Solving inverse problems using physically informed machine learning

Gregorio Pérez Bernal[a,1] and Nicolás Guarín Zapata[a,2]

This manuscript was compiled on May 28, 2025

Inverse Problems | Kolmogorov Arnold Networks | Physically informed networks | Differential equations | Numerical methods | Approximation theory

Studying differential equations is crucial in physics as it allows us to model and predict the behavior of dynamic systems in various fields where changes in a function can be modeled more simply than the function itself (1). A differential equation can be solved using two distinct approaches, direct and inverse. The inverse approach focuses on identifying the underlying causes from a set of observations, that is, determining the values of the parameters associated with a differential equation (2).
In addition to classical numerical methods, a new approach was proposed in 2019 to solve inverse problems: physics-informed neural networks (PINNs) (3). This framework allows approximating the solution of a partial differential equation using a multilayer perceptron (MLP) with a cost function to minimize based on physical laws, as well as observations of solutions of the equation.
In 2024, Kolmogorov–Arnold Networks (KANs) were introduced as a novel alternative to traditional MLPs, offering enhanced interpretability and, in certain cases, improving accuracy in function approximation tasks (4). Their effectiveness is rooted in the Kolmogorov–Arnold representation theorem, which states that any continuous multivariate function can be expressed as a finite sum of continuous univariate functions. This naturally raises the question: can physics-informed Kolmogorov–Arnold Networks (PIKANs) outperform physics-informed neural networks (PINNs) in approximating the inverse Poisson problem on an infinite domain?

1. Mathematical formulation of the problem

To properly comprehend the problem to solve, a short framework on the different mathematical tools that are used for solving inverse problems using machine learning techniques. In its most general form, an inverse problem attempts to find the parameters of a differential equation, taking into account observations from its solution and physical laws that govern the model.

A. Poisson equation on an infinite domain. Poisson's equation is a versatile differential equation, present in multiple and seemingly unrelated areas of physics. Poisson's inverse problem then has as a goal of finding the variable parameter $k(x, y)$ (see equation 1) given the structure of the differential equation, as well as observations from solutions of the differential equation in the desired domain.

$$\nabla \cdot (k\nabla u) = f \qquad [1]$$

The value of k can account for electrical permittivity, thermic conductivity, and even Young's modulus. In some of the areas of interest, one can encounter an infinite domain, a domain that has no bounds, as information regarding boundaries is either non existent or not available.

B. Artificial Neural Networks. The process of training a neural network involves optimizing a set of parameters to minimize a predefined loss function, typically the Mean Squared Error (MSE). Given a dataset with inputs x_i and corresponding target values y_i, the MSE loss function is defined as:

Significance Statement

The ability to infer the internal properties of a physical system from indirect observations lies at the heart of solving inverse problems. In this work, we explore and compare two physically informed machine learning frameworks—PINNs and PIKANs—for solving the inverse Poisson problem in an unbounded domain. While PINNs have shown promise in incorporating physics into data-driven models, KANs were recently introduced as a potentially more interpretable and accurate alternative. However, the performance of these architectures in the context of inverse problems had not been directly evaluated on infinite domains. We propose a new strategy to sample training points from a normal distribution, rather than the usual uniform distribution. Our findings show that, despite the theoretical strengths of KANs, PINNs consistently outperform them in terms of generalization and accuracy, particularly under noisy conditions. This highlights the importance of architectural choices and training strategies when deploying machine learning models to approximate physical laws.

Author affiliations: [a]School of Applied Science and Engineering, Universidad EAFIT, Medellín, Colombia

[1] nguarinz@eafit.edu.co
[2] gperezb1@eafit.edu.co

$$\mathrm{MSE} = \frac{1}{N}\sum_{i=1}^{N}\left(\hat{y}_i - y_i\right)^2, \qquad [2]$$

where $\hat{y}_i$ represents the network's predicted output, and N is the number of data points. The optimization process seeks to adjust the network's parameters, typically using gradient-based methods, to minimize the MSE.

C. Kolmogorov Arnold Networks. Kolmogorov-Arnold Networks (KANs) are neural networks inspired by the Kolmogorov-Arnold representation theorem, which asserts that any multivariate continuous function can be decomposed into a finite sum of univariate functions. Unlike traditional Multi-Layer Perceptrons (MLPs), which employ fixed activation functions at the nodes, KANs introduce learnable activation functions along the edges. This fundamental shift allows each edge to represent a non-linear transformation using a parameterized spline function, effectively eliminating the need for traditional linear weight matrices (4). In theory, any spline function will work in the construction of the activation functions, however, as purposed by (4) and (5), the most widely used spline interpolation method is **B-splines** with **k interpolation points**.
KANs leverage this architecture to enhance accuracy and interpretability, especially in applications involving Partial Differential Equations (PDEs) and data fitting (5). The network's ability to capture intricate functional relationships makes it particularly effective for low-dimensional problems where traditional networks often face overfitting or inefficiency (4).
Unlike MLPs, which rely on fixed activation functions like ReLU, sigmoid, or inverse tangent, KANs place these univariate splines on the edges of the network, allowing each edge to learn non-linear transformations directly. The output of a general KAN with L layers is then given by the following composition of activation functions:

$$\mathrm{KAN}(x) = (\Phi_{L-1} \circ \Phi_{L-2} \circ \cdots \circ \Phi_0)(x), \qquad [3]$$

This spline-based approach allows KANs to surpass the curse of dimensionality often encountered with traditional neural networks, making them highly efficient for a wide range of applications (4). Also, as KAN's activation functions are described using splines, the neurons are polynomials, which have the interesting property of being closed by addition, enhancing interpretability.

D. Physically informed neural networks (PINNs) for inverse problems. In accordance to (3), inverse problems can be solved by minimizing a loss function built as a combination of the residual of a differential equation and observed data. The residual of a differential equation measures how much a proposed solution fails to satisfy the equation. Specifically, it is the result of substituting the approximate solution into the differential equation and computing the difference between the left-hand side and the right-hand side. A residual of zero indicates that the solution exactly satisfies the equation, while a nonzero residual reflects the error or mismatch. For Poisson's equation, the residual is defined as shown on equation 4

$$\mathcal{R} = \nabla \cdot (k\nabla\hat{u}) - f \qquad [4]$$

Thus, as shown on equation 5, to approximate an inverse problem, a PINN minimizes the residual of a PDE and the MSE of observed solutions. This loss function is perfectly applicable to KANs, and physically informed Kolmogorov-Arnold networks are described just that way.

$$\mathcal{L}_{\mathrm{PDE}}(\theta) = \frac{1}{N_{\mathrm{PDE}}}\sum_{i=1}^{N_{\mathrm{PDE}}}\|\mathcal{R}_{\mathcal{F}}(x_i)\|^2 + \frac{1}{N_{obs}}\sum_{i=0}^{N_{obs}}||\hat{u} - u_{real}||^2 \qquad [5]$$

This loss can be minimized using various Algorithms. Adam is an adaptive optimization algorithm that adjusts learning rates using estimates of first and second moments of the gradients, making it efficient for training deep networks (6). L-BFGS is a memory-efficient quasi-Newton method that approximates second-order information to optimize smooth functions with faster convergence than gradient descent.

2. Literature Review

In the paper introducing KANs, it is shown that, for solving partial differential equations, a two-layer KAN with 10 neurons per layer is 100 times more accurate than an MLP with four layers of width 100 (error of 10^{-7} versus 10^{-5} in terms of mean squared error) and 100 times more efficient in terms of parameters (7). However, it is noted that KANs are significantly slower than MLPs during training due to their non-linear nature. This paper made no mention of KAN for inverse problems.
In (8), the authors introduce KAN as a tool for solving inverse problems, demonstrating a comparable approximation error compared to PINNs, particularly in equations with nonlinearities. They also propose Physics-Informed KANs (PIKANs) to tackle highly complex systems, such as those found in fluid dynamics, positioning KANs as an alternative to MLPs.
Zhang et al (9) performed full wave inversion (FWI) using PINNs in order to solve the inverse problem of the wave equation to get the values of wave speed underground. They proposed to use two different neural networks, one that solves the direct problem (finding u) and one that solves the inverse problem (finding c), while training them in parallel using the same loss function.

3. Methodology

The project was developed in four main phases: mathematical review and understanding, computational formulation of the problem, obtaining and analyzing results, and documentation. Below, these stages are briefly described.

A. Mathematical Review and Understanding. This phase involves reviewing the literature on KAN, PINN, and PIKAN, along with a detailed study of the Poisson equation. Sources will include academic journals, articles, and textbooks, focusing on solving inverse problems.

B. Computational Formulation of the Problem. The following steps were taken in order to formulate the problem.

B.1. Sampling of training points. As mentioned, the domain of interest for the inverse problem is unbounded. Therefore, if training points are sampled uniformly, their density will be evenly distributed across the domain. As a result, the gradients may not converge effectively outside the region of interest, leading to poor training performance in those areas. We propose sampling the points using a normal distribution with mean in the zone of interest. That way, for equations where the solution tends to stabilize when going to infinity, there are less collocation points as one moves from the zone of interest. However, since solutions are assumed to become constant when reaching infinity, the gradients will become zero, and this sampling approach will help the network to generalize outside the training domain. Figure 1 provides a graphic explanation of the sampling strategy used.

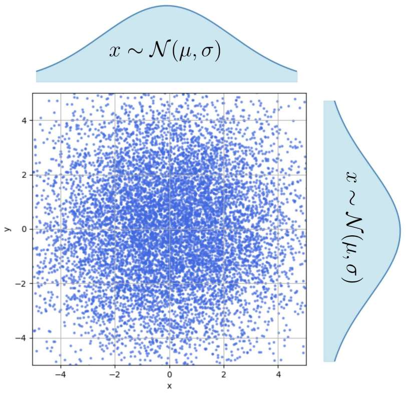

Fig. 1. Training points selection using a normal distribution

B.2. Network structure to solve inverse problem. The network designs include two fully connected networks, $\mathcal{NN}^u$, which solves the direct problem and $\mathcal{NN}^k$, trained to solve the inverse problem. The reasoning behind creating two networks instead of one fully connected network with two outputs comes from the idea of not having the solution for u bias the solution for k. Figure 2 shows the structure of the networks.

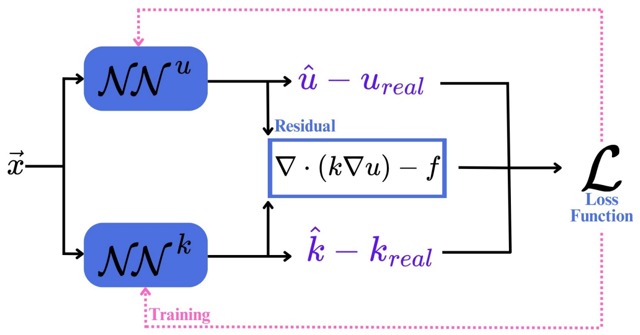

Fig. 2. Structure of training points selection

Note that the output of each network is derived using automatic differentiation (10) and the residual is calculated. The loss function to be minimized is then taken, taking the residual and the observed data. Both networks are trained at the same time, with the same loss function, in order to influence by some degree the solution.

B.3. Loss function. The loss function to minimize is formed by minimizing the residual of Poisson's equation (see equation 4) and adding that to the MSE for observations of both u and k. Given N_{pde} training points and N_{obs} observations of the solution, the losses that will make up the loss function are described as follows:

$$\mathcal{L}_{\text{PDE}}(\theta) = \frac{1}{N_{\text{PDE}}} \sum_{i=1}^{N_{\text{PDE}}} \|\mathcal{R}_{\mathcal{F}}(x_i)\|^2 \quad [6]$$

$$\mathcal{L}_{\text{u}} = \frac{1}{N_{obs}} \sum_{i=0}^{N_{obs}} ||\hat{u} - u_{real}||^2 \quad [7]$$

$$\mathcal{L}_{\text{k}} = \frac{1}{N_{obs}} \sum_{i=0}^{N_{obs}} ||\hat{k} - u_{k}||^2 \quad [8]$$

The total loss is then calculated by:

$$\mathcal{L} = \lambda_1 \mathcal{L}_{\text{PDE}} + \lambda_2 \mathcal{L}_{\text{u}} + \lambda_3 \mathcal{L}_{\text{k}} \quad [9]$$

Where λ_i refers to the "importance" that each loss should have.

B.4. Two step minimization (ADAM + L-BFGS). The training stage was defined to be carried out in two stages. Initially, the Adam optimizer was employed to rapidly explore the loss landscape and bring the network parameters close to a local minimum. Then, the L-BFGS algorithm, a quasi-Newton, was applied to refine the solution and achieve higher accuracy. This hybrid strategy leverages the robustness of Adam in navigating complex, high-dimensional spaces while being a stochastic method and the precision of L-BFGS in fine-tuning the model parameters, as it is a second order method.

C. Obtaining and Analyzing Results. In this phase, the network will be trained to solve the Poisson inverse problem, optimizing its hyperparameters (number of layers and neurons per layer). The results will be validated using metrics such as the mean squared error, and the findings will be analyzed and discussed. The problem will be tackled using PINNs and PIKAN.

C.1. Equation to solve. Using a method for manufactured solutions (11), we can construct the equation that we wish to solve.
We let the analytical solution of the equation and the real value of k to be as follows:

$$u(x, y) = e^{-\alpha(x^2+y^2)} \cos(\beta y) \quad [10]$$

$$k(y) = -1 + \frac{2}{1 + e^{-y/\epsilon}} \quad [11]$$

We manufacture the equation by taking equations 10 and 11 and placing them into Poissons equation, to get a manufactured value of the source term f, and thus, creating the Poisson equation that we wish to solve. Figure 3 shows the analytical and expected values for u and k, plotted over the domain of interest.

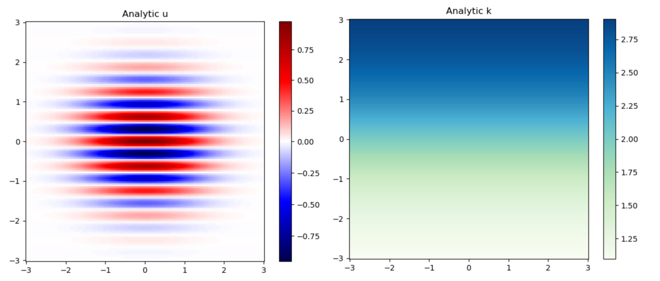

Fig. 3. Analytic solution of the problem to solve

C.2. Network parameters. To understand the configuration of both networks presented, some definitions regarding the parameters of the implementation must be clarified:

- Hidden layer: an intermediary layer positioned between the input and output layers of the neural network. It is responsible for transforming the input data into abstract representations through learnable parameters.

- Neurons per layer: Defines the number of nodes (activation functions) that are present on each hidden network of a network

Some parameters that are present only on KAN are as follows:

- Grid: Number of control points on the splines at every node.

- K: Degree of the interpolator polynomials that make up the B-splines.

C.3. KAN Configuration. A PIKAN with the following parameters was trained for 3000 epochs using ADAM's algorithm and for 500 epochs using the L-BFGS algorithm, using a discretization of 10,000 training points:

- Number of hidden layers: 5
- Number of neurons per layer: 10
- Grid: 4
- K: 4
- Number of parameters: 18,480

Due to the elevated training times (45 mins per epoch using L-BFGS), a bigger network was not considered.
Figure 4 shows a schematic of how a trained KAN with the structure purposed looks. Note that each "node" contains three control points, shown as functions. The shade of the plot describes how important each activation function is for the output of the model. Every network constructed was built and trained using the open source **pykan** implementation (12) on python *3.11.4*.

C.4. PINN configuration. A PINN with the following parameters was trained for 10,000 epochs using ADAM's algorithm and for 500 epochs using the L-BFGS algorithm, using a discretization of 10,000 training points:

- Number of hidden layers: 16
- Number of neurons per layer: 64
- Number of parameters:125,314

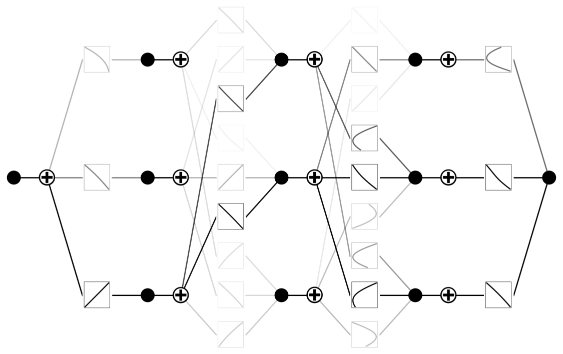

Fig. 4. Structure of a trained KAN. The intensity of each edge represents it's importance. *taken from* (4)

D. Documentation. Documentation will be conducted throughout the project, including interim reports and a final report. A GitHub repository will be used to ensure accessibility and reproducibility of the work.

4. Results

A. PINNs. A set of PINNs was trained using the network exposed previously. Figure 5 shows the training of the network when it is 22% trained. Figure 6 shows the PINN completely trained, and shows that the network can generalize on the unbounded domain. The network takes 18 : 49 minutes to be trained.

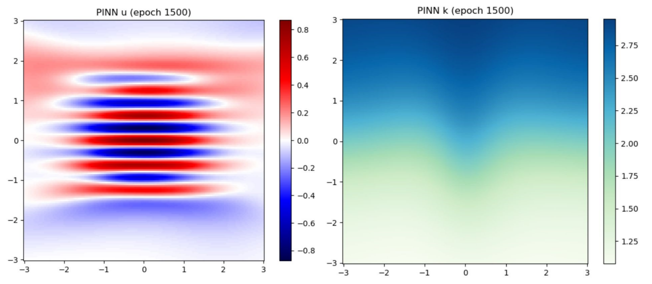

Fig. 5. PINN Training - Epoch 1,500

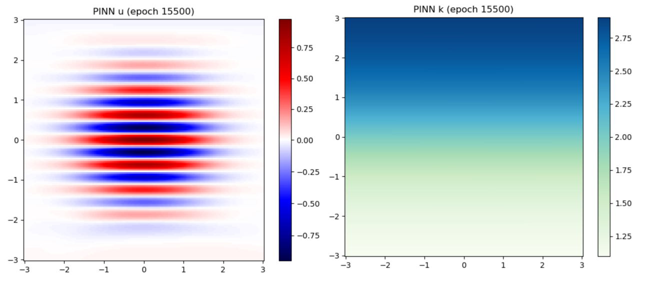

Fig. 6. PINN Training - Epoch 10,050

The network was trained using observed data with no noise, with 10% noise and with 20% noise. Note by figure 7 that, as expected, the algorithm converges to a significantly lower loss when trained without noisy data, mostly due to the ability that L-BFGS has to minimize. However, it should be noted that when exposed to noise, the L-BFGS does not have the outstanding performance that it has on clean data. This happens because L-BFGS is a deterministic algorithm, opposed to ADAM, which is stochastic.

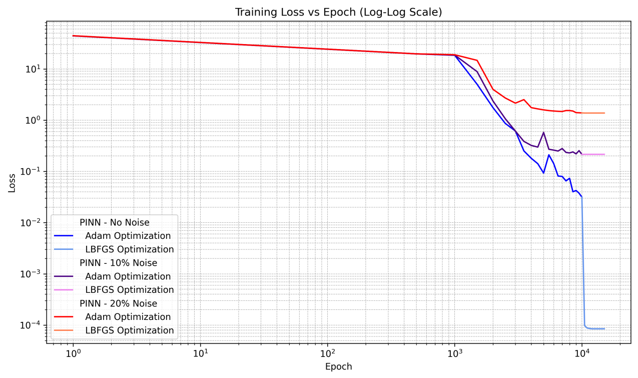

Fig. 7. Log-Log plot of loss during training

Figure 8 shows output of the model when trained with 20% noise. Note that the value of u is not as well generalized as in figure 6, and the learned value of k presents fluctuations with respect to the analytic solution.

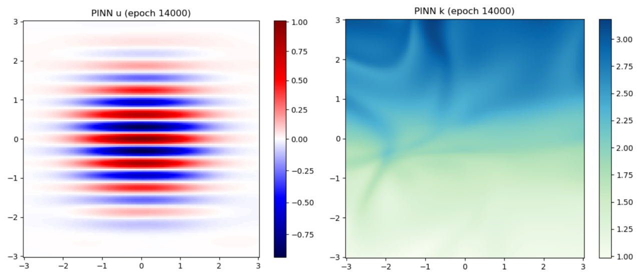

Fig. 8. Log-Log plot of loss during training

B. PIKANs. A set of PIKAN was trained using the network exposed previously. Figure 9 shows the PINN on epoch 10,000, and shows that the network can't really generalize on the unbounded domain. The network takes an outstanding 645 : 32 minutes to be trained.

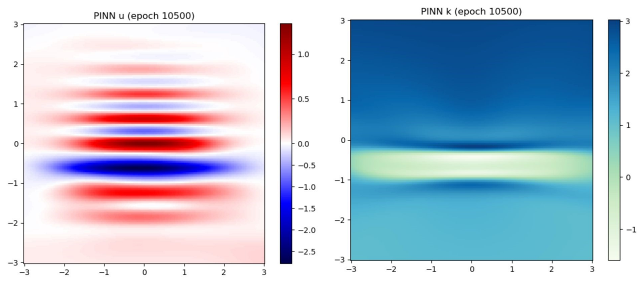

Fig. 9. PIKAN Training - Epoch 10,500

Note that even though u and k can be somehow approximated, the network doesn't even come close to generalizing the domain as well as the PINN does. In fact, the loss function of none of the PIKANs never goes lower than 10^{-1}, and when trained with noise, the loss function doesn't even reach 1.0. It is thought that this behavior occurs because, as KAN produces polynomials, polynomials tend to grow when approaching infinity, so no generalization is obtained. The opposite happens with PINNS, when activation functions, such as $\tan^{-1}$ do not grow when approaching infinity.

C. Comparison of both networks due to noise. Given the analytic solution for k described in previous sections, the relative error between $\hat{k}$ and k is computed. Table 1 shows the relative error calculated, noting that PINNs produce highly more accurate results.

Noise	PINN	KAN
No noise	0.057%	10.95%
10% noise	2.85%	23.4%
20% noise	6.7%	37.45%

Table 1. Relative errors for k for the inverse Poisson problem

5. Conclusions

- **This study demonstrates the potential and limitations of PINNs in solving inverse problems**: While both PINNs and PIKANs offer innovative strategies for parameter estimation in partial differential equations, their performance varies significantly depending on the domain, noise levels in the data and presence of boundary conditions, giving a clear advantage to PINNs.
- **PINNs outperform PIKANs in inverse problems on unbounded domains**Despite the theoretical advantages of KAN, the results demonstrate that PINNs generalize significantly better and yield much lower relative errors in estimating k, especially under noisy conditions.
- **Two-step optimization improves convergence but reveals limitations of model architecture** Combining Adam and L-BFGS optimizers provided effective convergence in both models, yet the performance gap persisted—highlighting that architectural design plays a role on the optimization strategy in this context.

6. Acknowledgements

A thanks to my wonderful advisor Nicolás Guarín-Zapata who guided me through the formulation and solution of the problem. Special thanks to Professor Elena Montilla for her guidance on how to formulate proper methodologies and objectives. I am, as always, most grateful for the unconditional support from my mother, brother, and aunt and extend my heartfelt gratitude to my family and friends.

7. References

1. G Evans, J Blackledge, P Yardley, *Numerical Methods for Partial Differential Equations*, Springer Undergraduate Mathematics Series. (Springer-Verlag, London, UK), (1999).
2. A Tarantola, *Inverse Problem Theory and Methods for Model Parameter Estimation.* (Society for Industrial and Applied Mathematics, Philadelphia, PA), (2005).
3. M Raissi, P Perdikaris, GE Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *J. Comput. Phys.* **378**, 686–707 (2018).
4. Z Liu, et al., Kolmogorov-arnold networks (kans). *arXiv preprint* (2024) Preprint, under review.

5. R Yu, W Yu, X Wang, Kan or mlp: A fairer comparison. *arXiv preprint* (2024) Preprint, under review.
6. DP Kingma, J Ba, Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).
7. Y Wang, et al., Kolmogorov-arnold-informed neural network: A physics-informed deep learning framework for solving pdes based on kolmogorov-arnold networks. *arXiv preprint* (2024) arXiv:2406.19756v1 [cs.LG].
8. Y Wang, et al., Kolmogorov-arnold-informed neural network: A physics-informed deep learning framework for solving pdes based on kolmogorov-arnold networks. *arXiv preprint arXiv:2406.19756* (2024).
9. Y Zhang, X Zhu, J Gao, Seismic inversion based on acoustic wave equations using physics-informed neural network. *IEEE Transactions on Geosci. Remote. Sens.* **61**, 4500511 (2023).
10. J Hare, An introduction to automatic differentiation (Lecture notes, Vision, Learning and Control Group, University of Southampton) (2023) Available at: https://rufflewind.com/2016-12-30/reverse-mode-automatic-differentiation.
11. L Shunn, F Ham, Method of manufactured solutions applied to variable-density flow solvers in *Center for Turbulence Research Annual Research Briefs*. (Stanford University, Stanford, CA), pp. 155–168 (2007).
12. X Liu, Pykan: Implementation of kolmogorov-arnold networks (kans) in python (https://github.com/KindXiaoming/pykan) (2023) Accessed: 2024-11-19.