Nuclear Science and Techniques (2023) 34:161
https://doi.org/10.1007/s41365-023-01313-0

Physics-constrained neural network for solving discontinuous interface K-eigenvalue problem with application to reactor physics

Qi-Hong Yang[1] · Yu Yang[1] · Yang-Tao Deng[1] · Qiao-Lin He[1] · He-Lin Gong[2] · Shi-Quan Zhang[1]

Received: 7 June 2023 / Revised: 28 July 2023 / Accepted: 25 August 2023

Abstract
Machine learning-based modeling of reactor physics problems has attracted increasing interest in recent years. Despite some progress in one-dimensional problems, there is still a paucity of benchmark studies that are easy to solve using traditional numerical methods albeit still challenging using neural networks for a wide range of practical problems. We present two networks, namely the Generalized Inverse Power Method Neural Network (GIPMNN) and Physics-Constrained GIPMNN (PC-GIPIMNN) to solve K-eigenvalue problems in neutron diffusion theory. GIPMNN follows the main idea of the inverse power method and determines the lowest eigenvalue using an iterative method. The PC-GIPMNN additionally enforces conservative interface conditions for the neutron flux. Meanwhile, Deep Ritz Method (DRM) directly solves the smallest eigenvalue by minimizing the eigenvalue in Rayleigh quotient form. A comprehensive study was conducted using GIPMNN, PC-GIPMNN, and DRM to solve problems of complex spatial geometry with variant material domains from the field of nuclear reactor physics. The methods were compared with the standard finite element method. The applicability and accuracy of the methods are reported and indicate that PC-GIPMNN outperforms GIPMNN and DRM.

Keywords Neural network · Reactor physics · Neutron diffusion equation · Eigenvalue problem · Inverse power method

1 Introduction

In the nuclear engineering domain, the fundamental mode solution of the K-eigenvalue problem based on the steady-state multigroup neutron diffusion theory is crucial for simulation and analysis of nuclear reactors. The eigenvalue equation can be expressed as follows:

$$
\begin{aligned}
&-\nabla\cdot(D_g\nabla\phi_g)+\Sigma_g^R\phi_g-\sum_{g'\neq g}\Sigma_{g'\to g}\phi_g' \\
&=\frac{1}{K}\chi_g\sum_{g'}\nu\Sigma_{g'}^{\mathrm{f}}\phi_g' \quad g=1,2,...,G,
\end{aligned}
\tag{1}
$$

where $\phi_g\equiv\phi_g(\mathbf{r})$ denotes the neutron flux at spatial point $\mathbf{r}$ in the g-th energy group, D_g, Σ_g^R, $\Sigma_{g'\to g}$, χ_g and $\nu\Sigma_{g'}^{\mathrm{f}}$ denote spatially dependent (possibly discontinuous) parameters that reflect the material properties in a reactor core [1]. Nuclear engineers and analysts must numerically determine the fundamental mode eigenvalue (commonly called $\mathrm{K_{eff}}$) and the corresponding eigenvector for a given geometry/material configuration. For numerical solution, Eq. (1) must be discretized and reduced to a set of G-coupled algebraic equations, which can be expressed using matrices as follows:

$$
M\Phi=\frac{1}{K}F\Phi. \tag{2}
$$

This is often termed as the generalized eigenvalue problem because coefficient matrices occur on both sides of the equation. Many mature numerical methods, such as the finite

This work was partially supported by the National Natural Science Foundation of China (No. 11971020), Natural Science Foundation of Shanghai (No.23ZR1429300), and Innovation Funds of CNNC (Lingchuang Fund).

✉ Qiao-Lin He
qlhejenny@scu.edu.cn

✉ He-Lin Gong
gonghelin@sjtu.edu.cn

[1] School of Mathdematics, Sichuan University, Chengdu 610065, China

[2] Paris Elite Institute of Technology, Shanghai Jiao Tong University, Shanghai 200240, China

Published online: 13 November 2023

difference method [2], nodal collocation method [3], finite-element method [4–6], and nodal expansion method [7–9], have been proposed to solve neutron diffusion equations. Among the methods, the nodal expansion method is widely used because it is easier to implement and requires less computational effort than other methods. The power iteration method is the most well-known numerical method for solving the principal K-eigenvalue [10]. A detailed review of conventional numerical methods for solving K-eigenvalue problems is available in recent nuclear reactor physics textbooks [1, 11, 12]. Although traditional and mature numerical methods are currently widely used in nuclear reactor physics to solve K-eigenvalue problems with acceptable engineering accuracy and computational cost, state-of-the-art neural networks to solve K-eigenvalue problems are still in the infancy stages. However, neural networks exhibit the potential to provide another option to solve nuclear engineering problems with the progress of new algorithms and hardware.

As mentioned above, many traditional numerical methods were proposed to solve neutron diffusion equations. However, a dense mesh is required to ensure highly accurate results. The model consumes more computational resources if the mesh is extremely dense. Inaccurate solutions are obtained if the mesh is coarse. Moreover, for high-dimensional problems, classical methods are either less efficient or not successful due to the problems involving dimensionality. A neural network can potentially enhance its performance by using a capable hypothesis space due to its relatively low statistical error [13].

Neural networks exhibit multiple potential benefits in solving nuclear engineering problems when compared with conventional numerical methods.

- They provide mesh-free solutions to approximate physics fields in nuclear reactors, in which mesh generation is significantly complex due to high heterogeneity of geometry and material.
- They provide a general framework to solve high-dimensional problems governed by parameterized PDEs, particularly for original neutron transport equation.This equation comprises seven variables: three spatial variables, two directional variables, one energy variable, and one temporal variable.
- They are able to seamlessly incorporate prior data (potentially with noise) into existing algorithms given that data assimilation is necessary as a post-process for conventional numerical methods [14–19].
- They provide a general framework to solve inverse problems with hidden physics. Conversely, they are typically prohibitively expensive and require different formulations and elaborate computer codes for conventional numerical methods.

In recent years, neural networks have been widely used to solve Partial Differential Equations (PDEs) [20] and achieved remarkable success.

Based on neural networks, a large number of methods were proposed to solve PDE, such as the deep backward stochastic differential equation (BSDE) method [21, 22], deep Galerkin method (DGM) [23], deep Ritz method (DRM) [24], and Physics-Informed Neural Network (PINN) [25]. The deep BSDE method reformulates PDEs using backward stochastic differential equations and approximates the gradient of an unknown solution using neural networks. Although DGM and PINN appear independently under two names in the literature, they are similar. They both train neural networks by minimizing the mean squared error loss of the residual equation. However, DRM reformulates the original problem into a variational problem and trains neural networks by minimizing the energy function of the variational problem. Specifically, DRM and PINN [25] attracted widespread attention. Extensive studies focused on these methods to solve a variety of problems, including fiber optics [26], hyperelasticity [27], solid mechanics [28], heat transfer problems [29–31], inverse problems [32–35], and uncertainty quantification [36–39]. However, a few studies focused on solving eigenvalue problems [24, 40–45].

The need to solve eigenvalue problems can be traced back to 2018 [24]. A deep Ritz method to solve variational problems was proposed, and several examples elucidated on how to use DRM to solve eigenvalue problems [24]. First, the original eigenvalue problem was transformed into a variational problem. Then, a specially defined loss function was constructed, termed as the Rayleigh quotient, using the variational principle [46]. The Rayleigh quotient is a well-known approximation of the eigenvalue of matrix $\boldsymbol{A}$, which is defined by $R=\frac{\boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x}}{\boldsymbol{x}^{\mathrm{T}}\ \boldsymbol{x}}$. Finally, they minimized the loss function and obtained the smallest eigenvalue. Similarly, some studies [41, 42] directly used PINN to solve eigenvalue problems. In contrast to DRM transferring the original problem to a variational problem, the PINN solves eigenvalue problems without variation. Neural networks are used in PINN to represent the function, and automatic differentiation (AD) [47] is used to acquire the vector impacted by the differential operators. The loss function is obtained using the Rayleigh quotient. The smallest eigenvalue and corresponding eigenvector were then solved using optimization tools. Moreover, the deep forward-backward stochastic differential equation (FBSDE) method [48] was proposed to solve eigenvalue problems, which is an expansion of the deep BSDE method. In the method, the eigenvalue problem is reformulated as a fixed-point problem of semigroup flow induced by the operator. Other studies [43, 44] proposed an alternative method to learn the eigenvalue problem by adding one or two regularization terms to the loss function. More recently, Ref.

[49], a neural network framework based on the power method [50] was presented to solve eigenvalue problems and smallest eigenvalue problems, where the eigenfunction is expressed by the neural network and iteratively solved following the idea of the power method or inverse power method. However, the scope was limited to linear operators and certain special eigenvalue problems.

In a recent study, PINN was applied to solve neutron diffusion equations [45, 51], where the authors used a free learnable parameter to approximate the eigenvalue and a novel regularization technique to exclude null solutions from the PINN framework. A conservative physics-informed neural network (cPINN) was proposed in discrete domains for nonlinear conservation laws [52]. Moreover, cPINN [53] was applied to solve heterogeneous neutron diffusion problems in one-dimensional cases, which develops PINN for each subdomain and considers additional conservation laws along the interfaces of subdomains (a general consideration in reactor physics [11]), which is involved in neural network training as the variable to be optimized.

More recently, a data-enabled physics-informed neural network (DEPINN) [40] was proposed to solve neutron diffusion eigenvalue problems. To achieve acceptable engineering accuracy for complex engineering problems, it is suggested that a very small amount of prior data from physical experiments be used to improve the training accuracy and efficiency. In contrast to PINN, which solves the neutron diffusion eigenvalue problem directly, an autoencoder-based machine learning method in combination with the reduced-order method [54, 55] was proposed [56] to solve the same problem. However, it still relies on solving governing equations with traditional numerical methods such as the finite difference method.

Although DRM provides a way to solve eigenvalue problems with neural networks, as shown in Sect. 4.2.2, results indicate that DRM is not stable when solving two-dimensional cases. First, DRM learns the eigenvalue and eigenfunction at the early stage of the training process. Subsequently, DRM attempts to learn a smaller eigenvalue after it is close to the true eigenvalue. Finally, DRM successfully learns one smaller eigenvalue that may be close to the true eigenvalue and one incorrect eigenfunction that is meaningless and far from the true eigenfunction. Additionally, the framework of a neural network based on the power method [50] is unsuitable for generalized eigenvalue problems. Therefore, it is necessary to propose a new algorithm to solve K-eigenvalue problems.

The study focuses on eigenvalue problems, which are also interface problems in which the eigenfunctions are continuous on the interface, and the derivatives of the eigenfunction are not continuous at the interface. Specifically, in the nuclear reactor physics domain, this is a general problem in which the reactor core is composed of fuel assemblies of different fissile nuclides enrichments [1]. Some studies focused on the use of neural networks to solve elliptic interface problems. Some researchers [57] use the idea of DRM and formulated the PDEs into the variational problems, which can be solved using the deep learning approach. They presented a novel mesh-free numerical method for solving elliptical interface problems based on deep learning [58]. They employed different neural networks in different subdomains and reformulated the problem as a least-squares problem. A similar case exists, in which the authors [53] enforce the interface conditions using piecewise neural network. In contrast to the methods, a discontinuity-capturing shallow neural network (DCSNN) [59] has been proposed for elliptic interface problems. The crucial concept of DCSNN is that a d-dimensional piecewise continuous function can be extended to a continuous function defined in $(d + 1)$-dimensional space, where the augmented coordinate variable labels the pieces of each subdomain. However, to the best of the authors' knowledge, only a few studies focused on the use of neural networks to solve eigenvalue problems that incorporate interface problems involving regions of different materials. However, challenges exist at least on three fronts.

- Designing a neural network that is more suitable for K-eigenvalue problems for more complicated/medium-size test problems.
- Dealing with the interface problem in a more general and understandable manner when designing the neural network.
- Proposing a framework effectively enhances the robustness of the neural network and improves the efficiency of utilizing the noisy prior data.

To address the aforementioned challenges and advance beyond the state-of-the-art research [45, 51, 53], we initially introduced the study [40]. This served as a preliminarily demonstration of the applicability of the PINN approach to reactor physics eigenvalue problems in complex engineering scenarios. The contributions of this study are as follows.

- Firstly, we extend the Inverse Power Method Neural Network (IPMNN)[49] to the so-called Generalized Inverse Power Method Neural Network (GIPMNN) to solve the smallest eigenvalue and the related eigenfunction of the generalized K-eigenvalue problems. Compared to DEPINN from our previous study [53], we omit the prior data in the training process and attempt to solve the K-eigenvalue problems from a data-free mathematical/numerical perspective.
- Then, we propose a Physics Constrained GIPMNN (PC-GIPMNN) to address the interface problem in a more general and understandable manner with respect to previous studies [45, 51, 53].

- Finally, we conduct a thorough comparative study of GIPMNN, PC-GIPMNN, and DRM using a variety of numerical tests. We evaluate the applicability and accuracy of these three methods using typical 1D and 2D test cases in reactor physics, particularly accounting for material discontinuities in different geometries. In the 1D example, we determine the optimal ratio of outer to inner iterations, a finding that may be particularly relevant for GIPMNN. Additionally, we observe the failure of DRM in the 2D experiments, whereas PC-GIPMNN consistently outperforms GIPMNN and DRM over a set number of epochs.

The rest of this paper is organized as follows. The governing equations for the eigenvalue problems are presented in Sect. 2. In Sect. 3, we propose GIPMNN and PC-GIPMNN and introduce DRM in our cases. In Sect. 4, the results of 1D and 2D test cases are listed to verify the three methods. Finally, conclusions and future research are discussed in Sect. 5.

2 K-eigenvalue problems

This section introduces the equations that govern the neutron criticality over a spatial domain. We recall Eqs. (1) and (2). The generalized K-eigenvalue problem can be formulated as follows:

$$\begin{cases} \mathcal{L}\phi = \lambda \mathcal{Q}\phi, & \text{in } \Omega, \\ \mathcal{B}\phi = g, & \text{on } \partial\Omega, \end{cases} \tag{3}$$

where domains $\Omega \subset \mathbb{R}^{\mathrm{d}}$, $\mathcal{L}$, $\mathcal{Q}$, and $\mathcal{B}$ denote the differential operators acting on the functions defined in the interior of Ω and at the boundary of $\Omega(\ \partial\Omega)$. Furthermore, ϕ denotes the eigenfunction of the system and λ denotes the associated eigenvalue. In this preliminary study, inspired by the notable work in [56], we utilize the one-group steady-state diffusion equation for criticality, framing it as a generalized eigenvalue problem. It is expressed as:

$$-\nabla \cdot (D\nabla\phi) + \Sigma^{\mathrm{a}}\phi = \lambda v \Sigma^{\mathrm{f}}\phi, \tag{4}$$

where eigenfunction ϕ denotes the neutron flux, which is a scalar quantity used in nuclear reactor physics. It corresponds to the total length travelled by all free neutrons per unit time and volume. Σ^{a}, Σ^{f}, and v denote the absorption cross section, fission cross section, and average number of neutrons produced per fission event, respectively. We follow [56] and use the diffusion coefficient approximation

$$D = \frac{1}{3(\Sigma^{\mathrm{a}} + \Sigma^{\mathrm{s}})}. \tag{5}$$

Here, Σ^{s} denotes the cross-section where a neutron scatters in a different direction. The eigenvalue λ is obtained by multiplying the neutron source in Eq. (4). The value balances the terms that produce neutrons with those that account for the losses. This is defined as the reciprocal of k_{eff}, i.e., $\lambda = \frac{1}{k_{\mathrm{eff}}}$, where

$$k_{\mathrm{eff}} = \frac{\text{number of neutrons in one generation}}{\text{number of neutrons in the preceding generation}}. \tag{6}$$

Two main boundary conditions are imposed on the diffusion equation. One condition represents a surface on which neutrons are reflected back into the domain (reflective condition), and the other condition represents surfaces that allow neutrons to escape from the system (vacuum or bare condition). Both the conditions are satisfied by relating the flux solution to its gradient on the boundary, i.e.,

$$-\frac{1}{2}D\nabla\phi \cdot \boldsymbol{n} = \begin{cases} \frac{1}{4}\phi & \text{bare surface}, \\ 0 & \text{reflective surface}, \end{cases} \tag{7}$$

where $\boldsymbol{n}$ denotes an outward point normal to the surface.

2.1 PINN as a Eigenvalue solver

In this subsection, we discuss using PINN to solve the generalized eigenvalue problem (Eq. (2)).

The eigenfunction of the operator $\mathcal{L}$ is approximated by $\mathcal{N}^{\theta}$, i.e., $\phi(\boldsymbol{x}) = \mathcal{N}^{\theta}(\boldsymbol{x})$. Then, $\mathcal{L}\phi$ and $\mathcal{B}\phi$ can be computed using the AD. For the boundary conditions: A penalty term (Eq. (8)) is added to the loss function in PINN, which penalizes the discrepancy between the approximated value on the boundary and exact boundary condition, where N_{b} denotes the number of sampling points on the boundary $\partial\Omega$ and $\boldsymbol{x}_i$ is one point in the sampling set $\{\boldsymbol{x}_i\}_{i=1}^{N_{\mathrm{b}}}$.

$$Loss_{\mathrm{b}} = \sum_{i=1}^{N_{\mathrm{b}}} |\mathcal{B}\Phi(\boldsymbol{x}_i) - g(\boldsymbol{x}_i)|^2. \tag{8}$$

$$\lambda = \frac{\langle \mathcal{L}\phi, \phi\rangle}{\langle \phi, \phi\rangle}. \tag{9}$$

As mentioned previously, we are concerned with the smallest eigenvalue and associated eigenfunction. Hence, the eigenvalue (Rayleigh quotient) is viewed as a loss term in PINN to attain the lowest eigenvalue.

Finally, the total loss function (Eq. (10)) in PINN corresponds to the weighted sum of the two objectives (Eq. (8)) and (Eq. (9)), where α and β corresponds to the weights.

$$Loss_{\mathrm{total}} = \alpha\lambda^2 + \beta \mathrm{Loss}_{\mathrm{b}}. \tag{10}$$

Unfortunately, PINN does not work for the cases in the study and even performs worse than DRM. Therefore, we only compared the results of our method with those of DRM.

Remark 1 It should be noted that the square of the eigenvalue is used as the loss function in PINN. Given that the smallest eigenvalue implies that the absolute value is the smallest, PINN attempts to determine a negative infinity value without using a square term.

3 Methodologies

In this section, we extend our previous study [49] and discuss the use of a neural network to numerically solve the smallest eigenvalue problem. The main concept is to use a neural network to approximate the eigenfunction and compute the eigenvalue based on the Rayleigh quotient using an eigenvector expressed by the points calculated using the eigenfunction.

3.1 Neural network architecture

Next, we discuss the neural network structure used to approximate eigenfunction $\phi(\boldsymbol{x})$. The neural network architecture employed in the study is the same as ResNet [60], which is built by stacking several residual modules. It is one of the most popular models used in Deep Learning, and it is also commonly used in the field for solving PDEs via neural networks [24, 61, 62]. Each module has one skip connection, and each block consists of two fully connected layers as shown in Fig. 1.

In the network, let $\boldsymbol{x}, \boldsymbol{x}_k \in \mathbb{R}^d$ be the input and let $\boldsymbol{W}_l$ and $\boldsymbol{b}_l$, $l = 1, 2, 3, 4$ and 5 be the parameters in the fully connected layers. We use $\boldsymbol{W}_{\mathrm{r}k}$ and $\boldsymbol{b}_{\mathrm{r}k}$ and $k = 1$ and 2 to denote the parameters in the residual connections. The results $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$ for the modules can be expressed as follows:

$$\boldsymbol{s}_1 = \sigma(\boldsymbol{W}_2(\sigma(\boldsymbol{W}_1\boldsymbol{x} + \boldsymbol{b}_1)) + \boldsymbol{b}_2) + \boldsymbol{r}_1(\boldsymbol{x}), \tag{11}$$

$$\boldsymbol{s}_2 = \sigma(\boldsymbol{W}_4(\sigma(\boldsymbol{W}_3\boldsymbol{s}_1 + \boldsymbol{b}_3)) + \boldsymbol{b}_4) + \boldsymbol{r}_2(\boldsymbol{s}_1), \tag{12}$$

where σ denotes the activation function and is chosen as tanh. Furthermore, $\boldsymbol{r}_{\mathrm{k}}$, $k = 1$, and 2 are functions in the residual connections, which can represented as follows:

$$\boldsymbol{r}_k(\boldsymbol{x}_k) = \sigma(\boldsymbol{W}_{\mathrm{r}k}\boldsymbol{x}_k + \boldsymbol{b}_{\mathrm{r}k}). \tag{13}$$

Subsequently, the neural network $\mathcal{N}^\theta(\boldsymbol{x})$ is expressed as

$$\mathcal{N}^\theta(\boldsymbol{x}) = \boldsymbol{W}_5\boldsymbol{s}_2(\boldsymbol{s}_1(\boldsymbol{x})) + \boldsymbol{b}_5. \tag{14}$$

Therefore, the eigenfunction $\phi(\boldsymbol{x}) = \mathcal{N}^\theta(\boldsymbol{x})$, where θ denotes neural network parameters.

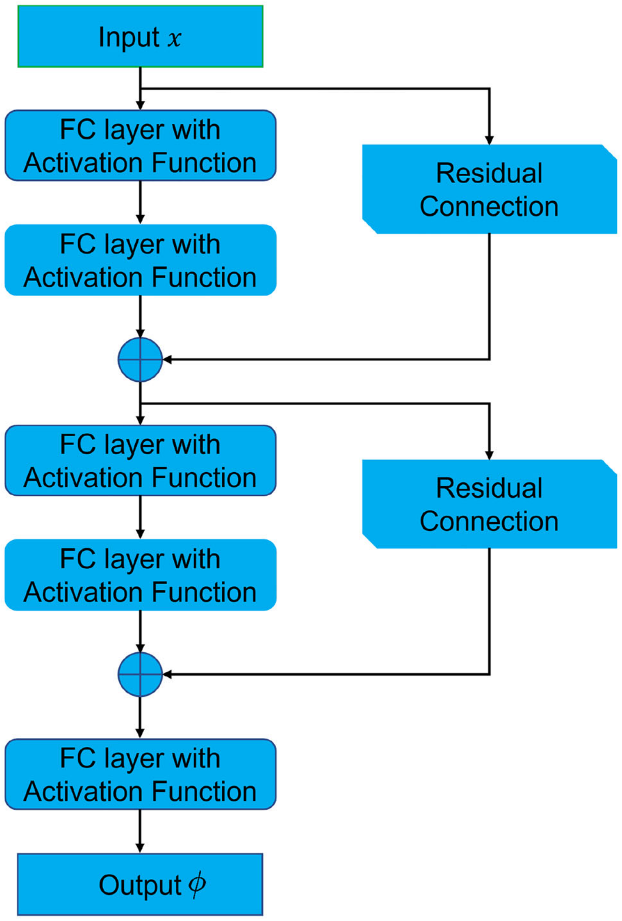

Input x
FC layer with Activation Function
FC layer with Activation Function
Residual Connection
FC layer with Activation Function
FC layer with Activation Function
Residual Connection
FC layer with Activation Function
Output ϕ

Fig. 1 Neural network architecture consists of two modules and one linear output layer. Each module contains two fully-connected layers and a skip connection

Remark 2 Given that $\phi(\boldsymbol{x})$ is the scalar of the neutron population and denotes the density of the free-moving neutron distribution over the spatial domain, we obtain $\phi(\boldsymbol{x}) \geq 0$. Therefore, the eigenfunction can be expressed as $\phi(\boldsymbol{x}) = (\mathcal{N}^\theta(\boldsymbol{x}))^2$.

3.2 Recap of inverse power method neural network

We consider the following linear eigenvalue problem:

$$\mathcal{L}\phi = \lambda\phi, \text{ in } \Omega, \tag{15}$$

which differs from the generalized eigenvalue problem $\mathcal{L}\phi = \lambda\mathcal{Q}\phi$.

Inspired by the idea of the inverse power method, IPMNN [49] was proposed to solve for the smallest eigenvalue and associated eigenfunction. Equation (16) depicts the key step of the inverse power method, and equation (17) is analogous to equation (16), where $\boldsymbol{A}$ denotes a matrix, and λ_{k-1} and

$\boldsymbol{\phi}_{k-1}$ denote the results of previous iteration. Therefore, λ_k and $\boldsymbol{\phi}_k$ are obtained by using Eq. (16).

$$\begin{cases} \boldsymbol{p}_k = A^{-1}\boldsymbol{\phi}_{k-1}, \\ \boldsymbol{\phi}_k = \dfrac{\boldsymbol{p}_k}{\|\boldsymbol{p}_k\|}. \\ \lambda_k = \dfrac{\langle \boldsymbol{A}\boldsymbol{\phi}_k, \boldsymbol{\phi}_k\rangle}{\langle \boldsymbol{\phi}_k, \boldsymbol{\phi}_k\rangle}. \end{cases} \tag{16}$$

$$\frac{\mathcal{L}\Phi_k}{\|\mathcal{L}\Phi_k\|} = \Phi_{k-1}. \tag{17}$$

Here, the neural network $\mathcal{N}^\theta$ represents the approximated eigenfunction Φ_k at the kth iteration of Eq. (17). Given that Φ_{k-1}, which is obtained from the last iterative step and follows the main idea of inverse power method, we must solve Φ_k using $\mathcal{P}_k = \mathcal{L}^{-1}\Phi_{k-1}$ and $\Phi_k = \mathcal{P}_k/\|\mathcal{P}_k\|$. However, it is difficult to obtain the inverse operator $\mathcal{L}^{-1}$ for the differential operator $\mathcal{L}$. Therefore, Φ_k is obtained without knowing the inverse operator. The term $\mathcal{L}\Phi_k$ is computed using AD in Eq. (17). The main idea is that it is not necessary to calculate $\mathcal{L}^{-1}$, and the eigenfunction can be approximated iteratively by minimizing the defined loss (18) to approach Eq. (17), where $\boldsymbol{x}_i \in S$ is the dataset, and N denotes the number of points in S. The eigenvalue in the kth iteration is obtained by using (19).

$$Loss_{\text{ipmnn}}(\theta) = \sum_{i=1}^{N}\left(\frac{\mathcal{L}\Phi_k(\boldsymbol{x}_i)}{\|\mathcal{L}\Phi_k\|} - \Phi_{k-1}(\boldsymbol{x}_i)\right)^2. \tag{18}$$

$$\lambda_k = \frac{\langle \mathcal{L}\Phi_k, \Phi_k\rangle}{\langle \Phi_k, \Phi_k\rangle}. \tag{19}$$

3.3 Generalized inverse power method neural network

In standard nuclear engineering procedures, we normally employ the inverse power method to solve for the smallest eigenvalue and associated eigenvector, which relies on the discretization of Eq. (3). IPMNN [49] was proposed to solve the smallest eigenvalue problem, which is a mesh-free method realized by a neural network. However, the method is restricted to solving the equation $\mathcal{L}\phi = \lambda\phi$, which is a simple form. Therefore, we propose GIPMNN to solve Eq. (3). Details of algorithm of GIPMNN is presented in Algorithm 1.

First, the generalized inverse power method is used to solve Eq. (20). The key step to focus on is given in Eq. (21), where $\boldsymbol{A}$ and $\boldsymbol{B}$ are two matrices and λ_{k-1} and $\boldsymbol{\phi}_{k-1}$ are the results of the previous iteration. Therefore, λ_k and $\boldsymbol{\phi}_k$ are obtained by using Eq. (21).

$$\boldsymbol{A}\boldsymbol{\phi} = \lambda\boldsymbol{B}\boldsymbol{\phi}. \tag{20}$$

$$\begin{cases} \boldsymbol{A}\boldsymbol{\phi}_k = \lambda_{k-1}\boldsymbol{B}\boldsymbol{\phi}_{k-1}, \\ \lambda_k = \dfrac{\langle \boldsymbol{A}\boldsymbol{\phi}_k, \boldsymbol{\phi}_k\rangle}{\langle \boldsymbol{B}\boldsymbol{\phi}_k, \boldsymbol{\phi}_k\rangle}. \end{cases} \tag{21}$$

We use a neural network $\mathcal{N}^\theta$ to represent the approximated eigenfunction Φ.

$$\begin{cases} \mathcal{L}\Phi_k = \lambda_{k-1}\mathcal{Q}\Phi_{k-1}, \\ \lambda_k = \dfrac{\langle \mathcal{L}\Phi_k, \Phi_k\rangle}{\langle \mathcal{Q}\Phi_k, \Phi_k\rangle}. \end{cases} \tag{22}$$

In GIPMNN, Eq. (22) is an analog of Eq. (21), where $\mathcal{L}$ and $\mathcal{Q}$ denote linear differential operators implemented by AD as opposed to specially discretized matrices. In a manner similar to the generalized inverse power method, we record the results λ_{k-1} of previous iteration. In contrast to the generalized inverse power method, instead of recording $\boldsymbol{\phi}_{k-1}$, we record $\mathcal{Q}\Phi_{k-1}$. It should be noted that Φ_{k-1} denotes the eigenfunction represented by the neural network in $(k-1)$th iteration, and $\mathcal{Q}\Phi_{k-1}$ is realized by AD. In k-th iteration, we directly compute Φ_k using the neural network, i.e., $\Phi_k = \mathcal{N}^\theta$, and calculate $\mathcal{L}\Phi_k$ using AD. We obtain Φ_k directly through the neural network instead of solving the equation $\mathcal{L}\Phi_k = \lambda_{k-1}\mathcal{Q}\Phi_{k-1}$, We define the loss function $Loss_{\text{gipmnn}}$ in Eq. (23) to propel the neural network to learn Φ_k. When the neural network converges, we obtain the smallest eigenvalue, and the associated eigenfunction can be expressed using a neural network.

$$Loss_{\text{gipmnn}} = \sum_{i=1}^{N}|\mathcal{L}\Phi_k(\boldsymbol{x}_i) - \lambda_{k-1}\mathcal{Q}\Phi_{k-1}(\boldsymbol{x}_i)|^2. \tag{23}$$

Remark 3 In the algorithm of GIPMNN, given that the initial function Φ_0 is not represented by the neural network, it is not possible to obtain $\mathcal{Q}\Phi_0$ using the AD. Therefore, we chose an arbitrary function $\mathcal{Q}\Phi_0$.

The Neumann and Robin boundary conditions were used for the eigenvalue problem. It is difficult to enforce them by encoding the boundary conditions into a neural network, as in [49, 63, 64], where the Dirichlet and periodic boundary conditions are used.

We form the loss function $Loss_{\text{b}}$ in Eq. (8), where N_{b} denotes the number of sampling points on $\partial\Omega$, and $\boldsymbol{x}_i$ is a point in the sampling set $\{\boldsymbol{x}_i\}_{i=1}^{N_{\text{b}}}$.

Therefore, the loss function is defined in Eqs. (24) and (25) as follows: where the surface indicates that neutrons are reflected back into the domain or that the surface allows neutrons to escape from the system.

Algorithm 1: GIPMNN for Finding the Smallest Eigenvalue

```
Given $N$ denotes the number of points for training the neural network, $N_b$ denotes the number of points on the boundary $\partial\Omega$, $Length$ denotes a measure of $\partial\Omega$, $N_{epoch}$ denotes the maximum number of epochs, and $\epsilon$ denotes the stopping criterion
Step 1: Build data set $S$ for the loss of the Rayleigh quotient and data set $S_b$ for the loss of boundary.
Step 2: Initialize a neural network with random initialization of parameters.
for $k = 1, 2, \cdots, N_{epoch}$ do
    Input all points in $S$ and $S_b$ into neural network $\mathcal{N}_\theta$.
    Let $\Phi_k(\boldsymbol{x}_i) = \mathcal{N}^\theta(\boldsymbol{x}_i)$, where $\boldsymbol{x}_i \in S \bigcup S_b$;
    $Loss_{drm} = \alpha \frac{<\mathcal{L}\Phi_k,\Phi_k>}{<\mathcal{Q}\Phi_k,\Phi_k>} + \beta \frac{Length}{N_b} \sum_{i=1}^{N_b} |\mathcal{B}\Phi_k(\mathbf{x}_i) - g(\mathbf{x}_i)|^2$.
    Update parameters $\theta$ of the neural network via gradient descent.
    $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_k)$, where $\eta$ is the learning rate and $\boldsymbol{\theta}_k$ is the vector of parameters in $k$-th iteration.
    if $Loss_{drm} < \epsilon$ then
        Record the best eigenvalue and eigenfunction.
        If the stopping criterion is met, then the iteration can be stopped.
    end
end
```

$$\begin{cases} Loss_{\mathrm{b}} = \sum_{i=1}^{N_b} |-\frac{1}{2} D(\boldsymbol{x}_i) \nabla\Phi(\boldsymbol{x}_i) \cdot \boldsymbol{n}|^2 & \text{bare surface,} \quad (24) \\ \sum_{i=1}^{N_b} |-\frac{1}{2} D(\boldsymbol{x}_i) \nabla\Phi(\boldsymbol{x}_i) \cdot \boldsymbol{n} - \frac{1}{4}\Phi(\boldsymbol{x}_i)|^2 & \text{reflective surface.} \quad (25) \end{cases}$$

The total loss function (Eq. (26)) denotes the weighted sum of the objectives (Eq. (23)) and (Eq. (8)). For the process of GIPMNN, refer to Algorithm 1.

$$Loss_{\mathrm{total}} = \alpha Loss_{\mathrm{gipmnn}} + \beta Loss_{\mathrm{b}}. \tag{26}$$

Remark 4 In Eq. (26), α and β denote the weights of the two losses. It is noted that $\beta \geq \alpha$ when training the neural network, particularly in the method GIPMNN. Given this issue, it is easy for a neural network to determine the eigenvalue and eigenfunction.

3.4 Physics constrained generalized inverse power method neural network

Although GIPMNN can solve the eigenvalue problems of Eq. (3), it is still difficult to solve eigenvalue problems with discontinuous coefficients in different regions. We discuss interface problems, which implies that the eigenfunction may be continuous at the interface, and the derivatives of the eigenfunction may not be continuous at the interface. The enforcement of interface conditions is very important for GIPMNN.

In the study, inspired by the idea of a piecewise deep neural network [58], we propose a PC-GIPMNN to solve eigenvalue problems with interface conditions. However, instead of employing different neural networks in different subdomains, we use only one neural network and multiple neurons in the output layer, as shown in Fig. 2. It should be noted that each neuron in the output layer corresponds to a subdomain. Therefore, we can obtain outputs in different subdomains that can be used to enforce the conditions at the interface.

Suppose that there are two domains Ω_l and Ω_r, with an interface Γ, which is the cross region between the two domains. Given the properties of the neutron population $\phi(\boldsymbol{x})$, we can summarize that the eigenfunction will satisfy two interface conditions, i.e., (27) and Eq. (28), where ϕ_l and ϕ_r represent the eigenfunctions defined in Ω_l and Ω_rm, respectively, and $\boldsymbol{n}$ denotes the normal vector pointing from Ω_r to Ω_l. D_l and D_r are the coefficients defined in Ω_l and Ω_r, respectively, which are discontinuous at the interface. Equation (27) indicates that the eigenfunction is continuous at the interface, i.e., the neutron flux is continuous at the interface. Equation (28) indicates that neutron flow is continuous at the interface.

$$\begin{cases} \phi_l = \phi_r, & (27) \\ -D_l \nabla\phi_l \cdot \boldsymbol{n} = -D_r \nabla\phi_r \cdot \boldsymbol{n}. & (28) \end{cases}$$

Assume that S_Γ corresponds to the set of points at the interface Γ and $|S_\Gamma|$ denotes the number of points in S_Γ. We then introduce two penalty terms to enforce the two interface conditions: Equation (29) and Eq. (30), where $\boldsymbol{x_i} \in S_\Gamma$.

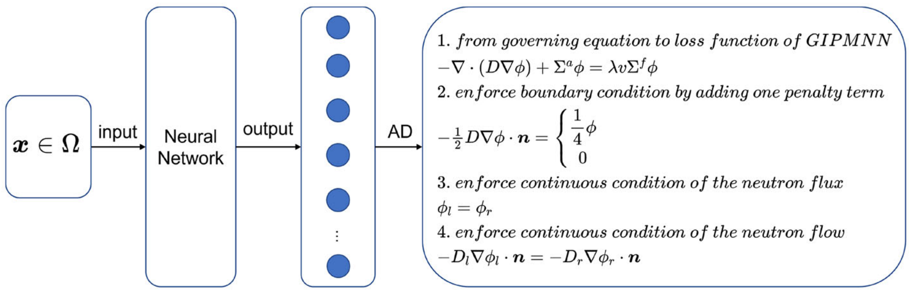

$\boldsymbol{x} \in \Omega$
input
Neural Network
output
AD
1. from governing equation to loss function of GIPMNN
$-\nabla \cdot (D\nabla\phi) + \Sigma^a\phi = \lambda v \Sigma^f \phi$
2. enforce boundary condition by adding one penalty term
$-\frac{1}{2} D\nabla\phi \cdot \boldsymbol{n} = \begin{cases} \frac{1}{4}\phi \\ 0 \end{cases}$
3. enforce continuous condition of the neutron flux
$\phi_l = \phi_r$
4. enforce continuous condition of the neutron flow
$-D_l\nabla\phi_l \cdot \boldsymbol{n} = -D_r\nabla\phi_r \cdot \boldsymbol{n}$

Fig. 2 Illustration of PC-GIPMNN architecture diagram. There are multiple neurons in the output layer, which denote the eigenfunctions in different subdomains

$$Loss_{i1} = \sum_{i=1}^{|S_\Gamma|} |\Phi_l(\boldsymbol{x_i}) - \Phi_r(\boldsymbol{x_i})|^2, \tag{29}$$

$$Loss_{i2} = \sum_{i=1}^{|S_\Gamma|} |(-D_l \nabla \Phi_l(\boldsymbol{x_i}) \cdot \boldsymbol{n}) - (-D_r \nabla \Phi_r(\boldsymbol{x_i}) \cdot \boldsymbol{n})|^2. \tag{30}$$

By combining (26), (29), and (30), Equation (31) is the total loss defined in our method PC-GIPMNN, where α, β, γ, and δ are the weights of the different losses. In subsequent experiments, we chose 1. In the study, we focused on the proposed algorithms and neglected the influence of weights. We are expecting that our proposed algorithms are universal and achieve better results, even without adjusting the weights. Therefore, we select all weights as 1.

$$Loss_{\text{total}} = \alpha Loss_{\text{gipmnn}} + \beta Loss_{\text{b}} + \gamma Loss_{i1} + \delta Loss_{i2}. \tag{31}$$

Moreover, the eigenfunction can be represented as:

$$\phi = l_l \phi_l + l_r \phi_r, \tag{32}$$

where, l_l and l_r denote the indicator functions, $l_l = 1$ in Ω_l, $l_l = 0$ in Ω_r, $l_r = 1$ in Ω_r and $l_r = 0$ in Ω_l.

Remark 5 Conservative PINN (cPINN) [53] developed PINN for each subdomain and considered additional conservation law along the subdomains' interfaces (a general consideration in reactor physics [11]). However, in neural network training, eigenvalue is involved as a variable to be optimized, and the numerical examples that are presented correspond to only one-dimensional cases. Furthermore, the relative errors of k_{eff} in these cases are 4.4800×10^{-04} and 3.3500×10^{-04}. Similarly, we use Eqs. (29) and (30) to enforce the interface conditions. As shown below, our methods are more generic and yield better results.

Remark 6 In [45], the impact of the interface conditions was ignored, and the relative errors of k_{eff} in their cases corresponded to 1.3×10^{-03} and 4.4×10^{-03}, respectively, and the study did not involve the smallest eigenvalue problem. In this study, we obtain the lowest eigenvalue using the inverse power method as opposed to using a free learnable parameter to approximate the eigenvalue. Our numerical results demonstrate that accurate results can be obtained in more complicated cases.

3.5 Deep Ritz method

DRM is a deep-learning-based method for numerically solving variational problems [24]. It reformulates the original PDEs into equivalent variational equations and defines the loss function based on variational formulations. The solutions of PDEs are represented by a neural network, and the derivatives are calculated using AD. DRM [24] is also used to solve eigenvalue problems. Furthermore, we specify how to use DRM to solve Eq. (3).

We consider the variational principle of the smallest eigenvalues.

$$\begin{cases} \min & \dfrac{\int_\Omega \mathcal{L}\phi \cdot \phi \mathrm{d}x}{\int_\Omega \mathcal{Q}\phi \cdot \phi \mathrm{d}x}, \\ \text{s.t.} & \mathcal{B}\phi|_{\partial\Omega} = g, \end{cases} \tag{33}$$

where Rayleigh quotient was used. The boundary conditions were enforced by adding a penalty term.

$$\min \quad \int_{\partial\Omega} |\mathcal{B}\phi - g|^2 \mathrm{d}s, \tag{34}$$

and the total loss function $Loss_{\text{drm}}$ is defined as:

$$Loss_{\text{drm}} = \alpha \frac{\int_\Omega \mathcal{L}\phi \cdot \phi \mathrm{d}x}{\int_\Omega \mathcal{Q}\phi \cdot \phi \mathrm{d}x} + \beta \int_{\partial\Omega} |\mathcal{B}\phi - g|^2 \mathrm{d}s, \tag{35}$$

where α and β denote the weights of different losses. We chose $\alpha = 1$ and $\beta = 1$ for our experiments.

After the optimal approximation is obtained by solving the optimization problem (35), we obtain the smallest eigenvalue $\lambda = \frac{\int_\Omega \mathcal{L}\phi \cdot \phi \mathrm{d}x}{\int_\Omega \mathcal{Q}\phi \cdot \phi \mathrm{d}x}$ and eigenfunction represented by the trained neural network. It should be noted that $\mathcal{L}\phi$, $\mathcal{Q}\phi$, and $\mathcal{B}\phi$ are computed using the AD. For the process of DRM, refer to Algorithm 2.

Remark 7 We enforced the boundary condition by adding a penalty term, (34). However, if the boundary condition is the Neumann or Robin boundary condition, we do not use the penalty term (34) because the boundary condition is incorporated into the Rayleigh quotient based on Green's first identity [65].

4 Experiments

In this section, we present numerical experiments to compare the applicability and accuracy of GIPMNN, PC-GIPMNN, and DRM for solving the smallest eigenvalue problems in reactor physics. In all the experiments below, we chose the Adam optimizer with an initial learning rate 10^{-3} to minimize the loss function. Furthermore, we trained the neural network with the architecture of ResNet on a server equipped with CentOS 7 system, one Intel Xeon Platinum 8358 2.60-GHz CPU, and one NVIDIA A100 80GB GPU. Moreover, unless otherwise specified, the activation function was selected as the tanh function.

Algorithm 2: DRM for Finding the Smallest Eigenvalue

```
Given $N$ denotes the number of points for training the neural network, $N_b$ denotes the number of points on the boundary $\partial\Omega$, $Length$ denotes a measure of $\partial\Omega$, $N_{epoch}$ denotes the maximum number of epochs, and $\epsilon$ denotes the stopping criterion
Step 1: Build data set $S$ for the loss of the Rayleigh quotient and data set $S_b$ for the loss of boundary.
Step 2: Initialize a neural network with random initialization of parameters.
for $k = 1, 2, \cdots, N_{epoch}$ do
    Input all points in $S$ and $S_b$ into neural network $\mathcal{N}_\theta$.
    Let $\Phi_k(\boldsymbol{x}_i) = \mathcal{N}^\theta(\boldsymbol{x}_i)$, where $\boldsymbol{x}_i \in S \bigcup S_b$;
    $Loss_{drm} = \alpha \frac{<\mathcal{L}\Phi_k, \Phi_k>}{<\mathcal{Q}\Phi_k, \Phi_k>} + \beta \frac{Length}{N_b} \sum_{i=1}^{N_b} |\mathcal{B}\Phi_k(\mathbf{x}_i) - g(\mathbf{x}_i)|^2$.
    Update parameters $\theta$ of the neural network via gradient descent.
    $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_k)$, where $\eta$ is the learning rate and $\boldsymbol{\theta}_k$ is the vector of parameters in $k$-th iteration.
    if $Loss_{drm} < \epsilon$ then
        Record the best eigenvalue and eigenfunction.
        If the stopping criterion is met, then the iteration can be stopped.
    end
end
```

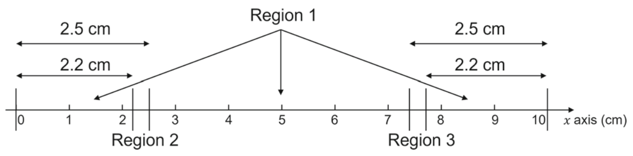


Fig. 3 Macroscopic cross-sections for the 1D slab reactor, which is modified based on the figure reported in [66]. There are three regions with different materials and functions

4.1 One-dimensional slab reactor

We consider one-dimensional case with a simple slab reactor consisting of a domain bounded by vacuum or bare surfaces, as shown in Fig. 3. The length of each slab reactor was 10 cm. It consists of fuel and control rod regions labeled 1, 2, and 3. The control rods were located between 2.2 and 2.5 cm and between 7.5 and 7.8 cm on the x-axis.

As shown in Fig. 3, there were two control rods in the one-dimensional slab reactor. Either device can be withdrawn or inserted. Three scenarios were designed to model the reactor and completely simulate the actions of the control rods. The three situations are labeled F1, F2, and F3 in Table 1. They indicate that both rods are withdrawn, only the left rod is inserted, and only the right rod is inserted. We also considered three other problems, R1, R2, and R3 in Table 1. Problem R1 is the same as F1, which is designed to simulate the withdrawal of both control rods. Here, we use R1 to facilitate comparison with R2 and R3. Although problems R2 and R3 denote that all the control rods are inserted, problem R2 resembles heavily inserted control rods, and problem R3 resembles slightly inserted control rods.

Table 1 Six sets of material cross-sections in the work [66] are used to test GIPMNN and DRM

Problem	Region	$\Sigma^a(\text{cm}^{-1})$	$\Sigma^s(\text{cm}^{-1})$	$\upsilon\Sigma^f(\text{cm}^{-1})$
F1	Region1	0.4	2.0	0.5
	Region2	0.4	2.0	0.5
	Region3	0.4	2.0	0.5
F2	Region1	0.4	2.0	0.5
	Region2	0.6	2.0	0.3
	Region3	0.4	2.0	0.5
F3	Region1	0.4	2.0	0.5
	Region2	0.4	2.0	0.5
	Region3	0.6	2.0	0.3
R1	Region1	0.4	2.0	0.5
	Region2	0.4	2.0	0.5
	Region3	0.4	2.0	0.5
R2	Region1	0.4	2.0	0.5
	Region2	0.6	2.0	0.3
	Region3	0.6	2.0	0.3
R3	Region1	0.4	2.0	0.7
	Region2	0.5	2.0	0.4
	Region3	0.5	2.0	0.4

To generate the necessary data to validate the accuracy of GIPMNN, PC-GIPMNN, and DRM, FreeFEM [67–73] is utilized to solve these problems in Table 1. FreeFEM is a partial differential equation solver for non-linear multi-physics systems using the finite element method. We choose number of cells in the x-direction $N = 1001$ and mesh size $\Delta x = 10^{-3}$. The solution of the baseline is obtained by FEM, and uniform cells are used to train GIPMNN, PC-GIPMNN, and DRM.

Remark 8 When solving all the parameter-dependent problems below, parameters Σ^a, Σ^s, and $\upsilon\Sigma^f$ are selected based on whether the data point $\boldsymbol{x}$ belongs to a related region. For PC-GIPMNN, the data points on the interface are considered to belong to multiple regions simultaneously. Therefore, we can enforce the interface conditions using data points on the interface.

Remark 9 As unsupervised algorithms, our methods use only points to train a neural network without any prior knowledge. Therefore, there was no test set for the proposed algorithm.

4.1.1 Using GIPMNN to solve for Eigenvalue

In this one-dimensional example, we select 20 neurons for each hidden layer in ResNet and $N_{\text{epoch}} = 50000$. Without a loss of generality, the weights α and β of the different losses are not adjusted to achieve better results. Therefore, α and β were set to one.

In Eq. (21), we must solve for $\boldsymbol{\phi}_k$ using the eigenvalues λ_{k-1} and $\boldsymbol{\phi}_{k-1}$. To accelerate the training process and obtain more accurate results using Algorithm 1, we split the iterations in the original Algorithm 1 into inner and outer iterations, which can be observed in Algorithm 3, where N_{inner} and N_{outer} denote the number of inner and outer iterations, respectively.

We chose the ratios of the outer and inner iterations as $1:1$, $1:10$, $1:100$, $1:1000$, and $1:10000$ to investigate the effect of the ratio on the results. Ratio $1:n$ implies that the outer code is executed once, whereas the inner code is executed n times. For comparison, the total number of iterations was fixed at $N_{\text{total}} = 100000$ and $N_{\text{total}} = N_{\text{inner}} \times N_{\text{outer}}$. The relative errors of k_{eff} and eigenfunction for different ratios of outer and inner iterations during the training process are shown in Fig. 4. The results of $k_{\text{eff}}^{\text{rel}}$ are shown in the first row and those of ϕ^{rel} are shown in the second row, where the relative errors of k_{eff} and eigenfunction are calculated by

$$k_{\text{eff}}^{\text{rel}} = \frac{|k_{\text{eff}}(FEM) - k_{\text{eff}}(NN)|}{k_{\text{eff}}(FEM)}, \tag{36}$$

$$\phi^{\text{rel}} = \frac{\max\limits_{\boldsymbol{x}}(|\phi(FEM) - \phi(NN)|)}{\max\limits_{\boldsymbol{x}}(|\phi(FEM)|)}, \tag{37}$$

respectively. As shown in the figures, the best ratio is $1:1$ because the relative errors of k_{eff} and eigenfunction of the ratio $1:1$ is relatively smaller than the others when training the neural network. The convergence worsens when the ratio of the outer and inner iterations changes from $1:1$ to $1:10000$. Therefore, we trained GIPMNN using a ratio $1:1$ to solve 1D and 2D problems.

Moreover, we fixed the number of outer iterations $N_{\text{outer}} = 1000$ and retrained the neural network using the ratios. The results are shown in Fig. 5. The opposite results are observed when compared with the results in Fig. 4. The ratio $1:1$ is the worst ratio because increases in inner iterations lead to a better approximation of the eigenfunction in the next outer iteration. This is consistent with the results of the inverse power method.

Algorithm 3: Iterations in Algorithm 1 are split into inner iterations and outer iterations.

```
for $k = 1, 2, \cdots, N_{outer}$ do
    for $j = 1, 2, \cdots, N_{inner}$ do
        Input all points in $S$ and $S_b$ into neural network $\mathcal{N}^\theta$.
        Let $\Phi_k(\boldsymbol{x}_i) = \mathcal{N}^\theta(\boldsymbol{x}_i)$, where $\boldsymbol{x}_i \in S \bigcup S_b$;
        $Loss_{gipmnn} = \sum_{i=1}^{N} |\mathcal{L}\Phi_k(\boldsymbol{x}_i) - \lambda_{k-1}\mathcal{Q}\Phi_{k-1}(\boldsymbol{x}_i)|^2$.
        $Loss_b = \sum_{i=1}^{N_b} |\mathcal{B}\Phi_k(\boldsymbol{x}_i) - g(\boldsymbol{x}_i)|^2$,
        $Loss_{total} = \alpha Loss_{gipmnn} + \beta Loss_b$,
        if $j = N_{inner}$ then
            $\Phi_{k-1} = \Phi_k / \|\Phi_k\|$.
            $\lambda_{k-1} = \frac{<(\mathcal{L}\Phi)_k, \Phi_k>}{<(\mathcal{Q}\Phi)_k, \Phi_k>}$,
        end
        Update parameters $\theta$ of the neural network via gradient descent.
        $\boldsymbol{\theta}_{k,j+1} = \boldsymbol{\theta}_{k,j} - \eta \nabla_{\boldsymbol{\theta}} Loss_{total}(\boldsymbol{\theta}_{k,j})$, where $\eta$ denotes the learning rate and $\boldsymbol{\theta}_{k,j}$ denotes the vector of parameters in $(kj)$-th iteration.
    end
    if $Loss_{total} < \epsilon$ then
        Record the best eigenvalue and eigenfunction.
        If the stopping criterion is met, then the iteration can be stopped.
    end
end
```

4.1.2 Using PC-GIPMNN to solve for Eigenvalue

We divided the slab reactor into five parts from left to right. The output layer included five neurons, and five functions were defined in different subdomains. Here, u, w, and q denote the functions defined in Region 1, v and p are those defined in Regions 2 and 3, respectively.

As shown in Fig. 3, the four points are denoted as x_{i1}=2.2, x_{i2}=2.5, $x_{i3} = 7.5$, and $x_{i4} = 7.8$. The interface conditions (29) and (30) are implemented as loss functions defined in (38) and (39) in the 1D example.

$$\begin{aligned} Loss_{i1} &= |u(x_{i1}) - v(x_{i1})|^2 + |v(x_{i2}) - w(x_{i2})|^2 \\ &\quad + |w(x_{i3}) - p(x_{i3})|^2 + |p(x_{i4}) - q(x_{i4})|^2, \end{aligned} \tag{38}$$

$$\begin{aligned} Loss_{i2} &= |(-D_1 u_x(x_{i1})) - (-D_2 v_x(x_{i1}))|^2 \\ &\quad + |(-D_2 v_x(x_{i2})) - (-D_1 w_x(x_{i2}))|^2 \\ &\quad + |(-D_1 w_x(x_{i3})) - (-D_3 p_x(x_{i3}))|^2 \\ &\quad + |(-D_3 p_x(x_{i4})) - (-D_1 q_x(x_{i4}))|^2. \end{aligned} \tag{39}$$

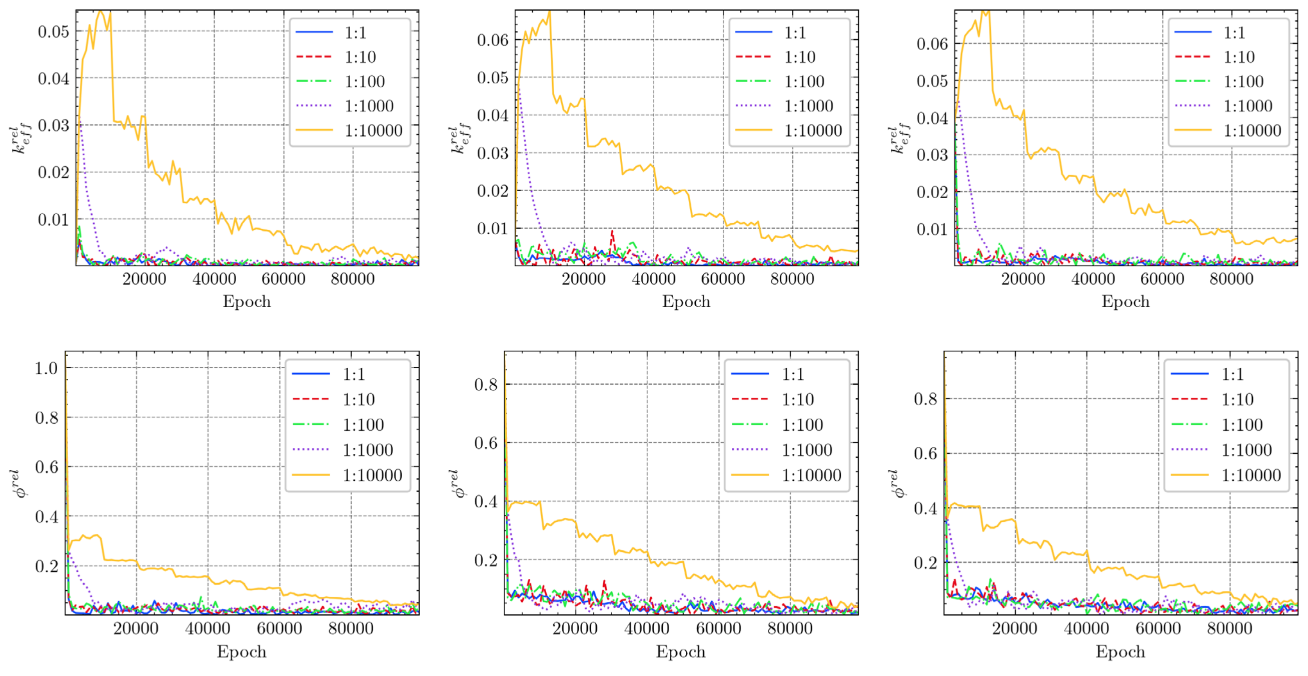
The eigenfunction can be expressed as follows:

$$\phi = ul_1 + vl_2 + wl_3 + pl_4 + ql_5, \tag{40}$$

where l_1, l_2, l_3, l_4, and l_5 are indicator functions that are 1 or 0 based on the input $\boldsymbol{x}$ inside or outside the subdomain.

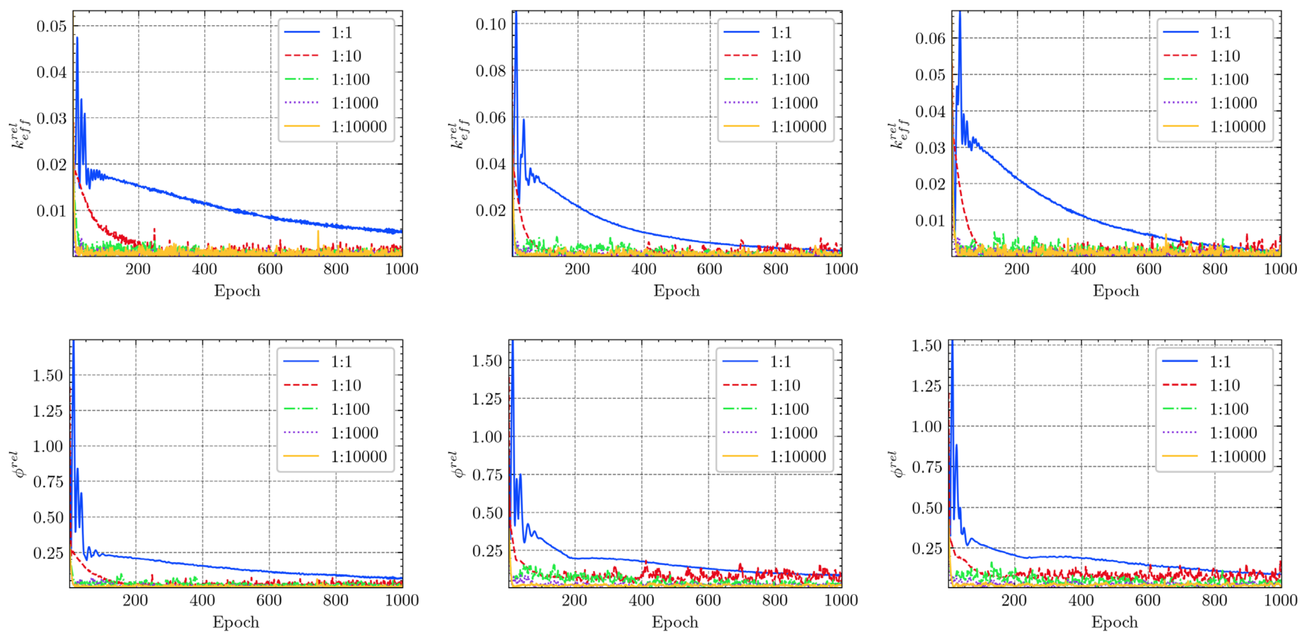
4.1.3 Using DRM to solve for Eigenvalue

The configuration of DRM is identical to that of GIPMNN. Given that it is a variational formulation, and the boundary condition is incorporated into the Rayleigh quotient, we do not use the penalty term to enforce the boundary condition. The loss function $Loss_{\text{drm}}$ is defined as:



(a) F1 (b) F2 (c) F2

Fig. 4 (Color online) Relative error of k_{eff} and ϕ for problems F1, F2, and F3 (from (a) to (c)) with respect to different ratios of outer and inner iterations during training process. The first row shows the results of $k_{\text{eff}}^{\text{rel}}$ and the second row shows the results of ϕ^{rel}. The neural network is trained with $N_{\text{total}} = 100000$



(a) F1 (b) F2 (c) F3

Fig. 5 (Color online) Relative error of k_{eff} and ϕ for problems F1, F2, and F3 (from (a) to (c)) with respect to different ratios of outer and inner iterations during training process. The first row shows the results of $k_{\text{eff}}^{\text{rel}}$ and second row shows the results of ϕ^{rel}. The neural network is trained with $N_{\text{outer}} = 1000$

$$Loss_{\mathrm{drm}} = \frac{\frac{Length_{\mathrm{slab}}}{N}\sum_{i=1}^{N} D|\nabla\phi(x_i)|^2 + \frac{1}{2}(\phi(x_l)^2 + \phi(x_r)^2) + \frac{Length_{\mathrm{slab}}}{N}\sum_{i=1}^{N}\Sigma^{\mathrm{a}}\phi(x_i)^2}{\frac{Length_{\mathrm{slab}}}{N}\sum_{i=1}^{N}\upsilon\Sigma^{\mathrm{f}}\phi(x_i)^2}, \tag{41}$$

where $Length_{\mathrm{slab}}$ denotes the length of the slab reactor, x_l and x_r denote points that show the positions of the endpoints of the slab reactor, and N denotes the number of points used to approximate the integral. Therefore, the smallest eigenvalue λ can be obtained after the neural network converges.

4.1.4 Results

The results for the one-dimensional example are listed in Tables 2, 3, and 6. The values of k_{eff} and their relative errors are listed in Table 2. For problems F1 and R1, which have continuous coefficients, the results obtained by GIPMNN are better than those obtained by PC-GIPMNN and DRM. For problem F2, the relative error of k_{eff} obtained by DRM is better than those obtained by other methods. For other problems with discontinuous coefficients, the results obtained using the PC-GIPMNN are better than those obtained by GIPMNN and DRM. The relative error of k_{eff} computed by PC-GIPMNN is approximately 10^{-5}, which is lower than 10^{-4} as computed by DRM. The relative errors in the eigenfunctions are listed in Table 3. For all problems, PC-GIPMNN can attain better results than GIPMNN and DRM.

In Fig. 6, the eigenfunctions of GIPMNN, PC-GIPMNN, and DRM are compared with those of FEM. Specifically, we follow the conventional normalization process in nuclear reactor physics [1], where the normalization constant is generally computed to make the average reactor power equal to unity; thus, the eigenfunctions are normalized by Eq. (42), where N denotes the number of training points in the entire domain.

$$\phi_{\mathrm{norm}} = \frac{N}{\sum_{i=1}^{N}\phi(x_i)}\phi. \tag{42}$$

The figures in the first row show the results for problems F1, F2, and F3 and those in the second row show the results for problems R1, R2, and R3. In each figure, we plot the eigenfunction obtained by FEM and compare the relative errors of the eigenfunctions computed by GIPMNN, PC-GIPMNN, and DRM. Evidently, the results obtained by PC-GIMPNN are better than those obtained by the other methods, which is consistent with the results in Table 3.

As reported in a previous study, [49], IPMNN can attain more accurate eigenvalues when compared to those obtained with DRM when linear eigenvalue problems without interface are considered. Therefore, IPMNN and GIPMNN are suitable to determine the eigenfunction of a problem with a strong form, and DRM is similar to FEM in that DRM is applicable for finding the eigenfunction of a problem with a weak form. Consequently, DRM is better than GIPMNN

Table 2 Results obtained by GIPMNN, PC-GIPMNN, and DRM compared with those obtained by FEM for problems in Table 1. $k_{\mathrm{eff}}^{\mathrm{rel}}$ denotes the relative error of k_{eff}

Problem	$k_{\mathrm{eff(FEM)}}$	$k_{\mathrm{eff(GIPMNN)}}$	$k_{\mathrm{eff(PC\text{-}GIPMNN)}}$	$k_{\mathrm{eff(DRM)}}$	$k^{\mathrm{rel}}_{\mathrm{eff(GIPMNN)}}$	$k^{\mathrm{rel}}_{\mathrm{eff(PC\text{-}GIPMNN)}}$	$k^{\mathrm{rel}}_{\mathrm{eff(DRM)}}$
F1	1.2127	**1.2127**	**1.2127**	1.2128	2.5142×10^{-06}	3.7743×10^{-06}	5.8568×10^{-05}
F2	1.1973	1.1970	**1.1973**	1.1972	2.3429×10^{-04}	2.6246×10^{-04}	1.1679×10^{-04}
F3	1.1973	1.1972	**1.1973**	1.1971	5.5503×10^{-05}	1.6898×10^{-05}	1.6809×10^{-04}
R1	1.2127	**1.2127**	**1.2127**	1.2128	2.5142×10^{-06}	3.7743×10^{-06}	5.8568×10^{-05}
R2	1.1715	1.1697	**1.1715**	1.1707	1.5629×10^{-03}	3.2020×10^{-05}	7.2244×10^{-04}
R3	1.6498	1.6484	**1.6498**	1.6487	8.5897×10^{-04}	2.9838×10^{-05}	6.7968×10^{-04}

The bold numbers are the best results

Table 3 Results obtained by GIPMNN, PC-GIPMNN, and DRM compared with those obtained by FEM for problems in Table 1. ϕ^{rel} denotes the relative error of eigenfunction

Problem	$\phi^{\mathrm{rel}}_{\mathrm{(GIPMNN)}}$	$\phi^{\mathrm{rel}}_{\mathrm{(PC\text{-}GIPMNN)}}$	$\phi^{\mathrm{rel}}_{\mathrm{(DRM)}}$
F1	2.7301×10^{-03}	6.3620×10^{-04}	2.6313×10^{-03}
F2	2.6279×10^{-02}	1.6140×10^{-03}	8.2217×10^{-03}
F3	2.7403×10^{-02}	1.0120×10^{-03}	6.2428×10^{-03}
R1	2.7301×10^{-03}	6.3620×10^{-04}	2.6313×10^{-03}
R2	2.5768×10^{-02}	7.5946×10^{-04}	6.8807×10^{-03}
R3	1.4559×10^{-02}	7.5257×10^{-04}	8.1038×10^{-03}

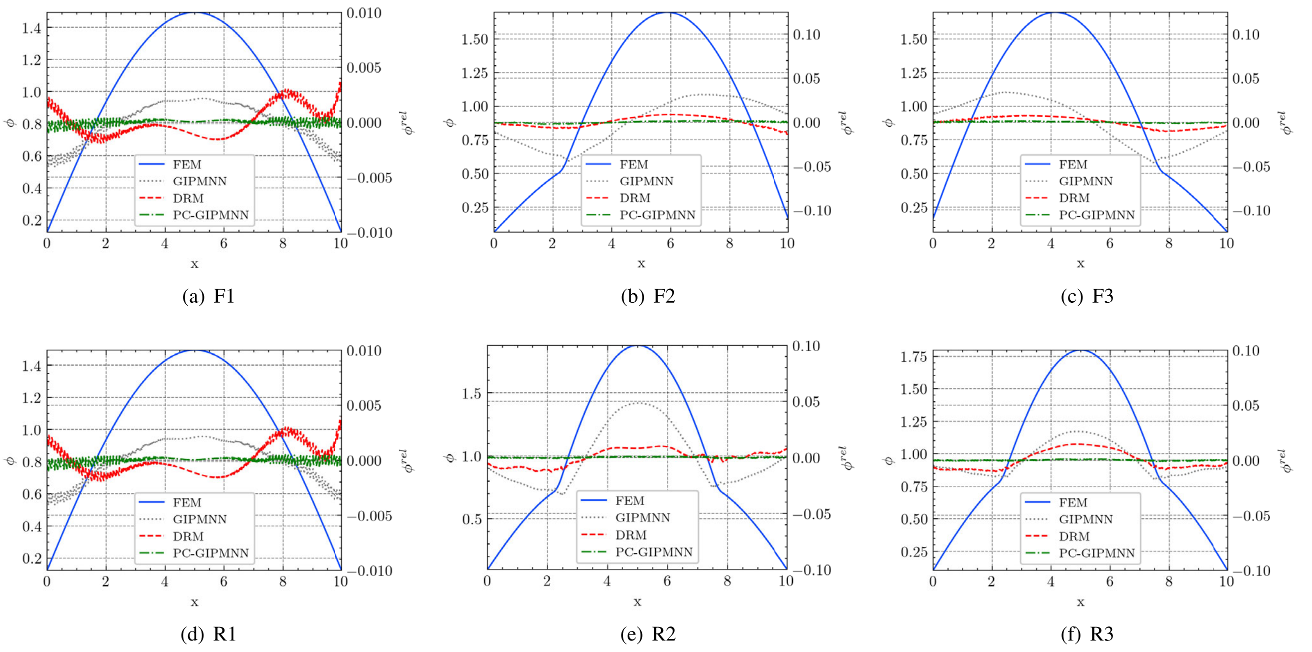


Fig. 6 (Color online) Eigenfunctions obtained by GIPMNN, PC-GIPMNN, and DRM are compared with those obtained by FEM for one-dimensional example. The eigenfunctions are normalized. The first row shows the results of problems F1, F2, and F3, and the second row shows the results of problems R1, R2, and R3

in this one-dimensional example with discontinuous coefficients. However, given that the interface conditions are well implemented in PC-GIPMNN, it successfully learns the eigenvalue and eigenfunction and achieves better results than GIPMNN and DRM, as shown in Tables 2 and 3.

Remark 10 Specifically, DRM is applicable to determine the eigenfunction of a problem with a weak form, which implies that the eigenfunction exhibits low regularity. Subsequently, as shown in the implementation of GIPMNN, it is necessary for the eigenfunction obtained from GIPMNN to exhibit high regularity. Therefore, the learned eigenfunction is in a strong form. Hence, GIPMNN is unable to obtain accurate values at the interface. However, PC-GIPMNN does not require the eigenfunction to exhibit a higher regularity at the interface but instead guarantees continuity and physical constraints by realizing the interface conditions. Therefore, PC-GIPMNN successfully learns the eigenvalue and eigenfunction and obtains better results when compared to GIPMNN and DRM.

4.2 Two-dimensional reactor

As shown in Fig. 7, a two-dimensional reactor is modeled in a square-shaped domain with 90-cm sides. The reactor was surrounded by a neutron reflector with graphite material, which implies that Robin boundary condition was selected. The main bulk of the reactor corresponds to the fuel region. Within its central region, four control rods can be inserted or withdrawn. When the control rods are withdrawn, materials in the regions are replaced with water, which corresponds to common practice in many reactor designs. Table 4 lists five different materials used in the mock reactor. However,

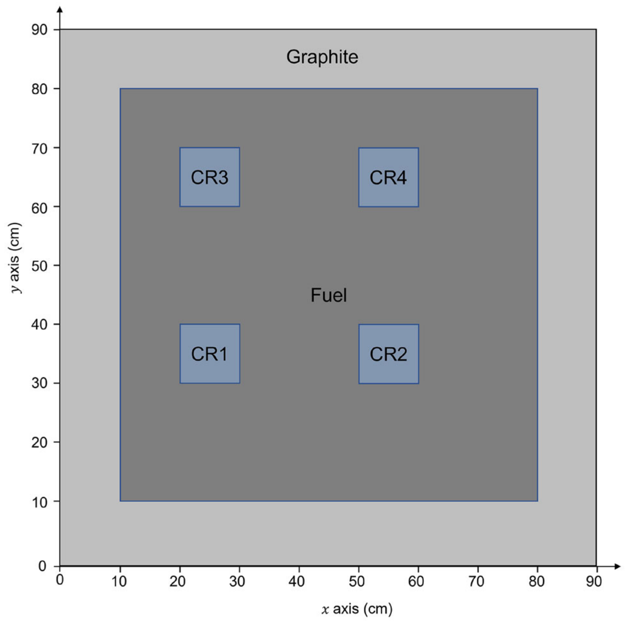


Fig. 7 Geometry of a 2D reactor core with graphite, fuel, and four control-rod regions labeled as 1, 2, 3, and 4. The figure is similar to the figure in a previous study [66]

Table 4 Cross section data for various materials of the 2D reactor. This data are similar to those reported in a previous study [66]

Material	$\Sigma^{a}(\mathrm{cm}^{-1})$	$\Sigma^{s}(\mathrm{cm}^{-1})$	$\upsilon\Sigma^{f}(\mathrm{cm}^{-1})$
Fuel 1	0.075	0.53	0.079
Fuel 2	0.072	0.53	0.085
Water	0.01	0.89	0.0
Control rod	0.38	0.2	0.0
Graphite	0.15	0.5	0.0

the two types of fuels in Table 4 are designed to simulate different fuel materials. Fuel type 1 defines the standard fuel used in most problems, and fuel type 2 defines an adjusted fuel composition that is different from fuel type 1. Fuel type 2 in problem R7 was used to test whether our methods can be affected by different types of fuels.

As shown in Table 5, there are 12 problems in validating the accuracy of the proposed method. Five full models which are labeled as F1-F5, were proposed to simulate the reactor with all control rods removed and then with only one control rod inserted in the regions of the control rods. As previously discussed, when the control rod was removed, the material in the region was replaced with water. Therefore, W is used to denote water in Table 5. Other seven reactor configurations, denoted by R1-R7, were proposed to simulate the cases where more control rods were inserted. Problems R1 and R2 are equivalent to the full model problems F1 and F2: Problems R3-R6 utilized different combinations of inserted and rejected control rods. It should be noted that in problem R7, the material configuration differs from the material configuration of other problems. The fuel type was replaced with fuel type 2, and the control rods were assumed to be partially inserted, which implies that the materials in the regions corresponded to a mix of control rods and water materials.

In the two-dimensional case, we use FreeFEM to solve the problems listed in Table 5. We chose uniform grids with $\Delta x = \frac{1}{90}$ and $\Delta y = \frac{1}{90}$. We trained GIPMNN, PC-GIPMNN, and DRM with $N_x = 91$ and $N_y = 91$.

4.2.1 Using GIPMNN and PC-GIMPNN to solve for Eigenvalue

The number of points $N = N_x N_y$ is used to calculate $Loss_{\mathrm{gipmnn}}$ and number of points $N_{\mathrm{b}} = 2(N_x - 2) + 2(N_y - 2) + 4$ is used to calculate $Loss_b$. The number of neurons is 20 for each hidden layer in ResNet and $N_{\mathrm{epoch}} = 500000$ for both GIPMNN and PC-GIPMNN. Without a loss of generality, α and β are set to one. As mentioned previously, we use the optimal ratio of the outer and inner iterations, which is $1:1$.

The 2D reactor is divided into six parts, as shown in Fig. 7. The output layer includes six neurons, and there are six functions that are defined in different subdomains and are labeled as u, v, w, r, p, and q, where u, v, w, and r denote functions defined in CR1, CR2, CR3, and CR4, and p and q denote functions defined for Fuel and Graphite, respectively. $S_{\mathrm{CR}1}$, $S_{\mathrm{CR}2}$, $S_{\mathrm{CR}3}$, $S_{\mathrm{CR}4}$, and S_{GF} denote different datasets at different interfaces.

For PC-GIPMNN, interface conditions (29) and (30) are implemented as loss functions (43) and (44) in the 2D example.

$$
\begin{aligned}
Loss_{i1} = & \sum_{i=1}^{|S_{\mathrm{CR}1}|} |u(\boldsymbol{x}_i) - p(\boldsymbol{x}_i)|^2 \\
& + \sum_{i=1}^{|S_{\mathrm{CR}2}|} |v(\boldsymbol{x}_i) - p(\boldsymbol{x}_i)|^2 + \sum_{i=1}^{|S_{\mathrm{CR}3}|} |w(\boldsymbol{x}_i) - p(\boldsymbol{x}_i)|^2 \\
& + \sum_{i=1}^{|S_{\mathrm{CR}4}|} |r(\boldsymbol{x}_i) - p(\boldsymbol{x}_i)|^2
\end{aligned}
$$

Table 5 Configurations for reactor problems with different materials. The configurations are similar to those in a previous study [66]. C-R denotes the control rod region, CR denotes a control rod material, and W denotes water

Problem	Fuel type	C-R1	C-R2	C-R3	C-R4
F1	Fuel 1	W	W	W	W
F2	Fuel 1	CR	W	W	W
F3	Fuel 1	W	CR	W	W
F4	Fuel 1	W	W	CR	W
F5	Fuel 1	W	W	W	CR
R1	Fuel 1	W	W	W	W
R2	Fuel 1	CR	W	W	W
R3	Fuel 1	W	CR	CR	W
R4	Fuel 1	W	W	CR	CR
R5	Fuel 1	CR	CR	W	CR
R6	Fuel 1	CR	CR	CR	CR
R7	Fuel 2	$\frac{1}{4}$CR+$\frac{1}{2}$W	$\frac{1}{10}$CR+$\frac{9}{10}$W	$\frac{3}{10}$CR+$\frac{7}{10}$W	$\frac{1}{5}$CR+$\frac{4}{5}$W

$$+\sum_{i=1}^{|S_{\mathrm{GF}}|}|p(\boldsymbol{x}_i)-q(\boldsymbol{x}_i)|^2, \tag{43}$$

$$\begin{aligned}&Loss_{i2}\\&=\sum_{i=1}^{|S_{\mathrm{CR1}}|}|(-D_{\mathrm{CR1}}\nabla u(\boldsymbol{x}_i)\cdot\boldsymbol{n})-(-D_{\mathrm{Fuel}}\nabla p(\boldsymbol{x}_i)\cdot\boldsymbol{n})|^2\\&+\sum_{i=1}^{|S_{\mathrm{CR2}}|}|(-D_{\mathrm{CR2}}\nabla v(\boldsymbol{x}_i)\cdot\boldsymbol{n})-(-D_{\mathrm{Fuel}}\nabla p(\boldsymbol{x}_i)\cdot\boldsymbol{n})|^2\\&+\sum_{i=1}^{|S_{\mathrm{CR3}}|}|(-D_{\mathrm{CR3}}\nabla w(\boldsymbol{x}_i)\cdot\boldsymbol{n})-(-D_{\mathrm{Fuel}}\nabla p(\boldsymbol{x}_i)\cdot\boldsymbol{n})|^2\\&+\sum_{i=1}^{|S_{\mathrm{CR4}}|}|(-D_{\mathrm{CR4}}\nabla r(\boldsymbol{x}_i)\cdot\boldsymbol{n})-(-D_{\mathrm{Fuel}}\nabla p(\boldsymbol{x}_i)\cdot\boldsymbol{n})|^2\\&+\sum_{i=1}^{|S_{\mathrm{GF}}|}|(-D_{\mathrm{Fuel}}\nabla p(\boldsymbol{x}_i)\cdot\boldsymbol{n})-(-D_{\mathrm{Graphite}}\nabla q(\boldsymbol{x}_i)\cdot\boldsymbol{n})|^2.\end{aligned} \tag{44}$$

The eigenfunction is expressed as follows:

$$\phi=ul_1+vl_2+wl_3+rl_4+pl_5+ql_6, \tag{45}$$

where l_1, l_2, l_3, l_4, l_5 and l_6 denote indicator functions.

Remark 11 In the experiment, it was important to use Eq. (42) instead of the original $L2$ norm, which is too small to optimize the neural network. Specifically, the L2 norm of the eigenfunction corresponds to one if we attempt to find a normalized eigenfunction. If the total number of points N is excessively high, then the value of each component of the eigenvector is excessively low to such an extent that it is difficult for the neural network to learn the eigenfunction.

4.2.2 Using DRM to solve for Eigenvalue and the failure of DRM during the 2D experiments

Given that the homogeneous Neumann boundary condition is used, the loss function in DRM can be defined as:

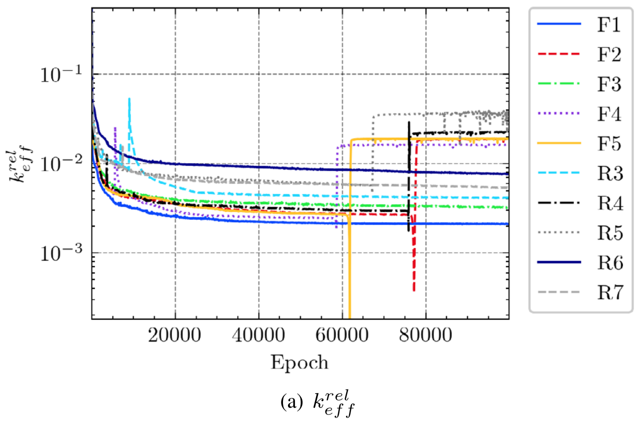
$$\begin{aligned}&Loss_{\mathrm{drm}}\\&=\frac{\frac{Area_{\mathrm{square}}}{N}\sum_{i=1}^{N}D|\nabla\phi(x_i)|^2+\frac{Area_{\mathrm{square}}}{N}\sum_{i=1}^{N}\Sigma^a\phi(x_i)^2}{\frac{Area_{\mathrm{square}}}{N}\sum_{i=1}^{N}\upsilon\Sigma^f\phi(x_i)^2},\end{aligned} \tag{46}$$

where $Area_{\mathrm{square}}$ denotes the area of the square domain as shown in Fig. 7.

We use the number of epochs $N_{\mathrm{epoch}}=500000$ in DRM. First, we found that the DRM can learn the eigenvalues and eigenfunctions at an early stage of the training process. Subsequently, the DRM attempts to learn a smaller eigenvalue after it is close to the true eigenvalue. Finally, the DRM successfully learns one smaller eigenvalue that may be close to the true eigenvalue and one terrible eigenfunction that is meaningless and far from the true eigenfunction. This phenomenon is illustrated in Fig. 8.

As listed in Table 6, although the neural network in DRM fails to learn the eigenfunction, the eigenvalue is close to the true value. This phenomenon may be caused by minimization of Rayleigh quotient. As mentioned in [49], the loss approaches zero, whereas the eigenvalue may not reach zero.

In Fig. 8, we observe that the failure of DRM during the 2D experiments occurs after $N_{\mathrm{epoch}}\geq 60000$. Therefore, we select $N_{\mathrm{epoch}}=50000$ to train DRM again and record the results. In a previous study [40], the stopping criteria of training process for PINN was investigated. In future studies, we will follow the technique discussed in [40] to determine the stopping criteria for DRM. In the next section, we compare



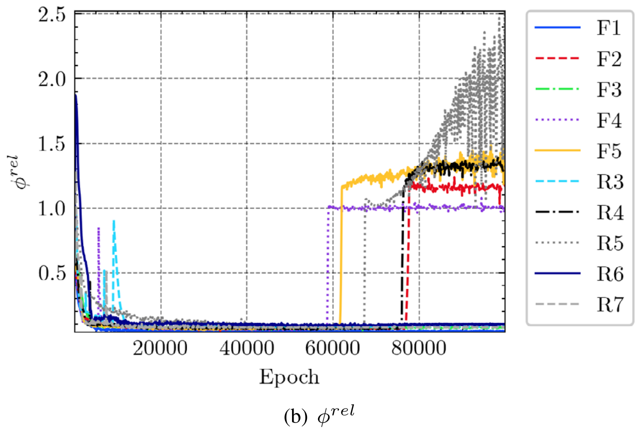


Fig. 8 (Color online) Failure of DRM during the 2D experiments. DRM fails to learn the eigenfunctions of F1-F5 and R3-R7

Table 6 Failure of DRM during the 2D experiments. The eigenfunction of F1-F5 and R1-R7 is not correct, but the eigenvalue is to the true value

Problem	$k_{\text{eff(FEM)}}$	$k_{\text{eff(DRM)}}$	$k^{\text{rel}}_{\text{eff(DRM)}}$	$\phi^{\text{rel}}{}_{\text{(DRM)}}$
F1	1.0118	1.0423	3.0156×10^{-02}	16.6580
F2	1.0052	1.0399	3.4538×10^{-02}	21.8882
F3	1.0000	1.0428	4.2782×10^{-02}	29.5368
F4	1.0079	1.0533	4.5361×10^{-02}	26.4549
F5	1.0052	1.0450	3.9598×10^{-02}	32.0516
R1	1.0118	1.0423	3.0156×10^{-02}	16.6580
R2	1.0052	1.0399	3.4538×10^{-02}	21.8882
R3	0.9921	1.0399	4.8250×10^{-02}	166.2996
R4	1.0017	1.0397	3.7884×10^{-02}	163.6593
R5	0.9780	1.0498	7.3430×10^{-02}	182.7224
R6	0.9668	1.0492	8.5210×10^{-02}	87.2200
R7	1.1018	1.1517	4.5337×10^{-02}	36.6777

the results of the DRM trained with $N_{\text{epoch}} = 50000$ with the results of GIPMNN and PC-GIPMNN.

Remark 12 The numerical results in Fig. 8 are not due to hardware problems or code errors but can be attributed to the nature of the neural network. The eigenvalue was approximated by constructing a Rayleigh quotient in the DRM. Then, the eigenvalue is treated as a loss function to optimize the neural network. However, this mechanism of minimizing the eigenvalue leads to overfitting of the neural network, as the neural network always attempts to find a point where the loss function tends toward zero.

4.2.3 Results

Similar to the results of the one-dimensional case, the relative errors of k_{eff} and eigenfunction ϕ are shown in Tables 7 and 8. It can be observed that both the relative errors for k_{eff} obtained via all the methods are small, and the results for the DRM are trained with $N_{\text{epoch}} = 50000$.

For problems F1, F2, F4, and R7, the relative error of k_{eff} obtained by DRM was smaller than that obtained by GIPMNN, except for problems F3, F5, R3, R4, R5, and R6. However, although the relative error of k_{eff} obtained by GIPMNN is small, the relative error of ϕ simulated by GIPMNN is larger than that obtained by DRM. It is observed that $k^{\text{rel}}_{\text{eff}}$ obtained by the PC-GIPMNN is smaller than that obtained by GIPMNN and DRM for all problems. Furthermore, the relative errors of ϕ computed by the PC-GIMPNN are smaller than those obtained by the GIPMNN and DRM for all problems. Therefore, the PC-GIPMNN can successfully learn eigenvalues and eigenfunctions.

In Fig. 9 and 10, the eigenfunctions computed by the FEM are shown in the first column, and the relative errors of the eigenfunctions obtained by the GIPMNN, PC-GIPMNN, and DRM are shown in the other columns for different problems. It is observed that the relative errors of the eigenfunction computed by the PC-GIPMNN and DRM are smaller than those obtained by the GIPMNN, which failed to learn some details. Compared to the eigenfunctions computed by the FEM, the results obtained by the PC-GIPMNN are the best among the three methods.

Table 7 Results obtained via GIPMNN, PC-GIPMNN, and DRM compared with FEM for problems in Table 5, $k^{\text{rel}}_{\text{eff}}$ denotes the relative error of k_{eff}. Especially, DRM is trained with $N_{\text{epoch}} = 50000$

Problem	$k_{\text{eff(FEM)}}$	$k_{\text{eff(GIPMNN)}}$	$k_{\text{eff(PC-GIPMNN)}}$	$k_{\text{eff(DRM)}}$	$k^{\text{rel}}_{\text{eff(GIPMNN)}}$	$k^{\text{rel}}_{\text{eff(PC-GIPMNN)}}$	$k^{\text{rel}}_{\text{eff(DRM)}}$
F1	1.0118	1.0094	**1.0121**	1.0096	2.3574×10^{-03}	2.8227×10^{-04}	2.1558×10^{-03}
F2	1.0052	1.0023	**1.0055**	1.0024	2.8782×10^{-03}	2.9864×10^{-04}	2.7674×10^{-03}
F3	1.0000	0.9968	**0.9999**	0.9965	3.2380×10^{-03}	1.2475×10^{-04}	3.4253×10^{-03}
F4	1.0079	1.0053	**1.0081**	1.0054	2.5913×10^{-03}	2.4834×10^{-04}	2.4439×10^{-03}
F5	1.0052	1.0025	**1.0054**	1.0024	2.7120×10^{-03}	3.1435×10^{-04}	2.7394×10^{-03}
R1	1.0118	1.0094	**1.0121**	1.0096	2.3574×10^{-03}	2.8227×10^{-04}	2.1558×10^{-03}
R2	1.0052	1.0023	**1.0055**	1.0024	2.8782×10^{-03}	2.9864×10^{-04}	2.7674×10^{-03}
R3	0.9921	0.9937	**0.9927**	0.9878	1.5981×10^{-03}	6.0649×10^{-04}	4.3297×10^{-03}
R4	1.0017	1.0014	**1.0015**	0.9987	3.5502×10^{-04}	2.4851×10^{-04}	3.0409×10^{-03}
R5	0.9780	0.9759	**0.9781**	0.9721	2.1223×10^{-03}	1.6160×10^{-04}	6.0148×10^{-03}
R6	0.9668	0.9639	**0.9680**	0.9584	3.0188×10^{-03}	1.2722×10^{-03}	8.6348×10^{-03}
R7	1.1018	1.0929	**1.1020**	1.0953	8.0309×10^{-03}	2.1827×10^{-04}	5.9042×10^{-03}

The bold numbers are the best results

Table 8 Results obtained via GIPMNN, PC-GIPMNN, and DRM when compared with FEM for problems in Table 5, ϕ^{rel} denotes the relative error of the eigenfunction. Especially, DRM is trained with $N_{\text{epoch}} = 50000$

Problem	$\phi^{\text{rel}}{}_{(\text{GIPMNN})}$	$\phi^{\text{rel}}{}_{(\text{PC-GIPMNN})}$	$\phi^{\text{rel}}{}_{(\text{DRM})}$
F1	4.6050×10^{-02}	3.0937×10^{-02}	4.4186×10^{-02}
F2	8.9731×10^{-02}	3.6713×10^{-02}	7.0763×10^{-02}
F3	8.9385×10^{-02}	4.1200×10^{-02}	6.7164×10^{-02}
F4	8.5110×10^{-02}	4.3430×10^{-02}	5.9907×10^{-02}
F5	7.9413×10^{-02}	4.6256×10^{-02}	6.6049×10^{-02}
R1	4.6050×10^{-02}	3.0937×10^{-02}	4.4186×10^{-02}
R2	8.9731×10^{-02}	3.6713×10^{-02}	7.0763×10^{-02}
R3	1.1047×10^{-01}	6.2990×10^{-02}	8.4720×10^{-02}
R4	1.3013×10^{-01}	2.9511×10^{-02}	6.8647×10^{-02}
R5	2.5852×10^{-01}	2.9718×10^{-02}	7.7923×10^{-02}
R6	4.6253×10^{-01}	5.1511×10^{-02}	9.1424×10^{-02}
R7	1.1612×10^{-01}	1.8635×10^{-02}	7.2148×10^{-02}

4.3 2D IAEA benchmark problem

We also considered the classical 2D IAEA benchmark problem reported in the study by Yang et al. [40], which was modeled using two-dimensional and two-group diffusion equations. Here, one-group neutron diffusion equation, defined in Eq. (4) is considered, and multigroup problems are considered in our future study. Its geometry is shown in Fig. 11. The main bulk of the reactor consists of two fuel regions, labeled 1 and 2, representing the two types of fuel materials. Within its central region, there are four control rods, which are all labeled as 3. The last region, labeled 4, was composed of water. Cross-sectional data for the 2D IAEA benchmark problem are presented in Table 9. It is worth noting that only one quarter of the reactor is shown in this figure because the rest can be inferred by symmetry along the x- and y-axes. Therefore, this 2D IAEA benchmark problem is confined to the two types of boundary conditions defined in Eq. (7). The problem is confined to the Neumann boundary condition on the x- and y-axes and to the Robin boundary condition on the other boundaries.

In this two-dimensional case, we used FreeFEM to solve the 2D IAEA benchmark problem with the parameters listed in Table 9. We selected uniform grids with $\Delta x = \frac{1}{170}$ and $\Delta y = \frac{1}{170}$. We trained GIPMNN, PC-GIPMNN and DRM with $N_x = 171$ and $N_y = 171$.

4.3.1 Using GIPMNN and PC-GIMPNN to solve for Eigenvalue

The number of points $N = N_x N_y$ is used to calculate $Loss_{\text{gipmnn}}$. The number of points N_{Nb} on the x- and y-axes and number of points N_{Rb} on the other boundaries were used to calculate $Loss_{\text{Nb}}$ and $Loss_{\text{Rb}}$, which enforced the Neumann and Robin boundary conditions. The number of neurons was 20 for each hidden layer in ResNet and $N_{\text{epoch}} = 500000$ for GIPMNN and PC-GIPMNN. As mentioned previously, we used the optimal ratio of the outer and inner iterations, which was 1 : 1.

The 2D reactor is divided into seven parts, as shown in Fig. 11. The output layer has seven neurons, and seven functions are defined in different subdomains and labeled as u, v, w, r, p, q, and h, where u, v, w, and r denote the functions defined in the control rods and p, q, and h are the functions defined in the fuel and water regions, respectively. S_{up}, S_{vp}, S_{wp}, S_{rp}, S_{rq}, S_{pq}, and S_{qh} denote different datasets at different interfaces.

For the PC-GIPMNN, the interface conditions (29) and (30) are implemented as the loss functions (47) and (48), respectively, in the 2D example.

$$\begin{aligned} Loss_{i1} = &\sum_{i=1}^{|S_{up}|} |u(\boldsymbol{x}_i) - p(\boldsymbol{x}_i)|^2 + \sum_{i=1}^{|S_{vp}|} |v(\boldsymbol{x}_i) - p(\boldsymbol{x}_i)|^2 \\ &+ \sum_{i=1}^{|S_{wp}|} |w(\boldsymbol{x}_i) - p(\boldsymbol{x}_i)|^2 \\ &+ \sum_{i=1}^{|S_{rp}|} |r(\boldsymbol{x}_i) - p(\boldsymbol{x}_i)|^2 + \sum_{i=1}^{|S_{rq}|} |r(\boldsymbol{x}_i) - q(\boldsymbol{x}_i)|^2 \\ &+ \sum_{i=1}^{|S_{pq}|} |p(\boldsymbol{x}_i) - q(\boldsymbol{x}_i)|^2 \\ &+ \sum_{i=1}^{|S_{qh}|} |q(\boldsymbol{x}_i) - h(\boldsymbol{x}_i)|^2, \end{aligned} \tag{47}$$

$$\begin{aligned} Loss_{i2} = &\sum_{i=1}^{|S_{up}|} |(-D_3 \nabla u(\boldsymbol{x}_i) \cdot \boldsymbol{n}) - (-D_2 \nabla p(\boldsymbol{x}_i) \cdot \boldsymbol{n})|^2 \\ &+ \sum_{i=1}^{|S_{vp}|} |(-D_3 \nabla v(\boldsymbol{x}_i) \cdot \boldsymbol{n}) - (-D_2 \nabla p(\boldsymbol{x}_i) \cdot \boldsymbol{n})|^2 \\ &+ \sum_{i=1}^{|S_{wp}|} |(-D_3 \nabla w(\boldsymbol{x}_i) \cdot \boldsymbol{n}) - (-D_2 \nabla p(\boldsymbol{x}_i) \cdot \boldsymbol{n})|^2 \end{aligned}$$

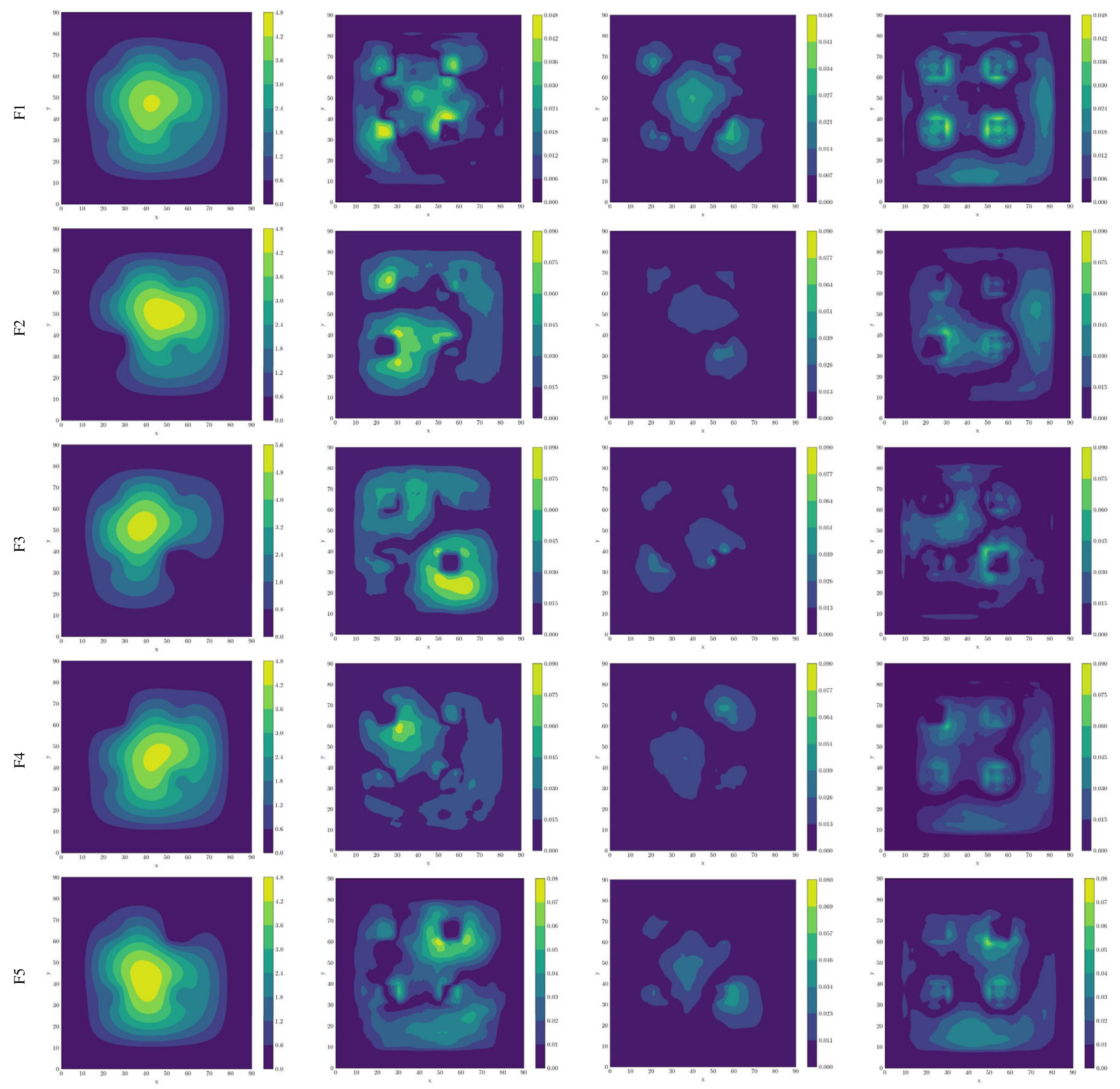


Fig. 9 (Color online) First column shows the heatmap of the eigenfunction of FEM (the first column) and the other columns show the heatmaps of the relative error of GIPMNN (the second column), PC-GIPMNN (the third column), and DRM (the fourth column) for problems F1, F2, F3, F4, and F5. Evidently, GIPMNN less favorable results than DRM. However, by enforcing the interface conditions, PC-GIPMNN outperforms GIPMNN and DRM, as shown in the third column

$$\begin{aligned}
&+\sum_{i=1}^{|S_{rp}|}|(-D_3\nabla r(\boldsymbol{x}_i)\cdot\boldsymbol{n})-(-D_2\nabla p(\boldsymbol{x}_i)\cdot\boldsymbol{n})|^2\\
&+\sum_{i=1}^{|S_{rq}|}|(-D_3\nabla r(\boldsymbol{x}_i)\cdot\boldsymbol{n})-(-D_1\nabla q(\boldsymbol{x}_i)\cdot\boldsymbol{n})|^2\\
&+\sum_{i=1}^{|S_{pq}|}|(-D_2\nabla p(\boldsymbol{x}_i)\cdot\boldsymbol{n})-(-D_1\nabla q(\boldsymbol{x}_i)\cdot\boldsymbol{n})|^2\\
&+\sum_{i=1}^{|S_{qh}|}|(-D_1\nabla q(\boldsymbol{x}_i)\cdot\boldsymbol{n})-(-D_4\nabla h(\boldsymbol{x}_i)\cdot\boldsymbol{n})|^2.
\end{aligned} \tag{48}$$

The eigenfunction can be expressed as follows:

$$\phi = ul_1 + vl_2 + wl_3 + rl_4 + pl_5 + ql_6 + hl_7, \tag{49}$$

where l_1, l_2, l_3, l_4, l_5, l_6, and l_7 are the indicator functions.

4.3.2 Using DRM to solve for Eigenvalue

Given that the homogeneous Neumann boundary condition is used, the loss function in DRM omits the impact of the Neumann boundary condition and focuses on the Robin boundary condition. The loss function is defined as follows:

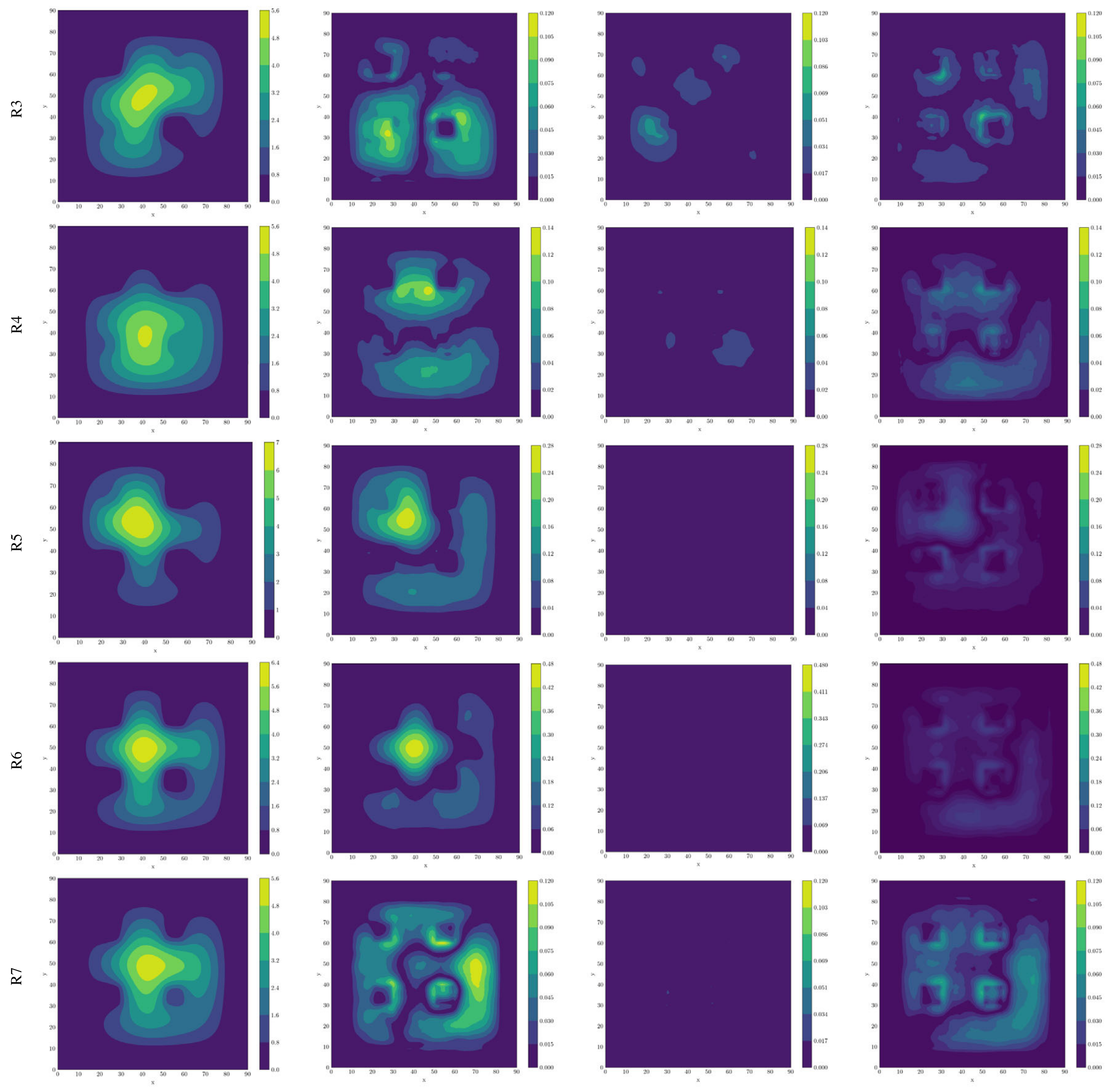


Fig. 10 (Color online) First column shows the heatmap of the eigenfunction of FEM (the first column) and the other columns show the heatmaps of the relative error of GIPMNN (the second column), PC-GIPMNN (the third column), and DRM (the fourth column) for problems R3, R4, R5, R6, and R7. Obviously, GIPMNN yields less favorable results than DRM. However, by enforcing the interface conditions, PC-GIPMNN outperforms GIPMNN and DRM, as shown in the third column

$$J = \frac{\frac{Area}{N}\sum_{i=1}^{N} D|\nabla\phi(x_i)|^2 + \frac{Length}{N_{\mathrm{Rb}}}\sum_{i=1}^{N_{\mathrm{Rb}}}\frac{1}{2}\phi(\boldsymbol{x}_i)^2 + \frac{Area}{N}\sum_{i=1}^{N}\Sigma^a\phi(x_i)^2}{\frac{Area}{N}\sum_{i=1}^{N}\upsilon\Sigma^f\phi(x_i)^2}, \tag{50}$$

where $Area$ denotes the area of all regions, $Length$ indicates the length of the boundaries other than the x- and y-axes in Fig. 11, and N_{Rb} denotes the number of points on the Robin boundary.

4.3.3 Results

As discussed above, we also trained the DRM with the number of epochs $N_{\mathrm{epoch}} = 500000$ and found that DRM failed

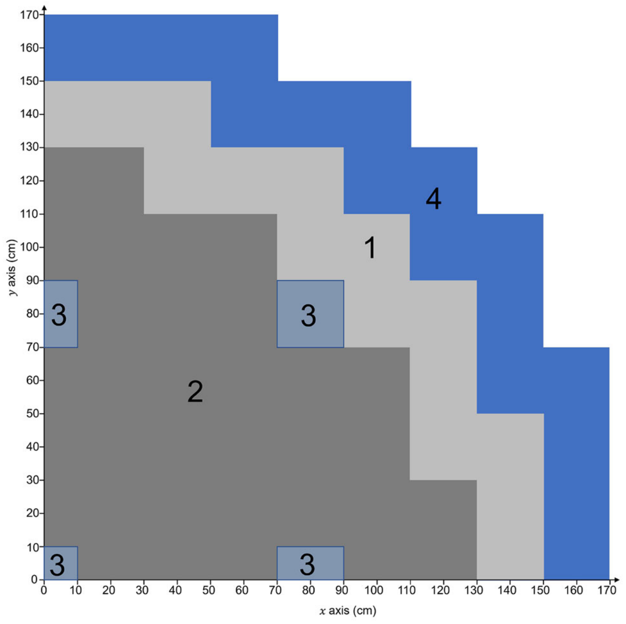


Fig. 11 (Color online) Geometry of the 2D IAEA benchmark problem with two fuel regions, four control-rod regions, and a water region labeled as 1, 2, 3 m and 4. This figure is similar to the figure reported in a previous study [40]

Table 9 Cross section data for the 2D IAEA benchmark problem

Region	$\Sigma^a(\text{cm}^{-1})$	$\Sigma^s(\text{cm}^{-1})$	$\upsilon\Sigma^f(\text{cm}^{-1})$	Material
1	2.0	0.53	0.079	Fuel 1
2	0.087	0.55	0.085	Fuel 2
3	0.38	0.20	0.0	Control rod
4	0.01	0.89	0.0	Water

to learn the eigenfunction again. In this case, DRM attains a good $k_{\text{eff}} = 0.9750$ and the relative errors of k_{eff} and ϕ are 6.4023×10^{-03} and 0.9557, respectively. Therefore, we retrained the DRM with $N_{\text{epoch}} = 50000$ and stored the best results.

The relative errors of k_{eff} and eigenfunction ϕ are shown in Table 10 and Table 11. It was observed that all three methods obtained good results, and the relative errors of k_{eff} of DRM were small. However, the ability of DRM to learn the eigenfunction was worse than that of GIPMNN and PC-GIPMNN and the relative errors of ϕ were the largest. Thus, it can be concluded that the PC-GIPMNN successfully learned the eigenvalues and eigenfunctions. The same conclusion can be drawn from the graphs in Fig. 12.

Table 11 Results obtained via GIPMNN, PC-GIPMNN, and DRM when compared with FEM for the 2D IAEA benchmark problem. ϕ^{rel} denotes the relative error of the eigenfunction. Especially, DRM is trained with $N_{\text{epoch}} = 50000$

Problem	$\phi^{\text{rel}}{}_{(\text{GIPMNN})}$	$\phi^{\text{rel}}{}_{(\text{PC-GIPMNN})}$	$\phi^{\text{rel}}{}_{(\text{DRM})}$
IAEA	8.5688×10^{-02}	4.7381×10^{-02}	8.9602×10^{-02}

For the 1D slab reactor, the computation time of FEM is 1.25 s, and the training times of DRM, GIPMNN, and PC-GIPMNN are 4788.37 s, 10614.57 s, and 16052.16 s, respectively. For the 2D reactor, the computation time of FEM is 3.64 s, and the training times of DRM, GIPMNN, and PC-GIPMNN are 5827.91 s, 18444.39 s, and 108072.41 s, respectively. For the 2D IAEA benchmark, the computation time of FEM is 5.22 s, and the training times of DRM, GIPMNN, and PC-GIPMNN are 29352.83 s, 64546.74 s, and 137812.92 s, respectively.

Although PC-GIPMNN was better than DRM and GIPMNN, it required significantly more training time to obtain accurate results. Compared with FEM, neural network methods require excessive time. However, neural networks are an emerging method and we believe that they will achieve better results in the near future.

5 Conclusion

In this study, we proposed two methods, GIPMNN and PC-GIPMNN, to solve generalized K-eigenvalue problems in nuclear reactor physics. We also conducted a comprehensive study of GIPMNN, PC-GIPMNN, and DRM. GIPMNN follows the main idea of the inverse power method to find the smallest eigenvalue. The PC-GIPMNN enforce interface conditions through multiple neurons in the output layer. The concept of DRM is to define the function of the Rayleigh quotient and form an optimization problem. Unlike DRM solving for the smallest eigenvalue by directly minimizing the eigenvalue (Rayleigh quotient), the GIPMNN and PC-GIPMNN attain the smallest eigenvalue using the iterative method. All the methods used neural networks to represent functions and the differential was implemented using AD. Finally, we applied these three methods to problems in reactor physics.

Three numerical experiments were conducted to verify the applicability and accuracy of the GIPMNN, PC-GIPMNN, and DRM. In the first 1D example, we used inner and outer

Table 10 Results obtained via GIPMNN, PC-GIPMNN, and DRM when compared with FEM for the 2D IAEA benchmark problem. Specifically, $k_{\text{eff}}^{\text{rel}}$ denotes the relative error of k_{eff}. Especially, DRM is trained with $N_{\text{epoch}} = 50000$

Problem	$k_{\text{eff(FEM)}}$	$k_{\text{eff(GIPMNN)}}$	$k_{\text{eff(PC-GIPMNN)}}$	$k_{\text{eff(DRM)}}$	$k_{\text{eff}}^{\text{rel}}{}_{(\text{GIPMNN})}$	$k_{\text{eff}}^{\text{rel}}{}_{(\text{PC-GIPMNN})}$	$k_{\text{eff}}^{\text{rel}}{}_{(\text{DRM})}$
IAEA	0.9688	0.9692	0.9691	**0.9685**	4.0370×10^{-04}	3.0812×10^{-04}	2.8218×10^{-04}

The bold numbers are the best results

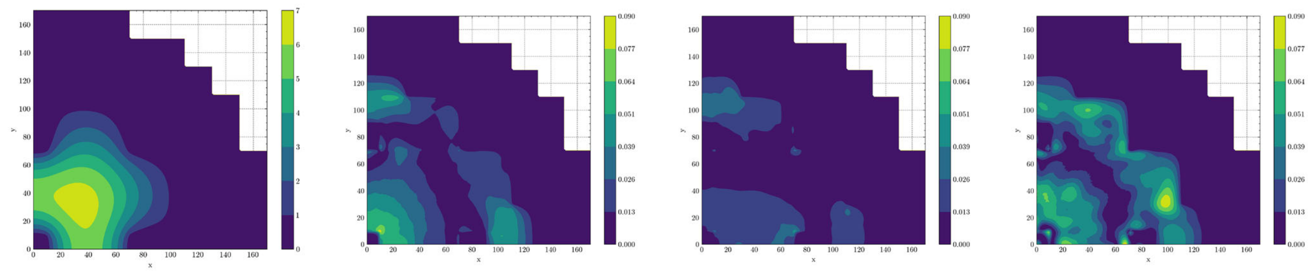


Fig. 12 (Color online) First column shows the heatmap of the eigenfunction of FEM (the first column) and the other columns show the heatmaps of the relative error of GIPMNN (the second column), PC-GIPMNN (the third column), and DRM (the fourth column) for the 2D IAEA benchmark problem. However, by enforcing the interface conditions, PC-GIPMNN outperforms GIPMNN and DRM, as shown in the third column

iterations for the simulation. According to our test, the best ratio of outer and inner iterations was 1 : 1. Furthermore, we compared the results of the GIPMNN, PC-GIPMNN, and DRM with those of the FEM. For the continuous problem, the solution learned by the GIPMNN was more accurate than those learned by the DRM and PC-GIPMNN. For interface problem, the eigenvalue and eigenfunction learned by PC-GIPMNN were better than that learned by DRM and GIPMNN. This is due to the interface conditions that are implemented in the loss function of PC-GIPMNN.

In the 2D examples, we observed the failure of DRM on the 2D experiments. The DRM can learn the eigenvalue and eigenfunction at the early stage of the training process, and the results decrease when N_{epoch} increases. Therefore, we selected $N_{\text{epoch}} = 50000$ to train the DRM and compare the results obtained by the GIPMNN, PC-GIPMNN, and DRM with those obtained by the FEM. The results show that the PC-GIPMNN outperforms the GIPMNN and DRM for the results of eigenvalue and eigenfunction. Moreover, the GIPMNN and PC-GIPMNN are more stable than the DRM.

Although good results were obtained, there are still some aspects that need to be examined in the future. First, given that the architecture of a neural network significantly influences the accuracy of our methods, it is important to design a universal network architecture to achieve high accuracy. For example, MLP is widely used in the current field of solving PDEs using neural networks, and ResNet [60] is effective in improving the convergence rate and may even obtain better results than MLP. Recently, a transformer [74] was used for operator learning and better results were obtained. Sifan Wang et al. [62] investigated the effect of MLP on operator learning and proposed a modified MLP [61], which is a new architecture of MLP that improves accuracy. Although GIPMNN and PC-GIPMNN are more stable than DRM and PC-GIPMNN is more accurate than DRM, they require a large number of iterations and a long training time to achieve good results. Therefore, improving convergence and reducing training time will be investigated in our future study. Furthermore, the failure of DRM on 2D experiments on the eigenvalue problem should be studied and clarified. Finally, for the interface problem, a suitable sampling algorithm can facilitate the training process and provide a better approximation. The next study could also involve the implementation of the proposed networks in emerging reactor digital twins [75–77] as core solvers.

Author Contributions All authors contributed to the study conception and design. Material preparation, data collection and analysis were performed by Qi-Hong Yang, Yu Yang, Yang-Tao Deng, Qiao-Lin He, He-Lin Gong, and Shi-Quan Zhang. The first draft of the manuscript was written by Qi-Hong Yang and all authors commented on previous versions of the manuscript. All authors read and approved the final manuscript.

Declarations

Conflict of interest The authors declare no conflict of interest.

References

1. A. Hébert, *Applied Reactor Physics*, 3rd edn. (Presses internationales Polytechnique, 2020)
2. T.B. Fowler, D.R. Vondy, Nuclear reactor core analysis code: Citation. (Jan 1969) https://www.osti.gov/biblio/4772428
3. A. Hébert, Development of the nodal collocation method for solving the neutron diffusion equation. Ann. Nuclear Energy **14**, 527–541 (1987). https://doi.org/10.1016/0306-4549(87)90074-0
4. L.A. Semenza, E.E. Lewis, E.C. Rossow, The application of the finite element method to the multigroup neutron diffusion equation. Nuclear Sci. Eng. **47**, 302–310 (1972). https://doi.org/10.13182/NSE72-A22416

5. A. Hébert, Application of a dual variational formulation to finite element reactor calculations. Ann. Nuclear Energy **20**, 823–845 (1993). https://doi.org/10.1016/0306-4549(93)90076-2
6. L. Wanai, G. Helin, Z. Chunyu, Solution of neutron diffusion problems by discontinuous Galerkin finite element method with consideration of discontinuity factors. J. Nuclear Eng. Radiat. Sci. **9**, 031503 (2023). https://doi.org/10.1115/1.4055379
7. R. Lawrence, Progress in nodal methods for the solution of the neutron diffusion and transport equations. Prog. Nuclear Energy **17**, 271–301 (1986). https://doi.org/10.1016/0149-1970(86)90034-X
8. K.S. Smith, An analytic nodal method for solving the two-group, multidimensional, static and transient neutron diusion equation. (Mar 1979) http://hdl.handle.net/1721.1/15979
9. P. An, Y. Ma, P. Xiao et al., Development and validation of reactor nuclear design code corca-3d. Nuclear Eng. Technol. **51**, 1721–1728 (2019). https://doi.org/10.1016/j.net.2019.05.015
10. A. Kuz'min, Iterative methods for solving nonlinear problems of nuclear reactor criticality. Phys. Atomic Nuclei **75**, 1551–1556 (2012). https://doi.org/10.1134/S1063778812130042
11. W.M. Stacey, *Nuclear reactor physics* (John Wiley & Sons, 2007). https://doi.org/10.1002/9783527611041
12. S. Marguet, *The physics of nuclear reactors* (Springer, 2018). https://doi.org/10.1007/978-3-319-59560-3
13. K. Tang, X. Wan, C. Yang, Das-pinns: a deep adaptive sampling method for solving high-dimensional partial differential equations. J. Comput. Phys. **476**, 111868 (2023). https://doi.org/10.1016/j.jcp.2022.111868
14. J.P. Argaud, B. Bouriquet, P. Erhard et al., *Data assimilation in nuclear power plant core* (Springer, 2010). https://doi.org/10.1007/978-3-642-12110-4_61
15. H. Gong, Y. Yu, Q. Li et al., An inverse-distance-based fitting term for 3d-var data assimilation in nuclear core simulation. Ann. Nuclear Energy **141**, 107346 (2020). https://doi.org/10.1016/j.anucene.2020.107346
16. S. Cheng, J. Chen, C. Anastasiou et al., *Generalised latent assimilation in heterogeneous reduced spaces with machine learning surrogate models*, vol. 94 (Springer, 2023). https://doi.org/10.1007/s10915-022-02059-4
17. S. Cheng, D. Lucor, J.P. Argaud, Observation data compression for variational assimilation of dynamical systems. J. Comput. Sci. **53**, 101405 (2021). https://doi.org/10.1016/j.jocs.2021.101405
18. S. Riva, C. Introini, S. Lorenzi et al., Hybrid data assimilation methods, part i: Numerical comparison between GEIM and PBDW. Ann. Nuclear Energy **190**, 109864 (2023). https://doi.org/10.1016/j.anucene.2023.109864
19. S. Riva, C. Introini, S. Lorenzi et al., Hybrid data assimilation methods, part II: application to the dynasty experimental facility. Ann. Nuclear Energy **190**, 109863 (2023). https://doi.org/10.1016/j.anucene.2023.109863
20. L.C. Evans, Partial differential equations. (2010) https://bookstore.ams.org/gsm-19-r/
21. J. Han, A. Jentzen et al., Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. Commun. Math. Stat. **5**, 349–380 (2017). https://doi.org/10.1007/s40304-017-0117-6
22. J. Han, A. Jentzen, W. E, Solving high-dimensional partial differential equations using deep learning. Proc. Nat. Acad. Sci. **115**, 8505–8510 (2018). https://doi.org/10.1073/pnas.1718942115
23. J. Sirignano, K. Spiliopoulos, Dgm: a deep learning algorithm for solving partial differential equations. J. Comput. Phys. **375**, 1339–1364 (2018). https://doi.org/10.1016/j.jcp.2018.08.029
24. W. E, B. Yu, The deep ritz method: a deep learning-based numerical algorithm for solving variational problems. Commun. Math. Stat. **6**, 1–12 (2018). https://doi.org/10.1007/s40304-018-0127-z
25. M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. J. Comput. Phys. **378**, 686–707 (2019). https://doi.org/10.1016/j.jcp.2018.10.045
26. X. Jiang, D. Wang, Q. Fan et al., Physics-informed neural network for nonlinear dynamics in fiber optics. Laser & Photon. Rev. **16**, 2100483 (2022). https://doi.org/10.1002/lpor.202100483
27. D.W. Abueidda, S. Koric, E. Guleryuz et al., Enhanced physics-informed neural networks for hyperelasticity. Int. J. Numer. Methods Eng. **124**, 1585–1601 (2023). https://doi.org/10.1002/nme.7176
28. D.W. Abueidda, Q. Lu, S. Koric, Meshless physics-informed deep learning method for three-dimensional solid mechanics. Int. J. Numer. Methods Eng. **122**, 7182–7201 (2021). https://doi.org/10.1002/nme.6828
29. R. Laubscher, Simulation of multi-species flow and heat transfer using physics-informed neural networks. Phys. Fluids **33**, 087101 (2021). https://doi.org/10.1063/5.0058529
30. S. Cai, Z. Wang, S. Wang et al., Physics-informed neural networks for heat transfer problems. J. Heat Trans. **143**, 060801 (2021). https://doi.org/10.1115/1.4050542
31. Y. Liu, R. Hu, A. Kraus et al., Data-driven modeling of coarse mesh turbulence for reactor transient analysis using convolutional recurrent neural networks. Nuclear Eng. Des. **390**, 111716 (2022). https://doi.org/10.1016/j.nucengdes.2022.111716
32. Y. Chen, L. Lu, G.E. Karniadakis et al., Physics-informed neural networks for inverse problems in nano-optics and metamaterials. Opt. Express **28**, 11618–11633 (2020). https://doi.org/10.1364/OE.384875
33. T. Kadeethum, T.M. Jørgensen, H.M. Nick, Physics-informed neural networks for solving inverse problems of nonlinear biot's equations: Batch training. (Jun 2020) https://onepetro.org/ARMAUSRMS/proceedings-abstract/ARMA20/All-ARMA20/447584
34. S. Cheng, I. Colin Prentice, Y. Huang et al., Data-driven surrogate model with latent data assimilation: application to wildfire forecasting. J. Comput. Phys. **464**, 111302 (2022). https://doi.org/10.1016/j.jcp.2022.111302
35. S. Cheng, J. Chen, C. Anastasiou et al., Generalised latent assimilation in heterogeneous reduced spaces with machine learning surrogate models. J. Sci. Comput. **94**, 11 (2023). https://doi.org/10.1007/s10915-022-02059-4
36. Y. Gao, M.K. Ng, Wasserstein generative adversarial uncertainty quantification in physics-informed neural networks. J. Comput. Phys. **463**, 111270 (2022). https://doi.org/10.1016/j.jcp.2022.111270
37. C. Oszkinat, S.E. Luczak, I. Rosen, Uncertainty quantification in estimating blood alcohol concentration from transdermal alcohol level with physics-informed neural networks. IEEE Trans. Neural Netw. Learn. Syst. (2022). https://doi.org/10.1109/TNNLS.2022.3140726
38. Y. Yang, P. Perdikaris, Adversarial uncertainty quantification in physics-informed neural networks. J. Comput. Phys. **394**, 136–152 (2019). https://doi.org/10.1016/j.jcp.2019.05.027
39. Y. Liu, D. Wang, X. Sun et al., Uncertainty quantification for multiphase-cfd simulations of bubbly flows: a machine learning-based Bayesian approach supported by high-resolution experiments. Reliab. Eng. Syst. Saf. **212**, 107636 (2021). https://doi.org/10.1016/j.ress.2021.107636
40. Y. Yang, H. Gong, S. Zhang et al., A data-enabled physics-informed neural network with comprehensive numerical study on solving neutron diffusion eigenvalue problems. Ann. Nuclear Energy **183**, 109656 (2023). https://doi.org/10.1016/j.anucene.2022.109656
41. I. Ben-Shaul, L. Bar, N. Sochen, Solving the functional eigen-problem using neural networks. (Jul 2020). https://doi.org/10.48550/arXiv.2007.10205

42. I. Ben-Shaul, L. Bar, N. Sochen, Deep learning solution of the eigenvalue problem for differential operators. Neural Comput. **35**, 1100–1134 (2023). https://doi.org/10.1162/neco_a_01583
43. H. Jin, M. Mattheakis, P. Protopapas, Unsupervised neural networks for quantum eigenvalue problems. (Oct 2020). https://doi.org/10.48550/arXiv.2010.05075
44. H. Jin, M. Mattheakis, P. Protopapas, Physics-informed neural networks for quantum eigenvalue problems. 2022 International Joint Conference on Neural Networks (IJCNN) 1–8 (2022). https://doi.org/10.1109/IJCNN55064.2022.9891944
45. M.H. Elhareef, Z. Wu, Physics-informed neural network method and application to nuclear reactor calculations: a pilot study. Nuclear Sci. Eng. **197**, 1–22 (2022). https://doi.org/10.1080/00295639.2022.2123211
46. V.L. Berdichevsky, Variational principles. Var. Princ. Contin. Mech. (2009). https://doi.org/10.1007/978-3-540-88467-5_1
47. A.G. Baydin, B.A. Pearlmutter, A.A. Radul et al., Automatic differentiation in machine learning: a survey. J. Mach. Learn. Res. **18**, 1–43 (2018)
48. J. Han, J. Lu, M. Zhou, Solving high-dimensional eigenvalue problems using deep neural networks: A diffusion monte carlo like approach. J. Comput. Phys. **423**, 109792 (2020). https://doi.org/10.1016/j.jcp.2020.109792
49. Q. Yang, Y. Deng, Y. Yang et al., Neural networks base on power method and inverse power method for solving linear eigenvalue problems. Comput. Math. Appl. **147**, 14–24 (2023). https://doi.org/10.1016/j.camwa.2023.07.013
50. W.H. Greub, *Linear algebra*, vol. 23 (Springer Science & Business Media, 2013). https://doi.org/10.1007/978-3-662-01545-2
51. M.H. ELHAREEF, Z. WU, Extension of the pinn diffusion model to k-eigenvalue problems. (2022) https://inis.iaea.org/search/search.aspx?orig_q=RN:54002255
52. 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. Comput. Methods Appl. Mech. Eng. **365**, 113028 (2020). https://doi.org/10.1016/j.cma.2020.113028
53. J. Wang, X. Peng, Z. Chen et al., Surrogate modeling for neutron diffusion problems based on conservative physics-informed neural networks with boundary conditions enforcement. Ann. Nuclear Energy **176**, 109234 (2022). https://doi.org/10.1016/j.anucene.2022.109234
54. A. Quarteroni, G. Rozza, *Reduced order methods for modeling and computational reduction*, vol. 9 (Springer, 2014). https://doi.org/10.1007/978-3-319-02090-7
55. J.S. Hesthaven, G. Rozza, B. Stamm, *Certified reduced basis methods for parametrized partial differential equations* (Springer, 2016). https://doi.org/10.1007/978-3-319-22470-1
56. T. Phillips, C.E. Heaney, P.N. Smith et al., An autoencoder-based reduced-order model for eigenvalue problems with application to neutron diffusion. Int. J. Numer. Methods Eng. **122**, 3780–3811 (2021). https://doi.org/10.1002/nme.6681
57. Z. Wang, Z. Zhang, A mesh-free method for interface problems using the deep learning approach. J. Comput. Phys. **400**, 108963 (2019). https://doi.org/10.1016/j.jcp.2019.108963
58. C. He, X. Hu, L. Mu, A mesh-free method using piecewise deep neural network for elliptic interface problems. J. Comput. Appl. Math. **412**, 114358 (2022). https://doi.org/10.1016/j.cam.2022.114358
59. W.F. Hu, T.S. Lin, M.C. Lai, A discontinuity capturing shallow neural network for elliptic interface problems. J. Comput. Phys. **469**, 111576 (2022). https://doi.org/10.1016/j.jcp.2022.111576
60. K. He, X. Zhang, S. Ren, et al., Deep residual learning for image recognition. (June 2016) https://openaccess.thecvf.com/content_cvpr_2016/html/He_Deep_Residual_Learning_CVPR_2016_paper.html
61. S. Wang, Y. Teng, P. Perdikaris, Understanding and mitigating gradient flow pathologies in physics-informed neural networks. SIAM J. Sci. Comput. **43**, A3055–A3081 (2021). https://doi.org/10.1137/20M1318043
62. S. Wang, H. Wang, P. Perdikaris, Improved architectures and training algorithms for deep operator networks. J. Sci. Comput. **92**, 35 (2022). https://doi.org/10.1007/s10915-022-01881-0
63. L. Lyu, K. Wu, R. Du, et al., Enforcing exact boundary and initial conditions in the deep mixed residual method. (Aug 2020). https://doi.org/10.48550/arXiv.2008.01491
64. S. Dong, N. Ni, A method for representing periodic functions and enforcing exactly periodic boundary conditions with deep neural networks. J. Comput. Phys. **435**, 110242 (2021). https://doi.org/10.1016/j.jcp.2021.110242
65. G.F. Carrier, C.E. Pearson, *Partial differential equations: theory and technique* (Academic Press, 2014)
66. A. Buchan, C. Pain, F. Fang et al., A pod reduced-order model for eigenvalue problems with application to reactor physics. Int. J. Numer. Methods Eng. **95**, 1011–1032 (2013). https://doi.org/10.1002/nme.4533
67. F. Hecht, New development in freefem++. J. Numer. Math. **20**, 251–266 (2012). https://doi.org/10.1515/jnum-2012-0013
68. J.P. Argaud, B. Bouriquet, F. De Caso et al., Sensor placement in nuclear reactors based on the generalized empirical interpolation method. J. Comput. Phys. **363**, 354–370 (2018). https://doi.org/10.1016/j.jcp.2018.02.050
69. H. Gong, Y. Yu, Q. Li, Reactor power distribution detection and estimation via a stabilized Gappy proper orthogonal decomposition method. Nuclear Eng. Des. **370**, 110833 (2020). https://doi.org/10.1016/j.nucengdes.2020.110833
70. H. Gong, Z. Chen, W. Wu et al., Neutron noise calculation: a comparative study between sp3 theory and diffusion theory. Ann. Nuclear Energy **156**, 108184 (2021). https://doi.org/10.1016/j.anucene.2021.108184
71. H. Gong, Z. Chen, Y. Maday et al., Optimal and fast field reconstruction with reduced basis and limited observations: Application to reactor core online monitoring. Nuclear Eng. Des. **377**, 111113 (2021). https://doi.org/10.1016/j.nucengdes.2021.111113
72. J. Argaud, B. Bouriquet, H. Gong et al., *Stabilization of (G) EIM in presence of measurement noise: application to nuclear reactor physics* (Springer, 2017). https://doi.org/10.1007/978-3-319-65870-4_8
73. H. Gong, S. Cheng, Z. Chen et al., Data-enabled physics-informed machine learning for reduced-order modeling digital twin: application to nuclear reactor physics. Nuclear Sci. Eng. **196**, 668–693 (2022). https://doi.org/10.1080/00295639.2021.2014752
74. S. Cao, Choose a transformer: Fourier or Galerkin. Adv. Neural Inf. Process. Syst. **34**, 24924–24940 (2021)
75. H. Gong, T. Zhu, Z. Chen et al., Parameter identification and state estimation for nuclear reactor operation digital twin. Ann. Nuclear Energy **180**, 109497 (2023). https://doi.org/10.1016/j.anucene.2022.109497
76. H. Gong, S. Cheng, Z. Chen et al., An efficient digital twin based on machine learning svd autoencoder and generalised latent assimilation for nuclear reactor physics. Ann. Nuclear Energy **179**, 109431 (2022). https://doi.org/10.1016/j.anucene.2022.109431
77. H. Gong, S. Cheng, Z. Chen et al., Data-enabled physics-informed machine learning for reduced-order modeling digital twin: Application to nuclear reactor physics. Nuclear Sci. Eng. **196**, 668–693 (2022). https://doi.org/10.1080/00295639.2021.2014752