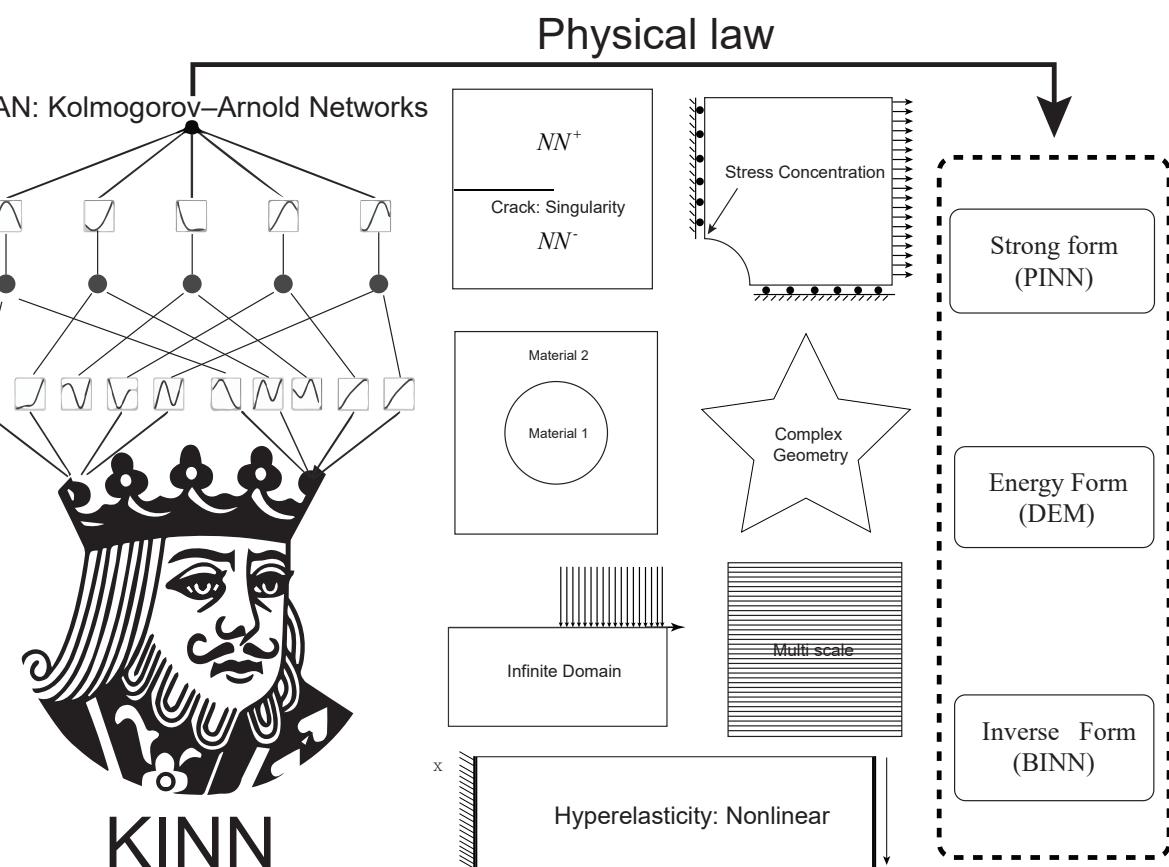


Graphical Abstract

arXiv:2406.11045v1 [cs.LG] 6 Jun 2024

Kolmogorov–Arnold-Informed neural network: A physics-informed deep learning framework for solving PDEs based on Kolmogorov–Arnold Networks

Yizheng Wang, Jia Sun, Jinshuai Bai, Cosmin Anitescu, Mohammad Sadegh Eshaghi, Xiaoying Zhuang, Timon Rabczuk, Yinghua Liu



Kolmogorov–Arnold-Informed neural network: A physics-informed deep learning framework for solving PDEs based on Kolmogorov–Arnold Networks

Yizheng Wang^{a,b}, Jia Sun^d, Jinshuai Bai^{a,e,f}, Cosmin Anitescu^b, Mohammad Sadegh Eshaghi^c, Xiaoying Zhuang^c, Timon Rabczuk^b, Yinghua Liu^{a,*}

^a*Department of Engineering Mechanics, Tsinghua University, Beijing 100084, China*

^b*Institute of Structural Mechanics, Bauhaus-Universität Weimar, Marienstr. 15, D-99423 Weimar, Germany*

^c*Institute of Photonics, Department of Mathematics and Physics, Leibniz University Hannover, Germany*

^d*Drilling Mechanical Department, CNPC Engineering Technology RD Company Limited, Beijing 102206, China*

^e*School of Mechanical, Medical and Process Engineering, Queensland University of Technology, Brisbane, QLD 4000, Australia*

^f*ARC Industrial Transformation Training Centre—Joint Biomechanics, Queensland University of*

Abstract

AI for partial differential equations (PDEs) has garnered significant attention, particularly with the emergence of Physics-informed neural networks (PINNs). The recent advent of Kolmogorov-Arnold Network (KAN) indicates that there is potential to revisit and enhance the previously MLP-based PINNs. Compared to MLPs, KANs offer interpretability and require fewer parameters. PDEs can be described in various forms, such as strong form, energy form, and inverse form. While mathematically equivalent, these forms are not computationally equivalent, making the exploration of different PDE formulations significant in computational physics. Thus, we propose different PDE forms based on KAN instead of MLP, termed Kolmogorov-Arnold-Informed Neural Network (KINN). We systematically compare MLP and KAN in various numerical examples of PDEs, including multi-scale, singularity, stress concentration, nonlinear hyperelasticity, heterogeneous, and complex geometry problems. Our results demonstrate that KINN significantly outperforms MLP in terms of accuracy and convergence speed for numerous PDEs in computational solid mechanics, except for the complex geometry problem. This highlights KINN's potential for more efficient and accurate PDE solutions in AI for PDEs.

Keywords: PINNs, Kolmogorov–Arnold Networks, Computational mechanics, AI for PDEs, AI for science

1. Introduction

A multitude of physical phenomena rely on partial differential equations (PDEs) for modeling. Thus, solving PDEs is essential for gaining an understanding of the behavior of both natural and engineered systems. However, once the boundary and initial conditions become complex, the exact solution of PDEs is often difficult to obtain [1]. At this point, various numerical methods are employed to solve PDEs for achieving approximate solutions, such as commonly used finite element methods [2, 3, 4, 5], mesh-free methods [6, 7, 8, 9, 10, 11], finite difference methods [12], finite volume methods [13], and boundary element methods [14].

Recently, AI for PDEs, an important direction of AI for science, refers to a class of algorithms that use deep learning to solve PDEs. There are three important approaches to AI for PDEs: Physics-Informed Neural Networks (PINNs) [15], operator learning [16], and Physics-Informed Neural Operator (PINO) [17]. We will review these three approaches. The first approach in AI for PDEs is PINNs [15]. Since the same PDE can have different forms of expression, each with varying accuracy and efficiency, different forms of PINNs have been developed based on these expressions. These forms include PINNs in strong form [15], PINNs in weak form (hp-VPINNs) [18], PINNs in energy form (DEM: Deep Energy Method) [1], and PINNs in inverse form (BINN:

*Corresponding author

Email addresses: wang-yz19@tsinghua.org.cn (Yizheng Wang), yhliu@mail.tsinghua.edu.cn (Yinghua Liu)

Boundary Element Method) [19]. It is important to emphasize that although these different mathematical descriptions of PDEs are equivalent, they are not computationally equivalent. Therefore, exploring different PDE descriptions is significant in computational physics. The second approach in AI for PDEs is operator learning, represented by DeepONet [16] and FNO (Fourier Neural Operator) [20]. Initially, proposed operator learning methods are purely data-driven, making them well-suited for problems with big data [21]. Notably, unlike the initially proposed PINNs, which can only solve specific problems and require re-solving when boundary conditions, geometries, or materials change, operator learning methods learn a series of mappings, i.e., a family of PDEs [20]. Operator learning can quickly provide solutions even when the above conditions change [23]. However, recent theories suggest that some operator learning algorithms exhibit aliasing error between the discrete representations and the continuous operators [22]. The third approach is PINO [17], which combines physical equations with operator learning. By incorporating physical equations during the training of operator learning, traditional operator learning can achieve higher accuracy [24, 25]. Additionally, PINO can utilize operator learning to first obtain a good approximate solution and then refine it using PDEs, greatly accelerating the computation of PDEs [26]. So why AI for PDEs can be successful?

AI for PDEs has succeeded mainly due to the neural network's strong approximation capability. The universal approximation theorem shows that a fully connected neural network (MLP), given enough hidden neurons and appropriate weight configurations, can approximate any continuous function with arbitrary precision [27, 28]. The universal approximation theorem is the cornerstone of MLP's application in AI for PDEs. However, another important theory, the Kolmogorov-Arnold representation theorem, also approximates any multivariate continuous function [29]. This theorem tells us that any multivariate function can be decomposed into a finite combination of univariate functions [30]. Based on the Kolmogorov-Arnold representation theorem, Hecht-Nielsen proposed the Kolmogorov network as approximate functions [31], but with only two layers. However, the nature of the univariate functions in the original Kolmogorov-Arnold representation theorem is often poor, making the Kolmogorov network with shallow layers nearly ineffective in practical applications [32].

Recently, Liu et al. proposed KAN (Kolmogorov-Arnold Network) [33], which deepens the shallow Kolmogorov network to create univariate functions with good properties. KAN is very similar to MLP, with the main difference being that KAN's activation functions need to be learned. In the original KAN, B-splines are used for activation function construction due to their excellent fitting ability [33]. Later, some studies modified the B-splines used as activation functions in KAN, replacing B-splines with Chebyshev orthogonal polynomials [34], radial basis functions [35], and wavelet transforms [36]. Function approximation theory in numerical analysis tells us that different weight functions form different orthogonal polynomials with various advantages, such as Legendre polynomials, Laguerre polynomials, Hermite orthogonal polynomials, and Chebyshev polynomials [37]. In theory, these orthogonal polynomials can all replace B-splines in KAN. Similarly, other traditional approximation functions can also replace B-splines, such as radial basis functions, wavelet transforms, and the renowned NURBS in IGA [38]. Therefore, theoretically, almost all traditional approximation functions can replace B-splines in KAN, making this work very extensive. However, the core of KAN is the introduction of a framework for composite functions that can learn activation functions.

As a result, we propose KINN, which is the KAN version of different forms of PDEs (strong form, energy form, and inverse form) in this work. Due to the extensive work on using different approximation functions instead of B-splines, we use the original B-spline version of KAN to directly compare it with MLP in different forms of PDEs. We systematically compare whether accuracy and efficiency are improved. Our motivation is simple: MLP has many parameters and lacks interpretability, with spectral bias problems [25], making it less accurate and interpretable when used as the approximate function for different forms of PINNs. However, KAN has fewer parameters than MLP, and because KAN's activation functions are B-splines, KAN's function construction is more aligned with the essence of numerical algorithms for solving PDEs. Therefore, it makes sense to believe that combining KAN with various forms of PINNs to replace MLP will have better results.

The outline of the paper is as follows. Section 2 introduces the mathematical descriptions of different PDEs, divided into strong form, energy form, and inverse form. Section 3 introduces the traditional KAN and our proposed KINN. Section 4 is numerical experiments and divided into three parts:

1. Problems that MLP cannot solve but KAN can: multi-scale problems, i.e., high and low-frequency mixed problems.

2. Problems where KAN is more accurate than MLP: crack singularity, stress concentration (plate with a circular hole), nonlinear hyperelastic problems, and heterogeneous problems.
3. Problems where KAN performs worse than MLP: complex boundary problems.

Finally, in Section 5, we summarize the current advantages and limitations of KINN and provide suggestions for the future research direction of KAN in solving PDEs.

2. Preparatory knowledge

In this section, we provide an overview of the different forms of AI for PDEs. Although the same PDEs can have different formulations, they all aim to solve the same underlying PDEs. The reason for researching these different formulations is that each form offers distinct advantages regarding computational efficiency and accuracy. Therefore, we introduce several different approaches to solving PDEs, including the strong form (PINNs: Physics-Informed Neural Networks), the energy form (DEM: Deep Energy Method), and the inverse form (BINNs: Boundary-Integral Neural Networks).

2.1. Introduction to the strong form of PDEs

We begin our discussion from the PDEs of boundary value problems, considering the following equations:

$$\begin{cases} \mathbf{P}(\mathbf{u}(\mathbf{x})) = \mathbf{f}(\mathbf{x}) & \mathbf{x} \in \Omega \\ \mathbf{B}(\mathbf{u}(\mathbf{x})) = \mathbf{g}(\mathbf{x}) & \mathbf{x} \in \Gamma \end{cases}, \quad (1)$$

where \mathbf{P} and \mathbf{B} (they could be nonlinear) are the domain operator and boundary operator of the differential equations, respectively, and Ω and Γ represent the domain and boundary, respectively. We use the weighted residual method to transform these equations into their weighted residual form:

$$\begin{cases} \int_{\Omega} [\mathbf{P}(\mathbf{u}(\mathbf{x})) - \mathbf{f}(\mathbf{x})] \cdot \mathbf{w}(\mathbf{x}) d\Omega = 0 & \mathbf{x} \in \Omega \\ \int_{\Gamma} [\mathbf{B}(\mathbf{u}(\mathbf{x})) - \mathbf{g}(\mathbf{x})] \cdot \mathbf{w}(\mathbf{x}) d\Gamma = 0 & \mathbf{x} \in \Gamma \end{cases}, \quad (2)$$

where $\mathbf{w}(\mathbf{x})$ is the weight function. The equations in their original form and weighted residual form are equivalent if $\mathbf{w}(\mathbf{x})$ is arbitrary. For numerical convenience, we often predefine the form of $\mathbf{w}(\mathbf{x})$ and get the residual form of the PDEs:

$$\mathbf{w}(\mathbf{x}) = \begin{cases} \mathbf{P}(\mathbf{u}(\mathbf{x})) - \mathbf{f}(\mathbf{x}) & \mathbf{x} \in \Omega \\ \mathbf{B}(\mathbf{u}(\mathbf{x})) - \mathbf{g}(\mathbf{x}) & \mathbf{x} \in \Gamma \end{cases}. \quad (3)$$

Eq. (2) is transformed into the integration form:

$$\begin{cases} \int_{\Omega} [\mathbf{P}(\mathbf{u}(\mathbf{x})) - \mathbf{f}(\mathbf{x})] \cdot [\mathbf{P}(\mathbf{u}(\mathbf{x})) - \mathbf{f}(\mathbf{x})] d\Omega = 0 & \mathbf{x} \in \Omega \\ \int_{\Gamma} [\mathbf{B}(\mathbf{u}(\mathbf{x})) - \mathbf{g}(\mathbf{x})] \cdot [\mathbf{B}(\mathbf{u}(\mathbf{x})) - \mathbf{g}(\mathbf{x})] d\Gamma = 0 & \mathbf{x} \in \Gamma \end{cases}. \quad (4)$$

Next, we approximate these integrals in Eq. (4) numerically, leading to the strong form of PINNs:

$$\mathcal{L}_{PINNs} = \frac{\lambda_r}{N_r} \sum_{i=1}^{N_r} |\mathbf{P}(\mathbf{u}(\mathbf{x}_i; \boldsymbol{\theta})) - \mathbf{f}(\mathbf{x}_i)|^2 + \frac{\lambda_b}{N_b} \sum_{i=1}^{N_b} |\mathbf{B}(\mathbf{u}(\mathbf{x}_i; \boldsymbol{\theta})) - \mathbf{g}(\mathbf{x}_i)|^2. \quad (5)$$

We optimize the above loss function to obtain the neural network approximation of the field variable $\mathbf{u}(\mathbf{x}; \boldsymbol{\theta})$:

$$\mathbf{u}(\mathbf{x}; \boldsymbol{\theta}) = \arg \min_{\boldsymbol{\theta}} \mathcal{L}_{PINNs}. \quad (6)$$

Thus, mathematically, the strong form of PINNs [15] essentially involves choosing the specific weight function $\mathbf{w}(\mathbf{x})$ as Eq. (3) as the residual form of the PDEs.

2.2. Introduction to the energy form of PDEs

In this chapter, we introduce the energy form of PDEs [1]. We consider the weight function $\mathbf{w}(\mathbf{x})$ in Eq. (2) as $\delta\mathbf{u}$, which leads to the Galerkin form. Eq. (2) can be written as:

$$\int_{\Omega} [\mathbf{P}(\mathbf{u}(\mathbf{x})) - \mathbf{f}(\mathbf{x})] \cdot \delta\mathbf{u} d\Omega = 0, \mathbf{x} \in \Omega. \quad (7)$$

Boundary conditions have not been introduced, because we will introduce Neumann boundary conditions and eliminate Dirichlet boundary conditions in the subsequent Gaussian integration formula. For simplicity, we consider a specific Poisson equation to illustrate the energy form:

$$\begin{cases} -\Delta(u(\mathbf{x})) = f(\mathbf{x}) & \mathbf{x} \in \Omega \\ u(\mathbf{x}) = \bar{u}(\mathbf{x}) & \mathbf{x} \in \Gamma^u, \\ \frac{\partial u(\mathbf{x})}{\partial n} = \bar{t}(\mathbf{x}) & \mathbf{x} \in \Gamma^t \end{cases}, \quad (8)$$

where Γ^u and Γ^t are the Dirichlet and Neumann boundary conditions, respectively. For the Poisson equation, the Galerkin form of Eq. (8) can be expressed as:

$$\int_{\Omega} [-\Delta(u(\mathbf{x})) - f(\mathbf{x})] \cdot \delta\mathbf{u} d\Omega = 0, \mathbf{x} \in \Omega. \quad (9)$$

Using the Gaussian integration formula, we can transform the above equation to:

$$\int_{\Omega} (-u_{,ii} - f) \delta\mathbf{u} d\Omega = \int_{\Omega} u_{,i} \delta u_{,i} d\Omega - \int_{\Gamma} u_{,i} n_i \delta u d\Gamma - \int_{\Omega} f \delta\mathbf{u} d\Omega = 0. \quad (10)$$

By incorporating the boundary conditions from Eq. (8) into Eq. (10), we obtain the Galerkin weak form:

$$\int_{\Omega} (-u_{,ii} - f) \delta\mathbf{u} d\Omega = \int_{\Omega} u_{,i} \delta u_{,i} d\Omega - \int_{\Gamma^t} \bar{t} \delta u d\Gamma - \int_{\Omega} f \delta\mathbf{u} d\Omega = 0. \quad (11)$$

Since $u(\mathbf{x})$ is given on Γ^u , the corresponding variation $\delta\mathbf{u} = 0$ on Γ^u . Here, we observe an interesting phenomenon: we must satisfy $u(\mathbf{x}) = \bar{u}(\mathbf{x})$ on Γ^u in advance, which involves constructing an admissible function. This is crucial for DEM (Deep energy form), and DEM essentially refers to a numerical algorithm that utilizes neural networks to solve the energy form of PDEs. Additionally, Eq. (11) includes the domain PDEs and the boundary conditions on Γ^t . Therefore, solving Eq. (11) is equivalent to solving Eq. (8).

We can further use the variational principle to write Eq. (11) as:

$$\delta\mathcal{L} = \int_{\Omega} u_{,i} \delta u_{,i} d\Omega - \int_{\Gamma^t} \bar{t} \delta u d\Gamma - \int_{\Omega} f \delta\mathbf{u} d\Omega \quad (12)$$

$$\mathcal{L} = \frac{1}{2} \int_{\Omega} u_{,i} u_{,i} d\Omega - \int_{\Gamma^t} \bar{t} u d\Gamma - \int_{\Omega} f u d\Omega \quad (13)$$

\mathcal{L} represents the potential energy. Eq. (12) is equivalent to Eq. (8), and we can observe that $\delta^2\mathcal{L} > 0$ (excluding zero solutions), indicating that we can solve for $u(\mathbf{x})$ by minimizing the energy:

$$u(\mathbf{x}) = \arg \min_u \mathcal{L}. \quad (14)$$

The essence of DEM is to approximate $u(\mathbf{x})$ using a neural network $u(\mathbf{x}; \boldsymbol{\theta})$, and then optimize Eq. (14):

$$u(\mathbf{x}; \boldsymbol{\theta}) = \arg \min_{\boldsymbol{\theta}} \mathcal{L}_{DEM} = \arg \min_u \left\{ \frac{1}{2} \int_{\Omega} u(\mathbf{x}; \boldsymbol{\theta})_{,i} u(\mathbf{x}; \boldsymbol{\theta})_{,i} d\Omega - \int_{\Gamma^t} \bar{t} u(\mathbf{x}; \boldsymbol{\theta}) d\Gamma - \int_{\Omega} f u(\mathbf{x}; \boldsymbol{\theta}) d\Omega \right\}. \quad (15)$$

Therefore, the core of DEM lies in the integration of the domain energy and boundary energy, as well as the construction of the admissible function. Integration strategies can use numerical analysis methods, such as simple Monte Carlo integration or more accurate methods like Gaussian integration or Simpson's Rule.

Here, we emphasize the construction of the admissible function. We use the concept of a distance network for the construction of the admissible function:

$$u(\mathbf{x}) = u_p(\mathbf{x}; \boldsymbol{\theta}_p) + D(\mathbf{x}) * u_g(\mathbf{x}; \boldsymbol{\theta}_g), \quad (16)$$

where $u_p(\mathbf{x}; \boldsymbol{\theta}_p)$ is the particular solution network that fits the Dirichlet boundary condition, such that it outputs $\bar{u}(\mathbf{x})$ when the input points are on Γ^u , and outputs any value elsewhere. The parameters $\boldsymbol{\theta}_p$ are optimized by:

$$\boldsymbol{\theta}_p = \arg \min_{\boldsymbol{\theta}_p} MSE(u_p(\mathbf{x}; \boldsymbol{\theta}_p), \bar{u}(\mathbf{x})), \mathbf{x} \in \Gamma^u, \quad (17)$$

where $D(x)$ is the distance network, which we approximate using radial basis functions [39, 40]. Other fitting functions can also be used. The effect is to output the minimum distance to the Dirichlet boundary:

$$D(\mathbf{x}) = \min_{\mathbf{y} \in \Gamma^u} \sqrt{(\mathbf{x} - \mathbf{y}) \cdot (\mathbf{x} - \mathbf{y})}, \quad (18)$$

where $u_g(\mathbf{x}; \boldsymbol{\theta}_g)$ is a standard neural network. When using the minimum potential energy principle, we only optimize $\boldsymbol{\theta}_g$:

$$u(\mathbf{x}; \boldsymbol{\theta}_p, \boldsymbol{\theta}_g) = \arg \min_{\boldsymbol{\theta}_g} \mathcal{L}_{DEM}. \quad (19)$$

Please note that not all PDEs have an energy form, but most have a corresponding weak form. The requirement that PDEs have the energy form is that PDEs need to satisfy the linear self-adjoint operator mathematically. The proof is provided in [Appendix A](#).

2.3. Introduction to the inverse form of PDEs

BINN (Boundary-Integral Type Neural Networks) essentially refers to a numerical algorithm that utilizes neural networks to solve the inverse form of PDEs. The inverse form of PDEs is often represented by boundary integral equations. Mathematically, boundary integral equations are derived from the weighted residual method shown in Eq. (2) using Gaussian integration. This process transforms differential operators from the trial function to the test function and uses the fundamental solution of the differential equation to convert all unknowns to the boundary.

To illustrate boundary integral equations, we start from Eq. (8) and transform the strong form into a weighted residual form:

$$\int_{\Omega} [-\Delta u(\mathbf{x}) \cdot w(\mathbf{x}) - f(\mathbf{x}) \cdot w(\mathbf{x})] d\Omega = 0. \quad (20)$$

Using Gaussian integration twice, we can transfer the Laplacian operator Δ to the weight function $w(\mathbf{x})$, incorporating the boundary conditions to obtain the inverse form:

$$\int_{\Omega} [-u \cdot (\Delta w) - f \cdot w] d\Omega + \int_{\Gamma^u} \bar{u} \frac{\partial w}{\partial \mathbf{n}} d\Gamma + \int_{\Gamma^t} u \frac{\partial w}{\partial \mathbf{n}} d\Gamma - \int_{\Gamma^u} \frac{\partial u}{\partial \mathbf{n}} w d\Gamma - \int_{\Gamma^t} \bar{t} w d\Gamma = 0, \quad (21)$$

where \mathbf{n} is the outer normal vector of the boundary. If we choose the weight function as the fundamental solution of the differential equation:

$$\Delta w(\mathbf{x}; \mathbf{y}) = \delta(\mathbf{x} - \mathbf{y}), \quad (22)$$

where $\delta(\mathbf{x} - \mathbf{y})$ is the Dirac delta function, the solution to the above equation is denoted as $w(\mathbf{x}; \mathbf{y}) = u^f(\mathbf{x}; \mathbf{y}) = -\ln(r)/(2\pi)$, with $r = \sqrt{(\mathbf{x} - \mathbf{y}) \cdot (\mathbf{x} - \mathbf{y})}$. Substituting Eq. (22) into Eq. (21), we get:

$$\begin{aligned} & c(\mathbf{y}) u(\mathbf{y}) + \int_{\Gamma^u} \frac{\partial u(\mathbf{x})}{\partial \mathbf{n}} u^f(\mathbf{x}; \mathbf{y}) d\Gamma - \int_{\Gamma^t} u(\mathbf{x}) \frac{\partial u^f(\mathbf{x}; \mathbf{y})}{\partial \mathbf{n}} \Gamma = \\ & - \int_{\Omega} f(\mathbf{x}) \cdot u^f(\mathbf{x}; \mathbf{y}) d\Omega + \int_{\Gamma^u} \bar{u}(\mathbf{x}) \frac{\partial u^f(\mathbf{x}; \mathbf{y})}{\partial \mathbf{n}} \Gamma - \int_{\Gamma^t} \bar{t}(\mathbf{x}) u^f(\mathbf{x}; \mathbf{y}) d\Gamma \end{aligned} \quad (23)$$

where $c(\mathbf{y})$ is determined by boundary continuity, being 0.5 on a smooth boundary, 1 inside the domain, and 0 outside the domain. It is evident that, when \mathbf{y} on the boundary, all unknowns in Eq. (23) are located on the boundary:

$$\begin{aligned} \frac{\partial u(\mathbf{x})}{\partial n}, \quad \mathbf{x} \in \Gamma^u \\ u(\mathbf{x}), \quad \mathbf{x} \in \Gamma^t. \end{aligned} \quad (24)$$

We use a neural network $\phi(\mathbf{x}; \boldsymbol{\theta})$ to approximate these unknowns $u(\mathbf{x})$ on Γ^t and $\partial u(\mathbf{x})/\partial n$ on Γ^u . Note that $c(\mathbf{y})u(\mathbf{y})$ is chosen as:

$$\begin{cases} c(\mathbf{y})u(\mathbf{y}) = c(\mathbf{y})\bar{u}(\mathbf{y}) & \mathbf{y} \in \Gamma^u, \\ c(\mathbf{y})u(\mathbf{y}) = c(\mathbf{y})\phi(\mathbf{y}; \boldsymbol{\theta}) & \mathbf{y} \in \Gamma^t. \end{cases} \quad (25)$$

The essence of BINN is to solve Eq. (23), and the loss function for BINN is:

$$\begin{aligned} \mathcal{L}_{BINN} = \frac{1}{N} \sum_{i=1}^{N_s} |R(\mathbf{y}_i; \boldsymbol{\theta})|, \quad \mathbf{y}_i \in \Gamma \\ R(\mathbf{y}; \boldsymbol{\theta}) = c(\mathbf{y})u(\mathbf{y}) + \int_{\Gamma^u} \frac{\partial \phi(\mathbf{x}; \boldsymbol{\theta})}{\partial n} u^f(\mathbf{x}; \mathbf{y}) d\Gamma - \int_{\Gamma^t} \phi(\mathbf{x}; \boldsymbol{\theta}) \frac{\partial u^f(\mathbf{x}; \mathbf{y})}{\partial n} d\Gamma \\ + \int_{\Omega} f(\mathbf{x}) \cdot u^f(\mathbf{x}; \mathbf{y}) d\Omega - \int_{\Gamma^u} \bar{u}(\mathbf{x}) \frac{\partial u^f(\mathbf{x}; \mathbf{y})}{\partial n} d\Gamma + \int_{\Gamma^t} \bar{t}(\mathbf{x}) u^f(\mathbf{x}; \mathbf{y}) d\Gamma. \end{aligned} \quad (26)$$

Here, N_s is the number of source points, i.e., the points at which the loss function is evaluated. It is noteworthy that numerical integration is critical for BINN, as each \mathbf{y}_i requires the computation of singular integrals on the boundary. For specific details, please refer to [19].

3. Method

The purpose of the previous chapter was to gain a better understanding of the different numerical formats for solving PDEs. With the basic concepts established, our idea is quite simple: we propose replacing the traditional MLP used in AI for PDEs with KAN (Kolmogorov–Arnold Networks) [33] in different PDEs forms (strong, energy, and inverse form), subsequently named KINN. In the following sections, we introduce the KAN network and KINN.

3.1. Kolmogorov–Arnold Networks

In KAN, the weights to be optimized depend on the number of input neurons l_i and output neurons l_o in each layer of KAN. We denote the activation function for the layer of KAN as ϕ_{ij} , where $i \in \{1, 2, \dots, l_o\}$ and $j \in \{1, 2, \dots, l_i\}$. Each element of the spline activation function ϕ_{ij} is determined by the number of grids size G and the order of the B-splines r . The specific expression is given by:

$$\phi_{ij}(\mathbf{X}) = \begin{bmatrix} \sum_{m=1}^{G_1+r_1} c_m^{(1,1)} B_m(x_1) & \sum_{m=1}^{G_2+r_2} c_m^{(1,2)} B_m(x_2) & \dots & \sum_{m=1}^{G_{l_i}+r_{l_i}} c_m^{(1,l_i)} B_m(x_{l_i}) \\ \sum_{m=1}^{G_1+r_1} c_m^{(2,1)} B_m(x_1) & \sum_{m=1}^{G_2+r_2} c_m^{(2,2)} B_m(x_2) & \dots & \sum_{m=1}^{G_{l_i}+r_{l_i}} c_m^{(2,l_i)} B_m(x_{l_i}) \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{m=1}^{G_1+r_1} c_m^{(l_o,1)} B_m(x_1) & \sum_{m=1}^{G_2+r_2} c_m^{(l_o,2)} B_m(x_2) & \dots & \sum_{m=1}^{G_{l_i}+r_{l_i}} c_m^{(l_o,l_i)} B_m(x_{l_i}) \end{bmatrix} \quad (27)$$

where G_j is the number of grids in the j -th input direction, and r_j is the order of the B-splines in the j -th input direction, with $j \in \{1, 2, \dots, l_i\}$. The coefficients $c_m^{(i,j)}$ are the B-spline coefficients, whose number is determined by the grid G_j and the order r_j , totaling $G_j + r_j$. Note that the grid division and order in each input direction are independent and can be selected individually. B_m is the basis function of the B-spline.

Table 1
The trainable parameter in KAN

Trainable parameters	Variable	Number	Description
$c_m^{(i,j)}$	spline_weight	$l_o * l_i * (G + r)$	Coefficients of B-spline in activation function $\phi(\mathbf{X})$
W_{ij}	base_weight	$l_i * l_o$	Linear transformation for nonlinear activation function $\sigma(\mathbf{X})$
S_{ij}	spline_scaler	$l_i * l_o$	Scaling factors of activation function $\phi(\mathbf{X})$

To enhance the fitting ability of the activation function, we introduce S_{ij} , a matrix of the same size as ϕ_{ij} , given by:

$$S_{ij} = \begin{bmatrix} s_{11} & s_{12} & \cdots & s_{1l_i} \\ s_{21} & s_{22} & \cdots & s_{2l_i} \\ \vdots & \vdots & \ddots & \vdots \\ s_{l_o 1} & s_{l_o 2} & \cdots & s_{l_o l_i} \end{bmatrix}. \quad (28)$$

The role of S is to adjust the magnitude of the activation function ϕ , i.e., $\phi = \phi \odot S$, where \odot denotes element-wise multiplication. Additionally, we have nonlinear activation and linear matrix multiplication operations, and the final output is given by:

$$\mathbf{Y} = [\sum_{\text{column}} \phi(\mathbf{X}) \odot S] + \mathbf{W} \cdot \sigma(\mathbf{X}), \quad (29)$$

where \mathbf{W} is the linear matrix operation, and σ is the nonlinear activation function. The inclusion of σ ensures smooth function fitting; without it, relying solely on B-splines could lead to a rough approximation. S and \mathbf{W} act as scaling factors, similar to normalization in machine learning. The term $\mathbf{W} \cdot \sigma(\mathbf{X})$ is a residual term, akin to the ResNet approach [41]. The code of KAN is adapted from <https://github.com/Blealtan/efficient-kan> and <https://github.com/KindXiaoming/pykan>

Assuming that the grid division and B-spline order in each input neuron direction are consistent, the number of $c_m^{(i,j)}$ are all $G + r$. The trainable parameters are shown in Table 1.

3.2. KINN

Fig. 1 illustrates the core idea of KINN¹, which uses KAN instead of MLP in the different forms of PDEs (strong, energy, and inverse form). Notably, as the output range can easily exceed the grid range in B-splines after a single KAN layer (we must set the grid range in advance), we apply a tanh activation function to the output of the KAN layer to constrain it within $[-1, 1]$ and better utilize the capabilities of the B-splines in this range. Thus, the output of each KAN layer undergoes an additional tanh activation:

$$\mathbf{Y}^{\text{new}} = \tanh(\mathbf{Y}) = \tanh \left\{ [\sum_{\text{column}} \phi(\mathbf{X}) \odot S] + \mathbf{W} \cdot \sigma(\mathbf{X}) \right\}. \quad (30)$$

However, we do not apply tanh to the final layer since the output does not necessarily lie within $[-1, 1]$. If the simulation domain is outside the $[-1, 1]$ grid range, we perform a scale normalization for input \mathbf{X} . For instance, we can find the smallest bounding box enclosing the geometry to be simulated, denoted as $[L, W, X_c, Y_c]$ as shown in Fig. 1, and then normalize the input as follows:

$$x^s = \frac{x - X_c}{L/2}; \quad y^s = \frac{y - Y_c}{W/2}. \quad (31)$$

Once the grid size, grid range, and order of the B-splines are determined, the KAN approximation function is fixed, forming a virtual grid approximation function as shown in Fig. 1. Next, we select the appropriate form

¹KINN refers to various PDEs forms (strong: PINNs, energy: DEM, and inverse: BINN) for solving PDEs based on KAN, so sometimes we use KAN directly to represent the various PDEs forms.

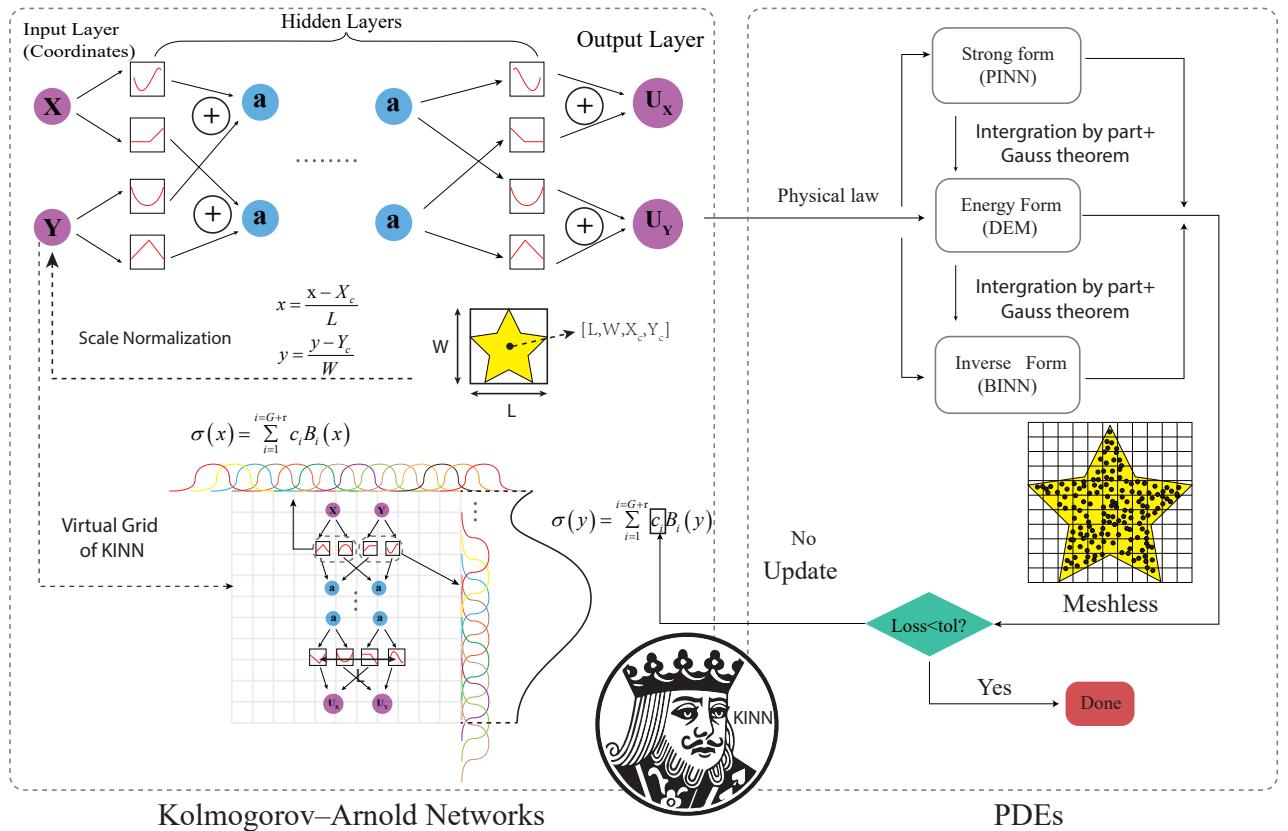


Fig. 1. Schematic of KINN: The idea of KINN is to replace MLP with Kolmogorov–Arnold Networks (KAN) in different PDEs forms (strong, energy, and inverse forms). In KAN, the main training parameters are the undetermined coefficients c_i of the B-splines in the activation function. KINN establishes the loss function based on different numerical formats of PDEs and optimizes the c_i . The virtual Grid is determined by the grid size in KAN.

of PDEs, such as strong form, energy form, or inverse form, as detailed in Section 2. After determining the PDE form, we randomly sample points within the domain and on the boundary to compute the corresponding loss functions, as the meshless random sampling in the strong form of PINNs. Finally, we optimize the trainable parameters in KAN, as shown in Table 1.

KINN shares similarities with finite element methods (FEM) and isogeometric analysis (IGA). The approximation function of KAN is like composition of functions based on NURBS in IGA or shape functions in FEM, as proven in Appendix B. Additionally, the highest derivative order of the PDEs that KINN can solve depends on the selected activation function, the number of layers, and the order of B-splines, as analyzed in Appendix C. Below, we present numerical results of solving PDEs using KINN. We compare the results of PDEs in strong form, energy form, and inverse form between MLP and the corresponding KINN. All computations are carried out on a single Nvidia RTX 4060 Ti GPU with 16GB memory.

4. Result

In this chapter, we demonstrate the performance of KINN in solving different PDE formats, including the strong, energy, and inverse forms. To compare the performance of MLP and KAN in solving PDEs, we ensure that all methods use the same optimization method and learning rate for both KAN and MLP unless otherwise specified. The only difference is the type of neural network used.

4.1. KAN can, MLP cannot

To demonstrate the advantages of KAN, we present a function-fitting example that MLP fails to achieve. Wang et al. [42] found that MLPs perform poorly on multi-scale PDE problems. According to the Neural Tangent Kernel (NTK) theory [43], MLPs are ineffective in fitting high-frequency components of functions. This is because the eigenvalues of the NTK matrix for high-frequency components are smaller than those for low-frequency components. To illustrate this point, we first introduce the NTK matrix:

$$\begin{aligned} \mathbf{K}_{ntk}(\mathbf{x}, \mathbf{x}') &= -\lim_{\eta \rightarrow 0} \frac{f(\mathbf{x}; \boldsymbol{\theta} - \eta \frac{\partial f}{\partial \boldsymbol{\theta}}|_{\mathbf{x}'}) - f(\mathbf{x}; \boldsymbol{\theta})}{\eta} \\ &= -\lim_{\eta \rightarrow 0} \frac{f(\mathbf{x}; \boldsymbol{\theta}) - \eta (\frac{\partial f}{\partial \boldsymbol{\theta}}|_{\mathbf{x}'}) \circ (\frac{\partial f}{\partial \boldsymbol{\theta}}|_{\mathbf{x}}) + O(\eta \frac{\partial f}{\partial \boldsymbol{\theta}}|_{\mathbf{x}'}) - f(\mathbf{x}; \boldsymbol{\theta})}{\eta} = (\frac{\partial f}{\partial \boldsymbol{\theta}}|_{\mathbf{x}}) \cdot (\frac{\partial f}{\partial \boldsymbol{\theta}}|_{\mathbf{x}'}) \end{aligned} \quad (32)$$

where f is the function to be fitted, such as the displacement field in mechanics. The elements of the NTK matrix are computed for all training points. If there are N points, the NTK matrix \mathbf{K}_{ntk} is an $N \times N$ non-negative definite matrix. The NTK matrix is calculated using the last term in Eq. (32), by computing the gradients at points \mathbf{x} and \mathbf{x}' and then taking their dot product. The limit form in Eq. (32) describes the NTK matrix, which measures the change at point \mathbf{x} when performing gradient descent at point \mathbf{x}' . As neural network parameters are interdependent, the NTK matrix can be used to measure the convergence of neural network algorithms (a larger change implies faster convergence).

Next, we consider the least squares loss function:

$$\begin{aligned} \mathcal{L} &= \frac{1}{2n} \sum_{i=1}^n [f(\mathbf{x}_i; \boldsymbol{\theta}) - y_i]^2 = \frac{1}{n} \sum_{i=1}^n l_i \\ l_i &= \frac{1}{2} [f(\mathbf{x}_i; \boldsymbol{\theta}) - y_i]^2 \end{aligned} \quad (33)$$

The gradient of Eq. (33) with respect to the parameters is:

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} &= \frac{1}{n} \sum_{i=1}^n \{[f(\mathbf{x}_i; \boldsymbol{\theta}) - y_i] \frac{\partial f(\mathbf{x}_i; \boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\} = \frac{1}{n} \sum_{i=1}^n \frac{\partial l_i}{\partial \boldsymbol{\theta}} \\ \frac{\partial l_i}{\partial \boldsymbol{\theta}} &= [f(\mathbf{x}_i; \boldsymbol{\theta}) - y_i] \frac{\partial f(\mathbf{x}_i; \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \end{aligned} \quad (34)$$

Considering the dynamic gradient and substituting Eq. (34):

$$\begin{aligned}\frac{df(\mathbf{x}_i; \boldsymbol{\theta})}{dt} &= \lim_{\eta \rightarrow 0} \frac{f(\mathbf{x}_i; \boldsymbol{\theta} - \eta \frac{\partial l_i}{\partial \boldsymbol{\theta}}|_{\mathbf{x}_i}) - f(\mathbf{x}_i; \boldsymbol{\theta})}{\eta} = -\frac{\partial l_i}{\partial \boldsymbol{\theta}}|_{\mathbf{x}_i} \cdot \frac{\partial f}{\partial \boldsymbol{\theta}}|_{\mathbf{x}_i} \\ &= -(\frac{\partial f}{\partial \boldsymbol{\theta}}|_{\mathbf{x}_i} \cdot \frac{\partial f}{\partial \boldsymbol{\theta}}|_{\mathbf{x}_i})[f(\mathbf{x}_i; \boldsymbol{\theta}) - y_i]\end{aligned}\quad (35)$$

Comparing Eq. (35) and Eq. (32), we obtain the vector form of Eq. (35):

$$\begin{aligned}\frac{d\mathbf{f}(\mathbf{X}; \boldsymbol{\theta})}{dt} &= -\mathbf{K}_{ntk} \cdot [\mathbf{f}(\mathbf{X}; \boldsymbol{\theta}) - \mathbf{Y}] \\ \mathbf{f}(\mathbf{X}; \boldsymbol{\theta}) &= [f(\mathbf{x}_1; \boldsymbol{\theta}), f(\mathbf{x}_2; \boldsymbol{\theta}), \dots, f(\mathbf{x}_n; \boldsymbol{\theta})]^T \\ \mathbf{Y} &= [y_1, y_2, \dots, y_n]^T\end{aligned}\quad (36)$$

Since a first-order linear ordinary differential equation system has an analytical solution, we obtain the particular solution of Eq. (36):

$$\mathbf{f}(\mathbf{X}; \boldsymbol{\theta}) = (\mathbf{I} - e^{-t\mathbf{K}_{ntk}}) \cdot \mathbf{Y} \quad (37)$$

Next, we perform an eigenvalue decomposition on $e^{-t\mathbf{K}_{ntk}}$. Since \mathbf{K}_{ntk} is a real symmetric matrix, it can be diagonalized as $\mathbf{K}_{ntk} = \mathbf{Q} \cdot \boldsymbol{\lambda} \cdot \mathbf{Q}^T$, where $\mathbf{Q} = [\mathbf{q}_1, \mathbf{q}_2, \dots, \mathbf{q}_n]$. \mathbf{q}_i are the eigenvectors of \mathbf{K}_{ntk} . Therefore:

$$e^{-t\mathbf{K}_{ntk}} = e^{-\mathbf{Q} \cdot \boldsymbol{\lambda} \cdot \mathbf{Q}^T t} = \mathbf{Q} \cdot e^{-\boldsymbol{\lambda} t} \cdot \mathbf{Q}^T \quad (38)$$

Substituting Eq. (38) into Eq. (37), we get:

$$\begin{aligned}\mathbf{f}(\mathbf{X}; \boldsymbol{\theta}) &= (\mathbf{I} - \mathbf{Q} \cdot e^{-\boldsymbol{\lambda} t} \cdot \mathbf{Q}^T) \cdot \mathbf{Y} \\ \mathbf{Q}^T[\mathbf{f}(\mathbf{X}; \boldsymbol{\theta}) - \mathbf{Y}] &= e^{-\boldsymbol{\lambda} t} \cdot \mathbf{Q}^T \cdot \mathbf{Y}\end{aligned}\quad (39)$$

$$\left[\begin{array}{c} \mathbf{q}_1^T \\ \mathbf{q}_2^T \\ \vdots \\ \mathbf{q}_n^T \end{array} \right] \left[\begin{array}{c} f(\mathbf{x}_1; \boldsymbol{\theta}) - y_1 \\ f(\mathbf{x}_2; \boldsymbol{\theta}) - y_2 \\ \vdots \\ f(\mathbf{x}_n; \boldsymbol{\theta}) - y_n \end{array} \right] = \left[\begin{array}{cccc} e^{-\lambda_1 t} & & & \\ & e^{-\lambda_2 t} & & \\ & & \ddots & \\ & & & e^{-\lambda_n t} \end{array} \right] \cdot \mathbf{Q}^T \cdot \mathbf{Y} \quad (40)$$

From Eq. (40), we observe that the residual $f(\mathbf{x}_i; \boldsymbol{\theta}) - y_i$ is inversely proportional to the eigenvalues of \mathbf{K}_{ntk} . The more evenly distributed the eigenvalues of \mathbf{K}_{ntk} , the better the neural network's fitting ability. However, the eigenvalues of fully connected MLPs are generally concentrated, and the large eigenvalue eigenvectors are usually low-frequency. This causes MLPs to prefer low frequencies. Although Fourier feature embeddings can partially mitigate the low-frequency preference, they introduce an additional hyperparameter σ . The above derivations and analyses are largely based on [42].

Naturally, we ask whether KAN can fundamentally solve the "spectral bias" of MLP.

To validate our hypothesis, we consider the following 1D Poisson problem and its analytical solution:

$$\begin{cases} \frac{\partial^2 u(x)}{\partial x^2} = -4\pi^2 \sin(2\pi x) - 250\pi^2 \sin(50\pi x), & x \in [0, 1] \\ u(0) = u(1) = 0 \\ u = \sin(2\pi x) + 0.1 \sin(50\pi x) \end{cases} \quad (41)$$

To simplify, we first consider function fitting. If function fitting fails, solving PDEs will certainly fail. To objectively validate, we solve the same problem as in [42], referencing the code at <https://github.com/PredictiveIntelligenceLab/MultiscalePINNs>. We fit $u = \sin(2\pi x) + 0.1 \sin(50\pi x)$ using MLP and KAN. As shown in Fig. 2a, we uniformly distribute 100 points in $[0, 1]$. The structure of KAN is $[1, 5, 5, 5, 1]$ while the structure of MLP is $[1, 100, 100, 100, 100, 1]$. We set the same learning rate (0.001) and optimizer (Adam) for objective

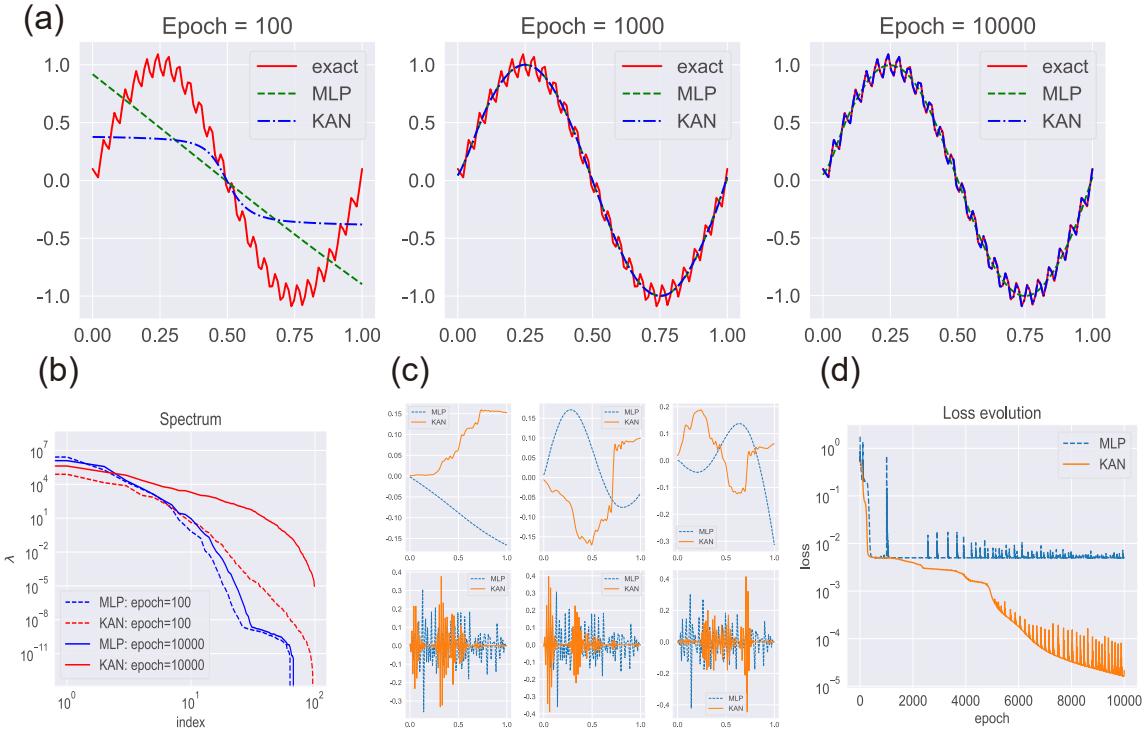


Fig. 2. Validation of the "spectral bias" of MLP and KAN in fitting $u = \sin(2\pi x) + 0.1 \sin(50\pi x)$. (a) Exact solutions, MLP predictions, and KAN predictions at epochs 100, 1000, and 10000, (b) Eigenvalue distributions of MLP and KAN, (c) The First row is the eigenvectors of the largest eigenvalue sorted from largest to smallest, and the second row is the eigenvectors of the three smallest eigenvalues, (d) Evolution of the loss functions of MLP and KAN.

comparison. Fig. 2b demonstrates that KAN has a more evenly distributed eigenvalue spectrum compared to MLP. Additionally, Fig. 2c shows the eigenvectors of KAN contain both high and low frequencies. The eigenvalues of the NTK matrix represent the convergence rate under the corresponding eigenvectors, as shown in Eq. (40). Specifically, the eigenvalues indicate the convergence speed of the residual in the eigenvalue space. The eigenvectors in Fig. 2c represent the basis modes fitted by the neural network. In summary, the eigenvalues and eigenvectors inform us of the convergence speed in the corresponding basis modes. KAN converges faster and with higher accuracy for problems with both low and high frequencies. Fig. 2c shows that in the feature vector space, KAN exhibits similar patterns to MLP in the low-frequency range. However, KAN simultaneously carries high-frequency information within the low-frequency range. The fact that the eigenvector of KAN under the largest eigenvalue includes both high and low frequencies indicates that KAN can rapidly converge on a mix of high and low frequencies, unlike MLP, which tends to favor low frequencies. This is because KAN's structure, utilizing B-splines, is more suitable for the multi-scale function approximation.

Thus, our results show that compared to MLP, KAN almost does not have the problem of "spectral bias" and can solve some problems that MLP cannot. The following numerical examples demonstrate the advantage of KAN in solving high and low-frequency mixed PDEs.

Since the above is a purely data-driven example, it is to demonstrate KAN's fitting ability for high and low-frequency problems. If MLP cannot fit a function approximation problem, it is unlikely to fit PDE-solving problems. As this work emphasizes the capabilities of KINN for solving PDEs, we solve the same heat conduction problem in [42]:

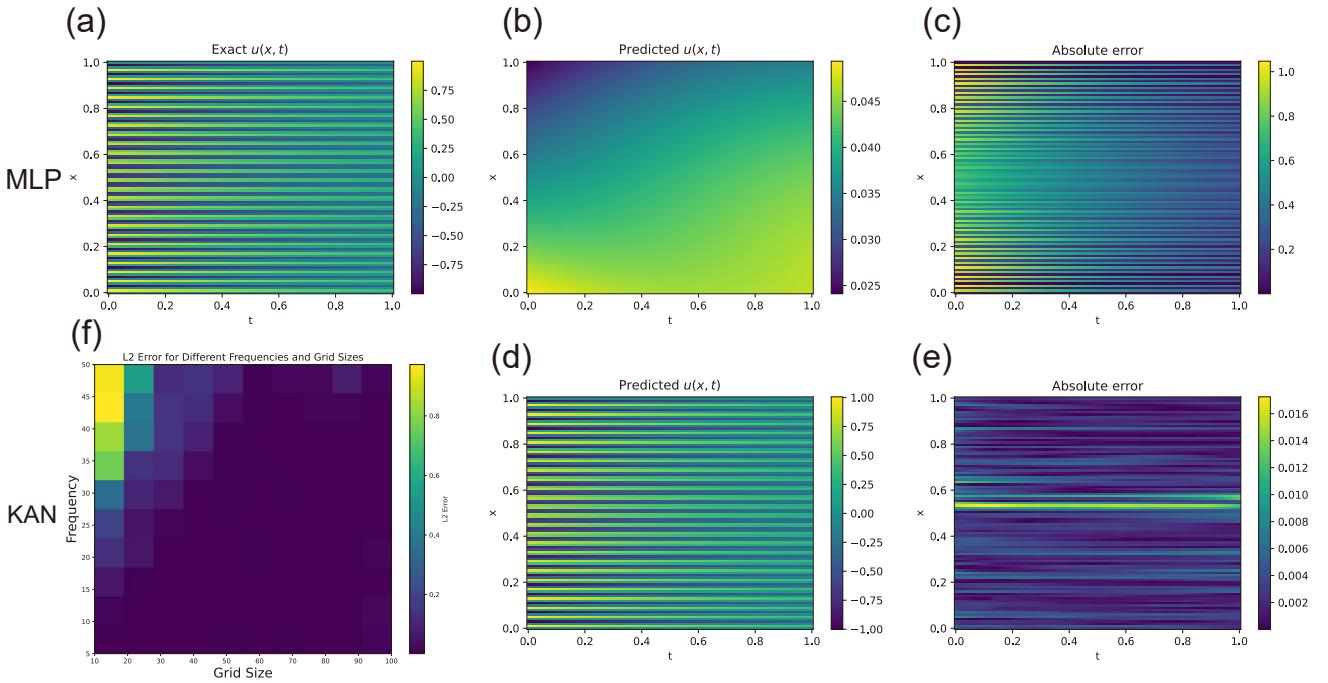


Fig. 3. MLP and KAN fitting the high and low-frequency mixed heat conduction problem. (a) Exact solution for frequency $F = 50$. (b) MLP prediction for frequency $F = 50$ after 5000 iterations. (c) The absolute error of MLP for frequency $F = 50$ after 5000 iterations. (d) KAN prediction for frequency $F = 50$ after 5000 iterations. (e) The absolute error of KAN for frequency $F = 50$ after 5000 iterations. (f) Relative error of KAN under different grid sizes and frequencies, with 3000 iterations and network structure [2,5,1].

$$\begin{aligned}
 u_t &= \frac{1}{(F\pi)^2} u_{xx}, \quad x \in [0, 1], t \in [0, 1] \\
 u(x, 0) &= \sin(F\pi x), \quad x \in [0, 1] \\
 u(0, t) &= u(1, t) = 0, \quad t \in [0, 1]
 \end{aligned} \tag{42}$$

where F is the frequency. The analytical solution we set for high and low frequency mix is:

$$u(x, t) = e^{-t} \sin(F\pi x) \tag{43}$$

The structure of MLP is [2,30,30,30,30,1], and the structure of KAN is [2,5,5,1]. Apart from the different network structures, all other settings are the same. Note that the grid size of KAN is 100 and the order is 3. Fig. 3 shows the performance of MLP and KAN in solving Eq. (42) (with $F = 50$). The results are after 5000 epochs, where we can see that MLP completely fails to solve this problem, but KAN achieves good results. This further highlights that KAN's "spectral bias" is far less severe than that of MLP.

However, during our numerical experiments, we found that if KAN's grid size is too small, it also fails to solve the problem with high frequency F . Therefore, Fig. 3(f) shows the relative error \mathcal{L}_2 of KAN under different frequencies F and grid sizes, with 3000 iterations and a network structure of [2,5,1]. The results indicate that KAN's "spectral bias" is related to the grid size. Nevertheless, compared to MLP, KAN's "spectral bias" is far less severe.

4.2. KAN are more accurate than MLP in solving PDEs

4.2.1. Problems with singularities

Mode III crack as shown in Fig. 4a is a benchmark problem used to verify algorithms for solving singular problems [39]. The PDEs for the mode III crack are as follows:

$$\begin{cases} \Delta(u(\mathbf{x})) = 0 & \mathbf{x} \in \Omega \\ \bar{u}(\mathbf{x}) = r^{\frac{1}{2}} \sin(\frac{1}{2}\theta) & \mathbf{x} \in \Gamma \end{cases}. \quad (44)$$

where Δ is the Laplacian operator, \mathbf{x} is the coordinate point, and Ω and Γ represent the square domain ($[-1, 1]^2$) and the boundary, respectively. The analytical solution $r^{\frac{1}{2}} \sin(\theta/2)$ is applied as the boundary condition. Here, r is the distance of \mathbf{x} from the origin point $(0,0)$, and θ is the angle in the polar coordinate system, with $x > 0, y = 0$ as the reference. Counterclockwise is considered a positive angle. The range of θ is $[-\pi, +\pi]$, which causes two different values of u to appear at the same coordinate at the crack ($x < 0, y = 0$). Therefore, this problem is strongly discontinuous, shown in Fig. 4b.

To demonstrate the performance of KINN, we compare the results of KINN with different solving formats, including traditional strong form by PINNs, energy form by DEM, and inverse form by BINN. The comparison is made based on the accuracy and efficiency of solving the same problem. Since the displacement field is discontinuous at the crack, we use CPINNs [44] for the strong form of PINNs. Displacement continuity and displacement derivative continuity conditions are incorporated into CPINN. The loss function for CPINNs is given by:

$$\begin{aligned} \mathcal{L}_{cpinn} = & \lambda_1 \sum_{i=1}^{N_{pde+}} |\Delta(u^+(\mathbf{x}_i))|^2 + \lambda_2 \sum_{i=1}^{N_{pde-}} |\Delta(u^-(\mathbf{x}_i))|^2 + \lambda_3 \sum_{i=1}^{N_b+} |u^+(\mathbf{x}_i) - \bar{u}(\mathbf{x}_i)|^2 + \lambda_4 \sum_{i=1}^{N_b-} |u^-(\mathbf{x}_i) - \bar{u}(\mathbf{x}_i)|^2 \\ & + \lambda_5 \sum_{i=1}^{N_I} |u^-(\mathbf{x}_i) - u^+(\mathbf{x}_i)|^2 + \lambda_6 \sum_{i=1}^{N_I} |\mathbf{n} \cdot (\nabla u^+(\mathbf{x}_i) - \nabla u^-(\mathbf{x}_i))|^2 \end{aligned} \quad (45)$$

where $\{\lambda_i\}_{i=1}^6 = \{1, 1, 50, 50, 10, 10\}$ are the hyperparameters for CPINN. N_{pde+} and N_{pde-} are the total number of sampled points in the upper and lower domains, respectively. N_{b+} and N_{b-} are the total number of sampled points on the upper and lower boundaries, respectively. N_I is the total number of interface points ($x > 0, y = 0$). Fig. 4d shows the sampling method for CPINNs and DEM is the same, with 4096 points in the domain (upper and lower domains combined), 256 points on each boundary, and 1000 points on the interface. We use an MLP fully connected neural network with a structure of $[2, 30, 30, 30, 30, 1]$, optimized with Adam and a learning rate of 0.001. CPINN divides the region into two areas along the crack: $y \geq 0$ and $y \leq 0$. u^+ and u^- represent different neural networks for the upper and lower regions, respectively, as shown in Fig. 4. Each neural network has 2911 trainable parameters, and both regions use the same neural network structure and optimization method.

The loss function for DEM is:

$$\mathcal{L}_{DEM} = \int_{\Omega} \frac{1}{2} (\nabla u) \cdot (\nabla u) d\Omega. \quad (46)$$

Since DEM is based on the principle of minimum potential energy, u must satisfy the essential boundary condition in advance:

$$u(\mathbf{x}) = u_p(\mathbf{x}; \boldsymbol{\theta}_p) + RBF(\mathbf{x}) \cdot u_g(\mathbf{x}; \boldsymbol{\theta}_g), \quad (47)$$

where u_p represents the particular network, and $\boldsymbol{\theta}_p$ denotes its parameters. The term RBF refers to the radial basis function, while u_g signifies the generalized network, with $\boldsymbol{\theta}_g$ being its parameters. The RBF points are uniformly distributed with an 11x11 grid, using Gaussian kernels:

$$RBF(\mathbf{x}) = \sum_{i=1}^{121} w_i \exp(-\gamma |\mathbf{x} - \mathbf{x}_i|) \quad (48)$$

where γ is a hyperparameter, set to 0.5, $|\mathbf{x} - \mathbf{x}_i|$ is the distance between \mathbf{x} and \mathbf{x}_i , and w_i is the trainable parameters of RBF. During the training process, we keep the parameters of the particular network and RBF fixed and only train the generalized neural network. It is important to highlight that the RBF equals zero at the essential boundary, and only the particular network is active at the essential boundary. The details of DEM are in Section 2.2. All network configurations and optimization methods are the same as CPINNs, with 2911 trainable parameters.

The crack problem involving a strong discontinuity across the crack surface is not feasible for BINN without using multiple networks. For simplicity, the upper-half crack plane ($y > 0$) is considered with full Dirichlet boundary conditions in BINN. To calculate the boundary integrals, a strategy demonstrated in our previous work [19] is employed in BINN. In this approach, the Gauss quadrature rule is adopted piecewise for the integrals without singularity, while regularization techniques are implemented to calculate the singular integrals. In this example, 80 source points with 800 integration points are allocated uniformly along the boundary. Then BINN is applied to solve the problem using MLP and KAN, respectively. The structures of MLP and KAN in KINN are [2,30,30,30,1] and [2,5,5,5, 1] respectively. The details of BINN are in Section 2.2.

The structure of KAN in KINN is [2,5,5,1], with only 600 parameters. We replace the MLP neural network in the above PINNs, DEM with KAN, forming the corresponding KINN versions (KAN in BINN is [2,5,5,5,1]). Fig. 5 shows the absolute error contour plots for CPINN, DEM, BINN, and KINN. The results indicate that KINN improves accuracy in terms of error distribution and maximum absolute error compared to various MLP-based numerical algorithms. Additionally, we quantify the accuracy and efficiency of KINN and traditional MLP algorithms. Table 2 shows that with fewer trainable parameters, KINN achieves lower relative \mathcal{L}_2 error:

$$\phi_{\mathcal{L}_2} = \frac{\int_{\Omega} |\phi_{pred} - \phi_{exact}|^2 d\Omega}{\int_{\Omega} |\phi_{exact}|^2 d\Omega}. \quad (49)$$

The error for BINN is the lowest, primarily because BINN directly uses the fundamental solution of the problem as the weight function, which is quite close to the analytical solution of the crack. Additionally, BINN directly imposes Dirichlet boundary conditions at the interface ($x > 0, y = 0$), and uses some technology for the numerical integration in Eq. (26). DEM and PINN, compared to BINN, implicitly contain more information about the equations. Although KINN has fewer trainable parameters, its computation time is currently about three times longer than the corresponding MLP version over 1000 epochs. This is because KAN is implemented by ourselves and does not have the optimized AI libraries like MLP, which have undergone extensive optimization. This indicates that there is still room for optimization in KAN within KINN, but it also highlights the great potential of KAN in solving PDE problems. This is because solving PDE problems essentially does not require an excessive number of parameters for simulation. Unlike fields such as computer vision and natural language processing, which require complex neural networks, solving PDEs is more akin to function fitting. The difference is that PDE solving does not provide the function directly but implicitly gives the function to be simulated through the mathematical equations of PDEs. Most function-fitting problems are not as complex as AI tasks in computer vision, so overly complex neural networks are not necessary. Therefore, the final convergence accuracy of KINN is higher than that of the MLP.

To more clearly show the training process of KINN and the corresponding algorithms, Figure Fig. 6a shows the evolution of the relative \mathcal{L}_2 error with the number of iterations. We can see that KINN converges faster and has higher accuracy compared to the corresponding MLP versions. The singularity at the crack tip is the most important characteristic of the crack problem. Considering the strain:

$$\varepsilon_{z\theta}|_{interface} = \frac{1}{r} \frac{\partial u}{\partial \theta} = \frac{1}{2\sqrt{r}} \cos\left(\frac{\theta}{2}\right)|_{\theta=0} = \frac{1}{2\sqrt{r}}, \quad (50)$$

where the interface is the line $x > 0, y = 0$. θ is from $[-\pi, \pi]$ shown in Fig. 6a. We compared PINNs, DEM, BINN, and the corresponding KINN versions. Figure Fig. 6b shows that KINN-DEM performs the best, but there is a jump in the result at $x = 0.5$. Note that the results of the six algorithms are all uniformly sampled.

Since the numerical integration scheme can affect the accuracy of DEM [45], we changed the integration scheme from Monte Carlo integration to triangular numerical integration, using 79,202 triangular integration

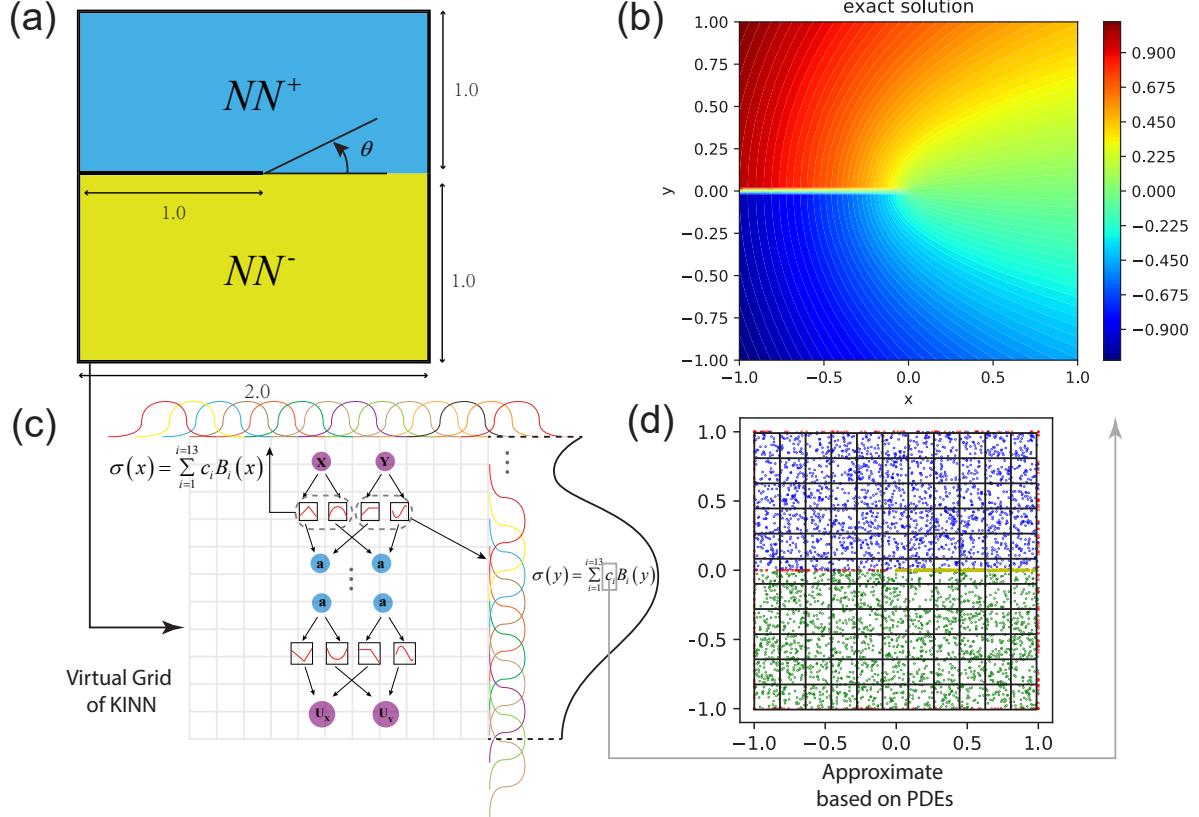


Fig. 4. Introduction to mode III crack: (a) The structure of the mode III crack, with a size of $[-1,1]^2$ in a square region. θ is from $[-\pi, \pi]$. The blue and yellow regions represent two neural networks since the displacement is discontinuous at the crack ($x < 0, y = 0$). Therefore, two neural networks are needed to fit the displacement above and below the crack. (b) The analytical solution for this problem: $u = r^{\frac{1}{2}} \sin(\theta/2)$, where r is the distance from the coordinate \mathbf{x} to the origin $x=y=0$, and θ is the angle in the polar coordinate system, with $x > 0, y = 0$ as the reference. Counterclockwise is considered a positive angle. (c) The grid distribution in KINN, with order=3 and grid size=10, is uniformly distributed in both x and y directions. (d) The meshless random sampling points for PINN, DEM, and KINN. The red points are essential boundary displacement points (256 points), the blue points are for the upper region neural network (2048 points), the green points are for the lower region neural network (2048 points), and the yellow points are the interface sampling points (1000 points).

Table 2

The comparison between different PINNs, DEM, and BINN algorithms based on MLP or KAN. Parameters mean the trained parameters in the architecture of NNs. Relative error represents the L_2 error at convergence. Grid size is the number of grids in the KAN network. Grid range is the initial range of the KAN grid. Order is the order of the B-splines in KAN. The architecture of NNs is the structure of the corresponding neural network. Parameters are the trainable parameters of the corresponding network. The time is the duration for 1000 epochs of the corresponding algorithms.

Algorithms	Relative error	Grid size	Grid range	Order	Architecture of NNs	Parameters	Time (Second, 1000 Epoch)
CPINNs_MLP	0.062535				[2, 30,30,30,30,1]	2911	11.15
KINN: CPINNs	0.029879	10	[-1,1]	3	[2, 5,5, 1]	600	39.94
DEM_MLP	0.024292				[2, 30,30,30,30,1]	2911	8.77
KINN: DEM	0.011166	10	[-1,1]	3	[2, 5,5, 1]	600	17.04
BINN_MLP	0.000872				[2, 30,30,30,30,1]	2911	3.23
KINN: BINN	0.000828	10	[-1,1]	3	[2, 5,5, 5,1]	975	14.17

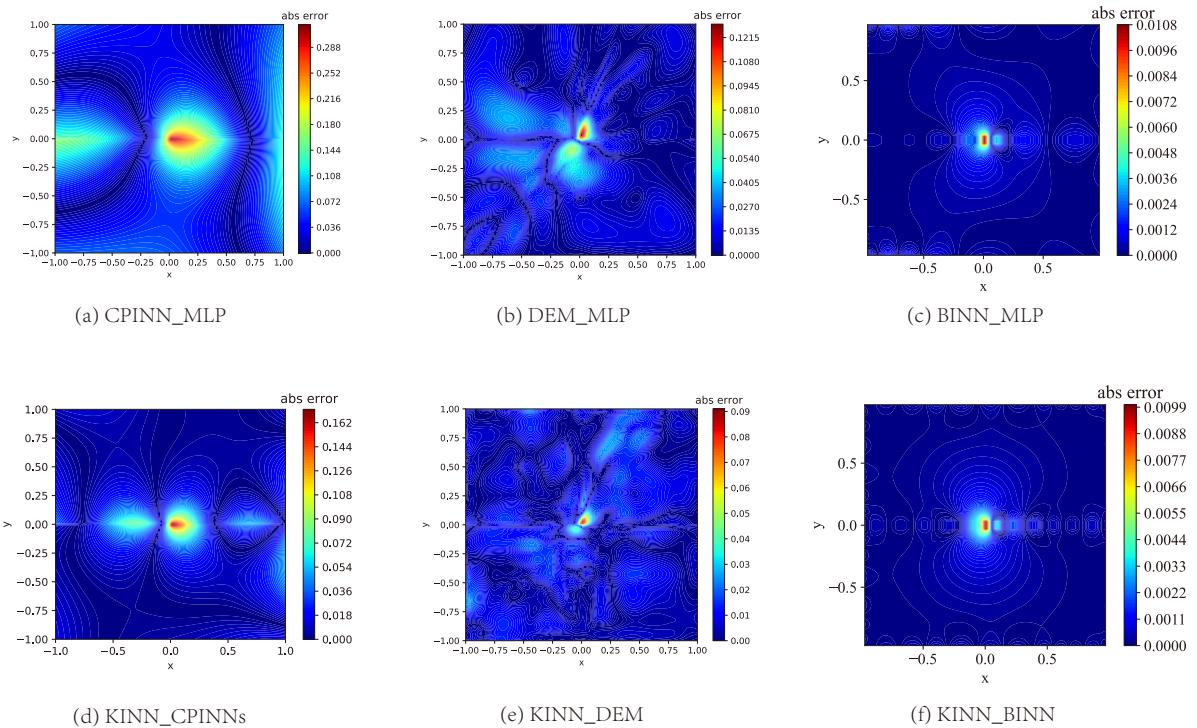


Fig. 5. The absolute error by CPINN, DEM, BINN, and KINN in mode III crack: CPINN_MLP, DEM_MLP, BINN_MLP respectively (first row), KINN_CPINN, KINN_DEM, KINN_BINN respectively (second row).

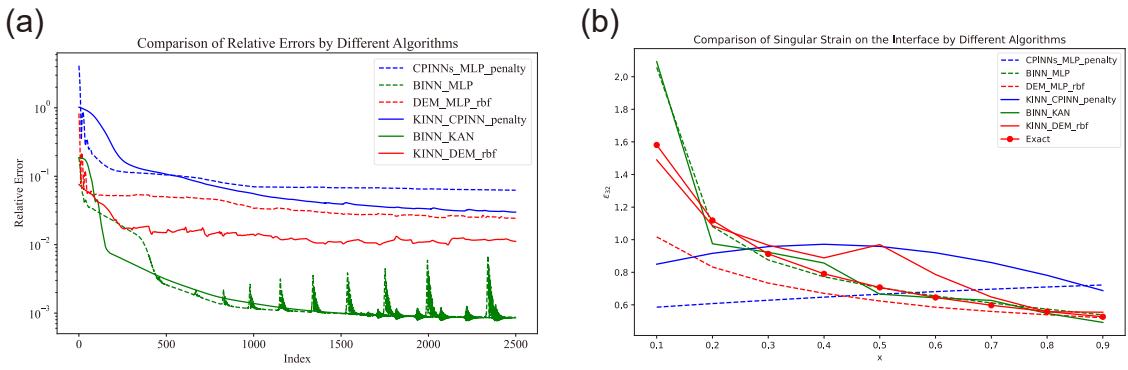


Fig. 6. The error evolution and singular displacement derivative at the interface for KINN and corresponding MLP algorithms in mode III crack: (a) Evolution of the relative \mathcal{L}_2 error. (b) Comparison of the singular displacement derivative $\partial u / \partial y$ ($\varepsilon_{z\theta}$) at $x > 0, y = 0$.

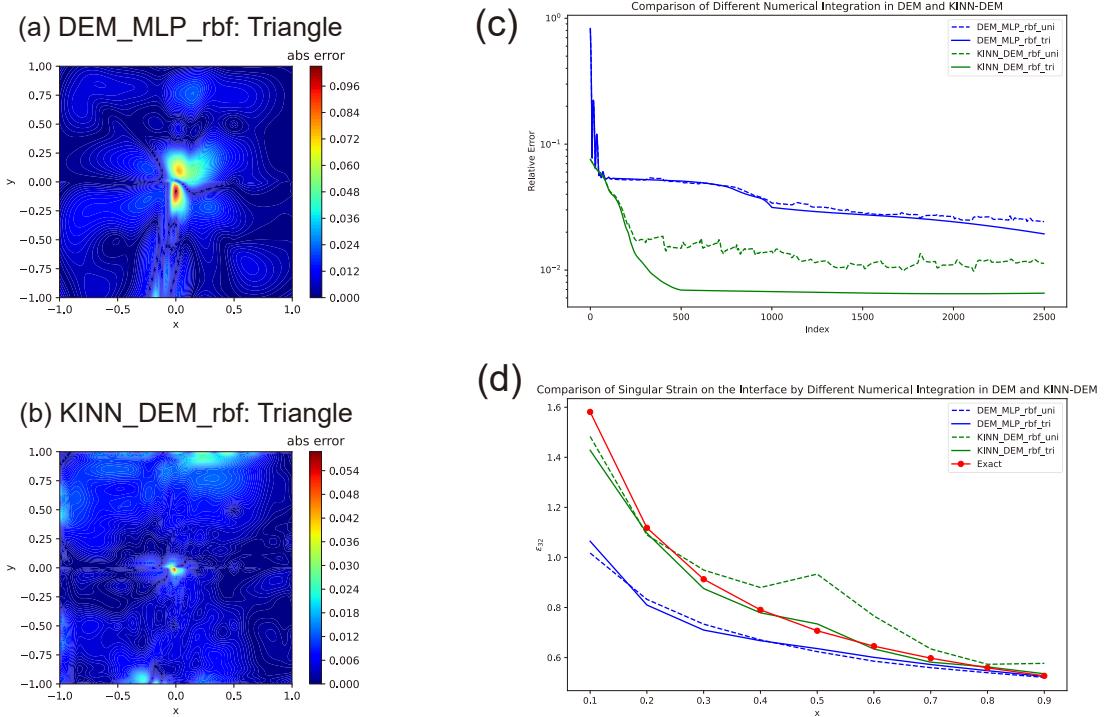


Fig. 7. Comparison of uniform Monte Carlo integration and triangular integration in KINN-DEM and KINN for mode III crack: (a) Absolute error contour plot for DEM with uniform Monte Carlo integration. (b) Absolute error contour plot for KINN-DEM with triangular grid integration. (c) Evolution of relative L_2 error for DEM with Monte Carlo and triangular integration. (d) Singular displacement derivative with Monte Carlo and triangular integration at the interface.

points. Note that the previous results were all uniformly sampled and used Monte Carlo integration to approximate the loss function in DEM. Triangular integration involves dividing the domain into triangular grids, finding the centroid of each triangle, and calculating the numerical integration based on the area of the triangles:

$$\int_{\Omega} f(\mathbf{x}) d\Omega = \sum_{i=1}^N f(\mathbf{x}_i) S_i, \quad (51)$$

where N is the total number of triangular grids, \mathbf{x}_i is the centroid of the i -th triangular grid, and S_i is the area of the corresponding triangular grid. Note that dividing the grid causes DEM to lose some meshless advantages, but the adaptability to complex geometries and simplicity of triangular grids make them easy to implement in DEM. Fig. 7 compares the triangular grid and uniform Monte Carlo integration. We can see that the absolute error contour plot for the triangular grid has smaller errors compared to Monte Carlo integration. Most importantly, we find that the triangular grid in DEM fits the singularity problem very well, especially in the KINN-DEM form, where the triangular grid almost perfectly matches the analytical solution.

Considering that KINN has some hyperparameters, such as grid size, B-spline order, and the depth and width of the KAN network, we study the impact of these hyperparameters on KINN. We use KINN-CPINNs here because the strong form by PINNs can solve almost any PDEs, making them more versatile than the energy form by DEM and the inverse form by BINN. Fig. 8 shows that with different grid sizes, the relative L_2 error in KINN-CPINNs reaches a minimum and then increases as the grid size continues to increase. This is consistent with the conclusions in the original KAN paper [33]. Excessive grid size leads to overfitting and

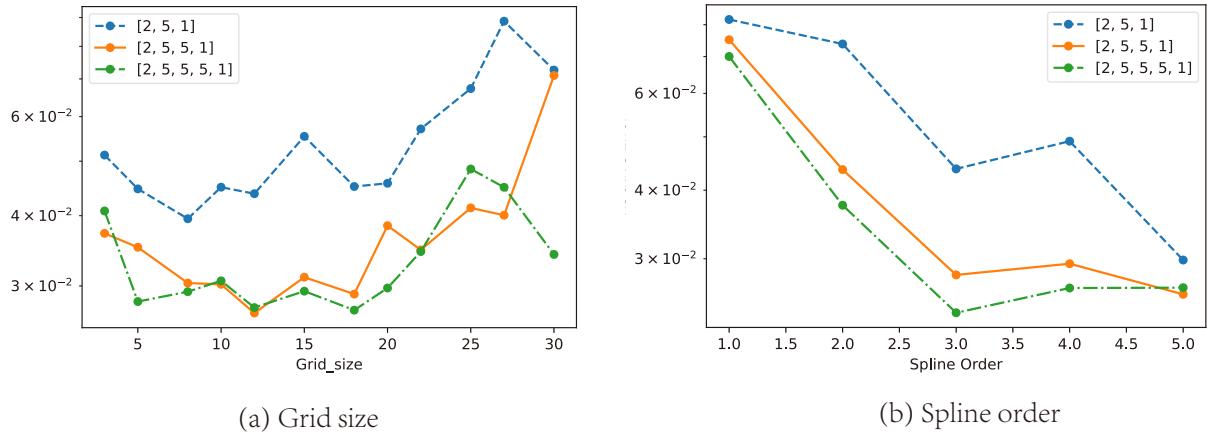


Fig. 8. Relative \mathcal{L}_2 error for KINN-CPINNs in mode III crack in different architectures of KAN with different grid sizes (a) and different B-spline orders (b).

an unsMOOTH fitted function. The grid size should be proportional to the complexity of the fitting function in PDE problems, meaning that the grid size should increase with the complexity of the function. Additionally, we fixed the grid size at 10 and adjusted the order of the B-spline. We found that increasing the spline order reaches a plateau, where further increases do not improve model accuracy. This is similar to the element order in FEM. Unfortunately, KINN does not currently exhibit grid convergence like FEM, where the grid size can be increased indefinitely. We believe that exploring this aspect further is an important direction for future research on KINN.

4.2.2. Stress concentration

The plate with a central hole is a commonly used benchmark in solid mechanics, as shown in Fig. 9. This problem is not only common in engineering but also features stress concentration near the hole. The geometry of the plate with a central hole has an original size of 40x40 mm, with a hole at the center and a radius of 5 mm. The left and right sides are subjected to a force of $t_x = 100$ N/mm, with an elastic modulus $E = 1000$ MPa and a Poisson's ratio $\nu = 0.3$. Due to symmetry, we simulate only the upper right quarter, with an edge length $L = 20$ mm. The boundary conditions are $u_x = 0$ at $x = 0$ and $u_y = 0$ at $y = 0$, with other boundaries subjected to force boundary conditions. We use a plane stress model, as shown in Fig. 9a.

As in the previous example, we use the strong form by PINNs, energy form by DEM, and inverse form by BINN to solve this problem, and then compare the results by MLP with the corresponding versions of KINN. For PINNs and DEM, we use the same collocation scheme with 75,298 points in the domain and 1,000 points

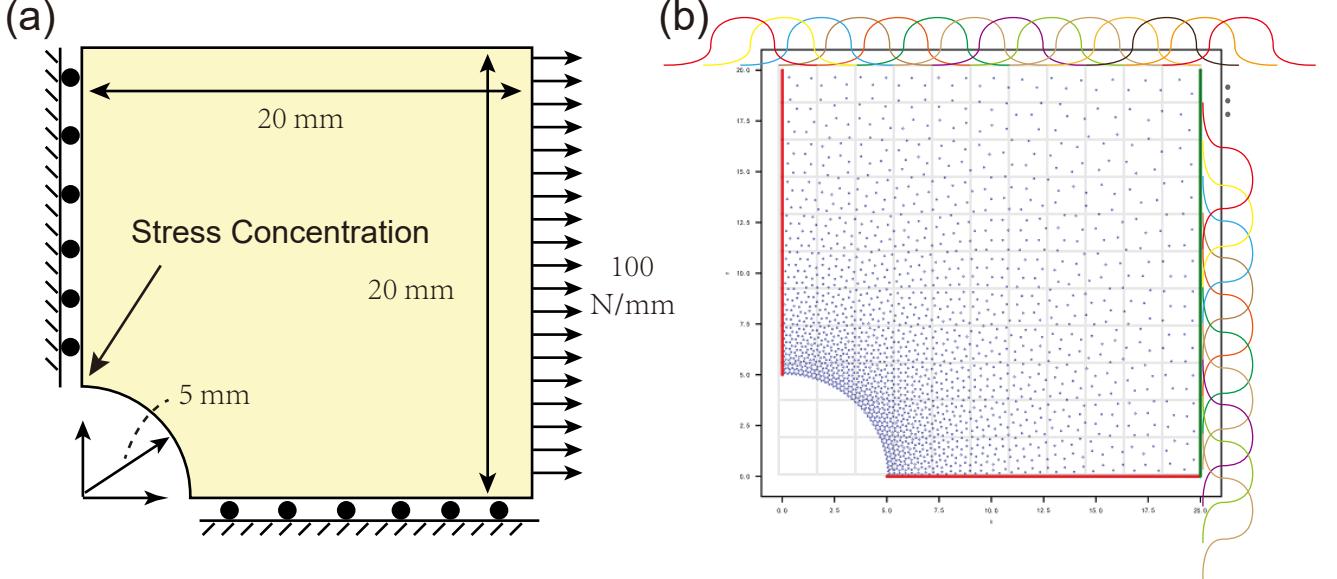


Fig. 9. The introduction of plate with central hole: (a) The geometry of the plate with a central hole, with a hole radius of 5 mm and a square plate with an edge length of 20 mm. The right side is subjected to a uniform load of 100 N/mm. (b) The grid distribution in KINN, with order=3 and grid size=10, is uniformly distributed in both x and y directions.

on each boundary (a total of 5 boundaries). The loss function for PINNs is:

$$\begin{aligned} \mathcal{L}_{pinns} = & \lambda_1 \sum_{i=1}^{N_{pde}} [|\sigma(\mathbf{x}_i)_{xx,x} + \sigma(\mathbf{x}_i)_{xy,y}|^2 + |\sigma(\mathbf{x}_i)_{yx,x} + \sigma(\mathbf{x}_i)_{yy,y}|^2] \\ & + \lambda_2 \sum_{i=1}^{N_{left}} |u_x(\mathbf{x}_i)|^2 + \lambda_3 \sum_{i=1}^{N_{left}} |\sigma_{xy}(\mathbf{x}_i)|^2 \\ & + \lambda_4 \sum_{i=1}^{N_{down}} |u_y(\mathbf{x}_i)|^2 + \lambda_5 \sum_{i=1}^{N_{down}} |\sigma_{xy}(\mathbf{x}_i)|^2 \\ & + \lambda_6 \sum_{i=1}^{N_{up}} [|\sigma_{yy}(\mathbf{x}_i)|^2 + |\sigma_{xy}(\mathbf{x}_i)|^2] \\ & + \lambda_7 \sum_{i=1}^{N_{right}} [|\sigma_{xx}(\mathbf{x}_i) - t_x|^2 + |\sigma_{xy}(\mathbf{x}_i)|^2] \\ & + \lambda_8 \sum_{i=1}^{N_{circle}} [|\sigma_{xx}(\mathbf{x}_i)n_x + \sigma_{xy}(\mathbf{x}_i)n_y|^2 + |\sigma_{yx}(\mathbf{x}_i)n_x + \sigma_{yy}(\mathbf{x}_i)n_y|^2], \end{aligned} \quad (52)$$

$$\sigma_{ij} = \frac{E}{1+\nu} \varepsilon_{ij} + \frac{\nu}{1+\nu} \varepsilon_{kk} \delta_{ij} \quad (53)$$

$$\varepsilon_{ij} = \frac{1}{2}(u_{i,j} + u_{j,i}) \quad (54)$$

We can observe that the loss function of PINNs in the strong form is very complex and has numerous hyperparameters. Although there are some techniques for adjusting hyperparameters [46, 42, 47], it is a challenge to get the optimal hyperparameters. Therefore, we manually set the hyperparameters $\{\lambda_i\}_{i=1}^8 =$

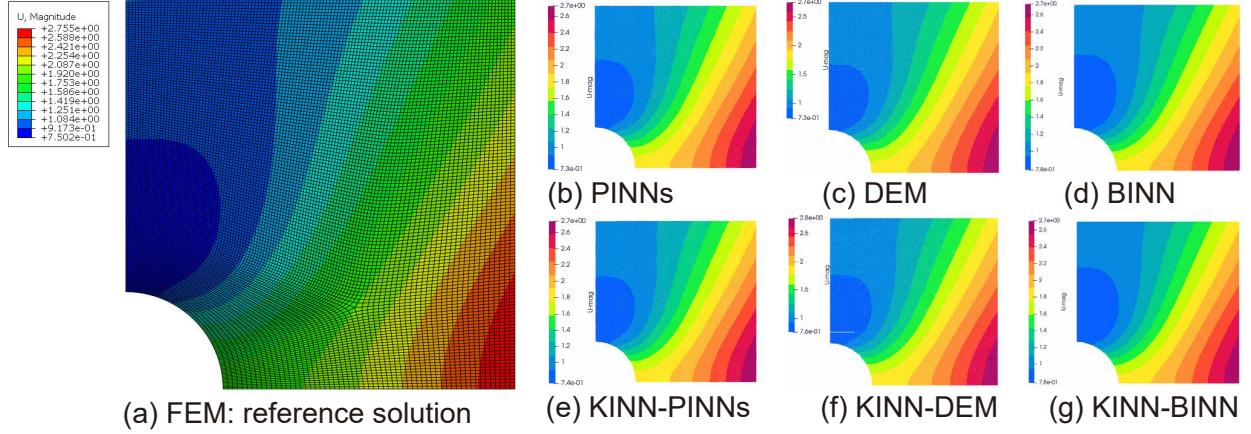


Fig. 10. Predicted displacement solutions for PINNs, DEM, BINN, and their corresponding KINN versions: (a) FEM reference solution, (b) PINNs, (c) DEM, (d) BINN, (e) KINN-PINNs, (f) KINN-DEM, (g) KINN-BINN.

$\{10, 2000, 1, 1, 2000, 1, 1, 1\}$. We use a neural network to approximate the displacement field:

$$\mathbf{u}(\mathbf{x}) \approx \mathbf{u}\left(\frac{\mathbf{x}}{L}; \boldsymbol{\theta}\right), \quad (55)$$

where $\boldsymbol{\theta}$ are the neural network parameters.

For DEM, the loss function is:

$$\begin{aligned} \mathcal{L}_{DEM} &= \int_{\Omega} \Psi d\Omega - \int_{\Gamma^{right}} t_x u_x d\Gamma, \\ \Psi &= \frac{1}{2} \sigma_{ij} \varepsilon_{ij}, \\ u_x &= \mathbf{x} \cdot \mathbf{u}_x \left(\frac{\mathbf{x}}{L}; \boldsymbol{\theta}\right), \\ u_y &= \mathbf{y} \cdot \mathbf{u}_y \left(\frac{\mathbf{x}}{L}; \boldsymbol{\theta}\right), \end{aligned} \quad (56)$$

We can see that DEM has no hyperparameters in the loss function and involves lower-order derivatives than strong form by PINNs, but it requires the displacement field to satisfy essential boundary conditions in advance. Here, we construct the admissible displacement field by multiplying the coordinates. Additionally, DEM requires the integration of internal energy and external work. As shown in Fig. 7, triangular integration is more accurate for DEM than Monte Carlo integration, so we use triangular grid integration for this example instead of Monte Carlo integration in DEM.

Fig. 10 shows the predicted absolute displacement ($u_{mag} = \sqrt{u_x^2 + u_y^2}$) for PINNs, DEM, BINN, and the corresponding KINN versions. FEM is used as the reference solution. The FEM mesh is shown in Fig. 10a. It uses 15,300 eight-node quadratic elements with reduced integration (CPS8R). We performed a mesh convergence analysis for the FEM solution, so it is reliable. The FEM computation time is 30.64s. Our numerical experiments demonstrated that using scale normalization techniques in Appendix D allows us to solve problems with a scale significantly greater than 1 (e.g., a plate with a hole having a characteristic size of 20).

Due to the stress concentration near the hole, traditional FEM requires a denser mesh near the stress concentration to ensure simulation accuracy. Traditionally, PINNs and DEM with MLP often perform poorly near the stress concentration in Monte Carlo integration. Therefore, we replace the MLP in PINNs and DEM with KAN to see if there is any improvement. The plate with a central hole exhibits the maximum u_y and the most significant stress concentration along the line $x = 0$, while the maximum u_x occurs along the line $y = 0$.

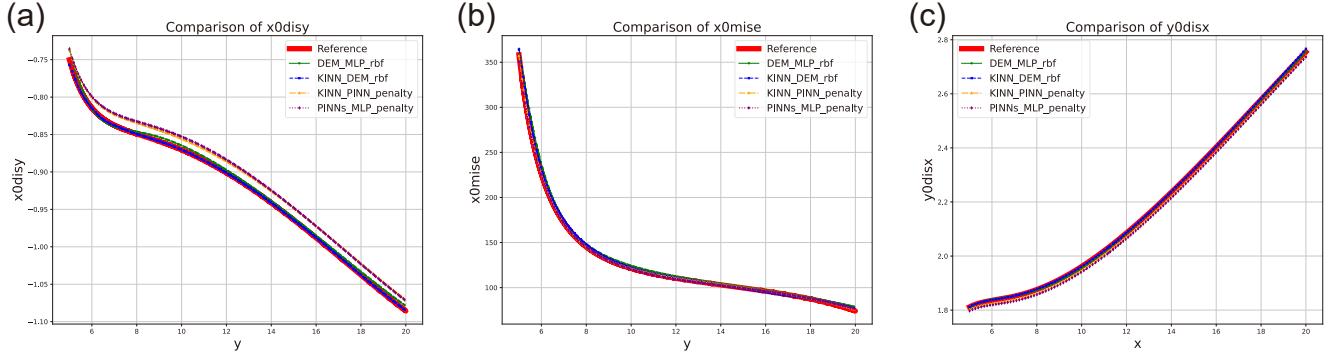


Fig. 11. Displacement and Mises stress along $x = 0$ and $y = 0$ for PINNs, DEM, BINN, and their corresponding KINN versions: (a) u_y along $x = 0$, (b) Mises stress along $x = 0$, (c) u_x along $y = 0$.

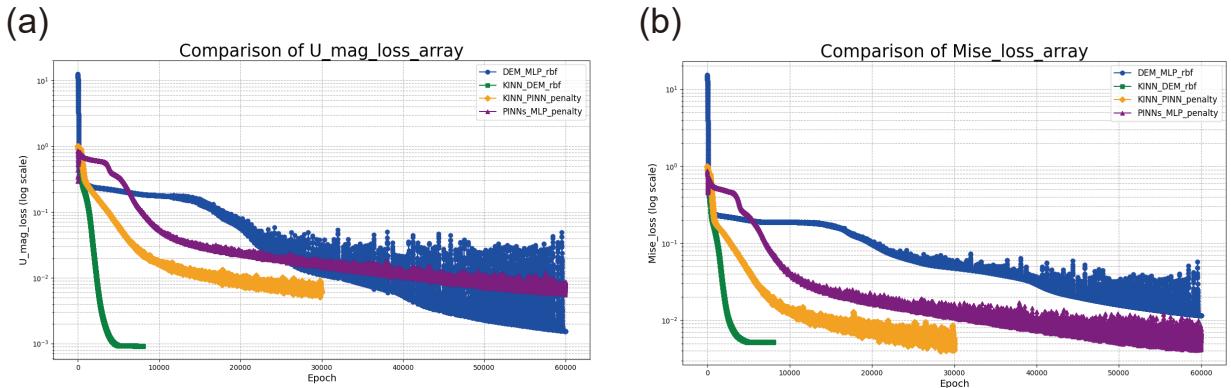


Fig. 12. Evolution of the relative \mathcal{L}_2 and \mathcal{H}_1 errors for PINNs, DEM, BINN, and their corresponding KINN versions: (a) Evolution of the relative \mathcal{L}_2 displacement error, (b) Evolution of the relative \mathcal{H}_1 error in Mises stress.

Thus, we compare the different algorithms in terms of u_y and Mises stress along $x = 0$ and u_x along $y = 0$. Fig. 11 shows that KINN-DEM achieves the highest accuracy, perfectly matching the reference solution. Fig. 12 shows the relative errors in displacement and Mises stress, indicating that KINN-DEM achieves almost the same accuracy as FEM. The most surprising thing is that the convergence speed of KINN-DEM is extremely fast.

In terms of efficiency, Table 3 shows the accuracy and efficiency of various algorithms.

4.2.3. Nonlinear PDEs

This example aims to verify the effectiveness of KINN in solving nonlinear problems. We consider a hyperelasticity problem [45]. Hyperelasticity problems are nonlinear due to the nonlinearity of both constitutive law and geometric equations. We consider the Neo-Hookean constitutive model for the hyperelasticity problem:

$$\begin{aligned} \Psi &= \frac{1}{2}\lambda(\ln J)^2 - \mu \ln J + \frac{1}{2}\mu(I_1 - 3) \\ J &= \det(\mathbf{F}) \\ I_1 &= \text{trace}(\mathbf{C}) \\ \mathbf{C} &= \mathbf{F}^T \mathbf{F} \end{aligned} \tag{57}$$

Table 3
Comparison of accuracy and errors for various algorithms in the plate with a central hole

Algorithms	Dis relative error (\mathcal{L}_2)	Mise relative error (\mathcal{H}_1)	Grid size	grid_range	order	Architecture of NNs	Parameters	Time (Second, 1000 Epoch)
PINNs_MLP	0.00858	0.00649				[2, 30,30,30,30,2]	2942	70.65
KINN-PINNs	0.00629	0.00481	15	[0,1]	3	[2, 5,5,5 2]	1400	1622
DEM_MLP	0.00154	0.0115				[2, 30,30,30,30,2]	2942	14.57
KINN-DEM	0.000919	0.00517	15	[0,1]	3	[2, 5,5,5 2]	1400	310
BINN-MLP	0.000545	0.00434				[2, 30,30,30,30,2]	2942	5.4
KINN-BINN	0.000226	0.00211	10	[0,20]	3	[2, 5,5,5 2]	1400	21.3

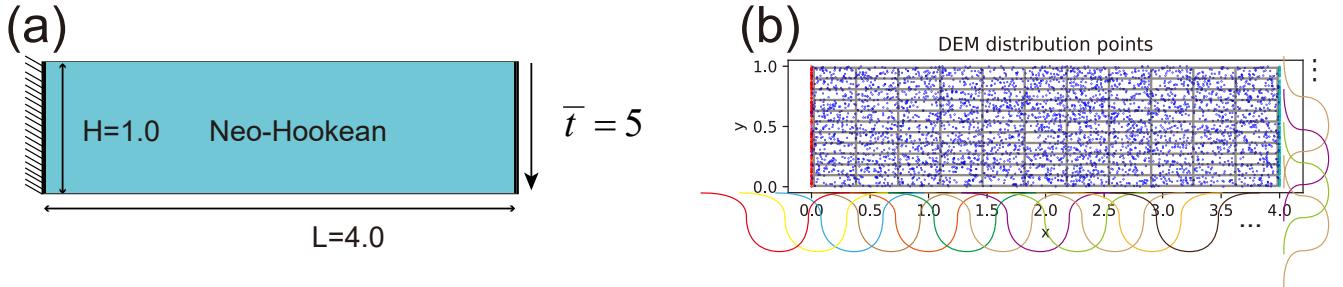


Fig. 13. Description of the Neo-Hookean hyperelastic cantilever beam problem: (a) Dimensions of the cantilever beam, height $H = 1.0$, length $L = 4.0$, $E = 1000$, and $\nu = 0.3$. The Neo-Hookean hyperelastic constitutive model is used. The left end $x = 0.0$ is fixed, and a uniformly distributed downward load $\bar{t} = 5$ is applied at the right end. (b) DEM point distribution: blue points are domain points, red points are essential boundary condition points, and green points are surface traction points.

where Ψ is the Neo-Hookean strain energy density function, and λ and μ are the Lamé parameters:

$$\begin{cases} \lambda = \frac{\nu E}{(1+\nu)(1-2\nu)} \\ \mu = \frac{E}{2(1+\nu)} \end{cases} \quad (58)$$

where E and ν are the elastic modulus and Poisson's ratio, respectively. \mathbf{C} is the Green tensor, and \mathbf{F} is the deformation gradient:

$$\mathbf{F} = \frac{\partial \mathbf{x}}{\partial \mathbf{X}} \quad (59)$$

where \mathbf{x} is the spatial coordinate, and \mathbf{X} is the material coordinate. The relationship between the spatial and material coordinates is $\mathbf{x} = \mathbf{X} + \mathbf{u}$, where \mathbf{u} is the displacement field. Since this problem has an energy form, it can be solved using DEM. Notably, solving this problem in the strong form using PINNs is complex and less efficient than DEM, as the solution order of PINNs is higher than that of DEM. For detailed proof, refer to [39]. Therefore, we only consider the energy form by DEM for this problem. The loss function of DEM is:

$$\mathcal{L} = \int_{\Omega} (\Psi - \mathbf{f} \cdot \mathbf{u}) d\Omega - \int_{\Gamma^t} \bar{\mathbf{t}} \cdot \mathbf{u} d\Gamma \quad (60)$$

where \mathbf{f} is the body force, and $\bar{\mathbf{t}}$ is the surface traction on the boundary Γ^t .

We use MLP and KAN networks to solve the benchmark 2D cantilever beam plane stress problem, as shown in Fig. 13. Since solving this problem in the strong form using PINNs is inconvenient, and BINN does not have a fundamental solution, we only use DEM to solve this problem.

For DEM, we adopt different integration methods, including Monte Carlo, trapezoidal rule, and Simpson's rule. We compare these integration schemes using KINN and DEM_MLP. We uniformly distribute 200 points in the length direction and 50 points in the width direction, resulting in 10,000 points in total.

We use the same MLP and KAN configurations as in the previous numerical examples, except for the network types. We use the LBFGS second-order optimization algorithm. Table 4 shows the accuracy and efficiency between MLP and KAN. Our reference solution is obtained using FEM. Due to stress singularities at the corners $(0,0)$ and $(0,1)$, we only compare the L_2 displacement error without considering the H_1 stress

Table 4
Accuracy and error comparison of various algorithms for the hyperelastic problem

Algorithms	Displacement Relative Error (\mathcal{L}_2)	Grid Size	Grid Range	Order	NN Architecture	Parameters	Time (Seconds, 100 Epochs)
DEM_MLP_mc	0.04756				[2, 30,30,30,30,2]	2942	37.34
KINN-DEM_mc	0.03334	15	[0,1]	3	[2, 5,5,5,2]	1400	96.44
DEM_MLP_trap	0.01524				[2, 30,30,30,30,2]	2942	35.76
KINN-DEM_trap	0.002536	15	[0,1]	3	[2, 5,5,5,2]	1400	99.01
DEM_MLP_simp	0.01753				[2, 30,30,30,30,2]	2942	39.87
KINN-DEM_simp	0.002181	15	[0,1]	3	[2, 5,5,5,2]	1400	104.4

error. We find that KINN improves accuracy under different integration schemes, and DEM requires higher precision in numerical integration. The error of Monte Carlo integration is larger than that of Simpson's and trapezoidal rule. Although Simpson's rule has higher polynomial accuracy than the trapezoidal rule, it does not significantly improve DEM accuracy. Therefore, DEM mainly requires two things: replacing MLP with KAN and using low-order numerical integration such as triangular integration or the trapezoidal rule, without necessarily using higher-order schemes like Simpson's or Gaussian integration. Importantly, DEM should not use Monte Carlo integration due to its low convergence. KAN in DEM does not need higher-order integration because KAN can adapt the local compact space shape to fit the target function, reducing the need for precise numerical integration. Empirically, DEM with Monte Carlo integration performs poorly. Although KAN has fewer trainable parameters than MLP, KAN takes longer to train for 100 epochs.

The y-direction displacement u_y is the most critical field function under the loading condition in the cantilever beam problem. Fig. 14 shows the absolute error contour plots of u_y for MLP and KAN in DEM. In any numerical integration method, KINN significantly reduces the maximum error and error distribution compared to MLP. Fig. 15 shows the evolution of the L_2 relative error, considering the absolute displacement $u_{mag} = \sqrt{u_x^2 + u_y^2}$.

Despite the stress singularity at the corners, we compare the absolute displacement and Von Mises stress along $x = 2$ and $y = 0.5$. Fig. 16 shows the predictions of absolute displacement and Von Mises stress for KINN and DEM_MLP under the three numerical integration methods. We observe that replacing MLP with KAN improves accuracy significantly for all numerical integrations.

4.2.4. Heterogeneous problem

Heterogeneous problems are commonly encountered in computational mechanics, characterized by the combination of materials with different properties, as shown in Fig. 17a. The section demonstrates whether KAN has an advantage over MLP in solving problems with materials of different properties.

The motivation for using KAN to solve heterogeneous materials is that KAN is based on spline interpolation, and splines are piecewise functions. On the other hand, since the heterogeneous problem is mathematically continuous in the original function but discontinuous in the derivative, KAN's piecewise function property theoretically has a natural advantage in solving such problems. Therefore, in this section, we experiment with KAN to solve heterogeneous material problems.

First, we consider a fitting problem to see if KAN has higher accuracy and convergence speed compared to MLP. We consider:

$$u(\mathbf{x}) = \begin{cases} \frac{r^4}{a_1} & r < r_0 \\ \frac{r^4}{a_2} + r_0^4 \left(\frac{1}{a_1} - \frac{1}{a_2} \right) & r \geq r_0 \end{cases} \quad (61)$$

where r is the distance from point \mathbf{x} to the origin $(0,0)$, $a_1 = 1/15$ and $a_2 = 1$, $r_0 = 0.5$, and the solution domain is the square region $[-1, 1]^2$. Our approach is simple: fit this piecewise function using KAN and MLP to observe whether KAN has a stronger fitting ability for piecewise functions. Fig. 18 shows the fitting capabilities of KAN and MLP for this piecewise function at different epochs, clearly demonstrating that KAN has a stronger fitting ability for piecewise functions compared to MLP. Here, the network structures for both KAN and MLP are: MLP is structured as [2,100,100,1], and using the tanh activation function, KAN's structure is [2,5,5,1], with a grid size of 20, grid range within the function interval, i.e., [-1,1], and spline order of 3. Except for the neural network structure, all other parameters are the same, such as learning rate and optimizer. It is worth

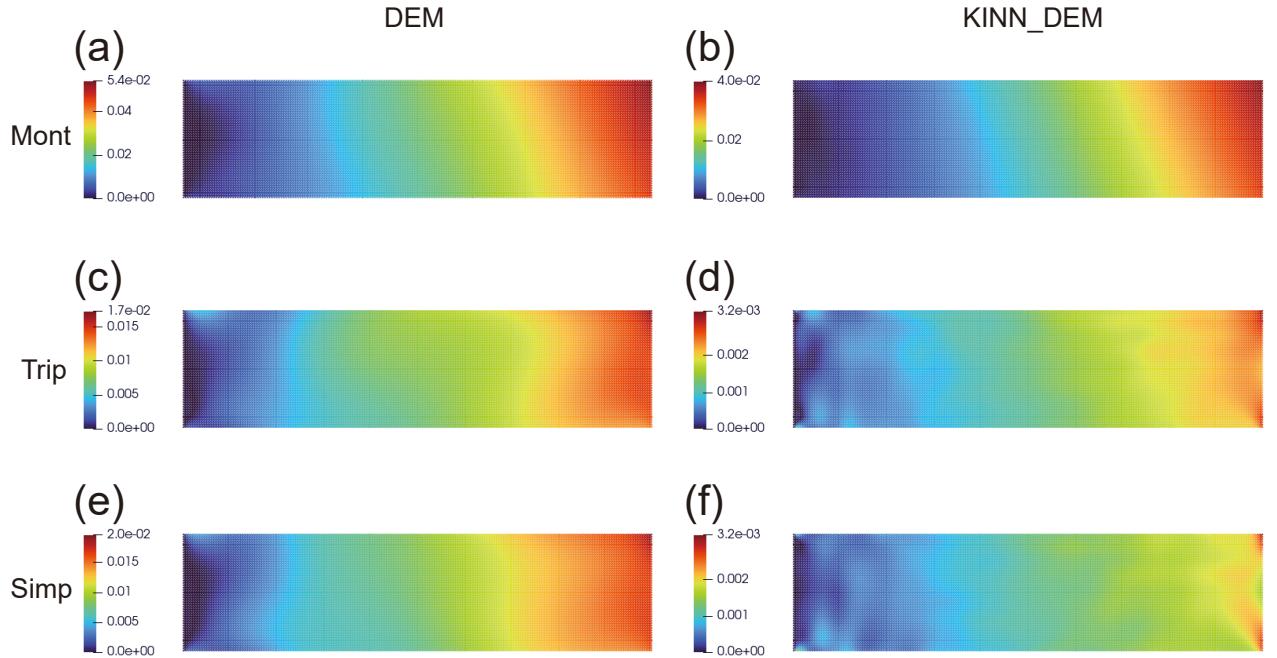


Fig. 14. Absolute error contours of the Neo-Hookean hyperelastic cantilever beam: (a,c,e) Absolute error contours for DEM with MLP, (b,d,f) Absolute error contours for DEM with KAN. (a, b), (c,d), and (e,f) correspond to Monte Carlo, trapezoidal, and Simpson's numerical integrations, respectively.

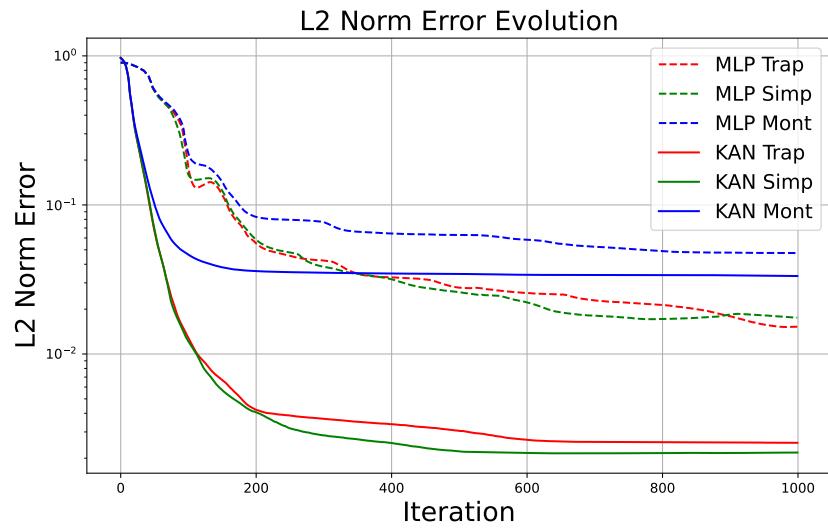


Fig. 15. Evolution of the L_2 relative error for the Neo-Hookean hyperelastic cantilever beam.

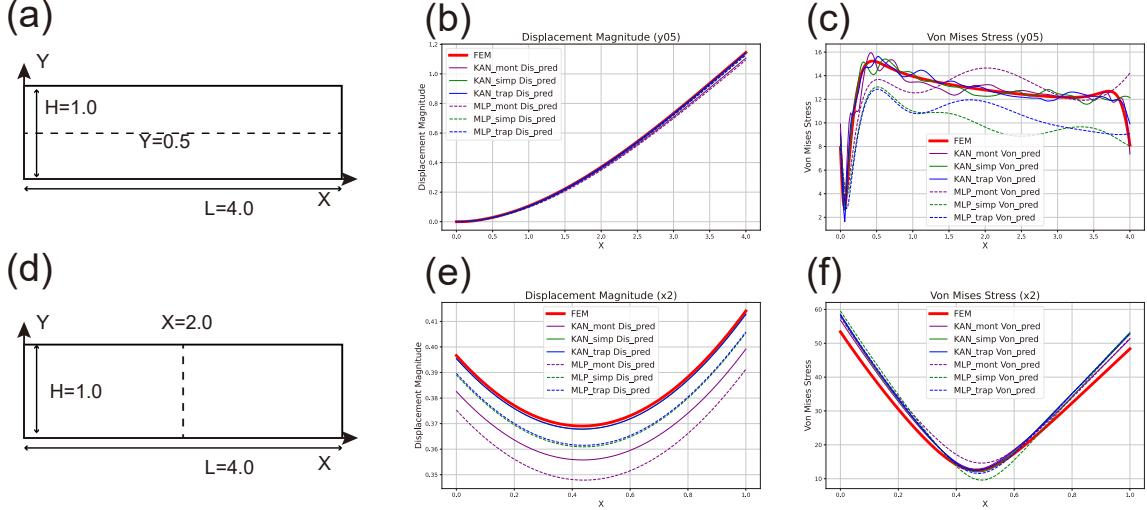


Fig. 16. Absolute displacement and Von Mises stress at $x = 2$ and $y = 0.5$ for the Neo-Hookean hyperelastic cantilever beam. The first row compares the absolute displacement (a) and Von Mises stress (b) along $y = 0.5$. The second row compares the absolute displacement (d) and Von Mises stress (e) along $x = 2.0$.

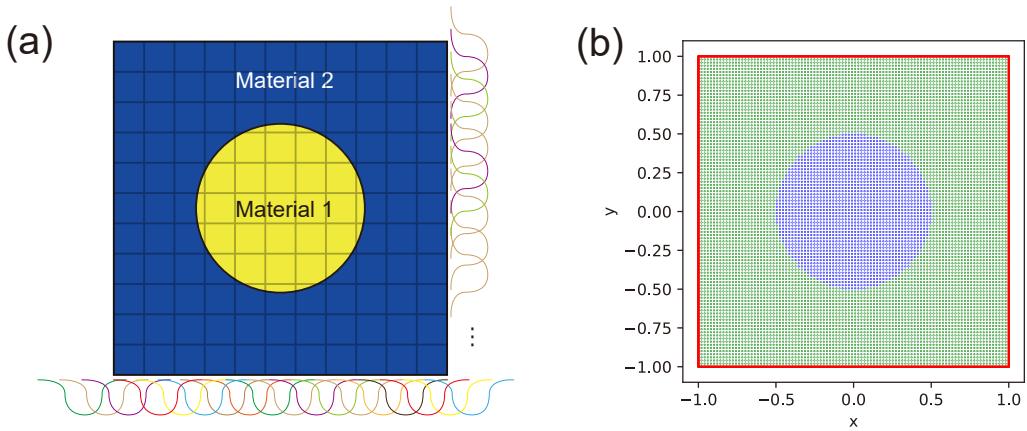


Fig. 17. Illustration of the heterogeneous problem: (a) Two different materials, with continuous displacement but discontinuous strain at the interface. (b) Point sampling illustration, with blue and green points in different regions and red points at the Dirichlet boundary condition.

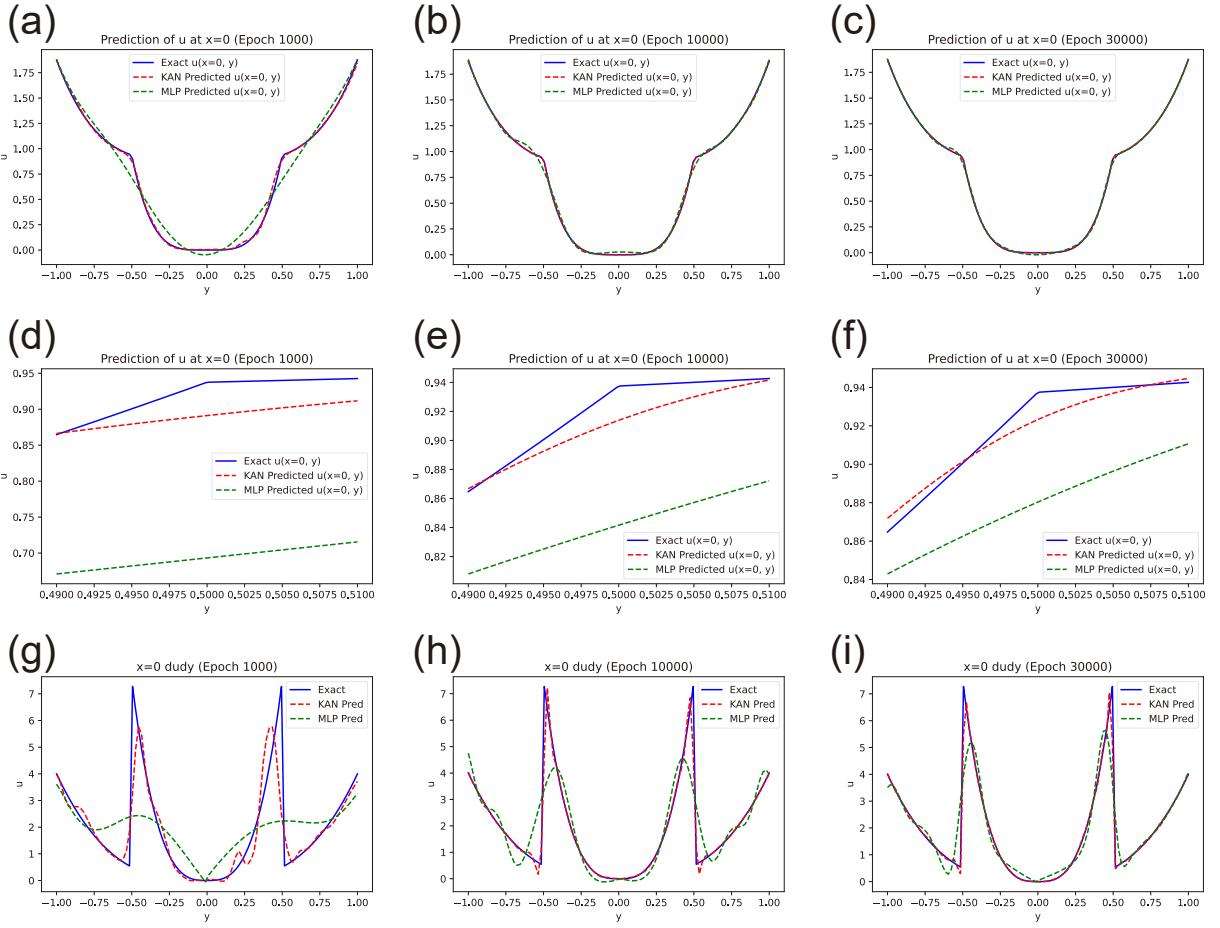


Fig. 18. Fitting curves for the heterogeneous problem at $x = 0$: MLP and KAN fitting the original function u at epoch=1000 (a), 10000 (b), and 30000 (c). MLP and KAN fitting the original function u near the interface at epoch=1000 (d), 10000 (e), and 30000 (f). MLP and KAN fitting the derivative du/dy at epoch=1000 (g), 10000 (h), and 30000 (i).

noting that considering the target function's C_0 continuity, we changed the MLP activation function from tanh to ReLU, which is more suited to the target function, but the accuracy still did not surpass KAN.

It is evident that KAN has the potential to be more accurate and faster for heterogeneous problems compared to MLP. Therefore, we explore its potential in solving heterogeneous PDE problems. We consider a simple Poisson problem for validation:

$$\begin{cases} a_1 \Delta u(\mathbf{x}) = 16r^2 & r < r_0 \\ a_2 \Delta u(\mathbf{x}) = 16r^2 & r \geq r_0 \\ u(\mathbf{x}) = \frac{r^4}{a_2} + r_0^4 \left(\frac{1}{a_1} - \frac{1}{a_2} \right) & \text{Boundary: } x = \pm 1, y = \pm 1 \end{cases} \quad (62)$$

The boundary condition is a pure Dirichlet boundary condition on the four edges of the square ($x = \pm 1, y = \pm 1$), with parameters the same as in Eq. (61). However, here we solve the function using PDEs instead of data-driven methods. The point sampling method is shown in Fig. 17b.

It is worth noting that subdomain forms of PINNs, such as CPINNs [44] and the energy form of DEM, CENN [39], require subdomain division. Traditional PINNs and DEM without subdomain division often perform poorly in solving heterogeneous problems based on MLP, as we will demonstrate in the following numerical examples. In the subdomain form, different regions are approximated by different neural networks, allowing interface

information to be added to the training process, and aiding neural network training. In the subdomain form, if we divide the subdomains along the material interface, we need to consider the following equations:

$$\begin{cases} a_1 \Delta u^+(\mathbf{x}) = 16r^2 & r < r_0 \\ a_2 \Delta u^-(\mathbf{x}) = 16r^2 & r \geq r_0 \\ \bar{u}(\mathbf{x}) = \frac{r^4}{a_2} + r_0^4 \left(\frac{1}{a_1} - \frac{1}{a_2} \right) & \text{Boundary: } x = \pm 1, y = \pm 1 \\ \mathbf{n} \cdot (a_1 \nabla u^+(\mathbf{x})) = \mathbf{n} \cdot (a_2 \nabla u^-(\mathbf{x})) & r = r_0 \\ u^+(\mathbf{x}) = u^-(\mathbf{x}) & r = r_0 \end{cases} \quad (63)$$

This is the strong form of the PDE. In CPINNs, the $u^+(\mathbf{x})$ in the region $r < r_0$ is approximated by one neural network, and the $u^-(\mathbf{x})$ in the region $r \geq r_0$ by another. CPINNs require two additional interface equations: the flux continuity condition $\mathbf{n} \cdot (a_1 \nabla u^+(\mathbf{x})) = \mathbf{n} \cdot (a_2 \nabla u^-(\mathbf{x}))$ and the original function continuity condition $u^+(\mathbf{x}) = u^-(\mathbf{x})$, introducing extra hyperparameters. In CENN, if we use the distance function from Eq. (16) to construct the admissible function that satisfies the essential boundary conditions, we only need to consider:

$$\begin{aligned} \mathcal{L}_{CENN} = & \frac{1}{2} \int_{\Omega^+} a_1 (\nabla u^+(\mathbf{x})) \cdot (\nabla u^+(\mathbf{x})) d\Omega + \frac{1}{2} \int_{\Omega^-} a_2 (\nabla u^-(\mathbf{x})) \cdot (\nabla u^-(\mathbf{x})) d\Omega \\ & + \beta \int_{\Gamma^{inter}} [u^+(\mathbf{x}) - u^-(\mathbf{x})]^2 d\Gamma \end{aligned} \quad (64)$$

where Γ^{inter} is the interface, i.e., $r = r_0$. Compared to the strong form, the interface flux continuity equation is not needed, only the original function continuity condition, as proven in [39].

Both CPINNs and CENN require subdomain division. The advantage of subdomain division is improved accuracy in solving heterogeneous problems by adding inductive bias. The disadvantage is increased computational cost due to additional interface information and more hyperparameters. We note that KAN has the nature of piecewise functions and theoretically is more suitable for solving heterogeneous problems, as the function in heterogeneous problems are piecewise, as shown in Fig. 18. Thus, we naturally propose whether using KAN instead of MLP can achieve good results without subdomains.

If no subdomain division is used, it is an advantage that approximating the entire region with one neural network does not require interface equations. Here we only verify the energy form of PINNs. When using one neural network to approximate the entire region, we only need to consider the following optimization function:

$$\mathcal{L} = \frac{1}{2} \int_{\Omega^+} a_1 (\nabla u) \cdot (\nabla u) d\Omega + \frac{1}{2} \int_{\Omega^-} a_2 (\nabla u) \cdot (\nabla u) d\Omega + \beta \int_{\Gamma^u} [u(\mathbf{x}) - \bar{u}(\mathbf{x})]^2 d\Gamma \quad (65)$$

Here we use a penalty function to satisfy the boundary conditions Γ^u . We conduct numerical experiments to verify, using DEM to solve this problem with both MLP and KAN, exploring whether KAN can achieve good results for heterogeneous problems without subdomain division. Fig. 19 compares DEM using KAN and MLP, clearly showing that KAN achieves good results without subdomain division for heterogeneous problems, whereas MLP does not. MLP struggles to bend the function at the interface $r = 0.5$. Note that KAN is a smooth function, so it still cannot perfectly fit near the interface, as shown in Figure Fig. 19e. However, compared to MLP, Fig. 19f clearly shows that KAN can better capture the interface information.

In conclusion, the numerical experiments show that KAN has better potential than MLP for solving heterogeneous problems without subdomain division. Of course, we can still use CENN or CPINNs with subdomain division to solve this problem, and we believe the accuracy and convergence speed will be even better.

4.3. KAN does not work well on complex geometries

In this example, we demonstrate the performance of KINN on complex geometries, as shown in Fig. 20. We solve the Laplace equation, with all boundary conditions being Dirichlet boundary conditions:

$$\begin{aligned} \Delta u(\mathbf{x}) &= 0, x \in \Omega \\ u(\mathbf{x}) &= \bar{u}(\mathbf{x}), x \in \Gamma \end{aligned} \quad (66)$$

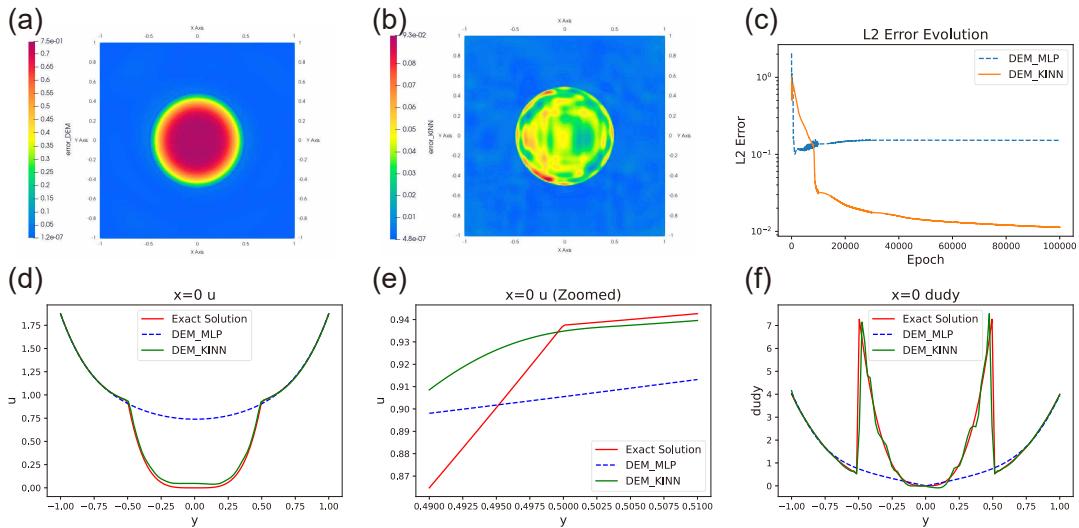


Fig. 19. PDEs solving for heterogeneous problems, comparing DEM with MLP and KAN without subdomains. (a) Absolute error contour of DEM using MLP. (b) Absolute error contour of DEM using KAN. (c) Comparison of the relative error \mathcal{L}_2 evolution trends between DEM using MLP and KAN. (d) Comparison of the predicted u and exact solution at $x = 0$ between DEM using MLP and KAN. (e) Zoomed comparison of the predicted u and exact solution near the interface $x = 0, y = 0.5$ between DEM using MLP and KAN. (f) Comparison of the predicted du/dy and exact solution at $x = 0$ between DEM using MLP and KAN.

The analytical solution is:

$$u(\mathbf{x}) = \sin(x) \sinh(y) + \cos(x) \cosh(y) \quad (67)$$

Note that the boundary condition $\bar{u}(\mathbf{x})$ is derived from the analytical solution. We solve the Koch snowflake and flower problems, as shown in Fig. 20. The analytical solutions and PDEs for these geometries are consistent.

In the Koch problem, we use the strong form of PINNs and the energy form DEM. The $u(\mathbf{x})$ in PINNs and DEM is constructed using the distance function from Eq. (16):

$$u(\mathbf{x}) = u_p(\mathbf{x}) + D(\mathbf{x}) * u_g(\mathbf{x}). \quad (68)$$

In the flower problem, we use the boundary element method (BIMN). Note that we solve all forms using both MLP and KAN. Fig. 21 shows the relative error contour plots, indicating that KAN does not perform well in solving complex geometric problems. Fig. 22 shows the relative error evolution trends for PINNs, DEM, and BIMN. The results indicate that KAN performs similarly or even worse than MLP in complex boundaries, particularly in BIMN. Table 5 summarizes the network structure, error, and time for each algorithm.

In conclusion, KINN does not show a significant improvement over MLP in solving complex geometries compared to regular geometries. The reason is that the grid range of KAN is rectangular in high dimensions, making it inherently more suitable for regular geometries. Additionally, there is a correlation between grid size and geometric complexity; the more complex the geometry, the larger the grid size should be. This is similar to the complexity of the fitted function, as shown in Fig. 3, where higher frequencies require larger grid sizes. However, increasing the grid size beyond a certain point can suddenly increase errors, as demonstrated in the original KAN paper [33] and our results in Fig. 8. Therefore, the grid size cannot be increased indefinitely and must correspond to the complexity of the fitted function to select an optimal grid size to prevent overfitting.

Thus, in complex geometries there is a conflict between the simplicity of the function and the complexity of the geometry, making it difficult to find a balance between grid size and the complexity of the geometries. This is a potential area for future exploration to improve KAN's accuracy in solving PDEs in complex geometries.

To increase the performance of KINN in complex geometries, mesh adaptation techniques from finite element methods, such as h-p-refinements [48], especially h-refinement, can be incorporated into KINN. H-refinement can be applied to modify the grid size in KAN. Another approach is to use the concept of isoparametric

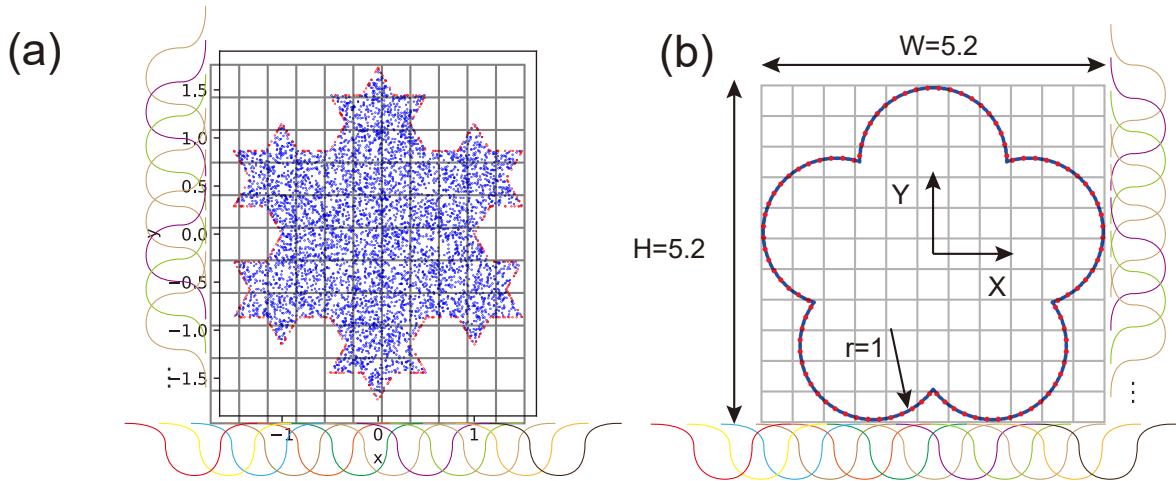


Fig. 20. The introduction of complex geometry problem: (a) Koch: Blue points are interior points, red points are Dirichlet boundary conditions. (b) Flower: Red points are source points in BINN, and blue points are integration points. Because there are too many blue points, they are presented like blue lines. The radius of each flower petal is 1.

Table 5

The comparison between different PINNs, DEM, and BINN algorithms based on MLP or KAN in complex geometry.

Algorithms	Relative error	Grid size	grid_range	order	Architecture of NNs	Parameters	Time (Second, 1000 Epoch)
PINNs_MLP	0.01939				[2, 30,30,30,30,1]	2911	13.32
KINN: PINNs	0.01188	10	[-1,1]	2	[2, 5,5,5,1]	910	30.18
DEM_MLP	0.01563				[2, 30,30,30,30,1]	2911	11.46
KINN: DEM	0.01565	3	[-1,1]	2	[2, 5,5,5,1]	280	11.38
BINN_MLP	0.0002061				[2, 30,30,30,30,1]	2911	3.99
KINN: BINN	0.004729	10	[-1,1]	3	[2, 5,5,5,1]	975	14.95

transformation from finite element methods [49] to transform complex geometries into a simpler reference domain. Alternatively, the idea of conformal transformations from mathematics can be used to handle complex geometries by mapping them into another easier-to-manage spatial domain to leverage the capabilities of KAN.

5. Conclusion

We compared KAN and MLP in different forms of PDE, proposing the KINN algorithm for solving PDEs in strong, energy, and inverse forms by KAN. We conducted systematic numerical experiments and validations on common benchmarks in engineering mechanics. The results show that KAN has higher accuracy and convergence speed than MLP in most PDE problems, particularly in singularity problems, stress concentration problems, nonlinear hyperelastic problems, and heterogeneous problems. However, KAN's efficiency is currently lower than MLP under the same epoch due to the lack of specific optimizations for the KAN algorithm. With optimization, the efficiency of KINN can be significantly improved. Additionally, we systematically analyzed KAN from the NTK perspective and found that it has a much smaller spectral bias than MLP, making it more suitable for solving high and low-frequency mixed problems. Finally, we discovered that KAN does not perform as well on complex geometric PDE problems, primarily due to the conflict between grid size and geometric complexity.

Nevertheless, KINN currently has limitations and areas for expansion. During our experiments, we found that an overly large grid size can cause KAN to fail, i.e., the error increases due to overfitting. Therefore, when using KINN, it is important to determine the grid size according to the problem's complexity. In the future, incorporating mesh adjustment techniques and concepts from traditional FEM can help KAN adjust the appropriate grid size. Furthermore, we have not yet addressed inverse problems, such as using pure data fitting

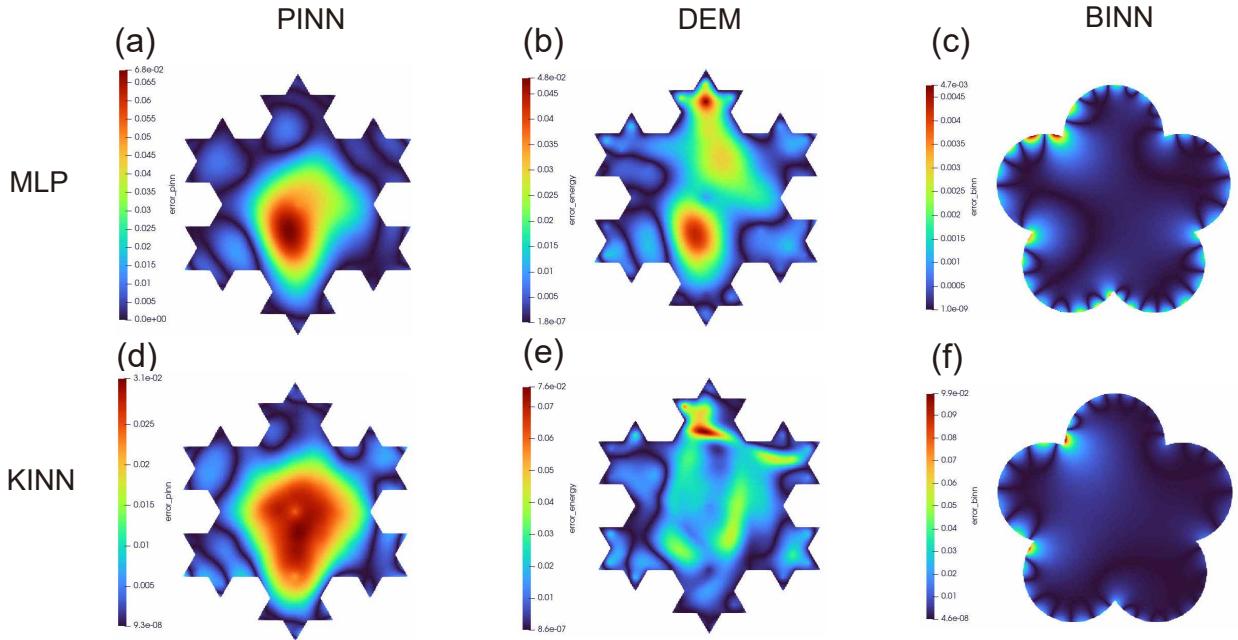


Fig. 21. The absolute error by PINN, DEM, BINN with MLP and KANN in complex geometry problem: PINN, DEM, BINN with all MLP (first row), PINN, DEM, BINN with all KAN (second row).

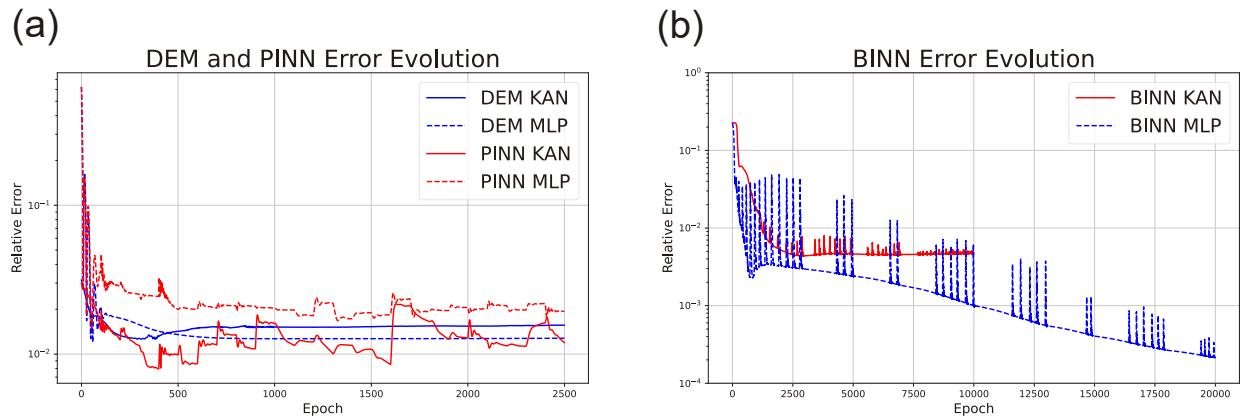


Fig. 22. Error evolution of KANN and corresponding MLP algorithms in complex geometry: (a) PINNs and DEM L_2 relative error evolution trend. (b) BINN L_2 relative error evolution trend.

to analyze function fields obtained from traditional algorithms and employing KAN to obtain simple symbolic representations of functions. Moreover, mesh adjustment methods from FEM can also be integrated into KINN, such as h-p-refinements [48], particularly h-refinement. The concept of h-refinement can be used to adjust the grid size in KAN. Alternatively, the idea of isoparametric transformation in finite element methods [49] can be used to map complex geometries to a simple reference domain. The mesh generation techniques in finite element analysis (h-p-refinements) and the concept of isoparametric transformation could both potentially address the issue of low accuracy in solving complex geometries with KINN in the future. Additionally, we have not yet explored the weak form. This is because the weak form requires consideration of different weight functions, making the research more complex compared to other forms. Moreover, the energy form is a special case of the weak form. Nonetheless, it is undeniable that studying the performance of KINN in the weak form will be an important direction for future research. Interested readers can refer to [18, 50] on the weak form.

In conclusion, we believe that KAN will become the cornerstone of AI for PDEs in the future, replacing MLP, because KAN is more in line with the essence of solving PDEs compared to MLP. KAN will be the "engine" in AI for PDEs.

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.

Acknowledgement

The study was supported by the Key Project of the National Natural Science Foundation of China (12332005) and scholarship from Bauhaus University in Weimar.

Appendix A. The requirement of PINNs to DEM

Consider a general form of PDEs as in Eq. (1), written in the Galerkin form:

$$\int_{\Omega} [\mathbf{P}(\mathbf{u}) - \mathbf{f}] \cdot \delta \mathbf{u} d\Omega - \int_{\Gamma} [\mathbf{B}(\mathbf{u}) - \mathbf{g}] \cdot \delta \mathbf{u} d\Gamma = 0 \quad (\text{A.1})$$

If the differential operator \mathbf{P} is linear and self-adjoint, the linear operator is:

$$\mathbf{P}\left(\sum_{i=1}^N \alpha_i \mathbf{u}_i\right) = \sum_{i=1}^N \alpha_i \mathbf{P}(\mathbf{u}_i) \quad (\text{A.2})$$

The self-adjoint operator must satisfy:

$$\int_{\Omega} \mathbf{P}(\mathbf{u}) \cdot \mathbf{v} d\Omega = \int_{\Omega} \mathbf{u} \cdot \mathbf{P}^*(\mathbf{v}) d\Omega + b(\mathbf{u}, \mathbf{v})$$

where, after integration by parts, $\mathbf{P} = \mathbf{P}^*$ and $b(\mathbf{u}, \mathbf{v})$ refers to the boundary integral term regarding \mathbf{u} and \mathbf{v} .

If the operator \mathbf{P} in Eq. (A.1) is linear and self-adjoint, we use the self-adjoint property of \mathbf{P} :

$$\begin{aligned} \int_{\Omega} \mathbf{P}(\mathbf{u}) \cdot \delta \mathbf{u} d\Omega &= \int_{\Omega} \frac{1}{2} \mathbf{P}(\mathbf{u}) \cdot \delta \mathbf{u} d\Omega + \int_{\Omega} \frac{1}{2} \mathbf{P}(\mathbf{u}) \cdot \delta \mathbf{u} d\Omega \\ &= \int_{\Omega} \frac{1}{2} \mathbf{P}(\mathbf{u}) \cdot \delta \mathbf{u} d\Omega + \int_{\Omega} \frac{1}{2} \mathbf{u} \cdot \mathbf{P}(\delta \mathbf{u}) d\Omega + b(\mathbf{u}, \delta \mathbf{u}) \end{aligned} \quad (\text{A.3})$$

Then, using the linear property of \mathbf{P} :

$$\begin{aligned}\int_{\Omega} \mathbf{P}(\mathbf{u}) \cdot \delta \mathbf{u} d\Omega &= \int_{\Omega} \frac{1}{2} \mathbf{P}(\mathbf{u}) \cdot \delta \mathbf{u} d\Omega + \int_{\Omega} \frac{1}{2} \mathbf{u} \cdot \delta \mathbf{P}(\mathbf{u}) d\Omega + b(\mathbf{u}, \delta \mathbf{u}) \\ &= \delta \left(\int_{\Omega} \frac{1}{2} \mathbf{P}(\mathbf{u}) \cdot \mathbf{u} d\Omega \right) + b(\mathbf{u}, \delta \mathbf{u})\end{aligned}\quad (\text{A.4})$$

Substituting Eq. (A.4) into Eq. (A.1):

$$\int_{\Omega} [\mathbf{P}(\mathbf{u}) - \mathbf{f}] \cdot \delta \mathbf{u} d\Omega - \int_{\Gamma} [\mathbf{B}(\mathbf{u}) - \mathbf{g}] \cdot \delta \mathbf{u} d\Gamma = \delta \left(\int_{\Omega} \frac{1}{2} \mathbf{P}(\mathbf{u}) \cdot \mathbf{u} d\Omega \right) + b(\mathbf{u}, \delta \mathbf{u}) - \int_{\Gamma} [\mathbf{B}(\mathbf{u}) - \mathbf{g}] \cdot \delta \mathbf{u} d\Gamma \quad (\text{A.5})$$

Usually, the two distribution integral terms can be written as a boundary integral functional, i.e.,

$$\delta(\Gamma(\mathbf{u})) = b(\mathbf{u}, \delta \mathbf{u}) - \int_{\Gamma} [\mathbf{B}(\mathbf{u}) - \mathbf{g}] \cdot \delta \mathbf{u} d\Gamma \quad (\text{A.6})$$

Therefore, Eq. (A.5) can be transformed into a variational form of a functional,

$$\begin{aligned}\delta\Pi &= \int_{\Omega} [\mathbf{P}(\mathbf{u}) - \mathbf{f}] \cdot \delta \mathbf{u} d\Omega - \int_{\Gamma} [\mathbf{B}(\mathbf{u}) - \mathbf{g}] \cdot \delta \mathbf{u} d\Gamma \\ \Pi &= \int_{\Omega} \left[\frac{1}{2} \mathbf{P}(\mathbf{u}) - \mathbf{f} \right] \cdot \mathbf{u} d\Omega + \Gamma(\mathbf{u})\end{aligned}\quad (\text{A.7})$$

Thus, solving the strong form of PDEs (PINNs) is equivalent to solving the stationary value problem of the functional. If the operator \mathbf{P} is of 2m even order and is linear and self-adjoint, the stationary value problem can further be transformed into an extremum problem (DEM), which we will not prove here.

Appendix B. Similarities between KAN and Finite Elements Method

For simplicity, let's first consider a 1D linear finite element, with N elements. The approximation function is

$$u^{fem}(x) = \sum_{i=1}^{N+1} N_i(x) u_i \quad (\text{B.1})$$

where $N(x)$ is the shape function in finite elements, and u_i is the nodal displacement. On the other hand, consider KAN with structure [1, 1] and grid size=N. The mathematical structure of KAN is:

$$u^{kan}(x) = K(x) = \sum_{i=1}^{N+k} B_i(x) c_i \quad (\text{B.2})$$

where $B_i(x)$ is the basis function of B-splines. For simplicity, we first disregard the additional residual function $\mathbf{W} \cdot \sigma(\mathbf{X})$ and the scaling factors \mathbf{S} for B-splines, as shown in Eq. (29). If we choose first-order splines, i.e., $k = 1$, then we find that

$$B_i(x) = N_i(x) \quad (\text{B.3})$$

This means that KAN, in this special structure, is equivalent to a linear 1D finite element. In higher orders, KAN and FEM differ. For example, consider a quadratic element with shape functions:

$$\begin{aligned}N_1 &= \frac{x(x-1)}{2} \\ N_2 &= (x+1)(1-x) \\ N_3 &= \frac{(x+1)x}{2}\end{aligned}\quad (\text{B.4})$$

On the other hand, if we consider $k = 2$ quadratic B-splines, and control the number of nodes to be the same (grid size is 2), for $-1 \leq x < 0$:

$$\begin{aligned} B_1 &= \frac{x^2}{2} \\ B_2 &= -x^2 - x + \frac{1}{2} \\ B_3 &= \frac{1}{2}x^2 + x + \frac{1}{2} \\ B_4 &= 0 \end{aligned} \tag{B.5}$$

and for $0 \leq x < 1$:

$$\begin{aligned} B_1 &= 0 \\ B_2 &= \frac{1}{2}x^2 - x + \frac{1}{2} \\ B_3 &= -x^2 + x + \frac{1}{2} \\ B_4 &= \frac{x^2}{2} \end{aligned} \tag{B.6}$$

Although it seems that KAN and quadratic FEM differ because order=2, grid size=2 KAN has four coefficients to determine while FEM has only three, we prove that KAN can degenerate to FEM. Setting $u^{fem}(x) = u^{kan}(x)$ for $-1 \leq x < 0$:

$$\begin{aligned} \sum_{i=1}^3 N_i(x)u_i &= \sum_{i=1}^4 B_i(x)c_i \\ \frac{x(x-1)}{2}u_1 + (x+1)(1-x)u_2 + \frac{(x+1)x}{2}u_3 &= \frac{x^2}{2}c_1 + (-x^2 - x + \frac{1}{2})c_2 + (\frac{1}{2}x^2 + x + \frac{1}{2})c_3 \\ (\frac{1}{2}u_1 - u_2 + \frac{1}{2}u_3)x^2 + (-\frac{1}{2}u_1 + \frac{1}{2}u_3)x + u_2 &= (\frac{1}{2}c_1 - c_2 + \frac{1}{2}c_3)x^2 + (-c_2 + c_3)x + \frac{1}{2}c_2 + \frac{1}{2}c_3 \end{aligned} \tag{B.7}$$

We obtain the relationship:

$$\begin{bmatrix} \frac{1}{2} & -1 & \frac{1}{2} \\ 0 & -1 & 1 \\ 0 & \frac{1}{2} & \frac{1}{2} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & -1 & \frac{1}{2} \\ -\frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} \tag{B.8}$$

$$\begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & -1 & \frac{1}{2} \\ 0 & -1 & 1 \\ 0 & \frac{1}{2} & \frac{1}{2} \end{bmatrix}^{-1} \begin{bmatrix} \frac{1}{2} & -1 & \frac{1}{2} \\ -\frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} = \begin{bmatrix} 1.75 & -1 & 0.25 \\ 0.25 & 1 & -0.25 \\ -0.25 & 1 & 0.25 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} \tag{B.9}$$

Similarly, for $0 \leq x < 1$:

$$\begin{aligned} \sum_{i=1}^3 N_i(x)u_i &= \sum_{i=1}^4 B_i(x)c_i \\ \frac{x(x-1)}{2}u_1 + (x+1)(1-x)u_2 + \frac{(x+1)x}{2}u_3 &= (\frac{1}{2}x^2 - x + \frac{1}{2})c_2 + (-x^2 + x + \frac{1}{2})c_3 + (\frac{x^2}{2})c_4 \\ (\frac{1}{2}u_1 - u_2 + \frac{1}{2}u_3)x^2 + (-\frac{1}{2}u_1 + \frac{1}{2}u_3)x + u_2 &= (\frac{1}{2}c_2 - c_3 + \frac{1}{2}c_4)x^2 + (-c_2 + c_3)x + \frac{1}{2}c_2 + \frac{1}{2}c_3 \end{aligned} \tag{B.10}$$

We obtain the relationship:

$$\begin{bmatrix} \frac{1}{2} & -1 & \frac{1}{2} \\ -1 & 1 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 \end{bmatrix} \begin{bmatrix} c_2 \\ c_3 \\ c_4 \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & -1 & \frac{1}{2} \\ -\frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} \tag{B.11}$$

$$\begin{bmatrix} c_2 \\ c_3 \\ c_4 \end{bmatrix} = \begin{bmatrix} 0.25 & 1 & -0.25 \\ -0.25 & 1 & 0.25 \\ 0.25 & -1 & 1.75 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} \quad (\text{B.12})$$

Comparing Eq. (B.9) and Eq. (B.12), we find that $\{c_1, c_2, c_3, c_4\}$ can be completely determined by $\{u_1, u_2, u_3\}$, i.e.,

$$\begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix} = \begin{bmatrix} 1.75 & -1 & 0.25 \\ 0.25 & 1 & -0.25 \\ -0.25 & 1 & 0.25 \\ 0.25 & -1 & 1.75 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} \quad (\text{B.13})$$

This demonstrates that KAN can completely encompass quadratic finite elements when order=2, and grid size=2. For higher dimensions instead of 1D, this conclusion holds to some extent. For example, consider a simple 2D linear element. With uniform N grids in both x and y directions, the FEM approximation is:

$$u(\mathbf{x}) = \sum_{i=1}^{(N+1)^2} N_i(\mathbf{x}) u_i \quad (\text{B.14})$$

where $N_i(\mathbf{x})$ is the shape function:

$$N_i(\mathbf{x}) = N_i^x(x) \cdot N_i^y(y) \quad (\text{B.15})$$

For KAN with structure [2, 1], grid size=N, order=1, the KAN function is:

$$K(\mathbf{x}) = \sum_{i=1}^{N+1} B_i(x) c_i^x + \sum_{i=1}^{N+1} B_i(y) c_i^y = \sum_{i=1}^{N+1} N_i^x(x) c_i^x + \sum_{i=1}^{N+1} N_i^y(y) c_i^y \quad (\text{B.16})$$

Comparing Eq. (B.14) and Eq. (B.16), we find significant differences in the forms of FEM and KAN. However, if we replace the additive combination of KAN activation functions in Eq. (B.16) with a multiplicative combination, it would exactly match the 2D function representation of FEM:

$$K_{modified}(\mathbf{x}) = \sum_{i=1}^{N+1} B_i(x) c_i^x * \sum_{i=1}^{N+1} B_i(y) c_i^y = \sum_{i=1}^{N+1} N_i^x(x) c_i^x * \sum_{i=1}^{N+1} N_i^y(y) c_i^y = \sum_{i=1}^{(N+1)^2} N_i(\mathbf{x}) c_i^x c_i^y. \quad (\text{B.17})$$

This suggests that KAN could be transformed into a Deep Finite Element Method by adjusting the activation functions from additive to multiplicative combinations. Note that the above proof is based on the very shallow layers of KAN. With deeper KAN structures, we believe KAN will exhibit much stronger fitting capabilities compared to FEM. This proof mainly illustrates the similarity between FEM and KAN, with the fundamental reason being that FEM's fitting functions are spline interpolations. For example, linear elements correspond to first-order splines, making FEM and KAN similar. Since KAN is composed of nested B-spline functions, KAN is mathematically a nested shape function in FEM:

$$S(S(\dots S(\mathbf{x}))) \quad (\text{B.18})$$

where S is the shape function in FEM.

Considering that B-splines are essentially a degenerate form of NURBS in IGA, where B-splines are modified to be non-uniform and combined with weight functions to form NURBS, mathematically, KAN is quite similar to nested NURBS in IGA. Currently, KAN still uses B-splines, but in the future, it could be replaced by NURBS instead of B-splines.

As a result, KAN is similar to nested functions of shape functions in FEM and NURBS in IGA.

Appendix C. The highest order of PDEs that KINN can solve

The key to solving high-order PDEs using neural networks is ensuring that higher-order derivatives are non-zero. We analyze the applicability conditions of KINN for high-order PDEs based on the structure of KAN.

The highest order of PDEs that KINN can solve is related to the order of the B-splines and the activation function. We consider the following single-layer KINN computation structure:

$$\mathbf{X}^{(\text{new})} = K(\mathbf{X}^{(\text{old})}) = \tanh \left\{ \left[\sum_{\text{column}} \phi(\mathbf{X}^{(\text{old})}) \odot \mathbf{S} \right] + \mathbf{W} \cdot \sigma(\mathbf{X}^{(\text{old})}) \right\}. \quad (\text{C.1})$$

If KINN has an $N + 1$ layer network structure $[n_0, n_1, n_2, \dots, n_N]$, then the overall computation is

$$\mathbf{X}^{(N)} = K^{(N)} \circ \dots \circ K^{(2)} \circ K^{(1)}(\mathbf{X}^{(0)}). \quad (\text{C.2})$$

Using the chain rule, we can derive the derivative of the output with respect to the input, which approximates the differential operator of PDEs:

$$\frac{\partial \mathbf{X}^{(N)}}{\partial \mathbf{X}^{(0)}} = \frac{\partial K^{(N)}(\mathbf{X}^{(N-1)})}{\partial \mathbf{X}^{(N-1)}} \frac{\partial K^{(N-1)}(\mathbf{X}^{(N-2)})}{\partial \mathbf{X}^{(N-2)}} \dots \frac{\partial K^{(2)}(\mathbf{X}^{(1)})}{\partial \mathbf{X}^{(1)}} \frac{\partial K^{(1)}(\mathbf{X}^{(0)})}{\partial \mathbf{X}^{(0)}}. \quad (\text{C.3})$$

It is clear from Eq. (C.3) that it involves the multiplication of similar tensors. We analyze one term:

$$\begin{aligned} \frac{\partial K^{(I+1)}(\mathbf{X}^{(I)})}{\partial \mathbf{X}^{(I)}} &= \frac{\partial \tanh(\mathbf{Y}^{(I)})}{\partial \mathbf{Y}^{(I)}} \frac{\partial \mathbf{Y}^{(I)}}{\partial \mathbf{X}^{(I)}} \\ \mathbf{Y}^{(I)} &= \left[\sum_{\text{column}} \phi(\mathbf{X}^{(I)}) \odot \mathbf{S} \right] + \mathbf{W} \cdot \sigma(\mathbf{X}^{(I)}) \end{aligned} \quad (\text{C.4})$$

For convenience, we use tensor operations, obtaining:

$$\frac{\partial \mathbf{Y}^{(I)}}{\partial \mathbf{X}^{(I)}} = \frac{\partial [S_{i^*j} \phi_{ij}(\mathbf{X}^{(I)}) + W_{ij} \sigma(X_j^{(I)})]}{\partial \mathbf{X}^{(I)}} = S_{i^*j} \frac{\partial \phi_{ij}(\mathbf{X}^{(I)})}{\partial \mathbf{X}^{(I)}} + W_{ij} \cdot \frac{\partial \sigma(X_j^{(I)})}{\partial \mathbf{X}^{(I)}} \quad (\text{C.5})$$

where the * denotes terms not included in the summation. Substituting the B-spline formula into Eq. (C.5):

$$\begin{aligned} \frac{\partial \mathbf{Y}^{(I)}}{\partial \mathbf{X}^{(I)}} &= S_{i^*j} \frac{\partial c_m^{(i,j)} B_m(X_{j^*})}{\partial \mathbf{X}^{(I)}} + W_{ij} \cdot \frac{\partial \sigma(X_j^{(I)})}{\partial \mathbf{X}^{(I)}} = S_{i^*j} [c_m^{(i,j)} \frac{\partial B_m(X_{j^*})}{\partial \mathbf{X}^{(I)}}] + W_{ij} \frac{\partial \sigma(X_j^{(I)})}{\partial \mathbf{X}^{(I)}} \\ \frac{\partial Y_i^{(I)}}{\partial X_j^{(I)}} &= S_{i^*k} [c_m^{(i,k)} \frac{\partial B_m(X_{k^*})}{\partial X_j^{(I)}}] + W_{ik} \frac{\partial \sigma(X_k^{(I)})}{\partial X_j^{(I)}} \end{aligned} \quad (\text{C.6})$$

Substituting Eq. (C.6) into Eq. (C.4), we find:

$$\frac{\partial K_i^{(I+1)}(\mathbf{X}^{(I)})}{\partial X_j^{(I)}} = \frac{\partial \tanh(Y_i^{(I)})}{\partial Y_m^{(I)}} \frac{\partial Y_m^{(I)}}{\partial X_j^{(I)}} = \frac{\partial \tanh(Y_i^{(I)})}{\partial Y_m^{(I)}} S_{m^*k} [c_q^{(m,k)} \frac{\partial B_q(X_{k^*})}{\partial X_j^{(I)}}] + \frac{\partial \tanh(Y_i^{(I)})}{\partial Y_m^{(I)}} W_{mk} \frac{\partial \sigma(X_k^{(I)})}{\partial X_j^{(I)}} \quad (\text{C.7})$$

From Eq. (C.7), we see that the activation function \tanh ensures that the first-order derivative of the single-layer KINN chain rule is non-zero. Since Eq. (C.3) involves multiple tensor multiplications from Eq. (C.7), the product rule of differentiation ensures that higher-order derivatives remain non-zero if each term in Eq. (C.7) is non-zero. The smoothness of \tanh guarantees that the higher-order tensor derivatives are non-zero.

If we remove the \tanh activation function, we get:

$$\frac{\partial K_i^{(I+1)}(\mathbf{X}^{(I)})}{\partial X_j^{(I)}} = \delta_{im} S_{m^*k} [c_q^{(m,k)} \frac{\partial B_q(X_{k^*})}{\partial X_j^{(I)}}] + \delta_{im} W_{mk} \frac{\partial \sigma(X_k^{(I)})}{\partial X_j^{(I)}} \quad (\text{C.8})$$

Table C.6

The role of each function in KINN: $\mathbf{X}^{(\text{new})} = K(\mathbf{X}^{(\text{old})}) = \tanh\{\sum_{\text{column}}[\phi(\mathbf{X}^{(\text{old})}) \odot \mathbf{S}] + \mathbf{W} \cdot \sigma(\mathbf{X}^{(\text{old})})\}$

Function in KINN	Description
ϕ	Enhances KAN's ability to fit the activation function by adding B-splines
σ	Residual term in KAN, similar to the ResNet
S	Scaling factor for ϕ
W	Scaling factor for σ
tanh	Improves function smoothness, scales output within the grid size, enhancing the effectiveness of B-splines

We find that Eq. (C.8) can still be non-zero due to $\partial\sigma(X_k^{(I)})/\partial X_j^{(I)}$ if $\sigma(X_k^{(I)})$ is infinitely smooth. However, this might compromise the fitting ability of the B-splines because $\partial B_q(X_{k^*})/\partial X_j^{(I)}$ could be zero, depending on the B-spline order, the number of KAN layers, and the highest order of the PDEs.

If we use only B-spline functions to construct KAN, such as:

$$\mathbf{X}^{(\text{new})} = K(\mathbf{X}^{(\text{old})}) = \sum_{\text{column}} [\phi(\mathbf{X}^{(\text{old})}) \odot \mathbf{S}], \quad (\text{C.9})$$

we might easily get zero for higher-order derivatives. For example, if the B-spline order is 3 and the KAN network structure is [2, 5, 1], the maximum order of KAN is $3 * (3 - 1) = 6$. Therefore, solving derivatives higher than the 6th order would result in zero derivatives, rendering the algorithm ineffective. Hence, each layer's tanh activation function not only pulls the output back within the grid size $[-1, 1]$ to better utilize the B-splines but also ensures there is no limit to the order of PDEs that can be solved.

Based on the above analysis, Table C.6 summarizes the core functions of each component in KINN.

To verify our theory, we solve a one-dimensional fourth-order PDE problem:

$$\begin{cases} \nabla^4 \phi = 0 & \theta \in [0, \pi] \\ \sigma_\theta = \frac{\partial^2 \phi}{\partial r^2} = -q & \theta = 0 \\ \sigma_\theta = \frac{\partial^2 \phi}{\partial r^2} = 0 & \theta = \pi \\ \tau_{r\theta} = -\frac{\partial}{\partial r} \left(\frac{1}{r} \frac{\partial \phi}{\partial \theta} \right) = 0 & \theta = 0 \\ \tau_{r\theta} = -\frac{\partial}{\partial r} \left(\frac{1}{r} \frac{\partial \phi}{\partial \theta} \right) = 0 & \theta = \pi \end{cases} \quad (\text{C.10})$$

This is the problem of an infinite wedge [26], as shown in Fig. C.23m, where ϕ is the Airy stress function. The analytical solution to this problem is

$$\phi = cr^2 [\alpha - \theta + \sin(\theta) \cos(\theta) - \cos^2(\theta) \tan(\alpha)] \quad (\text{C.11})$$

where $c = \frac{q}{2(\tan(\alpha) - \alpha)}$, α is the angle of the infinite wedge, here $\alpha = \pi$, $q = 5$, $E = 1000$, and $\mu = 0.3$.

We solve this problem using KINN-PINNs. Fig. C.23 shows the comparison between the KINN-PINNs predictions and the analytical solution. From the absolute error, we can see that KINN can solve this problem well. However, if we modify the structure of KAN by removing the activation function tanh and the residual term $\mathbf{W} \cdot \sigma(\mathbf{X})$:

$$\mathbf{Y}^{(\text{new})} = \tanh(\mathbf{Y}) = \sum_{\text{column}} [\phi(\mathbf{X}) \odot \mathbf{S}] \quad (\text{C.12})$$

and set the structure of KAN to [1,1] with a spline order of 3, we find that KINN cannot compute the fourth-order derivative. This result is consistent with our theoretical derivation. Therefore, when using KAN to solve PDEs, it is crucial to include the activation function tanh and the residual term $\mathbf{W} \cdot \sigma(\mathbf{X})$ to ensure the feasibility of solving high-order PDEs.

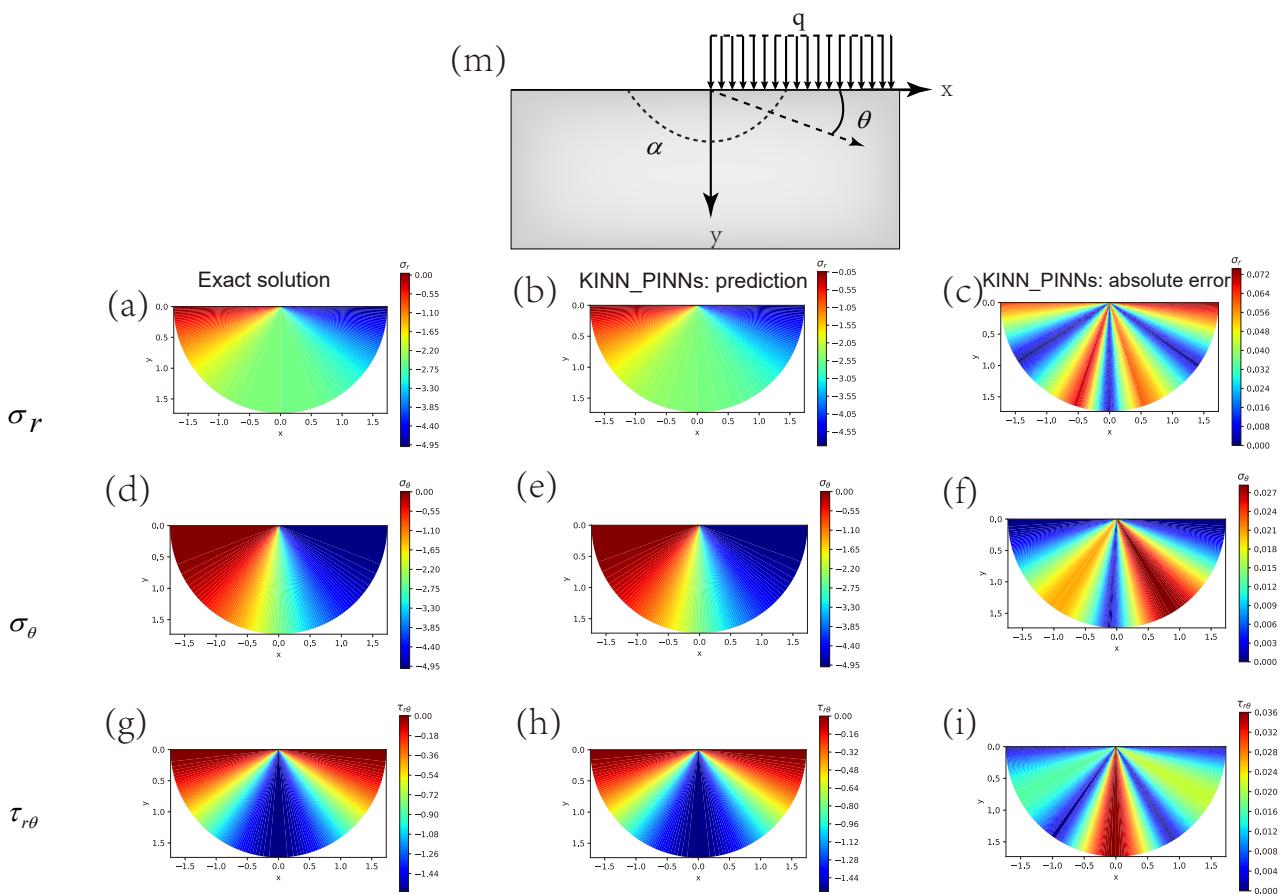


Fig. C.23. The infinite wedge problem and the performance of the KINN algorithm: (m) Schematic of the infinite wedge problem. (a,b,c) Analytical solution, KINN prediction, and absolute error of σ_r . (d,e,f) Analytical solution, KINN prediction, and absolute error of σ_θ . (g,h,i) Analytical solution, KINN prediction, and absolute error of $\tau_{r\theta}$.

Appendix D. Scale normalization in Deep Energy Method

Scale normalization is a crucial operation in KINN because it ensures that the input remains within the range of the B-spline basis functions, while we keep the grid size of the B-splines fixed at $[-1, 1]$. However, special attention needs to be given to DEM. When the characteristic size is particularly large, we consider Fig. 9 to illustrate this issue. Without scale normalization, this problem often uses the following admissible functions:

$$u_x = x\phi_x(\mathbf{x}; \boldsymbol{\theta}) \quad (\text{D.1})$$

$$u_y = y\phi_y(\mathbf{x}; \boldsymbol{\theta}) \quad (\text{D.2})$$

Without scale normalization, large coordinate gradients may arise during differentiation. For instance, consider the normal strain ε_{xx} obtained by differentiating Eq. (D.1) with respect to x :

$$\varepsilon_{xx} = \frac{\partial u_x}{\partial x} = \phi_x(\mathbf{x}; \boldsymbol{\theta}) + x \frac{\partial \phi_x(\mathbf{x}; \boldsymbol{\theta})}{\partial x} \quad (\text{D.3})$$

In Eq. (D.3), the term $x \frac{\partial \phi_x(\mathbf{x}; \boldsymbol{\theta})}{\partial x}$ can cause the normal strain ε_{xx} to become excessively large. Note that $\phi_x(\mathbf{x}; \boldsymbol{\theta})$ and $\frac{\partial \phi_x(\mathbf{x}; \boldsymbol{\theta})}{\partial x}$ do not typically increase significantly with size. This is because $\phi_x(\mathbf{x}; \boldsymbol{\theta})$ often includes a tanh activation function in the hidden layers, which scales the output to $[-1, 1]$. The derivative $\frac{\partial \phi_x(\mathbf{x}; \boldsymbol{\theta})}{\partial x}$ is related to the weights of the initial linear transformation in a simple fully connected neural network:

$$\begin{aligned} \frac{\partial \phi_x(\mathbf{x}; \boldsymbol{\theta})}{\partial x} &= \frac{\partial y^{(N)}(a^{(N-1)})}{\partial a^{(N-1)}} \frac{\partial a^{(N-1)}(\mathbf{y}^{(N-1)})}{\partial \mathbf{y}^{(N-1)}} \frac{\partial \mathbf{y}^{(N-1)}(\mathbf{a}^{(N-2)})}{\partial \mathbf{a}^{(N-2)}} \dots \frac{\partial a^{(1)}(\mathbf{y}^{(1)})}{\partial \mathbf{y}^{(1)}} \frac{\partial \mathbf{y}^{(1)}(x)}{\partial x} \\ &= \mathbf{W}^{(N)} \sigma'(\mathbf{y}^{(N-1)}) \mathbf{W}^{(N-1)} \dots \sigma'(\mathbf{y}^{(1)}) \mathbf{W}^{(1)} \end{aligned} \quad (\text{D.4})$$

where $\sigma'(\mathbf{y}^{(N-1)})$ is the derivative of the activation function with respect to $\mathbf{y}^{(N-1)}$. If the activation function is tanh, this term will not exceed 1. Hence, the range of Eq. (D.4) is:

$$\left| \frac{\partial \phi_x(\mathbf{x}; \boldsymbol{\theta})}{\partial x} \right| \leq \left| \mathbf{W}^{(N)} \mathbf{W}^{(N-1)} \dots \mathbf{W}^{(1)} \right| \quad (\text{D.5})$$

The derivative $\frac{\partial \phi_x(\mathbf{x}; \boldsymbol{\theta})}{\partial x}$ is related to the weights of the linear transformation and does not exhibit a size effect. Therefore, the magnitude of Eq. (D.3) is mainly influenced by the term $x \frac{\partial \phi_x(\mathbf{x}; \boldsymbol{\theta})}{\partial x}$, which depends on the size of x . To address this issue, we introduce scale normalization, modifying Eq. (D.1) and Eq. (D.2) as follows:

$$u_x = x\phi_x\left(\frac{\mathbf{x}}{L}; \boldsymbol{\theta}\right) \quad (\text{D.6})$$

$$u_y = y\phi_y\left(\frac{\mathbf{x}}{L}; \boldsymbol{\theta}\right) \quad (\text{D.7})$$

where L is the characteristic size of the object being simulated. Differentiating these expressions yields:

$$\varepsilon_{xx} = \frac{\partial u_x}{\partial x} = \phi_x\left(\frac{\mathbf{x}}{L}; \boldsymbol{\theta}\right) + \frac{x}{L} \frac{\partial \phi_x\left(\frac{\mathbf{x}}{L}; \boldsymbol{\theta}\right)}{\partial x} \quad (\text{D.8})$$

The benefit of scale normalization is that it eliminates the size effect on coordinate derivatives and maps the input to $\frac{\mathbf{x}}{L}$ (within $[-1, 1]$), enhancing the fitting ability of the neural network. This is applicable not only in MLP but also in KAN, where scale normalization is even more essential than MLP.

In this section, we mathematically explain why scale normalization is a critical trick in DEM. Furthermore, Eq. (D.6) and Eq. (D.7) have another advantage: the linear scaling of the solution is directly related to the size. If the size of the PDE simulation increases by a factor of L , the solution will scale by the same factor. The involvement of the coordinate x in Eq. (D.6) and Eq. (D.7) ensures that the solution scales appropriately with size, maintaining the scale relationship.

Appendix E. Supplementary code

The code of this work will be available at <https://github.com/yizheng-wang/Research-on-Solving-Partial-Differential-Equations-of-Solid-Mechanics-Based-on-PINN> after accepted.

References

- [1] E. Samaniego, C. Anitescu, S. Goswami, V. M. Nguyen-Thanh, H. Guo, K. Hamdia, X. Zhuang, T. Rabczuk, An energy approach to the solution of partial differential equations in computational mechanics via machine learning: Concepts, implementation and applications, Computer Methods in Applied Mechanics and Engineering 362 (2020) 112790.
- [2] O. C. Zienkiewicz, R. L. Taylor, J. Z. Zhu, The finite element method: its basis and fundamentals, Elsevier, 2005.
- [3] T. J. Hughes, The finite element method: linear static and dynamic finite element analysis, Courier Corporation, 2012.
- [4] K.-J. Bathe, Finite element procedures, Klaus-Jurgen Bathe, 2006.
- [5] J. N. Reddy, Introduction to the finite element method, McGraw-Hill Education, 2019.
- [6] X. Zhang, Z. Chen, Y. Liu, The material point method: a continuum-based particle method for extreme loading cases, Academic Press, 2016.
- [7] G.-R. Liu, D. Karamanlidis, Mesh free methods: moving beyond the finite element method, Appl. Mech. Rev. 56 (2) (2003) B17–B18.
- [8] T. Rabczuk, T. Belytschko, Cracking particles: a simplified meshfree method for arbitrary evolving cracks, International journal for numerical methods in engineering 61 (13) (2004) 2316–2343.
- [9] T. Rabczuk, T. Belytschko, A three-dimensional large deformation meshfree method for arbitrary evolving cracks, Computer methods in applied mechanics and engineering 196 (29-30) (2007) 2777–2799.
- [10] T. Rabczuk, J.-H. Song, X. Zhuang, C. Anitescu, Extended finite element and meshfree methods, Academic Press, 2019.
- [11] V. P. Nguyen, T. Rabczuk, S. Bordas, M. Duflot, Meshless methods: a review and computer implementation aspects, Mathematics and computers in simulation 79 (3) (2008) 763–813.
- [12] R. J. LeVeque, Finite difference methods for ordinary and partial differential equations: steady-state and time-dependent problems, SIAM, 2007.
- [13] M. Darwish, F. Moukalled, The finite volume method in computational fluid dynamics: an advanced introduction with OpenFOAM® and Matlab®, Springer, 2016.
- [14] C. A. Brebbia, J. C. F. Telles, L. C. Wrobel, Boundary element techniques: theory and applications in engineering, Springer Science & Business Media, 2012.
- [15] 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, Journal of Computational Physics 378 (2019) 686–707.
- [16] L. Lu, P. Jin, G. Pang, Z. Zhang, G. E. Karniadakis, Learning nonlinear operators via deeponet based on the universal approximation theorem of operators, Nature Machine Intelligence 3 (3) (2021) 218–229. doi:[10.1038/s42256-021-00302-5](https://doi.org/10.1038/s42256-021-00302-5).
- [17] Z. Li, H. Zheng, N. Kovachki, D. Jin, H. Chen, B. Liu, K. Azizzadenesheli, A. Anandkumar, Physics-informed neural operator for learning partial differential equations, ACM/JMS Journal of Data Science 1 (3) (2024) 1–27.
- [18] E. Kharazmi, Z. Zhang, G. E. Karniadakis, hp-vpinns: Variational physics-informed neural networks with domain decomposition, Computer Methods in Applied Mechanics and Engineering 374 (2021) 113547.
- [19] J. Sun, Y. Liu, Y. Wang, Z. Yao, X. Zheng, Binn: A deep learning approach for computational mechanics problems based on boundary integral equations, Computer Methods in Applied Mechanics and Engineering 410 (2023) 116012.
- [20] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, A. Anandkumar, Fourier neural operator for parametric partial differential equations, arXiv preprint arXiv:2010.08895 (2020).
- [21] L. Lu, X. Meng, S. Cai, Z. Mao, S. Goswami, Z. Zhang, G. E. Karniadakis, A comprehensive and fair comparison of two neural operators (with practical extensions) based on fair data, Computer Methods in Applied Mechanics and Engineering 393 (2022) 114778.
- [22] F. Bartolucci, E. de Bezenac, B. Raonic, R. Molinaro, S. Mishra, R. Alaifari, Representation equivalent neural operators: a framework for alias-free operator learning, Advances in Neural Information Processing Systems 36 (2024).
- [23] N. B. Kovachki, Z. Li, B. Liu, K. Azizzadenesheli, K. Bhattacharya, A. M. Stuart, A. Anandkumar, Neural operator: Learning maps between function spaces with applications to pdes., J. Mach. Learn. Res. 24 (89) (2023) 1–97.
- [24] S. Goswami, M. Yin, Y. Yu, G. E. Karniadakis, A physics-informed variational deeponet for predicting crack path in quasi-brittle materials, Computer Methods in Applied Mechanics and Engineering 391 (2022) 114587.
- [25] S. Wang, H. Wang, P. Perdikaris, Learning the solution operator of parametric partial differential equations with physics-informed deepnets, Science advances 7 (40) (2021) eabi8605.
- [26] Y. Wang, J. Sun, T. Rabczuk, Y. Liu, Dcem-pinns: A deep complementary energy method for solid mechanics, arXiv preprint arXiv:2302.01538 (2023).
- [27] G. Cybenko, Approximation by superpositions of a sigmoidal function, Mathematics of control, signals and systems 2 (4) (1989) 303–314.
- [28] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, Neural networks 2 (5) (1989) 359–366.
- [29] A. N. Kolmogorov, On the representation of continuous functions of several variables by superpositions of continuous functions of a smaller number of variables, American Mathematical Society, 1961.
- [30] J. Braun, M. Griebel, On a constructive proof of kolmogorov superposition theorem, Constructive approximation 30 (2009) 653–675.

- [31] R. Hecht-Nielsen, Kolmogorov mapping neural network existence theorem, in: Proceedings of the international conference on Neural Networks, Vol. 3, IEEE press New York, NY, USA, 1987, pp. 11–14.
- [32] T. Poggio, A. Banburski, Q. Liao, Theoretical issues in deep networks, *Proceedings of the National Academy of Sciences* 117 (48) (2020) 30039–30045.
- [33] Z. Liu, Y. Wang, S. Vaidya, F. Ruehle, J. Halverson, M. Soljacc, T. Y. Hou, M. Tegmark, Kan: Kolmogorov-arnold networks, arXiv preprint arXiv:2404.19756 (2024).
- [34] S. SS, Chebyshev polynomial-based kolmogorov-arnold networks: An efficient architecture for nonlinear function approximation, arXiv preprint arXiv:2405.07200 (2024).
- [35] Z. Li, Kolmogorov-arnold networks are radial basis function networks, arXiv preprint arXiv:2405.06721 (2024).
- [36] Z. Bozorgasl, H. Chen, Wav-kan: Wavelet kolmogorov-arnold networks, arXiv preprint arXiv:2405.12832 (2024).
- [37] F. B. Hildebrand, Introduction to numerical analysis, Courier Corporation, 1987.
- [38] T. J. Hughes, J. A. Cottrell, Y. Bazilevs, Isogeometric analysis: Cad, finite elements, nurbs, exact geometry and mesh refinement, *Computer methods in applied mechanics and engineering* 194 (39-41) (2005) 4135–4195.
- [39] Y. Wang, J. Sun, W. Li, Z. Lu, Y. Liu, Cenn: Conservative energy method based on neural networks with subdomains for solving variational problems involving heterogeneous and complex geometries, *Computer Methods in Applied Mechanics and Engineering* 400 (2022) 115491.
- [40] J. Bai, G.-R. Liu, A. Gupta, L. Alzubaidi, X.-Q. Feng, Y. Gu, Physics-informed radial basis network (pirbn): A local approximating neural network for solving nonlinear partial differential equations, *Computer Methods in Applied Mechanics and Engineering* 415 (2023) 116290.
- [41] 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.
- [42] S. Wang, H. Wang, P. Perdikaris, On the eigenvector bias of fourier feature networks: From regression to solving multi-scale pdes with physics-informed neural networks, *Computer Methods in Applied Mechanics and Engineering* 384 (2021) 113938. [doi:10.1016/j.cma.2021.113938](https://doi.org/10.1016/j.cma.2021.113938).
- [43] A. Jacot, F. Gabriel, C. Hongler, Neural tangent kernel: Convergence and generalization in neural networks, *Advances in neural information processing systems* 31 (2018).
- [44] A. D. Jagtap, E. Kharazmi, G. E. Karniadakis, Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems, *Computer Methods in Applied Mechanics and Engineering* 365 (2020) 113028.
- [45] V. M. Nguyen-Thanh, X. Zhuang, T. Rabczuk, A deep energy method for finite deformation hyperelasticity, *European Journal of Mechanics-A/Solids* 80 (2020) 103874.
- [46] S. Wang, Y. Teng, P. J. S. J. o. S. C. Perdikaris, Understanding and mitigating gradient flow pathologies in physics-informed neural networks, *SIAM Journal on Scientific Computing* 43 (5) (2021) A3055–A3081.
- [47] S. Wang, X. Yu, P. Perdikaris, When and why pinns fail to train: A neural tangent kernel perspective, *Journal of Computational Physics* 449 (2022) 110768.
- [48] O. Zienkiewicz, J. Zhu, N. Gong, Effective and practical h-p-version adaptive analysis procedures for the finite element method, *International journal for numerical methods in engineering* 28 (4) (1989) 879–891.
- [49] V. M. Nguyen-Thanh, C. Anitescu, N. Alajlan, T. Rabczuk, X. Zhuang, Parametric deep energy approach for elasticity accounting for strain gradient effects, *Computer Methods in Applied Mechanics and Engineering* 386 (2021) 114096. [doi:10.1016/j.cma.2021.114096](https://doi.org/10.1016/j.cma.2021.114096).
- [50] E. Kharazmi, Z. Zhang, G. E. Karniadakis, Variational physics-informed neural networks for solving partial differential equations, arXiv preprint arXiv:1912.00873 (2019).