Journal of Computational Physics 468 (2022) 111511

Contents lists available at ScienceDirect

Journal of Computational Physics

www.elsevier.com/locate/jcp

Physics-informed distribution transformers via molecular dynamics and deep neural networks

Difeng Cai

Department of Mathematics, Emory University, 400 Dowman Drive, GA 30322, Atlanta, USA

ARTICLE INFO

Article history:
Received 21 January 2022
Received in revised form 12 June 2022
Accepted 24 July 2022
Available online 1 August 2022

Keywords:
Quasirandom points
Deep neural networks
Molecular dynamics
Residual neural networks
Potential theory

ABSTRACT

Generating quasirandom points with high uniformity is a fundamental task in many fields. Existing number-theoretic approaches produce evenly distributed points in $[0,1]^d$ in asymptotic sense but may not yield a good distribution for a given set size. It is also difficult to extend those techniques to other geometries like a disk or a manifold. In this paper, we present a novel physics-informed framework to transform a given set of points into a distribution with better uniformity. We model each point as a particle and assign the system with a potential energy. Upon minimizing the energy, the uniformity of distribution can be improved correspondingly. Two kinds of schemes are introduced: one based on molecular dynamics and another based on deep neural networks. The new physics-informed framework serves as a black-box transformer that is able to improve given distributions and can be easily extended to other geometries such as disks, spheres, complex manifolds, etc. Various experiments with different geometries are provided to demonstrate that the new framework is able to transform poorly distributed input into one with superior uniformity.

© 2022 Elsevier Inc. All rights reserved.

1. Introduction

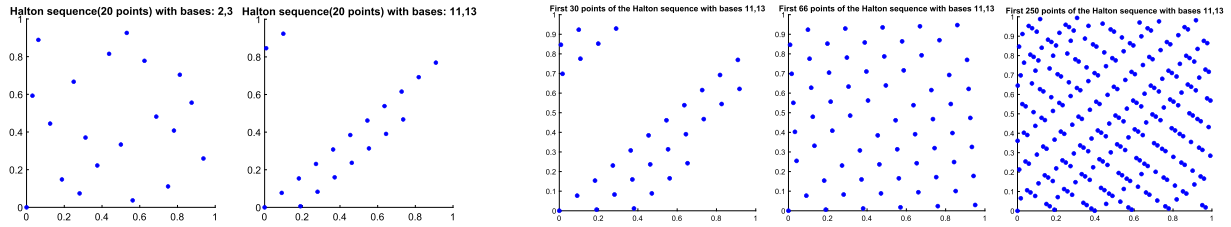
Generating quasirandom points with good uniformity plays a key role in various applications, such as numerical integration [1,2], computer graphics [3,4], image reconstruction [5], machine learning [6,7], etc. For example, in numerical integration, quasi-Monte Carlo methods aim to improve the convergence rate of Monte Carlo by using points that are evenly distributed in the unit cube $[0,1]^d$. A large amount of work is devoted to the study of uniformity of distribution in $[0,1]^d$, measured by the concept of *discrepancy* [8,9,1,2,10]. A set with *low* discrepancy is considered to have good uniformity. Various formulas have been developed to generate points with low discrepancy, including Halton sequence [11], Sobol′ sequence [12], digital sequence [9,1], lattice points [13,14], etc. Those points are shown to have a discrepancy that converges to zero in a certain rate as the number of points approaches infinity (cf. [1,10] or Section 2.2).

Despite the proved uniformity in the asymptotic sense, it is well-known that (cf. [15,16,10]), for a fixed set size N, the N points produced by existing formulas may display poor uniformity. For example, the two-dimensional Halton sequence with relatively prime bases b_1, b_2 is given by

$$x_n = (\phi_{b_1}(n-1), \phi_{b_2}(n-1)), \quad n = 1, 2, \ldots,$$

E-mail address: dcai7@emory.edu.

https://doi.org/10.1016/j.jcp.2022.111511
0021-9991/© 2022 Elsevier Inc. All rights reserved.



(a) Bases (2,3) and bases (11,13). (b) From left to right: 30, 66, 250 points.

Fig. 1. Points from the Halton sequence with different bases (left) and set sizes (right) can possess contrasting degrees of uniformity.

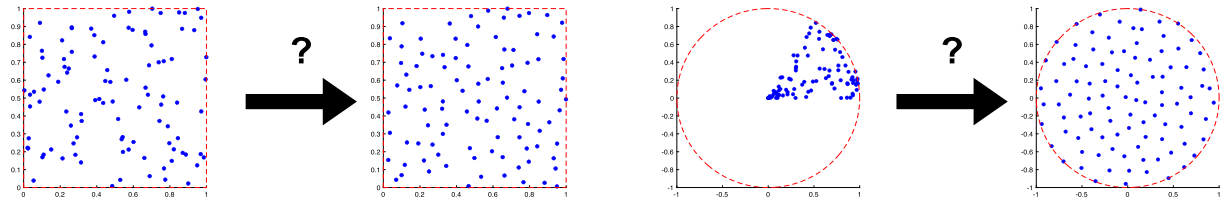


Fig. 2. Transforming random samples in $[0,1]^2$ (left) and a poor distribution in a disk (right) into better distributions.

where $\phi_b(n) = \sum_{k=1}^{\infty} n_k b^{-k} \in [0,1)$ denotes the base b radical inverse function and $n = \sum_{k=1}^{\infty} n_k b^{k-1}$ is the representation of n in base b (only finitely many n_k's are nonzero). Though the infinite set $\{x_n\}_{n=1}^{\infty}$ is shown to have low discrepancy, the finite N points $x_1, \ldots, x_N$ may have poor uniformity due to the choice of N or the bases. Fig. 1 illustrates this issue. The two plots in Fig. 1a show the first 20 points of the Halton sequences with two different pairs of bases. It is easy to see that the set with bases (2,3) demonstrates much better uniformity than the one with bases (11,13). Fig. 1b shows that different set sizes (first N points) of the Halton sequence (with bases 11,13) can display varying degrees of the uniformity, where $N = 66$ points achieve extraordinary uniformity while $N = 30$ points are not distributed evenly at all. When $N = 250$, the distribution has low discrepancy but there is a strong correlation between certain points which impairs the quasirandomness of the distribution.

We see from above that in the *pre-asymptotic* regime, points in a low discrepancy sequence may *not* necessarily possess good uniformity. To remedy a poor distribution, permutation or scrambling techniques have been proposed [15,17–19]. The new set of points obtained by applying those number-theoretic techniques can achieve significantly better uniformity. Those techniques are closely related to the construction of low discrepancy sequences and often assume that the input point set is from a low discrepancy sequence. In this paper, we are interested in the more general case where the input points can be unstructured random samples or poorly distributed in a domain as illustrated in Fig. 2. Note that even though existing work only focuses on studying distributions in the unit cube, the problem of improving poor distributions is also of great interest for other geometries like a disk, a sphere or more complex manifolds, for which the concept of discrepancy is *not* defined and existing machineries can not be applied.

In this paper, we consider the problem of improving the uniformity of a given set of possibly poorly distributed points (see Fig. 2). We present a novel physics-informed framework based on physics principles to transform a given distribution for better quasirandomness. It serves as a versatile tool complementary to existing number-theoretic approaches and, furthermore, naturally extends to improving distributions over a disk or a manifold.

Methodology and contributions. We introduce a physics-informed framework to modify a given distribution for better uniformity. Entirely different from existing approaches, we model each point as a particle and associate the system of particles with a potential energy. Adjusting the distribution for better uniformity corresponds to moving particles in the physical system towards a state with low potential energy. Based on this principle, we present two different schemes for transforming the given point set: (1) molecular dynamics(MD) simulation; (2) deep neural networks(DNN). The MD simulation is commonly used to study the interaction between particles and the evolution of particle systems governed by certain physics laws (cf. [20–24]). It has also been used to sample the potential energy surfaces [25,26]. In those applications, to study the interaction between different particles and the structure of molecules, of importance are chemical bonds, electric charges, van der Waals forces, etc. Different from existing work in MD simulations, we are interested in transforming the configuration and the only quantity that matters is the location of each particle instead of radius, chemical bond, charge, etc. Thus we model each point as a particle with equal mass and ignore the radius of the particle. The MD simulation is used to shift particles so as to resolve clumps and fill in holes in the system, thus creating a better distribution. We remark that using MD to achieve better uniformity has existed "informally" in the MD community, but no work has been done to investigate it systematically and quantitatively.

Contrary to MD simulation, in the DNN approach, we completely ignore the kinetic energy and approximate the equilibrium state by minimizing the total potential energy of all particles, where each particle is represented as a neural network. This enables the use of modern deep learning techniques to search for a state with low energy.

Compared to classical number-theoretic approaches, introducing energy into the study of distributions offers a lot more flexibility. It can be directly applied to improving the uniformity of poorly distributed points. When applied to random samples, the energy-based distribution transformer can efficiently improve the uniformity and thus can be used to generate quasirandom points. It offers a straightforward generalization to improving distributions on a disk, a sphere or a general manifold, whereas existing methods are limited to the unit cube. Generally speaking, the new framework presents a dynamical characterization of distributions and can potentially transfer the study of uniform distribution to other distributions of particular interest by simply changing the potential energy. Specifically, it can be used to

- improve the quasirandomness of points in a low discrepancy sequence;
- transform random samples into a distribution with better uniformity;
- transform given points on a disk or a manifold into a better distribution.

The rest of the manuscript is organized as follows. Section 2 reviews existing work on generating quasirandom points or low discrepancy sequences. The new physics-informed framework is presented in Section 3, where two kinds of distribution transformers, based on molecular dynamics and deep neural networks, respectively, are introduced. Section 4 presents the distribution transformers for disks, spheres, and manifolds in general. Section 5 investigates theoretically the potential energy and distributions. Numerical experiments are presented in Section 6 followed by a discussion in Section 7. A conclusion is drawn in Section 8.

Notation. Throughout the presentation, we use $|\cdot|$ to denote the magnitude of a vector (i.e. Euclidean norm). $\#X$ denotes the cardinality of a finite set X.

2. Low discrepancy sequences in $[0,1]^d$

In Section 2.1, we review the definition of discrepancy for a finite set of points in the unit cube $[0,1]^d$. Then we review in Section 2.2 existing techniques to construct low discrepancy points. Section 2.3 discusses limitations associated with existing approaches, which motivate the work in this manuscript.

2.1. Discrepancy

Discrepancy is a concept that measures the uniformity of a finite set of points in the unit cube $[0,1]^d$. Smaller discrepancy indicates better uniformity. Quasi-Monte Carlo methods consist in the design and use of low discrepancy points in place of random samples in the Monte Carlo method in order to improve the convergence rate for approximating integrals.

There are several definitions for describing the discrepancy (cf. [1,27]) and the most commonly used ones are the star discrepancy and the extreme discrepancy.

Definition 1. The star discrepancy $D_N^*(X)$ of $X=\{x_1,\dots,x_N\}$ in $[0,1]^d$ is defined by $D_N^*(X) := \sup_{J\in\mathcal{J}^*} |\#(X\cap J)/N - \lambda(J)|$, where λ denotes the Lebesgue measure in $\mathbb{R}^d$ and $\mathcal{J}^*$ is the family of all subintervals in $[0,1)^d$ of the form $\prod_{i=1}^d [0,a_i)$.

Definition 2. The extreme discrepancy $D_N(X)$ of $X=\{x_1,\dots,x_N\}$ in $[0,1]^d$ is defined by $D_N(X) := \sup_{J\in\mathcal{J}} |\#(X\cap J)/N - \lambda(J)|$, where $\mathcal{J}$ is the family of all subintervals in $[0,1)^d$ of the form $\prod_{i=1}^d [a_i,b_i)$.

Roughly speaking, if the number of points in $X\cap J$ is more or less proportional to the measure of J for any box J, then the discrepancy of X is small. If there are many clumps in X, then its discrepancy will be large. The star discrepancy in Definition 1 is also known as the L_∞-*star discrepancy* since the measure $\sup|\cdot|$ can be viewed as the L_∞ norm (cf. [28]). More generally, the L_p-*star discrepancy* is defined by using L_p norm to measure the difference $\#(X\cap J)/N-\lambda(J)$. As can be seen from the definition, the calculation of discrepancy is not straightforward in general. However, the L_2-star discrepancy for $X=\{x_i\}_{i=1}^N$ can be calculated directly using Warnock's formula [29]:

$$D_{L_2,N}^* = \frac{1}{3^d} - \frac{2^{1-d}}{N}\sum_{i=1}^N \Pi_{k=1}^d (1-(x_i^{(k)})^2) + \frac{1}{N^2}\sum_{i,j=1}^N C_{k,i,j}, \tag{1}$$

where $C_{k,i,j}=\min(1-x_i^{(k)}, 1-x_j^{(i)})$ and $x_i^{(k)}$ denotes the kth component of x_i. This discrepancy will be used in performing experiments in Section 6. We review in the next subsection some representative sets and sequences with low discrepancy.

2.2. Low discrepancy sets and sequences

Extensive research has been done to construct a set or a sequence of points $x_1, x_2, \cdots \in [0,1]^d$ such that $D_N^*(x_1, \ldots, x_N)$ is small when N is sufficiently large. The point set is considered to have low discrepancy if $D_N^*(x_1, \ldots, x_N) = O(N^{-1}(\log N)^d)$ for sufficiently large N. The topic is relatively mature and we follow the presentations in [1,10].

There are in general two classes of approaches in constructing low discrepancy points in $[0,1]^d$. Examples from the first class include Halton sequence [11], digital net [9,1], digital sequence [9,1], Sobol′ sequence [12,30], etc. The second class consists of lattice points [13,14,31].

Definition 3 *(Halton sequence).* Let $\phi_b(n) = \phi_b\left(\sum_{k=1}^{\infty} n_k b^{k-1}\right) = \sum_{k=1}^{\infty} n_k b^{-k}$ be the radical inverse function in base b. The Halton sequence in the pairwise relatively prime bases $b_1, \ldots, b_d$ is defined as

$$x_n = (\phi_{b_1}(n-1), \ldots, \phi_{b_d}(n-1)) \in [0,1)^d, \quad n = 1, 2, \ldots.$$

Definition 4 *(Elementary interval).* For an integer $b \geq 2$, a subinterval E of $[0,1)^d$ of the form $E = \prod_{i=1}^{d} [a_i b^{-p_i}, (a_i+1)b^{-p_i})$ with non-negative integers a_i, p_i such that $a_i < b^{p_i}$ $(1 \leq i \leq d)$ is called an elementary interval in base b.

Definition 5 *(Digital net).* Let $0 \leq t \leq m$ be integers. A (t, m, d)-net in base b is a set X of b^m points in $[0,1)^d$ such that the cardinality of $X \cap E$ is b^t for every elementary interval E in base b with $\lambda(E) = b^{t-m}$.

Note that the size of a digital net can *not* be arbitrary (the cardinality can only be b^m). Hence it is not *extensible*, meaning that we can not specify an arbitrary set size or add an arbitrary number of points to the set. The extensible version of a digital net is the so-called digital sequence.

Definition 6 *(Digital sequence).* Let $b \geq 2$ and $t \geq 0$ be integers. A sequence $x_0, x_1, \ldots$ of points in $[0,1)^d$ is a (t,d)*-sequence in base* b if, for all integers $k \geq 0$ and $m > t$, the point set consisting of the x_n with $kb^m \leq n < (k+1)b^m$ is a (t, m, d)-net in base b.

The popular Sobol′ sequence [12] is a (t,d)-sequence in base $b = 2$. According to discrepancy theory [1,27], the above sets or sequences all have low discrepancy in the sense that $D_N^*(X) = O(N^{-1}(\log N)^d)$, where X can be one of the following: (1) the first N terms of a Halton sequence; (2) a (t, m, d)-net in base b with $m > 0$; (3) the first N terms of a (t,d)-sequence in base b.

A different way of constructing low discrepancy points is given by the lattice rule [13,14,32–34]. The lattice points have a periodic structure and are associated with a generating vector.

Definition 7 *(Lattice point set).* Let $\mathbf{g} \in \mathbb{Z}^d$ and $N \in \mathbb{Z}_+$. For $k = 0, 1, \ldots, N-1$, define x_k to be the fractional part of $\frac{k\mathbf{g}}{N}$. The point set $X_N = \{x_0, x_1, \ldots, x_{N-1}\}$ in $[0,1)^d$ is called a *lattice point set* and $\mathbf{g}$ is called the *generating vector* of X_N.

Definition 8 *(Lattice sequence).* Let $\mathbf{g} \in \mathbb{Z}^d$ and ϕ_b be the radical inverse function in base b. The lattice sequence x_k $(k = 0, 1, \ldots)$ in base b is defined as the fractional part of $\phi_b(k)\mathbf{g}$.

The quality of a lattice set or sequence hinges on the choice of the generating vector $\mathbf{g}$. As described in the theorem below (cf. [1]), it can be shown that there does exist a good generating vector $\mathbf{g}$ such that the resulting lattice points have low discrepancy. In practice, constructing good generating vectors is nontrivial. Efficient algorithms for computing good generating vectors for the lattice rule can be found in [31,34].

Theorem 1. *There exists a generating vector* $\mathbf{g}$ *such that the lattice point set in Definition 7 has discrepancy* $D_N(X_N) = O(N^{-1}(\log N)^d)$.

Even though the low discrepancy points above are proved to achieve good uniformity as N approaches infinity, it is possible that, for a finite set size, the points are *not* evenly distributed. As shown in Fig. 1, the quality of low discrepancy points depends on the set size and parameters in the generating formula. In Section 2.3, we review the limitations of existing low discrepancy sequences.

2.3. Limitations

Low discrepancy sequence displays good uniformity in $[0,1]^d$ as the number of points becomes large enough. For a fixed size N, the uniformity of the N points from a low discrepancy sequence is unpredictable.

Pre-asymptotic regime Theoretically, the concept of *low discrepancy* is discussed in asymptotic sense only (cf. Section 2.2). For a finite set, as we have seen in Section 1, the points drawn from a low discrepancy sequence may not be distributed evenly.

Improving uniformity Permutation and scrambling techniques [15,17–19] can be used to significantly improve a set of poorly distributed low discrepancy points. For a general set of points which are not generated by any formula or are located on a disk or a manifold, however, those techniques may not be effective.

Geometry Note that discrepancy is only defined for the unit cube $[0,1]^d$. The low discrepancy sets and sequences in Section 2.2 all hinge on the special properties of the unit cube $[0,1]^d$, such as periodicity, tensor product structure, etc. In general, it is quite difficult to extend the algebraic or combinatoric machinery to generating good distributions over other geometries like a disk, a sphere, etc, which are of great interest in many applications (cf. [35–37]).

We aim to develop a general black-box distribution transformer that does not require any generating formula for the input data and has the potential to be extended to different geometries. An efficient distribution transformer enables a new way to construct quasirandom points, i.e., by simply transforming random samples.

3. Physics-informed distribution transformers

In this section, we present a versatile framework to deal with distributions from a new perspective. The key lies in assigning the given set of points with a potential energy and modeling the points as particles with pairwise interactions dictated by the potential. Due to the strong repulsive forces between particles that are too close, the system will evolve towards a state with lower potential energy, which corresponds to a better distribution with more evenly spaced points. The incorporation of interaction energy into the study of distributions greatly extends the scope of existing machinery and can be used to study various distributions without resorting to any algebraic or analytic formula.

Based on the energy principle, we introduce two different approaches to improve the quality of a given distribution: molecular dynamics(MD) simulations and deep neural networks(DNN). The MD approach relies on classical molecular dynamics simulations, where the system is equipped with both potential energy and kinetic energy and particles move according to Newton's law of motion. The DNN approach ignores the kinetic energy and searches for the equilibrium state via minimizing the potential energy, where the state is represented by a deep neural network. We first review the basics of molecular dynamics simulations and deep neural networks in Section 3.1. Then we introduce the corresponding distribution transformers in Section 3.2 and Section 3.3.

3.1. Review of molecular dynamics and neural networks

3.1.1. Molecular dynamics (MD)

Molecular dynamics(MD) is a popular tool to simulate the motion of particles in fluid dynamics, material sciences, chemical engineering, and biology. The system of particles are subject to pairwise interactions and the evolution of the system is obtained by numerically solving Newton's law of motion with time stepping. In practical applications, the simulation may involve millions of time steps and a large number of particles. For large-scale simulations, many high performance software packages have been developed, such as Desmond [38], GROMACS [39], LAMMPS [40,41], NAMD [42], OpenMM [43], etc. We review in the following some basic notions and algorithms in MD.

The evolution of a system of interacting particles is determined by the Newton's law of motion: $m_i \dfrac{d^2 x_i}{dt^2} = F_i$, where m_i and x_i denote the mass and location of the ith particle, respectively, and F_i denotes the force imposed on particle i. Given a potential function $U(x,y)$ and a system of N particles $x_1,\dots,x_N \in \mathbb{R}^d$, the potential field and force on particle i are given by

$$U_i = \sum_{\substack{j=1 \\ j\neq i}}^{n} U(x_j, x_i), \quad F_i = -\nabla_{x_i} U_i, \tag{2}$$

respectively. The numerical scheme to compute approximate solution to the Newton's equation is called an *integrator*. There are several integrators for molecular dynamics simulations. The most popular class of integrators in molecular dynamics are the Verlet algorithms, including the basic Verlet algorithm [44], the Verlet leapfrog algorithm [45], the velocity Verlet algorithm [46]. The velocity Verlet algorithm is based on the following update formula, which is widely used in practice due to the computational efficiency.

$$x(t+\Delta t) = x(t) + v(t)\Delta t + \frac{F}{2m}\Delta t^2, \quad v(t+\Delta t) = v(t) + \frac{F(t+\Delta t)+F(t)}{2m}\Delta t,$$

where x denotes the location of a particle, v denotes the velocity and Δt denotes the step size for time discretization. The complete algorithm is shown in Algorithm 1. Compared to other schemes, the velocity Verlet algorithm is easy to implement and memory-efficient. Algorithm 1 will be used in Section 3.2 to design the MD-based distribution transformer.

Algorithm 1 Velocity Verlet Algorithm.

Input: Initial positions $\{x_i\}_{i=1}^N$ and velocities $\{v_i\}_{i=1}^N$
Parameters: Potential U, mass m, step size Δt, maximum iteration number M
Output: Positions x_i and velocities v_i after M time steps
1: For each particle i, compute the force $F_i = -\nabla_{x_i} U_i$ as in (2)
2: Update the positions $x_i = x_i + v_i \Delta t + \frac{F_i}{2m} \Delta t^2$
3: Update the velocities partially $v_i = v_i + \frac{F_i}{2m} \Delta t$
4: Update the force field F_i using the new positions x_i
5: Complete the update of velocities $v_i = v_i + \frac{F_i}{2m} \Delta t$
6: Go back to Step 1 unless the number of iterations reaches M
7: **return** $\{x_i\}_{i=1}^N$, $\{v_i\}_{i=1}^N$

3.1.2. Deep neural networks (DNN)

The past decade has seen unprecedented success for the applications of deep neural networks. Due to the expressive power and flexible architecture, deep neural networks have been an indispensable tool in a variety of fields, such as image processing, patter recognition, natural language understanding, artificial intelligence, etc. DNN is also used as a tool for solving scientific computing problems such as partial differential equations (cf. [47–49]). In the following, we review the basics of feedforward deep networks. Mathematically, a *neural network* is a mapping $F_\theta : \mathbb{R}^m \to \mathbb{R}^n$ defined as a composition of a sequence of affine and nonlinear mappings, i.e.,

$$F_\theta = \sigma \circ A_L \circ \sigma \circ A_{L-1} \circ \cdots \sigma \circ A_2 \circ \sigma \circ A_1(x), \tag{3}$$

where $\sigma : \mathbb{R} \to \mathbb{R}$ is called the *activation function* and is applied elementwisely to the input vector or matrix, and each A_k is an affine transformation in the form of $A_k(x) := W_k x + b_k$. Here the matrix W_k is called the *weight matrix* and the vector b_k is the *bias vector*. The symbol θ in (3) denotes the set of all parameters, including weight matrices and bias vectors. Vector input in each composition corresponds to a *layer*, where the *input layer* is x, the *output layer* is $\sigma \circ A_L$, and the intermediate ones are called *hidden layers*. The number L measures the *depth* of the neural network. The length of vector b_k is called the *width* of the kth layer. A *deep* neural network is a neural network with more than one hidden layer.

The activation function σ is usually a nonlinear monotone function. Some commonly used activation functions include ReLU $\sigma(x) = \max(x, 0)$, Sigmoid $\sigma(x) = 1/(1 + e^{-x})$, hyperbolic tangent $\sigma(x) = \tanh(x)$, etc. It is well-known that deep neural networks can approximate any smooth functions [50,51]. Because of the universal approximation property, neural networks are used extensively to model possibly complicated mappings. The parameter set θ is determined by training the neural network, i.e., minimizing a loss function. The definition of the loss function depends on the particular application. The loss function is in general a complicated nonconvex function and a variety of optimization techniques have been developed, such as stochastic gradient descent (SGD) [52,53], ADAGRAD [54], Adam [55], extragradient method [56,57], etc. In practice, due to the large number of parameters and complicated loss function, there is *no* guarantee that those techniques can find any global or local minimum. A solution that significantly reduces the initial loss to a certain level is considered acceptable.

3.2. Distribution transformer via MD

We present two distribution transformers based on molecular dynamics. The first one in Algorithm 2 transforms the input points directly, while the second one in Algorithm 3 applies a random shift to the input points before the transformation.

Basic MD-based distribution transformer. The basic MD-based distribution transformer is given in Algorithm 2. It applies the velocity Verlet algorithm to the particles with several modifications. Firstly, the proposed transformer in Algorithm 2 contains the confinement potential in the definition of total energy, which is not included in the velocity Verlet algorithm in Algorithm 1. Namely, in Algorithm 2, we define the potential field on particle i as

$$U_i = \gamma V(x_i) + \sum_{\substack{j=1 \\ j \neq i}}^{n} U(x_j, x_i) \tag{4}$$

with V the confinement potential and γ the confinement strength. Secondly, since the updated position of a particle can go out of the cube $[0, 1]^d$, for each updated position x, we compute $x = x - \lfloor x \rfloor$ to ensure that each particle lies inside $[0, 1]^d$ during the transformation in Algorithm 2. Here $\lfloor x \rfloor$ denotes the floor function and is applied componentwisely to vector x. A similar but slightly different strategy is used in the lattice rule, where instead of using the floor function, the *fractional part* of a point is chosen to obtain an admissible point in $[0, 1]^d$. A discussion of the two approaches and why we use the floor function is given in Remark 3.1.

Given a set of points, Algorithm 2 is easy to implement and only requires the locations of points. The MD-based model is quite flexible as the user is free to modify the model by adjusting the potential function and model parameters $\Delta t, M, \rho, m$, etc. We discuss the choice of parameters later in this section.

Algorithm 2 Distribution transformer via molecular dynamics.

Input: N points $x_1, \dots, x_N \in [0,1]^d$
Parameters: Potentials U, V, confinement strength γ, mass m, step size Δt, maximum iteration number M, energy reduction rate $\rho \in (0,1]$
Output: N transformed points $y_1, \dots, y_N \in [0,1]^d$

1: Initialize velocities $v_i = 0$ for $i = 1, \dots, N$
2: Compute the potential U_i in (4) and force field $F_i = -\nabla_{x_i} U_i$ for all points
3: Compute the total potential energy $\mathcal{E}_* = \sum_{\substack{i,j=1 \\ i<j}}^{N} U(x_i, x_j) + \gamma \sum_{i=1}^{N} V(x_i)$
4: Set the mass of each particle to be

$$m = 100 \Delta t^2 N^{\frac{1}{d}-1} \Big(\sum_{i=1}^{N} |F_i|^2\Big)^{1/2}, \tag{5}$$

where $|\cdot|$ denotes the magnitude of a vector
5: Run the velocity Verlet algorithm in Algorithm 1 with an extra step $x = x - \lfloor x \rfloor$ every time after a position is updated. In each time step, compute the total potential energy $\mathcal{E}$ of the system. If $\mathcal{E} < \rho \mathcal{E}_*$, update $\mathcal{E}_* = \mathcal{E}$ and record the locations of points
6: **return** The configuration $y_1, \dots, y_N$ with the lowest energy

Applying the MD-based transformer to random samples yields an efficient way of generating quasirandom points. Compared to low discrepancy sequences in Section 2.2, the new dynamical approach is more flexible as it is not based on any strict formula and allows points to shift in the domain. The new model is less sensitive to parameters due to the continuous framework.

MD-based distribution transformer with random shift. The transformer in Algorithm 2 works well for random input but may not be effective for input distributions with a high level of symmetry, such as the rightmost distribution in Fig. 1b. This is because the forces imposed on each particle may cancel out due to the symmetry and thus the overall force on each particle is extremely small. As a result, directly applying Algorithm 2 may hardly change the distribution. To resolve this issue, inspired by the random scrambling techniques [17,18] for improving low discrepancy sequences, we propose a modified MD-based transformer with *random shift*. The key step is to perform a random shift to the input before applying the MD-based transformer in Algorithm 2. The full scheme is summarized in Algorithm 3. As we will see in Section 6, the random shift version is very effective for improving the quasirandomness of distributions with certain symmetry, for which the basic MD-based transformer in Algorithm 2 does not work well.

Algorithm 3 MD-based distribution transformer with *random shift*.

Input: N points $x_1, \dots, x_N \in [0,1]^d$
Parameters: Random shift size ν, potential U, V, confinement strength γ, mass m, step size Δt, maximum iteration number M, energy reduction rate $\rho \in (0,1]$
Output: N transformed points $y_1, \dots, y_N \in [0,1]^d$

1: Apply random shift to each input point: $\hat{x}_i = x_i + \nu t_i$, where $t_i \in [0,1]$ is random
2: Restrict each point to $[0,1]^d$: $\hat{x}_i = \hat{x}_i - \lfloor \hat{x}_i \rfloor$
3: Apply Algorithm 2 to $\hat{x}_1, \dots, \hat{x}_N$ to obtain the transformed points $y_1, \dots, y_N$
4: **return** $y_1, \dots, y_N$

Choice of parameters The step size Δt is usually a small number compared to the length scale of the domain, for example, 0.0001 to 0.001 for the unit square.

The choice of mass affects how fast each particle moves in each time step and consequently the speed of the transformation. Larger mass yields slower move while smaller mass yields faster and more dramatic change of locations in each time step. Using a large mass may lead to a large number of iterations for the system to attain a good distribution, while a small mass may cause the system to go out of control since the change can be too wild during each iteration. We choose the mass to be given by (5) for the following reason. In each time step, we'd like the shift contributed from forces $\frac{F}{2m}\Delta t^2$ to be comparable to the scale of $O(N^{-1/d})$, which is the scale of spacing for equispaced grid points in $[0,1]^d$. Hence m is chosen to be of order $O(N^{1/d}\overline{|F|}\Delta t^2)$, where $\overline{|F|} = N^{-1}\left(\sum_{i=1}^{N} |F_i|^2\right)^{1/2}$ denotes the average magnitude of force. The coefficient 100 in (5) is not essential and can be replaced by other suitable values. One can also choose a different mass other than the one in (5). Numerical results in Section 6 show that the choice of mass in (5) yields rapid convergence from given random samples to a quasirandom distribution (after only a few time steps). Nevertheless, it is interesting to investigate the impact of mass on the performance of the algorithm as well as the interplay between m and other quantities like $N, \Delta t, M$. A more comprehensive study of the model parameters will be reported in a later date.

The energy reduction rate $\rho \in (0,1]$ affects which configuration the user needs to keep. If $\rho = 1$, then every time we reach a state with a potential energy lower than the current $\mathcal{E}_*$, the state is stored and $\mathcal{E}_*$ is updated. If ρ is small, this means that we only save the state that leads to a noticeable reduction in energy. In Section 6, good test results are obtained with $\rho = 0.99$ and a dramatic decrease in energy is uncommon if the input points are random samples, which spread over the domain with potentially multiple holes and clumps, but are not considered too poorly distributed.

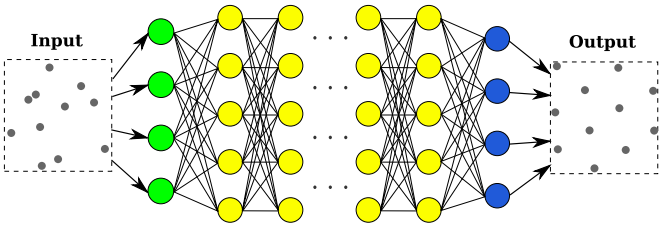


Fig. 3. Deep neural network as a distribution transformer.

Remark 3.1. The two techniques for restricting a point to $[0,1]^d$ - using the floor function and choosing fractional part - are slightly different. To see this, consider a scalar number x. If $x \geq 0$, then $x - \lfloor x \rfloor$ is equal to the fractional part of x. If $x < 0$, for example, $x = -0.1$, then $x - \lfloor x \rfloor = -0.1 - (-1) = 0.9$, while the fractional part of x is 0.1. We choose to use $x - \lfloor x \rfloor$ due to its physical interpretation. Assume that we tile the unit cube $[0,1]^d$ to make a periodic structure that fills the whole space. Then the location of $x - \lfloor x \rfloor$ in $[0,1]^d$ is equal to the relative location of x inside the cube that contain x. For example, if we consider $d = 1$, then the relative location of $x = -0.1$ in $[-1, 0]$ is same as $x - \lfloor x \rfloor = 0.9$ in $[0, 1]$.

Remark 3.2. Note that both the technique $x = x - \lfloor x \rfloor$ in Algorithm 2 and the fractional part selection in lattice rule [13,32, 34] are limited to the special geometry of $[0,1]^d$. They become invalid for dealing with points in other geometries such as spheres or more complex manifolds. The neural network-based approach introduced in Section 3.3 enables a more flexible treatment of different geometries.

3.3. Distribution transformer via DNN

Different from the molecular dynamics approach, we can also completely ignore the kinetic energy of the system and search for a state with low potential energy. This is achieved via parametrizing the desired low-energy state as a neural network and minimizing a loss function representing the total energy. An illustration of the basic model is shown in Fig. 3. The full algorithm is presented in Algorithm 4.

Basic idea. For a point $x \in [0,1]^d$, we represent the corresponding transformed output y as a residual neural network (ResNet) [58] $y = x + G(x;\theta)$, where G is a neural network parametrized by θ. The output y will be determined after the ResNet $x + G(x;\theta)$ is trained to minimize the loss function, defined as the total interaction energy:

$$\mathcal{E}(x_1, \ldots, x_N) = \sum_{\substack{i,j=1 \\ i<j}}^{N} U(x_i, x_j) + \gamma \sum_{i=1}^{N} V(x_i), \tag{6}$$

where $U(x,y)$ denotes the interaction potential, $V(x)$ denotes the confinement potential, and $\gamma \geq 0$ is a parameter for confinement strength.

There are several choices of the interaction potential and the confinement potential. In this manuscript, we consider the following interaction potentials.

$$\text{Coulomb potential:} \quad U(x,y) = \frac{1}{|x-y|}; \quad \text{Yukawa potential:} \quad U(x,y) = \frac{e^{-\mu|x-y|}}{|x-y|}, \tag{7}$$

where the constant μ is the damping factor (also called screening strength) in the screened interaction. If U is chosen as the Coulomb potential, a confinement potential is often needed, i.e. $\gamma > 0$ in (6). If U is chosen as the Yukawa potential, we can choose $\gamma = 0$ as discussed in Section 5.

The confinement potential V is usually in the form of $V(x) = |x-c|^k$, where c is the center of the region and $k > 0$ is an even number. In [59,60], $V(x) = |x-c|^2$; in [61], $V(x) = |x-c|^4$. In general, $V(x)$ is a non-negative function such that the closer x is to the boundary of the domain, the larger $V(x)$ is.

Note that the output of ResNet $y = x + G(x;\theta)$ is *not* guaranteed to stay in the unit cube $[0,1]^d$, so the model can not be used directly since the output y has to be restricted to the unit cube. One simple remedy is to apply a sigmoid activation function σ (whose range is $(0,1)$) to the ResNet as follows:

$$y = \sigma(x + G(x;\theta)). \tag{8}$$

However, this destroys the ResNet architecture (as the identity map can not be recovered in the "short connection" [58]) and will yield unsatisfying results as demonstrated in Section 6. This is consistent with the finding in [58] on the improved stability and faster convergence provided by the ResNet architecture. To circumvent this issue, we propose in the following an unconstrained model.

Unconstrained model. To ensure that the output is always inside the unit cube, we parametrize each point x as $x = \frac{1}{2} + \frac{1}{2}\sin\alpha$, where both '+' and 'sin' are applied componentwisely. Since $x \in [0,1]^d$ is always guaranteed and the mapping is surjective, $\alpha \in \mathbb{R}^d$ is a free variable without any constraint. The energy function in the free latent variables becomes

$$\mathcal{E}_c(\alpha_1, \ldots, \alpha_N) = \sum_{\substack{i,j=1 \\ i<j}}^{N} U(\tfrac{1}{2} + \tfrac{1}{2}\sin\alpha_i, \tfrac{1}{2} + \tfrac{1}{2}\sin\alpha_j) + \gamma \sum_{i=1}^{N} V(\tfrac{1}{2} + \tfrac{1}{2}\sin\alpha_i), \quad \alpha_i \in \mathbb{R}^d. \tag{9}$$

Given $x \in [0,1]^d$, we can compute the free variable as $\alpha = \arcsin(2x-1)$. In order to transform x, it suffices to transform α into β and set $y = \frac{1}{2} + \frac{1}{2}\sin\beta$. We use a ResNet to model the transformation of the free variable α:

$$\beta = \alpha + G(\alpha; \theta). \tag{10}$$

The neural network is trained via minimizing the energy $\mathcal{E}_c(\beta_1, \ldots, \beta_N)$ and the output distribution in $[0,1]^d$ is set to be $y_i = \frac{1}{2} + \frac{1}{2}\sin\beta_i$. The full deep neural network (DNN)-based algorithm for transforming a given distribution is presented in Algorithm 4.

Algorithm 4 Distribution transformer via deep neural network.

Input: N points $x_1, \ldots, x_N \in [0,1]^d$
Output: N transformed points $y_1, \ldots, y_N \in [0,1]^d$
1: Compute $\alpha_i = \arcsin(2x_i - 1)$ for all x_i, where arcsin is applied componentwisely
2: Let $\beta_i = \alpha_i + G(\alpha_i; \theta)$ be the ResNet in (10) parametrized by θ
3: Train the ResNet via minimizing $\mathcal{E}_c(\beta_1, \ldots, \beta_N)$ to obtain θ^* and $\beta_i^* = \alpha_i + G(\alpha_i; \theta^*)$
4: **return** $y_i = \frac{1}{2} + \frac{1}{2}\sin\beta_i^*$ for $i = 1, \ldots, N$

One advantage of DNN-based approach compared to the MD-based approach in Section 3.2 is that it is able to yield highly uniform distributions similar to equispaced grid points. See Section 6 for numerical results. The framework can be applied to dealing with distributions in a general domain as long as the system energy is chosen suitably. In Section 4, we will consider disks, spheres, and manifolds in general.

Remark 3.3. It should be noted that, when training a deep neural network, it is almost impossible in practice to find the exact global minimum, so θ^* in Algorithm 4 is *not* a minimizer in general. In fact, there is *no* need to find the global minimum. As long as the computed solution significantly reduces the energy and improves the distribution, it serves as a desirable output.

Remark 3.4 *(Relationship between the MD and DNN approaches).* The DNN-based approach optimizes the energy function with respect to parameters in the neural network. As a comparison, the MD-based approach can be viewed as optimizing with respect to the output space directly. In fact, the time step Δt in the MD-based approach in Algorithm 2 can be interpreted as the learning rate (or its variant like the square root of the learning rate, up to some multiplicative constants) in gradient descent algorithms; the force field is exactly the negative gradient of the objective - the potential energy; the velocity Verlet algorithm in Algorithm 1 can be viewed as a gradient descent algorithm with momentum, where the updates in Steps 2 and 3 of Algorithm 1 constitute a momentum update. We see that the update in the MD-based approach resembles applying a gradient descent-type algorithm to the energy function over the output space. It should be pointed out that directly applying gradient descent algorithms over the output space is generally invalid because each time, the new update can lie outside the geometry of interest. Hence postprocessing as in Algorithm 2 (Step 5) is needed to restrict the point to the domain of interest. Overall, this approach can be infeasible if the underlying geometry is too complicated to warrant an effective postprocessing step, as mentioned in Remark 3.2. The more flexible DNN-based approach can circumvent this issue and be applied to complicated geometries as discussed in Section 4.

4. Distribution transformers on disks, spheres, general manifolds

Generating distributions on a disk, a sphere or a general manifold has vast applications across various disciplines, from earth models [62,63], structural chemistry [35,64], to computer graphics [65,37], topology optimization [36], etc. In this section, we illustrate the flexibility of the proposed physics-informed framework by transforming distributions on disks, spheres, and manifolds in general. We employ the DNN-based model.

Distribution transformer on the unit disk. To transform points in the unit disk, directly minimizing (6) is not easy to implement since each point has to be restricted to the disk during the training of the neural network. Similar to Algorithm 4, we parametrize each point x in the unit disk using *free* variables α, β as

$$x = \left(\tfrac{1}{2}\cos\beta(1+\sin\alpha), \tfrac{1}{2}\sin\beta(1+\sin\alpha)\right), \quad \alpha, \beta \in \mathbb{R}. \tag{11}$$

The total energy expressed in terms of free variables $\begin{bmatrix}\alpha_i\\ \beta_i\end{bmatrix} \in \mathbb{R}^2$ becomes:

$$\mathcal{E}_d\left(\begin{bmatrix}\alpha_1\\ \beta_1\end{bmatrix},\dots,\begin{bmatrix}\alpha_N\\ \beta_N\end{bmatrix}\right) = \sum_{\substack{i,j=1\\ i<j}}^{N} U(x_i(\alpha_i,\beta_i), x_j(\alpha_j,\beta_j)) + \gamma \sum_{i=1}^{N} V(x_i(\alpha_i,\beta_i)), \tag{12}$$

where $x_i(\alpha_i,\beta_i)$ is the parametrization in (11). The interaction potential U and confinement potential V can be chosen as discussed in Section 3.3.

We use a ResNet architecture to model the transformation of free variables:

$$\begin{bmatrix}\tilde{\alpha}\\ \tilde{\beta}\end{bmatrix} = \begin{bmatrix}\alpha\\ \beta\end{bmatrix} + G(\begin{bmatrix}\alpha\\ \beta\end{bmatrix};\theta). \tag{13}$$

Then we train the neural network via minimizing the corresponding energy $\mathcal{E}_d$. The full algorithm is presented in Algorithm 5.

Algorithm 5 Distribution transformer on the unit disk.

Input: N points $x_1,\dots,x_N$ on the unit disk
Output: N transformed points $y_1,\dots,y_N$ on the unit disk
1: For each x_i, set $\alpha_i = \arcsin(2|x_i|-1)$ and β_i the angle of x_i
2: Let $\begin{bmatrix}\tilde{\alpha}\\ \tilde{\beta}\end{bmatrix}$ be the ResNet in (13) parametrized by θ
3: Train the ResNet via minimizing $\mathcal{E}_d\left(\begin{bmatrix}\tilde{\alpha}_1\\ \tilde{\beta}_1\end{bmatrix},\dots,\begin{bmatrix}\tilde{\alpha}_N\\ \tilde{\beta}_N\end{bmatrix}\right)$ to obtain θ^* and $\begin{bmatrix}\tilde{\alpha}_i^*\\ \tilde{\beta}_i^*\end{bmatrix}$
4: **return** y_i parametrized by $\tilde{\alpha}_i^*, \tilde{\beta}_i^*$ as in (11)

The energy-based framework provides a flexible way to manipulate points and generate distributions according the specific need. Note that modifying γ will generate point distributions with different levels of concentration towards the center of the disk. A larger γ will result in more points near the center while setting $\gamma = 0$ will yield a ground state with more points near the boundary.

Distribution transformer on the unit sphere. The case of a sphere is very different from a cube or a disk because a sphere is a closed surface, i.e., with *no* boundary. Due to this property, there is *no* boundary effect and a confinement potential is *not* needed.

To transform the points on the unit sphere, we parametrize each point using the spherical coordinate

$$x = (\sin\alpha\cos\beta, \sin\alpha\sin\beta, \cos\alpha) \in S^2, \quad \alpha,\beta \in \mathbb{R}. \tag{14}$$

The system energy expressed in terms of the free variables α_i, β_i is

$$\mathcal{E}_s\left(\begin{bmatrix}\alpha_1\\ \beta_1\end{bmatrix},\dots,\begin{bmatrix}\alpha_N\\ \beta_N\end{bmatrix}\right) = \sum_{\substack{i,j=1\\ i<j}}^{N} U(x_i(\alpha_i,\beta_i), x_j(\alpha_j,\beta_j)), \quad \alpha_i,\beta_i \in \mathbb{R}, \tag{15}$$

where $x_i(\alpha_i,\beta_i)$ is the spherical coordinate in (14) and U can be chosen as the Coulomb potential.

Same to (13), we model the transformed free variables $\begin{bmatrix}\tilde{\alpha}\\ \tilde{\beta}\end{bmatrix}$ as a ResNet. The neural network is trained via minimizing $\mathcal{E}_s$ and the transformer is given in Algorithm 6.

Algorithm 6 Distribution transformer on the unit sphere.

Input: N points $x_1,\dots,x_N$ on the unit sphere
Output: N transformed points $y_1 \dots y_N$ on the unit sphere
1: For each $x_i \in S^2$, compute the corresponding angle parameters α_i, β_i
2: Let $\begin{bmatrix}\tilde{\alpha}\\ \tilde{\beta}\end{bmatrix}$ be the ResNet in (13) parametrized by θ
3: Train the ResNet via minimizing $\mathcal{E}_s\left(\begin{bmatrix}\tilde{\alpha}_1\\ \tilde{\beta}_1\end{bmatrix},\dots,\begin{bmatrix}\tilde{\alpha}_N\\ \tilde{\beta}_N\end{bmatrix}\right)$ to obtain θ^* and $\begin{bmatrix}\tilde{\alpha}_i^*\\ \tilde{\beta}_i^*\end{bmatrix}$
4: **return** $y_i \in S^2$ parametrized by $\tilde{\alpha}_i^*, \tilde{\beta}_i^*$ via (14)

Distribution transformer on a manifold. The idea in Algorithm 6 the can be generalized to a general manifold, as summarized in Algorithm 7. Assume a manifold $\Gamma \subset \mathbb{R}^d$ is parametrized by $\psi : [-1,1]^{d-1} \to \Gamma$. We express each point $x \in \Gamma$ as $x = \psi(\sin\alpha)$, $\alpha \in \mathbb{R}^{d-1}$. Then we can define a ResNet similar to (13) and optimize the energy function for a system of given points. The new locations can eventually be obtained as in Algorithm 7.

Algorithm 7 Distribution transformer on a manifold.

Input: N points $x_1 = \psi(t_1), \ldots, x_N = \psi(t_N)$ on the manifold Γ parametrized by $\psi : [-1,1]^{d-1} \to \Gamma$
Output: N transformed points $y_1, \ldots, y_N$ on the manifold
1: For each $t_i \in [-1,1]^{d-1}$, compute angle parameters $\alpha_i \in [-\pi, \pi]^{d-1}$ such that $\sin\alpha_i = t_i$
2: Let $\tilde{\alpha} = \alpha + G(\alpha;\theta) \in \mathbb{R}^{d-1}$ denote the ResNet parametrized by θ
3: Train the ResNet via minimizing $\mathcal{E}_s(\tilde{\alpha}_1, \ldots, \tilde{\alpha}_N)$ to obtain θ^* and $\tilde{\alpha}_i^*$
4: **return** $y_i = \psi(\sin\tilde{\alpha}_i^*) \in \Gamma$

Metrics for general geometries. For points on a general geometry such as a manifold, the definition of discrepancy is no longer valid and metrics *not* limited to a certain geometry are needed to measure the uniformity of a set $X = \{x_1, \ldots, x_N\}$. The inverse minimal pairwise distance $\frac{1}{\min\limits_{i\neq j}|x_i - x_j|}$ is a simple metric valid for arbitrary geometries. It has been used as a metric in meshless methods to avoid the instability for discretizing and solving partial differential equations (cf. [66]). A relatively small value indicates good spacing between points. A drawback of this metric is that it depends on the minimal distance between two points and the same value can correspond to drastically different configurations with arbitrarily many pairs of points achieving the minimal distance. Different from geometric metrics, another metric we can use to indicate uniformity is the low-rank approximation error to certain kernel matrix. Recent study [7,67] shows that points with good uniformity will benefit the low-rank approximation to kernel matrices and clustered points in small regions impede not only accuracy but also numerical stability in low-rank approximations. Inspired by this result, we use X to compute low-rank factors to the kernel matrix and then use the approximation error as a metric to indicate the uniformity of X. Formally, we compute the low rank approximation error for a large kernel matrix $K_{\Omega\Omega} = [\kappa(p,q)]_{p,q\in\Omega}$ associated with a smooth symmetric kernel $\kappa(x,y)$ and a set $\Omega = \{q_i\}_{i=1}^M$ with $M \gg N$. The artificial set Ω is created such that it contains X and a large number of points randomly distributed on the underlying geometry. Examples of the kernel κ are shown in (21). The metric for X is defined as

$$||K_{\Omega\Omega} - K_{\Omega X} K_{XX}^{+} K_{\Omega X}^{T}||/||K_{\Omega\Omega}||, \tag{16}$$

where A^+ denotes the pseudoinverse of matrix A, $||\cdot||$ denotes the 2-norm, and $K_{XY} := [\kappa(x,y)]_{x\in X, y\in Y}$. It is expected that, a set X with better uniformity gives a smaller error. Experiments in [7] and Section 6 show that this algebraic metric is able to distinguish good distributions from bad ones (with poor uniformity). Compared to the concept of discrepancy, the metrics above work in a much more general setting and can be computed easily. We would like to emphasize that it is still unknown which metric works best or should be used for measuring uniformity of a set on a general geometry.

Remark 4.1. Distributing points on a sphere is a classical problem with diverse interests in chemistry, spherical design, physics [35,68–70]. The problem of minimizing the total Coulomb potential over a set of points $x_1, \ldots, x_N$ on the unit sphere is known as the *Thomson problem* [69]. The optimal set of points that minimize the energy is referred as *Fekete points* [71]. The Coulomb interaction $\frac{1}{|x_i - x_j|}$ can be replaced with the more general Riesz s-energy $\frac{1}{|x_i - x_j|^s}$. The Thomson problem is still an open problem [72]. For some special values of N, the solution is known to be evenly distributed on the sphere. For example, when $N = 4$, the convex hull of the optimal distribution of points is a tetrahedron [68]; when $N = 12$, the convex hull is a icosahedron [73].

Remark 4.2. For simple geometries like a cube or a ball, we can first generate evenly distributed points in the bounding box and then remove points outside the geometry. However, this "generate-and-remove" approach does not work for more general geometries such as manifolds (a sphere, for example). It is easy to see that, if the underlying geometry is a manifold, points evenly spread the bounding box are unlikely to lie exactly on the manifold. Another issue is that it may not be straightforward or computationally efficient to check whether each point belongs to the underlying geometry. On the contrary, the DNN-based approach is powerful enough to handle non-cube geometries, as can be seen from various experiments in Section 6.

5. Interaction energy

In this section, we consider the Coulomb and Yukawa potentials and discuss the interplay between the total energy and the distribution of points.

Coulomb interaction The Coulomb potential between two particles is inverse proportional to their distance. For a system of electrons in a bounded region, if the total energy is defined as the sum of all Coulomb interactions, then the equilibrium distribution is *not* the equispaced distribution, as can be seen from the proposition below for the one-dimensional case.

Proposition 1. *Let* $\mathcal{E}(x_1,\dots,x_N)=\sum_{\substack{i,j=1\\ i<j}}^{N}\frac{1}{|x_i-x_j|}$ *be the Coulomb interaction energy of N points in* $[0,1]$. *Then the set of N equispaced points* $x_i^*=(i-1)h$ *with* $h:=\frac{1}{N-1}$ *and* $1\le i\le N$ *does not minimize* $\mathcal{E}$.

Proof. For any N points $x_1<x_2<\cdots<x_N$, the partial derivative of $\mathcal{E}$ is given by

$$\frac{\partial\mathcal{E}}{\partial x_i}=-\sum_{l=1}^{i-1}\frac{1}{|x_i-x_l|^2}+\sum_{l=i+1}^{N}\frac{1}{|x_i-x_l|^2}.$$

At the equispaced points $(x_1^*,\dots,x_N^*)$, the partial derivative with respect to x_2 is

$$\frac{\partial\mathcal{E}}{\partial x_2}=-\frac{1}{h^2}+\sum_{l=3}^{N}\frac{1}{(l-2)^2h^2}=\sum_{k=2}^{N-2}\frac{1}{k^2h^2}\ge\frac{1}{4h^2}.$$

We see that the partial derivative is large (assuming N is large), so $\mathcal{E}$ is far from being minimal when the N points are equispaced.

The result above indicates that, in order to obtain evenly distributed distributions, a confinement potential needs to be included in the total energy, as in (6). More precisely, we see from the proof of Proposition 1 that for the equispaced distribution, particles located near the boundary experience large Coulomb forces while those near the center are relatively stable due to the cancellation of forces from opposite directions. Hence it is impossible to obtain a nearly uniform distribution at equilibrium. To resolve the boundary effect, a confinement potential V is needed in practice, as in (6). It is used to create a relatively large energy when the point is close to the boundary.

Yukawa interaction The Yukawa interaction is a screened Coulomb interaction where the long range interaction can be suppressed by the damping factor. Consequently, the boundary effect may not be noticeable and the confinement potential may not be necessary. A quantitative description is given in the proposition below for Yukawa interaction.

Proposition 2. *Let* $\mathcal{E}(x_1,\dots,x_N)=\sum_{\substack{i,j=1\\ i<j}}^{N}K(x_i,x_j)$ *be the total interaction energy of N points in* $[0,1]$ *associated with interaction* $K(\cdot,\cdot)$. *Define* $h:=\frac{1}{N-1}$ *and*

$$\mu:=-\frac{3}{2}h^{-1}\ln h. \tag{17}$$

Then at the equispaced points $x_i^*=(i-1)h$,

$$\frac{\partial\mathcal{E}}{\partial x_i}(x_1^*,\dots,x_N^*)\to 0\quad as\quad N\to\infty,\quad i=2,\dots,N-1.$$

Proof. Consider the Yukawa interaction $K(x,y)=\frac{e^{-\mu|x-y|}}{|x-y|}$. Then it can be computed that

$$\frac{\partial\mathcal{E}}{\partial x_i}=\sum_{l=1}^{i-1}\left(-\frac{e^{-\mu r_{il}}}{r_{il}^2}-\frac{\mu e^{-\mu r_{il}}}{r_{il}}\right)+\sum_{l=i+1}^{N}\left(\frac{e^{-\mu r_{il}}}{r_{il}^2}+\frac{\mu e^{-\mu r_{il}}}{r_{il}}\right),$$

where $r_{il}:=|x_i-x_l|$. At the equispaced points $(x_1^*,\dots,x_N^*)$, we deduce for $2\le i\le N-1$ that,

$$\left|\frac{\partial\mathcal{E}}{\partial x_i}\right|\le\left|\frac{\partial\mathcal{E}}{\partial x_2}\right|=\sum_{k=2}^{N-2}\left(\frac{e^{-\mu kh}}{k^2h^2}+\frac{\mu e^{-\mu kh}}{kh}\right)\le\frac{e^{-2\mu h}}{h^2}\sum_{k=2}^{N-2}\frac{1}{k^2}+\frac{\mu e^{-2\mu h}}{h}\sum_{k=2}^{N-2}\frac{1}{k}. \tag{18}$$

Since in this case $\mu=-\frac{3}{2}h^{-1}\ln h$, we compute that

$$\frac{e^{-2\mu h}}{h^2}=h=\frac{1}{N-1}\quad\text{and}\quad\frac{\mu e^{-2\mu h}}{h}=-\frac{3}{2}h\ln h=\frac{3\ln(N-1)}{2(N-1)}.$$

Plug it in to (18), we arrive at

$$\left|\frac{\partial\mathcal{E}}{\partial x_i}\right|\le\frac{1}{N-1}\sum_{k=2}^{N-2}\frac{1}{k^2}+\frac{2\ln(N-1)}{N-1}\sum_{k=2}^{N-2}\frac{1}{k}<\frac{2}{N-1}+\frac{2\ln(N-1)(\ln N+1)}{N-1}, \tag{19}$$

where we have used the fact that $\sum_{k=1}^{\infty} \frac{1}{k^2} = \frac{\pi^2}{6} < 2$ and $\sum_{k=1}^{N} \frac{1}{k} < \ln N + 1$. It follows immediately from (19) that $\frac{\partial \mathcal{E}}{\partial x_i} \to 0$ as $N \to \infty$.

From the proof above, it can be seen that, the sum in either (18) is in fact dominated by the first term due to the super-exponential decay of the sequence with respect to k. Therefore, in order to bound the sum, it suffices to bound the first term. This provides a guidance for choosing μ in higher dimensions and the formula (17) still applies with properly defined h. For a system of N particles in $[0,1]^d$, we may choose μ as in (17) with $h := O(N^{-1/d})$. That is $\mu = O(\frac{1}{d} N^{1/d} \log N)$

6. Experiments

In this section, we perform numerical experiments to investigate the proposed physics-informed distribution transformers: MD-based one and DNN-based one. The implementation details are presented in Section 6.1. Test results are shown in Section 6.2. The deep neural network is implemented with PyTorch [74,75].

6.1. Implementation details

MD-based distribution transformer For the MD-based transformer, we use the Coulomb interaction $U(x,y) = \frac{1}{|x-y|}$ and the quadratic confinement potential $V(x) = |x-c|^2$. The parameters are chosen as follows: time step $\Delta t = 0.0002$, mass is defined in (5), maximum iteration number $M = 20$, energy reduction rate $\rho = 0.99$.

DNN-based distribution transformer For all DNN-based transformers, we use Adam [55] as the optimizer with beta parameters $(0.5, 0.9)$ and epsilon parameter 10^{-6} (cf. [55]). The learning rate is set to be 0.001. The activation function is chosen as Leaky ReLU [76]. The loss function is defined according to the geometry and the potential used, which will be detailed in each experiment. For the Yukawa potential, the damping factor is chosen as follows according to the discussion in Section 5.

$$\mu = d^{-1} N^{1/d} \log N. \tag{20}$$

6.2. Test results

In this section, we present several experiments to investigate the performance of the proposed distribution transformers for improving the uniformity of a given distribution. For Tests 1,2,3, to evaluate the uniformity quantitatively, we compute the L_2 star discrepancy using Warnock's formula in (1). The inverse minimum pairwise distance is used in Test 5. For *non-cube* geometries as in Tests 6,7,8, we use the kernel matrix approximation error described in (16) to measure the uniformity of a configuration quantitatively.

Test 1. DNN-based transformer with unconstrained model. In this test, we compare the performance of DNN-based transformers based on two models, one in (8) with original variable restricted to $[0,1]^d$ and one in (10) with unconstrained latent variable. We employ the Yukawa potential and define the loss function as the sum of all pairwise Yukawa interactions with damping factor in (20). In the neural network, three hidden layers with dimensions 8, 16, 8 are used. The maximum number of epochs is set to be 10000. For the given $N = 100$ random samples, we see from Fig. 4 (top row) that the model in (8) with artificial restriction fails to yield a good distribution within 10000 epochs, while the unconstrained model in (10) produces highly evenly-spaced points in Fig. 4 (bottom-right). It can be seen that a decent distribution is obtained after 2000 epochs using the model in (10). Quantitatively, the corresponding discrepancy plot is shown in Fig. 5. The decay of discrepancy during the training of neural network is easily seen from the plot.

Test 2. Transforming random samples with MD-based transformer. In this test, we use molecular dynamics (MD)-based transformer in Algorithm 2 to improve the distribution of $N = 300$ random samples in the unit square. The model parameters are described in Section 6.1. The energy function is the total Coulomb potential plus confinement potential with confinement strength $\gamma = 10N$. The results are shown in Fig. 6, where we plot the force field (blue arrow) on each particle in the 8 interior subfigures, corresponding to the iteration steps $0, 1, 2, 3, 4, 7, 10, 16$ in the velocity Verlet algorithm. The plot of the corresponding discrepancies is also shown. The length of each blue arrow in Fig. 6 indicates the strength of the force field. We see that the particles move in the direction to create better uniformity. Furthermore, it only takes less than 20 iterations to transform the random input in Fig. 6 to an output with better uniformity.

Test 3. Improving low discrepancy sequence with MD-based transformer (random shift version). We see from Fig. 1b (also 1st plot in Fig. 7) that the first 250 points from the Halton sequence with bases 11,13 do not display enough quasirandomness due to the strong correlation between points. In this test, we apply the MD-based transformer with random shift in Algorithm 3 to the Halton points in order to obtain better quasirandomness. The random shift is chosen as 0.1 and other parameters are the same as in Test 2. We plot the distributions at time steps $1, 6, 17$ in Fig. 7 along with the output and the discrepancies corresponding to those distributions. We see that the output points achieve better quasirandomness than the input distribution, with reduced discrepancy. Also, the random shift does help to destroy the symmetric structure in the input distribution and contributes to the quasirandomness obtained by Algorithm 3. As a comparison to permutation

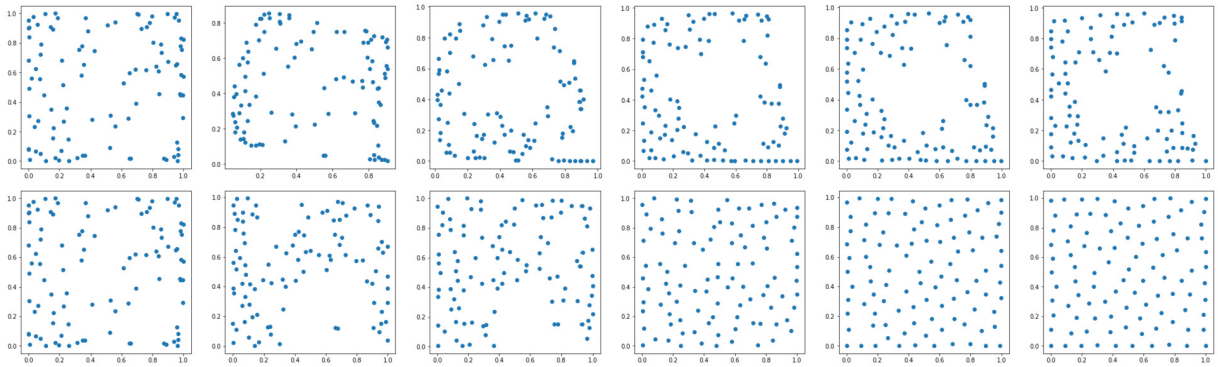


Fig. 4. Test 1: Restricted model (top) in (8) and unconstrained model (bottom) in (10). Left to right: input distribution, transformed distributions after 500, 1000, 2000, 5000, 10000 epochs.

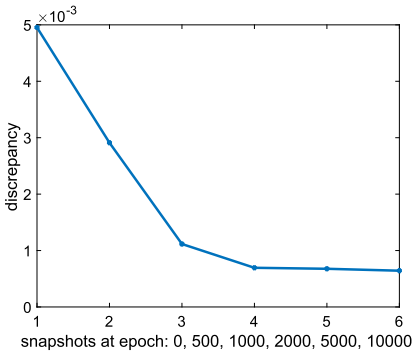


Fig. 5. Test 1: discrepancy plot for distributions in Fig. 4(bottom).

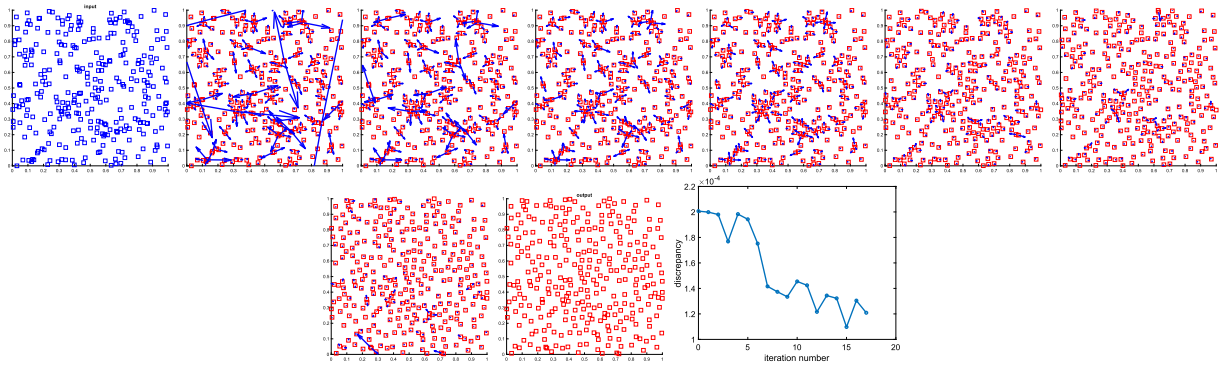


Fig. 6. Test 2: MD-based transformation of random samples. Top-left to bottom-right: input, force fields of the system at time steps 0, 1, 2, 3, 7, 10, 16, output, discrepancy plot. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

or scrambling techniques, in Fig. 8, we include the output obtained by the reverse permutation [19]. It is easy to see from Fig. 8 that the permutation method does *not* resolve the high correlations between points as the MD-based method does. In general, though number-theory based formulas are easy to compute, they are limited to special input sets and meanwhile can not be applied to *non-cube* geometries. These limitations motivate the development of the methods in this manuscript to handle more general input sets and geometries.

Test 4. Transforming points in a disk. We apply Algorithm 5 to transforming $N = 100$ given points in the unit disk in Fig. 9 (left). The energy is chosen as Coulomb interaction plus the confinement potential $V(x) = |x|^2$ with confinement strength $\gamma = 3N$. In the neural network, four hidden layers with dimensions 4, 16, 32, 8 are used and the maximum number of epochs is set to be 50000. Note that since the input distribution in Fig. 9 is extremely poorly distributed in the disk, to obtain a good transformation, we expect a larger neural network and more epochs compared to Test 1. It is easy to see from Fig. 9 that the DNN-based transformer is able to yield a point distribution with extremely high uniformity despite

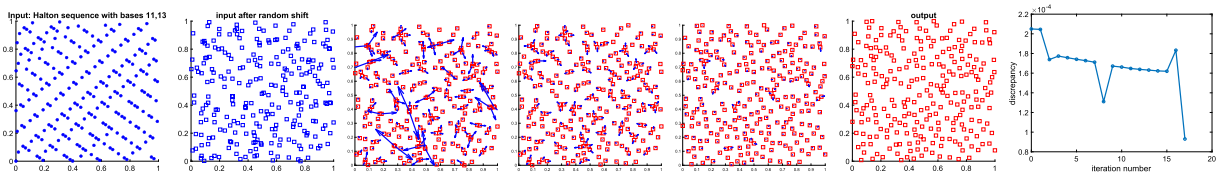


Fig. 7. Test 3: Transforming 250 Halton points with bases 11, 13. Left to right: input, randomly shifted input, system with force fields at time steps 1, 6, 17, output, discrepancy plot.

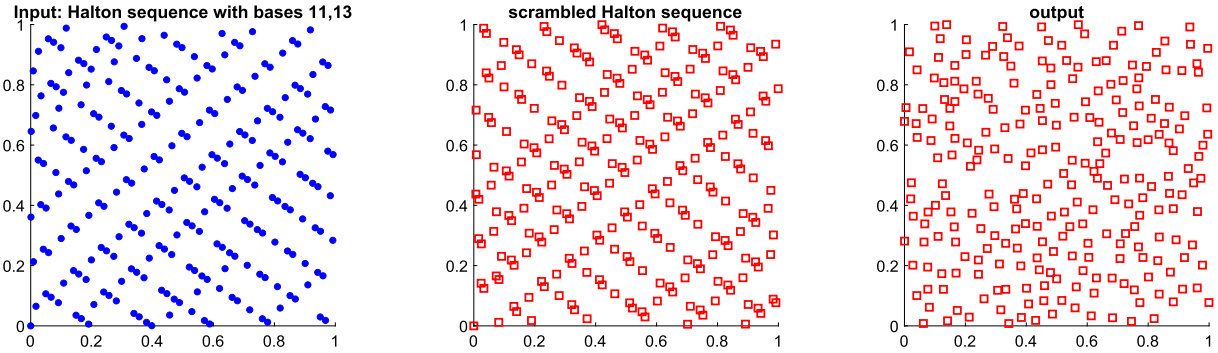


Fig. 8. Comparison of the number theoretic approach and the proposed approach for Test 3. Left: input Halton set; Middle: output of reverse permutation [19]; Right: output of MD-based approach.

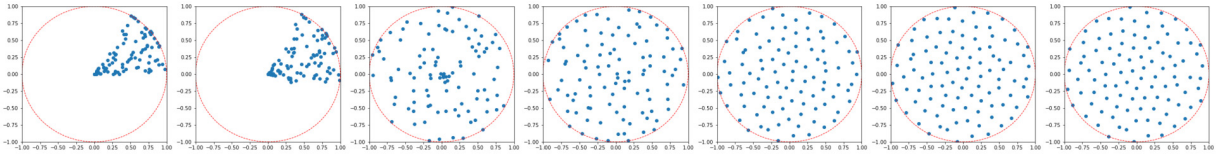


Fig. 9. Test 4: Transforming points in a disk. Left to right: input distribution, transformed distributions after 500, 1000, 2000, 10000, 30000, 50000 epochs.

the fact that input points are concentrated in a small region, namely, 1/6 of the domain. This demonstrates the potential of DNN-based transformers to generate highly structured points with good uniformity.

Test 5. Transforming points on a sphere. We apply Algorithm 6 to transforming $N = 100$ given points on the unit sphere in Fig. 10 (top-left). The energy is chosen as the total pairwise Coulomb interactions. The neural network contains four hidden layers with dimensions 4, 8, 16, 8. The maximum number of epochs is set to be 50000. Fig. 10 shows the distributions during transformation, where the color of each point is measured by the value of its z component in the Cartesian coordinate. We see that the input points (top-left plot) are poorly distributed on the sphere and the output distribution (bottom-right plot) is much better. To illustrate the uniformity quantitatively, we compute the inverse minimal pairwise distances $1/\min_{i\neq j}|x_i - x_j|$ corresponding to the 7 distributions in Fig. 10 and plot the curve in Fig. 10(bottom-right). We see that the minimal pairwise distance of the output distribution is significantly larger than the input distribution, which indicates that the points do become more evenly spaced after the transformation.

Test 6. Transforming points on a complex manifold in $\mathbb{R}^3$. We apply Algorithm 7 to 100 points on a complex flower-shaped manifold in $\mathbb{R}^3$. See Fig. 11 (top left) for an illustration of the manifold. As can be seen from the second figure (top row) in Fig. 11, a majority of the points are clustered at the bottom of the manifold. To transform the points, we use the same neural network architecture as in Test 5. The metric for measuring uniformity is chosen as the kernel matrix approximation error as described in (16), where four different kernel functions - Gaussian, multiquadric, inverse quadratic, inverse multiquadric - are tested:

$$e^{-0.2||x-y||^2},\quad \sqrt{1+0.2||x-y||^2},\quad \frac{1}{1+0.1||x-y||^2},\quad \frac{1}{\sqrt{1+0.1||x-y||^2}}. \tag{21}$$

The results for the transformation and the metric are shown in Fig. 11. Visually, it can be seen from the scatter plots in Fig. 11 that during the training, the points clustered at the bottom of the manifold are transformed to disperse over the entire manifold. Quantitatively, from Fig. 11 (bottom), we see that the matrix approximation error decays effectively, which reflects the improvement of the uniformity of the distribution.

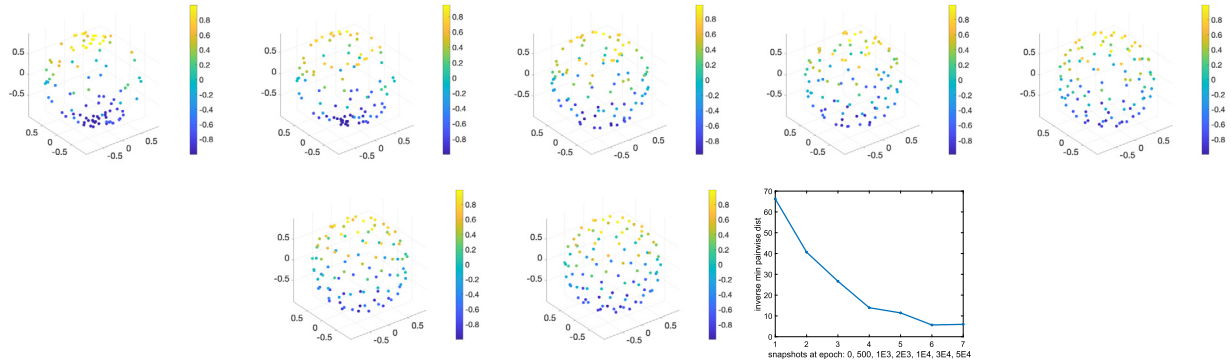


Fig. 10. Test 5: Transforming points on a sphere. Top-left to bottom-right: input, distributions after 500, 1000, 2000, 10000, 30000, 50000 epochs, inverse minimum pairwise distance plot.

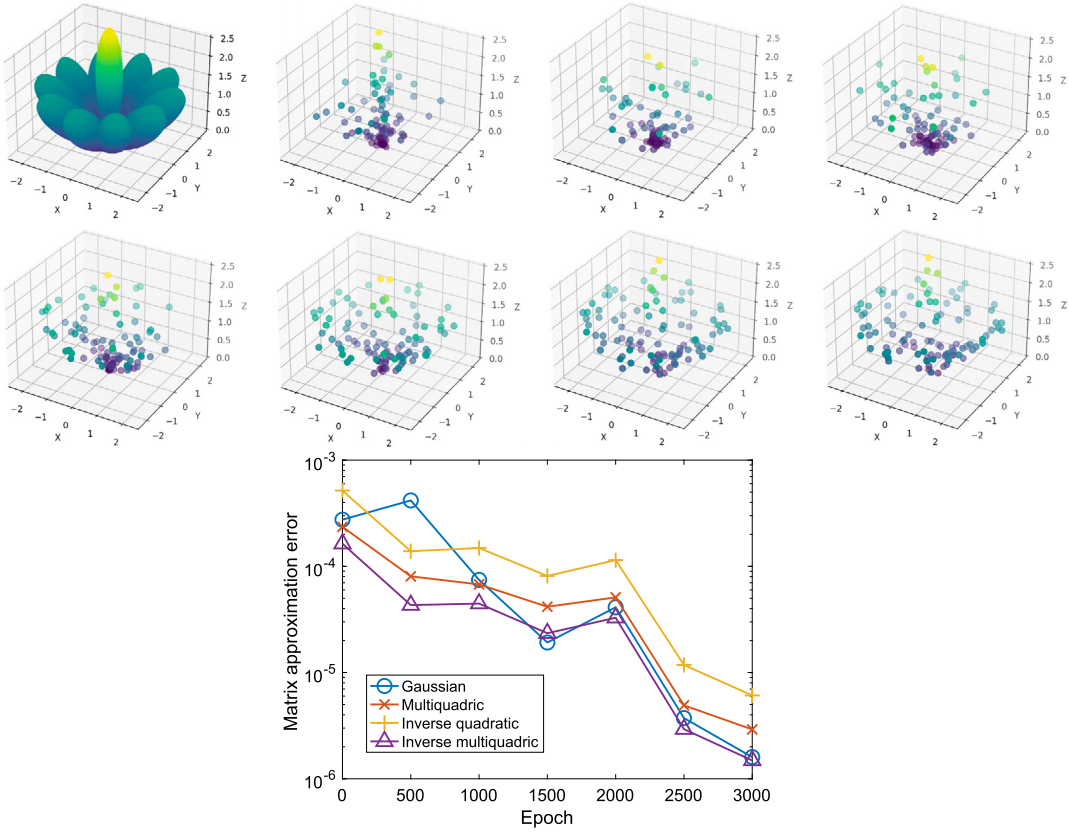


Fig. 11. Test 6: Transforming points on a complex manifold in $\mathbb{R}^3$. Top left to bottom right: geometry, input points, points at epoch=500k($k = 1, 2, \ldots, 6$). Bottom: matrix approximation errors for kernels in (21) vs epoch.

Test 7. Transforming points on a hypersphere in $\mathbb{R}^4$. We apply Algorithm 7 (with a different interval for parameters) to 100 poorly distributed points on the unit hypersphere S^3 in $\mathbb{R}^4$:

$$(\cos u \cos v \cos w,\ \cos u \cos v \sin w,\ \cos u \sin v,\ \sin u) \in \mathbb{R}^4.$$

The same neural network architecture as in Test 6 is used. The input points are generated with u sampled from $\mathcal{U}(-\frac{\pi}{2}, \frac{\pi}{2})$ and v, w sampled from $\mathcal{U}(0, 0.2\pi)$, where $\mathcal{U}(a, b)$ denotes the uniform distribution on $[a, b]$. This is to make most points cluster in a small region on the hypersphere. The kernel matrix approximation error in (16) is used as the metric. The results for the experiment are shown in Fig. 12 and Fig. 13. Fig. 12 shows the matrix approximation errors associated with points on the hypersphere in $\mathbb{R}^4$ and the projections into three dimensions. Fig. 13 presents visualizations of the three dimensional projections: (1,2,3) and (2,3,4) during the transformation. The decay of errors in Fig. 12 reflects the improved

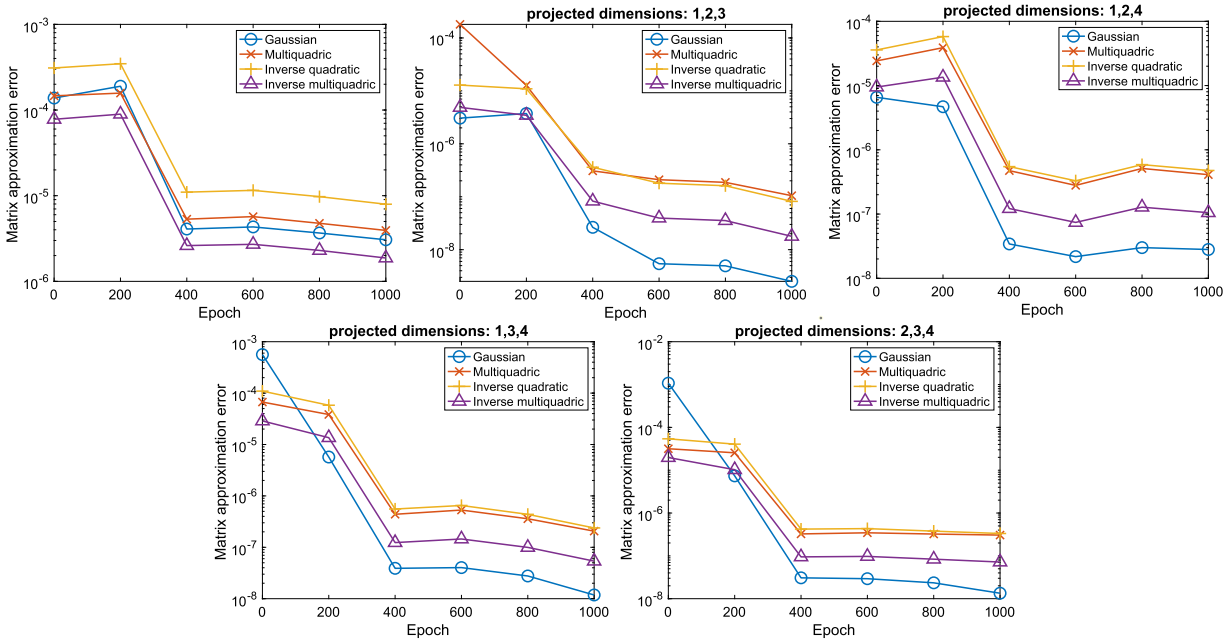


Fig. 12. Test 7: Matrix approximation error vs epoch. Top left to bottom right: points on the hypersphere in $\mathbb{R}^4$, projections to dimensions (1,2,3), (1,2,4), (1,3,4), (2,3,4).

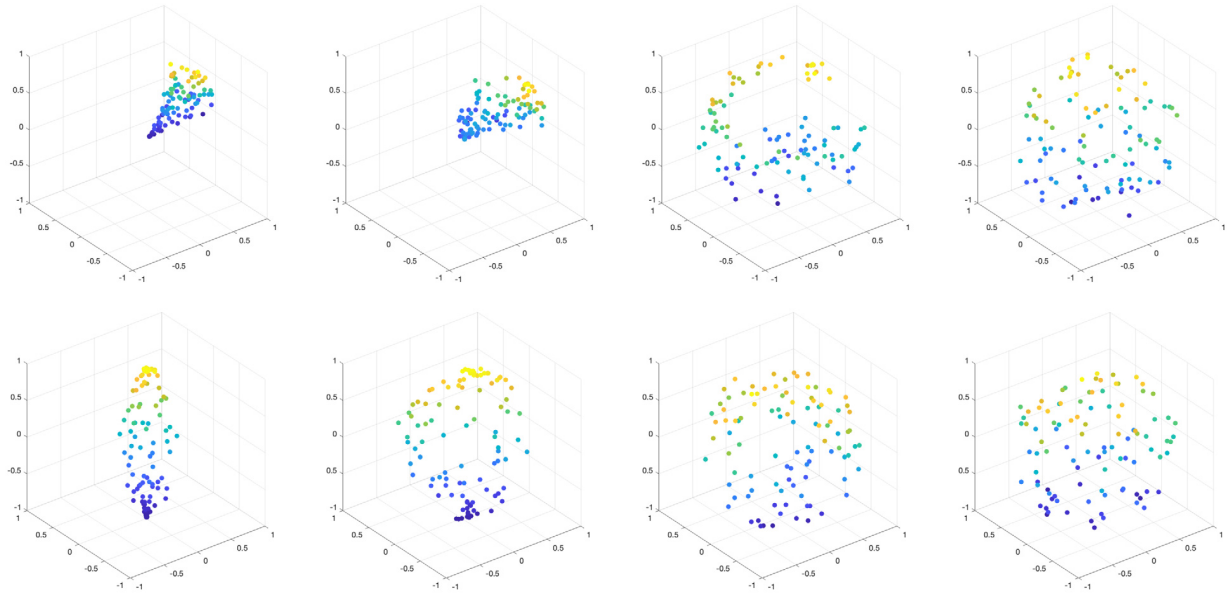


Fig. 13. Test 7: Each row shows 3D projections of 4D points on a hypersphere at epoch=0,200,400,1000. Top: dimension (1,2,3); Bottom: dimension (2,3,4). Quantitative results are shown in Fig. 12.

uniformity of the input after the transformation. We see that the improvement is not only reflected in the original four dimensional space but also observed in each of the three dimensional projections. It can also be seen that only around 500 epochs are needed to refine the distribution substantially. The experiment shows that the proposed distribution transformer also works for high dimensional geometry.

Test 8. Transforming samples from a distribution. We apply Algorithm 7 (with a different interval for parameters) to learn a distribution transformer applied to batches of 100 samples from the following distribution on the unit sphere:

$$x = (\sin(\theta/2)\cos(\phi), \sin(\theta/2)\sin(\phi), \cos(\theta/2)),$$

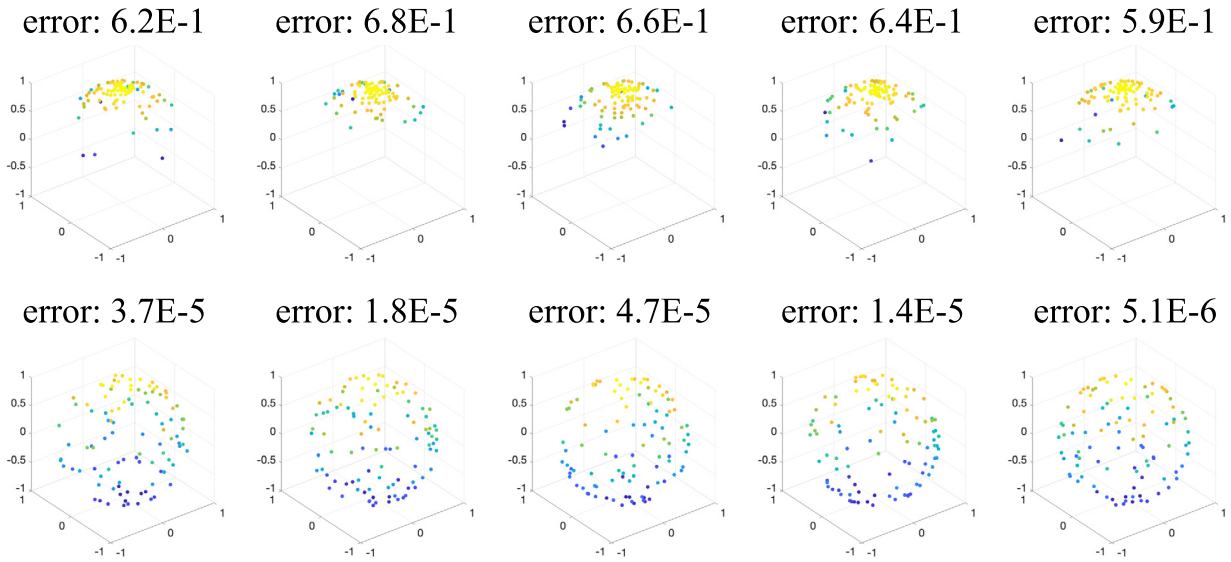


Fig. 14. Test 8: Transforming five sets of samples clustered around the north pole on the sphere: input(top) and output(bottom). The title for each plot is the metric in (16) for the configuration with kernel $\kappa(x, y) = e^{-||x-y||^2}$. Significantly improved approximation accuracy (from 1 digit to 5 or 6 digits) reflects improved uniformity of points.

where $\theta \sim \mathcal{N}(0, 1)$ is the standard normal distribution and $\phi \sim \mathcal{U}(0, 2\pi)$ is the uniform distribution on $[0, 2\pi]$. Samples from this distribution are more likely to cluster at the north pole, where the probability density peaks. We use the algebraic metric in (16) to measure the uniformity and the kernel function is chosen as $\kappa(x, y) = e^{-||x-y||^2}$. Fig. 14 presents the results of transforming five sets of input samples (top row) and to more evenly distributed outputs (bottom row). The metric (low-rank approximation error) is shown as the title in each configuration plot. A substantially smaller approximation error indicates better uniformity of points. From the numeric values in Fig. 14, it can be seen that, each poorly distributed input only gives one digit of accuracy at most, while the corresponding output can achieve five to six digits of approximation accuracy, reflecting the much improved uniformity.

7. Discussion

7.1. Comparison of MD-based and DNN-based transformers

The MD-based transformer is given by an explicit update formula, which is easy to compute. When applied to random samples or points from a low discrepancy sequence, it can quickly improves the quasirandomness. However, it is not able to achieve extremely high uniformity as DNN-based transformers do and can not be extended easily to other geometries. The DNN-based transformer is able to produce distributions with superior uniformity regardless of the input distribution. It can be easily extended to different geometries as long as an unconstrained parametrization is available. However, there are several drawbacks. The DNN-based approach requires solving a nonconvex minimization problem and the training of the deep neural network is more time-consuming than running MD simulations.

Overall, the MD-based transformer can be used as a fine-tuner to quickly improve the uniformity or quasirandomness of a given distribution. If the given distribution is far from being evenly distributed and high uniformity is needed, then the DNN-based transformer is a better option. The DNN-based transformer also works for disks and manifolds. Though the basic residual network (ResNet) is used in the manuscript, other neural network architectures can also be incorporated into the DNN-based transformer. Using the state-of-the-art optimization tools may also improve the performance of the proposed transformer.

7.2. Issues and future work

Quadratic complexity The proposed framework requires computing all pairwise interactions between particles and thus the computation cost has $O(N^2)$ complexity for a system of N particles, which is not optimal. The quadratic cost can be circumvented by using fast multipole method or hierarchical matrices (cf. [77–82,67]), which employ a hierarchically low-rank representation to approximate the dense kernel matrix and yield $O(N)$ complexity in time and space.

Training of neural network The potential numerical instability in training deep neural networks has long been a challenging problem in deep learning [51,52,55]. To transform a large number of points with DNN-based transformer, a large neural

network is needed in order to model the complicated system. Deeper and wider hidden layers may cause numerical stability issues during the training stage. How to stabilize the training is a major problem to investigate in order to apply the proposed method to large-scale datasets.

Neural network architectures In the manuscript, we use a residual neural network (ResNet) to model the transformation. It will be our future work to explore appropriate architectures for the distribution transformer leveraging the state-of-the-art development in neural networks. For example, for generating configurations with good uniformity, generative models such as normalizing flows [83–88] appear promising due to its power in modeling complex mappings between distributions.

Choice of energy for other distributions and geometries The choice of energy dictates the final output distribution of the proposed transformers. It is interesting to study which energy should be used to obtain a desired point distribution (not necessarily uniform) over a specific geometry. This will greatly expand the scope of applications of the proposed approach, for example, to numerical simulations in fluid, mechanical, electromagnetic problems where adaptive discretization (non-uniform mesh) is often needed in order to resolve singular behaviors of solutions (cf. [89–93]).

8. Conclusion

In this manuscript, we introduce a physics-informed framework for improving given distributions. As an initial attempt to leverage physics principles and deep neural networks for improving the quasirandomness of a given distribution, two kinds of distribution transformers are introduced: one based on molecular dynamics (MD), another based on deep neural networks (DNN). The MD-based transformer serves as an efficient fine-tuner which can quickly improve the uniformity of random samples or the quasirandomness of low discrepancy sequence. The DNN-based transformer is more powerful as it is able to achieve superior uniformity in different geometries regardless of the given distribution. Various experiments are presented to demonstrate the quality of the proposed transformers on improving the given possibly poorly distributed points over different geometries. Compared to existing methods, the new approach provides a flexible framework that allows dealing with general point distributions instead of points generated by a certain formula. Future work includes improving the computational efficiency via incorporating hierarchical matrices, leveraging the state-of-the-art neural network architecture to investigate more general geometries.

CRediT authorship contribution statement

Difeng Cai: Conceptualization, Formal analysis, Investigation, Methodology, Resources, Software, Validation, Visualization, Writing – original draft, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

The research is supported by NSF grants OAC-2003720 and DMS-2038118. The author would like to thank anonymous referees for valuable suggestions.

References

[1] H. Niederreiter, Random Number Generation and Quasi-Monte Carlo Methods, SIAM, 1992.
[2] W.J. Morokoff, R.E. Caflisch, Quasi-random sequences and their discrepancies, SIAM J. Sci. Comput. 15 (6) (1994) 1251–1279.
[3] D. Dobkin, D. Eppstein, D. Mitchell, Computing the discrepancy with applications to supersampling patterns, ACM Trans. Graph. 15 (4) (1996) 354–376.
[4] P. Christensen, A. Kensler, C. Kilpatrick, Progressive Multi-Jittered Sample Sequences, Computer Graphics Forum, vol. 37, Wiley Online Library, 2018, pp. 21–33.
[5] H. Perrier, D. Coeurjolly, F. Xie, M. Pharr, P. Hanrahan, V. Ostromoukhov, Sequences with Low-Discrepancy Blue-Noise 2-D Projections, Computer Graphics Forum, vol. 37, Wiley Online Library, 2018, pp. 339–353.
[6] J. Cheng, M. Druzdzel, Computational investigation of low-discrepancy sequences in simulation algorithms for Bayesian networks, arXiv:1301.3841, 2013.
[7] D. Cai, J. Nagy, Y. Xi, Fast deterministic approximation of symmetric indefinite kernel matrices with high dimensional datasets, SIAM J. Matrix Anal. Appl. 43 (2) (2022) 1003–1028.
[8] J.F. Koksma, A general theorem from the theory of uniform distribution modulo 1, Math. B (Zutphen) 11 (1942) 7–11.
[9] H. Niederreiter, Point sets and sequences with small discrepancy, Monatshefte Math. 104 (4) (1987) 273–337.
[10] J. Dick, F. Pillichshammer, Digital Nets and Sequences: Discrepancy Theory and Quasi–Monte Carlo Integration, Cambridge University Press, 2010.
[11] J.H. Halton, Algorithm 247: radical-inverse quasi-random point sequence, Commun. ACM 7 (12) (1964) 701–702.
[12] I.M. Sobol, Multidimensional Quadrature Formulas and Haar Functions, Izdat. Nauka, Moscow, 1969.
[13] I. Sloan, Lattice methods for multiple integration, J. Comput. Appl. Math. 12 (1985) 131–143.
[14] I. Sloan, et al., Lattice Methods for Multiple Integration, Oxford University Press, 1994.

[15] H. Faure, Good permutations for extreme discrepancy, J. Number Theory 42 (1) (1992) 47–56.
[16] A. Owen, Quasi-Monte Carlo Sampling, Monte Carlo Ray Tracing: Siggraph, vol. 1, 2003, pp. 69–88.
[17] A. Owen, Randomly permuted (t, m, s)-nets and (t, s)-sequences, in: Monte Carlo and Quasi-Monte Carlo Methods in Scientific Computing, Springer, 1995, pp. 299–317.
[18] L. Kocis, W. Whiten, Computational investigations of low-discrepancy sequences, ACM Trans. Math. Softw. 23 (2) (1997) 266–294.
[19] B. Vandewoestyne, R. Cools, Good permutations for deterministic scrambled Halton sequences in terms of L2-discrepancy, J. Comput. Appl. Math. 189 (1–2) (2006) 341–361.
[20] M. Karplus, G.A. Petsko, Molecular dynamics simulations in biology, Nature 347 (6294) (1990) 631–639.
[21] T.R. Cuthbert, N.J. Wagner, M.E. Paulaitis, G. Murgia, B. D'Aguanno, Molecular dynamics simulation of penetrant diffusion in amorphous polypropylene: diffusion mechanisms and simulation size effects, Macromolecules 32 (15) (1999) 5017–5028.
[22] A. Genreith-Schriever, R. De Souza, Field-enhanced ion transport in solids: reexamination with molecular dynamics simulations, Phys. Rev. B 94 (22) (2016) 224304.
[23] E.C. Neyts, P. Brault, Molecular dynamics simulations for plasma-surface interactions, Plasma Process. Polym. 14 (1–2) (2017) 1600145.
[24] O. Lindblom, T. Ahlgren, K. Heinola, Molecular dynamics simulations of hydrogen isotope exchange in tungsten vacancies, Nucl. Mater. Energy 29 (2021) 101099.
[25] S. Rampino, Configuration-space sampling in potential energy surface fitting: a space-reduced bond-order grid approach, J. Phys. Chem. A 120 (27) (2016) 4683–4692.
[26] H. Schlegel, Exploring potential energy surfaces for chemical reactions: an overview of some practical methods, J. Comput. Chem. 24 (12) (2003) 1514–1527.
[27] L. Kuipers, H. Niederreiter, Uniform Distribution of Sequences, Courier Corporation, 2012.
[28] C. Doerr, M. Gnewuch, M. Wahlström, Calculation of discrepancy measures and applications, in: A Panorama of Discrepancy Theory, Springer, 2014, pp. 621–678.
[29] T.T. Warnock, Computational investigations of low-discrepancy point sets, in: Applications of Number Theory to Numerical Analysis, Elsevier, 1972, pp. 319–343.
[30] S. Joe, F. Kuo, Constructing Sobol′ sequences with better two-dimensional projections, SIAM J. Sci. Comput. 30 (5) (2008) 2635–2654.
[31] I. Sloan, F. Kuo, S. Joe, Constructing randomly shifted lattice rules in weighted Sobolev spaces, SIAM J. Numer. Anal. 40 (5) (2002) 1650–1665.
[32] J. Lyness, An introduction to lattice rules and their generator matrices, IMA J. Numer. Anal. 9 (3) (1989) 405–419.
[33] H. Niederreiter, The existence of good extensible polynomial lattice rules, Monatshefte Math. 139 (4) (2003) 295–307.
[34] R. Cools, F. Kuo, D. Nuyens, Constructing embedded lattice rules for multivariate integration, SIAM J. Sci. Comput. 28 (6) (2006) 2162–2188.
[35] H. Kroto, J. Heath, S. O'Brien, R. Curl, R. Smalley, C 60: buckminsterfullerene, Nature 318 (6042) (1985) 162–163.
[36] M. Saka, Optimum geometry design of geodesic domes using harmony search algorithm, Adv. Struct. Eng. 10 (6) (2007) 595–606.
[37] A. Tkach, M. Pauly, A. Tagliasacchi, Sphere-meshes for real-time hand modeling and tracking, ACM Trans. Graph. 35 (6) (2016) 1–11.
[38] K. Bowers, et al., Scalable algorithms for molecular dynamics simulations on commodity clusters, in: SC'06: Proceedings of the 2006 ACM/IEEE Conference on Supercomputing, IEEE, 2006, p. 43.
[39] D. Van Der Spoel, E. Lindahl, B. Hess, G. Groenhof, A. Mark, H. Berendsen, GROMACS: fast, flexible, and free, J. Comput. Chem. 26 (16) (2005) 1701–1718.
[40] S. Plimpton, Fast parallel algorithms for short-range molecular dynamics, J. Comput. Phys. 117 (1) (1995) 1–19.
[41] Lammps molecular dynamics simulator, https://lammps.sandia.gov.
[42] J. Phillips, et al., Scalable molecular dynamics with NAMD, J. Comput. Chem. 26 (16) (2005) 1781–1802.
[43] P. Eastman, et al., OpenMM 7: rapid development of high performance algorithms for molecular dynamics, PLoS Comput. Biol. 13 (7) (2017) e1005659.
[44] L. Verlet, Computer "experiments" on classical fluids. I. Thermodynamical properties of Lennard-Jones molecules, Phys. Rev. 159 (1) (1967) 98.
[45] D. Frenkel, B. Smit, Understanding Molecular Simulation: from Algorithms to Applications, vol. 1, Elsevier, 2001.
[46] W. Swope, H. Andersen, P. Berens, K. Wilson, A computer simulation method for the calculation of equilibrium constants for the formation of physical clusters of molecules: application to small water clusters, J. Chem. Phys. 76 (1) (1982) 637–649.
[47] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, J. Comput. Phys. 378 (2019) 686–707.
[48] J. Han, A. Jentzen, E. Weinan, Solving high-dimensional partial differential equations using deep learning, Proc. Natl. Acad. Sci. USA 115 (34) (2018) 8505–8510.
[49] S.L. Brunton, B.R. Noack, P. Koumoutsakos, Machine learning for fluid mechanics, Annu. Rev. Fluid Mech. 52 (2020) 477–508.
[50] G. Cybenko, Approximation by superpositions of a sigmoidal function, Math. Control Signals Syst. 2 (4) (1989) 303–314.
[51] I. Goodfellow, Y. Bengio, A. Courville, Y. Bengio, Deep Learning, vol. 1, MIT Press, Cambridge, 2016.
[52] L. Bottou, Online Algorithms and Stochastic Approximations, Online Learning and Neural Networks, 1998.
[53] J. Spall, Introduction to Stochastic Search and Optimization: Estimation, Simulation, and Control, vol. 65, John Wiley & Sons, 2005.
[54] J. Duchi, E. Hazan, Y. Singer, Adaptive subgradient methods for online learning and stochastic optimization, J. Mach. Learn. Res. 12 (7) (2011).
[55] D. Kingma, J. Ba Adam, A method for stochastic optimization, in: Y. Bengio, Y. LeCun (Eds.), 3rd International Conference on Learning Representations, ICLR 2015, 2015.
[56] G. Korpelevich, The extragradient method for finding saddle points and other problems, Matecon 12 (1976) 747–756.
[57] Y. Nesterov, Dual extrapolation and its applications to solving variational inequalities and related problems, Math. Program. 109 (2) (2007) 319–344.
[58] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778.
[59] A. Melzer, Mode spectra of thermally excited two-dimensional dust Coulomb clusters, Phys. Rev. E 67 (1) (2003) 016411.
[60] A. Radzvilavičius, E. Anisimovas, Topological defect motifs in two-dimensional Coulomb clusters, J. Phys. Condens. Matter 23 (38) (2011) 385301.
[61] I. Shapir, A. Hamo, S. Pecker, C.P. Moca, Ö. Legeza, G. Zarand, S. Ilani, Imaging the electronic Wigner crystal in one dimension, Science 364 (6443) (2019) 870–875.
[62] Y. Capdeville, E. Chaljub, J. Montagner, Coupling the spectral element method with a modal solution for elastic wave propagation in global Earth models, Geophys. J. Int. 152 (1) (2003) 34–67.
[63] J. Simpson, Current and future applications of 3-d global Earth-ionosphere models based on the full-vector Maxwell's equations FDTD method, Surv. Geophys. 30 (2) (2009) 105–130.
[64] V. Manoharan, Colloidal matter: packing, geometry, and entropy, Science 349 (6251) (2015).
[65] C. Gotsman, X. Gu, A. Sheffer, Fundamentals of spherical parameterization for 3d meshes, in: ACM SIGGRAPH 2003 Papers, 2003, pp. 358–363.
[66] G.-R. Liu, Meshfree Methods: Moving Beyond the Finite Element Method, CRC Press, 2009.
[67] D. Cai, H. Huang, E. Chow, Y. Xi, Data-driven construction of hierarchical matrices with nested bases, arXiv preprint, arXiv:2206.01885, 2022, https://arxiv.org/abs/2206.01885.
[68] V. Yudin, Minimum potential energy of a point system of charges, Diskretn. Mat. 4 (2) (1992) 115–121.
[69] J. Thomson, XXIV. On the structure of the atom, Philos. Mag. 7 (39) (1904) 237–265.

[70] P. Delsarte, J.-M. Goethals, J. Seidel, Spherical codes and designs, in: Geometry and Combinatorics, Elsevier, 1991, pp. 68–93.
[71] M. Fekete, Über die Verteilung der Wurzeln bei gewissen algebraischen Gleichungen mit ganzzahligen Koeffizienten, Math. Z. 17 (1) (1923) 228–249.
[72] S. Smale, Mathematical problems for the next century, Math. Intell. 20 (2) (1998) 7–15.
[73] N. Andreev, An extremal property of the icosahedron, East J. Approx. 2 (4) (1996) 459–462.
[74] PyTorch, https://pytorch.org.
[75] A. Paszke, et al., PyTorch: an imperative style, high-performance deep learning library, in: H. Wallach, et al. (Eds.), Advances in Neural Information Processing Systems 32, Curran Associates, Inc., 2019, pp. 8024–8035.
[76] A. Maas, A. Hannun, A. Ng, Rectifier nonlinearities improve neural network acoustic models, in: Proc. ICML, vol. 30, Citeseer, 2013, p. 3.
[77] J. Carrier, L. Greengard, V. Rokhlin, A fast adaptive multipole algorithm for particle simulations, SIAM J. Sci. Stat. Comput. 9 (4) (1988) 669–686, https://doi.org/10.1137/0909044.
[78] D. Cai, J. Xia, A stable matrix version of the fast multipole method: stabilization strategies and examples, Electron. Trans. Numer. Anal. 54 (2021) 581–609.
[79] S. Börm, L. Grasedyck, W. Hackbusch, Introduction to hierarchical matrices with applications, Eng. Anal. Bound. Elem. 27 (5) (2003) 405–422.
[80] W. Hackbusch, Hierarchical Matrices: Algorithms and Analysis, Springer Series in Computational Mathematics, Springer, Berlin Heidelberg, 2015.
[81] D. Cai, E. Chow, L. Erlandson, Y. Saad, Y. Xi, SMASH: structured matrix approximation by separation and hierarchy, Numer. Linear Algebra Appl. 25 (6) (2018) e2204.
[82] L. Erlandson, D. Cai, Y. Xi, E. Chow, Accelerating parallel hierarchical matrix-vector products via data-driven sampling, in: 2020 IEEE International Parallel and Distributed Processing Symposium, IPDPS, 2020, pp. 749–758.
[83] G. Papamakarios, E. Nalisnick, D.J. Rezende, S. Mohamed, B. Lakshminarayanan, Normalizing flows for probabilistic modeling and inference, arXiv preprint, arXiv:1912.02762, 2019.
[84] D. Rezende, S. Mohamed, Variational inference with normalizing flows, in: International Conference on Machine Learning, 2015, pp. 1530–1538.
[85] D.P. Kingma, P. Dhariwal Glow, Generative flow with invertible 1x1 convolutions, in: Advances in Neural Information Processing Systems, vol. 31, 2018, pp. 10215–10224.
[86] C. Durkan, A. Bekasov, I. Murray, G. Papamakarios, Neural spline flows, in: Advances in Neural Information Processing Systems, 2019, pp. 7509–7520.
[87] N. De Cao, W. Aziz, I. Titov, Block neural autoregressive flow, in: Proceedings of the 35th Uncertainty in Artificial Intelligence Conference, in: Proceedings of Machine Learning Research, vol. 115, 2020, pp. 1263–1273.
[88] D. Cai, Y. Ji, H. He, Q. Ye, Y. Xi, AUTM flow: atomic unrestricted time machine for monotonic normalizing flows, in: The 38th Conference on Uncertainty in Artificial Intelligence, 2022, https://openreview.net/forum?id=SlctILjcxc.
[89] R. Verfürth, A posteriori error estimation and adaptive mesh-refinement techniques, J. Comput. Appl. Math. 50 (1) (1994) 67–83.
[90] D. Cai, Z. Cai, A hybrid a posteriori error estimator for conforming finite element approximations, Comput. Methods Appl. Mech. Eng. 339 (2018) 320–340.
[91] D. Cai, Z. Cai, S. Zhang, Robust equilibrated a posteriori error estimator for higher order finite element approximations to diffusion problems, Numer. Math. 144 (1) (2020) 1–21.
[92] D. Cai, Z. Cai, S. Zhang, Robust equilibrated error estimator for diffusion problems: mixed finite elements in two dimensions, J. Sci. Comput. 83 (1) (2020) 1–22.
[93] D. Cai, Z. Cai, Hybrid a posteriori error estimators for conforming finite element approximations to stationary convection-diffusion-reaction equations, arXiv preprint, arXiv:2107.06341.