Tensor-train decomposition. *SIAM J. Sci. Comput.* **33**, 2295–2317 (2011)
53. Papamakarios, G., Pavlakou, T., Murray, I.: Masked autoregressive flow for density estimation. *Adv. Neural Inf. Process. Syst.* **30** (2017)
54. Penrose, R.: Applications of negative dimensional tensors. *Combinat. Math. Appl.* **1**, 221–244 (1971)
55. Perez-Garcia, D., Verstraete, F., Wolf, M.M., Cirac, J.I.: Matrix product state representations. arXiv preprint [quant-ph/0608197](https://arxiv.org/abs/quant-ph/0608197) (2006)
56. Ranganath, R., Gerrish, S., Blei, D.: Black box variational inference. In: *Artificial Intelligence and Statistics*, PMLR, pp. 814–822 (2014)
57. Rezende, D., Mohamed, S.: Variational inference with normalizing flows. In: *International Conference on Machine Learning*, PMLR, pp. 1530–1538 (2015)
58. Rezende, D.J., Mohamed, S., Wierstra, D.: Stochastic backpropagation and approximate inference in deep generative models. In: *International Conference on Machine Learning*, PMLR, pp. 1278–1286 (2014)
59. Robeva, E., Seigal, A.: Duality of graphical models and tensor networks. *Inf. Inference J. IMA* **8**, 273–288 (2019)
60. Savostyanov, D., Oseledets, I.: Fast adaptive interpolation of multi-dimensional arrays in tensor train format. In: *The International Workshop on Multidimensional (ND) Systems*. IEEE pp. 1–8 (2011)



61. Schmidhuber, J.: Generative adversarial networks are special cases of artificial curiosity (1990) and also closely related to predictability minimization (1991). *Neural Netw.* **127**, 58–66 (2020)
62. Shi, T., Ruth, M., Townsend, A.: Parallel algorithms for computing the tensor-train decomposition. arXiv preprint [arXiv:2111.10448](https://arxiv.org/abs/2111.10448) (2021)
63. Stein, E.M., Shakarchi, R.: *Real Analysis: Measure Theory, Integration, and Hilbert Spaces*. Princeton University Press, Princeton (2009)
64. Steinlechner, M.: Riemannian optimization for high-dimensional tensor completion. *SIAM J. Sci. Comput.* **38**, S461–S484 (2016)
65. Szeg, G.: *Orthogonal Polynomials.*, vol. 23, American Mathematical Society (1939)
66. Tabak, E.G., Vanden-Eijnden, E.: Density estimation by dual ascent of the log-likelihood. *Commun. Math. Sci.* **8**, 217–233 (2010)
67. Tang, X., Hur, Y., Khoo, Y., Ying, L.: Generative modeling via tree tensor network states. arXiv preprint [arXiv:2209.01341](https://arxiv.org/abs/2209.01341) (2022)
68. Temme, K., Verstraete, F.: Stochastic matrix product states. *Phys. Rev. Lett.* **104**, 210502 (2010)
69. Tzen, B., Raginsky, M.: Neural stochastic differential equations: deep latent Gaussian models in the diffusion limit. arXiv preprint [arXiv:1905.09883](https://arxiv.org/abs/1905.09883) (2019)
70. Vieijra, T., Vanderstraeten, L., Verstraete, F.: Generative modeling with projected entangled-pair states. arXiv preprint [arXiv:2202.08177](https://arxiv.org/abs/2202.08177) (2022)
71. Wang, W., Aggarwal, V., Aeron, S.: Tensor train neighborhood preserving embedding. *IEEE Trans. Signal Process.* **66**, 2724–2732 (2018)
72. White, S.R.: Density-matrix algorithms for quantum renormalization groups. *Phys. Rev. B* **48**, 10345 (1993)
73. Young, N.: *An Introduction to Hilbert Space*. Cambridge University Press, Cambridge (1988)
74. Zhang, L., Wang, L., et al.: Monge-ampère flow for generative modeling. arXiv preprint [arXiv:1809.10188](https://arxiv.org/abs/1809.10188) (2018)

## Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor (e.g. a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.