Available online at www.sciencedirect.com
ScienceDirect
Comput. Methods Appl. Mech. Engrg. 374 (2021) 113547
Computer methods in applied mechanics and engineering
www.elsevier.com/locate/cma

hp-VPINNs: Variational physics-informed neural networks with domain decomposition

Ehsan Kharazmi[a,*], Zhongqiang Zhang[b], George E.M. Karniadakis[a,c]

[a] *Division of Applied Mathematics, Brown University, 170 Hope St, Providence, RI 02906, USA*
[b] *Department of Mathematical Sciences, Worcester Polytechnic Institute, 100 Institute Rd, Worcester, MA 01609, USA*
[c] *Pacific Northwest National Laboratory, Richland, WA 99354, USA*

Received 9 March 2020; received in revised form 25 October 2020; accepted 26 October 2020
Available online 4 December 2020

Abstract

We formulate a general framework for *hp*-variational physics-informed neural networks (*hp*-VPINNs) based on the nonlinear approximation of shallow and deep neural networks and *hp*-refinement via domain decomposition and projection onto the space of high-order polynomials. The *trial space* is the space of neural network, which is defined globally over the entire computational domain, while the *test space* contains piecewise polynomials. Specifically in this study, the *hp*-refinement corresponds to a *global approximation with a local learning* algorithm that can efficiently localize the network parameter optimization. We demonstrate the advantages of *hp*-VPINNs in both accuracy and training cost for several numerical examples of function approximation and in solving differential equations.

Keywords: Physics-informed learning; VPINNs; Variational neural network; Domain decomposition; Automatic differentiation; *hp*-refinement; Partial differential equations

1. Introduction

Neural networks (NN) have gained a lot of attention more recently in solving differential equations, see e.g. [1–8]. They offer a nonlinear approximant via the composition of hidden layers in a variety of network structures and activation functions, and their universal approximation properties provide an alternative approach for solving differential equations. In general, the nonlinear approximation [9,10] extends the approximants to reside in a nonlinear space and does not limit the approximation to linear spaces; it contains different approaches such as wavelet analysis [11], dictionary learning [12], adaptive pursuit and compressed sensing [13–16], adaptive splines [9], radial basis functions [17], Gaussian kernels [18], and neural networks [19–21].

Due to the nature of nonlinear approximations of neural networks, solving differential equations using NNs is formulated as optimization problems where it is crucial to design appropriate loss functions to optimize the quantities of interests. Based on the method of variational/weighted residuals [22], several solvers have been developed, such as deep Galerkin method (DGM) [23] based on the least squares, physics-informed neural networks (PINNs) [3,24]

* Corresponding author.
E-mail address: ehsan_kharazmi@brown.edu (E. Kharazmi).

https://doi.org/10.1016/j.cma.2020.113547
0045-7825/

based on the collocation methods, and variational physics-informed neural networks (VPINNs) [25,26] based on the Galerkin method. Along this path, we develop a method which is called *hp*-Variational Physics Informed Neural Networks (*hp*-VPINNs) based on the sub-domain Petrov–Galerkin method. The neural network still serves as the trial space but, compared to all the aforementioned works, the sub-domain Petrov–Galerkin methods allow *hp*-refinement via domain decomposition as *h*-refinement and projection onto space of high order polynomials as *p*-refinement.

In this work, we consider the following problem

$$\mathcal{L}^{\mathbf{q}}u(\mathbf{x},t) = f(\mathbf{x},t), \quad (\mathbf{x},t) \in \Omega \times (0,T], \tag{1.1}$$

$$u(\mathbf{x},t) = h(\mathbf{x},t), \quad (\mathbf{x},t) \in \partial\Omega \times [0,T], \quad u(\mathbf{x},0) = g(\mathbf{x}), \quad \mathbf{x} \in \Omega, \tag{1.2}$$

where the bounded domain $\Omega \subset \mathbb{R}^d$ with boundaries $\partial\Omega$, $T > 0$, and $u(\mathbf{x},t) : \Omega \times [0,T] \to \mathbb{R}$ describes the underlying physical phenomena modeled by the above governing equation. The operator $\mathcal{L}^{\mathbf{q}}$ is usually comprised of the identity and the differential operators with some parameters $\mathbf{q}$. We assume that $\tilde{u}(\mathbf{x},t;\mathbf{W},\mathbf{b})$ is a NN approximation (trial solution) of $u(\mathbf{x},t)$ in (1.1)–(1.2). Specifically, the NN is comprised of ℓ hidden layers with $\mathcal{N}_i$ neurons in each layer and activation function σ that takes the following form

$$u_{NN}(\mathbf{x},t;\mathbf{W},\mathbf{b}) = \mathcal{G} \circ T^{(\ell)} \circ T^{(\ell-1)} \circ \cdots \circ T^{(1)}(\mathbf{x}). \tag{1.3}$$

In the output layer, the linear mapping is $\mathcal{G} : \mathbb{R}^{\mathcal{N}_\ell} \to \mathbb{R}$, and in each hidden layer $i = 1, 2, \ldots, \ell$, the nonlinear mapping is $T^{(i)}(\cdot) = \sigma(\mathbf{W}_i \times \cdot + \mathbf{b}_i)$ with weights $\mathbf{W}_i \in \mathbb{R}^{\mathcal{N}_i \times \mathcal{N}_{i-1}}$ and biases $\mathbf{b}_i \in \mathbb{R}^{\mathcal{N}_i}$, where $\mathcal{N}_0 = d$ is the input dimension. Then, we define the *strong-form residual* $r(\tilde{u})$, the boundary residual $r_b(\tilde{u})$, and the initial residual $r_0(\tilde{u})$ as

$$\begin{aligned} r(\tilde{u}) &= \mathcal{L}^{\mathbf{q}}\tilde{u} - f, \quad \forall (\mathbf{x},t) \in \ \Omega \times (0,T], \\ r_b(\tilde{u}) &= \tilde{u} - h, \quad \forall (\mathbf{x},t) \in \partial\Omega \times [0,T], \\ r_0(\tilde{u}) &= \tilde{u} - g, \quad \forall (\mathbf{x},t) \in \ \Omega \times \{t = 0\}. \end{aligned} \tag{1.4}$$

The residuals are measures to the extent to which the approximation $\tilde{u}$ satisfies Eqs. (1.1)–(1.2). Ideally, the exact solution is recovered when all the residuals are identically zero. The weighted integrals of the residuals are obtained by projecting them onto a properly chosen space of test (weighting) functions V and then set to zero; this leads to the variational form of the problem. Specifically, we choose some test functions v_j such that

$$\begin{aligned} \mathcal{R}_j(\tilde{u}) &= \int_{\Omega\times(0,T]} r(\tilde{u}) v_j \, dx \, dt = 0, \\ \mathcal{R}_{b,j}(\tilde{u}) &= \int_{\partial\Omega\times(0,T]} r_b(\tilde{u}) v_j \, dx \, dt = 0, \\ \mathcal{R}_{0,j}(\tilde{u}) &= \int_{\Omega} r_0(\tilde{u}) v_j \, dx = 0. \end{aligned} \tag{1.5}$$

To solve the nonlinear system resulting from these equations, we formulate it as the following minimization problem:

$$\min_{\mathbf{W},\mathbf{b}} \mathcal{J}(\tilde{u}, v), \tag{1.6}$$

where

$$\mathcal{J}(\tilde{u}, v) = \tau \sum_{j=1}^{N_r} \mathcal{R}_j^2(\tilde{u}) + \tau_b \sum_{j=1}^{N_b} \mathcal{R}_{b,j}^2(\tilde{u}) + \tau_0 \sum_{j=1}^{N_0} \mathcal{R}_{0,j}^2(\tilde{u}). \tag{1.7}$$

The parameters $\{\tau, \tau_b, \tau_0\}$ denote the weight coefficients in the loss function. They may be user-specified or tuned manually or automatically, e.g., in practice based on the numerical experiment in each problem; their optimal bound, however, is still an open problem in the literature [27].

Different choices of trial function $\tilde{u}$ and test function v_j in (1.7) correspond to various numerical methods. Most of these methods are well established and analyzed in the literature when linear approximations are used. Here, we focus on the nonlinear approximation of $\tilde{u}$ and various choices of test functions, and briefly discuss some choices of test functions when the trial function is shallow/deep NN; see Table 1 for comparison.

Table 1
Various numerical methods based on different approximation and test functions.

Test function	Trial function	
v	$\tilde{u}$	$\tilde{u}$ =DNNs
Delta Dirac	collocation	PINNs [3]
$v = r(\tilde{u})$	Least square	DGM [23]
Polynomials (global)	Petrov–Galerkin	VPINNs [25]
Polynomials (piece wise)	Petrov–Galerkin	VarNet [26]
Nonoverlapping (2.1)	Sub-domain Petrov–Galerkin	*hp*-VPINNs

The *Dirac delta test functions*, $v(\mathbf{x}, t) = \delta(\mathbf{x} - \mathbf{x}_r)\delta(t - t_r)$, correspond to the collocation method. These test functions project the residuals onto a finite set of collocation points, making the equation to be satisfied at these points. The collocation formulation is used in [1] and PINNs [3]. The PINN formulation has been recently successfully employed in many physical problems such as discovering turbulence models from scattered/noisy measurements [28], high speed flows [4], stochastic differential equation by generative adversarial networks [24], fractional differential equations [29], and adaptive activation functions [30,31]. A Python package, namely SciANN, has been also developed based on this formulation [32]. Specifically, PINNs use the following functional

$$L^{\mathfrak{s}} = \frac{1}{N_r}\sum_{i=1}^{N_r}\left|r(\mathbf{x}_r^i, t_r^i)\right|^2 + \tau_b\frac{1}{N_b}\sum_{i=1}^{N_b}\left|r_b(\mathbf{x}_b^i, t_b^i)\right|^2 + \tau_0\frac{1}{N_0}\sum_{i=1}^{N_0}\left|r_0(\mathbf{x}_0^i)\right|^2, \tag{1.8}$$

where the residuals r, r_b and r_0 are given in (1.4) and $\{(\mathbf{x}_r^i, t_r^i)\}_{i=1}^{N_r}$, $\{(\mathbf{x}_b^i, t_b^i)\}_{i=1}^{N_b}$ and $\{\mathbf{x}_r^i\}_{i=1}^{N_0}$ are collocation points in their domains. We use the superscript $\mathfrak{s}$ to refer to the loss function associated with the strong-form of the residual. The *deep Galerkin method* [23,33] also employs the nonlinear approximation of NNs, however, it takes the test functions to be $v = r(\tilde{u})$ and essentially forms a least square method. Other formulations include the deep Ritz method [2] and its extension to deep Nitsche method [7] with essential boundary conditions. Weak adversarial network [34,35] converts the problem into an operator norm minimization problem induced from the weak formulation and consider the test function to be a deep adversarial network with separate parameters.

The variational formulation of PINNs, namely VPINNs [25], takes the nonlinear approximation of DNN as the approximation function. It projects the residuals onto the space of polynomials and thus forms a Petrov–Galerkin method. It has been shown in [25] that in VPINNs the *variational residuals* can be obtained analytically for the case of shallow networks. Specifically, the VPINN formulation uses the following functional

$$L^{\mathfrak{v}} = \frac{1}{K}\sum_{j=1}^{K}\left|\mathcal{R}_j\right|^2 + \tau_b\frac{1}{N_b}\sum_{i=1}^{N_b}\left|r_b(\mathbf{x}_b^i, t_b^i)\right|^2 + \tau_0\frac{1}{N_0}\sum_{i=1}^{N_0}\left|r_0(\mathbf{x}_0^i)\right|^2, \tag{1.9}$$

that takes the test functions v_j $(1 \le j \le K)$ from orthogonal polynomials. The superscript $\mathfrak{v}$ refers to the loss function associated with the variational-form of the residual. Other formulations based on the variational form of the problem have been developed. VarNet [26,36] takes the test functions to be the piece-wise linear shape functions of finite element method and D3M [8] formulation includes the reformulation of problem (1.1)–(1.2) into a system of first-order equations.

In this paper, we develop the *hp*-VPINN method to unify the current developments in deep learning methods for partial differential equations based on residuals of equations using least-squares. In particular, we focus on the variational formulation by taking a different set of test functions, which are non-overlapping on each sub-domain; see the next section for more details. Our formulation leads to reducing the order of differential operators via integration-by-parts, and thus can work more efficiently with rough solutions/input data such as singularities, steep solution, and sharp changes; see Section 5 for an example with a corner singularity. It also provides the flexibility of domain decomposition by *hp*-refinement, where the choice of the current refinements is based on the point-wise values of the residuals. Therefore, it constructs both local and global approximations; see Table 2. We also show in Table 1 that different choices of test functions lead to different methods. We note that the optimal choice of test functions in the current developments is an important open question and requires further analysis. We place our focus here in this paper on the basis of piecewise high-order polynomials. Moreover, we discuss how the current

Table 2
Local/Global trial function versus local/global test function.

	Local trial functions	Global trial functions
Local test functions	Conservative VPINNs [37]	*hp*-VPINNs
Global test functions	–	VPINNs [25] varNet [26], D3M [8]

formulation leads to the PINN formulation and how it relaxes the smoothness requirements on the input data; see Section 2.

The rest of the paper is organized as follows: In Section 2, we present the construction of *hp*-VPINNs. Then, we examine the efficiency of the proposed method in approximating several functions in Section 3. In Sections 4 and 5, we present the details of calculations for elliptic problems in one- and two-dimensions. In Section 6, we show how the proposed method is modified to solve an inverse problem using a linear advection–diffusion equation. The codes and some of the examples presented in this paper are open-source and available at https://github.com/ehsankharazmi/hp-VPINNs.

2. *hp* -Variational Physics-Informed Neural Network (*hp*-VPINN)

The *hp*-VPINN formulation is based on the following localized test functions, defined over nonoverlapping sub-domains $V_k,\ k = 1, 2, \ldots, N_{sd}$ of partition of the set V ($\Omega \times (0, T]$ or Ω in this work). The test function defined on a subset $V_k \subset V$ reads

$$v_k = \begin{cases} \bar{v} \neq 0, & \text{over } V_k, \\ 0, & \text{over } V_k^c, \end{cases} \qquad V_k \cup V_k^c = V, \tag{2.1}$$

that leads to a sub-domain method. The non-vanishing test function $\bar{v}$ is a polynomial of order to be chosen in practice.

We define the *elemental variational residual* as

$$\mathcal{R}^{(e)} = \left(\mathcal{L}^{\mathfrak{q}} u_{NN} - f, v\right)_{\Omega_e \times \Gamma_e}, \tag{2.2}$$

which is enforced for the admissible local test function within element e. Subsequently, we define the *variational loss function* as

$$L^{\mathfrak{v}} = \sum_{e=1}^{N_{el}} \frac{1}{K^{(e)}} \sum_{k=1}^{K^{(e)}} \left|\mathcal{R}_k^{(e)}\right|^2 + \tau_b \frac{1}{N_b} \sum_{i=1}^{N_b} \left|r_b(\mathbf{x}_b^i, t_b^i)\right|^2 + \tau_0 \frac{1}{N_0} \sum_{i=1}^{N_0} \left|r_0(\mathbf{x}_0^i)\right|^2, \tag{2.3}$$

where $K^{(e)}$ is the total number of test functions in element e, the term $\mathcal{R}_k^{(e)}$ is the kth entry of the corresponding tensor associated with element e, and r_b and r_0 have the same form as in (1.4). We refer to Sections 4 and 5 for detailed derivation of *hp*-VPINNs for one- and two-dimensional problems, respectively.

The projection of strong-form residuals onto test functions additionally adds two major truncation and numerical integration errors into the existing approximation and generalization errors of DNNs. Increasing the number of test functions in order to eliminate the truncation error may further complicate the loss function and thus increase the chance of optimization failure in practice. In the case of shallow networks, the variational residual is obtained analytically [25] that completely removes the numerical integration error. However, the compositional structure of hidden layers in DNNs makes it almost impossible to analytically compute the integrals in the variational loss function. Hence, we need to employ proper numerical integration techniques in the case of deep networks, which opens up new problems on developing and analyzing numerical integration methods for functions represented by DNNs. In this work, we adopt the Gauss quadrature rules. To avoid the curse of dimensionality in high-dimensional problems, we can employ numerical approaches such as quasi-Monte Carlo integration [38] or sparse grid quadratures [39,40].

Remark 2.1. Domain decomposition provides the opportunity to employ individual networks in each sub-domain and then assign optimization to a specific computer node. The overall solution is then reconstructed by stitching the solution by individual networks in each sub-domain. This parallelization strategy can drastically decrease the

total computational cost, however, it requires a special treatment at the interfaces, i.e. sub-domains boundaries, such that all individual networks are properly communicating to optimize the total loss function; see [37]. We note that, however, as in the current *hp*-VPINN formulation, even though we decompose the domain into several sub-domain, we still employ a single DNN to approximate the solution over the entire computational domain. In this setting, we avoid the burden of interface treatment while the parallelization may not be trivial as we only have a single loss function associated with DNN.

Remark 2.2. The activation function σ has similar forms in (1.3) for each neuron. However, it may also have different domain and image dimensionality based on the structure of network [30,31]. An adaptive basis viewpoint of DNNs is also given in [41].

Now we can discuss *the connection of this formulation with PINN and other formulations*. In fact, if we take the test functions to be piecewise constants, e.g., $c_{i,j}$ on the domain $\Omega_j \times (t_i, t_{i+1}]$, we have in (1.5),

$$\mathcal{R}_j(\tilde{u}) = c_{i,j} \int_{\Omega_j \times (t_i, t_{i+1}]} r(\tilde{u})\, dx\, dt = 0. \tag{2.4}$$

When the residual is continuous over the sub-domain, there exists one point (x_j^*, t_i^*) in the same sub-domain such that $\mathcal{R}_j(\tilde{u}) = c_{i,j} r(\tilde{u})(x_j^*, t_i^*) = 0$. Similar interpretations apply to the other two formulas in (1.5). We are then led exactly to the PINN formulation. Moreover, when the network residual $r(u_{NN})$ is not continuous, the formulation is still well-defined as long as the integrals are well-defined. For example, when we take the ReLU activation function ($\sigma(x) = \max(0, x)$), $r(u_{NN})$ is only in L^2 for Poisson equations. Thus, we cannot use the PINN formulation, however, we can still use the current formulation by performing integration-by-parts or simply using the averages over the sub-domains. In this case, *we may think of the current formulation as the weak formulation of PINN.*

The *hp*-VPINN seems similar to the VarNet in [26]. However, in our formulation we impose orthogonality while VarNet does not. In the current formulation, the elements in the basis over one subdomain are orthogonal to the elements of the basis in another sudomain while no orthogonality is imposed in VarNet. Further, if we take the orthogonal basis elements in one subdomain, we then have a complete orthogonal basis, which allows us to have high-order polynomials for smooth solutions over smooth subdomains. In VarNets, only piecewise polynomials of low orders are used. VarNet can be seen as a version of our previous work VPINN [25]. We expect that the orthogonality will lead to some convenience in future analysis and design of adaptive strategies.

3. Variational neural networks (VNNs) for function approximation

Let us consider the problem of approximating the target function $u(\mathbf{x}) : \Omega \to \mathbb{R}$ by $u_{NN}(\mathbf{x})$. We define the approximation residual as $r(\mathbf{x}) = u(\mathbf{x}) - u_{NN}(\mathbf{x})$. The setup can be viewed as follows: We let w_b and w_0 be zero and thus define the corresponding loss function

$$\mathrm{L}^{\mathfrak{v}} = \sum_{e=1}^{N_{el}} \frac{1}{K^{(e)}} \sum_{k=1}^{K^{(e)}} \left| \mathcal{R}_k^{(e)} \right|^2, \quad \mathcal{R}_k^{(e)} = \int_{\Omega_e} \left(u_{NN}(\mathbf{x}) - u(\mathbf{x})\right)\, v_k^{(e)}(\mathbf{x})\, d\Omega_e, \tag{3.1}$$

where $K^{(e)}$ is the number of test functions employed in the element e. The VNN formulation with the loss function (3.1) inherits all of the advantages *hp*-VPINNs, i.e. *hp*-refinement, employing different test functions in each element e, and the flexibility of adaptively choosing the proper number of test functions in each element e.

We construct a fully connected network with ℓ hidden layers, each with $\mathcal{N}$ neurons and tanh activation functions (if not specifically mentioned otherwise). We use Legendre polynomials as test functions, i.e. $v_k(x) = P_{k-1}(x)$, $k = 1, 2, \ldots, K$. We also use the Gauss quadrature rule with Q quadrature points to compute the integrals. We consider two different approaches to approximate the function: (*i*) global or single element VNN where $N_{el} = 1$ and $v_k^{(e)}$'s are smooth functions v_k's, defined over the single element; (*ii*) local or elemental VNN where $N_{el} > 1$ and $v_k^{(e)}$'s are locally defined but only one of them is non-zero; (*iii*) multi-elemental VNN where $N_{el} > 1$ and $v_k^{(e)}$'s are locally defined and all are non-zero. In approach (*ii*), the network captures the target function only on the restricted local elements, where $v_k^{(e)}$ is non-zero.

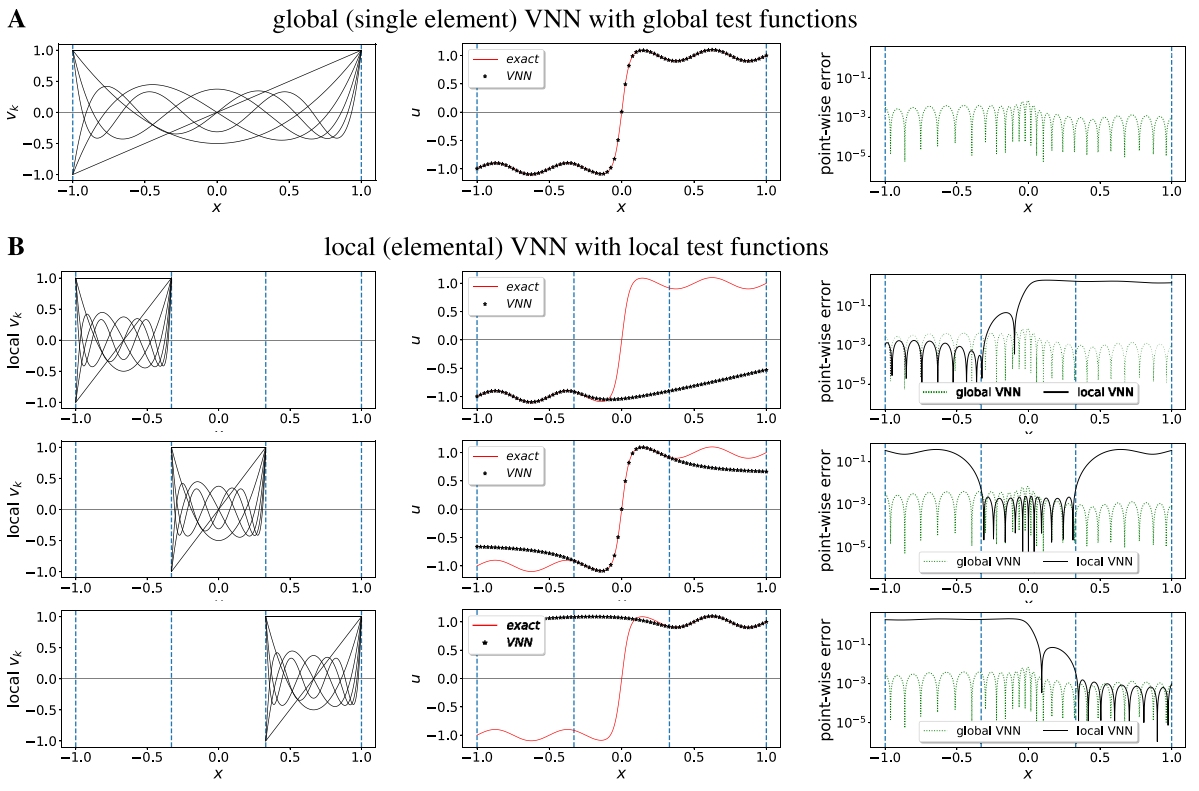

Fig. 1. VNN continuous function approximation. (**A**) global VNN: (left) the test functions, defined over the whole computational domain; (middle) the exact function and VNN approximation; (right) point-wise error. (**B**) local VNN: (left column) the test functions, defined locally over the individual sub-domains; (middle column) the exact function and VNN approximation; (right column) point-wise error. The dashed blue lines are the sub-domain boundaries. The VNN parameters are $\{\ell = 4, \mathcal{N} = 20, K = 60, Q = 80\}$. We use Adam optimizer with learning rate 10^{-3}.

Example 3.1 (*Continuous Function Approximation*)**.** We consider a smooth target function of the form

$$u^{exact} = 0.1\ \sin(4\pi x) + \tanh(20x),$$

which is defined over the domain $x \in \Omega = [-1, 1]$. We use VNN to approximate the target function, using the global and local test functions. The results are shown in Fig. 1.

The considered function is smooth, continuous, and thus, can be approximated accurately using VNN with $N_{el} = 1$. Using $\ell = 4$, $\mathcal{N} = 20$, $K = 60$, and $Q = 80$, we obtain L^∞ error of $O(10^{-3})$. By dividing the domain into three equally spaced sub-domains, we define the test functions over each sub-domain to locally approximate the target function. The key point here in local VNN is to focus the learning process by zooming into the sub-domain, where we are more interested to approximate accurately. In this setting, the network parameters are specifically optimized such that the network solely captures the function within that sub-domain. The local VNN results in a slightly more accurate approximation in each sub-domain, compared to global VNN. We show later that this setting also extends the approximation beyond the local sub-domain.

Example 3.2 (*Discontinuous Function Approximation*)**.** We consider a piecewise continuous target function of the form

$$u^{exact} = \begin{cases} 2\sin(4\pi x) & x \in [-1, 0), \\ 6 + \mathrm{e}^{1.2x}\sin(12\pi x) & x \in (0, 1], \end{cases}$$

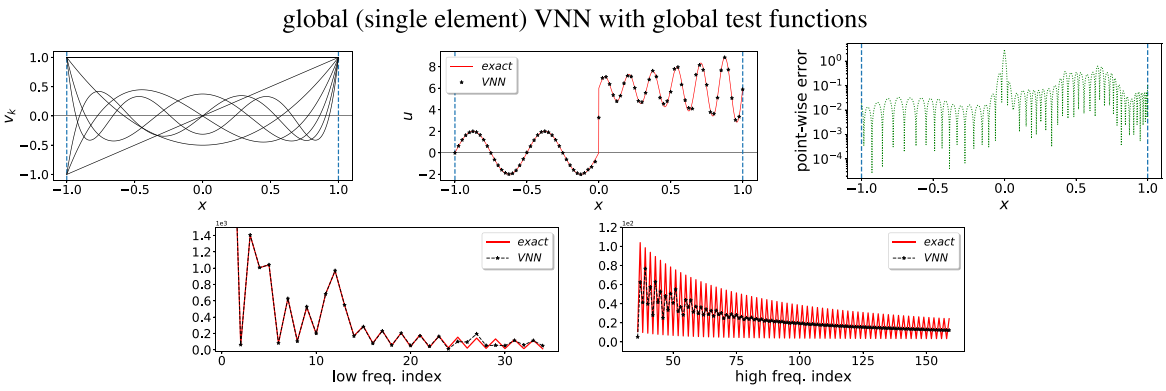

Fig. 2. VNN discontinuous function approximation: global (single element) VNN. Top row from left: test functions defined over the whole computational domain, the exact function and VNN prediction, and point-wise error. Bottom row: the target function and prediction in low and high frequency domain. The VNN parameters are $\{\ell = 4, \mathcal{N} = 20, K = 60, Q = 80\}$. We use Adam optimizer with learning rate 10^{-3}.

which is defined over the domain $x \in \Omega = [-1, 1]$ and has a jump of magnitude 6 at $x = 0$. We use VNN to approximate the target function, using the global and local test functions. The results are shown in Figs. 2, 3, and 4.

The exact solution is comprised of sinusoidal waves with frequencies 4π and 12π, and a discontinuity. It is interesting to compare the approximations in the Fourier domain, where the two sinusoidal waves are represented by low frequency index and the discontinuity is represented by high frequency index. Fig. 2 shows the results of global (single element) VNN with global test functions in approximating the discontinuous function, where we see that the L^∞ error is of order $O(10^{-1})$, happening close to the discontinuity. In the Fourier domain, the network learns the low frequency index of the target function, however, fails to capture the high frequency index. In Fig. 3, we show the results of local (elemental) VNN with local test functions, in which by defining a relatively small sub-domain close to the discontinuity, we make sure that the network can capture the high frequency index very accurately. In Fig. 4, we show the results of multi-elemental VNN, where we see that the network can capture the target function accurately in the Fourier domain. We observe in our simulations that in general a DNN regresses a function by first learning the discontinuity and then the low to high frequencies. The multi-elemental setting, however, can optimally change this learning pattern by considering different domain decompositions.

Example 3.3. We consider the target function given in Example 3.1. We study the approximation error convergence of global (single element) VNN for different activation functions and network depth. The results are shown in Fig. 5.

The compositional structure of DNNs is responsible for their high expressivity and thus a (relatively) deeper network is assumed to provide a more accurate regression approximation. We see in Fig. 5 that by increasing the depth of network in VNN formulation while keeping the width constant, the error drops with different rates for various activation functions. The error saturates after certain depth, which is mainly because the network parameters cannot be further optimized more accurately.

We recall that the loss function in VNN formulation is based on the projection of discrepancy of network output and the target function onto polynomial function space, where a successful minimization of loss function leads to convergence in that space. We observe, however, that the error in H^1-norm also drops and, therefore, in addition to accurate approximation of the target function, the network also learns the first derivative of the target function (but less accurately). The accuracy and convergence rate strongly depend on the choice of activation function as we observe that ReLu is not successful in learning the derivative of target function compared to sine and tanh activations. This is an important feature of local VNN, as the network can further capture the target function beyond the local support by following the trend of its first derivative at the boundary of a sub-domain, while the loss function is only obtained over the local sub-domain; we further discuss this feature in the following.

Learning Out-of-The-Box. In the previous examples of local VNN, where we only train the network within a single sub-domain, we observe that the network can capture the target function with less accuracy slightly outside of

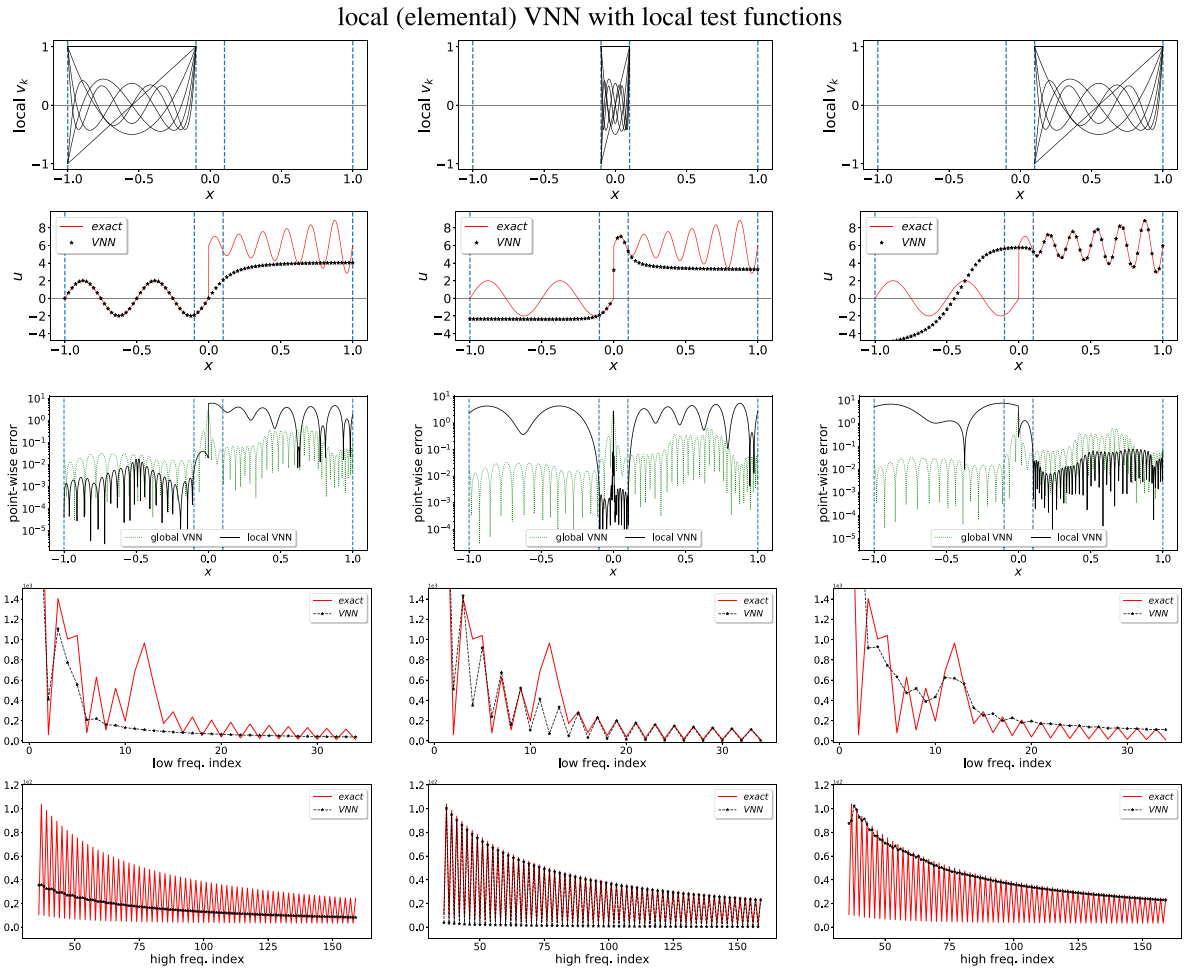

Fig. 3. VNN discontinuous function approximation: local (elemental) VNN. The dashed blue lines are the sub-domain boundaries. The row-wise captions are; First: locally defined test functions over each sub-domain. Second: the exact function and VNN prediction. Third: point-wise error. Fourth and fifth: low and high frequency indexes of the exact function and VNN prediction. The VNN parameters are $\ell = 4$, $\mathcal{N} = 20$, $K = 60$ local test functions, $Q = 80$ quadrature points in each sub-domain, and Adam optimizer with learning rate 10^{-3}.

that sub-domain. In fact, in addition to learning the target function, the network learns the derivative(s) of the target function within that sub-domain. The regularity of target function, localization of VNN, and structure of the network are important for more accurately predicting outside of the sub-domain. Fig. 6 shows an example, where the target function is $u^{exact} = \sin(8\pi x)$. We define the local test functions over a symmetric sub-domain $[-0.2, 0.2]$; see the dashed blue line. We observe that after locally learning the function within this sub-domain, the network follows the same trend at the sub-domain boundaries and therefore extrapolates outside the sub-domain. A zoomed-in plot of the left and right boundaries of the sub-domain is shown in Fig. 6.

4. One-dimensional Poisson's equation

Here, we discuss in detail the derivation of our proposed formulation *hp*-VPINN for the one-dimensional problem. Let $u(x) : \Omega \to \mathbb{R}$, where $\Omega = [-1, 1]$. We consider the Poisson's equation given as

$$-\frac{d^2u(x)}{dx^2} = f(x), \tag{4.1}$$

$$u(-1) = g, \quad u(1) = h, \tag{4.2}$$

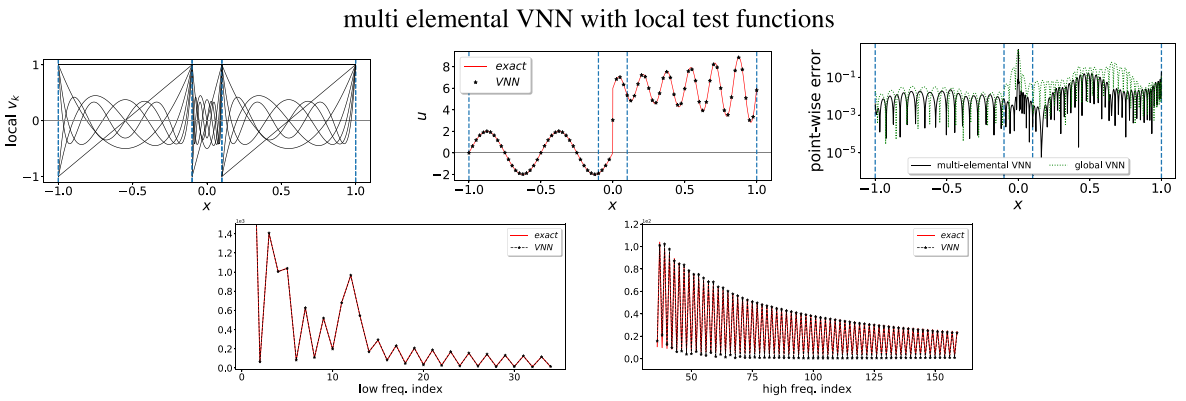

Fig. 4. VNN discontinuous function approximation: multi-elemental VNN. Top left: the domain is divided into three sub-domains and the test functions are defined locally over each sub-domain. The dashed blue lines are the sub-domains boundaries. Top middle: the exact function and VNN prediction. Top right: the point-wise error. Bottom row: the low and high frequency indexes of exact function and VNN prediction. The VNN parameters are $\ell = 4$, $\mathcal{N} = 20$, $K = 60$ local test functions, $Q = 80$ quadrature points in each sub-domain, and Adam optimizer with learning rate 10^{-3}.

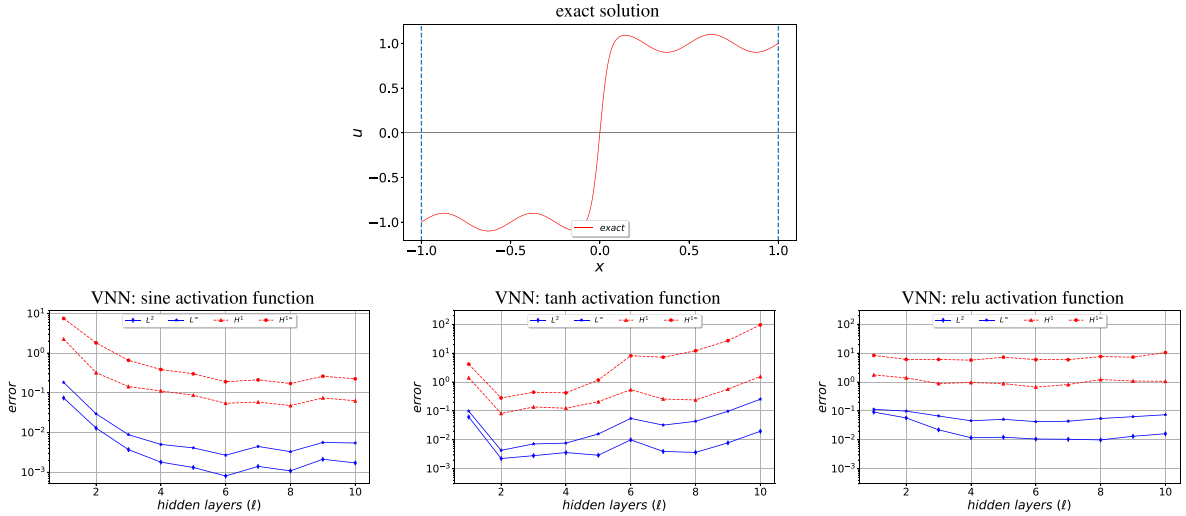

Fig. 5. Global (single element) VNN prediction of the exact function in 3.1 (Top) and error convergence with network depth (bottom) for sine, tanh, and ReLu activation functions. VNN is a fully connected network with parameters $\mathcal{N} = 20$, $K = 60$, $Q = 80$, and Adam optimizer with learning rate 10^{-3}. The results are averaged over 8 different network initialization.

where h and g are constants and we assume the force term $f(x)$ is available at some quadrature points. Let the approximate solution be $u(x) \approx \tilde{u}(x) = u_{NN}(x)$, then the strong-form residual (1.4) becomes

$$\begin{aligned} r(x) &= -\frac{d^2 u_{NN}(x)}{dx^2} - f(x), \quad x \in (-1, 1), \\ r_b(x) &= u_{NN}(x) - u(x), \qquad\quad x = \pm 1. \end{aligned} \tag{4.3}$$

We divide the domain $\Omega = [-1, 1]$ into non-overlapping elements $\Omega_e = [x_{e-1}, x_e]$ by defining a domain decomposition grid as $\{-1 = x_0, x_1, \ldots, x_{N_{el}} = 1\}$. We choose a set of localized nonoverlapping test functions $v_k(x)$, given in (2.1) with the nonvanishing function to be the high-order polynomials $P_{k+1}(x) - P_{k-1}(x)$, in which

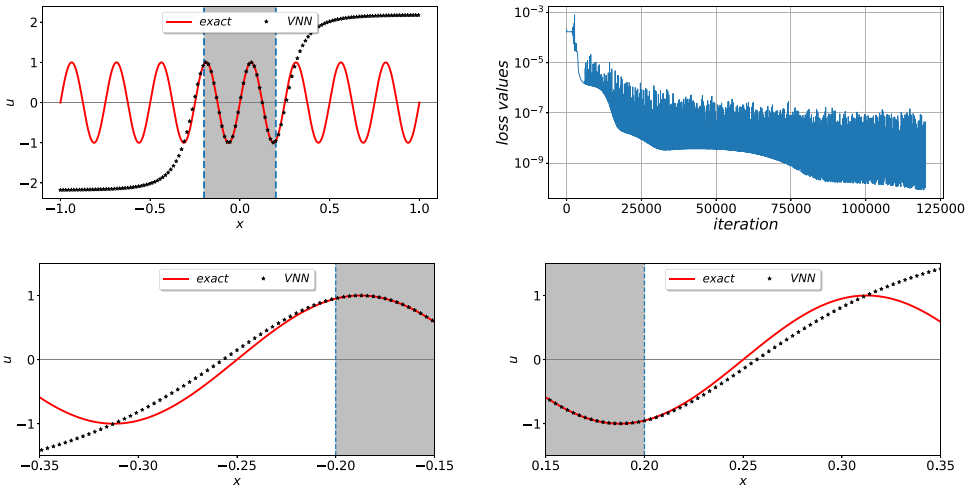

Fig. 6. VNN function approximation: learning out of the local element. The top left panel shows the exact function and VNN prediction. The shaded area is the sub-domain over which the test functions are locally defined and the network is trained. The bottom row shows the zoomed-in frame of the left and right boundaries of the sub-domain. The VNN parameters are $\ell = 4$, $\mathcal{N} = 20$, $K = 60$, $Q = 80$, and Adam optimizer with learning rate 10^{-3}.

$P_k(x)$ is Legendre polynomial of order k. The variational residual then becomes

$$\mathcal{R}_k = \sum_{e=1}^{N_{el}} \mathcal{R}_k^{(e)} = \sum_{e=1}^{N_{el}} \int_{x_{e-1}}^{x_e} \left(-\frac{d^2 u_{NN}(x)}{dx^2} - f(x) \right) v_k^{(e)}(x)\, dx, \tag{4.4}$$

where, in each term $\mathcal{R}_k^{(e)}$, $e = 1, 2, \ldots, N_{el}$, the integral variable x belongs to the sub-domain Ω_e. We can define the following three variational residual forms by integrating by parts the first term of $\mathcal{R}_k^{(e)}$. Thus,

$$^{(1)}\mathcal{R}_k^{(e)} = -\int_{x_{e-1}}^{x_e} \frac{d^2 u_{NN}(x)}{dx^2} v_k^{(e)}(x)\, dx - F_k^{(e)}, \tag{4.5}$$

$$^{(2)}\mathcal{R}_k^{(e)} = \int_{x_{e-1}}^{x_e} \frac{du_{NN}(x)}{dx} \frac{dv_k^{(e)}(x)}{dx}\, dx - \frac{du_{NN}(x)}{dx} v_k^{(e)}(x) \Big|_{x_{e-1}}^{x_e} - F_k^{(e)}, \tag{4.6}$$

$$^{(3)}\mathcal{R}_k^{(e)} = -\int_{x_{e-1}}^{x_e} u_{NN}(x) \frac{d^2 v_k^{(e)}(x)}{dx^2}\, dx - \frac{du_{NN}(x)}{dx} v_k^{(e)}(x) \Big|_{x_{e-1}}^{x_e} + u_{NN}(x) \frac{dv_k^{(e)}(x)}{dx} \Big|_{x_{e-1}}^{x_e} - F_k^{(e)}, \tag{4.7}$$

in which $F_k^{(e)} = \int_{x_{e-1}}^{x_e} f(x)\, v_k^{(e)}(x)\, dx$. Because $v_k^{(e)}(x)$ has a compact support over Ω_e, the first boundary term in (4.6) and (4.7) vanishes. The corresponding *variational loss function* for each case takes the form

$$L^{\mathfrak{v}(i)} = \sum_{e=1}^{N_{el}} \frac{1}{K^{(e)}} \sum_{k=1}^{K^{(e)}} \left| {}^{(i)}\mathcal{R}_k^{(e)} \right|^2 + \frac{\tau_b}{2} \left(\left| u_{NN}(-1) - g \right|^2 + \left| u_{NN}(1) - h \right|^2 \right), \quad i = 1, 2, 3, \tag{4.8}$$

where $K^{(e)}$ is the number of test functions in element e. For each element e, where $x \in [x_{e-1}, x_e]$, we transform the variational residual into the standard domain $\xi \in [-1, 1]$ via a proper affine mapping to compute the integrals.

4.1. Numerical results

We examine the performance of VPINN by considering different numerical examples. We construct a fully connected neural network with $\ell = 4$ layers and $\mathcal{N} = 20$ neurons in each layer with sine activation function. We employ up to order 60 Legendre polynomials and perform the integral using 80 Gauss–Lobatto quadrature points and weights (in each element). We write our formulation in Python, and employ Tensorflow to take advantage of

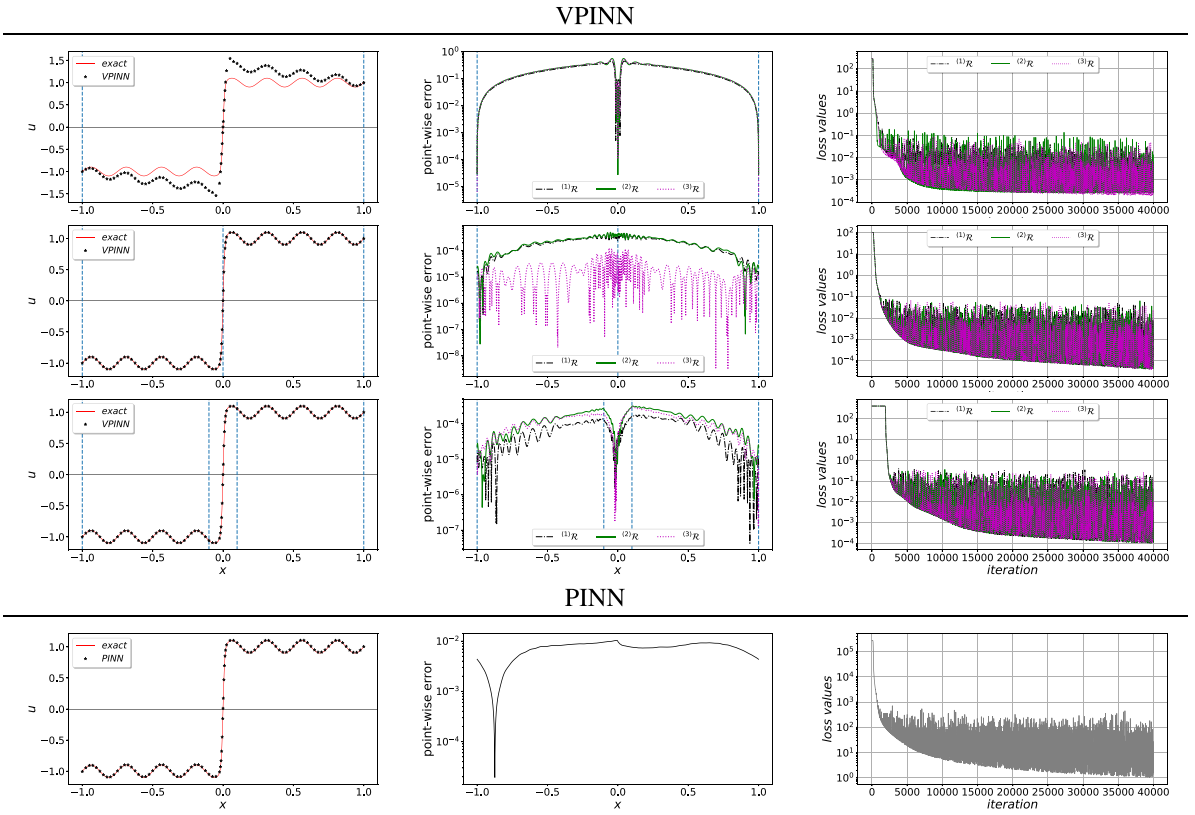

Fig. 7. One-dimensional Poisson's equation with steep solution (4.9): comparison of VPINN and PINN. Top panel shows the h-refinement of VPINN with single element (first row), two elements (second row), and three elements (third row). Column-wise captions: (left) the exact solution and VPINN prediction, (middle) point-wise error, (right) loss function versus training iterations. The VPINN parameters are $\{\ell = 4, \mathcal{N} = 20, \tau_b = 1\}$ and $\{K = 60, Q = 80\}$ in each element. The PINN parameters are $\{\ell = 4, \mathcal{N} = 20, N_r = 500, \tau_b = 10\}$. The networks are fully connected with sine activation function and we use Adam optimizer with learning rate 10^{-3}.

its automatic differentiation capability. We also use the extended stochastic gradient descent Adam algorithm [42] to optimize the loss function.

Example 4.1. We solve the problem (4.1)–(4.2) with exact solutions of the form

$$\text{steep solution: } u^{exact}(x) = 0.1\sin(8\pi x) + \tanh(80x), \tag{4.9}$$

$$\text{boundary layer solution: } u^{exact}(x) = 0.1\sin(5\pi x) + e^{\frac{0.01-(x+1)}{0.01}}. \tag{4.10}$$

In each case, we obtain the force term by substituting the exact solution in (4.1). The results are shown in Figs. 7, 8 and 10.

Fig. 7 shows the VPINN and PINN approximation to the Poisson's equation with steep solution (4.9). In VPINN, we see that the point-wise error is oscillatory, which is expected due to the modal nature of test functions. Compared with PINN results, the error is orders of magnitude less, yet it does not oscillate in PINN. Similar behavior is observed in the other example of boundary layer exact solution, shown in Fig. 10. It should be noted that for PINN to accurately capture a sharp change in the solution, we need to provide a larger number of residual points especially closer to the location of sharp change. We also observe that in the steep and boundary layer cases, the force term becomes very large, leading to a large loss value initially, which may sometimes results in an optimization failure. We also observe that to learn the boundary layer correctly, larger weights are assigned to the boundary term in the loss function of PINN (see e.g. [27]) than the weights for boundary in our formulation.

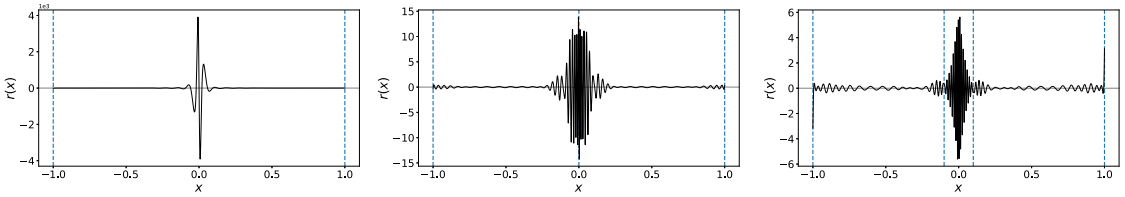

Fig. 8. One-dimensional Poisson's equation with steep solution (4.9): comparison of point-wise residual for adaptive h-refinements in VPINNs. Left: single element, Middle: two elements, Right: three elements. From left to right plots correspond to the first to third rows of Fig. 7.

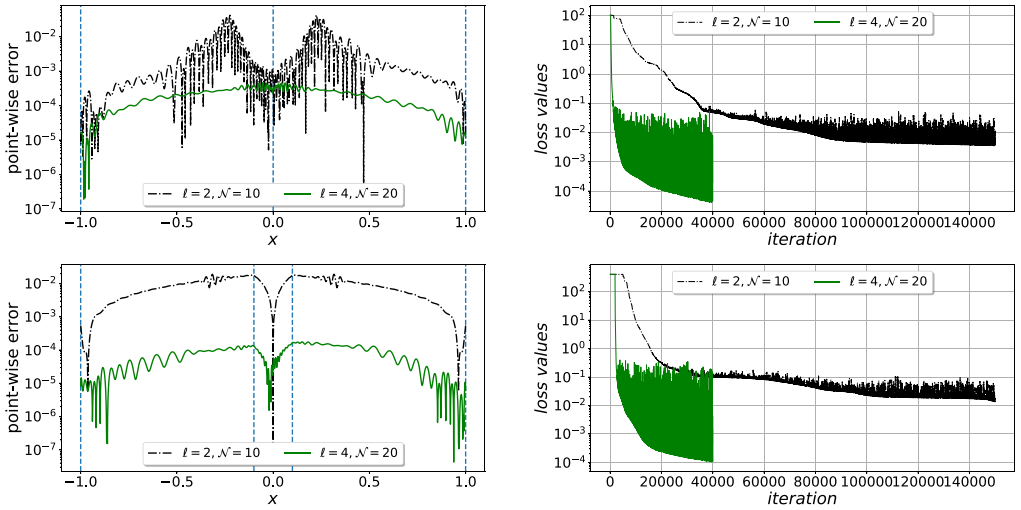

Fig. 9. One-dimensional Poisson's equation with steep solution (4.9): comparison of large ($\ell = 4, \mathcal{N} = 20$) and small ($\ell = 2, \mathcal{N} = 10$) network structure in VPINNs with ${}^{(1)}\mathcal{R}$ formulation. Top: h-refinement with $N_{el} = 2$. Bottom: h-refinement with $N_{el} = 3$. The VPINN parameters are $\tau_b = 1$ and $K = 60, Q = 80$ in each element. The networks are fully connected with sine activation function and we use Adam optimizer with learning rate 10^{-3}.

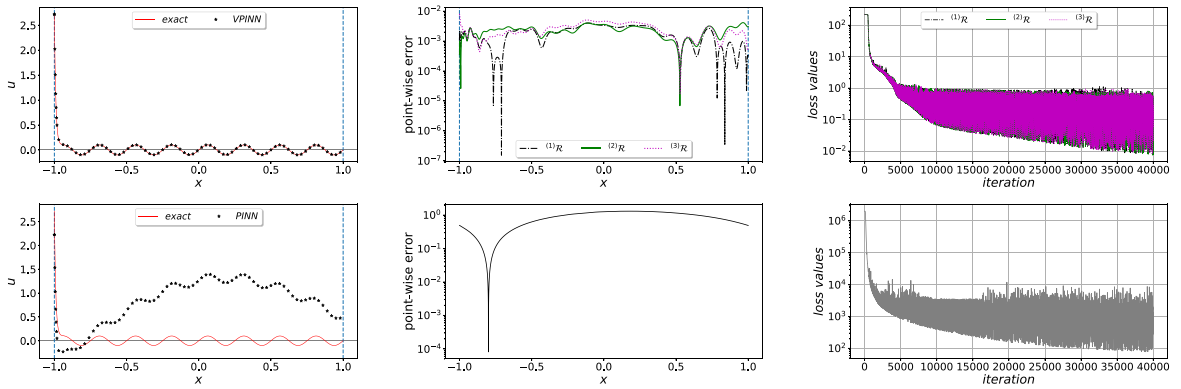

Fig. 10. One-dimensional Poisson's equation with boundary layer solution (4.10): comparison of VPINN (top row) and PINN (bottom row). Column-wise captions: (left) the exact solution and VPINN prediction, (middle) point-wise error, (right) loss function versus training iterations. The VPINN parameters are $\{\ell = 4, \mathcal{N} = 20, K = 60, Q = 80, \tau_b = 1\}$. The PINN parameters are $\{\ell = 4, \mathcal{N} = 20, N_r = 500, \tau_b = 10\}$. The networks are fully connected with sine activation function, and we use Adam optimizer with learning rate 10^{-3}.

The choice of network hyper-parameters, i.e. depth, width, activation functions, etc. is based on our numerical experiments and may not necessarily be an optimal choice. As a comparison, we show two small and large networks in Fig. 9 with $\ell = 2, \mathcal{N} = 10$, and $\ell = 4, \mathcal{N} = 20$, respectively. We observe that the optimization in the small

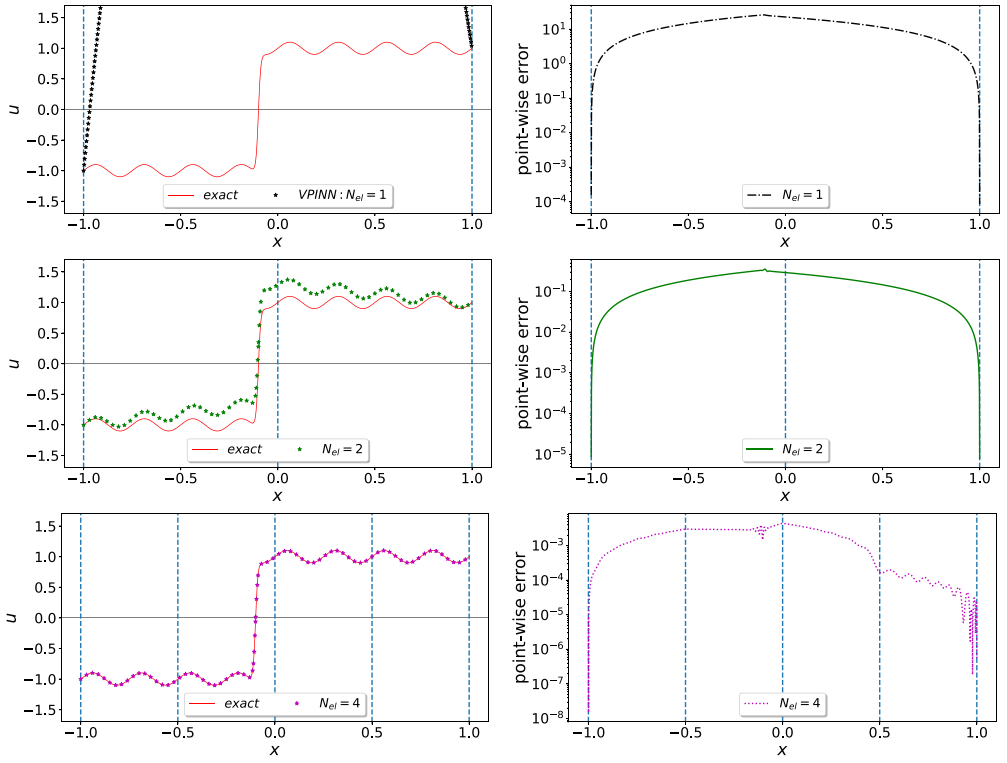

Fig. 11. One-dimensional Poisson's equation with asymmetric steep solution (4.11). The location of sharp change is not known a priori and is predicted by increasing number of elements in VPINN. Left column: the exact solution and VPINN prediction. Right column: point-wise error. The VPINN is based on the ${}^{(1)}\mathcal{R}$ formulation and has the parameters $\{\ell = 4, \mathcal{N} = 20, K = 60, Q = 80, \tau_b = 1\}$. The network is fully connected with sine activation function, and we use Adam optimizer with learning rate 10^{-3}.

network is much slower. We stop the optimization after 40000 iteration in the large network where the loss function is of order 10^{-4}, while in the small network after 150000 iteration the loss is of order 10^{-2}. Therefore, in this case the larger network provides a more accurate approximation of the solution.

Example 4.2. We solve the problem (4.1)–(4.2) with asymmetric steep solution of the form

$$\text{asymmetric steep solution: } u^{exact}(x) = 0.1\sin(8\pi x) + \tanh(80(x + 0.1)), \tag{4.11}$$

where the sharp change happens slightly off the origin. We assume that the location of sharp change is not known a priori and is obtained by successive domain decompositions into larger number of sub-domains; the results are shown in Fig. 11.

It is interesting to note that since the solution is asymmetric while the test functions are symmetric, the single-element VPINN does a pretty inaccurate approximation compared to the symmetric steep function (4.9). However, as we increase the number of elements, the network eventually captures the solution.

5. Two-dimensional Poisson's equation

Here, we discuss in detail the derivation of our proposed formulation *hp*-VPINN for the two-dimensional problem. Let $u(x, y) : \Omega \to \mathbb{R}$, where $\Omega = [-1, 1] \times [-1, 1]$. We consider the two-dimensional Poisson's equation

$$\nabla^2 u(x, y) = f(x, y), \tag{5.1}$$

subject to Dirichlet boundary conditions $h(x, y)$, and we assume the force term $f(x, y)$ is available at some quadrature points. Let the approximate solution be $u(x, y) \approx \tilde{u}(x, y) = u_{NN}(x, y)$, then the strong-form residual

(1.4) becomes

$$r(x,y) = \nabla^2 u_{NN}(x,y) - f(x,y), \quad \forall\{x,y\} \in \{x_r^i, y_r^i\}_{i=1}^{N_r} \in \Omega, \tag{5.2}$$
$$r_b(x,y) = u_{NN}(x,y) - h(x,y), \quad \forall\{x,y\} \in \{x_b^i, y_b^i\}_{i=1}^{N_b} \in \partial\Omega.$$

We construct a discrete finite dimensional test space by choosing finite set of size K_1 and K_2 of admissible test functions in x and y, respectively, and using the tensor product rule as

$$\widetilde{V} = \text{span}\big\{v_{k_1k_2}(x,y) = \phi_{k_1}(x)\phi_{k_2}(y),\ k_m = 1,2,\dots,K_m,\ m = 1,2\big\}. \tag{5.3}$$

The variational residual then reads as

$$\mathcal{R}_{k_1k_2} = \big(\nabla^2 u_{NN}(x,y) - f(x,y),\ v_{k_1k_2}(x,y)\big)_\Omega. \tag{5.4}$$

We define grids in x and y as $\{-1 = x_0, x_1, \dots, x_{N_{el_x}} = 1\}$ and $\{-1 = y_0, y_1, \dots, y_{N_{el_y}} = 1\}$, respectively, divide the domain Ω into structured sub-domains by constructing non-overlapping elements $\Omega_{e_xe_y} = [x_{e_x-1}, x_{e_x}] \times [y_{e_y-1}, y_{e_y}]$, $e_x = 1,2,\dots,N_{el_x}$, $e_y = 1,2,\dots,N_{el_y}$. Therefore, the variational residual becomes

$$\mathcal{R}_{k_1k_2} = \sum_{e_x=1}^{N_{el_x}} \sum_{e_y=1}^{N_{el_y}} \mathcal{R}_{k_1k_2}^{(e_xe_y)}, \tag{5.5}$$
$$\mathcal{R}_{k_1k_2}^{(e_xe_y)} = \int_{x_{e_x-1}}^{x_{e_x}} \int_{y_{e_y-1}}^{y_{e_y}} \big(\nabla^2 u_{NN}(x,y) - f(x,y)\big)\ \phi_{k_1}^{(e_x)}(x)\,\phi_{k_2}^{(e_y)}(y)\,dx\,dy,$$

where $k_1 = 1,2,\dots,K_1$ and $k_2 = 1,2,\dots,K_2$. We note that we can employ different number of test functions in each element, however, for simplicity in the derivation of formulation, we assume that we have similar number of test functions in all elements. We also note that the nonvanishing part of $\phi_{k_1}(x)$ and $\phi_{k_2}(y)$ has similar structure as the one-dimensional case. Therefore, the local test functions $v_{k_1k_2}^{(e)}(x,y)$ have the compact support over Ω_e. Therefore, for all elements, we have

$$\phi_{k_1}^{(e_x)}(x_{e_x-1}) = \phi_{k_1}^{(e_x)}(x_{e_x}) = 0, \quad k_1 = 1,2,\dots,K_1, \tag{5.6}$$
$$\phi_{k_2}^{(e_y)}(y_{e_y-1}) = \phi_{k_2}^{(e_y)}(y_{e_y}) = 0, \quad k_2 = 1,2,\dots,K_2.$$

By integrating by parts in the first term of $\mathcal{R}_{k_1k_2}^{(e_xe_y)}$, we can define the following variational residual forms, in which the term $F_{k_1k_2}^{(e_xe_y)}$ is associated with the integral of the force term $f(x,y)$.

$${}^{(1)}\mathcal{R}_{k_1k_2}^{(e_xe_y)} = \int_{x_{e_x-1}}^{x_{e_x}} \int_{y_{e_y-1}}^{y_{e_y}} \left(\frac{\partial^2 u_{NN}}{\partial x^2} + \frac{\partial^2 u_{NN}}{\partial y^2}\right) \phi_{k_1}^{(e_x)}(x)\,\phi_{k_2}^{(e_y)}(y)\,dx\,dy - F_{k_1k_2}^{(e_xe_y)}. \tag{5.7}$$

$${}^{(2)}\mathcal{R}_{k_1k_2}^{(e_xe_y)} = -\int_{x_{e_x-1}}^{x_{e_x}} \int_{y_{e_y-1}}^{y_{e_y}} \left(\frac{\partial u_{NN}}{\partial x}\,\frac{d\phi_{k_1}^{(e_x)}(x)}{dx}\,\phi_{k_2}^{(e_y)}(y) + \frac{\partial u_{NN}}{\partial y}\,\phi_{k_1}^{(e_x)}(x)\,\frac{d\phi_{k_2}^{(e_y)}(y)}{dy}\right) dx\,dy - F_{k_1k_2}^{(e_xe_y)}. \tag{5.8}$$

$$\begin{aligned}{}^{(3)}\mathcal{R}_{k_1k_2}^{(e_xe_y)} = &\int_{x_{e_x-1}}^{x_{e_x}} \int_{y_{e_y-1}}^{y_{e_y}} \left(u_{NN}(x,y)\,\frac{d^2\phi_{k_1}^{(e_x)}(x)}{dx^2}\,\phi_{k_2}^{(e_y)}(y) + u_{NN}(x,y)\,\phi_{k_1}^{(e_x)}(x)\,\frac{d^2\phi_{k_2}^{(e_y)}(y)}{dy^2}\right) dx\,dy \\ &- \int_{y_{e_y-1}}^{y_{e_y}} \left(\frac{\partial u_{NN}}{\partial x}\,\frac{d\phi_{k_1}^{(e_x)}(x)}{dx}\Big|_{x_{e_x-1}}^{x_{e_x}}\right) \phi_{k_2}^{(e_y)}(y)\,dy + \int_{x_{e_x-1}}^{x_{e_x}} \left(\frac{\partial u_{NN}}{\partial y}\,\frac{d\phi_{k_2}^{(e_y)}(y)}{dy}\Big|_{y_{e_y-1}}^{y_{e_y}}\right) \phi_{k_1}^{(e_x)}(x)\,dx \\ &- F_{k_1k_2}^{(e_xe_y)}.\end{aligned} \tag{5.9}$$

We reduce the order of tensor $\mathcal{R}_{k_1k_2}^{(e_xe_y)}$ to one by stacking its entries into vector of size K_1K_2. Subsequently, we define the *variational loss function* as

$$L^{\mathfrak{v}(i)} = \sum_{e_x=1}^{N_{el_x}} \sum_{e_y=1}^{N_{el_y}} \frac{1}{K_1K_2} \sum_{k=1}^{K_1K_2} \left|{}^{(i)}\mathcal{R}_k^{(e_xe_y)}\right|^2 + \tau_b \frac{1}{N_b} \sum_{i=1}^{N_b} \left|r_b(x_b^i, y_b^i)\right|^2, \quad i = 1,2,3, \tag{5.10}$$

where the term ${}^{(i)}\mathcal{R}_k^{(e_x e_y)}$ is the kth entry of the corresponding reduced tensor associated with element $e_x e_y$, and r_b has the same form as in (1.4). We note that the integrals in the variational residuals can be mapped into standard element $\{\xi, \eta\} \in [-1, 1] \times [-1, 1]$ via proper affine mapping.

5.1. Numerical results

We examine the performance of VPINN by considering different numerical examples. We construct a fully connected neural network with different depth/width/activation functions. We employ Legendre polynomials in each direction x and y, and perform the integral in each element by employing the proper number of Gauss quadrature points using tensor product rule. We write our formulation in Python, and employ Tensorflow to take advantage of its automatic differentiation capability. We also use the extended stochastic gradient descent Adam algorithm [42] to optimize the loss function.

Example 5.1. We solve the homogeneous two-dimensional Poisson's equation, i.e. (5.1) with $f(x, y) = 0$, over the bi-unit square domain $\Omega = [-1, 1] \times [-1, 1]$. The exact solution is given as

$$u^{exact}(x, y) = \frac{2(1 + y)}{(3 + x)^2 + (1 + y)^2}. \tag{5.11}$$

The results are shown in Fig. 12.

In this case, the exact solution is smooth and thus an accurate approximation can be obtained by using a relatively small network with $\ell = 3$, $\mathcal{N} = 5$, and tanh activation function. We compare the point-wise approximation error of PINN and *hp*-VPINN (with single domain and multiple sub-domains). The PINN formulation uses $N_r = 100$ residual and $N_b = 80$ boundary points randomly drawn from a uniform distribution. In *hp*-VPINN formulation, we use 5 test functions in each direction x and y, and employ 10×10 quadrature points. In this case, the domain decomposition does not improve the approximation and the point-wise error is of order $O(10^{-4})$ in all formulations. However, the domain decomposition can be later used in parallel computation, where each sub-domain can be individually solved in separate computer node, and thus, further improve the total computational costs. We note that the ${}^{(1)}\mathcal{R}$ and ${}^{(2)}\mathcal{R}$ formulations produce similar error level and we only show the results for latter one. The ${}^{(3)}\mathcal{R}$ is not considered here as the boundary terms can cause further complication in the loss function.

Example 5.2. We solve the two-dimensional Poisson's equation with the following exact solution with a steep change along x direction and a sinusoidal behavior in y direction as

$$u^{exact}(x, y) = (0.1 \sin(2\pi x) + \tanh(10x)) \times \sin(2\pi y), \tag{5.12}$$

where the force function is obtained by substituting the exact solution in (5.1). The results are shown in Fig. 13.

In this case, we use a wider network with $\ell = 3$, $\mathcal{N} = 20$, and tanh activation function to accurately capture the steep change at $x = 0$. We study the error convergence in the *hp*-VPINN formulation by successively increasing the number of sub-domains in the domain decomposition. In each sub-domain, we use 5 test functions in each direction x and y, and employ 10×10 quadrature points. Fig. 13 shows the convergence of error as we increase the number of division along x and y axes. For $N_{el_x} = N_{el_y} = 8$, the error is of order $O(10^{-3})$.

Example 5.3. We solve the homogeneous two-dimensional Poisson's equation, i.e. (5.1) with $f(x, y) = 0$, over the L-shaped domain Ω. The *hp*-VPINN results are shown in Fig. 14. For comparison, we also present the numerical solution obtained by using the spectral element method (SEM) [43]. The solution and comparison of PINN with SEM are given in [44].

The exact solution is not available in this case and thus we consider the SEM solution [43,44] as a benchmark solution (the SEM uses total 12 equal elements with degree of polynomial 10×10 in x and y directions). The difficulty in this example is to accurately approximate the solution at the sharp edge $x = y = 0$. We see in [44] that the PINN formulation produces the largest error at this vertex while preserving better accuracy over the rest of the domain; it uses $N_r = 1200$ residual and $N_b = 120$ boundary points randomly drawn from a uniform distribution. We observe similar behavior in the *hp*-VPINN formulation too, however, we report a better accuracy over the

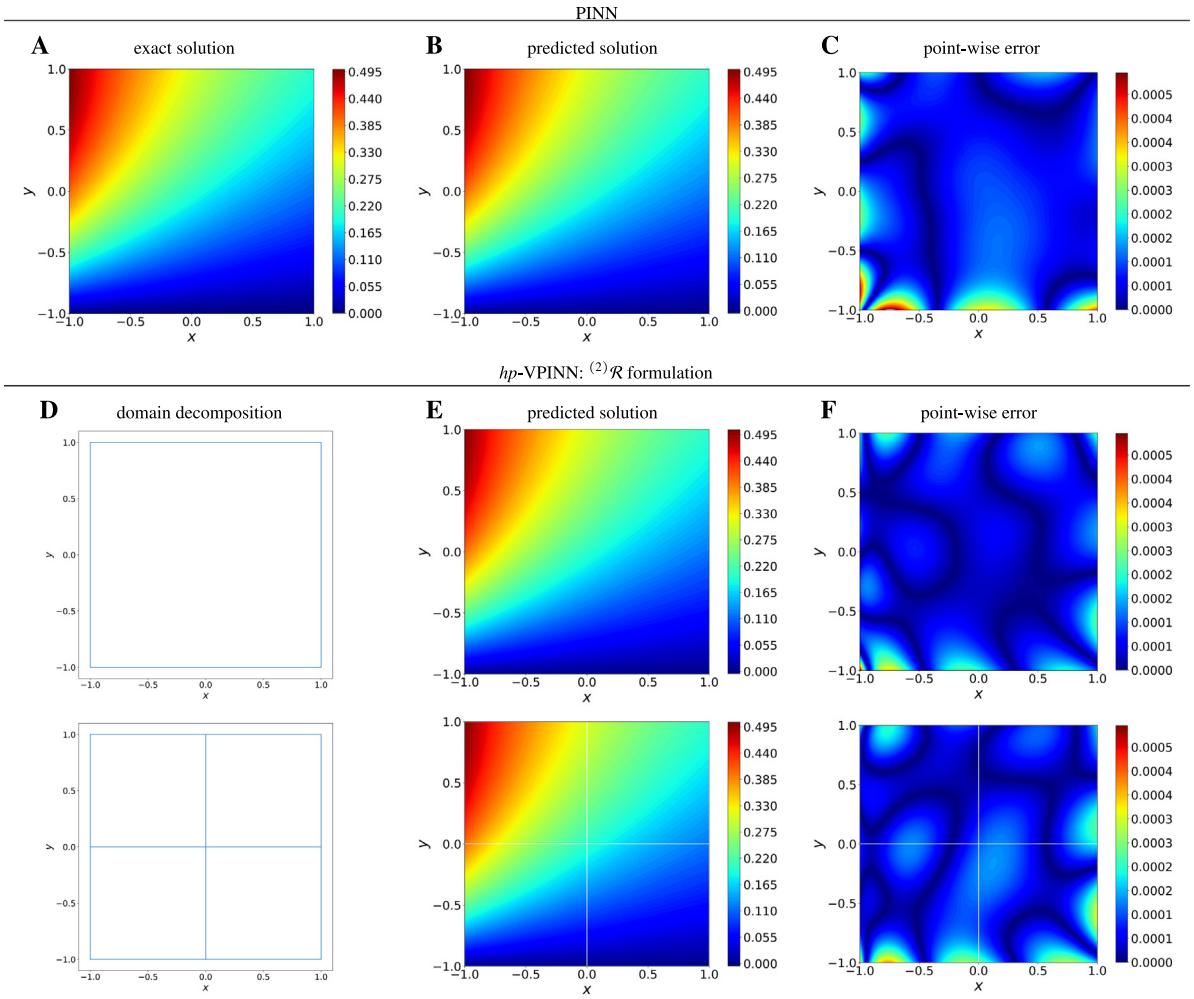

Fig. 12. Two-dimensional homogeneous Poisson's equation. Top panel: (**A**) Exact solution (5.11), (**B**) PINN prediction, and (**C**) PINN point-wise error. Bottom panel: (**D** column) h-refinement via domain decomposition $N_{el_x} = N_{el_y} = 1$ and 2, (**E** column) hp-VPINN prediction, and (**F** column) hp-VPINN point-wise error. In all cases, the network is fully connected with $\ell = 3$, $\mathcal{N} = 5$, and tanh activation function. The PINN parameters are $\{N_r = 100, N_b = 80\}$ random residual and boundary points and $\tau_b = 10$. The hp-VPINN parameters are $\{K_1 = K_2 = 5, Q = 10 \times 10\}$ in each sub-domain (element), $N_b = 80$ boundary points, and $\tau_b = 10$. We use Adam optimizer with learning rate 10^{-3}.

interior domain by refining the domain decomposition. In a coarse decomposition, we divide the domain into three sub-domains of equal sizes as shown in Fig. 14. Then, in a fine decomposition, we divide the domain into total of 35 sub-domains with different sizes. This finer h-refinement is suggested based on the point-wise values of residuals shown in Fig. 15. In both cases, we use 5 test functions in each direction x and y, and employ 10×10 quadrature points in each sub-domain. We can see that in the fine domain decomposition, the interior domain error decreases, while the error at the sharp edge is still dominant. In all hp-VPINN formulations, we use a fully connected network with $\ell = 3$, $\mathcal{N} = 20$, and tanh activation function. We also note that ${}^{(1)}\mathcal{R}$ and ${}^{(2)}\mathcal{R}$ formulations produce similar error level in this test case.

6. Advection diffusion equation (inverse and forward problems)

In this section, we consider both linear and nonlinear advection–diffusion equations, including an inverse problem for the linear advection–diffusion equation.

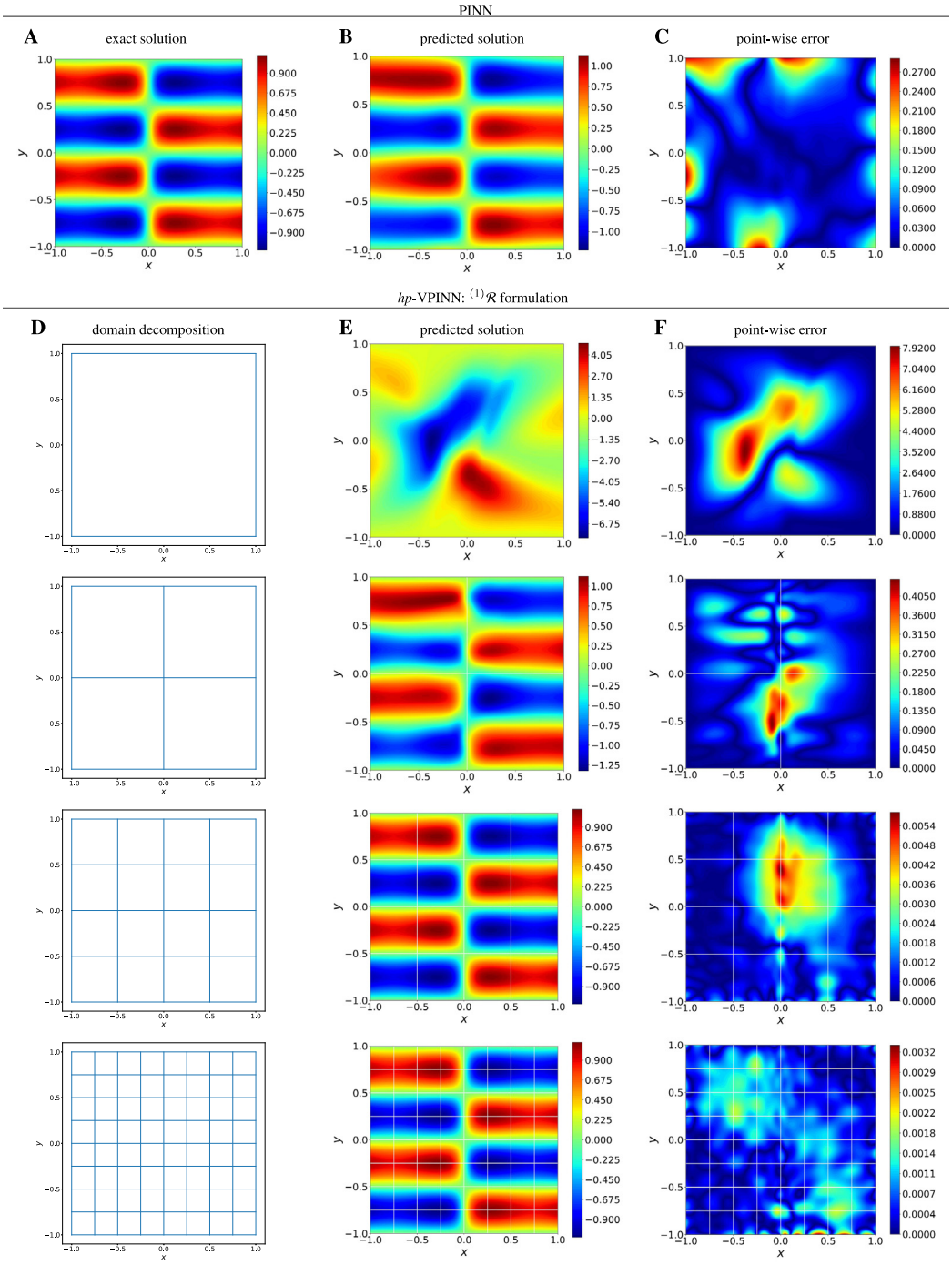

Fig. 13. Two-dimensional Poisson's equation with steep exact solution. Top panel: (**A**) Exact solution (5.12), (**B**) PINN prediction, and (**C**) PINN point-wise error. Bottom panel: (**D** column) h-refinement via domain decomposition $N_{el_x} = N_{el_y} = 1$, 2, 4, and 8, (**E** column) hp-VPINN prediction, and (**F** column) hp-VPINN point-wise error. In all cases, the network is fully connected with $\ell = 3$, $\mathcal{N} = 5$, and tanh activation function. The PINN parameters are $\{N_r = 1000, N_b = 80\}$ random residual and boundary points and $\tau_b = 10$. The hp-VPINN parameters are $\{K_1 = K_2 = 5, Q = 10 \times 10\}$ in each sub-domain (element), $N_b = 80$ boundary points, and $\tau_b = 10$. We use Adam optimizer with learning rate 10^{-3}.

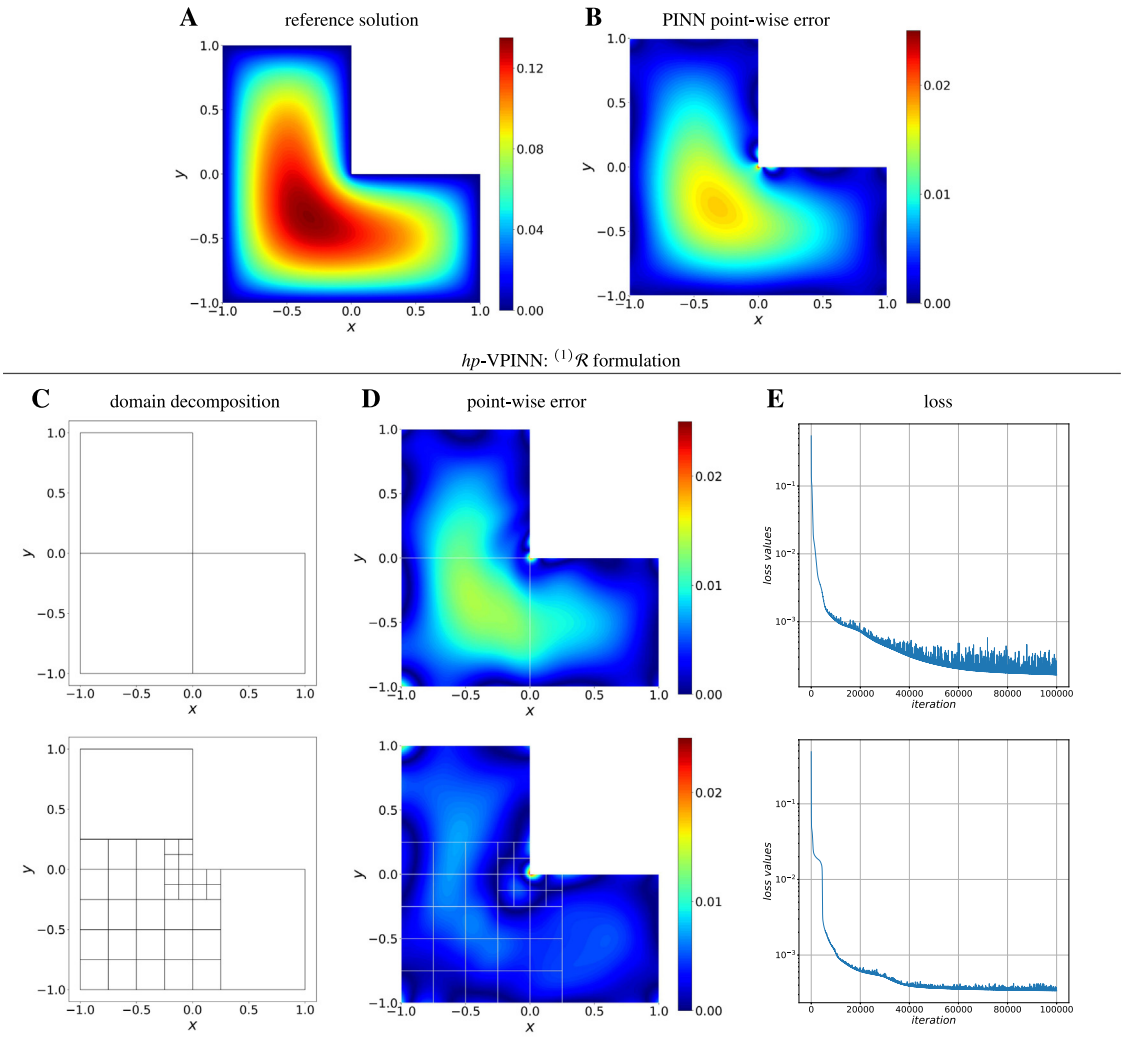

Fig. 14. Two-dimensional homogeneous Poisson's equation in L-shaped domain. Top panel: (**A**) The reference solution u_{SEM}, and (**B**) PINN point-wiser error $|u_{NN} - u_{\text{SEM}}|$. Bottom panel: (**C** column) Adaptive h-refinement via domain decomposition, (**D** column) hp-VPINN point-wise error $|u_{NN} - u_{\text{SEM}}|$, and (**E** column) loss value versus training iterations. The hp-VPINN is fully connected with $\ell = 3$, $\mathcal{N} = 5$, tanh activation function, and with parameters $\{K_1 = K_2 = 5, Q = 10 \times 10\}$ in each sub-domain (element) and $\tau_b = 10$. We use Adam optimizer with learning rate 10^{-3}.

Given a sparse observation/measurement of a mathematical model's solution, one seeks an accurate estimation of unobservable quantities/parameters. Such estimation is usually formulated as PDE-constraint optimization problems. Sparse observation and incomplete and noisy data make the mapping from data to the unknown parameters challenging to estimate. The optimization problem generally does not lend itself to a straightforward solution. Moreover, the PDE is usually solved by a proper numerical method. For example, finite difference and spectral methods are used to solve the PDE in adjoint state methods [45–48]. Other methods have also been developed; see [49–51] and the references therein with many applications.

As an alternative of classical numerical methods, the PINN formulation can incorporate the model parameters into the neural network parameters in an inverse problem setting [52,53]. The gradient with respect to the physical coordinates and unknown model parameters is computed using the automatic differentiation, as employed in

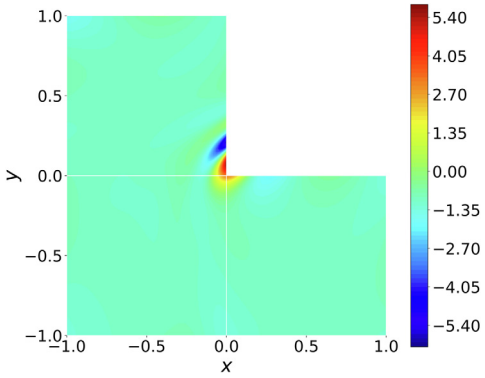

Fig. 15. Two-dimensional homogeneous Poisson's equation in L-shaped domain. The point-wise value of residual corresponding to the first domain decomposition in Fig. 14. We adaptively refine the mesh in the neighborhood of the corner to further improve the approximation.

Tensorflow [54]. Thus, the training algorithm can simultaneously optimize neural network and model parameters. The PINN formulation also uses the capability of overparameterized DNNs to incorporate noisy data inputs; See, e.g., [37,55,56]. Another method called physics constrained learning has also been developed [57,58] that numerically enforces the physical constraint and uses the reverse mode automatic differentiation and forward Jacobian propagation to extract the gradients and Jacobian matrices.

Below, we use VPINNs in place of PINNs to solve the inverse problem for parameter estimation, where the PDE constraint is enforced in a variational setting. We show that the VPINN formulation can also solve inverse problems, although the VPINN formulation does not necessarily perform more accurately for inverse problems.

Let $u(t,x) : \Omega \to \mathbb{R}$, where $\Omega = [0,1] \times [-1,1]$. We consider the $(1+1)$-dimensional advection diffusion equation (ADE)

$$\begin{aligned} &\frac{\partial u}{\partial t} + v\frac{\partial u}{\partial x} = \kappa \frac{\partial^2 u}{\partial x^2}, \\ &u(-1,t) = u(1,t) = 0, \\ &u(x,0) = -\sin(\pi x), \end{aligned} \tag{6.1}$$

where the constant coefficients $v = 1$ and $\kappa = 0.1/\pi$ are the advection velocity and the diffusivity coefficient, respectively. When the diffusion coefficients κ is small the advection becomes dominant, which complicates the solution close to the right boundary $x = 1$ as the no slip boundary condition is imposed. The analytical solution of the ADE problem (6.1) is given in [59] in terms of infinite series summation. We use 800 number of terms to compute the analytical solution and compare with our proposed method.

We let the approximate solution be $u(t,x) \approx \tilde{u}(t,x) = u_{NN}(t,x)$ and note that in the transient problem, time can be thought of as another dimension and thus the formulation of variational residuals become similar to the previous examples. The space–time domain Ω is decomposed into $N_{el_t} \times N_{el_x}$ structured non-overlapping sub-domains (elements) $\Omega_{e_t e_x} = [t_{e_t-1}, t_{e_t}] \times [x_{e_x-1}, x_{e_x}]$, $e_t = 1, 2, \ldots, N_{el_t}$, $e_x = 1, 2, \ldots, N_{el_x}$ via constructing the temporal and spatial grids $\{0 = t_0, t_1, \ldots, t_{N_{el_t}} = 1\}$ and $\{-1 = x_0, x_1, \ldots, x_{N_{el_x}} = 1\}$, respectively. Fig. 16 shows the h-refinement of the hp-VPINN method by considering different domain decompositions, i.e. $N_{el_t} = N_{el_x} = 1$, 2, and 4. The point-wise error is shown based on the ${}^{(1)}\mathcal{R}$ formulation. We also report similar point-wise error for the ${}^{(2)}\mathcal{R}$ formulation.

Example 6.1 (*Diffusivity Estimation*)**.** We consider the ADE (6.1) and let $\mathbf{q} = \{v, \kappa\}$ be the set of model parameters, where the advection velocity is known to be a constant $v = 1$ and the diffusion coefficient κ is unknown. Although in this case we have the analytical solution, we assume that the (observed/measured) values of exact solution $u^\star(t^\star, x^\star)$ is only available as time series at three (sensor) locations along the x axis, i.e., $x^\star = \{-0.5, 0, 0.5\}$. We randomly select 5 data points at each sensor, and thus in total 15 measurements all over the whole domain; an example of

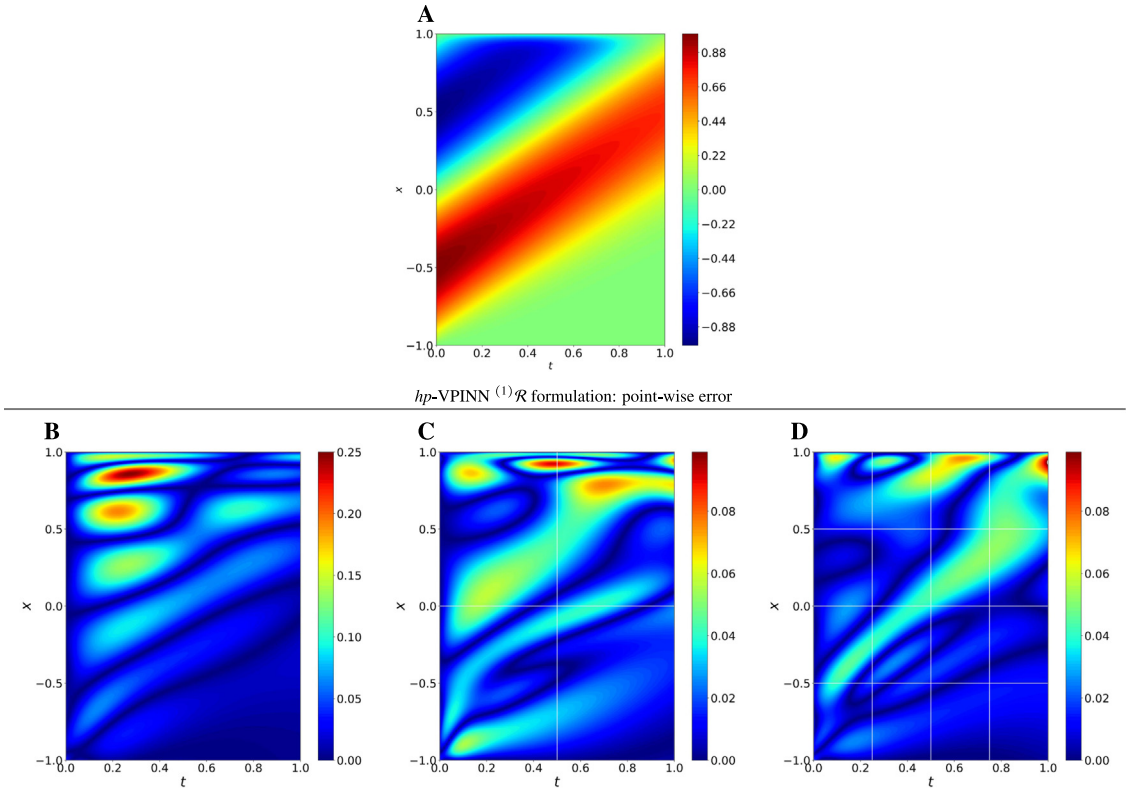

Fig. 16. (1 + 1)-dimensional advection diffusion equation. (**A**) the exact solution. (**B**, **C**, and **D**) Point-wise error for the h-refinement of hp-VPINN with $^{(1)}\mathcal{R}$ formulation for $N_{el_t} = N_{el_x} = 1$, 2, and 4. In all cases, the network is fully connected with $\ell = 3$, $\mathcal{N} = 5$, and tanh activation function. The hp-VPINN parameters are $\{K_1 = K_2 = 5, Q = 10 \times 10\}$ in each sub-domain (element), $N_b = 80$ boundary points, and $\tau_b = 10$. We use Adam optimizer with learning rate 10^{-3}.

these points is shown as black squares in Fig. 17. We pose the inverse problem diffusivity estimation as follows:

> given the measurement set $\{t_i^\star, x_i^\star, u_i^\star(t_i^\star, x_i^\star)\}_{i=1}^{N^\star}$, estimate the diffusion coefficient κ in ADE (6.1).

The results are shown in Fig. 17.

The additional data points from given measurements/observations in the inverse problem are added as the following extra term in the variational loss function (1.9)

$$\tau^\star \frac{1}{N^\star} \sum_{i=1}^{N^\star} \left| u_{NN}(t_i^\star, x_i^\star) - u^\star(t_i^\star, x_i^\star) \right|^2. \tag{6.2}$$

We use the VPINN formulation by constructing a fully connected neural network with tanh activation function, parameters $\ell = 3, \mathcal{N} = 5, K_1 = K_2 = 5, Q = 10 \times 10, \tau_b = \tau_0 = \tau^\star = 10, N_b = 160, N_i = 80, N^\star = 15$, and Legendre test functions in both space and time direction. We recall that the parameters K_1 and K_2 are the number of test functions in space and time. We also note that here we use the $^{(1)}\mathcal{R}$ formulation in the VPINN.

The unknown diffusivity coefficient κ is initialized by one and as the network learns its parameter, the value of κ converges to its exact value. The estimation is averaged over 10 different cases of randomly selected $N^\star$ points. The convergence of mean value of κ, its standard deviation, and also values of loss function are shown in Fig. 17. We observe that after convergence of diffusion coefficients, the point-wise error has only a large magnitude close to the right boundary at $t = 1$.

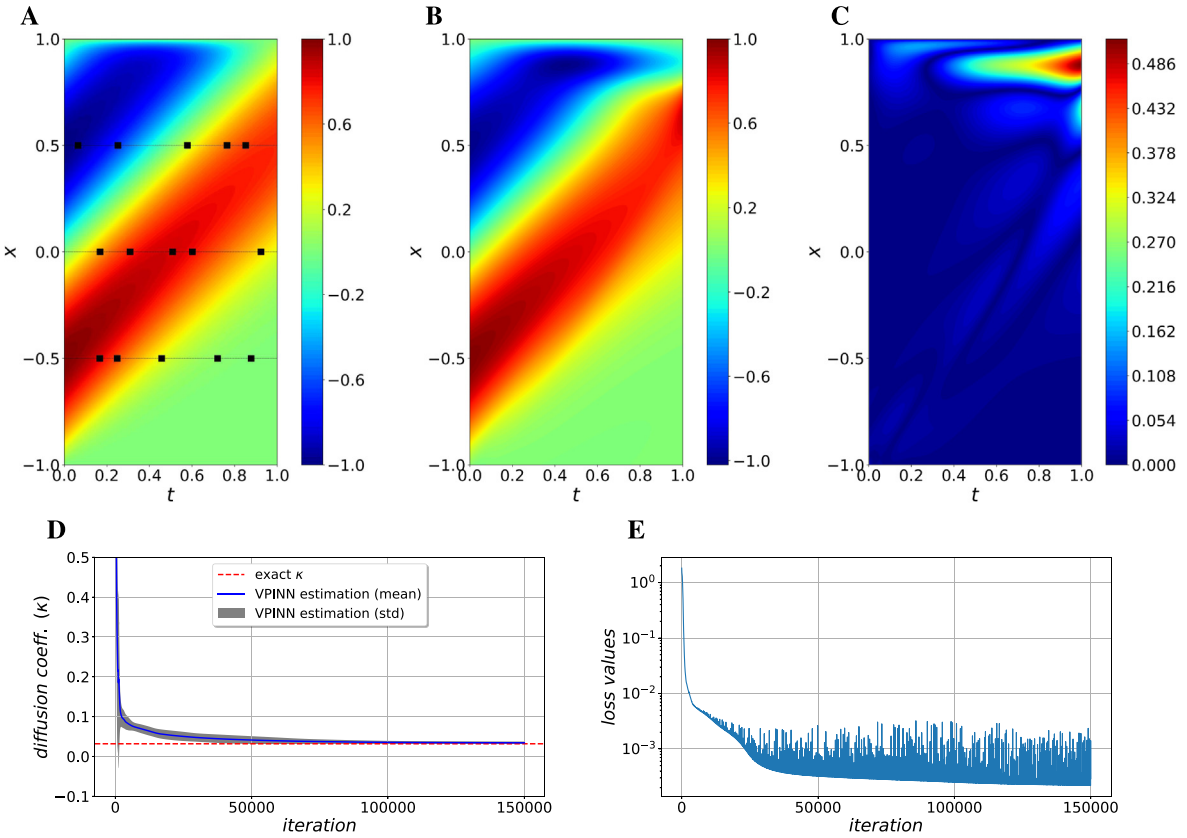

Fig. 17. Diffusivity estimation in (1 + 1)-dimensional ADE. (**A**) exact solution and one realization of $N^\star = 15$ randomly selected pints at the sensor locations. (**B**) VPINN prediction. (**C**) VPINN point-wise error. (**D**) convergence of diffusivity coefficient and (**E**) loss values versus training iterations.

Example 6.2 (*Advection Equation With Discontinuous Solution*)**.** We consider the advection equation by letting the diffusion coefficient $\kappa = 0$ and the velocity $v = 1$ in (6.1). The initial condition is given as

$$u(x, 0) = u_0 = \begin{cases} 1 & x \in [-0.2, 0.2], \\ 0 & \text{elsewhere.} \end{cases}$$

The exact solution in this case is $u^{exact}(x, t) = u_0(x - t)$.

The double discontinuity of the exact solution, which advects in time, makes it challenging for numerical methods to accurately capture the exact solution and the sharp changes at the discontinuities. We show in Fig. 18 that a deep network with 8 hidden layers can accurately capture the solution; the predicted solution versus the exact solution is plotted at different times $t = 0, 0.5, 1$.

Example 6.3 (*Viscous Burger's Equation*)**.** We consider the nonlinear advection diffusion equation, i.e., the Burger's equation

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \kappa \frac{\partial^2 u}{\partial x^2}$$

with $\kappa = 0.01/\pi$, $u(x, 0) = -\sin(\pi x)$ and homogeneous Dirichlet boundary conditions.

The Burger's equation develops a steep interior layer even with a smooth initial condition after a certain time. The emerging steep solution makes it quite challenging for most numerical methods to accurately capture the solution. Here, we use the h-refinement capacity of developed method to show that a refined mesh close to the steep part

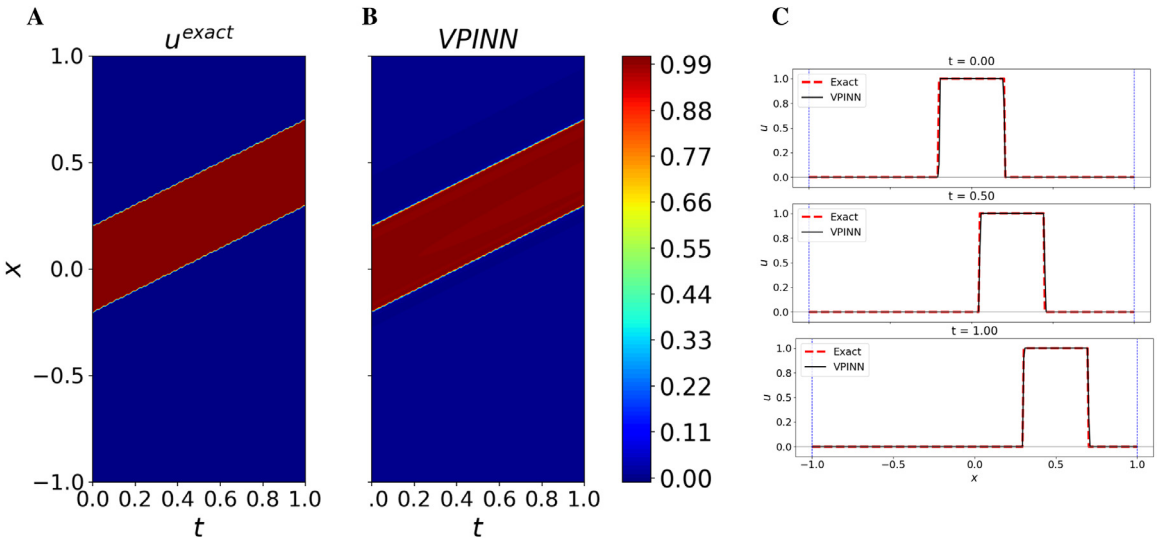

Fig. 18. (1 + 1)-dimensional advection equation with discontinuous initial condition. (**A**) exact solution, (**B**) VPINN prediction, and (**C**) time slices at $t = 0, 0.5, 1$. The *hp*-VPINN is a single domain based on the ${}^{(1)}\mathcal{R}$ formulation with parameters $\{K_1 = K_2 = 20, Q = 80 \times 80\}$, and $\tau_b = 50$. We use tanh activation function, Legendre polynomials test function in both time and space, and Adam optimizer with learning rate 10^{-3}.

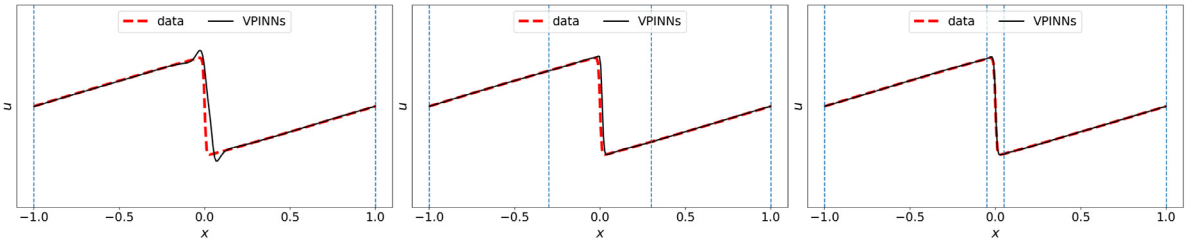

Fig. 19. (1 + 1)-dimensional Burgers equation. From left to right: adaptive *h*-refinement that leads to accurate approximation of the steep layer. The plots show the solution at final time $t = 1$. The blue dashed lines are the sub-domain boundaries.

of the solution can help to efficiently resolve the entire solution. We use a fully connected network with 4 hidden layers and 20 neurons in each layer. We employ the tanh activation function and obtain the variational form based on the ${}^{(1)}\mathcal{R}$ formulation. The test functions are Chebyshev polynomials of order 20 in time and space. Accordingly, we also use the Chebyshev points to compute the integrals in the variational residual. We show the results at the final time $t = 1$ for three cases of refinement in space in Fig. 19. In this figure, we denote the data as the solution obtained by the spectral element method [43].

7. Summary and discussion

We developed the *hp*-VPINN method to unify the current developments in deep neural networks for solving partial differential equations based on residuals of equations using least-squares. In particular, we focused on the variational formulation in the context of the sub-domain Petrov–Galerkin method, where the trial space is the space of neural networks and test space is localized non-overlapping high order polynomials. We showed that with integration-by-parts, *hp*-VPINN can work more efficiently with rough solutions/input data such as singularities, steep solution, and sharp changes.

We showed the efficiency and accuracy of *hp*-VPINNs and comparison with PINN in several examples of function approximation and solving differential equations. We developed the method in detail for one- and two-dimensional problems, derived the corresponding variational loss functions, and discussed the *hp*-refinement and convergence of

solution in solving equations with non-smooth solutions. Some convergence analysis of the proposed methods can be found in recent work on NN for linear PDEs [60,61].

In this manuscript, we placed our focus on the projection onto the space of higher-order polynomials. The optimal choice of test functions in the current developments is an important open question and requires further analysis. We expect that this optimal choice of test functions can be made by designing effective error indicators and applying adaptive strategies. Our heuristics is that choosing a basis should be similar to those in spectral/*hp* element methods. Further developments of error indicators for refinements and choosing a basis are needed to fully understand these basis functions' role. Currently, the choice of the test functions was made by either knowing a priori qualitative properties of the solutions to the underlying problems or the simplest error indicator — the values of the residuals on uniform points. In the latter case, *h*-refinement was imposed in the region of large values of residuals.

Moreover, rigorous and thorough comparison with other numerical methods such as finite element (FE) and finite difference (FD) methods are needed. However, the comparison depends on many factors, such as the choices of training points and basis functions, domain partitioning, size of networks and implementation, etc. Our proposed method is promising in inverse problems, yet the comparison with classical methods is not ready. We have made the codes for some of examples available on GitHub[1] for interested readers and invite them to further explore along the avenue of this research direction.

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.

Acknowledgments

This work was supported by the Applied Mathematics Program within the Department of Energy, USA on the PhILMs project (DE-SC0019453) and the DARPA CompMods program on the DeepM&Mnet project (HR00112090062).

References

[1] J. Berg, K. Nyström, A unified deep artificial neural network approach to partial differential equations in complex geometries, Neurocomputing 317 (2018) 28–41, http://dx.doi.org/10.1016/j.neucom.2018.06.056, URL http://www.sciencedirect.com/science/article/pii/S092523121830794X.

[2] W. E, B. Yu, The deep Ritz method: A deep learning-based numerical algorithm for solving variational problems, Commun. Math. Stat. 6 (1) (2018) 1–12.

[3] 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.

[4] Z. Mao, A.D. Jagtap, G.E. Karniadakis, Physics-informed neural networks for high-speed flows, Comput. Methods Appl. Mech. Engrg. 360 (2020) 112789.

[5] Y. Khoo, J. Lu, L. Ying, Solving for high-dimensional committor functions using artificial neural networks, Res. Math. Sci. 6 (1) (2019) 1.

[6] E. Samaniego, C. Anitescu, S. Goswami, V.M. Nguyen-Thanh, H. Guo, K. Hamdia, X. Zhuang, T. Rabczuk, An energy approach to the solution of partial differential equations in computational mechanics via machine learning: Concepts, implementation and applications, Comput. Methods Appl. Mech. Engrg. 362 (2020) 112790.

[7] Y. Liao, P. Ming, Deep Nitsche method: Deep Ritz method with essential boundary conditions, 2019, arXiv preprint arXiv:1912.01309.

[8] K. Li, K. Tang, T. Wu, Q. Liao, D3M: A deep domain decomposition method for partial differential equations, arXiv e-prints, 2019, arXiv:1909.12236.

[9] R.A. DeVore, Nonlinear approximation, Acta Numer. 7 (1998) 51–150.

[10] R.A. DeVore, Nonlinear approximation and its applications, in: Multiscale, Nonlinear and Adaptive Approximation, Springer, 2009, pp. 169–201.

[11] I. Daubechies, Ten Lectures on Wavelets, Vol. 61, SIAM, 1992.

[12] S. Tariyal, A. Majumdar, R. Singh, M. Vatsa, Greedy deep dictionary learning, 2016, arXiv preprint arXiv:1602.00203.

[13] G. Davis, Adaptive Nonlinear Approximations (Ph.D. thesis), New York University, Graduate School of Arts and Science, 1994.

[14] H. Ohlsson, A.Y. Yang, R. Dong, S.S. Sastry, Nonlinear basis pursuit, in: 2013 Asilomar Conference on Signals, Systems and Computers, IEEE, 2013, pp. 115–119.

[15] E.J. Candès, et al., Compressive sampling, in: Proceedings of the International Congress of Mathematicians, Vol. 3, Madrid, Spain, 2006, pp. 1433–1452.

[1] https://github.com/ehsankharazmi/hp-VPINNs.

[16] E.J. Candès, M.B. Wakin, An introduction to compressive sampling [a sensing/sampling paradigm that goes against the common knowledge in data acquisition], IEEE Signal Process. Mag. 25 (2) (2008) 21–30.
[17] R. DeVore, A. Ron, Approximation using scattered shifts of a multivariate function, Trans. Amer. Math. Soc. 362 (12) (2010) 6205–6229.
[18] T. Hangelbroek, A. Ron, Nonlinear approximation using Gaussian kernels, J. Funct. Anal. 259 (1) (2010) 203–219.
[19] H.N. Mhaskar, C.A. Micchelli, Approximation by superposition of sigmoidal and radial basis functions, Adv. in Appl. Math. 13 (3) (1992) 350–373.
[20] H.N. Mhaskar, T. Poggio, Function approximation by deep networks, 2019, arXiv preprint arXiv:1905.12882.
[21] I. Daubechies, R. DeVore, S. Foucart, B. Hanin, G. Petrova, Nonlinear approximation and (deep) ReLU networks, 2019, arXiv preprint arXiv:1905.02199.
[22] B.A. Finlayson, L.E. Scriven, The method of weighted residuals—A review, Appl. Mech. Rev. 19 (9) (1966) 735–748.
[23] J. Sirignano, K. Spiliopoulos, DGM: A deep learning algorithm for solving partial differential equations, J. Comput. Phys. 375 (2018) 1339–1364.
[24] L. Yang, D. Zhang, G.E. Karniadakis, Physics-informed generative adversarial networks for stochastic differential equations, 2018, arXiv preprint arXiv:1811.02033.
[25] E. Kharazmi, Z. Zhang, G.E. Karniadakis, Variational physics-informed neural networks for solving partial differential equations, 2019, arXiv preprint arXiv:1912.00873.
[26] R. Khodayi-Mehr, M.M. Zavlanos, VarNet: Variational neural networks for the solution of partial differential equations, 2019, arXiv preprint arXiv:1912.07443.
[27] S. Wang, Y. Teng, P. Perdikaris, Understanding and mitigating gradient pathologies in physics-informed neural networks, 2020, arXiv preprint arXiv:2001.04536.
[28] M. Raissi, H. Babaee, P. Givi, Deep learning of turbulent scalar mixing, Phys. Rev. Fluids 4 (12) (2019) 124501.
[29] G. Pang, L. Lu, G.E. Karniadakis, FPINNs: Fractional physics-informed neural networks, SIAM J. Sci. Comput. 41 (4) (2019) A2603–A2626.
[30] A.D. Jagtap, K. Kawaguchi, G.E. Karniadakis, Adaptive activation functions accelerate convergence in deep and physics-informed neural networks, J. Comput. Phys. (2019) 109–136.
[31] A.D. Jagtap, K. Kawaguchi, G.E. Karniadakis, Locally adaptive activation functions with slope recovery term for deep and physics-informed neural networks, 2019, arXiv preprint arXiv:1909.12228.
[32] E. Haghighat, R. Juanes, SciANN: A Keras wrapper for scientific computations and physics-informed deep learning using artificial neural networks, 2020, arXiv preprint arXiv:2005.08803.
[33] A. Al-Aradi, A. Correia, D.d.F. Naiff, G. Jardim, Y. Saporito, Applications of the deep Galerkin method to solving partial integro-differential and Hamilton-Jacobi-Bellman equations, 2019, arXiv preprint arXiv:1912.01455.
[34] Y. Zang, G. Bao, X. Ye, H. Zhou, Weak adversarial networks for high-dimensional partial differential equations, J. Comput. Phys. (2020) 109409.
[35] G. Bao, X. Ye, Y. Zang, H. Zhou, Numerical solution of inverse problems by weak adversarial networks, 2020, arXiv preprint arXiv:2002.11340.
[36] R. Khodayi-mehr, M.M. Zavlanos, Deep learning for robotic mass transport cloaking, 2018, arXiv preprint arXiv:1812.04157.
[37] A.D. Jagtap, E. Kharazmi, G.E. Karniadakis, Conservative physics-informed neural networks on discrete domains for conservation laws, Comput. Methods Appl. Mech. Engrg. 365 (2020) 113028.
[38] W.J. Morokoff, R.E. Caflisch, Quasi-Monte Carlo integration, J. Comput. Phys. 122 (2) (1995) 218–230.
[39] S. Smolyak, Quadrature and interpolation formulas for tensor products of certain classes of functions, Sov. Math. Dokl. 4 (1963) 240–243.
[40] E. Novak, K. Ritter, High dimensional integration of smooth functions over cubes, Numer. Math. 75 (1) (1996) 79–97.
[41] E.C. Cyr, M.A. Gulian, R.G. Patel, M. Perego, N.A. Trask, Robust training and initialization of deep neural networks: An adaptive basis viewpoint, 2019, arXiv preprint arXiv:1912.04862.
[42] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, 2014, arXiv preprint arXiv:1412.6980.
[43] G.E. Karniadakis, S.J. Sherwin, Spectral/*hp* Element Methods for Computational Fluid Dynamics, Oxford University Press, New York, 2013.
[44] L. Lu, X. Meng, Z. Mao, G.E. Karniadakis, DeepXDE: A deep learning library for solving differential equations, 2019, arXiv preprint arXiv:1907.04502.
[45] A.M. Bradley, PDE-Constrained Optimization and the Adjoint Method, 2010.
[46] Y. Cao, S. Li, L. Petzold, R. Serban, Adjoint sensitivity analysis for differential-algebraic equations: The adjoint DAE system and its numerical solution, SIAM J. Sci. Comput. 24 (3) (2003) 1076–1089.
[47] J. Tromp, C. Tape, Q. Liu, Seismic tomography, adjoint methods, time reversal and banana-doughnut kernels, Geophys. J. Int. 160 (1) (2005) 195–216.
[48] E. Kharazmi, M. Zayernouri, Fractional sensitivity equation method: Application to fractional model construction, J. Sci. Comput. 80 (1) (2019) 110–140.
[49] R. Plessix, A review of the adjoint-state method for computing the gradient of a functional with geophysical applications, Geophys. J. Int. 167 (2) (2006) 495–503.
[50] G. Allaire, A review of adjoint methods for sensitivity analysis, uncertainty quantification and optimization in numerical codes, Ing. Automob. 836 (2015) 33–36.
[51] T. van Leeuwen, F.J. Herrmann, A penalty method for PDE-constrained optimization in inverse problems, Inverse Problems 32 (1) (2015) 015007.

[52] M. Raissi, P. Perdikaris, G.E. Karniadakis, Machine learning of linear differential equations using Gaussian processes, J. Comput. Phys. 348 (2017) 683–693.
[53] M. Raissi, G.E. Karniadakis, Hidden physics models: Machine learning of nonlinear partial differential equations, J. Comput. Phys. 357 (2018) 125–141.
[54] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al., Tensorflow: A system for large-scale machine learning, in: 12th {USENIX} Symposium on Operating Systems Design and Implementation, {OSDI} 16, 2016, pp. 265–283.
[55] X. Meng, G.E. Karniadakis, A composite neural network that learns from multi-fidelity data: Application to function approximation and inverse PDE problems, J. Comput. Phys. 401 (2020) 109020.
[56] L. Yang, X. Meng, G.E. Karniadakis, B-PINNs: Bayesian physics-informed neural networks for forward and inverse PDE problems with noisy data, 2020, arXiv preprint arXiv:2003.06097.
[57] K. Xu, E. Darve, Physics constrained learning for data-driven inverse modeling from sparse observations, 2020, arXiv preprint arXiv:2002.10521.
[58] K. Xu, A.M. Tartakovsky, J. Burghardt, E. Darve, Inverse modeling of viscoelasticity materials using physics constrained learning, 2020, arXiv preprint arXiv:2005.04384.
[59] A. Mojtabi, M.O. Deville, One-dimensional linear advection–diffusion equation: Analytical and finite element solutions, Comput. & Fluids 107 (2015) 189–195.
[60] Y. Shin, J. Darbon, G.E. Karniadakis, On the convergence and generalization of physics informed neural networks, 2020, arXiv preprint arXiv:2004.01806.
[61] Y. Shin, Z. Zhang, G.E. Karniadakis, Error estimates of residual minimization using neural networks for linear PDEs, 2020, arXiv:2010.08019.