

hp-VPINNs: Variational Physics-Informed Neural Networks With Domain Decomposition

Ehsan Kharazmi^{a,*}, Zhongqiang Zhang^b, George Em 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.*

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 space of high-order polynomials. The *trial space* is the space of neural network, which is defined globally over the whole computational domain, while the *test space* contains the piecewise polynomials. Specifically in this study, the *hp*-refinement corresponds to a *global approximation with local learning* algorithm that can efficiently localize the network parameter optimization. We demonstrate the advantages of *hp*-VPINNs in accuracy and training cost for several numerical examples of function approximation and 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, 2, 3, 4, 5, 6, 7, 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, 14, 15, 16], adaptive splines [9], radial basis functions [17], Gaussian kernels [18], and neural networks [19, 20, 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] 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

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

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

*Corresponding author

Email address: ehsan_kharazmi@brown.edu (Ehsan Kharazmi)

where the bounded domain $\Omega \subset \mathbb{R}^d$ with boundaries $\partial\Omega$, $T > 0$, and $u(\mathbf{x}, t) : \Omega \times [0, T] \rightarrow \mathbb{R}$ describes the underlying physical phenomena modeled by the above governing equation. The operator \mathcal{L}^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 N_i neurons in each layer and activation function σ that takes the following form

$$u_{NN}(\mathbf{x}, t; \mathbf{W}, \mathbf{b}) = g \circ T^{(\ell)} \circ T^{(\ell-1)} \circ \dots \circ T^{(1)}(\mathbf{x}). \quad (1.3)$$

In the output layer, the linear mapping is $g : \mathbb{R}^{N_\ell} \rightarrow \mathbb{R}$, and in each hidden layer $i = 1, 2, \dots, \ell$, the nonlinear mapping is $T^{(i)}(\cdot) = \sigma(\mathbf{W}_i \times \cdot + \mathbf{b}_i)$ with weights $\mathbf{W}_i \in \mathbb{R}^{N_i \times N_{i-1}}$ and biases $\mathbf{b}_i \in \mathbb{R}^{N_i}$, where $N_0 = d$ 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

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

The residuals are measures to the extent to which the approximation \tilde{u} satisfies the equations (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

$$\mathcal{R}_j(\tilde{u}) = \int_{\Omega \times (0, T]} r(\tilde{u}) v_j \, dx \, dt = 0, \quad (1.5)$$

$$\mathcal{R}_{b,j}(\tilde{u}) = \int_{\partial\Omega \times (0, T]} r_b(\tilde{u}) v_j \, dx \, dt = 0, \quad (1.6)$$

$$\mathcal{R}_{0,j}(\tilde{u}) = \int_{\Omega} r_0(\tilde{u}) v_j \, dx = 0. \quad (1.7)$$

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

$$\min_{\mathbf{W}, \mathbf{b}} \mathcal{J}(\tilde{u}, v), \quad (1.8)$$

where

$$\mathcal{J}(\tilde{u}, v) = w \sum_{j=1}^{N_r} \mathcal{R}_j^2(\tilde{u}) + w_b \sum_{j=1}^{N_b} \mathcal{R}_{b,j}^2(\tilde{u}) + w_0 \sum_{j=1}^{N_0} \mathcal{R}_{0,j}^2(\tilde{u}). \quad (1.9)$$

The parameters $\{w, w_b, w_0\}$ 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].

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

test function	trial function	
v	\tilde{u}	$\tilde{u} = \text{DNNs}$
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

Different choices of trial function \tilde{u} and test function v_j in (1.9) correspond to various numerical methods. Most of these methods are well established and analyzed in the literature when linear approximations are used. Here, we

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

	local trial functions	global trial functions
local test functions	conservative VPINNs [36]	hp -VPINNs
global test functions	–	VPINNs [25] varNet [26], D3M [8]

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.

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]. Specifically, PINNs use the following functional

$$L^s = \frac{1}{N_r} \sum_{i=1}^{N_r} \left| r(\mathbf{x}_r^i, t_r^i) \right|^2 + \tau_b \frac{1}{N_b} \sum_{i=1}^{N_b} \left| r_b(\mathbf{x}_b^i, t_b^i) \right|^2 + \tau_0 \frac{1}{N_0} \sum_{i=1}^{N_0} \left| r_0(\mathbf{x}_0^i) \right|^2, \quad (1.10)$$

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 $\{\mathbf{x}_0^i\}_{i=1}^{N_0}$ are collocation points in their domains. We use the superscript s to refer to the loss function associated with the strong-form of the residual. The *deep Galerkin method* [23, 32] 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.

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

$$L^v = \frac{1}{K} \sum_{j=1}^K \left| \mathcal{R}_j \right|^2 + \tau_b \frac{1}{N_b} \sum_{i=1}^{N_b} \left| r_b(\mathbf{x}_b^i, t_b^i) \right|^2 + \tau_0 \frac{1}{N_0} \sum_{i=1}^{N_0} \left| r_0(\mathbf{x}_0^i) \right|^2, \quad (1.11)$$

that takes the test functions v_j ($1 \leq j \leq K$) from orthogonal polynomials. The superscript 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, 33] takes the test functions to be the piece-wise linear shape functions of finite element method, D3M [8] formulation includes the reformulation of problem (1.1)-(1.2) into a system of first-order equations, and WAN [34, 35] develops an adversarial framework by taking the test function to be a separate network.

In this paper, we develop the hp -VPINNs by taking a different set of test functions, which are non-overlapping on each sub-domain of the domain; see the next section for more details. Our formulation has the flexibility to construct both local and global approximations with locally/globally defined test functions. We show the comparison with other methods that use NN in Table 2. The flexibility of hp -VPINN formulation allows us to accommodate the singularities, steep solution, and sharp changes; see Section 5 for an example with corner singularity. Moreover, it allows us to adaptively select the orthogonal polynomials over the sub-domains with smooth solution. Thus, we may better balance the training cost by mainly focusing on optimizing the network parameters based on sub-domains with less regular solutions, leading to a localized learning process. For the integrals in (1.5) and (1.9), we may perform integration by parts to reduce the regularity requirement before applying numerical quadrature rules for further discretization. For elliptic problems, we test the effects of performing no integration by parts or performing it once or twice; see Sections 4 and 5.

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.

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_j , $j = 1, 2, \dots, N_{sd}$ of partition of the set V ($\Omega \times (0, T]$ or Ω in this work). The test function defined on a subset $V_j \subset V$ reads

$$v_j = \begin{cases} \bar{v} \neq 0, & \text{over } V_j, \\ 0, & \text{over } V_i^c, \end{cases} \quad V_j \cup V_i^c = V, \quad (2.1)$$

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

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

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

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

where $K^{(e)}$ is the total number of test functions in element e , the term $\mathcal{R}_k^{(e)}$ is the k -th 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 [37] or sparse grid quadratures [38, 39].

Remark 2.1. *Domain decomposition can provide the opportunity to assign the network optimization in each sub-domain to a specific computer node. 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 whole computational domain. In this setting, 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 [31, 30]. An adaptive basis viewpoint of DNNs is also given in [40].*

3. Variational Neural Networks (VNNs) for Function Approximation

Let us consider the problem of approximating the target function $u(\mathbf{x}) : \Omega \rightarrow \mathbb{R}$ 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

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

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 N neurons and tanh activation functions (if not specifically mentioned otherwise). We use Legendre polynomials as test functions, i.e. $v_k(x) = P_{k-1}(x)$, $k = 1, 2, \dots, K$. 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.

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.

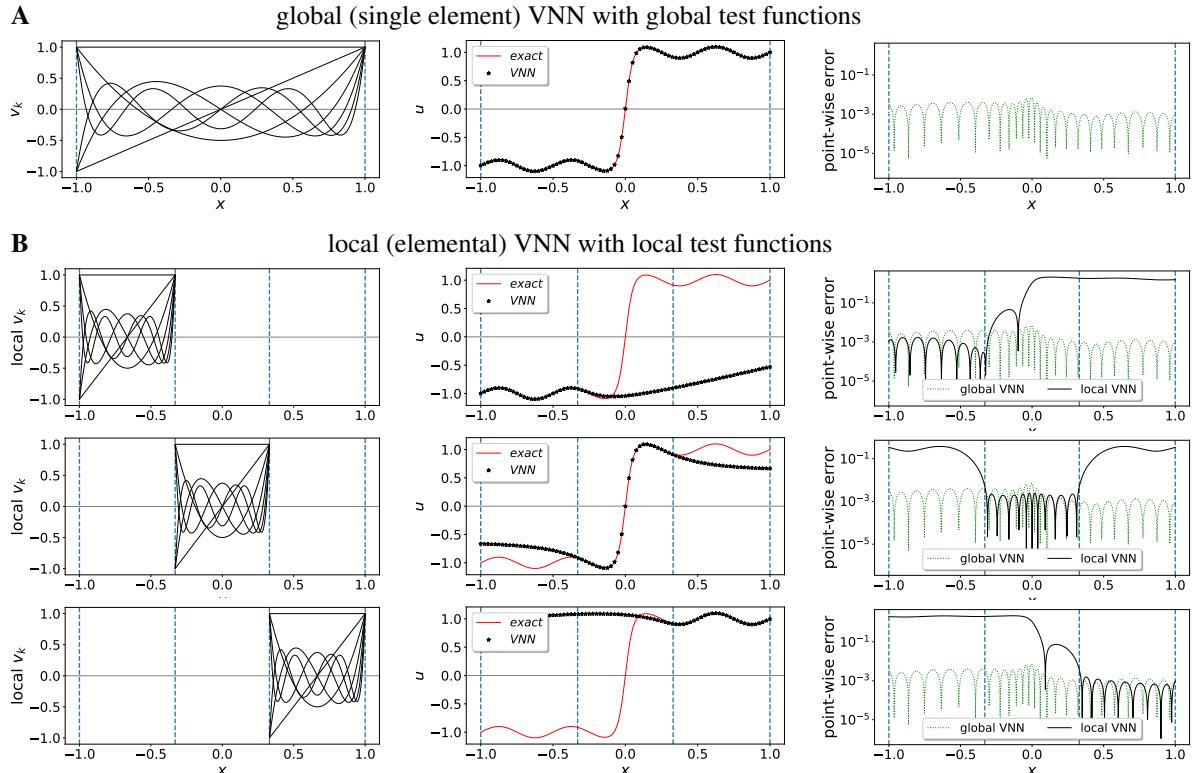


Figure 1: VNN continuous function approximation. (A) global VNN: (left) the test functions, defined over the whole computational domain; (middle) the exact function and VNN approximation; (right) point-wise error. (B) local VNN: (left column) the test functions, defined locally over the individual sub-domains;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

The considered function is smooth, continuous, and thus, can be approximated accurately using VNN with $N_{el} = 1$. Using $\ell = 4$, $N = 20$, $K = 60$, 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*

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

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.

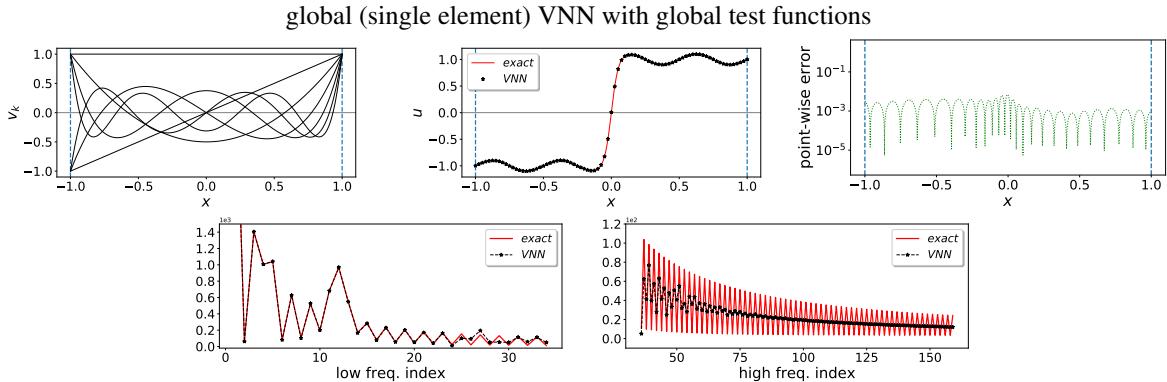


Figure 2: VNN discontinuous function approximation: global (single element) VNN. Top row from left: test functions defined over the whole computational domain, the exact function and VNN prediction, and point-wise error. Bottom row: the target function and prediction in low and high frequency domain.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

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. Figure 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 note 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 (Convergence By Depth). *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

local (elemental) VNN with local test functions

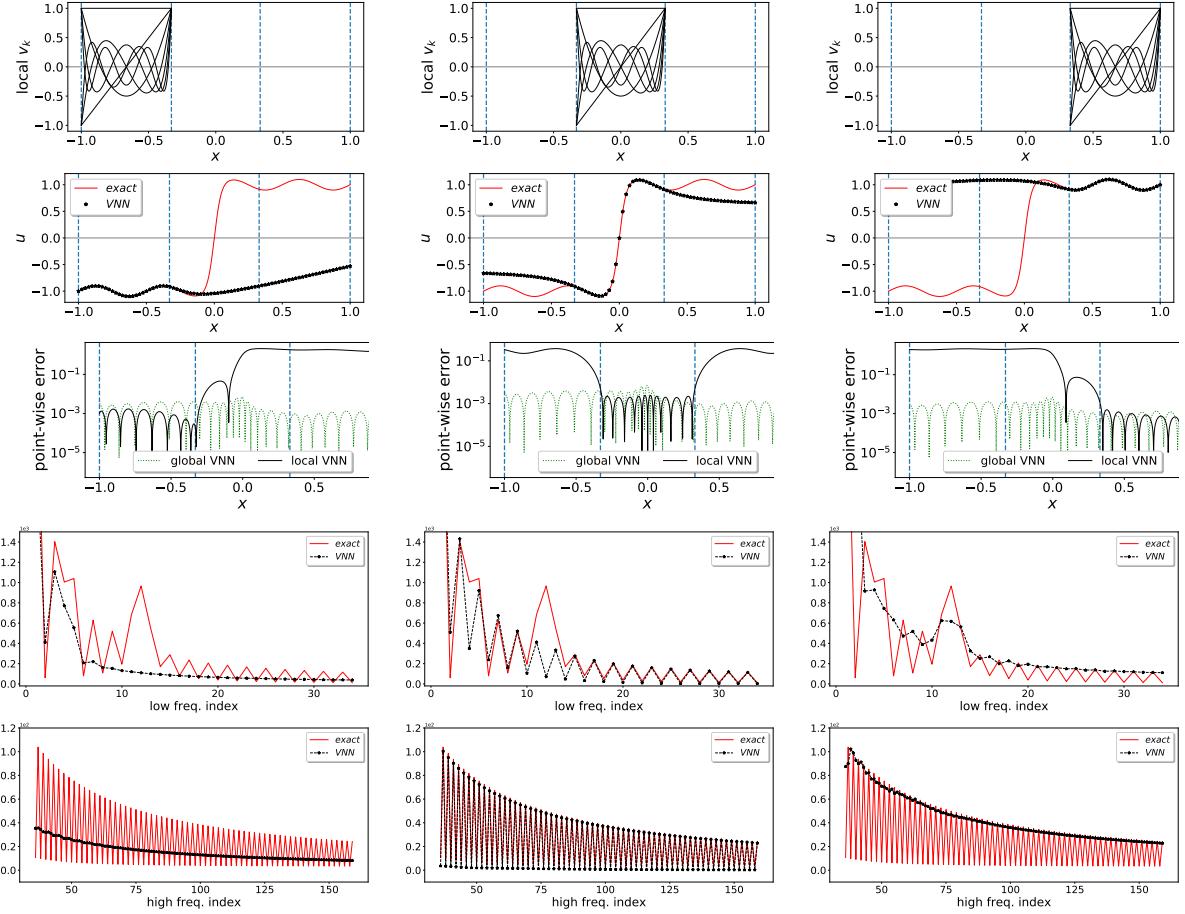


Figure 3: VNN discontinuous function approximation: local (elemental) VNN. The dashed blue line are the sub-domain boundaries. The row-wise captions are; First: locally defined test functions over each sub-domain. Second: the exact function and VNN prediction. Third: point-wise error. Fourth and fifth: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

multi elemental VNN with local test functions

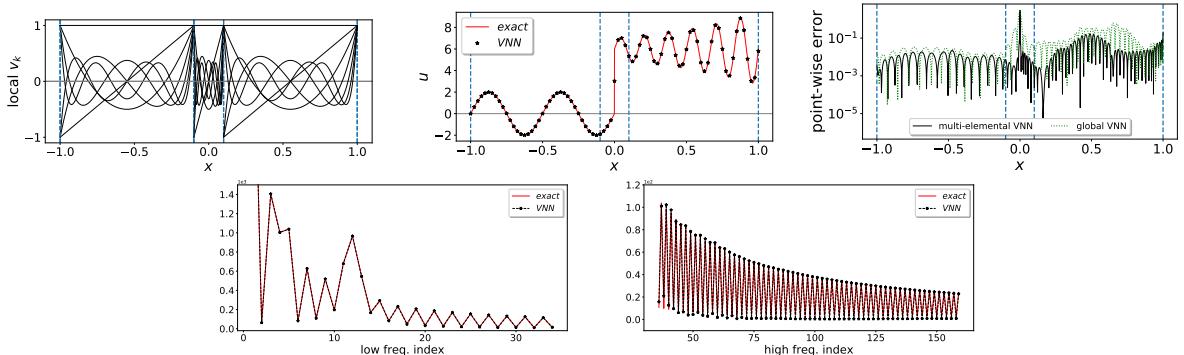


Figure 4: VNN discontinuous function approximation: multi-elemental VNN. Top left: the domain is divided into three sub-domains and the test functions are defined locally over each sub-domain. The dashed blue line are the sub-domains boundaries. Top middle: the exact function and VNN prediction. Top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

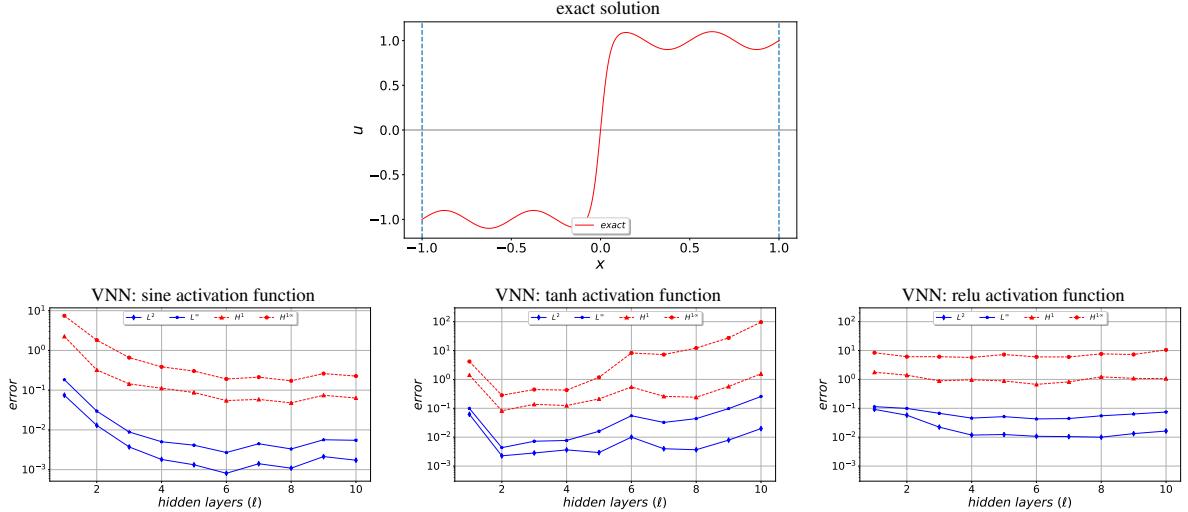


Figure 5: Global (single element) VNN prediction of the exact function in 3.1 (Top) and error convergence with network depth (bottom) for sine, tanh, and ReLu activation functions. VNN is a fully connected network with parameters $N = 20$, $K = 60$, $Q = 80$, and Adam optimizer with learning rate 10^{-3}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

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 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. Figure 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 $u(x) : \Omega \rightarrow \mathbb{R}$, where $\Omega = [-1, 1]$. We consider the Poisson's equation given as

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

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

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

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

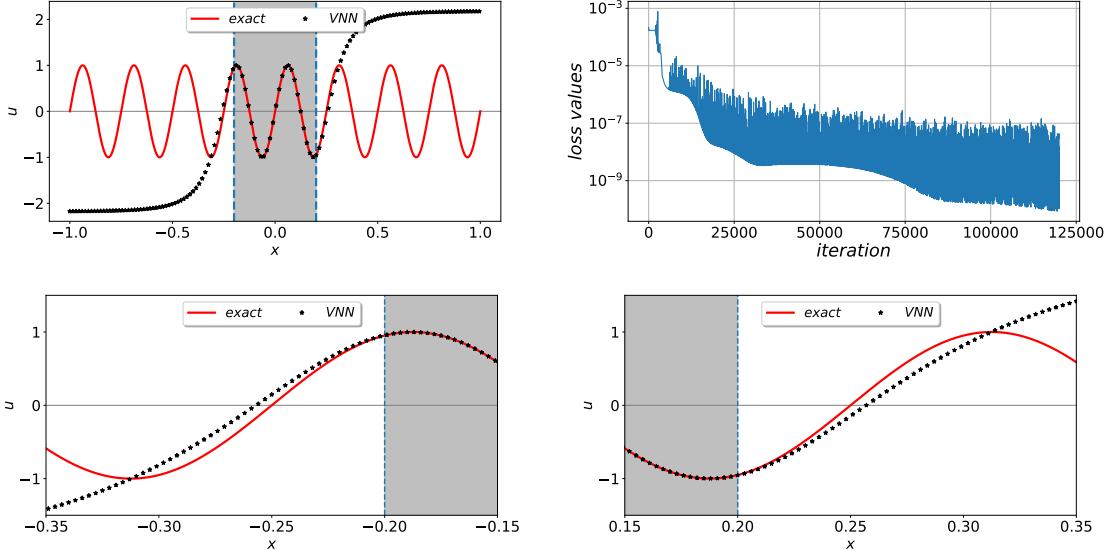


Figure 6: VNN function approximation: learning out of the local element. The top left panel shows the exact function and VNN prediction. The shaded area is the sub-domain over which the test functions are locally defined and the network is trained. The bottom row shows the zoomed-in frame of the left and right boundaries of the sub-domain. The VNN parameters are $\ell = 4$, $N = 20$, $K = 60$, $Q = 80$, and Adam optimizer with learning rate 10^{-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 $\{-1 = x_0, x_1, \dots, x_{N_{el}} = 1\}$. We choose a set of localized nonoverlapping test functions $v_k(x)$, given in (2.1) with the nonvanishing function be the high-order polynomials. The variational residual then becomes

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

where, in each term $\mathcal{R}_k^{(e)}$, $e = 1, 2, \dots, N_{el}$, 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,

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

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

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

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

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

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 $N = 20$ 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 its automatic differentiation capability. We also use the extended stochastic gradient descent Adam algorithm [41] to optimize the loss function.

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

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

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

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

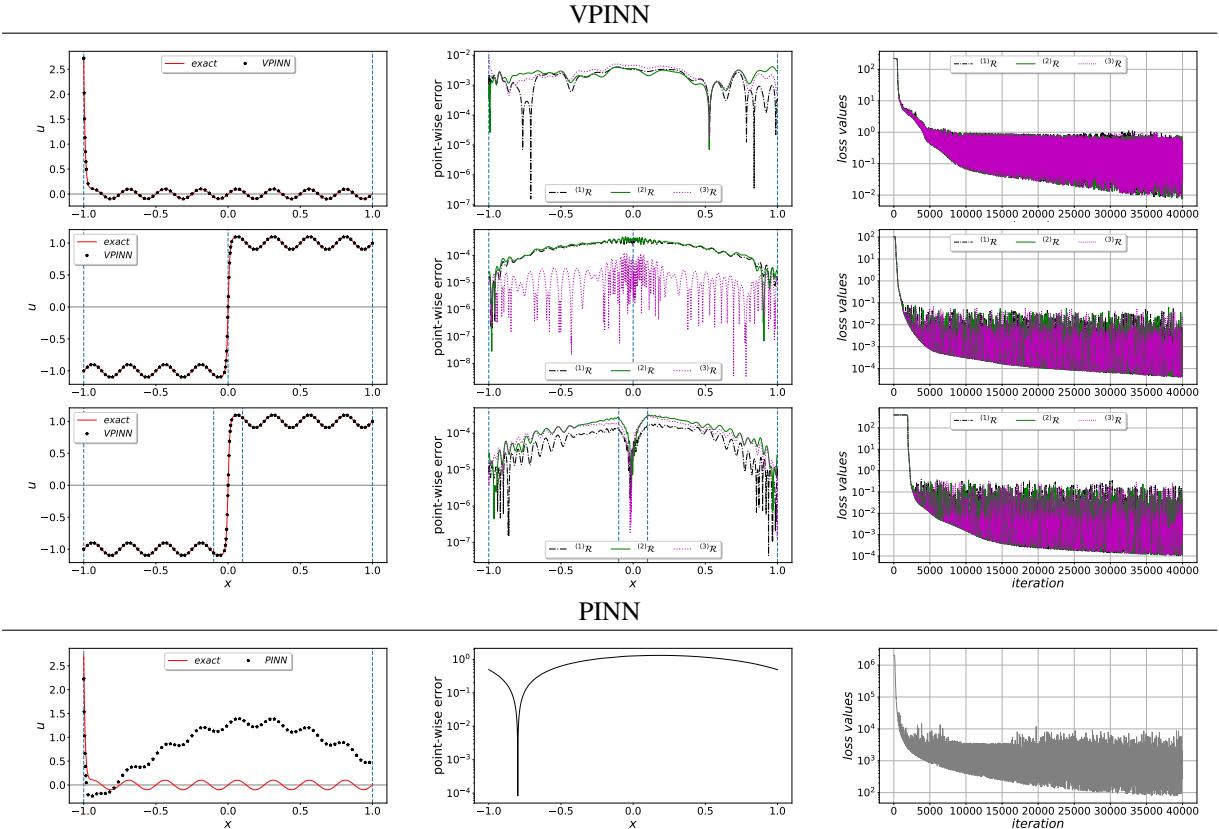


Figure 7: One-dimensional Poisson's equation with steep solution (4.9): comparison of VPINN and PINN. Top panel shows the h -refinement of VPINN with single element (first row), two elements (second row), and three elements (third row). Column-wise captions: (left) the exact solution and VPINN prediction,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

Figure 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. 8. 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 note that in the steep and boundary layer cases, the force term becomes very large, leading to a large loss value initially, which may sometimes result in an optimization failure. Unlike VPINN, we need to give a higher weight to the boundary term in the loss function in PINN to make sure that the network learns the boundary correctly.

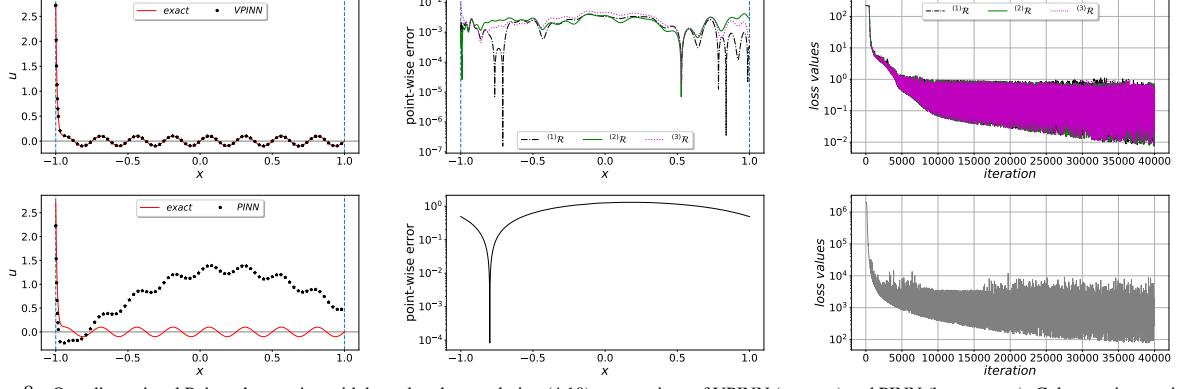


Figure 8: One-dimensional Poisson's equation with boundary layer solution (4.10): comparison of VPINN (top row) and PINN (bottom row). Column-wise captions: (left) the exact solution and VPINN prediction, (middle) point-wise error, (right) loss function versus training iterations. The VPINN parameters 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} .

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

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

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. 9.

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 $u(x, y) : \Omega \rightarrow \mathbb{R}$, where $\Omega = [-1, 1] \times [-1, 1]$. We consider the two-dimensional Poisson's equation

$$\nabla^2 u(x, y) = f(x, y), \quad (5.1)$$

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

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

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

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

The variational residual then reads as

$$\mathcal{R}_{k_1 k_2} = (\nabla^2 u_{NN}(x, y) - f(x, y), v_{k_1 k_2}(x, y))_\Omega. \quad (5.4)$$

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 $\Omega_{e_x e_y} = [x_{e_x-1}, x_{e_x}] \times [y_{e_y-1}, y_{e_y}]$, $e_x = 1, 2, \dots, N_{el_x}$, $e_y = 1, 2, \dots, N_{el_y}$. Therefore, the variational residual becomes

$$\mathcal{R}_{k_1 k_2} = \sum_{e_x=1}^{N_{el_x}} \sum_{e_y=1}^{N_{el_y}} \mathcal{R}_{k_1 k_2}^{(e_x e_y)}, \quad (5.5)$$

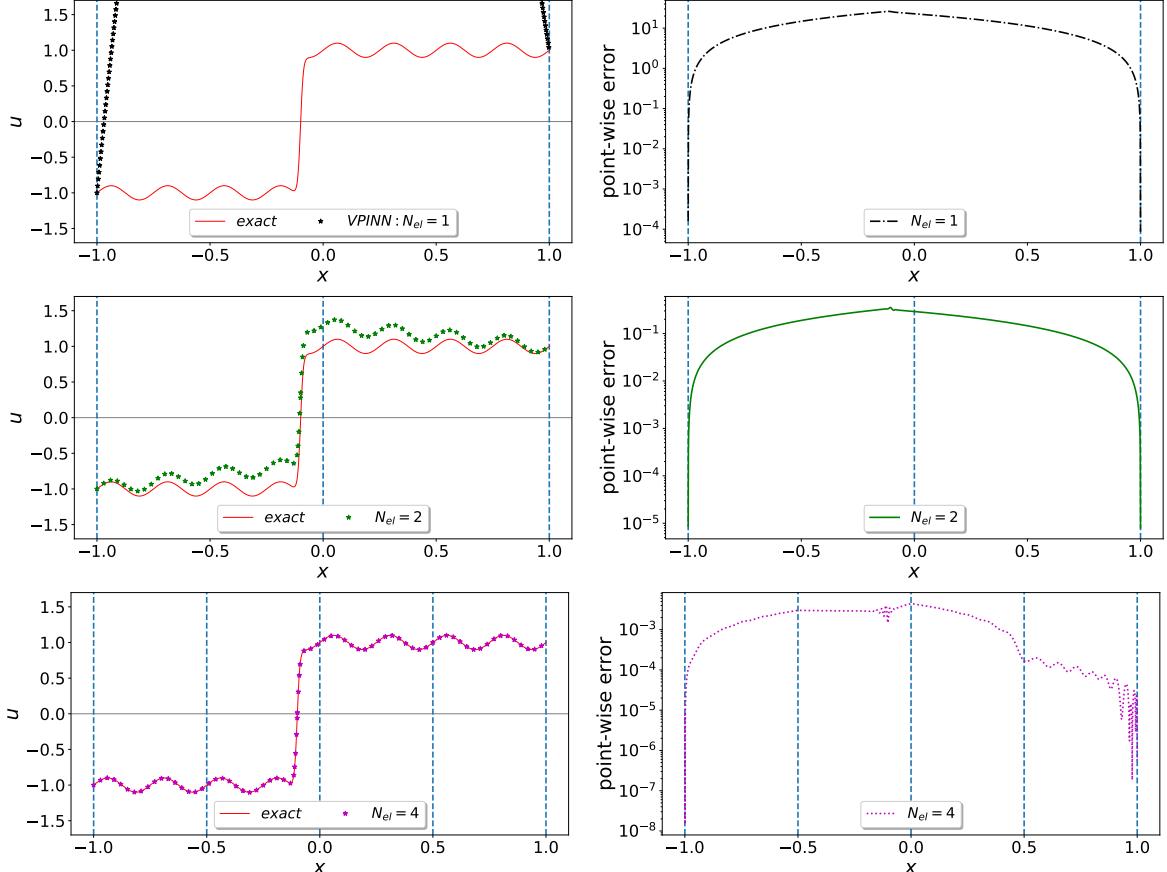


Figure 9: One-dimensional Poisson's equation with asymmetric steep solution (4.11). The location of sharp change is not known a priori and is predicted by increasing number of elements in VPINN. Left column: the exact solution and VPINN prediction. Right column: point-wise error. The VPINN is based on the $(^{(1)}\mathcal{R})$ formulation and has the parameters $\{\ell = 4, N = 20, K = 60, Q = 80, \tau_b = 1\}$. The network is fully connected with sine activation function, and we use Adam optimizer with learning rate 10^{-3} .

$$\mathcal{R}_{k_1 k_2}^{(e_x e_y)} = \int_{x_{e_x-1}}^{x_{e_x}} \int_{y_{e_y-1}}^{y_{e_y}} (\nabla^2 u_{NN}(x, y) - f(x, y)) \phi_{k_1}^{(e_x)}(x) \phi_{k_2}^{(e_y)}(y) dx dy,$$

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 similar to the one-dimensional case, the local test functions $v_{k_1 k_2}^{(e)}(x, y)$ have the compact support over Ω_e . Therefore, for all elements, we have

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

By integrating by parts in the first term of $\mathcal{R}_{k_1 k_2}^{(e_x e_y)}$, we can define the following variational residual forms, in which the term $F_{k_1 k_2}^{(e_x e_y)}$ is associated with the integral of the force term $f(x, y)$.

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

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

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

We reduce the order of tensor $\mathcal{R}_{k_1 k_2}^{(e_x e_y)}$ to one by stacking its entries into vector of size $K_1 K_2$. Subsequently, we define the *variational loss function* as

$$L^{v(i)} = \sum_{e_x=1}^{N_{elx}} \sum_{e_y=1}^{N_{ely}} \frac{1}{K_1 K_2} \sum_{k=1}^{K_1 K_2} \left| {}^{(i)}\mathcal{R}_k^{(e_x e_y)} \right|^2 + \tau_b \frac{1}{N_b} \sum_{i=1}^{N_b} \left| r_b(x_b^i, y_b^i) \right|^2, \quad i = 1, 2, 3, \tag{5.10}$$

where the term ${}^{(i)}\mathcal{R}_k^{(e_x e_y)}$ is the k -th 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 [41] 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. 10.

In this case, the exact solution is smooth and thus an accurate approximation can be obtained by using a relatively small network with $\ell = 3$, $N = 5$, 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. 11.

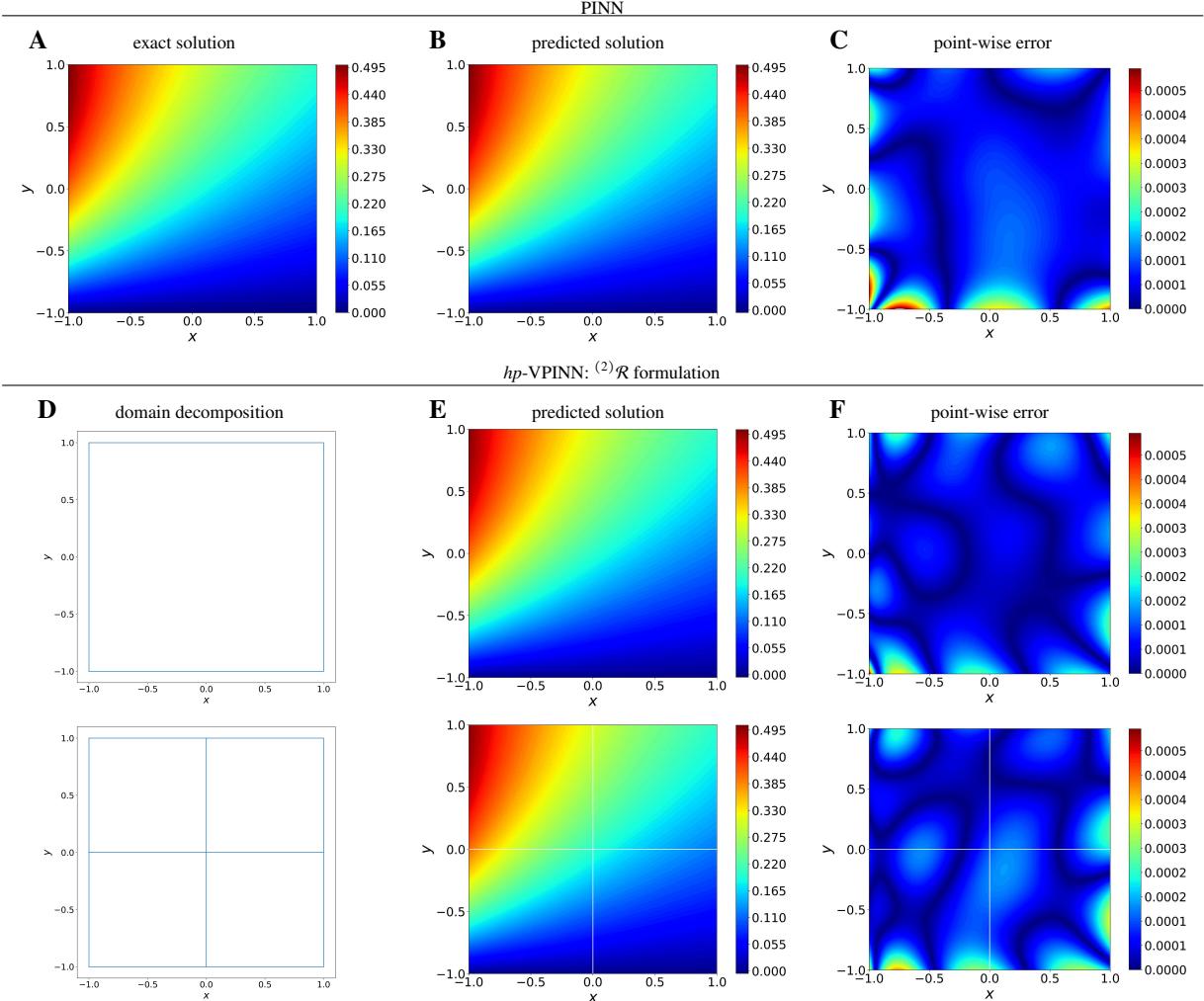


Figure 10: Two-dimensional homogeneous Poisson's equation. Top panel: (A) Exact solution (5.11), (B) PINN prediction, and (C) PINN point-wise error. Bottom panel: (D) column h -refinement via domain decomposition $N_{el_x} = N_{el_y} = 1$ and 2, (E) column hp -VPINN prediction, and (F) column hp -VPINN 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} .

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. Figure 11 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. 12. For comparison, we also present the numerical solution obtained by using the spectral element method (SEM) [42]. The solution and comparison of PINN with SEM is given in [43].

The exact solution is not available in this case and thus we consider the SEM solution [42, 43] 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 [43] 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 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. 12. Then, in a fine decomposition, we divide the domain into total of 35 sub-domains with different sizes. 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 $\ell = 3$, $N = 20$, 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)

Let $u(t, x) : \Omega \rightarrow \mathbb{R}$, 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 coefficient 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 [44] 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) $\Omega_{e_t e_x} = [t_{e_t-1}, t_{e_t}] \times [x_{e_x-1}, x_{e_x}]$, $e_t = 1, 2, \dots, N_{el_t}$, $e_x = 1, 2, \dots, N_{el_x}$ via constructing the temporal and spatial grids $\{0 = t_0, t_1, \dots, t_{N_{el_t}} = 1\}$ and $\{-1 = x_0, x_1, \dots, x_{N_{el_x}} = 1\}$, respectively. Figure 13 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.

• **Diffusivity Estimation.** The inverse problem is defined as: given a (sparse observation/measurement of) solution of mathematical model, one seeks to obtain an accurate estimation of the corresponding model parameters. It has been shown in [45, 46] that the PINN formulation can incorporate the model parameters into the neural network parameters. This provides a framework to take the advantage of iterative methods in the context of neural network by letting the training algorithm to simultaneously optimize neural network and model parameters. By following a similar approach, we use VPINNs to solve the inverse problem for parameter estimation. We note that the VPINN formulation does not necessarily perform more accurately when compared to PINN formulation in solving the inverse problem; yet, we intent to show that it has this capability.

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 $u^*(t^*, x^*)$ is only available as time series at three (sensor) locations along the x axis, i.e., $x^* = \{-0.5, 0, 0.5\}$. We randomly select 5 data points at each sensor, and thus in total 15 measurements all over the whole domain; an example of these points is shown as black squares in Fig. 14. We pose the inverse problem diffusivity estimation as follows:

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

The results are shown in Fig. 14.

The additional data points from given measurements/observations in the inverse problem are added as the following extra term in the variational loss function (1.11)

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

We use the VPINN formulation by constructing a fully connected neural network with tanh activation function, parameters $\ell = 3, N = 5, K_1 = K_2 = 5, Q = 10 \times 10, \tau_b = \tau_0 = \tau^* = 10, N_b = 160, N_i = 80, N^* = 15$, 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^* points. The convergence of mean value of κ , its standard deviation, and also values of loss function are shown in Fig. 14. We observe that after convergence of diffusion coefficients, the point-wise error has only a large magnitude close to the right boundary at $t = 1$.

7. Summary

We developed the hp -VPINN formulation to solve differential equations in the context of sub-domain Petrov-Galerkin method. The trial space is the space of neural networks and test space is the space of localized non-overlapping high order polynomials. We formulated the method in a general form, following the method of weighted residuals, where different choices of test functions lead to different numerical methods. We showed the efficiency and accuracy of hp -VPINNs, compared to other methods, in several examples of function approximation and solving differential equations. We developed the method in detail for one- and two-dimensional problems and obtained the corresponding variational loss functions. We discussed the hp -refinement and convergence of solution in solving equations with non-smooth solution. Moreover, we examined the efficiency of hp -VPINNs in solving the inverse problem of parameter estimation in advection diffusion equation. For time-dependent problems and long-time integration, it may be more efficient to develop a discrete in time version of hp -VPINN as was done in [3].

Acknowledgement

This work was supported by the Applied Mathematics Program within the Department of Energy on the PhILMs project (DE-SC0019453).

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. doi:<https://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, *Communications in Mathematics and Statistics* 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, *Journal of Computational Physics* 378 (2019) 686–707.
- [4] Z. Mao, A. D. Jagtap, G. E. Karniadakis, Physics-informed neural networks for high-speed flows, *Computer Methods in Applied Mechanics and Engineering* 360 (2020) 112789.
- [5] Y. Khoo, J. Lu, L. Ying, Solving for high-dimensional committor functions using artificial neural networks, *Research in the Mathematical Sciences* 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, *Computer Methods in Applied Mechanics and Engineering* 362 (2020) 112790.
- [7] Y. Liao, P. Ming, Deep Nitsche method: Deep Ritz method with essential boundary conditions, *arXiv preprint arXiv:1912.01309* (2019).
- [8] K. Li, K. Tang, T. Wu, Q. Liao, D3M: A deep domain decomposition method for partial differential equations, *arXiv e-prints* (2019) arXiv:1909.12236arXiv:1909.12236.

- [9] R. A. DeVore, Nonlinear approximation, *Acta Numerica* 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, arXiv preprint arXiv:1602.00203 (2016).
- [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.
- [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 Processing Magazine* 25 (2) (2008) 21–30.
- [17] R. DeVore, A. Ron, Approximation using scattered shifts of a multivariate function, *Transactions of the American Mathematical Society* 362 (12) (2010) 6205–6229.
- [18] T. Hangelbroek, A. Ron, Nonlinear approximation using Gaussian kernels, *Journal of Functional Analysis* 259 (1) (2010) 203–219.
- [19] H. N. Mhaskar, C. A. Micchelli, Approximation by superposition of sigmoidal and radial basis functions, *Advances in Applied Mathematics* 13 (3) (1992) 350–373.
- [20] H. N. Mhaskar, T. Poggio, Function approximation by deep networks, arXiv preprint arXiv:1905.12882 (2019).
- [21] I. Daubechies, R. DeVore, S. Foucart, B. Hanin, G. Petrova, Nonlinear approximation and (deep) ReLU networks, arXiv preprint arXiv:1905.02199 (2019).
- [22] B. A. Finlayson, L. E. Scriven, The method of weighted residualsA review, *Applied Mechanics Review* 19 (9) (1966) 735–748.
- [23] J. Sirignano, K. Spiliopoulos, DGM: A deep learning algorithm for solving partial differential equations, *Journal of Computational Physics* 375 (2018) 1339–1364.
- [24] L. Yang, D. Zhang, G. E. Karniadakis, Physics-informed generative adversarial networks for stochastic differential equations, arXiv preprint arXiv:1811.02033 (2018).
- [25] E. Kharazmi, Z. Zhang, G. E. Karniadakis, Variational physics-informed neural networks for solving partial differential equations, arXiv preprint arXiv:1912.00873 (2019).
- [26] R. Khodayi-Mehr, M. M. Zavlanos, VarNet: Variational neural networks for the solution of partial differential equations, arXiv preprint arXiv:1912.07443 (2019).
- [27] S. Wang, Y. Teng, P. Perdikaris, Understanding and mitigating gradient pathologies in physics-informed neural networks, arXiv preprint arXiv:2001.04536 (2020).
- [28] M. Raissi, H. Babaei, P. Givi, Deep learning of turbulent scalar mixing, *Physical Review Fluids* 4 (12) (2019) 124501.
- [29] G. Pang, L. Lu, G. E. Karniadakis, fPINNs: Fractional physics-informed neural networks, *SIAM Journal on Scientific Computing* 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, *Journal of Computational Physics* (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, arXiv preprint arXiv:1909.12228 (2019).
- [32] 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, arXiv preprint arXiv:1912.01455 (2019).
- [33] R. Khodayi-mehr, M. M. Zavlanos, Deep learning for robotic mass transport cloaking, arXiv preprint arXiv:1812.04157 (2018).
- [34] Y. Zang, G. Bao, X. Ye, H. Zhou, Weak adversarial networks for high-dimensional partial differential equations, arXiv preprint arXiv:1907.08272 (2019).
- [35] G. Bao, X. Ye, Y. Zang, H. Zhou, Numerical solution of inverse problems by weak adversarial networks, arXiv preprint arXiv:2002.11340 (2020).
- [36] A. D. Jagtap, E. Kharazmi, G. E. Karniadakis, Conservative physics-informed neural networks on discrete domains for conservation laws, *Computer Methods in Applied Mechanics and Engineering - under revision* (2020).
- [37] W. J. Morokoff, R. E. Cafisch, Quasi-Monte Carlo integration, *Journal of Computational Physics* 122 (2) (1995) 218–230.
- [38] S. Smolyak, Quadrature and interpolation formulas for tensor products of certain classes of functions, *Soviet Math. Dokl.* 4 (1963) 240–243.
- [39] E. Novak, K. Ritter, High dimensional integration of smooth functions over cubes, *Numerische Mathematik* 75 (1) (1996) 79–97.
- [40] 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, arXiv preprint arXiv:1912.04862 (2019).
- [41] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980 (2014).
- [42] G. E. Karniadakis, S. J. Sherwin, *Spectral/hp Element Methods for Computational Fluid Dynamics*, Oxford University Press, New York, 2013.
- [43] L. Lu, X. Meng, Z. Mao, G. E. Karniadakis, DeepXDE: A deep learning library for solving differential equations, arXiv preprint arXiv:1907.04502 (2019).
- [44] A. Mojtabi, M. O. Deville, One-dimensional linear advection–diffusion equation: Analytical and finite element solutions, *Computers & Fluids* 107 (2015) 189–195.
- [45] M. Raissi, P. Perdikaris, G. E. Karniadakis, Machine learning of linear differential equations using Gaussian processes, *Journal of Computational Physics* 348 (2017) 683–693.
- [46] M. Raissi, G. E. Karniadakis, Hidden physics models: Machine learning of nonlinear partial differential equations, *Journal of Computational Physics* 357 (2018) 125–141.

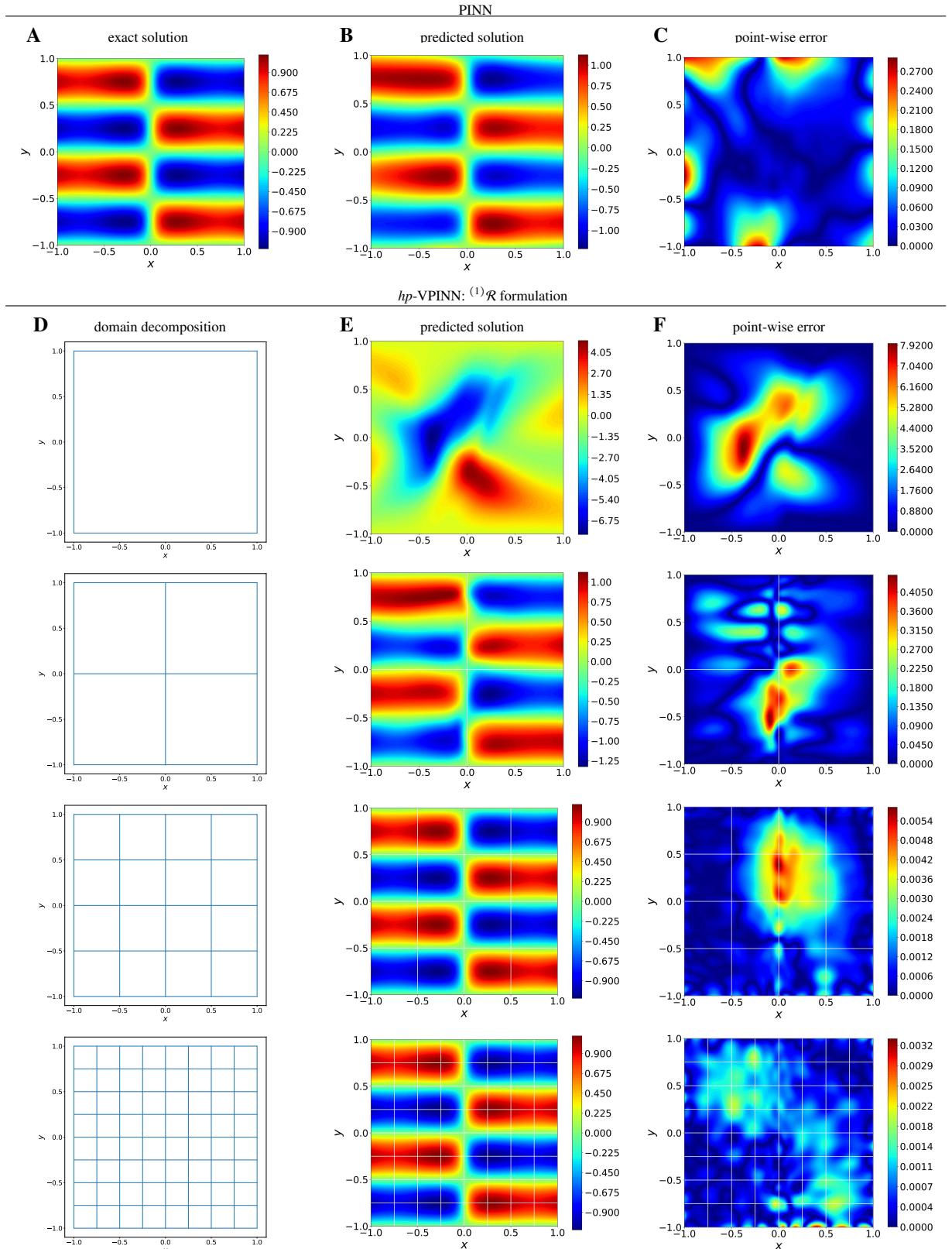


Figure 11: Two-dimensional Poisson's equation with steep exact solution. Top panel: (A) Exact solution (5.12), (B) PINN prediction, and (C) PINN point-wise error. Bottom panel: (D column) h -refinement via domain decomposition $N_{el_x} = N_{el_y} = 1, 2, 4$, and 8 , (E column) hp -VPINN prediction, and (F column) hp -VPINN point-wise error. In all cases, the network is fully connected with $\ell = 3$, $\mathcal{N} = 18$ 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} .

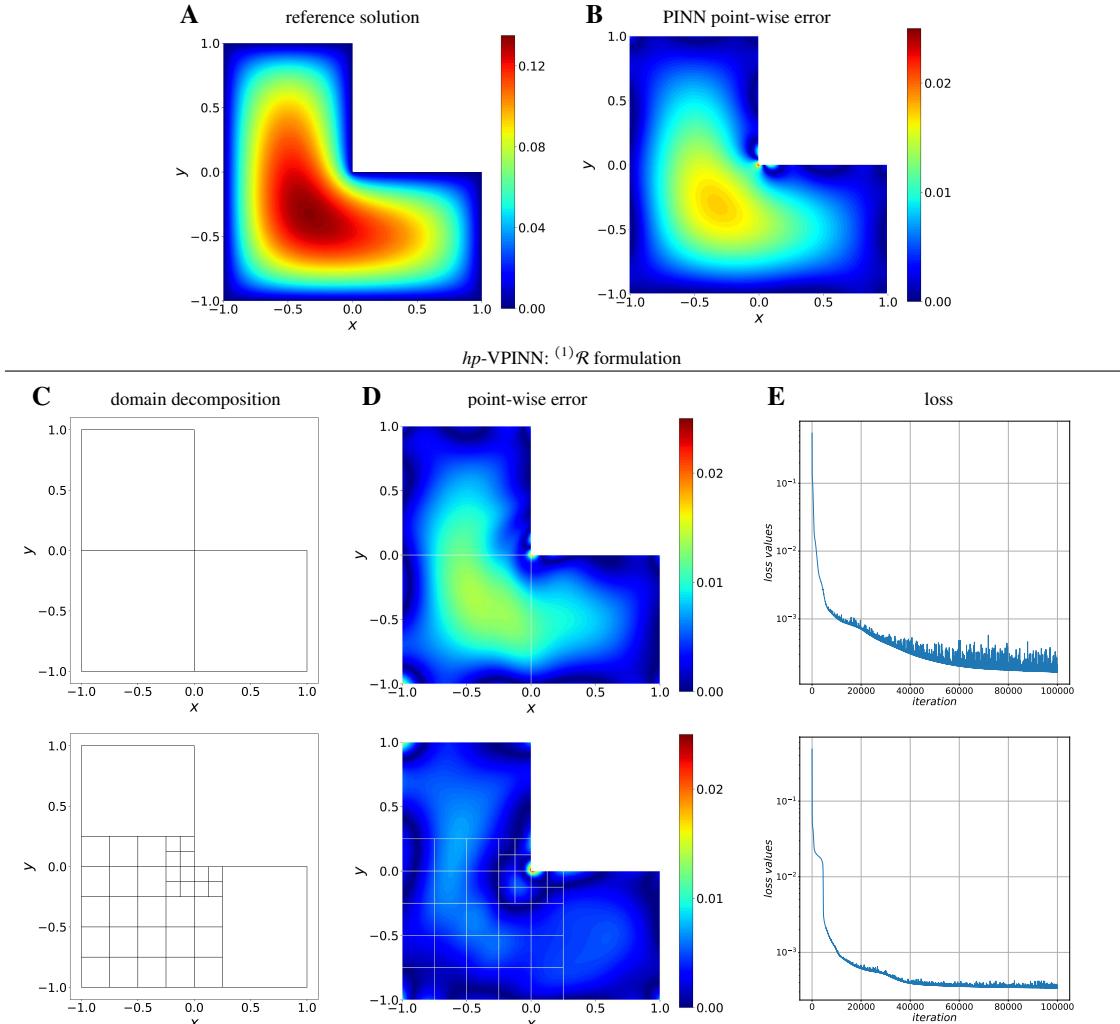


Figure 12: Two-dimensional homogeneous Poisson's equation in L-shaped domain. Top panel: (A) The reference solution u_{SEM} , and (B) PINN point-wise error $|u_{\text{NN}} - u_{\text{SEM}}|$. Bottom panel: (C column) Adaptive h -refinement via domain decomposition, (D column) hp -VPINN point-wise error $|u_{\text{NN}} - u_{\text{SEM}}|$, and (E column) loss value versus training iterations. The hp -VPINN is fully connected with $\ell = 3$, $N = 5$, 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} .

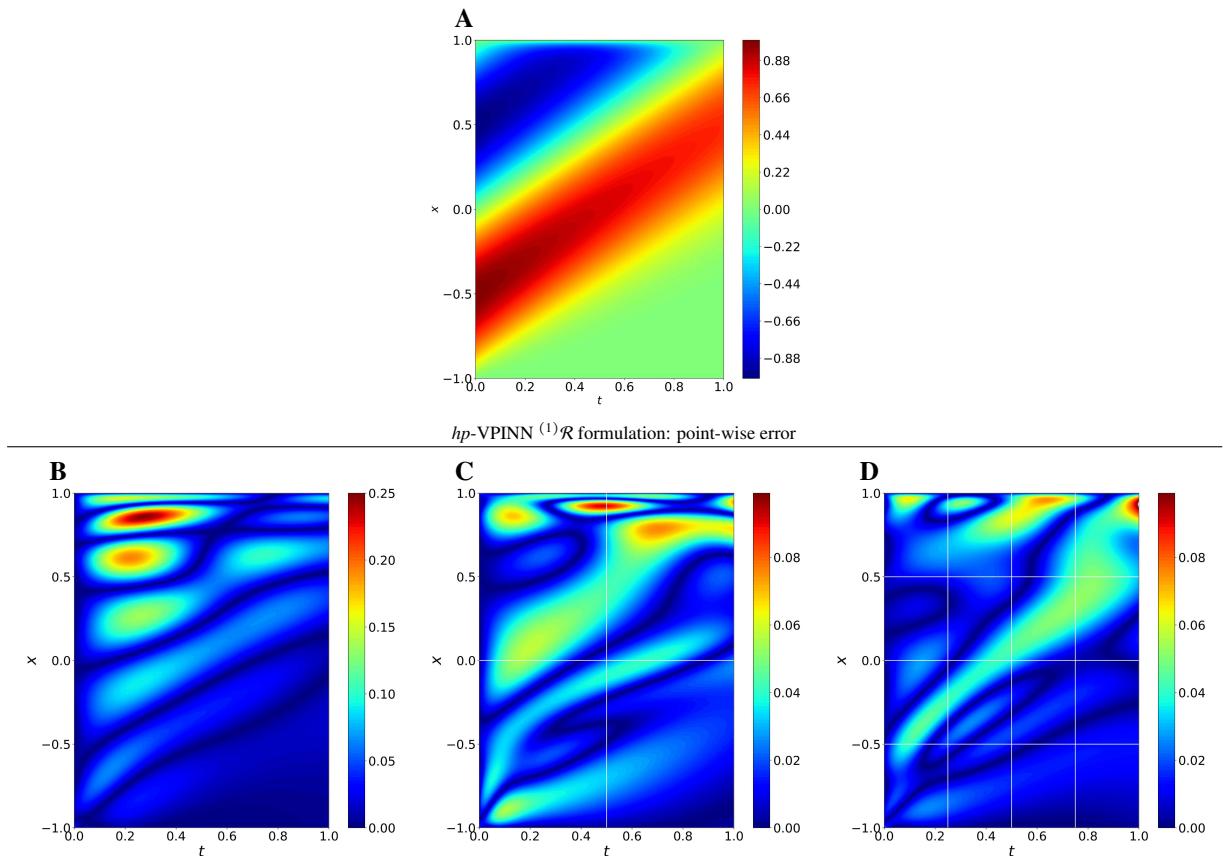


Figure 13: (1+1)-dimensional advection diffusion equation. (A) the exact solution. (B, C, and D) The h -refinement of hp -VPINN with $^{(1)}\mathcal{R}$ formulation for $N_{el_t} = N_{el_x} = 1, 2$, and 4 . In all cases, the network is fully connected with $\ell = 3$, $N = 5$, and tanh activation function. 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} .

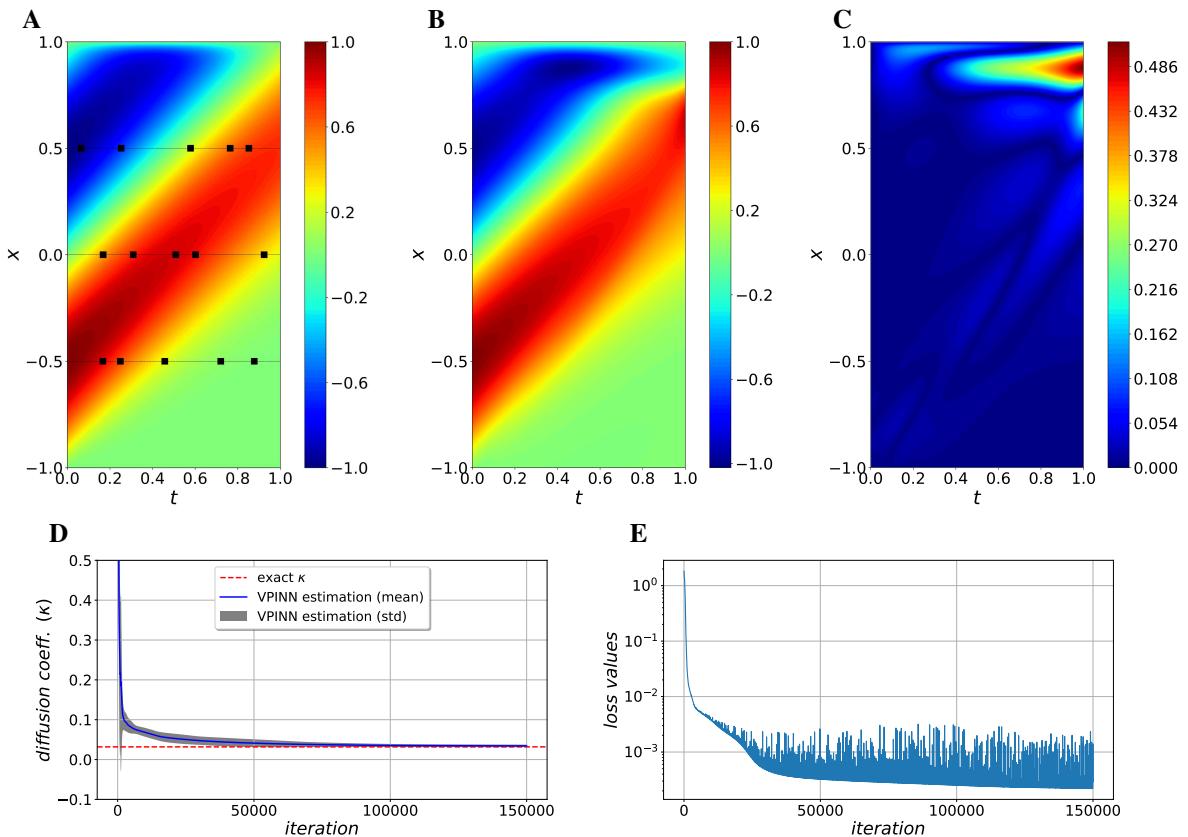


Figure 14: Diffusivity estimation in (1+1)-dimensional ADE. (A) exact solution and one realization of $N^* = 15$ randomly selected points at the sensor locations. (B) VPINN prediction. (C) VPINN point-wise error. (D) convergence of diffusivity coefficient and (E) loss values versus training iterations.