Computer Methods in Applied Mechanics and Engineering 429 (2024) 117135

Contents lists available at ScienceDirect

Comput. Methods Appl. Mech. Engrg.

journal homepage: www.elsevier.com/locate/cma

Interface PINNs (I-PINNs): A physics-informed neural networks framework for interface problems

Antareep Kumar Sarma [a], Sumanta Roy [a], Chandrasekhar Annavarapu [a,*], Pratanu Roy [b], Shriram Jagannathan [c]

[a] Department of Civil Engineering, Indian Institute of Technology Madras, Chennai, 600036, Tamil Nadu, India
[b] Atmospheric, Earth and Energy Division, Lawrence Livermore National Laboratory, Livermore, 94551, CA, United States
[c] Nvidia Corporation, Santa Clara, 95051, CA, United States

ARTICLE INFO

Keywords:
Physics-informed neural networks
Heterogeneous materials
Deep learning
Discontinuous fields
Physics-informed machine learning
Interface problems

ABSTRACT

We present a novel physics-informed neural networks (PINNs) framework for modeling interface problems, termed Interface PINNs (I-PINNs). I-PINNs uses different neural networks for any two subdomains separated by a sharp interface such that the neural networks differ only through their activation functions while the other parameters remain identical. The performance of I-PINNs, conventional PINNs, and other existing domain-decomposition PINNs methods such as extended PINNs (XPINNs) and multi-domain PINN (M-PINN) is compared through several one-dimensional, two-dimensional, and three-dimensional benchmark elliptic interface problems. The results demonstrate that I-PINNs provides a root-mean-square-error accuracy, at least two orders of magnitude better than conventional PINNs and XPINNs at approximately one-tenth of the computational cost of conventional PINNs and half the cost of XPINNs. Additionally, while I-PINNs and M-PINN provide comparable accuracies, M-PINN is found to be approximately 50% more expensive.

Recently, much attention has focused on a novel class of computational methods called the physics-informed neural networks (PINNs) [1]. PINNs approximates unknown field variables using feed-forward neural networks to minimize the loss functionals constructed from the residuals of governing field equations, initial and boundary conditions, and/or observed data, at randomly selected collocation points across the computational domain over which the field variables are sought. Using the governing equations to construct loss functionals allows PINNs to address a critical bottleneck of purely data-driven modeling approaches, i.e., the need for large quantities of high-quality training data, procuring which is often prohibitively expensive. Since their inception, PINNs has been applied for modeling a broad spectrum of engineering applications in solid mechanics [2–6], fluid mechanics [7–9], flow in porous media [10–12], and heat transfer [13,14].

Despite their popularity, the application of PINNs for interface problems remains limited. Such problems are ubiquitous and arise in various applications, including subsurface mechanics [15–22], composites [23–29], and materials science [30–32]. The presence of interfaces introduces weak discontinuities or strong discontinuities in the field variables. Numerical methods that rely on an underlying grid to construct approximate solutions require either conforming grids or intrusive data-structure modifications for such problems (see for e.g., [33–43]). PINNs is naturally suited to approximate such fields, given its meshless character. Notable attempts to solve interface problems with PINNs include the work of Zhang et al. [44,45], where physics-informed convolutional neural networks were used to model transient flow in heterogeneous reservoirs. The contribution to the loss functional from the PDE residual in these methods was evaluated through a finite volume approximation. Alternatively, the method of ensemble PINNs (ePINNs) was

* Corresponding author.
E-mail address: annavarapuc@civil.iitm.ac.in (C. Annavarapu).

https://doi.org/10.1016/j.cma.2024.117135
Received 20 March 2024; Received in revised form 3 May 2024; Accepted 5 June 2024
Available online 20 June 2024

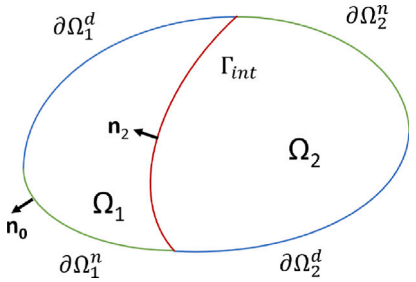

Fig. 1. Schematic of the problem domain with two regions Ω_1 and Ω_2 separated by an interface Γ_{int}.

introduced by Aliakbari et al. [46] for transport modeling in heterogeneous materials, where an ensemble of parallel networks feed into a main network trained by PINNs. Both these methods, however, involve significant modifications to the underlying PINNs architecture.

Methods using more conventional PINNs architectures for modeling heterogeneous materials with sharp discontinuities in material properties borrowed ideas from the finite element method to improve the PINNs approximations for such materials. For instance, mixed formulations where approximations for both the primary variable and its gradient are constructed using separate neural networks were proposed in [47–50]. Additionally, Henkes et al. [51] proposed two strategies for handling heterogeneous problems: adaptively sampling the collocation points with more density near material interfaces, akin to adaptive *h*-refinement in the finite element method, and using domain decomposition to divide the entire domain into multiple square subdomains with flux continuity conditions enforced at each fictional interface. It bears emphasis that domain decomposition and prescribing flux continuity constraints at fictional interfaces was proposed as an enhancement to traditional PINNs framework even for homogeneous materials in the pioneering work of Jagtap et al. [52,53]. The methods proposed by Jagtap et al. were called conservative PINNs (cPINNs) and later extended PINNs (XPINNs), and demonstrated significantly improved approximation accuracy over conventional PINNs. These ideas were also incorporated to model fluid-flow and heat conduction with discontinuous material coefficients in several follow-up studies (see for e.g., Bandai and Ghezzehei [54], Alhubail et al. [55] and the multi-domain physics-informed neural network (M-PINN) framework of Zhang et al. [56]). However, in these methods, the neural networks used in each fictitious subdomain were different; therefore, the methods require a larger number of trainable parameters compared to conventional PINNs.

The objective of this study is twofold. Firstly, we numerically demonstrate the limitations of the conventional PINNs architecture for approximating discontinuous fields due to the use of smooth activation functions. Secondly, we propose a novel architecture termed Interface PINNs (I-PINNs) for approximating both strong and weak discontinuities. The key ingredients of I-PINNs are (a) domain decomposition such that the surface of discontinuity lies along a subdomain boundary and (b) the use of different activation functions in each subdomain while keeping all other parameters identical. The proposed approach shares similarities with the XPINNs framework of Jagtap et al. [53], namely domain decomposition and using different neural networks for different subdomains. However, there are key distinctions that bear emphasis. Firstly, the method is applied to a broad class of elliptic interface problems that may have jumps in both the primary variables and fluxes at the interface. Secondly, unlike XPINNs, the neural networks in different subdomains differ only in their activation functions while keeping the other parameters identical. We demonstrate the performance of I-PINNs on a series of illustrative one-dimensional (1D), two-dimensional (2D), and three-dimensional (3D) examples with complex interface geometries. Through these examples, we showcase the versatility and applicability of I-PINNs in addressing complex interface problems across various spatial dimensions.

The rest of this paper is organized as follows. Section 1 describes the governing equations for a model elliptic interface problem. Section 2 presents a brief overview of conventional PINNs and its limitations in approximating discontinuous functions. Here, I-PINNs and its implementation details are also introduced. Section 3 and Section 4 present several representative 1D, 2D, and 3D numerical examples that compare the accuracy and the cost of I-PINNs with conventional PINNs, XPINNs and M-PINN. Finally, the key conclusions are summarized, and an outlook for the work is provided in Section 5.

1. Model formulation

We begin by considering a typical Poisson's problem with discontinuous coefficients in a domain Ω, with a boundary $\partial\Omega$, partitioned into two non-overlapping subdomains, Ω_1, and Ω_2 by a sharp interface Γ_{int} as shown in Fig. 1. The boundary $\partial\Omega$ has an outward pointing normal $\mathbf{n_0}$ and is partitioned into disjoint sets $\partial\Omega_1$ and $\partial\Omega_2$ such that $\partial\Omega = \overline{\partial\Omega_1 \cup \partial\Omega_2}$.

The governing equations in each domain Ω_{m} (for m=1 and 2) are

$$\begin{aligned} \nabla\cdot(\kappa_{\text{m}}\nabla u_{\text{m}}) &= -f_{\text{m}} \quad \text{in} \quad \Omega_{\text{m}}, \\ u_{\text{m}} &= \Lambda_{\text{m}}^{\text{d}} \quad \text{on} \quad \partial\Omega_{\text{m}}^{\text{d}}, \\ \kappa_{\text{m}}\nabla u_{\text{m}}\cdot\mathbf{n_0} &= \Lambda_{\text{m}}^{\text{n}} \quad \text{on} \quad \partial\Omega_{\text{m}}^{\text{n}}, \end{aligned} \tag{1}$$

where ∇ is the gradient operator, and $\kappa_{\text{m}} > 0$, and f_{m} are given scalar fields that can be discontinuous across the interface. The disjoint sets $\partial\Omega_{\text{m}}^{\text{d}}$ and $\partial\Omega_{\text{m}}^{\text{n}}$ represent the Dirichlet and Neumann portions of the boundary such that $\partial\Omega_{\text{m}} = \overline{\partial\Omega_{\text{m}}^{\text{d}} \cup \partial\Omega_{\text{m}}^{\text{n}} \cup \Gamma_{\text{int}}}$. The

prescribed values of the boundary data for the primary variable and the flux are $\Lambda^{\mathrm{d}}_{\mathrm{m}}$ and $\Lambda^{\mathrm{n}}_{\mathrm{m}}$, respectively. The following jump conditions are provided at the interface Γ_{int}

$$\begin{aligned} [\![u]\!] &= p \qquad \text{on} \quad \Gamma_{\mathrm{int}}, \\ [\![\kappa \nabla u]\!] \cdot \mathbf{n_2} &= q \qquad \text{on} \quad \Gamma_{\mathrm{int}}, \end{aligned} \tag{2}$$

where $\mathbf{n_2}$ is the normal to the interface and points away from Ω_2. The bracket operator $[\![\odot]\!] = \odot_2 - \odot_1$ represents a jump in the quantity $\odot$, and p and q are given scalar fields at the interface.

2. Methodology

2.1. Conventional PINNs framework

PINNs is an unsupervised machine learning technique that can approximate the solutions to partial differential equations (PDEs), without being reliant on a labeled training dataset. PINNs transforms the original problem of solving the governing equations into one of seeking the parameter-set $\boldsymbol{\theta}$ that minimizes a loss functional $\zeta(\boldsymbol{\theta})$, constructed from the residuals of the governing equations, boundary and initial conditions. For the model interface problem described in Section 1, the loss functional $\zeta(\boldsymbol{\theta})$ is formulated as

$$\zeta(\boldsymbol{\theta}) = \mathrm{MSE}_{\mathrm{eq}} + \mathrm{MSE}^{\mathrm{d}}_{\mathrm{bc}} + \mathrm{MSE}^{\mathrm{n}}_{\mathrm{bc}} + \mathrm{MSE}^{\mathrm{d}}_{\mathrm{ic}} + \mathrm{MSE}^{\mathrm{n}}_{\mathrm{ic}}, \tag{3}$$

where $\mathrm{MSE}_{\mathrm{eq}}, \mathrm{MSE}^{\mathrm{d}}_{\mathrm{bc}}, \mathrm{MSE}^{\mathrm{n}}_{\mathrm{bc}}, \mathrm{MSE}^{\mathrm{d}}_{\mathrm{ic}}$, and $\mathrm{MSE}^{\mathrm{n}}_{\mathrm{ic}}$ are the mean-squared error (MSE) losses incurred from the residuals of the governing partial differential equation, the Dirichlet and Neumann boundary conditions, the interface jump conditions for the primary variable, and flux, respectively, and are defined as

$$\begin{aligned} \mathrm{MSE}_{\mathrm{eq}} &= \frac{1}{\mathrm{N}_{\Omega}} \sum_{\mathrm{i}=1}^{\mathrm{N}_{\Omega}} (\nabla \cdot (\kappa_{\mathrm{i}} \nabla u^{\theta}_{\mathrm{i}}) + f_{\mathrm{i}})^2, \\ \mathrm{MSE}^{\mathrm{d}}_{\mathrm{bc}} &= \frac{1}{\mathrm{N}_{\partial\Omega^{\mathrm{d}}}} \sum_{\mathrm{i}=1}^{\mathrm{N}_{\partial\Omega^{\mathrm{d}}}} \alpha^{\mathrm{d}}_{\mathrm{bc}} (u^{\theta}_{\mathrm{i}} - \Lambda^{\mathrm{d}}_{\mathrm{i}})^2, \\ \mathrm{MSE}^{\mathrm{n}}_{\mathrm{bc}} &= \frac{1}{\mathrm{N}_{\partial\Omega^{\mathrm{n}}}} \sum_{\mathrm{i}=1}^{\mathrm{N}_{\partial\Omega^{\mathrm{n}}}} \alpha^{\mathrm{n}}_{\mathrm{bc}} (\kappa_{\mathrm{i}} \nabla u^{\theta}_{\mathrm{i}} \cdot \mathbf{n_0} - \Lambda^{\mathrm{n}}_{\mathrm{i}})^2, \\ \mathrm{MSE}^{\mathrm{d}}_{\mathrm{ic}} &= \frac{1}{\mathrm{N}_{\Gamma_{\mathrm{int}}}} \sum_{\mathrm{i}=1}^{\mathrm{N}_{\Gamma_{\mathrm{int}}}} \alpha_{\mathrm{int}} ([\![u^{\theta}_{\mathrm{i}}]\!]_{\mathrm{int}} - p_{\mathrm{i}})^2, \\ \mathrm{MSE}^{\mathrm{n}}_{\mathrm{ic}} &= \frac{1}{\mathrm{N}_{\Gamma_{\mathrm{int}}}} \sum_{\mathrm{i}=1}^{\mathrm{N}_{\Gamma_{\mathrm{int}}}} \alpha_{\mathrm{int}} ([\![\kappa_{\mathrm{i}} \nabla u^{\theta}_{\mathrm{i}}]\!]_{\mathrm{int}} \cdot \mathbf{n_2} - q_{\mathrm{i}})^2. \end{aligned} \tag{4}$$

At a given input point $\mathbf{x}_{\mathrm{i}}$, the following quantities are now defined. u^{θ}_{i} represents the neural network approximation to the target function u. $\kappa_{\mathrm{i}}, f_{\mathrm{i}}, \Lambda^{\mathrm{d}}_{\mathrm{i}}, \Lambda^{\mathrm{n}}_{\mathrm{i}}, p_{\mathrm{i}}$, and q_{i} are given scalar fields representing the material property, the forcing function, prescribed boundary data for the primary variable and flux, and the prescribed interface jumps for the primary variable and flux, respectively. $\mathrm{N}_{\Omega}, \mathrm{N}_{\partial\Omega^{\mathrm{d}}}, \mathrm{N}_{\partial\Omega^{\mathrm{n}}}$, and $\mathrm{N}_{\Gamma_{\mathrm{int}}}$ are the number of collocation points in the domain over which the residual of the PDE, the Dirichlet boundary conditions, the Neumann boundary conditions, and the interface conditions are minimized, respectively. $\alpha^{\mathrm{d}}_{\mathrm{bc}}, \alpha^{\mathrm{n}}_{\mathrm{bc}}$, and α_{int} are the penalty parameters used to enforce the Dirichlet boundary conditions, Neumann boundary conditions, and the interface conditions (ICs), respectively. Please note that it is indeed possible to use distinct penalty parameters to enforce the jump conditions on the primary variable and the flux at the interface, and a single value is used here only for convenience.

The neural network approximation u^{θ} is obtained using a feed-forward network consisting of a set of neurons arranged in layers that execute calculations consecutively. Neurons in adjacent layers of the fully-connected network are linked, whereas neurons within one single layer are not connected. For a deep neural network with L layers consisting of an input layer, an output layer, and $\mathrm{L}-2$ hidden layers, the network approximation can be represented as a combination of L mathematical functions $f_{\mathrm{L}}(\mathbf{x}_{\mathrm{L}}, \boldsymbol{\theta}_{\mathrm{L}})$ as

$$u^{\theta}(\mathbf{x}, \boldsymbol{\theta}) = f_{\mathrm{L}} \circ f_{\mathrm{L\text{-}1}} \circ f_{\mathrm{L\text{-}2}} \cdots \circ f_1(\mathbf{x}, \boldsymbol{\theta}). \tag{5}$$

The symbol $\circ$ denotes composition of functions and should read as the following recursive operation $f_{\mathrm{L}} \circ f_{\mathrm{L\text{-}1}} = f_{\mathrm{L}}(f_{\mathrm{L\text{-}1}}(\mathbf{x}_{\mathrm{L\text{-}1}}, \boldsymbol{\theta}_{\mathrm{L\text{-}1}}))$. For the Lth layer, the function $f_{\mathrm{L}}(\mathbf{x}_{\mathrm{L}}, \boldsymbol{\theta}_{\mathrm{L}})$ is defined as

$$f_{\mathrm{L}}(\mathbf{x}_{\mathrm{L}}, \boldsymbol{\theta}_{\mathrm{L}}) = \sigma(\mathbf{w}^T_{\mathrm{L}} \mathbf{x}_{\mathrm{L}} + \mathbf{b}_{\mathrm{L}}), \tag{6}$$

where $\mathbf{x}_{\mathrm{L}}$ is the input vector and $\boldsymbol{\theta}_{\mathrm{L}}$ is the set of parameters consisting of the weight matrix $\mathbf{w}_{\mathrm{L}}$ and the bias vector $\mathbf{b}_{\mathrm{L}}$ for the Lth layer. $\sigma(\odot)$ is the activation function that is applied to the output of each neuron to introduce non-linear features in the approximation.

The MSE losses from the governing equation and the Neumann boundary conditions in the loss functional $\zeta(\boldsymbol{\theta})$ depend on the derivatives of the neural network approximation to the field variable u^{θ} with respect to the input variable $\mathbf{x}$. These derivatives are computed using automatic differentiation (AD) which computes the derivatives of a function accurately and efficiently without the need for manual derivation. It decomposes complex functions into elementary operations and applies the chain rule to these

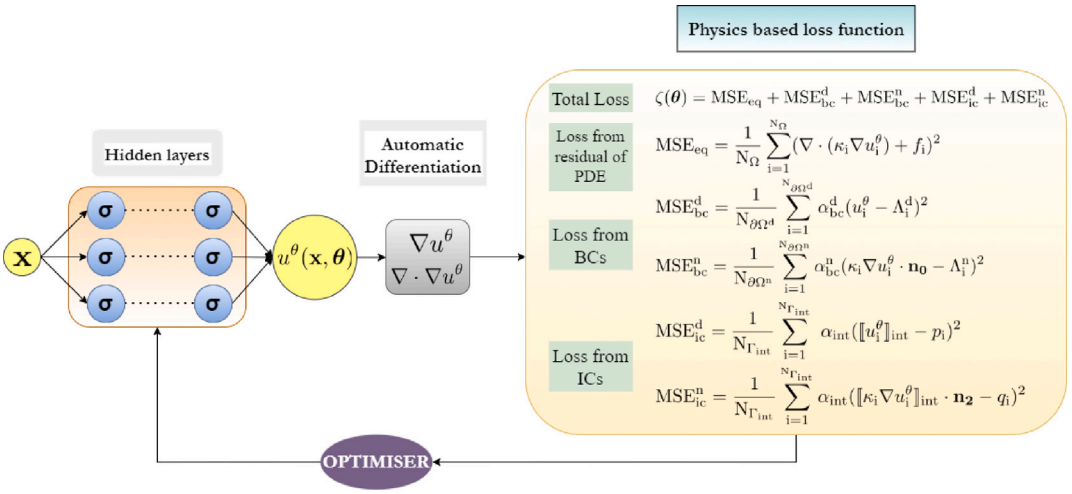

Fig. 2. Schematic of the conventional physics-informed neural networks (PINNs) framework for the model interface problem described in Eqs. (1)–(2).

operations to calculate the derivative of the entire function [57]. We use the AD functionality from the JAX library in this work to compute the derivatives [58].

Once the loss function is constructed, the adaptive moment (Adam) estimation optimizer (see [59]) is used to obtain the optimized set of neural network (NN) parameters $\boldsymbol{\theta}$ that minimize the loss function $\zeta(\boldsymbol{\theta})$. It is worth highlighting that a higher-order optimizer, e.g. L-BFGS [60,61] can indeed be used to further improve the approximation accuracy. As the primary goal of this study is to compare the performance of I-PINNs with existing methods, we use Adam optimizer while being aware of its limitations. This algorithm computes the exponential moving average of the gradients (m_{t}) and squared gradients (v_{t}), and uses their *bias-corrected* forms ($\hat{m}_{\mathrm{k}}$ and $\hat{v}_{\mathrm{k}}$) to update the set of parameters $\boldsymbol{\theta}$ as

$$\boldsymbol{\theta}_{\mathrm{k+1}} = \boldsymbol{\theta}_{\mathrm{k}} - \frac{\eta}{\sqrt{\hat{v}_{\mathrm{k}}} + \epsilon}\hat{m}_{\mathrm{k}}, \tag{7a}$$

$$m_{\mathrm{k}} = \beta_1 m_{\mathrm{k-1}} + (1-\beta_1) g_{\mathrm{k}}, \tag{7b}$$

$$\hat{m}_{\mathrm{k}} = \frac{m_{\mathrm{k}}}{1-\beta_1}, \tag{7c}$$

$$v_{\mathrm{k}} = \beta_2 v_{\mathrm{k-1}} + (1-\beta_2) g_{\mathrm{k}}^2, \tag{7d}$$

$$\hat{v}_{\mathrm{k}} = \frac{v_{\mathrm{k}}}{1-\beta_2}, \tag{7e}$$

where η is the learning rate and $\boldsymbol{\theta}_{\mathrm{k}}$ is the set of parameters at kth iteration. g_{k} refers to the gradient of the loss function $\zeta(\boldsymbol{\theta}_{\mathrm{k}})$ with respect to the parameters $\boldsymbol{\theta}_{\mathrm{k}}$. β_1 and β_2 are hyper-parameters that control the decay rate. ϵ is a small constant added for numerical stability. The initial set of parameters, $\boldsymbol{\theta}_0$, is initialized randomly. Fig. 2 presents a schematic of the conventional architecture of PINNs for the model problem considered in Section 1.

2.2. Approximation of discontinuous functions with neural networks

The model problem presented in Section 2.1 could admit both strongly and weakly discontinuous solutions depending on the nature of the interface coupling conditions. The objective of this section is to study the efficacy of NNs with conventional architecture to approximate such functions.

The loss functional for approximating any general function $f(x)$ with conventional NNs is formulated as

$$\zeta(\boldsymbol{\theta}) = \frac{1}{N_\Omega}\sum_{i=1}^{N_\Omega}\{u^\theta(x_i) - f(x_i)\}^2, \tag{8}$$

where u^θ is the NN approximation to $f(x)$ and N_Ω is the number of collocation points in the domain over which the residual is computed.

The universal approximation theorem guarantees that a sufficiently complex NN can exactly represent an arbitrarily smooth function $f(x)$. Here our goal is to evaluate the performance of NNs to approximate discontinuous functions and C^0 continuous

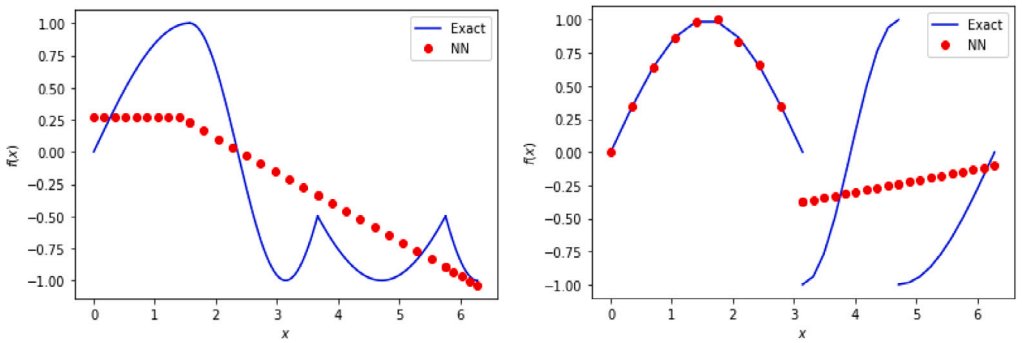

Fig. 3. Approximation of trigonometric functions with (a) a weak discontinuity, and (b) both strong and weak discontinuities, by conventional neural network architecture with one hidden layer and six neurons.

functions. To that end, let us consider the following trial functions.

$$f_1(x) = \begin{cases} \sin x & \text{on } x < \frac{\pi}{2} \\ -\cos 2x & \text{on } \frac{\pi}{2} \le x < \frac{7\pi}{6} \\ \sin x & \text{on } \frac{7\pi}{6} \le x < \frac{11\pi}{6} \\ -\cos 2x & \text{on } x \ge \frac{11\pi}{6} \end{cases}; \quad f_2(x) = \begin{cases} \sin x & \text{on } x < \pi \\ -\cos 2x & \text{on } \pi \le x < \frac{3\pi}{2} \\ \sin x & \text{on } x \ge \frac{3\pi}{2} \end{cases}.$$

$f_1(x)$ and $f_2(x)$ are both trigonometric functions, which are plotted in Figs. 3(a) and 3(b), respectively. While $f_1(x)$ admits a discontinuous derivative (i.e., a weak discontinuity) in the domain of interest, $f_2(x)$ admits both strong and weak discontinuities.

The red markers show the NN approximation to $f_1(x)$ and $f_2(x)$ when the loss functional $\zeta(\theta)$ constructed as per Eq. (8) is minimized. It is clearly observed from the results that this approach which uses a single smooth activation function throughout the domain including at the discontinuity (termed as the conventional NN framework in the subsequent discussion) tries to have a continuous approximation throughout the domain and fails to resolve the discontinuities. In these approximations, we have used a relatively simple architecture with one hidden layer of 6 neurons and rectified linear unit (ReLU) as the activation function. For brevity, we do not present results obtained using other activation functions such as swish, sigmoid, tanh, etc., but similar behavior is observed with those as well.

Although a more complex architecture with more hidden layers and a larger number of neurons per layer could be used to improve the results, it is found that doing so still does not necessarily result in very accurate results, particularly in the vicinity of the discontinuity. In general, it is hard to learn a function with jumps using a smooth activation function and large approximation errors are observed at the locations of discontinuities [62]. This issue can also be described in the context of the optimization process. To tweak the network and improve the output's fit to the training data, neural networks often use back-propagation/gradient descent. This indicates that the loss function should be differentiable at each instant. If the function being approximated contains a significant jump discontinuity, then most standard loss functions used in the optimization step of the neural network will fail at the discontinuity. Moreover, the differentiable loss function ensures a continuous approximation throughout the domain, ignoring the jump at the discontinuities.

2.3. Modified NN architecture for approximating discontinuous functions

The universal approximation theorem states that a feed-forward network with a smooth activation function can approximate only continuous functions [62,63]. Results shown in Fig. 3 are also in agreement with the theorem. Although a recent study by Ismailov [64] provides a theoretical framework for a specialized three-layered artificial neural network to approximate discontinuous functions, the direct applicability of this framework to practical scenarios remains non-trivial given the nonconstructive nature of the method for determining the activation function in the third layer.

Therefore, in practice, to approximate the jumps at the discontinuity accurately, the neural network architecture must be modified to make the network aware of the discontinuities at specific points in the domain. Here, we explore domain-decomposition-based approaches to approximate discontinuous functions using artificial neural networks. For illustration, let us consider a 1D piecewise continuous function $u : \Omega \to \mathbb{R}^1$ such that

$$u(x) = \phi(x) + j(x), \tag{9}$$

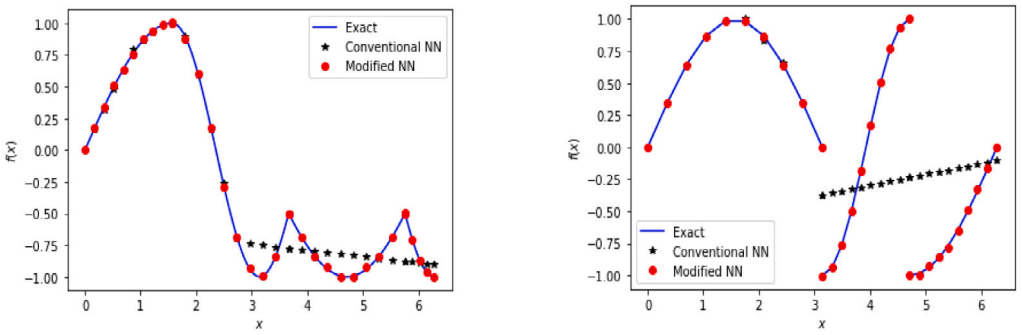

Fig. 4. Approximations of trigonometric functions with (a) weak discontinuities, and (b) strong discontinuities using the proposed approach of using different activation functions across an interface and their comparison with the approximations obtained from a conventional neural network framework.

where $\phi(x)$ is an arbitrarily smooth function in the domain Ω and $j(x)$ is a jump function defined as

$$j(x) = \begin{cases} 0 & \text{in } x < x_{\text{int}} \\ \psi(x) & \text{on } x \geq x_{\text{int}}, \end{cases}$$

where $\psi(x)$ is another arbitrarily smooth function in the domain Ω. The location of discontinuity x_{int} divides the domain into two non-overlapping subdomains Ω^1 and Ω^2 such that $\Omega = \Omega^1 \cup \Omega^2$. The function $u(x)$ can be expressed additively in terms of the arbitrarily smooth functions $\phi(x)$ and $\psi(x)$ as

$$u(x) = \phi(x) + \mathcal{H}(x - x_{\text{int}})\psi(x), \tag{10}$$

where $\mathcal{H}(x)$ is a Heaviside function such that

$$\mathcal{H}(x) = \begin{cases} 0 & \text{on } x < 0 \\ 1 & \text{on } x \geq 0. \end{cases}$$

As such, the function $u(x)$ can now be expressed in terms of two arbitrarily smooth functions limited to the subdomains Ω^1 and Ω^2 such that

$$u(x) = \begin{cases} \phi(x) & \text{in } \Omega^1 \\ \phi(x) + \psi(x) & \text{in } \Omega^2. \end{cases}$$

For illustration, consider a neural network approximation to this function with a single layer given by

$$u^\theta(x) = \sigma(\mathbf{w}^T\mathbf{x} + \mathbf{b}). \tag{11}$$

This approximation would either converge to $\phi(x)$ or $\phi(x) + \psi(x)$ implying that although the approximation is likely to be accurate far away from the interface in either subdomain, it is unlikely to converge to the correct solution at the interface. A neural network approximation to this piecewise continuous function can now be constructed in one of two ways

$$u^\theta(x) = \begin{cases} \sigma^1((\mathbf{w}^1)^T\mathbf{x}^1 + \mathbf{b}^1) & \text{in } \Omega^1 \\ \sigma^2((\mathbf{w}^2)^T\mathbf{x}^2 + \mathbf{b}^2) & \text{in } \Omega^2. \end{cases}$$

In the above approximation, the activation functions could either be identical or distinct in the component subdomains. However, the weights and biases are necessarily distinct in each subdomain, which allows the neural network approximation to converge to a different function. This is the key idea of multi-domain PINN (M-PINN) discussed in Zhang et al. [56] or XPINNs applied to fluid flow modeling in heterogeneous media as described in Alhubail et al. [55]. If weights and biases must be independently prescribed for each subdomain, then indeed one must perform domain decomposition with a boundary specified at the interface and separate neural networks are needed for each subdomain.

However, one could also construct a piecewise continuous approximation, where the weights and biases are identical in the two subdomains, but the activation functions are different across an interface (see also, [65]) such that

$$u^\theta(x) = \begin{cases} \sigma^1(\mathbf{w}^T\mathbf{x}^1 + \mathbf{b}) & \text{in } \Omega^1 \\ \sigma^2(\mathbf{w}^T\mathbf{x}^2 + \mathbf{b}) & \text{in } \Omega^2. \end{cases} \tag{12}$$

The approach proposed in Eq. (12) has an advantage of using fewer trainable parameters in comparison to the alternatives as the neural networks in each component subdomain are only different through their activation functions. We evaluate the performance

Table 1
Comparison of hyper-parameters used in approximating discontinuous functions by traditional and modified architecture.

Hyper-parameters	$f_1(x)$		$f_2(x)$	
	Traditional	Modified	Traditional	Modified
Hidden Layers	2	1	2	1
Neurons	50	6	19	3
Training Points	30	25	70	20
Activation Function	ReLU	ReLU – swish	ReLU	ReLU–swish
Iterations	50 000	50 000	50 000	20 000
RMSE	2.2×10^{-3}	8.1×10^{-4}	3.1×10^{-3}	5.6×10^{-4}
Cost	4	1	5	1

of the proposed approach to approximate the strongly and weakly discontinuous functions $f_1(x)$ and $f_2(x)$ defined in Section 2.2 and plot the results in Fig. 4.

It is evident from Fig. 4 that the proposed approach is capable of approximating the discontinuous functions accurately. On the other hand, with the same hyper-parameters, the conventional NN framework has a tendency to produce continuous approximation throughout the domain. Conventional NN framework can be made more capable by increasing the number of hidden layers and neurons in the network. We gradually increase the network complexity of the conventional NN framework and assess the improvement in their approximation accuracy as a result of this change. At a certain complexity, it is observed that the conventional framework approximates the functions with better accuracy. However, the computational cost required to achieve similar order of approximation by the conventional framework is much higher than that of the proposed method. We use a metric named 'cost' as a representative of the computational time required to complete the simulation, and it is normalized with respect to the proposed modified architecture. The cost of the conventional framework is approximately 5 times that of the proposed approach. Furthermore, the accuracy of the conventional framework is one order of magnitude lower than the proposed approach, despite using many more hidden layers and number of neurons per layer (see Table 1).

As observed in Fig. 4, the proposed architecture requires fewer neurons and training points to approximate strongly discontinuous and weakly discontinuous functions more accurately than the conventional architecture.

2.4. Interface PINNs

We now incorporate the ideas described above into a PINNs framework and propose a modified PINNs architecture tailored for interface problems. We term this modified PINNs architecture as 'Interface PINNs' (I-PINNs). Like PINNs, I-PINNs utilizes deep neural networks to construct an approximation to the field variable. However, I-PINNs constructs an approximation $u^{\theta}_{\mathrm{m}}(\mathbf{x}_{\mathrm{m}}, \boldsymbol{\theta})$ to the field variable $u_{\mathrm{m}}(\mathbf{x}_{\mathrm{m}})$ in each subdomain Ω_{m}, that satisfies the governing partial differential equations and boundary conditions of Eqs. (1)–(2). A deep neural network approximation, $u^{\theta}_{\mathrm{m}}(\mathbf{x}, \boldsymbol{\theta})$, with L layers, including an input layer, an output layer, and L-2 hidden layers is expressed as a composite of L mathematical functions $f_{\mathrm{L}}(\mathbf{x}_{\mathrm{mL}}, \boldsymbol{\theta}_{\mathrm{L}})$ given by

$$u^{\theta}_{\mathrm{m}}(\mathbf{x}_{\mathrm{m}}, \boldsymbol{\theta}) = f_{\mathrm{L}} \circ f_{\mathrm{L\text{-}1}} \circ \ldots \circ f_1(\mathbf{x}_{\mathrm{m}}, \boldsymbol{\theta}). \tag{13}$$

Like earlier, the symbol $\circ$ denotes the composition of functions, such that $f_{\mathrm{L}} \circ f_{\mathrm{L\text{-}1}} = f_{\mathrm{L}}(f_{\mathrm{L\text{-}1}}(\mathbf{x}_{\mathrm{L\text{-}1}}, \boldsymbol{\theta}_{\mathrm{L\text{-}1}}))$ is a recursive operation. For the Lth layer, and mth subdomain, $f_{\mathrm{L}}(\mathbf{x}_{\mathrm{mL}}, \boldsymbol{\theta}_{\mathrm{L}})$ is defined as

$$f_{\mathrm{L}}(\mathbf{x}_{\mathrm{mL}}, \boldsymbol{\theta}_{\mathrm{L}}) = \sigma_{\mathrm{m}}(\mathbf{w}_{\mathrm{L}}^{T} \mathbf{x}_{\mathrm{mL}} + \mathbf{b}_{\mathrm{L}}). \tag{14}$$

In this equation, $\mathbf{x}_{\mathrm{mL}}$ denotes the input vector to the Lth layer in the mth subdomain, and $\boldsymbol{\theta}_{\mathrm{L}}$ represents the parameters of that layer, including the weight matrix $\mathbf{w}_{\mathrm{L}}$ and the bias vector $\mathbf{b}_{\mathrm{L}}$. The function $\sigma_{\mathrm{m}}(\odot)$ is the activation function applied to the output of each neuron in each subdomain Ω_{m}, introducing nonlinear characteristics into the approximation. This is to be contrasted with (a) conventional PINNs where a single activation function is used for both the subdomains and domain decomposition is not performed and (b) M-PINN (and XPINNs) where domain decomposition is performed, but the parameter set $\boldsymbol{\theta}$ is also different for different subdomains.

For illustration, the loss functional $\zeta(\boldsymbol{\theta})$ for solving the Eqs. (1) and (2) with a single interface and two subdomains is defined below

$$\zeta(\boldsymbol{\theta}) = \mathrm{MSE}_{\mathrm{eq}} + \mathrm{MSE}^{\mathrm{d}}_{\mathrm{bc}} + \mathrm{MSE}^{\mathrm{n}}_{\mathrm{bc}} + \mathrm{MSE}^{\mathrm{d}}_{\mathrm{ic}} + \mathrm{MSE}^{\mathrm{n}}_{\mathrm{ic}}, \tag{15}$$

where $\mathrm{MSE}_{\mathrm{eq}}$, $\mathrm{MSE}^{\mathrm{n}}_{\mathrm{bc}}$, $\mathrm{MSE}^{\mathrm{d}}_{\mathrm{bc}}$, $\mathrm{MSE}^{\mathrm{n}}_{\mathrm{int}}$, and $\mathrm{MSE}^{\mathrm{d}}_{\mathrm{int}}$ are the mean-squared residual values evaluated for the governing PDE, the Neumann boundary conditions, Dirichlet boundary conditions and the interface coupling conditions, respectively and are defined as

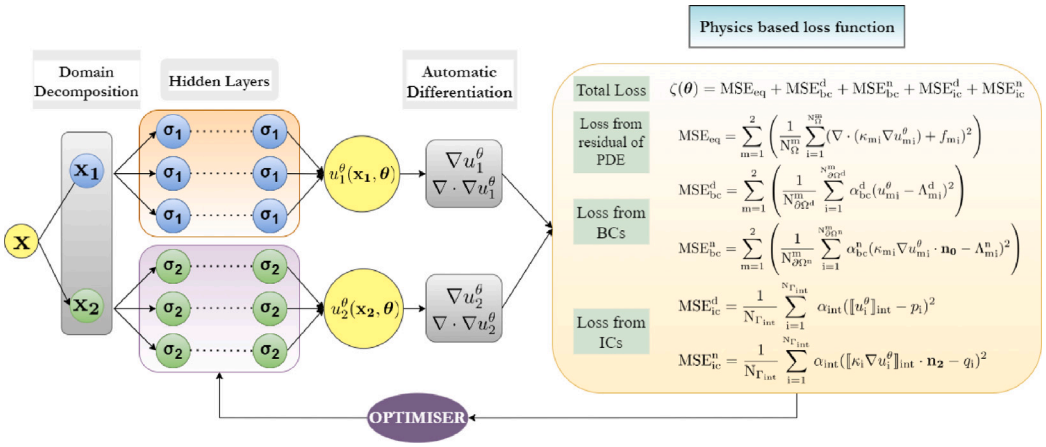

Fig. 5. Schematic of architecture of Interface-PINNs for a problem in 1D over a domain having two regions separated by an interface. This involves segregating input variables over the two domains, which are then fed into two separate neural networks with identical parameters and different activation functions.

$$
\begin{aligned}
\mathrm{MSE}_{\mathrm{eq}} &= \sum_{\mathrm{m}=1}^{2}\left(\frac{1}{\mathrm{N}_{\Omega}^{\mathrm{m}}}\sum_{\mathrm{i}=1}^{\mathrm{N}_{\Omega}^{\mathrm{m}}}(\nabla\cdot(\kappa_{\mathrm{mi}}\nabla u_{\mathrm{mi}}^{\theta})+f_{\mathrm{mi}})^2\right),\\
\mathrm{MSE}_{\mathrm{bc}}^{\mathrm{d}} &= \sum_{\mathrm{m}=1}^{2}\left(\frac{1}{\mathrm{N}_{\partial\Omega^{\mathrm{d}}}^{\mathrm{m}}}\sum_{\mathrm{i}=1}^{\mathrm{N}_{\partial\Omega^{\mathrm{d}}}^{\mathrm{m}}}\alpha_{\mathrm{bc}}^{\mathrm{d}}(u_{\mathrm{mi}}^{\theta}-\Lambda_{\mathrm{mi}}^{\mathrm{d}})^2\right),\\
\mathrm{MSE}_{\mathrm{bc}}^{\mathrm{n}} &= \sum_{\mathrm{m}=1}^{2}\left(\frac{1}{\mathrm{N}_{\partial\Omega^{\mathrm{n}}}^{\mathrm{m}}}\sum_{\mathrm{i}=1}^{\mathrm{N}_{\partial\Omega^{\mathrm{n}}}^{\mathrm{m}}}\alpha_{\mathrm{bc}}^{\mathrm{n}}(\kappa_{\mathrm{mi}}\nabla u_{\mathrm{mi}}^{\theta}\cdot\mathbf{n_0}-\Lambda_{\mathrm{mi}}^{\mathrm{n}})^2\right),\\
\mathrm{MSE}_{\mathrm{ic}}^{\mathrm{d}} &= \frac{1}{\mathrm{N}_{\Gamma_{\mathrm{int}}}}\sum_{\mathrm{i}=1}^{\mathrm{N}_{\Gamma_{\mathrm{int}}}}\alpha_{\mathrm{int}}(\llbracket u_{\mathrm{i}}^{\theta}\rrbracket_{\mathrm{int}}-p_{\mathrm{i}})^2,\\
\mathrm{MSE}_{\mathrm{ic}}^{\mathrm{n}} &= \frac{1}{\mathrm{N}_{\Gamma_{\mathrm{int}}}}\sum_{\mathrm{i}=1}^{\mathrm{N}_{\Gamma_{\mathrm{int}}}}\alpha_{\mathrm{int}}(\llbracket\kappa_{\mathrm{i}}\nabla u_{\mathrm{i}}^{\theta}\rrbracket_{\mathrm{int}}\cdot\mathbf{n_2}-q_{\mathrm{i}})^2,
\end{aligned}
\tag{16}
$$

where m denotes the subdomain number and $\mathrm{N}_{\Omega}^{\mathrm{m}}, \mathrm{N}_{\partial\Omega^{\mathrm{d}}}^{\mathrm{m}}$, and $\mathrm{N}_{\partial\Omega^{\mathrm{n}}}^{\mathrm{m}}$ are the number of collocation points over which the residuals of the governing equations, Dirichlet and Neumann boundary conditions are respectively evaluated for each subdomain. The other symbols have the same definitions as before and are self-explanatory. For problems with multiple subdomains and more than one interface, the extension of the above is straightforward and is omitted here for conciseness. We present a schematic of the I-PINNs framework in Fig. 5 and its implementation is described in Algorithm 1.

It must be emphasized here that although both I-PINNs and M-PINN incorporate the contributions arising from the interface constraints to the loss functionals as per Eq. (16), XPINNs treats them differently. The XPINNs implementation used in the numerical experiments of Section 3 enforces flux continuity and the average value of the jump in the primary variable at the interface, as per 'Remark 5' of Jagtap et al. [53].

3. Numerical examples

This section presents the numerical results of applying I-PINNs on several 1D, 2D, and 3D interface problems. Through these examples, we will compare the performance of I-PINNs against conventional PINNs. The activation functions used in the numerical examples are the standard hyperbolic tangent (tanh), sigmoid, swish, a modified version of the swish function called swish_mod, softmax, and exponential linear unit (ELU). The functions are defined as

$$
\begin{aligned}
&\tanh(x)=\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}; \qquad \mathrm{sigmoid}(x)=\frac{1}{1+e^{-x}};\\
&\mathrm{swish}(x)=\frac{x}{1+e^{-x}}; \qquad \mathrm{swish\_mod}(x)=\frac{0.6x}{2+e^{-2x}};\\
&\mathrm{softmax}(x_{\mathrm{i}})=\frac{e^{x_{\mathrm{i}}}}{\sum_{\mathrm{j}=1}^{\mathrm{N}}e^{x_{\mathrm{j}}}}; \qquad \mathrm{ELU}(x)=\begin{cases} x & \text{if } x>0\\ e^{x}-1 & \text{if } x\le 0\end{cases};\\
&\mathrm{ReLU}(x)=\max(0,x).
\end{aligned}
$$

Algorithm 1 Algorithm for Interface PINNs

Step 1: Generating random weights and biases $\theta(\mathbf{w}, \mathbf{b})$
Step 2: Defining neural network functions for all subdomains
$f_1(\theta, \mathbf{x}_1), f_2(\theta, \mathbf{x}_2), ..., f_{\mathrm{m}}(\theta, \mathbf{x}_{\mathrm{m}})$
Step 3: Generating N_Ω randomly distributed collocation points over domain Ω
Step 4: Segregating the collocation points in different subdomains $(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_{\mathrm{m}})$
Step 5: Defining the objective functional
a) compute necessary derivatives, $\frac{\partial f_{\mathrm{k}}}{\partial \mathbf{x}_{\mathrm{k}}}$, $\frac{\partial^2 f_{\mathrm{k}}}{\partial^2 \mathbf{x}_{\mathrm{k}}}$ etc.
a) compute loss functional
$$\zeta(\theta) = \mathrm{MSE}_{\mathrm{eq}} + \mathrm{MSE}^{\mathrm{d}}_{\mathrm{bc}} + \mathrm{MSE}^{\mathrm{n}}_{\mathrm{bc}} + \mathrm{MSE}^{\mathrm{d}}_{\mathrm{ic}} + \mathrm{MSE}^{\mathrm{n}}_{\mathrm{ic}}$$
Step 6:
for i=0 to max_iterations **do**
Update the parameter set
$$\theta^{i+1} = \theta^i - \eta \cdot \nabla_\theta(\zeta^i(\theta))$$
end for
Step 7: Computing root-mean-square error of the solution obtained
Step 8: Postprocessing and visualizing the results

We compare the performance of I-PINNs and PINNs on computational cost and numerical accuracy. To measure the accuracy of the approximation, we define a root-mean-square error (RMSE) measure as

$$\mathrm{RMSE} = \sqrt{\frac{\sum_{\mathrm{i}=1}^{N} (u^{\theta}_{\mathrm{i}} - u^{\mathrm{ex}}_{\mathrm{i}})^2}{N}}, \tag{17}$$

where u^{θ}_{i} is the neural network approximation to the field u evaluated at the test point i, $u^{\mathrm{ex}}_{\mathrm{i}}$ is the exact value of the field at the test point i, and N is the total number of test points generated randomly over the domain. For two-dimensional and three-dimensional problems, in addition to RMSE, an absolute error metric is used to plot the spatial variation of the error

$$\|e\| = |u^{\theta}_{\mathrm{i}} - u^{\mathrm{ex}}_{\mathrm{i}}|. \tag{18}$$

The relative cost of the two methods is compared by reporting a non-dimensional cost metric such that

$$C = \frac{t_{\mathrm{m}}}{t_{\mathrm{IP}}}, \tag{19}$$

where t_{m} is the total run time for the method under consideration, and t_{IP} is the total run time for the I-PINNs method.

The hyper-parameters used for all the methods under consideration are kept as close to each other as possible. However, it should be emphasized that the hyper-parameters used here are identified through trial and error and are not necessarily optimal. All the methods could benefit from an improvement in their approximation accuracy through automated hyper-parameter tuning. Both PINNs and I-PINNs were implemented as Python notebooks in Google's collaborative environment Google Colab [66], and the JAX library [58] was used to perform automatic differentiation. As a shorthand, we will label the results obtained using Interface PINNs as I-PINNs and those obtained using conventional PINNs as PINNs in all the plots.

3.1. One-dimensional (1D) examples

3.1.1. Poisson's equation with three interfaces

As a first example, we consider Poisson's equation with discontinuous coefficients defined over the 1D computational domain $\Omega = [0, 1]$ with three material interfaces specified at $x = 0.25$, $x = 0.5$, and $x = 0.75$. These material interfaces subdivide the domain into four non-overlapping subdomains $\Omega_1 = [0, 0.25]$, $\Omega_2 = [0.25, 0.5]$, $\Omega_3 = [0.5, 0.75]$, and $\Omega_4 = [0.75, 1]$. The governing equations in each region Ω_{m} (for m = 1, 2, 3, and 4), the boundary conditions at the external boundaries and the interfaces are

$$\begin{aligned} \frac{d}{dx}\left(\kappa_{\mathrm{m}} \frac{du_{\mathrm{m}}}{dx}\right) &= f_{\mathrm{m}} \quad \text{in } \Omega_{\mathrm{m}}, \\ u_1 &= 0 \quad \text{at } x = 0, \\ u_4 &= 0 \quad \text{at } x = 1, \\ [\![u]\!] &= 0 \text{ at } x = 0.25, 0.5, 0.75, \\ \left[\!\left[\kappa \frac{du}{dx}\right]\!\right] &= 0 \text{ at } x = 0.25, 0.5, 0.75. \end{aligned} \tag{20}$$

The material constants corresponding to four distinct regions are explicitly defined as follows: $\kappa_1 = 1$, $\kappa_2 = 0.1$, $\kappa_3 = 0.5$, and $\kappa_4 = 0.75$. Additionally, the source term f_{m} is uniformly set to 1 for all four regions. The set of Eqs. (20) have a closed form

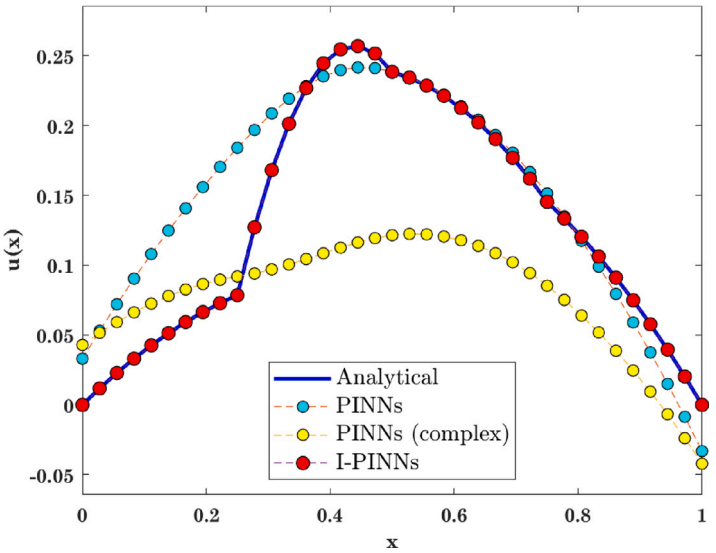

Fig. 6. Comparison of the approximation to Eqs. (20) obtained by PINNs and I-PINNs. The closed-form solution is shown as a solid blue line, while the I-PINNs approximation is shown through red marker circles. The PINNs approximation is obtained using two different sets of hyper-parameters, identical hyper-parameters as I-PINNs (shown in cyan) and with increased network complexity (shown in yellow). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

solution, given as

$$u_1 = \frac{-x^2}{2\kappa_1} + \frac{c_1 x}{\kappa_1}, \quad u_2 = \frac{-x^2}{2\kappa_2} + \frac{c_2 x}{\kappa_2} + \frac{c_3}{\kappa_2},$$
$$u_3 = \frac{-x^2}{2\kappa_3} + \frac{c_4 x}{\kappa_3} + \frac{c_5}{\kappa_3}, \quad u_4 = \frac{-x^2}{2\kappa_4} + \frac{c_6 x}{\kappa_4} + \frac{c_7}{\kappa_4}. \tag{21}$$

The constants $c_1 - c_7$ are functions of the material constants and are given by the following expressions

$$\begin{aligned} c_1 = c_2 = c_4 = c_6 &= \frac{\kappa_1\kappa_2\kappa_3 + \kappa_2\kappa_3\kappa_4 + 3\kappa_1\kappa_3\kappa_4 + 5\kappa_1\kappa_2\kappa_4}{8(\kappa_1\kappa_2\kappa_3 + \kappa_2\kappa_3\kappa_4 + \kappa_1\kappa_3\kappa_4 + \kappa_1\kappa_2\kappa_4)}, \\ c_3 &= -\frac{\kappa_1\kappa_2\kappa_3 - \kappa_2^2\kappa_3 - 2\kappa_2^2\kappa_4 + 2\kappa_1\kappa_2\kappa_4 + \kappa_1\kappa_3\kappa_4 - \kappa_2\kappa_3\kappa_4}{16(\kappa_1\kappa_2\kappa_3 + \kappa_2\kappa_3\kappa_4 + \kappa_1\kappa_3\kappa_4 + \kappa_1\kappa_2\kappa_4)}, \\ c_5 &= -\frac{\kappa_1\kappa_2\kappa_3 - 2\kappa_1\kappa_3^2 - 3\kappa_2\kappa_3^2 + 3\kappa_1\kappa_2\kappa_4 - 3\kappa_2\kappa_3\kappa_4}{16(\kappa_1\kappa_2\kappa_3 + \kappa_2\kappa_3\kappa_4 + \kappa_1\kappa_3\kappa_4 + \kappa_1\kappa_2\kappa_4)}, \\ c_7 &= -\frac{\kappa_1\kappa_2\kappa_3 + \kappa_1\kappa_2\kappa_4 - \kappa_1\kappa_3\kappa_4 - \kappa_2\kappa_3\kappa_4}{8(\kappa_1\kappa_2\kappa_3 + \kappa_2\kappa_3\kappa_4 + \kappa_1\kappa_3\kappa_4 + \kappa_1\kappa_2\kappa_4)}. \end{aligned} \tag{22}$$

Fig. 6 plots the PINNs and I-PINNs approximations to Eqs. (20) and compares them against the closed-form solution shown as the blue solid line in the plot. I-PINNs approximation is obtained using the hyper-parameters listed in the second column of Table 2 and is shown in Fig. 6 with red marker circles. The network employed is relatively shallow, with only two hidden layers. Each hidden layer has 12 neurons and is trained with 40 uniformly distributed points over the computational domain. We used four different activation functions in the four subdomains, namely, the modified swish function in Ω^1, hyperbolic tangent (tanh) in Ω^2, swish in Ω^3, and the sigmoid function in Ω^4. The network is trained for 50,000 iterations. It is observed that the I-PINNs approximation shows excellent agreement with the closed-form solution throughout the domain, including at the three locations of discontinuities, resulting in an RMSE of 4.26×10^{-5}.

Two different PINNs approximations are obtained from two distinct sets of hyper-parameters and plotted in Fig. 6. The PINNs approximation shown with cyan marker circles is obtained by employing a relatively simple network with the hyper-parameters provided in the third column of Table 2. For this approximation, except for using a single activation function in all subdomains, all other hyper-parameters are identical to those used in I-PINNs. The PINNs approximation shown with yellow marker circles uses a much more complex network to improve its chances of accurately approximating the solution. For this approximation, four hidden layers with 50 neurons each are used. The loss function is trained over 300 collocation points for 80,000 iterations. The complete list of hyper-parameters is provided in the fourth column of Table 2. It can be observed from the plot that the PINNs approximation fails to detect the weak discontinuities in both cases. For both cases, the RMSE for the PINNs approximation is three orders of magnitude higher than that obtained using I-PINNs at 3.96×10^{-2} and 7.96×10^{-2}, respectively despite a tenfold increase in computational cost for PINNs with complex architecture.

Table 2
Hyper-parameters used in training conventional PINNs model and proposed I-PINNs model for approximating the solution to Eq. (20).

Parameters	I-PINNs	PINNs	PINNs (complex)
# of hidden layers	2	2	4
# of neurons per hidden layer	12	12	50
# of training points (N_Ω)	40	40	300
α_{int} at each interface	1	1	10
α_{bc} at each boundary	1	1	10
Activation function	swish_mod–tanh –swish–sigmoid	swish	swish
Iterations	50 000	50 000	80 000
RMSE	4.26×10^{-5}	3.96×10^{-2}	7.96×10^{-2}
Cost	1	1	10.63

Table 3
Comparison of hyper-parameters used in proposed I-PINNs model for approximating the solution to Eq. (20) using different combinations of activation functions.

Combination	α_{bc}	α_{int}	RMSE	Cost
swish – tanh – swish_mod – sigmoid	10	1	4.32×10^{-5}	1
swish_mod – sigmoid – tanh – swish	1	1	5.15×10^{-5}	1
swish – tanh – sigmoid – swish_mod	1	1	7.28×10^{-5}	1

Although the results from the I-PINNs approximation are only plotted for a fixed order of activation functions listed in Table 2, the RMSE obtained using a different combination of activation functions is reported in Table 3 and remains approximately the same. We also noticed that for various combinations of activation functions, the network complexity (i.e., the number of hidden layers and neurons per layer) remains unchanged. However, the penalty parameters used to enforce the boundary conditions at the external boundaries and the interface had to be tuned in a few cases. Parameters not listed in Table 3 remain unchanged from Table 2. Therefore, one can conclude that I-PINNs can approximate discontinuous solutions well as long as the activation functions on either side of the interface are different.

3.1.2. Effect of using an alternating pair of activation functions

We now investigate the robustness of our I-PINNs framework by using an alternating pair of activation functions between adjacent subdomains instead of using as many activation functions as the number of subdomains. Using an alternating pair of activation functions is advantageous because it may be impractical to increase the number of activation functions arbitrarily if the number of material interfaces in the computational domain is large. According to Eq. (10), as long as the activation function pair maintains a jump, the neural network approximation should be able to represent the discontinuities in the solution at the subdomain interface. As such, we compare the performance of the method when an alternating pair of activation functions is used with the case when a distinct activation function is used in each subdomain in Fig. 7, where the solution to Eq. (20) is plotted with material constants being considered the same as described in Section 3.1.1. Fig. 7(a) depicts the I–PINNs approximation when using four distinct activation functions in four subdomains, whereas Fig. 7(b) shows the I–PINNs approximation obtained by employing a pair of activation functions across each interface. The two plots demonstrate that both approximations have accurately captured the solution and associated discontinuities. However, we observed that on using an alternating pair of activation functions, the network complexity and the number of training points had to be increased to maintain the same level of accuracy in the solution. For the case of alternating pair of activation functions, we had to use three hidden layers with ten neurons each, and the network was trained over 88 points. By contrast, when using four distinct activation functions in the four subdomains, we had to use two hidden layers of 12 neurons each, and the number of training points was 40. Table 4 lists the hyper-parameters required by both approaches for the preceding example. Consequently, although using an alternating pair of activation functions makes the model simpler to implement for highly heterogeneous problems, it may come with an additional cost. For this particular example, the approach using an alternating pair of activation functions (swish and tanh) was 1.5 times as expensive as the method using four distinct activation functions.

3.2. Two-dimensional (2D) examples

In this section, we evaluate the effectiveness of I-PINNs for 2D problems by testing it on three examples with straight and curved interfaces.

3.2.1. Straight interface

We consider the model interface problem defined in Section 1. The computational domain is a square region such that, $\Omega = [0, 1] \times [0, 1]$, with an interface $\Gamma_{\text{int}} = \{\mathbf{x} : y = x\}$. The interface subdivides the domain into two non-overlapping subdomains

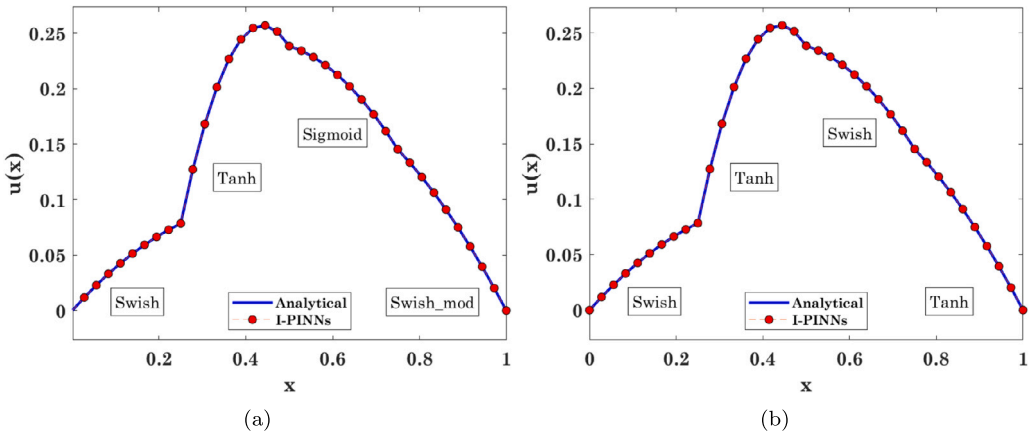

Fig. 7. I-PINNs approximation of the solution to Eq. (20) with, (a) four different activation functions, (b) a pair of activation functions used alternately.

Table 4
Comparison between hyper-parameters used in training the I-PINNs model with different activation functions and a pair of activation functions alternating across an interface for approximating the solution to Eq. (20).

Parameters	Different activation functions	A pair of activation functions
# of hidden layers	2	3
# of neurons per hidden layer	12	10
# of training points (N_Ω)	40	88
α_{int} at each interface	1	1
α_{bc} at each boundary	1	1
Activation function	swish – tanh – sigmoid – swish_mod	swish – tanh
Iterations	50 000	50 000
RMSE	7.28×10^{-5}	6.56×10^{-5}
Cost	1	1.5

$\Omega_1 = \{\mathbf{x} : \quad y \leq x\} \subset \Omega$ and $\Omega_2 = \{\mathbf{x} : \quad y > x\} \subset \Omega$. The model problem, for m = 1, and m = 2, is given as follows

$$\begin{aligned} \nabla \cdot (\kappa_{\text{m}} \nabla u_{\text{m}}) &= 1 && \text{in } \Omega_{\text{m}}, \\ u_{\text{m}} &= \Lambda^{\text{d}}_{\text{m}} && \text{on } \partial\Omega^{\text{d}}_{\text{m}}, \\ [\![u]\!] &= 0 && \text{on } \Gamma_{\text{int}}, \\ [\![\kappa \nabla u]\!] \cdot \mathbf{n_2} &= \frac{-y}{2\sqrt{2}} && \text{on } \Gamma_{\text{int}}, \end{aligned} \tag{23}$$

where $\partial\Omega^{\text{d}}_1 = \{\mathbf{x} : x = 1 \cup y = 0\}$ and $\partial\Omega^{\text{d}}_2 = \{\mathbf{x} : x = 0 \cup y = 1\}$ are the parts of the external boundary with prescribed Dirichlet boundary conditions. Considering $\kappa_1 = 0.5$ and $\kappa_2 = 0.25$, the set of Eqs. (23) have a closed-form analytical solution given as

$$u(\mathbf{x}) = \begin{cases} x^2 + xy & \text{in } \Omega_1, \\ x^2 + y^2 & \text{in } \Omega_2. \end{cases} \tag{24}$$

The known boundary data Λ^{d}_1 and Λ^{d}_2 at the Dirichlet boundaries are prescribed according to Eq. (24).

We use two hidden layers with ten neurons in each layer for approximating Eqs. (23) using both PINNs and I-PINNs. The model is optimized over 420 training points, of which 20 are taken at the interface. We also apply a penalty value of 10 at the interface. Although the results of I-PINNs approximation are presented only for the swish-tanh combination of activation functions, we observed that other pairs of activation functions also provided comparable accuracy. However, those results are not reported here for brevity. Hyper-parameters used in training both PINNs and I-PINNs are presented in Table 5.

The contour plot of the analytical solution is shown in Fig. 8(a). Figs. 8(b) and 8(d) present the PINNs and I-PINNs approximations of the solution to Eqs. (23), respectively. It is observed that the PINNs approximation fails to capture the weak discontinuity at the interface, resulting in a relatively larger absolute error near the interface. By contrast, I-PINNs approximation accurately captures the discontinuity (see Figs. 8(c) and 8(e)). The RMSEs of PINNs and I-PINNs approximations are four orders of magnitude apart and are 7.36×10^{-3} and 2.43×10^{-7}, respectively.

We now attempt to increase the complexity of the conventional PINNs model by employing more neurons and hidden layers with a much larger number of interface and boundary collocation points. Fig. 9 illustrates one of the trials using a conventional PINNs model with a network of five hidden layers with 50 neurons each and optimized over 950 collocation training points. The loss at

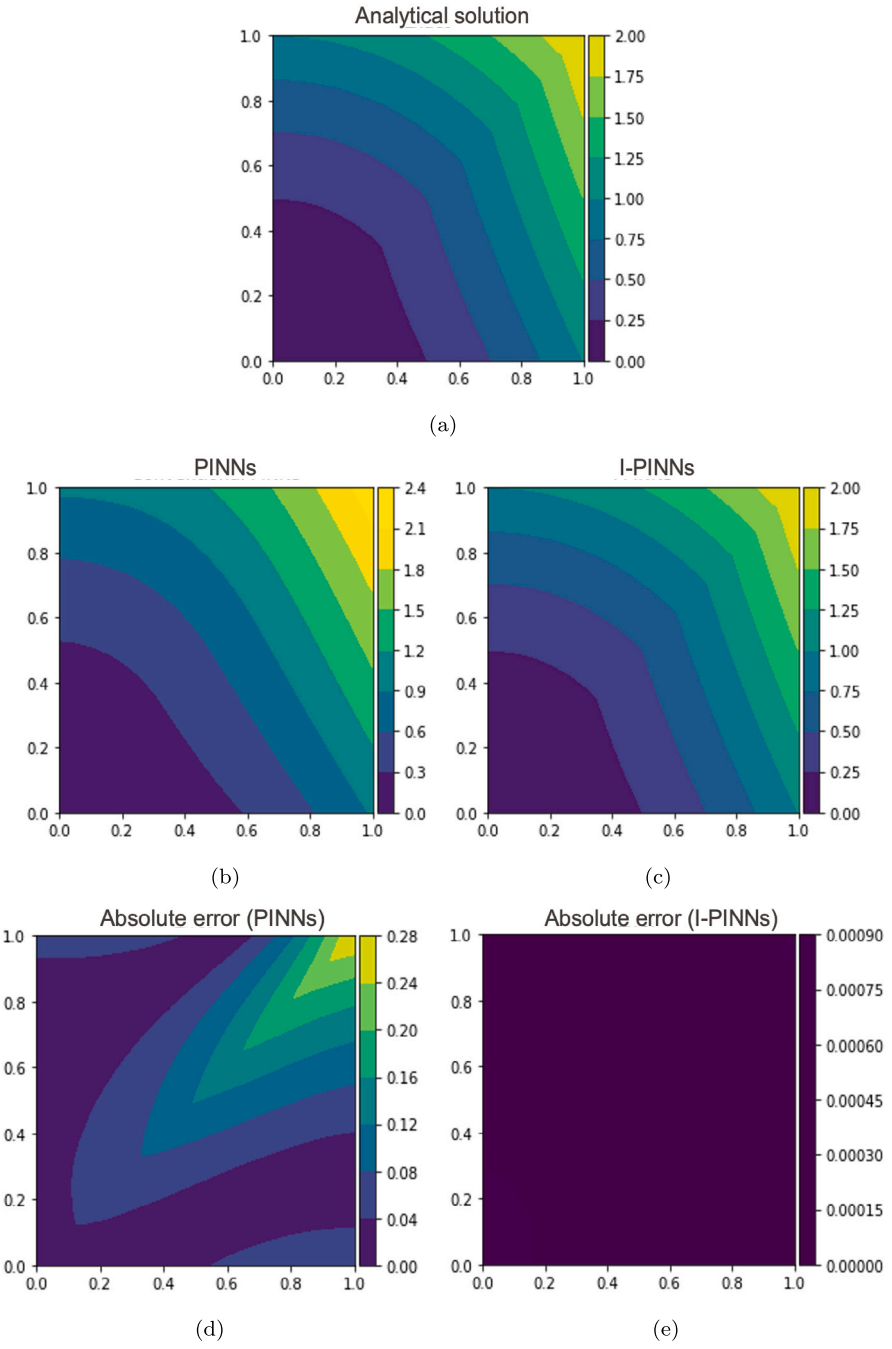

Fig. 8. Contour plot of (a) the analytical solution to Eq. (23) and approximation of the solution by, (b) PINNs, (c) I-PINNs and absolute error obtained with (d) PINNs (RMSE $= 7.36 \times 10^{-3}$), (e) I-PINNs (RMSE $= 2.43 \times 10^{-7}$).

interfaces and boundaries is penalized by a factor of 100. Although approximation accuracy improves by one order of magnitude, it is still inadequate near the interface. The solution plot demonstrates that the approximation is continuous over the entire domain and that discontinuity at the interface is not accurately represented. The subsequent error plot also indicates a larger error near the interface, which ultimately reflects the fact that a single network with a single activation function cannot capture the discontinuity at the interface. Furthermore, because a more complex architecture with much larger number of hidden layers and neurons per layer was used, the computational cost is over ten times as much as I-PINNs.

Table 5
Hyper-parameters used in training both conventional PINNs and I-PINNs model for approximating the solution to Eq. (23).

Parameter	PINNs	PINNs (complex)	I-PINNs
# of hidden layers	2	5	2
# of neurons per hidden layer	10	50	10
# of training points (N_Ω)	420	950	420
# of interface training points (N_{int})	20	50	20
α_{int} at interface	10	100	10
α_{bc} at each boundary	1	100	1
Activation function	swish	swish	swish – tanh
Iterations	30 000	100 000	30 000
RMSE	7.36×10^{-3}	4.26×10^{-4}	2.43×10^{-7}
Cost	1	10.3	1

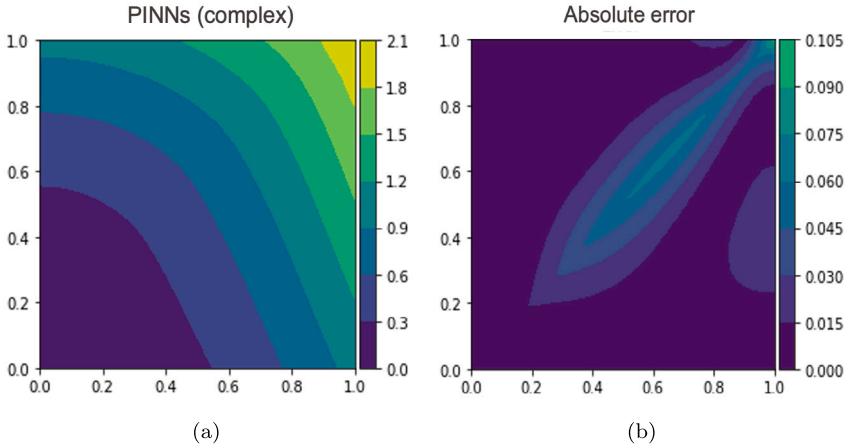

Fig. 9. Contour plot of (a) solution to Eq. (23) by a complex model of conventional PINNs (cost = 10.3), (b) absolute error obtained.

Having considered a straight interface in the preceding example, we now examine problems with curved interfaces to assess the robustness of I-PINNs for arbitrarily shaped interfaces.

3.2.2. Circular interface

We now consider the Poisson's equation defined in a square region such that $\Omega = [-1, 1] \times [-1, 1]$ with a circular interface, $\Gamma_{\text{int}} = \{\mathbf{x} : x^2 + y^2 = 0.25\}$, which subdivides the domain into two regions $\Omega_1 = \{\mathbf{x} : \ x^2 + y^2 \leq 0.25\} \subset \Omega$ and $\Omega_2 = \{\mathbf{x} : \ x^2 + y^2 > 0.25\} \subset \Omega$. The model problem for m = 1, and m = 2, is

$$\begin{aligned} \nabla \cdot (\kappa_{\text{m}} \nabla u_{\text{m}}) &= 0 && \text{in } \ \Omega_{\text{m}}, \\ u_2 &= 1 + \log 2\sqrt{x^2 + y^2} && \text{on } \ \partial\Omega_2^{\text{d}}, \\ [\![u]\!] &= 0 && \text{on } \Gamma_{\text{int}}, \\ [\![\kappa \nabla u]\!] \cdot \mathbf{n}_2 &= -2\kappa_2 && \text{on } \Gamma_{\text{int}}, \end{aligned} \tag{25}$$

where $\partial\Omega_2^{\text{d}} = \{\mathbf{x} : x = -1 \cup x = 1 \cup y = -1 \cup y = 1\}$ is the external boundary where Dirichlet boundary conditions are specified. We choose the values of the two material constants κ_1 and κ_2 as 0.1 and 1 respectively. The set of Eqs. (25) have the following closed-form solution

$$u(\mathbf{x}) = \begin{cases} 1 & \text{in } \Omega_1, \\ 1 + \log 2\sqrt{x^2 + y^2} & \text{in } \Omega_2. \end{cases} \tag{26}$$

Both PINNs and I-PINNs are used to solve Eqs. (25) using the hyper-parameters listed in the first and third columns of Table 6. A contour plot of the analytical solution to Eqs. (25) is plotted in Fig. 10(a). Fig. 10(b) and Fig. 10(c) show the PINNs and I-PINNs approximations, respectively. The corresponding absolute error plots of the two the approximations are plotted in Fig. 10(d) and Fig. 10(e), respectively. As expected, it is observed that the I-PINNs model is able to reproduce the exact solution much more accurately. The RMSEs of the PINNs and I-PINNs approximations are three orders of magnitude apart and are 2.5×10^{-3} and 1.5×10^{-6}, respectively.

Table 6
Hyper-parameters used in training both conventional PINNs and I-PINNs model for approximating the solution to Eq. (25).

Parameter	PINNs	PINNs (complex)	I-PINNs
# of hidden layers	3	5	3
# of neurons per hidden layer	10	50	10
# of training points (N_Ω)	675	950	675
# of interface training points (N_{int})	50	50	50
α_{int} at interface	1	100	1
α_{bc} at each boundary	1	100	1
Activation function	swish	swish	swish – tanh
Iterations	30 000	100 000	30 000
RMSE	2.5×10^{-3}	3.3×10^{-4}	1.5×10^{-6}
Cost	1	10.78	1

Table 7
Hyper-parameters used in training both PINNs and I-PINNs model for approximating the solution to the petal-shaped interface problem.

Parameter	PINNs	PINNs (complex)	I-PINNs
# of hidden layers	4	5	4
# of neurons per hidden layer	10	50	10
# of training points (N_Ω)	950	1300	950
# of interface training points (N_{int})	50	75	50
α_{int} at interface	1	100	1
α_{bc} at each boundary	1	100	1
Activation function	swish	swish	swish – tanh
Iterations	30 000	100 000	30 000
RMSE	1.3×10^{-2}	6.2×10^{-3}	9.5×10^{-6}
Cost	1	10	1

We also demonstrate the approximation results in Fig. 11, from one of the trials using a more complex PINNs architecture with five hidden layers with 50 neurons each. The model was trained using 950 collocation points. Additionally, we apply a 100-fold penalty to the loss at interfaces and boundaries. Although approximation accuracy improves by one order of magnitude (RMSE of 3.3×10^{-4}), it is still two orders of magnitude worse than I-PINNs. Moreover, using a complex PINNs architecture results in a much higher computational cost (almost 11 times that of I-PINNs).

3.2.3. Petal shaped interface

A petal-shaped interface with its geometry defined by the following collection of points is defined

$$\begin{aligned} X &= 0.02\sqrt{5} + (0.5 + 0.2\sin 5\theta)\cos\theta, \\ Y &= 0.02\sqrt{5} + (0.5 + 0.2\sin 5\theta)\cos\theta, \end{aligned} \tag{27}$$

where $-\pi \le \theta \le \pi$. Poisson's equation is considered on a computational domain $\Omega = [-1,1] \times [-1,1]$. The interior and exterior regions to the interface are Ω_1 and Ω_2, respectively. The model problem in these regions is defined as

$$\begin{aligned} &\nabla \cdot (\kappa_{\text{m}} \nabla u_{\text{m}}) = f_{\text{m}} \qquad \text{in} \quad \Omega_{\text{m}}, \\ &u_2 = 0.1(x^2+y^2)^2 - 0.01\log\left(2\sqrt{x^2+y^2}\right) \quad \text{on} \quad \partial\Omega_2^{\text{d}}, \\ &[\![u]\!] = 0.1(x^2+y^2)^2 - 0.01\log\left(\sqrt{x^2+y^2}\right) - (x^2+y^2), \qquad \text{on} \ \Gamma_{\text{int}}, \\ &[\![\kappa \nabla u]\!] \cdot \mathbf{n_2} = (4(x^2+y^2) - 0.1(x^2+y^2)^{-1} - 2)(xn_1 + yn_2) \qquad \text{on} \ \Gamma_{\text{int}}, \end{aligned} \tag{28}$$

where $\partial\Omega_2^{\text{d}}$ is the entire external boundary of the domain. The forcing functions are considered as $f_1 = 4$ and $f_2 = 16(x^2+y^2)$ in Ω_1 and Ω_2 respectively. For $\kappa_1 = 4$ and $\kappa_2 = 10$, the governing equations admit the following piece-wise continuous solution

$$u(\mathbf{x}) = \begin{cases} x^2 + y^2 & \text{in } \Omega_1, \\ 0.1(x^2+y^2)^2 - 0.01\log(2\sqrt{x^2+y^2}) & \text{in } \Omega_2. \end{cases} \tag{29}$$

The analytical solution is plotted as a contour plot in Fig. 12(a). Figs. 12(b) and 12(c) show the PINNs and I-PINNs approximations, each plotted to the same scale using the hyper-parameters listed in Table 7. For a better visual representation of the approximation, specifically at the interface, we utilize a scatter plot with 27,000 points in the domain. The error plots are plotted as contour plots. We observe from the plots that the PINNs approximation fails to predict the inner region, including the interface and has a high RMSE of 1.3×10^{-2}. However, the I-PINNs approximation is much better both at the interface and in the region interior to the interface with an RMSE of 9.54×10^{-6}. Corresponding error plots (see Figs. 12(d), 12(e)) depict the contrasting accuracy of

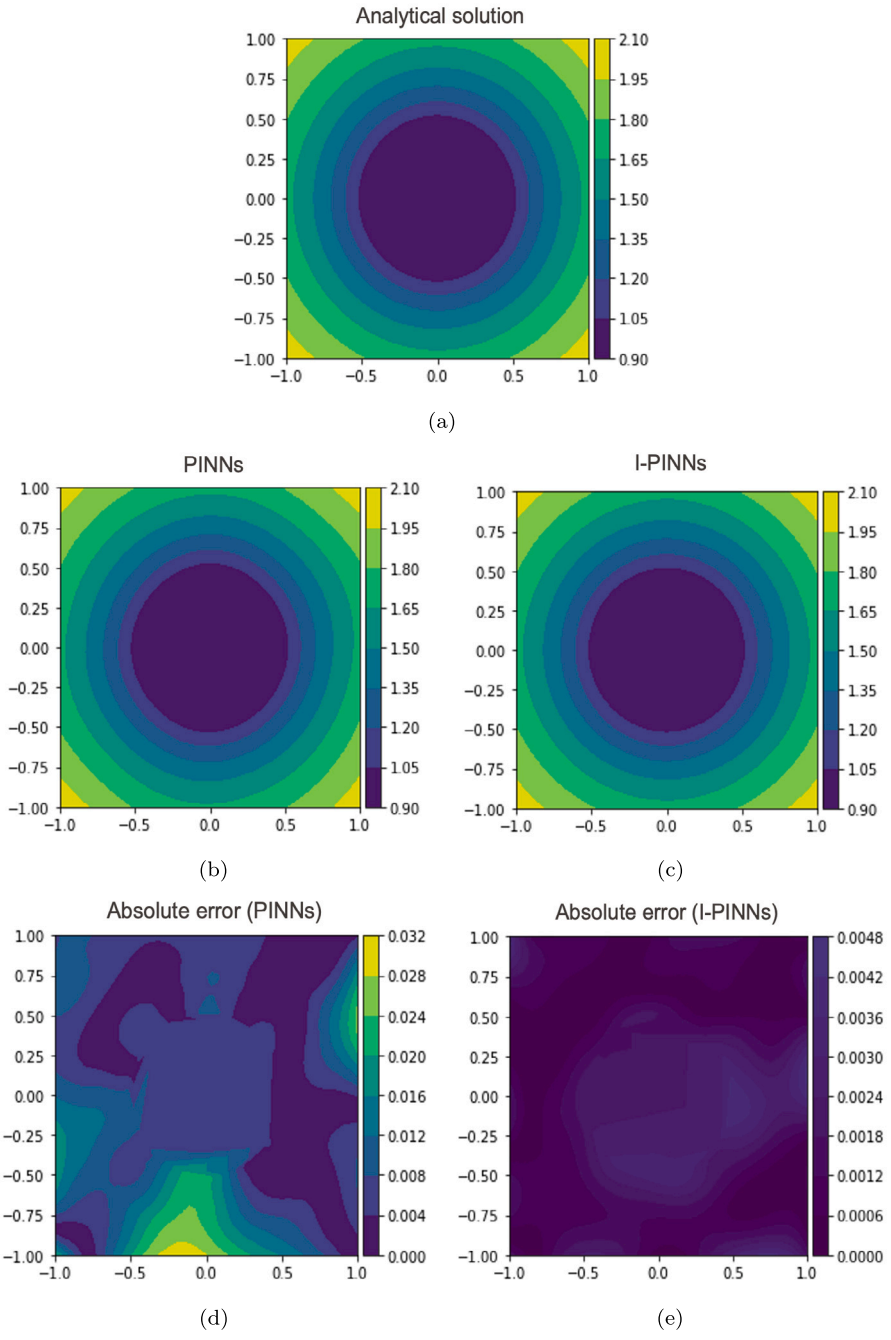

Fig. 10. Contour plot of (a) the analytical solution to Eq. (25) and approximation of the solution by (a) PINNs, (b) I-PINNs and absolute error obtained with (c) PINNs (RMSE $= 2.5 \times 10^{-3}$), (d) I-PINNs (RMSE $= 1.53 \times 10^{-6}$).

approximation by the two architectures. This example validates the capability of the proposed architecture to approximate both strong and weak discontinuities at the interfaces. It must be highlighted that the architecture used in this example is more complex compared to the previous example. Here, four hidden layers with ten neurons per layer were used compared to three hidden layers with ten neurons per layer in the example with a circular interface. However, this is not entirely surprising because in addition to the interface topology being more complex, the analytical solution also exhibits both strong and weak discontinuities here which was not the case in the previous examples.

PINNs approximation is also constructed by employing 5 hidden layers with 50 neurons in each hidden layer over 1300 collocation points. We increase the penalty factor applied in the loss terms at interface and boundaries to 100. Results show that

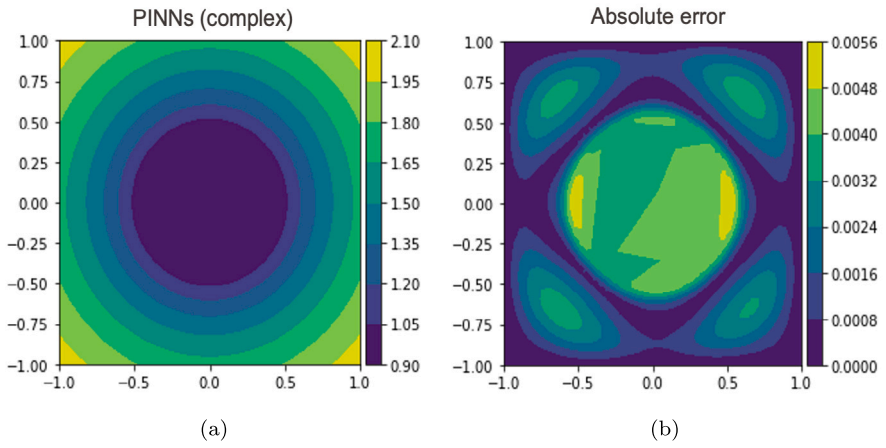

Fig. 11. Contour plot of (a) solution to Eq. (25) using complex network of conventional PINNs (cost = 10.78), (b) absolute error obtained.

the approximation accuracy improves only marginally (RMSE of 6.2×10^{-3}) despite a tenfold increase in the computational cost. The approximation continues to produce large errors near the interface (see Fig. 13). The solution plot shows that the approximation is continuous throughout the domain, and discontinuity at the interface is not captured. As such, we can conclude that a single network with a single smooth activation function is unable to capture the discontinuity at the interface.

3.3. Three-dimensional (3D) examples

In this section, we evaluate the effectiveness of I-PINNs for 3D problems by testing it on a problem with an ellipsoidal interface, followed by another example with a popcorn-shaped interface.

3.3.1. Ellipsoidal interface

Poisson's equation with discontinuous coefficients is solved on the computational domain $\Omega = [-1,1]\times[-1,1]\times[-1,1]$. An ellipsoidal interface $\Gamma_{\text{int}} = \{\mathbf{x} : \psi(\mathbf{x}) = 0, \psi(\mathbf{x}) = 2x^2+3y^2+6z^2-1.69\}$ subdivides the domain into two non-overlapping subdomains $\Omega_1 = \{\mathbf{x} : \ \psi(\mathbf{x}) \le 0\}$ and $\Omega_2 = \{\mathbf{x} : \ \psi(\mathbf{x}) > 0\}$. The following model problem is considered. Given $f_1 = 0, f_2 = -12, \kappa_1 = 2$, and $\kappa_2 = 60$, find u_{m} (for $m = 1$, and $m = 2$) such that

$$\begin{aligned}
\nabla\cdot(\kappa_{\text{m}}\nabla u_{\text{m}}) &= -f_{\text{m}} && \text{in } \ \Omega_{\text{m}},\\
u_2 &= \Lambda_2^{\text{d}} && \text{on } \ \partial\Omega_2^{\text{d}},\\
[\![u]\!] &= (x+y+z)-(x^2+y^2+z^2) && \text{on } \ \Gamma_{\text{int}},\\
[\![\kappa\nabla u]\!]\cdot\mathbf{n}_2 &= (60-4x)4x+(60-4y)6y+(60-4z)12z && \text{on } \ \Gamma_{\text{int}},
\end{aligned} \tag{30}$$

where $\partial\Omega_2^{\text{d}} = \{\mathbf{x} : x = -1|1 \cup y = -1|1 \cup z = -1|1\}$ is the Dirichlet boundary where the boundary data is specified as per the following closed-form analytical solution.

$$u(x,y,z) = \begin{cases} x^2+y^2+z^2 & \text{in } \Omega_1,\\ x+y+z & \text{in } \Omega_2. \end{cases} \tag{31}$$

The contour plot of the analytical solution is plotted in Fig. 14. Fig. 14(a) shows the variation of the field on the exterior faces of the cubical domain. The domain is also clipped to visualize the solution field in the interior of the cubical domain and along the interface, as shown in Figure 14(b).

The problem is approximated by both PINNs and I-PINNs using the same set of hyper-parameters listed in Table 8 and the approximations are plotted in Figs. 15(a) and 15(b), respectively. The absolute errors of the PINNs and I-PINNs approximations evaluated against the closed-form solution are plotted in Figs. 16(a) and 16(b), respectively. The plots demonstrate that the I-PINNs approximation closely agrees with the closed-form solution, and accurately captures the strong and weak discontinuities at the interface. On the other hand, the PINNs approximation fails to estimate the solution, resulting in high errors specifically near the interface. However, it is worth noting that achieving convergence in 3D requires approximately twice as many iterations as 1D and 2D. In addition, the number of collocation points used for training are also higher, making the model much more computationally intensive. Finally, it is worth highlighting that the RMSEs for 3D approximation is two orders of magnitude higher than 1D and 2D problems discussed earlier.

In line with the previous examples, we also attempt to approximate the solution using PINNs with increased network complexity. The complex PINNs framework employs five hidden layers with 20 neurons each and is trained over 3575 collocation points. We also increase the penalty factor applied in the loss terms at interface and boundaries to 100. Figs. 17(a) and 17(b) plot the solution and

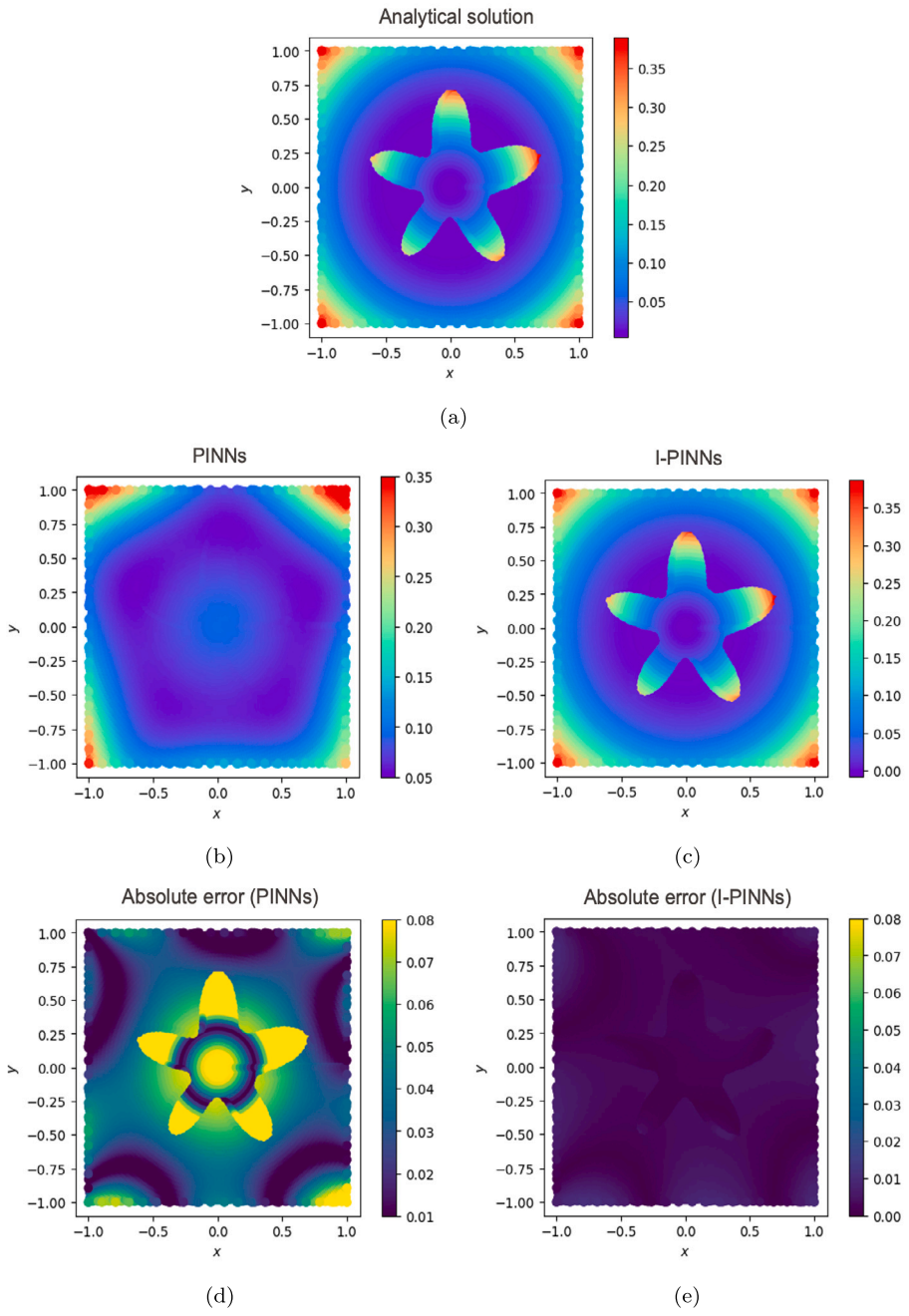

Fig. 12. Contour plot of (a) the analytical solution to the petal-shaped interface problem and approximation to the solution by, (b) PINNs, (c) I-PINNs and absolute error obtained with (d) PINNs (RMSE $= 1.3 \times 10^{-2}$, (e) I-PINNs (RMSE $= 9.54 \times 10^{-6}$).

the absolute error in the solution, respectively, from the complex PINNs model. The plots highlight that, despite utilizing a highly complex network, the approximation generated by the conventional PINNs model exhibits a lack of accuracy near the interface and consequently results in an imprecise approximation of the analytical solution. This further underscores the necessity of implementing I-PINNs architecture in approximating discontinuities at the interface.

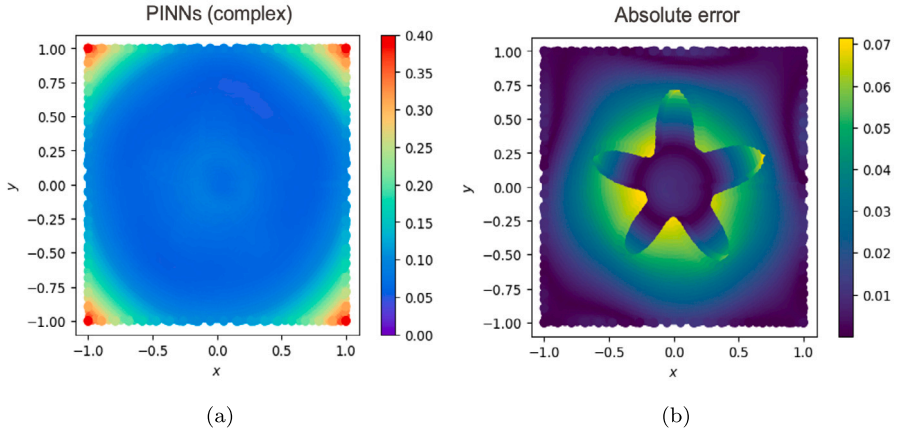

Fig. 13. Contour plot of (a) solution to the petal-shaped interface problem using a complex network of PINNs (cost = 10), (b) absolute error obtained.

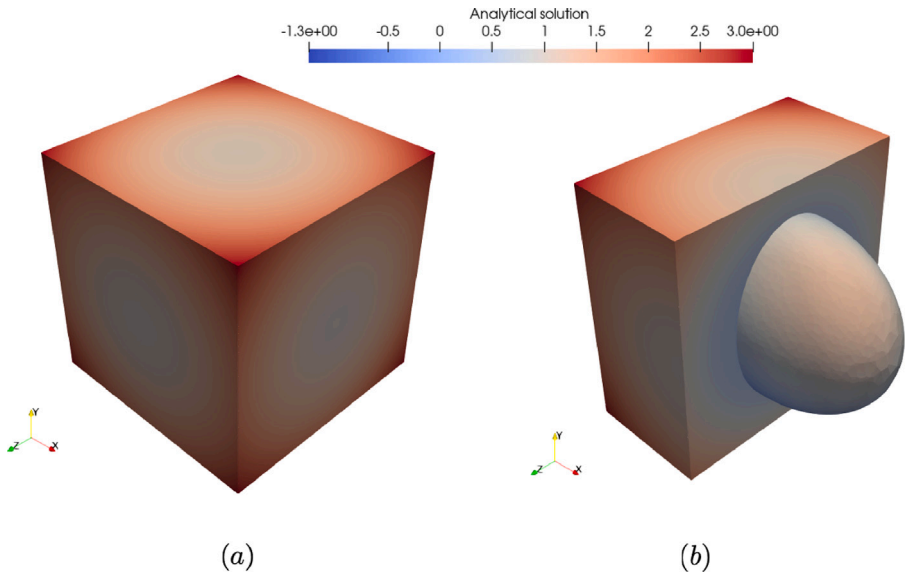

Fig. 14. Contour plot of the analytical solution to the ellipsoidal interface problem over (a) the exterior faces of the domain Ω, and (b) a clipped view in the interior of the domain and along the interface.

Table 8
Hyper-parameters used in training both PINNs and I-PINNs model for approximating the solution to the ellipsoidal interface problem.

Parameter	PINNs	PINNs (complex)	I-PINNs
# of hidden layers	2	5	2
# of neurons per hidden layer	20	20	20
# of training points (N_Ω)	1928	3575	1928
# of interface training points (N_{int})	200	250	200
α_{int} at interface	10	100	10
α_{bc} at each boundary	20	100	20
Activation function	ELU	ELU	ELU – softmax
Iterations	100 000	100 000	100 000
RMSE	3.52	5.4×10^{-1}	1.2×10^{-3}
Cost	1	8.5	1

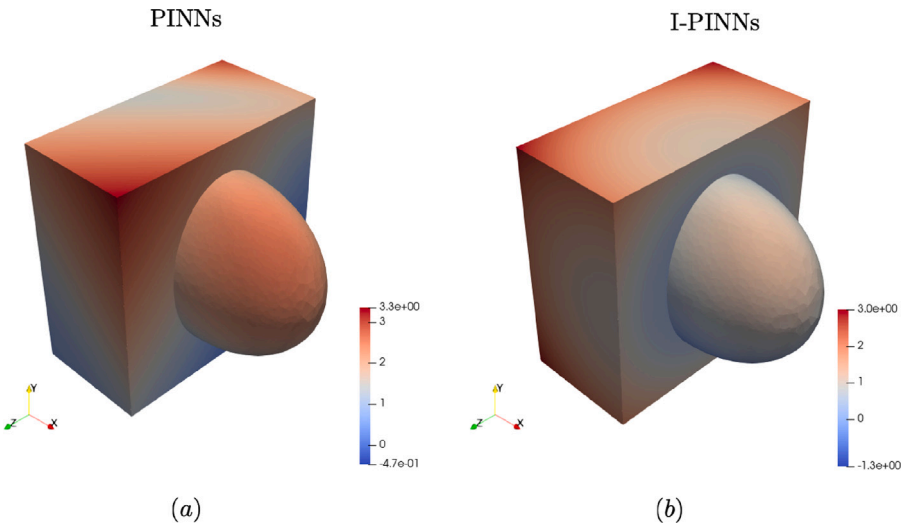

Fig. 15. Contour plots of (a) the PINNs approximation, and (b) the I-PINNs approximation to the ellipsoidal interface problem.

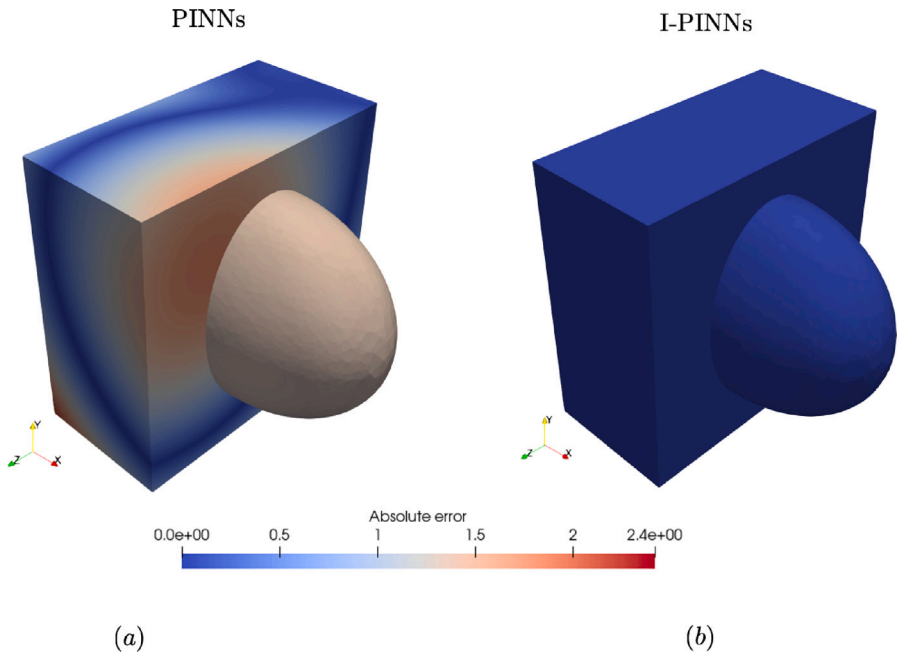

Fig. 16. Contour plots of the absolute errors in (a) the PINNs approximation (RMSE = 3.52), and (b) the I-PINNs approximation (RMSE = 1.2×10^{-3}) to the ellipsoidal interface problem.

3.3.2. Popcorn-shaped interface:

For our final example, we aim to increase the complexity of the problem by introducing an interface shaped like popcorn. The equation of the popcorn-shaped interface is expressed as $\Gamma_{\text{int}} = \{\mathbf{x} : \psi(\mathbf{x}) = 0\}$, where $\psi(\mathbf{x})$ is defined as

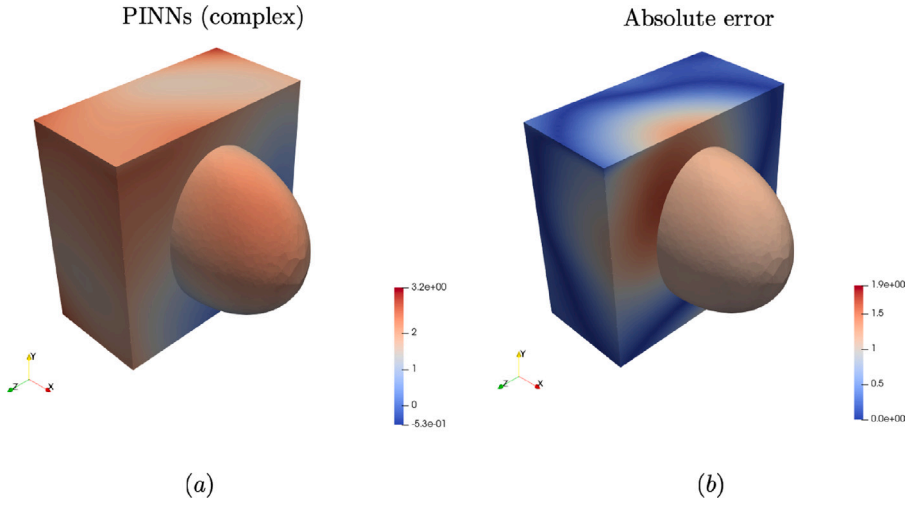

Fig. 17. Contour plots of (a) the PINNs approximation, and (b) the absolute error in approximation for the ellipsoidal interface problem using a complex PINNs architecture.

$$
\begin{aligned}
\psi(\mathbf{x}) &= \sqrt{x^2+y^2+z^2} - r_0 - S(x,y,z),\\
S(x,y,z) &= \sum_{k=0}^{11} A e^{-((x-x_k)^2+(y-y_k)^2+(z-z_k)^2)/\sigma^2},\\
(x_k,y_k,z_k) &= \frac{r_0}{\sqrt{5}}\left[2cos\left(\frac{2k\pi}{5}\right), 2sin\left(\frac{2k\pi}{5}\right), 1\right],\\
&\quad 0 \le k \le 4,\\
(x_k,y_k,z_k) &= \frac{r_0}{\sqrt{5}}\left[2cos\left(\frac{(2(k-5)-1)\pi}{5}\right), 2sin\left(\frac{(2(k-5)-1)\pi}{5}\right), -1\right],\\
&\quad 5 \le k \le 9,\\
(x_{10},y_{10},z_{10}) &= (0,0,r_0),\\
(x_{11},y_{11},z_{11}) &= (0,0,-r_0).
\end{aligned}
\tag{32}
$$

The constants in the Eqs. (32) are considered as $r_0 = 0.6$, A = 2, $\sigma = 0.2$. The computational domain, problem statement, and analytical solution remain the same as the ellipsoidal interface problem described in Section 3.3.1, except for the jump in flux at the interface $\Gamma_{\text{int}} = \{\mathbf{x} : \psi(\mathbf{x}) = 0\}$, which is given as

$$
\begin{aligned}
[\![\kappa \nabla u]\!] \cdot \mathbf{n}_2 = &\left(60-4x\right)\left(\frac{x}{\sqrt{x^2+y^2+z^2}} + 2S(x,y,z)(\frac{x-x_k}{\sigma^2})\right)\\
&+\left(60-4y\right)\left(\frac{y}{\sqrt{x^2+y^2+z^2}} + 2S(x,y,z)(\frac{y-y_k}{\sigma^2})\right)\\
&+\left(60-4z\right)\left(\frac{z}{\sqrt{x^2+y^2+z^2}} + 2S(x,y,z)(\frac{z-z_k}{\sigma^2})\right).
\end{aligned}
\tag{33}
$$

The analytical solution given by Eq. (31) is plotted in Fig. 18. Like the previous example, we also plot the analytical solution by clipping the cubic domain so that the solution field is visualized in the interior of the domain and along the interface.

Figs. 19(a) and 19(b) depict the approximations of the problem using PINNs and I-PINNs, respectively. Both models use the same set of hyper-parameters listed in Table 9. The plots clearly show that the PINNs model fails to estimate the solution accurately, while the I-PINNs approximation closely agrees with the analytical solution by accurately capturing the discontinuities at the interface. We also compare the approximations with the closed-form solution by computing the absolute error and plot it in Figs. 20(a) and 20(b). The error plots indicate that the I-PINNs approximation significantly outperforms the PINNs model, particularly near the interface.

Nevertheless, it is worth highlighting that to achieve comparable levels of accuracy as the ellipsoidal-interface problem, the number of training points for the popcorn-shaped interface problem are more than two-fold. This indicates that the required number of training points depends on interface topology. More complex the interface shape, the higher the number of required training points.

In line with the previous examples, we also attempt to approximate the solution using PINNs with increased network complexity. The complex PINNs framework employs five hidden layers with 20 neurons each and is trained over 6113 collocation points. We

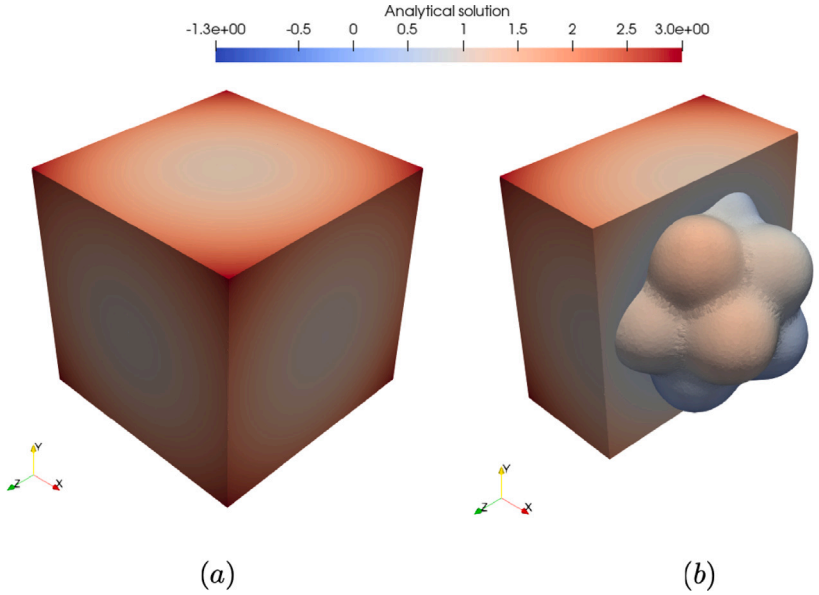

Fig. 18. Contour plot of the analytical solution to the popcorn-shaped interface problem over (a) the exterior faces of the domain Ω, and (b) a clipped view in the interior of the domain and along the interface.

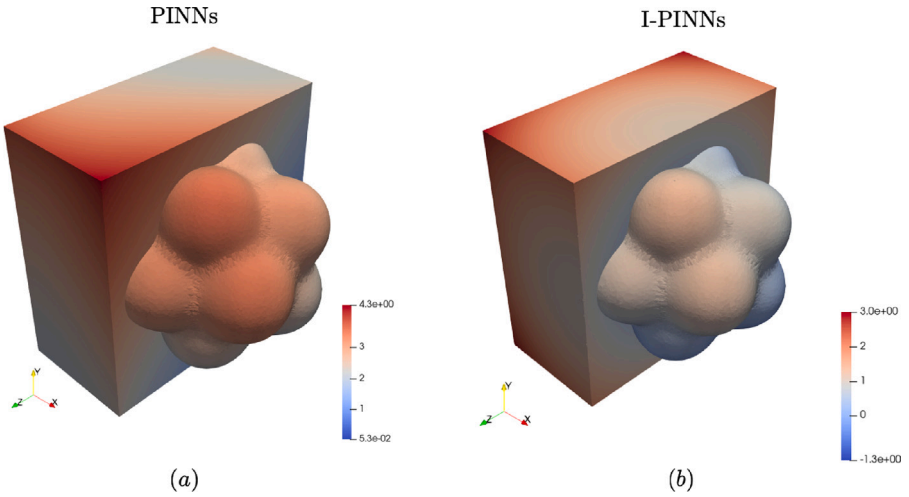

Fig. 19. Contour plots of (a) the PINNs approximation, and (b) the I-PINNs approximation to the popcorn-shaped interface problem.

also increase the penalty factor applied in the loss terms at interface and boundaries to 100. Figs. 21(a) and 21(b) illustrate the solution and the absolute error in the solution approximated by this complex PINNs framework, respectively. The plots highlight that, despite utilizing a highly complex network, the approximation generated by the PINNs exhibits a lack of accuracy near the interface and consequently results in an inaccurate approximation of the analytical solution (RMSE = 1.191).

This example essentially demonstrates the efficacy of the I-PINNs architecture in handling complex geometries and accurately approximating the solution field in the presence of sharp jumps or discontinuities at the interface. The results further highlight the limitations of conventional PINNs models and the potential of the proposed I-PINNs architecture to overcome the difficulties in solving problems with interface discontinuities.

4. Comparison with XPINNs and M-PINN

In this section, we conduct three experiments (in 1D, 2D, and 3D) to compare the performance of I-PINNs with other commonly used domain-decomposition-based PINNs frameworks, i.e., XPINNs and M-PINN, in terms of computational accuracy, and cost. For a fair comparison, the hyper-parameters for all three methods are chosen as close to each other as possible. However, it is noteworthy

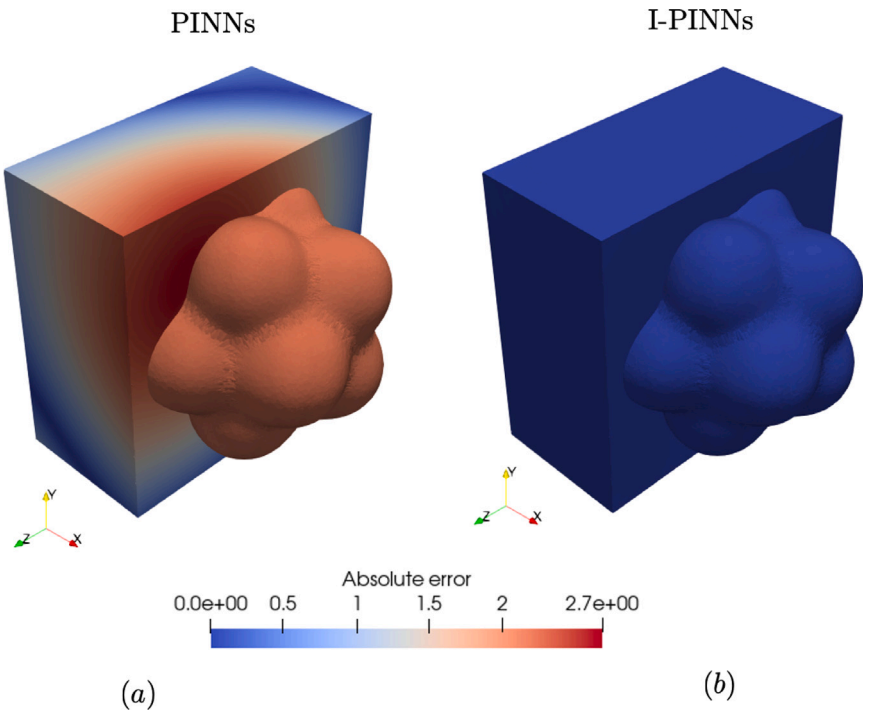

Fig. 20. Contour plots of the absolute errors in (a) the PINNs approximation (RMSE = 1.2), and (b) the I-PINNs approximation (RMSE = 5.5×10^{-3}) to the popcorn-shaped interface problem.

Table 9
Hyper-parameters used in training both PINNs and I-PINNs model for approximating the solution to the popcorn-shaped interface problem.

Parameter	PINNs	PINNs (complex)	I-PINNs
# of hidden layers	2	5	2
# of neurons per hidden layer	5	20	5
# of training points (N_Ω)	6113	6113	6113
# of interface training points (N_{int})	1200	1200	1200
α_{int} at interface	10	100	10
α_{bc} at each boundary	20	100	20
Activation function	swish	swish	swish – tanh
Iterations	100 000	100 000	100 000
RMSE	1.191	8.4×10^{-1}	5.5×10^{-3}
Cost	1	8.8	1

that XPINNs and M-PINN employ a higher number of parameters compared to I-PINNs because they require different weights and biases for each subdomain, whereas I-PINNs does not. To maintain a consistent *bias–variance trade-off* across all three models, we also proportionately increase the number of training points used in XPINNs and M-PINN relative to I-PINNs.

4.1. Poisson's equation in 1D

We report results for a jump problem studied in Xu et al. [67]. The problem considers the Poisson's equation with discontinuous coefficients defined over the domain $\Omega = [0, 1]$ with one material interface at $x = 0.5$. The governing equations in each subdomain Ω_{m} (for m = 1 and 2), the boundary conditions at the external boundaries and the interface are

$$\begin{aligned} \frac{d}{dx}\left(\kappa_{\text{m}} \frac{du_{\text{m}}}{dx}\right) &= 56x^6 \qquad \text{in } \Omega_{\text{m}}, \\ u_1 &= -1 \quad \text{at } x = 0, \\ u_2 &= 769/512 \quad \text{at } x = 1, \\ [\![u]\!] &= 2 \text{ at } x = 0.5, \\ \left[\!\left[\kappa \frac{du}{dx}\right]\!\right] &= 0 \text{ at } x = 0.5. \end{aligned} \tag{34}$$

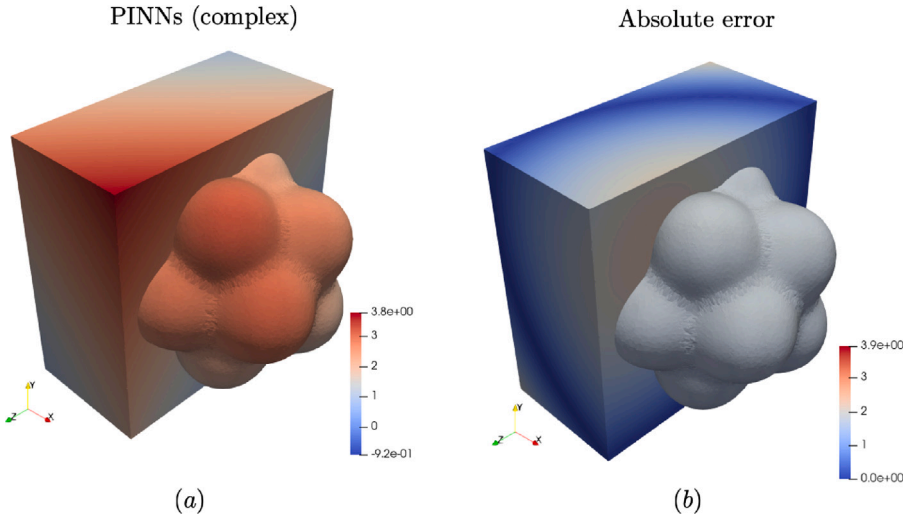

Fig. 21. Contour plots of (a) the PINNs approximation, and (b) the absolute error in approximation for the solution to the popcorn-shaped interface problem using complex PINNs architecture.

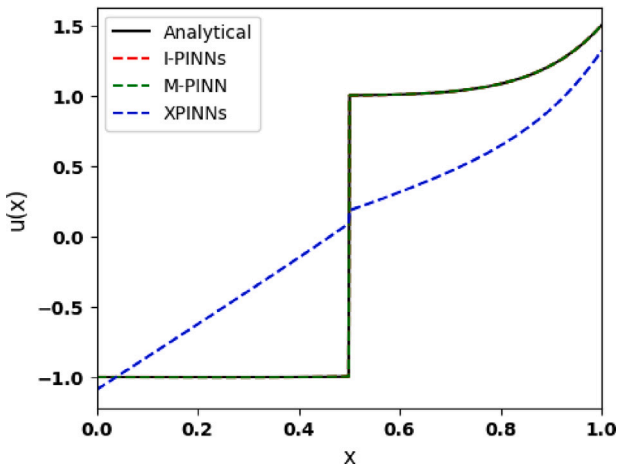

Fig. 22. Approximation of the solution to the 1D problem with Eq. (34) by I-PINNs, M-PINN, XPINNs.

The material constants in the two regions are explicitly defined as: $\kappa_1 = 1$ and $\kappa_2 = 2$. The set of Eqs. (34) have a closed form solution, given as

$$u_1 = x^8 - 1, \quad u_2 = \frac{1}{2}\left(x^8 + 1/256\right) + 1. \tag{35}$$

Table 10 presents the RMSE of the approximations, normalized computational cost, and the hyper-parameters used to train the I-PINNs, M-PINN, and XPINNs models for approximating the solution to Eq. (34). It is evident from the results that XPINNs is the least accurate of the three methods, with approximation accuracies three orders of magnitude worse ($\mathcal{O}(10^{-1})$) than those of I-PINNs and M-PINN ($\mathcal{O}(10^{-4})$). This is also observed in Fig. 22, where the XPINNs approximation significantly deviates from the closed-form solution throughout the domain, while I-PINNs and M-PINN both agree well with the closed-form solution. XPINNs and M-PINN are identical methods except for their treatment of interface constraints, therefore, the poor accuracy of XPINNs for this class of jump problems can be attributed to the way in which the jump conditions are incorporated in the loss functional. Although both I-PINNs and M-PINN have comparable accuracies, they differ significantly in terms of their computational expense. The total training cost for M-PINN is approximately 50% higher than I-PINNs because the per-iteration cost in M-PINN is much higher.

Table 10
Hyper-parameters used in training I-PINNs, M-PINN and XPINNs models for approximating the solution to Eq. (34) pertaining to the 1D problem.

Parameter	I-PINNs	M-PINN	XPINNs
# of neural network	1	2	2
# of hidden layers in each network	2	2	2
# of neurons per hidden layer	20	15	15
# of parameters	481	572	572
# of training points (N_Ω)	53	57	57
α_{int} at interface	20	20	20
α_{bc} at each boundary	20	20	20
Activation function	swish – tanh	tanh	tanh
Iterations	35 000	35 000	35 000
RMSE	4.53×10^{-4}	8.02×10^{-4}	5.73×10^{-1}
Cost	1	1.43	1.56

Table 11
Hyper-parameters used in training I-PINNs, M-PINN and XPINNs models for approximating the solution to Eq. (36).

Parameter	I-PINNs	M-PINN	XPINNs
# of neural network	1	5	5
# of hidden layers in each network	2	2	2
# of neurons per hidden layer	20	10	10
# of parameters	501	755	755
# of training points (N_Ω)	1036	2608	2608
α_{int} at interface	1	1	1
α_{bc} at each boundary	10	10	10
Activation function	sigmoid – swish_mod	sigmoid	sigmoid
Iterations	35 000	35 000	35 000
RMSE	2.2×10^{-6}	2.1×10^{-6}	2.4×10^{-1}
Cost	1	1.8	2.0

4.2. Poisson's equation in 2D

Let us again consider Poisson's equation, now defined in a square computational domain, $\Omega = [0,1] \times [0,1]$. Four subdomains with different material constants are considered within this domain and are defined as, $\Omega_1 = \{\mathbf{x} : (x-0.25)^2 + (y-0.25)^2 \leq 0.05^2\}$, $\Omega_2 = \{\mathbf{x} : (x-0.25)^2 + (y-0.75)^2 \leq 0.2^2\}$, $\Omega_3 = \{\mathbf{x} : (x-0.75)^2 + (y-0.75)^2 \leq 0.15^2\}$, $\Omega_4 = \{\mathbf{x} : (x-0.75)^2 + (y-0.25)^2 \leq 0.1^2\}$. The rest of the domain is defined as, $\Omega_5 = \Omega \setminus (\overline{\Omega_1 \cup \Omega_2 \cup \Omega_3 \cup \Omega_4})$. The equations of the interfaces are

$$
\begin{aligned}
\Gamma^1_{\text{int}} &= \{\mathbf{x} : (x-0.25)^2 + (y-0.25)^2 = 0.05^2\},\\
\Gamma^2_{\text{int}} &= \{\mathbf{x} : (x-0.25)^2 + (y-0.75)^2 = 0.20^2\},\\
\Gamma^3_{\text{int}} &= \{\mathbf{x} : (x-0.75)^2 + (y-0.75)^2 = 0.15^2\},\\
\Gamma^4_{\text{int}} &= \{\mathbf{x} : (x-0.75)^2 + (y-0.25)^2 = 0.10^2\}.
\end{aligned}
$$

The model problem in these five regions is defined as

$$
\begin{aligned}
\nabla \cdot (\kappa_{\text{m}} \nabla u_{\text{m}}) &= 1 && \text{in } \Omega_{\text{m}},\\
u_5 &= \Lambda^{\text{d}} && \text{on } \partial\Omega^{\text{d}}_5,\\
[\![u]\!]_{\text{j}} &= \lambda_{\text{j}} && \text{on } \Gamma^{\text{j}}_{\text{int}},\\
[\![\kappa \nabla u]\!]_{\text{j}} \cdot \mathbf{n_j} &= \chi_{\text{j}} && \text{on } \Gamma^{\text{j}}_{\text{int}},
\end{aligned}
\tag{36}
$$

where $\partial\Omega^{\text{d}}_5 = \{\mathbf{x} : x = 0 \cup x = 1 \cup y = 0 \cup y = 1\}$ is the external boundary with Dirichlet boundary conditions. Considering the material constants, $\kappa_1 = \frac{1}{4}$, $\kappa_2 = \frac{1}{6}$, $\kappa_3 = \frac{1}{10}$, $\kappa_4 = \frac{1}{12}$ and $\kappa_5 = \frac{1}{2}$, the set of Eqs. (36) have a closed-form analytical solution given as

$$
u(\mathbf{x}) = \begin{cases} x^2 + y^2 & \text{in } \Omega_1,\\ 3x^2 + xy & \text{in } \Omega_2,\\ 4x^2 + y^2 & \text{in } \Omega_3,\\ x^2 + 5y^2 & \text{in } \Omega_4,\\ x^2 + xy & \text{in } \Omega_5. \end{cases}
\tag{37}
$$

The known boundary data Λ^{d} at the Dirichlet boundaries and the jumps in primary variable and flux at the interfaces (Γ^j_{int}, j = 1 to 4), λ_{j} and χ_{j} respectively, are prescribed according to Eq. (37).

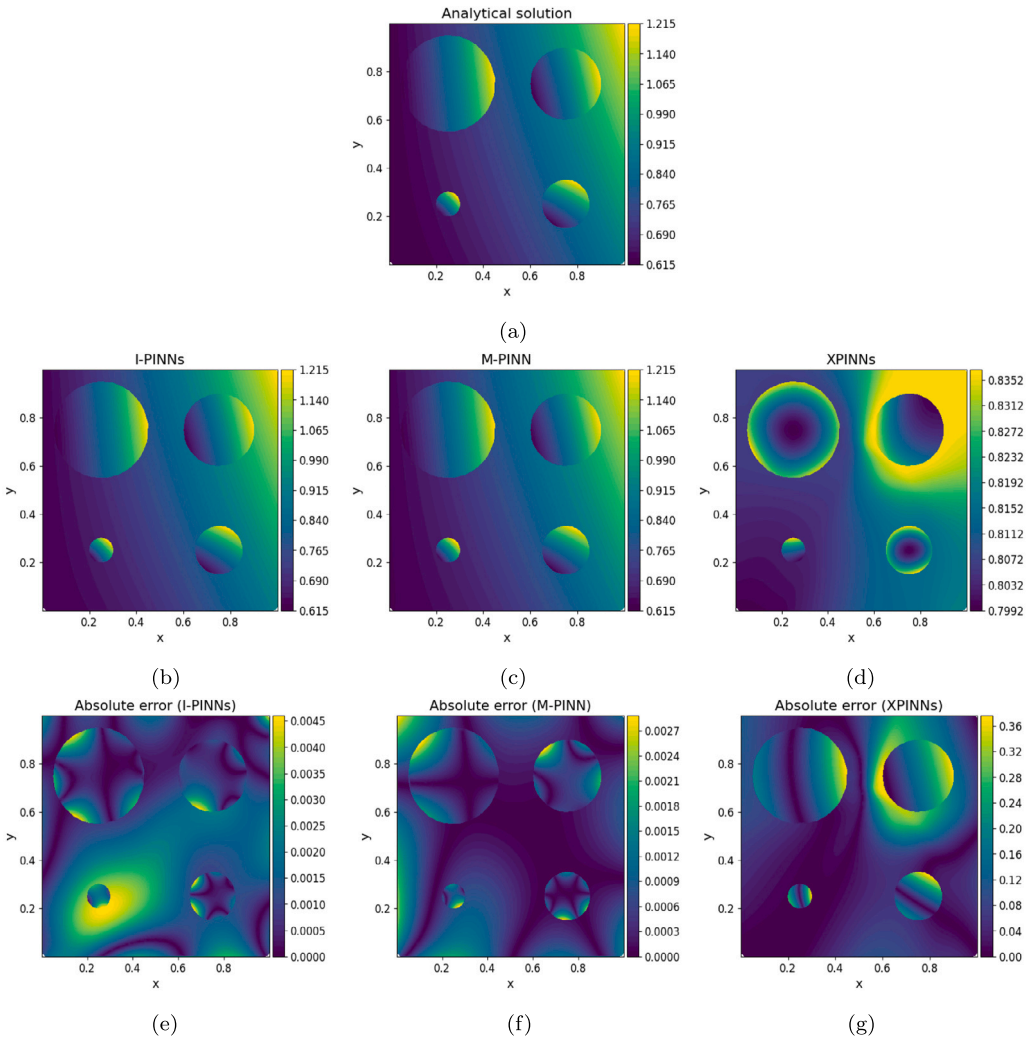

Fig. 23. (a) Contour plot of the analytical solution to Eqs. (36) and approximation to the solution by, (b) I-PINNs, (c) M-PINN, (d) XPINNs and absolute error plot of the approximation obtained with (d) I-PINNs (RMSE = 2.2×10^{-6}, (e) M-PINN (RMSE = 2.1×10^{-6}), (f) XPINNs (RMSE = 2.4×10^{-1}).

The three frameworks, viz., I-PINNs, XPINNs, and, M-PINN are utilized for solving Eqs. (36) with the hyper-parameters listed in Table 11. Contour plots depicting the solution to Eqs. (36), approximated by the three frameworks, are presented in Fig. 23. It is evident that both I-PINNs and M-PINN methods accurately reproduce the exact solution, with RMSEs of both approximations being in the order of 10^{-6} (see Table 11). However, the increased accuracy of the M-PINN is attained at a relatively higher computational cost, as it employs five distinct neural networks with different parameter sets, unlike I-PINNs. Although the individual subdomain neural networks in M-PINN have fewer neurons than I-PINNs, the total number of parameters utilized in M-PINN exceeds that of I-PINNs by as much as 50%. For this particular example, the computational cost of M-PINNs is 80% higher than I-PINNs indicating a direct relationship between the computational cost and the number of trainable parameters. Although these differences may seem minor for simple academic problems, they may become more pronounced when dealing with complex practical problems featuring numerous interfaces and/or coupled field equations. Like the 1D example, XPINNs was the worst performer in terms of both computational cost and accuracy. Despite being twice as expensive as I-PINNs, it could only approximate the solution to an RMSE of $\mathcal{O}(10^{-1})$.

4.3. Poisson's equation in 3D

As a last example, we consider a 3D extension of the problem considered in the previous subsection, and compare the performance of I-PINNs, M-PINN and XPINNs. A cubic computational domain, $\Omega = [0,1] \times [0,1] \times [-1,1]$ is considered. Four subdomains with different material constants are considered which are defined as, $\Omega_1 = \{\mathbf{x} : (x-0.25)^2 + (y-0.25)^2 + z^2 \leq 0.05^2\}$, $\Omega_2 = \{\mathbf{x} : (x-0.25)^2 + (y-0.75)^2 + z^2 \leq 0.2^2\}$, $\Omega_3 = \{\mathbf{x} : (x-0.75)^2 + (y-0.75)^2 + z^2 \leq 0.15^2\}$, $\Omega_4 = \{\mathbf{x} : (x-0.75)^2 + (y-0.25)^2 + z^2 \leq 0.1^2\}$.

Table 12
Hyper-parameters used in training I-PINNs, M-PINN and XPINNs models for approximating the solution to Eq. (38).

Parameter	I-PINNs	M-PINN	XPINNs
# of neural network	1	5	5
# of hidden layers in each network	3	3	3
# of neurons per hidden layer	50	50	50
# of parameters	5351	26 755	26 755
# of training points (N_Ω)	2096	5372	5372
α_{int} at interface	50	50	50
α_{bc} at each boundary	50	50	50
Activation function	swish – tanh	tanh	tanh
Iterations	75 000	75 000	75 000
RMSE	6.5×10^{-3}	4.2×10^{-3}	4.9×10^{-1}
Cost	1	1.6	2.82

The rest of the domain is defined as, $\Omega_5 = \Omega \setminus \overline{(\Omega_1 \cup \Omega_2 \cup \Omega_3 \cup \Omega_4)}$. The equations of the interfaces are

$$\begin{aligned}
\Gamma^1_{\text{int}} &= \{\mathbf{x} : (x-0.25)^2 + (y-0.25)^2 + z^2 = 0.05^2\},\\
\Gamma^2_{\text{int}} &= \{\mathbf{x} : (x-0.25)^2 + (y-0.75)^2 + z^2 = 0.20^2\},\\
\Gamma^3_{\text{int}} &= \{\mathbf{x} : (x-0.75)^2 + (y-0.75)^2 + z^2 = 0.15^2\},\\
\Gamma^4_{\text{int}} &= \{\mathbf{x} : (x-0.75)^2 + (y-0.25)^2 + z^2 = 0.10^2\}.
\end{aligned}$$

The model problem in these five regions is defined as

$$\begin{aligned}
\nabla \cdot (\kappa_{\text{m}} \nabla u_{\text{m}}) &= 1 && \text{in } \Omega_{\text{m}},\\
u_5 &= \Lambda^{\text{d}} && \text{on } \partial\Omega^{\text{d}}_5,\\
[\![u]\!]_{\text{j}} &= \lambda_{\text{j}} && \text{on } \Gamma^{\text{j}}_{\text{int}},\\
[\![\kappa \nabla u]\!]_{\text{j}} \cdot \mathbf{n_j} &= \chi_{\text{j}} && \text{on } \Gamma^{\text{j}}_{\text{int}},
\end{aligned} \tag{38}$$

where $\partial\Omega^{\text{d}}_5 = \{\mathbf{x} : x = 0 \cup x = 1 \cup y = 0 \cup y = 1 \cup z = -1 \cup z = 1\}$ is the external boundary where Dirichlet boundary conditions are specified. Considering the material constants, $\kappa_1 = \frac{1}{6}$, $\kappa_2 = \frac{1}{8}$, $\kappa_3 = \frac{1}{14}$, $\kappa_4 = \frac{1}{18}$ and $\kappa_5 = \frac{1}{4}$, the set of Eqs. (38) have a closed-form analytical solution given as

$$u(\mathbf{x}) = \begin{cases}
x^2 + y^2 + z^2 & \text{in } \Omega_1,\\
3x^2 + 2y + z^2 & \text{in } \Omega_2,\\
4x^2 + y^2 + 2z^2 & \text{in } \Omega_3,\\
x^2 + 5y^2 + 3z^2 & \text{in } \Omega_4,\\
x^2 + xy + z^2 & \text{in } \Omega_5.
\end{cases} \tag{39}$$

The known boundary data Λ^{d} at the Dirichlet boundaries and the jumps in primary variable and flux at the interfaces (Γ^j_{int}, j = 1 to 4), λ_{j} and χ_{j} respectively, are prescribed according to Eq. (39).

I-PINNs, M-PINN, and XPINNs, are employed to solve Eq. (38) using the hyper-parameters outlined in Table 12. Both I-PINNs and M-PINN achieve a reasonable approximation of the solution with an RMSE of $\mathcal{O}(10^{-3})$. However, like the 1D and 2D examples, M-PINN requires 60% more training time than I-PINNs to achieve the same level of accuracy. Similarly, XPINNs again exhibits the worst RMSE among the three frameworks, at $\mathcal{O}(10^{-1})$, with a computational cost 2.82 times higher than I-PINNs. The approximated solutions by all three frameworks, along with their corresponding absolute error plots, are presented in Fig. 24. For better visualization, the domain is clipped across plane $z = 0$ to observe the variation of the solution field in the interior of the domain and along the interfaces. As observed in the absolute error plot, although the errors for I-PINNs and M-PINN are comparable, X-PINNs exhibits large errors.

5. Summary

We present a novel physics-informed neural networks (PINNs) framework for modeling elliptic interface problems with discontinuous coefficients. Through specific numerical examples, we demonstrate the limitations of conventional PINNs to approximate solutions with strong and weak discontinuities. Conventional PINNs fails to capture strong and weak discontinuities at the interface because neural networks construct naturally smooth approximations. To address this limitation, we propose a modified architecture of PINNs, called Interface PINNs (I-PINNs), which involves the use of domain decomposition and different activation functions across an interface.

We present several one-dimensional, two-dimensional, and three-dimensional numerical examples where the conventional PINNs and I-PINNs are compared with respect to their accuracy and cost. It is observed that I-PINNs provide more accurate approximations than conventional PINNs on using identical hyper-parameters. For all examples, the root-mean-square error (RMSE) of the I-PINNs approximations is at least two orders of magnitude better than conventional PINNs with identical hyper-parameters. Even with more

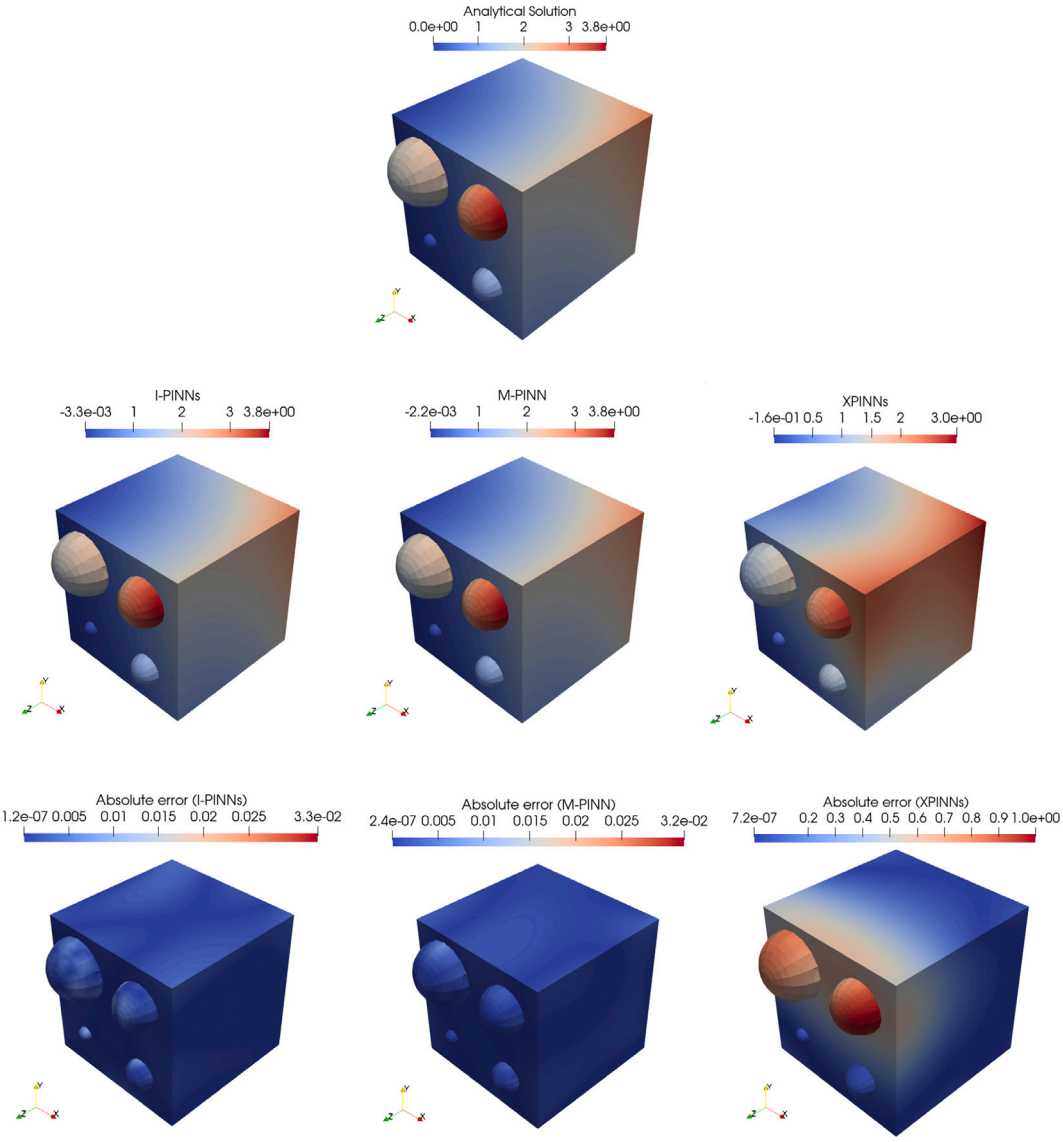

Fig. 24. Contour plots of approximation to the solution to Eq. (38) by the three frameworks – I-PINNs, M-PINN, and XPINNs, as well as their corresponding absolute error plots. The domain is clipped at $z = 0$ to visualize the solution field in the interior of the cuboidal domain and along the interfaces.

complex layer architectures, conventional PINNs fails to provide the same accuracy as I-PINNs despite being almost ten times as expensive in their run-time. Moreover, we conducted a detailed comparison with two commonly used domain-decomposition-based PINNs frameworks, namely M-PINN and XPINNs, for 1D, 2D, and 3D examples with interface jumps. The results demonstrate that I-PINNs, provides comparable accuracy to M-PINN at a significantly lower computational cost (by $\sim$ 40%–80%). XPINNs fared poorly for this class of problems and yielded worse RMSE (by at least two orders of magnitude compared to I-PINNs) despite being approximately twice as computationally expensive. Interestingly, our findings suggest that the key factor for achieving good accuracy for jump problems is ensuring that different activation functions are used across the interface. Going forward, the extension of the proposed method to transient and coupled problems is of interest and is a subject of our future research.

CRediT authorship contribution statement

Antareep Kumar Sarma: Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Formal analysis. **Sumanta Roy:** Writing – review & editing, Visualization, Software, Investigation, Formal analysis. **Chandrasekhar Annavarapu:** Writing – review & editing, Supervision, Software, Resources, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Conceptualization. **Pratanu Roy:** Software, Methodology, Investigation, Formal analysis, Conceptualization. **Shriram Jagannathan:** Methodology, Formal analysis.

Declaration of competing interest

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

Data availability

No data was used for the research described in the article.

Acknowledgments

Chandrasekhar Annavarapu gratefully acknowledges the support from ExxonMobil Corporation, USA to the Subsurface Mechanics and Geo-Energy Laboratory under the grant SP22230020CEEXXU008957. The support from the Ministry of Education, Government of India and IIT Madras under the grant SB20210856CEMHRD008957 is also gratefully acknowledged. Pratanu Roy's contribution was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. The authors would like to thank Dr. Dakshina Murthy Valiveti and Dr. Xiao-Hui Wu from ExxonMobil Corporation for helpful discussions.

References

[1] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, J. Comput. Phys. 378 (2019) 686–707.
[2] M. Vahab, E. Haghighat, M. Khaleghi, N. Khalili, A physics-informed neural network approach to solution and identification of biharmonic equations of elasticity, J. Eng. Mech. 148 (2) (2022) 04021154.
[3] Y. Gu, C. Zhang, M.V. Golub, Physics-informed neural networks for analysis of 2D thin-walled structures, Eng. Anal. Bound. Elem. 145 (2022) 161–172.
[4] E. Zhang, M. Dao, G.E. Karniadakis, S. Suresh, Analyses of internal structures and defects in materials using physics-informed neural networks, Sci. Adv. 8 (7) (2022) eabk0644.
[5] S. Goswami, C. Anitescu, S. Chakraborty, T. Rabczuk, Transfer learning enhanced physics informed neural network for phase-field modeling of fracture, Theor. Appl. Fract. Mech. 106 (2020) 102447.
[6] Y. Ghaffari Motlagh, P.K. Jimack, R. de Borst, Deep learning phase-field model for brittle fractures, Internat. J. Numer. Methods Engrg. 124 (3) (2022) 543–780.
[7] X. Jin, S. Cai, H. Li, G.E. Karniadakis, NSFnets (Navier-Stokes flow nets): Physics-informed neural networks for the incompressible Navier-Stokes equations, J. Comput. Phys. 426 (2021) 109951.
[8] C. Rao, H. Sun, Y. Liu, Physics-informed deep learning for incompressible laminar flows, Theor. Appl. Mech. Lett. 10 (3) (2020) 207–212.
[9] A. Kumar, S. Ridha, M. Narahari, S.U. Ilyas, Physics-guided deep neural network to characterize non-Newtonian fluid flow for optimal use of energy resources, Expert Syst. Appl. 183 (2021) 115409.
[10] J.M. Hanna, J.V. Aguado, S. Comas-Cardona, R. Askri, D. Borzacchiello, Residual-based adaptivity for two-phase flow simulation in porous media using physics-informed neural networks, Comput. Methods Appl. Mech. Engrg. 396 (2022) 115100.
[11] M.M. Almajid, M.O. Abu-Al-Saud, Prediction of porous media fluid flow using physics informed neural networks, J. Pet. Sci. Eng. 208 (2022) 109205.
[12] H. Du, Z. Zhao, H. Cheng, J. Yan, Q. He, Modeling density-driven flow in porous media by physics-informed neural networks for CO2 sequestration, Comput. Geotech. 159 (2023) 105433.
[13] S. Cai, Z. Wang, S. Wang, P. Perdikaris, G.E. Karniadakis, Physics-informed neural networks for heat transfer problems, J. Heat Transfer 143 (6) (2021).
[14] X. Liu, W. Peng, Z. Gong, W. Zhou, W. Yao, Temperature field inversion of heat-source systems via physics-informed neural networks, Eng. Appl. Artif. Intell. 113 (2022) 104902.
[15] E.R. Kral, K. Komvopoulos, Three-dimensional finite element analysis of subsurface stress and strain fields due to sliding contact on an elastic-plastic layered medium, J. Tribol. 119 (2) (1997) 332–341, http://dx.doi.org/10.1115/1.2833223.
[16] H. Pu, P.J. Fox, Model for coupled large strain consolidation and solute transport in layered soils, Int. J. Geomech. 16 (2) (2016) 04015064.
[17] P.J. Fox, H.-F. Pu, J.D. Berles, CS3: Large strain consolidation model for layered soils, J. Geotech. Geoenviron. Eng. 140 (8) (2014) 04014041.
[18] R.R. Settgast, O.Y. Vorobiev, J.P. Morris, E.B. Herbold, M.A. Homel, C. Annavarapu, Modeling of fracture opening by explosive products, in: U.S. Rock Mechanics/Geomechanics Symposium, 2017, pp. ARMA–2017–1048, Vol. All Days.
[19] R.R. Settgast, P. Fu, S.D. Walsh, J.A. White, C. Annavarapu, F.J. Ryerson, A fully coupled method for massively parallel simulation of hydraulically driven fractures in 3-dimensions, Int. J. Numer. Anal. Methods Geomech. 41 (5) (2017) 627–653, http://dx.doi.org/10.1002/nag.2557.
[20] D. Vogler, R.R. Settgast, C. Annavarapu, C. Madonna, P. Bayer, F. Amann, Experiments and simulations of fully hydro-mechanically coupled response of rough fractures exposed to high-pressure fluid injection, J. Geophys. Res.: Solid Earth 123 (2) (2018) 1186–1200, http://dx.doi.org/10.1002/2017JB015057.
[21] D.M. Valiveti, C.A. Srinivas, V. Dyadechko, Grid modification during simulated fracture propagation. Google Patents, US Patent 11,608,730, 2023.
[22] C.A. Srinivas, D.M. Valiveti, V. Dyadechko, Model for coupled porous flow and geomechanics for subsurface simulation. Google Patents, US Patent App. 17/011, 254, 2021.
[23] N. Mehra, L. Mu, T. Ji, X. Yang, J. Kong, J. Gu, J. Zhu, Thermal transport in polymeric materials and across composite interfaces, Appl. Mater. Today 12 (2018) 92–130.
[24] F. Erdogan, G. Gupta, Layered composites with an interface flaw, Int. J. Solids Struct. 7 (8) (1971) 1089–1107.
[25] S. Khan, A. Muixí, C. Annavarapu, A. Rodríguez-Ferran, Adaptive phase-field modeling of fracture propagation in bi-layered materials, Eng. Fract. Mech. 292 (2023) 109650.
[26] S. Khan, C. Annavarapu, A. Rodríguez-Ferran, Investigation on the effect of material mismatch between two dissimilar materials using an adaptive phase-field method, Int. J. Adv. Eng. Sci. Appl. Math. 15 (4) (2023) 173–186, http://dx.doi.org/10.1007/s12572-023-00347-2.
[27] I. Jain, A. Muixí, C. Annavarapu, S.S. Mulay, A. Rodríguez-Ferran, Adaptive phase-field modeling of fracture in orthotropic composites, Eng. Fract. Mech. 292 (2023) 109673.
[28] I. Jain, C. Annavarapu, S.S. Mulay, A. Rodríguez-Ferran, Numerical modeling of fracture propagation in orthotropic composite materials using an adaptive phase-field method, Int. J. Adv. Eng. Sci. Appl. Math. 15 (4) (2023) 144–154, http://dx.doi.org/10.1007/s12572-023-00331-w.
[29] S. Khan, A. Muixí, C. Annavarapu, A. Rodríguez-Ferran, Phase field modeling of crack growth in layered media, SSRN (2024) http://dx.doi.org/10.2139/ssrn.4783632.

[30] D. Medlin, G. Snyder, Interfaces in bulk thermoelectric materials: a review for current opinion in colloid and interface science, Curr. Opin. Colloid & Interface Sci. 14 (4) (2009) 226–235.

[31] R. Cammarata, Surface and interface stress effects on interfacial and nanostructured materials, Mater. Sci. Eng. A 237 (2) (1997) 180–184.

[32] Z. Suo, J.W. Hutchinson, Sandwich test specimens for measuring interface crack toughness, Mater. Sci. Eng. A 107 (1989) 135–143.

[33] J.L. Hellrung, L. Wang, E. Sifakis, J.M. Teran, A second order virtual node method for elliptic problems with interfaces and irregular domains in three dimensions, J. Comput. Phys. 231 (4) (2012) 2015–2048, http://dx.doi.org/10.1016/j.jcp.2011.11.023.

[34] I.-L. Chern, Y.-C. Shu, A coupling interface method for elliptic interface problems, J. Comput. Phys. 225 (2) (2007) 2138–2174, http://dx.doi.org/10.1016/j.jcp.2007.03.012.

[35] K. Pan, Y. Tan, H. Hu, An interpolation matched interface and boundary method for elliptic interface problems, J. Comput. Appl. Math. 234 (1) (2010) 73–94, http://dx.doi.org/10.1016/j.cam.2009.12.006.

[36] C. Annavarapu, M. Hautefeuille, J.E. Dolbow, A robust Nitscheś formulation for interface problems, Comput. Methods Appl. Mech. Engrg. 225 (2012) 44–54.

[37] M. Hautefeuille, C. Annavarapu, J.E. Dolbow, Robust imposition of Dirichlet boundary conditions on embedded surfaces, Internat. J. Numer. Methods Engrg. 90 (1) (2012) 40–64.

[38] C. Annavarapu, M. Hautefeuille, J.E. Dolbow, Stable imposition of stiff constraints in explicit dynamics for embedded finite element methods, Internat. J. Numer. Methods Engrg. 92 (2) (2012) 206–228.

[39] W. Jiang, Y. Liu, C. Annavarapu, A weighted Nitsche's method for interface problems with higher-order simplex elements, Comput. Mech. 69 (5) (2022) 1115–1129.

[40] W. Jiang, C. Annavarapu, J.E. Dolbow, I. Harari, A robust Nitsche's formulation for interface problems with spline-based finite elements, Internat. J. Numer. Methods Engrg. 104 (7) (2015) 676–696.

[41] C. Annavarapu, M. Hautefeuille, J.E. Dolbow, A nitsche stabilized finite element method for frictional sliding on embedded interfaces. Part II: Intersecting interfaces, Comput. Methods Appl. Mech. Engrg. 267 (2013) 318–341, http://dx.doi.org/10.1016/j.cma.2013.08.008.

[42] C. Annavarapu, M. Hautefeuille, J.E. Dolbow, A nitsche stabilized finite element method for frictional sliding on embedded interfaces. Part I: Single interface, Comput. Methods Appl. Mech. Engrg. 268 (2014) 417–436, http://dx.doi.org/10.1016/j.cma.2013.09.002.

[43] C. Annavarapu, An efficient finite element method for interface problems (Ph.D. thesis), Duke University, 2013.

[44] Z. Zhang, X. Yan, P. Liu, K. Zhang, R. Han, S. Wang, A physics-informed convolutional neural network for the simulation and prediction of two-phase Darcy flows in heterogeneous porous media, J. Comput. Phys. 477 (2023) 111919.

[45] Z. Zhang, A physics-informed deep convolutional neural network for simulating and predicting transient Darcy flows in heterogeneous reservoirs without labeled data, J. Pet. Sci. Eng. 211 (2022) 110179.

[46] M. Aliakbari, M. Soltany Sadrabadi, P. Vadasz, A. Arzani, Ensemble physics informed neural networks: A framework to improve inverse transport modeling in heterogeneous domains, Phys. Fluids 35 (5) (2023).

[47] A. Alhubail, M. Fahs, F. Lehmann, H. Hoteit, Physics-informed neural networks for modeling flow in heterogeneous porous media: A decoupled pressure-velocity approach, in: International Petroleum Technology Conference, IPTC, 2024, D021S084R002.

[48] S. Rezaei, A. Harandi, A. Moeineddin, B.-X. Xu, S. Reese, A mixed formulation for physics-informed neural networks as a potential solver for engineering problems in heterogeneous domains: Comparison with finite element method, Comput. Methods Appl. Mech. Engrg. 401 (2022) 115616.

[49] F. Lehmann, M. Fahs, A. Alhubail, H. Hoteit, A mixed pressure-velocity formulation to model flow in heterogeneous porous media with physics-informed neural networks, Adv. Water Resour. 181 (2023) 104564.

[50] A. Harandi, A. Moeineddin, M. Kaliske, S. Reese, S. Rezaei, Mixed formulation of physics-informed neural networks for thermo-mechanically coupled systems and heterogeneous domains, Internat. J. Numer. Methods Engrg. 125 (4) (2024) e7388.

[51] A. Henkes, H. Wessels, R. Mahnken, Physics informed neural networks for continuum micromechanics, Comput. Methods Appl. Mech. Engrg. 393 (2022) 114790.

[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. Engrg. 365 (2020) 113028.

[53] A.D. Jagtap, G.E. Karniadakis, Extended physics-informed neural networks (XPINNs): A generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations, in: AAAI Spring Symposium: MLPS, 2021, pp. 2002–2041.

[54] T. Bandai, T.A. Ghezzehei, Forward and inverse modeling of water flow in unsaturated soils with discontinuous hydraulic conductivities using physics-informed neural networks with domain decomposition, Hydrol. Earth Syst. Sci. 26 (16) (2022) 4469–4495.

[55] A. Alhubail, X. He, M. AlSinan, H. Kwak, H. Hoteit, Extended physics-informed neural networks for solving fluid flow problems in highly heterogeneous media, in: International Petroleum Technology Conference, IPTC, 2022, D031S073R001.

[56] B. Zhang, G. Wu, Y. Gu, X. Wang, F. Wang, Multi-domain physics-informed neural network for solving forward and inverse problems of steady-state heat conduction in multilayer media, Phys. Fluids 34 (11) (2022).

[57] A.G. Baydin, B.A. Pearlmutter, A.A. Radul, J.M. Siskind, Automatic differentiation in machine learning: a survey, J. Mach. Learn. Res. 18 (2018) 1–43.

[58] J. Bradbury, R. Frostig, P. Hawkins, M.J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, Q. Zhang, JAX: composable transformations of Python+NumPy programs, 2018.

[59] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, 2014, arXiv preprint arXiv:1412.6980.

[60] T. De Ryck, F. Bonnet, S. Mishra, E. de Bézenac, An operator preconditioning perspective on training in physics-informed machine learning, 2023, arXiv preprint arXiv:2310.05801.

[61] P. Roy, S. Castonguay, Exact enforcement of temporal continuity in sequential physics-informed neural networks, 2024, arXiv preprint arXiv:2403.03223.

[62] R.R. Selmic, F.L. Lewis, Neural-network approximation of piecewise continuous functions: application to friction compensation, IEEE Trans. Neural Netw. 13 (3) (2002) 745–751.

[63] B. Llanas, S. Lantarón, F.J. Sáinz, Constructive approximation of discontinuous functions by neural networks, Neural Process. Lett. 27 (2008) 209–226.

[64] V.E. Ismailov, A three layer neural network can represent any multivariate function, J. Math. Anal. Appl. 523 (1) (2023) 127096.

[65] A. Sarma, C. Annavarapu, P. Roy, S. Jagannathan, D. Valiveti, Variational interface physics informed neural networks (VI-PINNs) for heterogeneous subsurface systems, in: ARMA US Rock Mechanics/Geomechanics Symposium, ARMA, 2023, pp. ARMA–2023.

[66] E. Bisong, Google colaboratory, in: Building Machine Learning and Deep Learning Models on Google Cloud Platform: A Comprehensive Guide for Beginners, A Press, Berkeley, CA, 2019, pp. 59–64, http://dx.doi.org/10.1007/978-1-4842-4470-8_7.

[67] M. Xu, Z. Zhao, J. Niu, L. Guo, Y. Lin, A simplified reproducing kernel method for 1-D elliptic type interface problems, J. Comput. Appl. Math. 351 (2019) 29–40.