algorithms

Article

Meshfree Variational-Physics-Informed Neural Networks (MF-VPINN): An Adaptive Training Strategy

Stefano Berrone [1] and Moreno Pintore [2,3,*]

[1] Dipartimento di Scienze Matematiche, Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Torino, Italy; stefano.berrone@polito.it
[2] MEGAVOLT Team, Inria, 48 Rue Barrault, 75013 Paris, France
[3] Laboratoire Jacques-Louis Lions, Sorbonne Center for Artificial Intelligence, Sorbonne Université, 4 Place Jussieu, 75005 Paris, France
* Correspondence: moreno.pintore@inria.fr

Abstract: In this paper, we introduce a Meshfree Variational-Physics-Informed Neural Network. It is a Variational-Physics-Informed Neural Network that does not require the generation of the triangulation of the entire domain and that can be trained with an adaptive set of test functions. In order to generate the test space, we exploit an a posteriori error indicator and add test functions only where the error is higher. Four training strategies are proposed and compared. Numerical results show that the accuracy is higher than the one of a Variational-Physics-Informed Neural Network trained with the same number of test functions but defined on a quasi-uniform mesh.

Keywords: VPINN; meshfree; Physics-Informed Neural Networks; error estimator; patches

MSC: 65N12; 65N15; 65N50; 68T05; 92B20

check for updates

Citation: Berrone, S.; Pintore, M. Meshfree Variational-Physics-Informed Neural Networks (MF-VPINN): An Adaptive Training Strategy. *Algorithms* **2024**, *17*, 415. https://doi.org/10.3390/a17090415

Academic Editors: Shuai Li and Dunhui Xiao

Received: 22 July 2024
Revised: 8 September 2024
Accepted: 11 September 2024
Published: 19 September 2024

1. Introduction

Physics-Informed Neural Networks (PINNs) are a rapidly emerging numerical technique used to solve partial Differential equations (PDEs) by means of a deep neural network. The first idea can be traced back to the works of Lagaris et al. [1–3], but, thanks to the hardware advancements and the existence of deep learning packages like Tensorflow [4], Pytorch [5] and JAX [6], they have recently became popular since the works of Raissi et al. [7,8], published in [9]. In its original formulation, the approximate solution is computed as the output of a neural network trained to minimize the PDE residual on a set of collocation points inside the domain and on its boundary.

The growing interest in PINNs is strictly related to their flexibility. In fact, with minor changes to the implementation, it is possible to solve a huge variety of problems. For example, exploiting the nonlinear nature of the involved neural network, nonlinear [10,11] and high-dimensional PDEs [12] can be solved without the need for globalization methods or additional nonlinear solvers. Moreover, by changing the neural network's input dimensions or suitably adapting the loss function, it is possible to solve parametric [13,14] or inverse [15,16] problems. When external data are available, they can also be used to guide the optimization phase and improve the PINN accuracy [17].

In order to improve the original PINN proposed in [9] and to adapt it to solve specific problems, several generalizations have been proposed. For example, the deep Ritz method (DRM) [18–20] looks for a minimizer of the PDE energy functional and, in the deep Galerkin method (DGM) [21–23], an approximation of the L^2 norm of the PDE residual is minimized. It is also possible to exploit domain decomposition strategies [24,25] as in the conservative PINN (CPINN) [26], in the parallel PINN [27], in the extended PINN (XPINN) [28], or in the Finite Basis PINN (FBPINN) [29]. Moreover, it is even possible to change the neural network architecture or the training strategy as in [14,30–35]; between the methods based on different architectures, we highlight some works based on the novel Kolmogorov–Arnold Network (KAN) [36] architecture [37,38] and on a Large Language Model (LLM) [39]. More extensive overviews of the existing approaches can be found in [40–43]. In the context of the current work, an important extension is the Variational-Physics-Informed Neural Network (VPINN) [44,45], where the weak formulation of the problem is used to construct the loss function.

In this work, we focus on VPINNs. As discussed in [44–47], in order to train a VPINN, one needs to choose a suitable space of test functions, compute the variational residuals against all the test functions on the basis of such a space, and minimize a linear combination of these residuals. Since a spatial mesh is required to define the test functions, the VPINN cannot be considered a meshfree method, even though it is an extension of the PINN, which is meshfree. In this work, we present an adaptive Meshfree VPINN (MF-VPINN) that does not require a global triangulation of the domain but is trained with the same loss function and neural network architecture of a standard VPINN. Note that the MF-VPINN and the original VPINN can solve the same differential problems because the neural network is trained with the same loss functions. We also highlight that they can solve problems where the solution has low regularity that cannot be solved with standard PINNs, for example, in the presence of singular forcing terms, thanks to the weak formulation of the PDE without introducing further approximations or regularizations. However, one of the VPINN's limitations is that a triangulation of the entire domain is required to define the test functions. Generating it may be very expensive or even impractical for very complex geometries (like, for example, the ones in [48]) and in moderate- or high-dimensional problems, for which automatic mesh-generation algorithms do not exist. For such domains, it is therefore highly advisable or computationally necessary to use a meshfree method such as the original PINN or the proposed MF-VPINN. Moreover, when dealing with complex geometries for which a mesh can be hardly generated, the refinement of the mesh for adaptive methods can be very difficult. In this paper, we describe an algorithm that solves the problem and provides a reliable solution.

The paper is organized as follows. In Section 2, we introduce the problem we are interested in. In particular, we focus on the problem discretization in Section 2.1 and on the MF-VPINN loss function in Section 2.2. Then, an a posteriori error estimator is presented in Section 2.3 and used in Section 2.4 to iteratively generate the required test functions. Numerical results are presented in Section 3. In Section 3.1, we describe the model implementation and some strategies to improve the model efficiency, in Section 3.2 we compare different approaches to generate the test functions and compare their performance and, in Section 3.3, we analyze the role of the error estimator introduced in Section 2.3. Similar tests are performed on a different problem in Section 3.4 to describe possible extensions on more complex domains. Finally, we conclude the paper in Section 4 and discuss future perspectives and ideas.

2. Problem Formulation

Let us consider the following second-order elliptic problem, defined on a polygonal or polyhedral domain $\Omega \subset \mathbb{R}^n$ with a Lipshitz boundary $\Gamma = \partial\Omega$:

$$\begin{cases} Lu := -\nabla \cdot (\mu \nabla u) + \boldsymbol{\beta} \cdot \nabla u + \sigma u = f & \text{in } \Omega\,, \\ u = g & \text{on } \Gamma\,, \end{cases} \tag{1}$$

where $\mu, \sigma \in \mathrm{L}^\infty(\Omega)$, $\boldsymbol{\beta} \in (\mathrm{W}^{1,\infty}(\Omega))^n$ satisfy $\mu \geq \mu_0$, $\sigma - \frac{1}{2}\nabla \cdot \boldsymbol{\beta} \geq 0$ in Ω for some constant $\mu_0 > 0$, whereas $f \in L^2(\Omega)$ and $g = \bar{u}_{|\Gamma}$ for some $\bar{u} \in \mathrm{H}^1(\Omega)$.

In order to derive the corresponding variational formulation, we define the bilinear form a and the linear form F as

$$a : V \times V \to \mathbb{R}\,, \qquad a(w,v) = \int_\Omega \mu \nabla w \cdot \nabla v + \boldsymbol{\beta} \cdot \nabla w\, v + \sigma w\, v\,, \tag{2}$$

$$F : V \to \mathbb{R}, \qquad F(v) = \int_\Omega f\, v\,; \tag{3}$$

where V is the function space $V = H_0^1(\Omega)$. We denote by $\alpha \geq \mu_0$ the coercivity constant of a and by $\|a\|$ and $\|F\|$ the continuity constants of a and F. Then, the variational formulation of Problem (1) reads as follows: Find $u \in \bar{u} + V$ such that

$$a(u, v) = F(v) \qquad \forall v \in V\,. \tag{4}$$

2.1. Problem Discretization

In order to numerically solve Problem (4), one needs to choose suitable finite-dimensional approximations of the trial space $\bar{u} + V$ and of the test space V. A Galerkin formulation is considered when we consider a finite-dimensional space V_h^{trial} for the trial space $\bar{u} + V_h^{\text{trial}}$ and a finite-dimensional test space V_h^{test}, with $V_h^{\text{trial}} = V_h^{\text{test}}$; whereas a Petrov–Galerkin formulation is considered otherwise. In this work, we consider a Petrov–Galerkin formulation in which the trial space is approximated by a set of functions $V^{\mathcal{NN}}$ of the form $V^{\mathcal{NN}} = \bar{u} + V_h^{\text{trial}}$, with V_h^{trial} represented by a neural network suitably modified to enforce the Dirichlet boundary conditions, and the test space is a space V_h of piecewise linear functions.

The neural network considered in the following is a standard fully connected feed-forward neural network. Given the number L of layers and a set of matrices $A_\ell \in \mathbb{R}^{N_\ell \times N_{\ell-1}}$ and vectors $b_\ell \in \mathbb{R}^{N_\ell}$, $\ell = 1, \ldots, L$ containing the neural network's trainable weights, the function $w : \mathbb{R}^n \to \mathbb{R}$ associated with the considered neural network architecture is:

$$\begin{aligned} &x_0 = \boldsymbol{x}, \\ &x_\ell = \rho(A_\ell x_{\ell-1} + b_\ell), \qquad \ell = 1, \ldots, L-1, \\ &w(\boldsymbol{x}) = A_L x_{L-1} + b_L. \end{aligned} \tag{5}$$

where $\rho : \mathbb{R} \to \mathbb{R}$ is a nonlinear function applied element-wise to the vector $A_\ell x_{\ell-1} + b_\ell$. In this section, we use $\rho(x) = \tanh(x)$; other common choices include, but are not limited to, $\rho(x) = \text{ReLU}(x) = \max\{0, x\}$, $\rho(x) = \text{RePU}(x) = \max\{0, x^p\}$ for $1 < p \in \mathbb{N}$, $\rho(x) = 1/(1 + e^{-x})$ and $\rho(x) = \log(1 + e^x)$. Note that, in order to represent a function $w : \mathbb{R}^n \to \mathbb{R}$, the layer widths N_ℓ of the first and last layers are chosen as $N_0 = n$ and $N_L = 1$. We denote by $W^{\mathcal{NN}}$ the set of functions that can be represented as in (5) for any combination of the neural network weights and by $\boldsymbol{w}^{\mathcal{NN}}$ the vector containing all the trainable weights of the neural network.

The function w defined in (5) is independent of the differential problem that has to be solved and is, in most papers on PINNs or related models, trained to minimize both the residual of the equation and a term penalizing the discrepancy between $w_{|\Gamma}$ and g. Instead, we add a non-trainable layer B to the neural network architecture in order to automatically enforce the required boundary conditions without the need to learn them during the training. As described in [49], the operator B acts on the neural network output as

$$Bw = \phi w + \bar{g}, \tag{6}$$

where $\phi : \Omega \to \mathbb{R}$ is a function vanishing on Γ and strictly positive inside Ω, and $\bar{g} : \Omega \to \mathbb{R}$ is a suitable extension of $g : \Gamma \to \mathbb{R}$. The advantages of such an approach are also described in [50]. Then, the discrete trial space approximating $\bar{u} + V$ can be defined as

$$V^{\mathcal{NN}} = \{v^{\mathcal{NN}} \in \bar{u} + V : v^{\mathcal{NN}} = Bw \text{ for some } w \in W^{\mathcal{NN}}\}\,.$$

On the other hand, the discrete test space V_h is not associated with the neural network and only contains known test functions. In standard VPINNs, one generates a triangulation $\mathcal{T}$ of the domain Ω and then defines V_h as the space of functions that coincide with a polynomial of order $p \in \mathbb{N}$ inside each element of $\mathcal{T}$. Instead, we want to construct a discrete space V_h of functions independent from a global triangulation $\mathcal{T}$. Moreover, since in [47] it has been proven that the VPINN convergence rate with respect to mesh refinement decreases when the order of the test functions is increased, we are interested in a space V_h that only contains piecewise linear functions. For the sake of simplicity, we only consider the case where $n = 2$; the discussion can be directly generalized to the more general case $n \in \mathbb{N}$.

Let $\hat{P} \subset \mathbb{R}^n$ be a reference patch. In the following discussion, $\hat{P}$ can be any arbitrary star-shaped polygon with $N_{\hat{P}}$ vertices and the dimension of its kernel strictly greater than zero. Nevertheless, in the numerical experiments, we only consider the reference patch $\hat{P} = [0,1]^2$ to avoid any unnecessary computational overhead. Let $\mathcal{M} = \{M_i\}_{i=1}^{n_{\text{patches}}}$ be a set of affine mappings such that $M_i : \hat{P} \to P_i \subset \Omega$, where we denote as P_i the patch obtained transforming the reference patch $\hat{P}$ through the map M_i. We assume that $\mathcal{P} = \{P_i\}_{i=1}^{n_{\text{patches}}}$ is a cover of Ω, i.e., $\cup_{i=1}^{n_{\text{patches}}} P_i = \Omega$, and we admit overlapping patches.

Let us consider the triangulation $\hat{\mathcal{T}} = \{\hat{T}_j : 1 \le j \le N_{\hat{P}}\}$ of $\hat{P}$ obtained by connecting each vertex with a single point $\boldsymbol{c}_{\hat{P}}$ in its kernel. It is then possible to define a piecewise linear function $\hat{\varphi}$ vanishing on the border of $\hat{P}$ such that $\hat{\varphi}(\boldsymbol{c}_{\hat{P}}) = 1$ and $\hat{\varphi}_{|\hat{T}_j} \in \mathbb{P}_1(\hat{T}_j)$, for any $j = 1, \dots, N_{\hat{P}}$. Then, we define the discrete test space V_h as $V_h = \text{span}\{\varphi_i : i = 1, \dots, n_{\text{patches}}\}$, where $\varphi_i \in V$ is the piecewise linear function:

$$\varphi_i(\boldsymbol{x}) = \begin{cases} \hat{\varphi}(M_i^{-1}(\boldsymbol{x})), & \boldsymbol{x} \in P_i, \\ 0, & \boldsymbol{x} \notin P_i. \end{cases} \tag{7}$$

We remark that the only required triangulation is $\hat{\mathcal{T}}$, which contains only $N_{\hat{P}}$ triangles (in the numerical tests in this paper, $N_{\hat{P}} = 4$). Instead, there exists no mesh on Ω and the test functions φ_i and their supports P_i are all independent. Therefore, the proposed method is said to be meshfree. A simple example of a set of patches $\mathcal{P}$ with $n_{\text{patches}} = 7$ on the domain $\Omega = [0,1]^2$ is shown in Figure 1. For the sake of simplicity, in this work, we consider a squared reference patch $\hat{P}$ with $\boldsymbol{c}_{\hat{P}}$ coinciding with its center, and we let each mapping M_i represent a combination of scalings and translations.

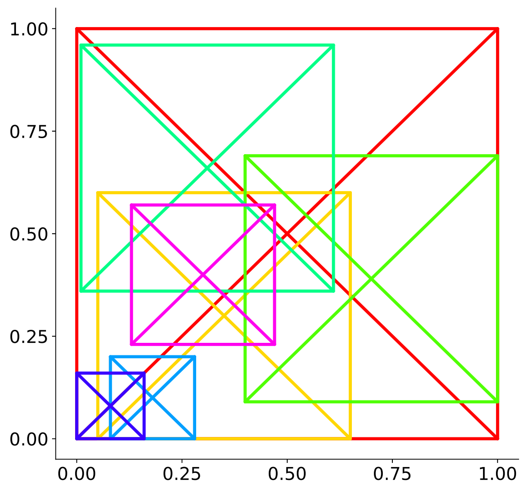


Figure 1. Graphical representation of a set $\{P_i\}_{i=1}^{n_{\text{patches}}}$ obtained from a squared reference patch $\hat{P}$ with $\boldsymbol{c}_{\hat{P}}$ in its center covering the domain $\Omega = (0,1)^2$.

Using the introduced finite-dimensional set of functions $V^{\mathcal{NN}}$ and V_h, it is possible to discretize Problem (4) as follows: Find $u^{\mathcal{NN}} \in V^{\mathcal{NN}}$ such that

$$a(u^{\mathcal{NN}}, v) = F(v) \qquad \forall v \in V_h\,. \tag{8}$$

2.2. Loss Function

In this section, we derive the loss function used to train the neural network. It has to be computable, and its minimizer has to be an approximate solution of Problem (1). We highlight that, when a standard PINN is used, the loss function can be seen as a discrete cost penalizing the residual of (1) directly. In this context, instead, the loss function penalizes the variational residuals of (4) as in standard VPINNs. This is the key difference that differentiates the VPINNs (and its extension proposed in this manuscript) from the other generalizations of the original PINN introduced in Section 1.

Let us consider a quadrature rule of order $q \geq 2$ on each triangle $T_j \in \hat{\mathcal{T}}$, $j = 1, \dots, N_{\hat{P}}$, uniquely identified by a set of nodes and weights $\{(\tilde{\boldsymbol{\xi}}^j_\ell, \tilde{\omega}^j_\ell) : \ell \in I^{T_j}\}$. The nodes and weights of a composite quadrature formula of order q on $\hat{P}$ can be obtained as

$$\{(\hat{\boldsymbol{\xi}}_\ell, \hat{\omega}_\ell) : \ell \in I^{\hat{P}}\} = \bigcup_{j=1}^{N_{\hat{P}}} \{(\tilde{\boldsymbol{\xi}}^j_\ell, \tilde{\omega}^j_\ell) : \ell \in I^{T_j}\}.$$

Then, the corresponding quadrature rule of order q of an arbitrary patch P_i is defined as

$$\left\{(\boldsymbol{\xi}^i_\ell, \omega^i_\ell) : \ell \in I^{\hat{P}} | \boldsymbol{\xi}^i_\ell = M_i(\hat{\boldsymbol{\xi}}_\ell), \omega^i_\ell = \hat{\omega}_\ell \frac{\text{area}(P_i)}{\text{area}(\hat{P})}\right\}. \tag{9}$$

Using the quadrature rule in (9), it is possible to define an approximate restriction on each patch of the forms a and F as follows:

$$a^i_h(w, v) = \sum_{\ell \in I^{\hat{P}}} [\mu \nabla w \cdot \nabla v + \boldsymbol{\beta} \cdot \nabla w\, v + \sigma w v](\boldsymbol{\xi}^i_\ell)\, \omega^i_\ell \approx a_{P_i}(w, v)\,, \tag{10}$$

$$F^i_h(v) = \sum_{\ell \in I^{\hat{P}}} [f v](\boldsymbol{\xi}^i_\ell)\, \omega^i_\ell \approx F_{P_i}(v)\,, \tag{11}$$

where $a_{P_i}(w, v)$ and $F_{P_i}(v)$ are defined as in (2) and (3) but restricting the supports of the integrals to P_i. We remark that, since it is not possible to compute integrals involving a neural network exactly, we can only use the forms a^i_h and F^i_h in the loss function. Exploiting the linearity of $a(w, v)$ and $F(v)$ with respect to v to consider only the basis $\{\varphi_i\}_{i=1}^{n_\text{patches}}$ of V_h as set of test functions, we approximate Problem (8) as follows: Find $u^{\mathcal{NN}} \in V^{\mathcal{NN}}$ such that

$$a^i_h(u^{\mathcal{NN}}, \varphi_i) = F^i_h(\varphi_i) \qquad \forall i = 1, \dots, n_\text{patches}\,. \tag{12}$$

Then, in order to cast Problem (12) into an optimization problem, we define the residuals

$$r_{h,i}(w) = F^i_h(\varphi_i) - a^i_h(w, \varphi_i)\,, \qquad i = 1, \dots, n_\text{patches} \tag{13}$$

and the loss function

$$R^2_h(w; \mathcal{P}) = \frac{1}{n_\text{patches}} \sum_{i=1}^{n_\text{patches}} \gamma_i r^2_{h,i}(w)\,, \tag{14}$$

where γ_i are suitable positive scaling coefficients. In this work, we use $\gamma_i = \text{area}(P_i)^{-1}$ to give the same importance to each patch. Note that this is equivalent to normalizing the quadrature rules involved in (10) and (11); this way, each residual $r_{h,i}$ can be regarded as a linear combination of the MF-VPINN value and derivatives independent of the size of the support of the patch P_i. We also highlight that the loss function depends on the choice of $\mathcal{M}$ since all the used test functions are generated starting from the corresponding mappings $M_i \in \mathcal{M}$. We are now interested in a practical procedure to obtain a set $\widetilde{\mathcal{P}}$ such that the approximate solution computed minimizing $R^2_h(\cdot; \widetilde{\mathcal{P}})$ is as accurate as possible with $\widetilde{\mathcal{P}}$ being as small as possible.

2.3. The a Posteriori Error Estimator

The goal of this section is to derive an error estimator associated with an arbitrary patch P_i, with $i \in \{1, \dots, n_{\text{patches}}\}$. To do so, we rely on the a posteriori error estimator proposed in [46]. It has been proven to be efficient and reliable; therefore, such an estimator allows us to know where the error is larger without knowing the exact solution of the PDE. Let us consider the patch P_i, formed by the triangles $T_{i,1}, \dots, T_{i,N_{\hat{p}}}$ and a triangulation $\mathcal{T}_i$ of Ω such that $T_{i,j} \in \mathcal{T}_i$, for every $j = 1, \dots, N_{\hat{p}}$. We remark that the triangulation $\mathcal{T}_i$ does not have to be explicitly generated; it is only used to properly define all the quantities introduced in [46] required to derive the proposed error estimator.

Let $V_h^i = \text{span}\{\psi_j^i : 1 \le j \le \dim(V_h^i)\}$ be the space of piecewise linear functions defined on $\mathcal{T}_i$. Where $\{\psi_j^i : 1 \le j \le \dim(V_h^i)\}$ is a Lagrange basis of V_h^i. It is then possible to define two constants c_h^i and C_h^i, with $0 < c_h^i < C_h^i$, such that

$$c_h^i |v|_{1,\Omega} \le \|\boldsymbol{v}\|_2 \le C_h^i |v|_{1,\Omega} \qquad \forall v \in V_h^i\,, \tag{15}$$

where $v = \sum_{j=1}^{\dim(V_h^i)} v_j \psi_j^i$ is an arbitrary element of V_h^i associated with the expansion coefficients $\boldsymbol{v} = \left\{v_1, \dots, v_{\dim(V_h^i)}\right\}$ and $\|\boldsymbol{v}\|_2 = \left(\sum_{j=1}^{\dim(V_h^i)} v_i^2\right)^{1/2}$.

Then, given an integer $k \ge 0$, for any element $E \in \mathcal{T}_i$, we define the projection operator $\Pi_{E,k} : L^2(E) \to \mathbb{P}_k(E)$ such that

$$\int_E \Pi_{E,k}\phi = \int_E \phi \qquad \forall \phi \in L^2(E)\,. \tag{16}$$

We also denote by $\{(\boldsymbol{\xi}_\ell^E, \omega_\ell^E) : \ell \in I^E\}$ a quadrature formula of order q on E and define the quadrature-based discrete seminorm:

$$\|v\|_{0,E,\omega} = \left(\sum_{\ell \in I^E} v^2(\boldsymbol{\xi}_\ell^E)\,\omega_\ell^E\right)^{1/2}. \tag{17}$$

We require the weights and nodes of this quadrature rule to coincide with the ones introduced in (9) when E is a triangle included in P_i (i.e., when $E \in \{T_{i,1}, \dots, T_{i,N_{\hat{p}}}\}$). We can now introduce all the terms involved in the a posteriori error estimator.

Let $\eta_{\text{rhs},1}(E)$ and $\eta_{\text{rhs},2}(E)$ be the quantities:

$$\begin{aligned} \eta_{\text{rhs},1}(E) &= h_E \|f - \Pi_{E,q-1} f\|_{0,E}\,, \\ \eta_{\text{rhs},2}(E) &= h_E \|f - \Pi_{E,q-1} f\|_{0,E,\omega} + \|f - \Pi_{E,q} f\|_{0,E,\omega}\,. \end{aligned} \tag{18}$$

They measure the oscillations of the forcing term with respect to its polynomial projections in various norms. Similar oscillations are also measured for the diffusion, convection and reaction terms by the terms $\eta_{\text{coef},i}(E)$ for $i = 1, \dots, 6$:

$$\begin{aligned} \eta_{\text{coef},1}(E) &= \|\mu \nabla u^{\mathcal{NN}} - \Pi_{E,q}(\mu \nabla u^{\mathcal{NN}})\|_{0,E}\,, \\ \eta_{\text{coef},2}(E) &= h_E \|\boldsymbol{\beta} \cdot \nabla u^{\mathcal{NN}} - \Pi_{E,q-1}(\boldsymbol{\beta} \cdot \nabla u^{\mathcal{NN}})\|_{0,E}\,, \\ \eta_{\text{coef},3}(E) &= h_E \|\sigma u^{\mathcal{NN}} - \Pi_{E,q-1}(\sigma u^{\mathcal{NN}})\|_{0,E}\,, \\ \eta_{\text{coef},4}(E) &= \|\mu \nabla u^{\mathcal{NN}} - \Pi_{E,q}(\mu \nabla u^{\mathcal{NN}})\|_{0,E,\omega}\,, \\ \eta_{\text{coef},5}(E) &= h_E \|\boldsymbol{\beta} \cdot \nabla u^{\mathcal{NN}} - \Pi_{E,q-1}(\boldsymbol{\beta} \cdot \nabla u^{\mathcal{NN}})\|_{0,E,\omega}\,, \\ &\qquad + \|\boldsymbol{\beta} \cdot \nabla u^{\mathcal{NN}} - \Pi_{E,q}(\boldsymbol{\beta} \cdot \nabla u^{\mathcal{NN}})\|_{0,E,\omega} \\ \eta_{\text{coef},6}(E) &= h_E \|\sigma u^{\mathcal{NN}} - \Pi_{E,q-1}(\sigma u^{\mathcal{NN}})\|_{0,E,\omega} \\ &\qquad + \|\sigma u^{\mathcal{NN}} - \Pi_{E,q}(\sigma u^{\mathcal{NN}})\|_{0,E,\omega}\,, \end{aligned} \tag{19}$$

where $u^{\mathcal{NN}}$ is the output of the neural network after the enforcement of the Dirichlet boundary conditions through the operator B and h_E is the diameter of E. Then, let us define the term $\eta_{res}(E)$, which measures how well the equation is satisfied, as

$$\eta_{\text{res}}(E) = h_E \| \operatorname{bulk}_E(u^{\mathcal{NN}}) \|_{0,E} + h_E^{1/2} \sum_{e \subset \partial E} \| \operatorname{jump}_e(u^{\mathcal{NN}}) \|_{0,e}\,, \tag{20}$$

where

$$\operatorname{bulk}_E(u^{\mathcal{NN}}) = \Pi_{E,q-1} f + \nabla \cdot \Pi_{E,q}(\mu \nabla u^{\mathcal{NN}}) - \Pi_{E,q-1}(\boldsymbol{\beta} \cdot \nabla u^{\mathcal{NN}} + \sigma u^{\mathcal{NN}})$$

$$\operatorname{jump}_e(u^{\mathcal{NN}}) = \Pi_{E_1,q}(\mu \nabla u^{\mathcal{NN}}) \cdot \boldsymbol{n} - \Pi_{E_2,q}(\mu \nabla u^{\mathcal{NN}}) \cdot \boldsymbol{n}\,.$$

Note that $\operatorname{jump}_e(u^{\mathcal{NN}})$ measures the interelemental jumps of $\Pi_{E,q}(\mu \nabla u^{\mathcal{NN}})$ across the edge e with normal unit vector $\boldsymbol{n}$ shared by the elements E_1 and E_2.

Finally, we introduce the approximate elemental forms:

$$a_h^{i,E}(w,v) = \sum_{\ell \in I^E} [\mu \nabla w \cdot \nabla v + \boldsymbol{\beta} \cdot \nabla w\, v + \sigma w v](\boldsymbol{\xi}_\ell^E)\, \omega_\ell^E\,, \tag{21}$$

$$F_h^{i,E}(v) = \sum_{\ell \in I^E} [f v](\boldsymbol{\xi}_\ell^E)\, \omega_\ell^E\,, \tag{22}$$

where $\boldsymbol{\xi}_\ell^E$ and ω_ℓ^E, $\ell \in I^E$, are the nodes and weights used in Equation (17). With such forms, it is possible to define the residuals

$$r_{h,i,j}(w) = \sum_{E \in \mathcal{T}_i} F_h^{i,E}(\psi_j^i) - a_h^{i,E}(w, \psi_j^i)\,, \qquad j = 1, \dots, \dim\left(V_h^i\right)$$

and the quantity $\eta_{\text{loss}}(E)$ as

$$\eta_{\text{loss}}(E) = C_h \sqrt{\sum_{j \in I_h^E} r_{h,i,j}^2(u^{\mathcal{NN}})}\,. \tag{23}$$

Here, denoting the support of the function $\psi_j^i \in V_h^i$ by $\operatorname{supp} \psi_j^i$, the elemental index set

$$I_h^E = \{ j \in I_h : E \subset \operatorname{supp} \psi_j^i \}$$

is the set containing the indices of the functions whose support contains E. It is then possible to estimate the error between the unknown exact solution u and its MF-VPINN approximation $u^{\mathcal{NN}}$ by means of the computable quantities in Equations (18)–(20) and (23) as

$$|u - u^{\mathcal{NN}}|_{1,E} \lesssim \left(\eta_{\text{res}}^2(E) + \eta_{\text{loss}}^2(E) + \sum_{i=1}^{6} \eta_{\text{coef},i}^2(E) + \sum_{i=1}^{2} \eta_{\text{rhs},i}^2(E) \right)^{1/2}. \tag{24}$$

Once more, we refer to [46] for the proof of such a statement.

We recall that our goal is to obtain a computable error estimator associated with a single patch P_i. When evaluated on an element $E \in P_i$, the quantity on the right-hand side of Equation (24) implicitly depends on several elements in V_h^i that do not belong to P_i because of the presence of $\eta_{\text{res}}^2(E)$ and $\eta_{\text{loss}}^2(E)$. Therefore, such an estimator is not computable without generating the triangulation $\mathcal{T}_i$ and the corresponding space V_h^i. Instead, we look for an error estimator that does not control the error on the entire patch but only in a neighborhood $\mathcal{N}_i$ of its center $\boldsymbol{c}_{P_i} = M_i\left(\boldsymbol{c}_{\hat{P}}\right)$. This can be carried out by considering only the terms whose computation involves geometric elements containing $\boldsymbol{c}_{P_i}$ and the only function ψ_j^i that does not vanish on $\boldsymbol{c}_{P_i}$. Note that such a function is the function φ_i defined in (7). Therefore, the error estimator η_i that controls the error in $\mathcal{N}_i$ can be computed as

$$\eta_i = \left[\eta_{\text{res},i}^2 + C_h^2 r_{h,i}^2(u^{\mathcal{NN}}) + \sum_{j=1}^{N_{\hat{P}}} \left(\sum_{k=1}^{6} \eta_{\text{coef},k}^2(T_{i,j}) + \sum_{k=1}^{2} \eta_{\text{rhs},k}^2(T_{i,j}) \right) \right]^{1/2}, \tag{25}$$

where $\eta_{\text{res},i}$ is defined as

$$\eta_{\text{res},i} = \sum_{j=1}^{N_{\hat{P}}} \left(h_{T_{i,j}} \| \operatorname{bulk}_{T_{i,j}}(u^{\mathcal{NN}}) \|_{0,T_{i,j}} + h_{P_i}^{1/2} \| \operatorname{jump}_{e_{i,j}}(u^{\mathcal{NN}}) \|_{0,e_{i,j}} \right). \tag{26}$$

In (26), we denote by h_{P_i} the diameter of the patch P_i and by $e_{i,j}, j = 1, \dots, N_{\hat{P}}$ the edges connecting its vertices with $\boldsymbol{c}_{\boldsymbol{P}_i}$.

Since η_i can be seen as an approximation of the right-hand side of (24), we use it as an indicator of the error $|u - u^{\mathcal{NN}}|_{1,\mathcal{N}_i}$. It is important to remark that η_i can be computed without generating $\mathcal{T}_i$ and V_h^i. In fact, its computation involves only the function φ_i, the triangles partitioning P_i and the edges connecting its vertices with its center.

2.4. The Choice of $\mathcal{M}$ and $\mathcal{P}$

In this section, the procedure adopted to generate the set of test functions used to train the MF-VPINN is described. We propose an iterative approach, in which the MF-VPINN is initially trained with very few test functions, and then other test functions are added in the regions of the domain in which the H_1 norm of the error is larger. We anticipate that, as shown in Section 3.3, generating test functions in regions where $r_{h,i}^2$ is large may not lead to accurate solutions because $r_{h,i}^2$ is not proportional to the H_1 error. Therefore, such a choice may increase the density of test functions where they are not required while maintaining only a few test functions in regions in which the error is large. Instead, we use the error indicator η_i defined in (25).

Let us initially consider a cover $\mathcal{P}_0 = \{P_i\}_{i=1}^{n_{\text{patches}}}$ of Ω comprising a few patches (i.e., n_{patches} is a small integer) and the corresponding set of mappings $\mathcal{M}_0 = \{M_i\}_{i=1}^{n_{\text{patches}}}$ and test functions $\{\varphi_i\}_{i=1}^{n_{\text{patches}}}$. These sets induce a loss function $R_h^2(w; \mathcal{P}_0)$ as defined in (14), which is used to train an MF-VPINN. After this initial training, one computes $\eta_i^\gamma = \gamma_i \eta_i$ for each patch $P_i \in \mathcal{P}_0$ and stores the result in the array $\boldsymbol{\eta} = \left[\eta_1^\gamma, \dots, \eta_{n_{\text{patches}}}^\gamma\right]$. Note that η_i^γ is a suitable rescaling of η_i to get rid of dependence from the size of P_i. Let us choose a threshold $1 \le \tau_0 \le n_{\text{patches}}$, sort $\boldsymbol{\eta}$ in descending order obtaining $\boldsymbol{\eta}_{\text{sort}} = \left[\eta_{s_1}^\gamma, \dots, \eta_{s_{n_{\text{patches}}}}^\gamma\right]$ (where we denote by $[s_1, \dots, s_{n_{\text{patches}}}]$ the index set corresponding to a suitable permutation of $[1, \dots, n_{\text{patches}}]$) and consider the vector $\overline{\boldsymbol{\eta}}_0 = \left[\eta_{s_1}^\gamma, \dots, \eta_{s_{\tau_0}}^\gamma\right]$. It is possible to note that $\overline{\boldsymbol{\eta}}_0$ contains only the τ_0 worst values of the indicator; it thus allows us to understand where the error is higher and where additional test functions are required to increase the model accuracy.

It is then possible to move forward with the second iteration of the iterative training. For each patch P_i such that $\eta_i^\gamma \in \overline{\boldsymbol{\eta}}_0$, we generate k_{new} new patches $P_i^k, k = 1, \dots, k_{\text{new}}$ with centers inside P_i and areas such that $\text{area}(P_i) < \cup_{k=1}^{k_{\text{new}}} \text{area}(P_i^k) < c \cdot \text{area}(P_i)$, where $c > 1$ is a tunable parameter. In the numerical experiments, we use $c = 1.25$. There exist different strategies to choose the number, the dimension, and the position of the centers of the new patches. Such strategies are described in Section 3 with particular attention to the effects of these choices on the MF-VPINN accuracy.

Let us denote by $\mathcal{P}_1$ the set $\mathcal{P}_1 = \mathcal{P}_0 \cup \{P_{s_1}^k\}_{k=1}^{k_{\text{new}}} \cup \dots \cup \{P_{s_{\tau_0}}^k\}_{k=1}^{k_{\text{new}}}$ and by $\mathcal{M}_1$ the corresponding set of mappings. Then, it is possible to define the loss function $R_h^2(w; \mathcal{P}_1)$, continue the training of the previously trained MF-VPINN, compute the error indicator η_i^γ for each patch $P_i \in \mathcal{P}_1$, and obtain the vector $\overline{\boldsymbol{\eta}}_1$ used to decide where to insert the new patches to generate $\mathcal{P}_2$. In general, iterating this procedure, it is possible to compute a set of patches $\mathcal{P}_m$ and of mappings $\mathcal{M}_m$ from the previously obtained sets $\mathcal{P}_{m-1}$ and $\mathcal{M}_{m-1}$. Technical optimization details are discussed in Section 3.1.

3. Numerical Results

In this section, we provide several numerical results to show the performance of the training strategy described in Section 2.4. In Section 3.1, we describe the structure of the MF-VPINN implementation and highlight some details that have to be taken into account in order to increase the efficiency of the training phase. Different strategies to choose the position of the new patches are discussed in Section 3.2. The importance of the use of the error indicator is remarked in Section 3.3 with additional numerical examples. An example on a more complex domain is shown in Section 3.4 to discuss some ideas to adapt the proposed strategies in more complex domains.

3.1. Implementation Details

The computer code used to perform the experiment is implemented in Python using the Python package Tensorflow [4] to generate the neural network architecture and train the MF-VPINN. Using the notation introduced in Section 2.1, the used neural network consists of $L = 5$ layers with $N_\ell = 50$ neurons in each hidden layer (i.e., for $\ell = 1, \ldots, L-1$); the activation function is the hyperbolic tangent in each hidden layer. For the first iteration of the iterative training, the neural network weights in the ℓ-th layer are initialized with a glorot normal distribution, i.e., a truncated normal distribution with mean 0 and standard deviation equal to $\sqrt{2/(N_{\ell-1} + N_\ell)}$. Then, for the subsequent iterations, their are initialized with the weights obtained at the end of the previous one.

During the first iteration of the training (during the minimization of $R_h^2(\cdot;\mathcal{P}_0)$), the optimization is carried out by exploiting the ADAM optimizer [51] with an exponentially decaying learning rate from 10^{-2} to 10^{-4} and with the second-order L-BFGS optimizer [52]. Then, from the second training iteration, we only use the L-BFGS optimizer. We remark that L-BFGS allows a very fast convergence but only if the initial starting point is close enough to the problem's solution. Therefore, in the first training iteration, we use ADAM to obtain a first approximation of the solution that is then improved via L-BFGS. Then, since the m-th training iteration starts from the solution computed during the $(m-1)$-th one, we assume that the starting point is close enough to the solution of the new optimization problem (associated with a difference loss function with more patches) and we only use L-BFGS to increase the training efficiency.

During the m-th iteration of the training, the training set consists of all the quadrature nodes $\boldsymbol{\xi}_\ell^i$, for any $\ell \in I^{\hat{P}}$ and for any patch $P_i \in \mathcal{P}_m$ as defined in (9). The order of the chosen quadrature rule is $q = 3$ inside each triangle. The Dirichlet boundary conditions are imposed by means of the operator B defined in (6). In this operator, for our first numerical test, the function ϕ is a polynomial bubble vanishing on Γ and $\overline{g}$ is the output of a neural network trained to interpolate the boundary data. For the numerical test in Section 3.4, instead, ϕ is computed as in [50] and $g = 0$. To decrease the training time, the functions ϕ, $\nabla\phi$, $\overline{g}$ and $\nabla\overline{g}$ are evaluated only once at the beginning of the m-th training iteration and they are then combined to evaluate $Bu^{\mathcal{NN}}$ and its gradient (where $u^{\mathcal{NN}}$ is the output of the last layer of the neural network). The derivatives of $u^{\mathcal{NN}}$ and $\overline{g}$ are computed via automatic differentiation [53] due to the complexity of their analytical expressions.

The output of the model is the value of the function $Bu^{\mathcal{NN}}$ and its gradient evaluated at the input points. Such values are then suitably combined using sparse and dense tensors to compute the quantity $R_h^2(Bu^{\mathcal{NN}};\mathcal{P}_m)$. The sparse tensors contain the evaluation of φ_i and $\nabla\varphi_i$ at each input point, whereas the dense ones store the quadrature weights, the vector $\boldsymbol{\gamma} = \{\gamma_i\}_{i=1}^{n_{\text{patches}}}$ and the evaluation of μ, $\boldsymbol{\beta}$, σ and f at the input points. We highlight that all these tensors have to be computed once at the beginning of the m-th training iteration (updating the ones of the $(m-1)$-th iteration) to significantly decrease the training computational cost.

As discussed in Section 2.1, we assume that all the patches and test functions can be generated from a reference patch $\hat{P}$. For each patch $P_i \in \mathcal{P}_m$, one has to generate all the data structures required to assemble the loss function and the error indicator η_i. To do so, it is possible to explicitly construct all the tensors required to assemble the term $\hat{a}_h(w, \hat{\varphi})$ and all the terms involved in the computation of the reference error indicator $\hat{\eta}$ only once, at the beginning of the first iteration of the training. Then, all these tensors can be suitably rescaled to obtain the ones corresponding to the patches and test functions involved in the loss function and error indicators computations.

To stabilize the MF-VPINN, we introduce the L^2 regularization term

$$\mathcal{L}_{\text{reg}}(u^{\mathcal{NN}}) = \lambda_{\text{reg}} \|\mathbf{u}^{\mathcal{NN}}\|_2^2,$$

where $\mathbf{u}^{\mathcal{NN}}$ is the set of weights of the neural network introduced in Section 2.1. In our numerical experiments, we use $\lambda_{\text{reg}} = 10^{-5}$. During the m-th iteration of the training, such a quantity is added to $R_h^2(Bu^{\mathcal{NN}}; \mathcal{P}_m)$ to obtain the training loss function

$$\mathcal{L}_m(u^{\mathcal{NN}}) = R_h^2(Bu^{\mathcal{NN}}; \mathcal{P}_m) + \mathcal{L}_{\text{reg}}(\mathbf{w}^{\mathcal{NN}}), \tag{27}$$

which has to be minimized accurately enough. Indeed, if $\mathcal{L}_m$ is minimized poorly, the new patches $\mathcal{P}_{m+1} \backslash \mathcal{P}_m$ may be added in regions where they are not necessary because the accuracy of $Bu^{\mathcal{NN}}$ may still improve during the training and may not be inserted in areas where they are required. Note that, in order to compute the numerical solution, the MF-VPINN has to be trained multiple times with a different set of patches $\mathcal{P}_m$ to minimize the losses $\{\mathcal{L}_m\}$. Since such an iterative training may be expensive, we propose an early stopping strategy [54] based on the discussed error indicator to reduce its computational cost. In its basic version, early stopping consists of evaluating a chosen metric on a validation set in order to know when the neural network accuracy on data that are not present in the training set start worsening. Interrupting the training there prevents overfitting and improves generalization. In our context, instead, we can directly track the behavior of the MF-VPINN H^1 error on each patch through the corresponding error indicator to understand when it stops decreasing. Therefore, given the set of patches $\mathcal{P}_m$, the chosen metric is the linear combination $ES_m = \sum_{i=1}^{\dim(\mathcal{P}_m)} \eta_i^{\gamma}$. Numerical results showing the performance of this strategy are presented in Sections 3.2 and 3.4.

3.2. Adaptive Training Strategies

Let us consider the Poisson problem:

$$\begin{cases} -\Delta u = f & \text{in } \Omega\,, \\ \ \ u = g & \text{on } \Gamma\,, \end{cases} \tag{28}$$

defined on the unit square $\Omega = (0,1)^2$. The forcing term f and the boundary condition g are chosen such that the exact solution is, in polar coordinates,

$$u(r,\theta) = r^{\frac{2}{3}} \sin\left(\frac{2}{3}\left(\theta + \frac{\pi}{2}\right)\right). \tag{29}$$

We use this function, represented in Figure 2, because the solution u is such that $u \in H^{5/3-\varepsilon}(\Omega)$ but $u \in C^\infty(\Omega \backslash \mathcal{N}_0)$, where we denote by $\mathcal{N}_0$ a neighborhood of the origin. Therefore, we know that an efficient distribution of patches has to be characterized by a high density only near the origin.

Below, we propose, in order of complexity, three alternatives to construct the new patches after having marked the ones with the higher error indicator. The first strategy is the most simple and intuitive, and the new patches are randomly generated with centers inside the marked patches, whereas the second strategy and third one place the new centers on a small local cartesian grid to ensure a more regular distribution. The difference between the second and the third strategies is that the marked patches are removed to increase the efficiency and we add a constraint to the marking procedure to ensure more regular distributions of the new patches.

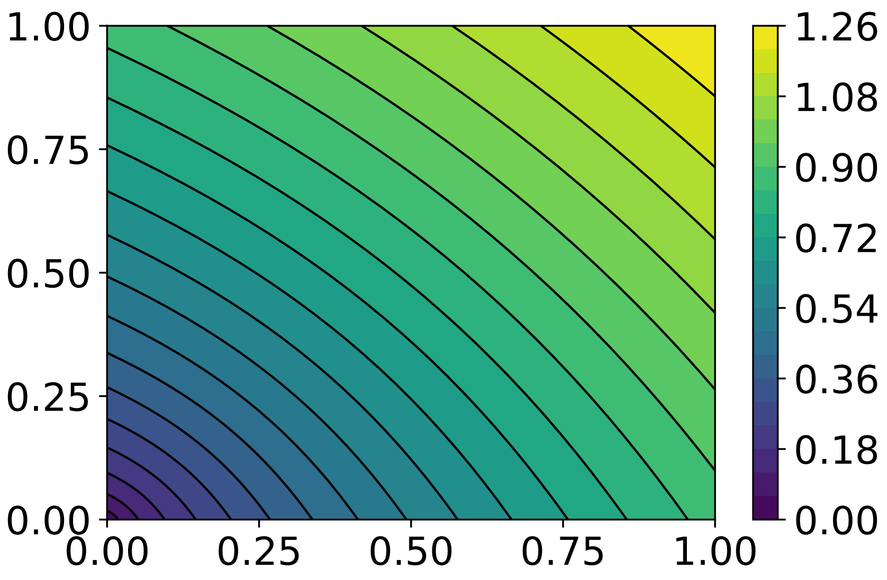


Figure 2. Graphical representation of the solution u in (29).

Strategy #1: Random patch centers with uniform distribution

To solve Problem (28), as a first strategy, we consider the reference patch $\hat{P} = (0,1)^2$ and generate a sequence of sets of patches. During the first training iteration, we use $\mathcal{P}_0 = \{\hat{P}\}$ since this is already a cover of Ω. During the second iteration, we enrich the set of patches as $\mathcal{P}_1 = \mathcal{P}_0 \cup \{P_1, P_2, P_3, P_4\}$ where P_1, P_2, P_3 and P_4 are squared patches with edge $h_i = 0.6$, $i = 1, \dots, 4$ and centers

$$\boldsymbol{c}_{\boldsymbol{P}_1} = (0.3, 0.3), \qquad \boldsymbol{c}_{\boldsymbol{P}_2} = (0.7, 0.3),$$

$$\boldsymbol{c}_{\boldsymbol{P}_3} = (0.3, 0.7), \qquad \boldsymbol{c}_{\boldsymbol{P}_4} = (0.7, 0.7).$$

This allows us to start from a homogeneous distribution of patches before utilizing the error indicator to choose the location of the new patches. Then, to decide how many patches have to be added to $\mathcal{P}_{m-1}$ to generate $\mathcal{P}_m$, we choose $\widetilde{\tau}_m$ such that

$$\widetilde{\tau}_m = \dim\left(\left\{\widetilde{\tau} \in \{1, \dots, \dim(\mathcal{P}_{m-1}\} : \frac{\sum_{i=1}^{\widetilde{\tau}} \eta_{s_i}^{\gamma}}{\sum_{i=1}^{\dim(\mathcal{P}_{m-1})} \eta_i^{\gamma}} < 0.75\right\}\right) + 1 \tag{30}$$

and fix

$$\tau_m = \min(\lceil 0.3 \cdot \dim(\mathcal{P}_{m-1}) \rceil, \widetilde{\tau}_m). \tag{31}$$

Note that (30) allows us to consider the smallest set of patches such that the corresponding error indicators contribute at least 75% of the global error indicator ES_{m-1}, whereas (31) is considered to limit the maximum number of patches that can be added for efficiency reasons.

Then, to generate the generic set of patches $\mathcal{P}_m$, we fix a multiplication factor C_M to decide how many new patches have to be inserted inside each patch P_i such that $\eta_i^{\gamma} \in \overline{\boldsymbol{\eta}}_{m-1}$. Inside each chosen patch P_i, C_M centers $\widetilde{\boldsymbol{c}}_{\boldsymbol{P}_i^k} = (\widetilde{x}_i^k, \widetilde{y}_i^k)$, $k = 1, \dots, C_M$, are randomly generated with a uniform distribution and the new patches' edges' lengths are chosen as $h_i^k = \lambda \frac{A_{\text{ratio}}}{\sqrt{C_M}} h_i$. Here, λ is a random real value from the uniform distribution $U\left(\left[\frac{9}{10}, \frac{10}{9}\right]\right)$, and the scaling coefficient $\frac{A_{\text{ratio}}}{\sqrt{C_M}}$ is chosen such that the sum of the areas of the new patches is A_{ratio} times the area of the original patch P_i. In the numerical experiments, we use $A_{\text{ratio}} = 1.25$. This way, it is possible to allow the new patches to overlap and keep the area of the region $P_i \backslash \left(\cup_{k=1}^{C_M} P_i^k\right)$ reasonably small.

We remark that, with this strategy, it may happen that some patches are outside Ω. In order to avoid this risk, we move the centers $\widetilde{\boldsymbol{c}}_{P_i^k}$ to obtain the actual patches centers $\boldsymbol{c}_{P_i^k}$ as follows:

$$\boldsymbol{c}_{P_i^k} = (x_i^k, y_i^k) \leftarrow \left(\max\left\{\min\left\{\widetilde{x}_i^k, 1-\frac{h_i^k}{2}\right\}, \frac{h_i^k}{2}\right\}, \max\left\{\min\left\{\widetilde{y}_i^k, 1-\frac{h_i^k}{2}\right\}, \frac{h_i^k}{2}\right\}\right). \quad (32)$$

We remark that, when the patch P_i is very close to a vertex of the domain, it is possible that multiple original centers $\widetilde{\boldsymbol{c}}_{P_i^k}$ are such that the distance of both $\widetilde{x}_i^k$ and $\widetilde{y}_i^k$ from the x and y coordinates of the domain vertex is smaller than $h_i^k/2$. In this case, it is important to consider the random coefficient λ in the definition of h_i^k to avoid updating all these centers with the same point; otherwise, multiple new patches would coincide (because they would share the same center and size).

For the numerical test, we consider $C_M = 4$ and $C_M = 9$. Using significantly more accurate quadrature rules, we compare the approximate solution with the exact one defined in (29) and compute the relative H^1 error $\|u - u^{\mathcal{NN}}\|_1 / \|u\|_1$ at the end of each training iteration. The obtained errors are shown as blue circles ($C_M = 4$) and red triangles ($C_M = 9$) in Figure 3. It can be noted that, with both values of C_M, when more patches are used, the error is smaller, even though the convergence rate is limited by the low regularity of the solution. It is also interesting to observe the positions and sizes of the used patches; such information is summarized in Figures 4 and 5. In such figures, each dot is in the center of a patch P_i, and its size and color represent the size h_i^2 and the scaled indicator η_i^γ associated with P_i. It can be noted that, even if the new centers are chosen randomly in the few selected patches, the final distribution is the expected one. In fact, most of the patches cluster around the origin, whereas the rest of the domain is covered by fewer patches. Nevertheless, we highlight that, when $C_M = 9$, there are more small and medium patches far from the origin, yielding a more uniform covering of the areas far from the singular point and a slightly better accuracy.

Strategy #2: Fixed patch centers

From the results discussed in *Strategy #1*, it can be observed that choosing the position of the new centers randomly may lead to non-uniform patch distribution in regions far from the singular point. In order to obtain better distributions, let us fix a priori the position of the new centers. Let us consider the reference patch $\hat{P} = (0,1)^2$ and the points

$$\begin{aligned} \hat{\boldsymbol{c}}_1 &= (0.25, 0.25), & \hat{\boldsymbol{c}}_2 &= (0.75, 0.25), \\ \hat{\boldsymbol{c}}_3 &= (0.25, 0.75), & \hat{\boldsymbol{c}}_4 &= (0.75, 0.75), \end{aligned} \quad (33)$$

when $C_M = 4$ and

$$\begin{aligned} \hat{\boldsymbol{c}}_1 &= (0.2, 0.2), & \hat{\boldsymbol{c}}_2 &= (0.2, 0.5), & \hat{\boldsymbol{c}}_3 &= (0.2, 0.8), \\ \hat{\boldsymbol{c}}_4 &= (0.5, 0.2), & \hat{\boldsymbol{c}}_5 &= (0.5, 0.5), & \hat{\boldsymbol{c}}_6 &= (0.5, 0.8), \\ \hat{\boldsymbol{c}}_7 &= (0.8, 0.2), & \hat{\boldsymbol{c}}_8 &= (0.8, 0.5), & \hat{\boldsymbol{c}}_9 &= (0.8, 0.8), \end{aligned} \quad (34)$$

when $C_M = 9$. At the end of the $(m-1)$-th training iteration, if $\eta_i^\gamma \in \overline{\boldsymbol{\eta}}_{m-1}$, the C_M centers inside P_i are chosen as $\boldsymbol{c}_{P_i^k} = M_i(\hat{\boldsymbol{c}}_k)$, $k = 1, \ldots, C_M$. Once more, to avoid patches partially outside Ω, we update such centers as in (32). We highlight that defining the new centers as in (33) and in (34) and the length h_i^k of the edges of the new patches as in *Strategy #1*, then the new patches with centers inside P_i form a cover of P_i, i.e., $P_i \subsetneq \cup_{k=1}^{C_M} P_i^k$. Such a property does not hold if the new centers are randomly chosen.

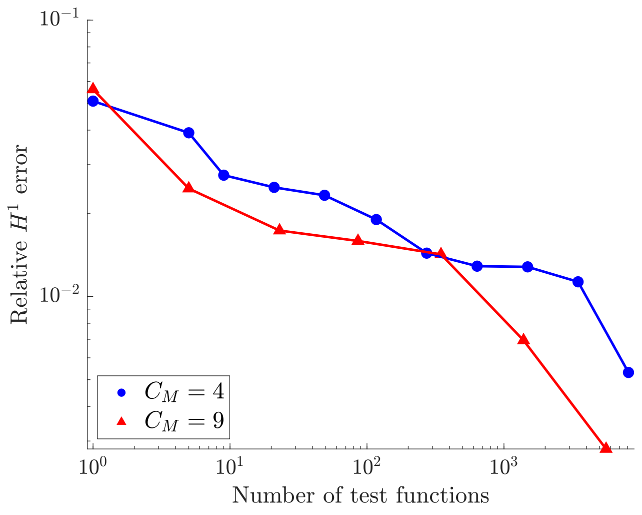


Figure 3. *Strategy #1*: Relative H^1 errors obtained at the end of each training iteration for $C_M = 4$ (blue circles) and $C_M = 9$ (red triangles).

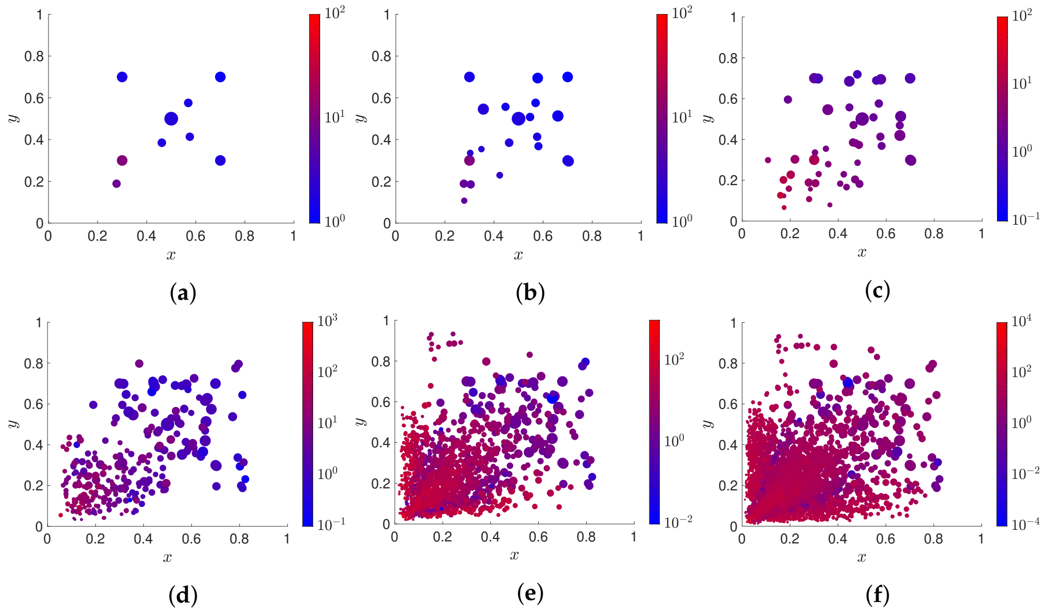


Figure 4. *Strategy #1*: Patches used to train the MF-VPINN with $C_M = 4$. Each dot represents a patch P_i, its position is the center $\mathbf{c}_{\mathbf{P}_i}$ of the patch, its size is proportional to the patch size h_i^2, and its color is associated with the quantity η_i^γ. (**a**) Representation of $\mathcal{P}_2$; (**b**) Representation of $\mathcal{P}_3$; (**c**) Representation of $\mathcal{P}_4$; (**d**) Representation of $\mathcal{P}_6$; (**e**) Representation of $\mathcal{P}_8$; (**f**) Representation of $\mathcal{P}_9$.

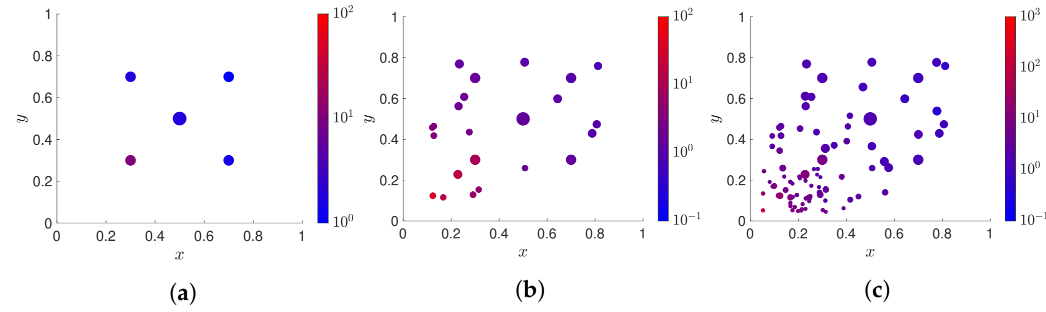


Figure 5. *Cont*.

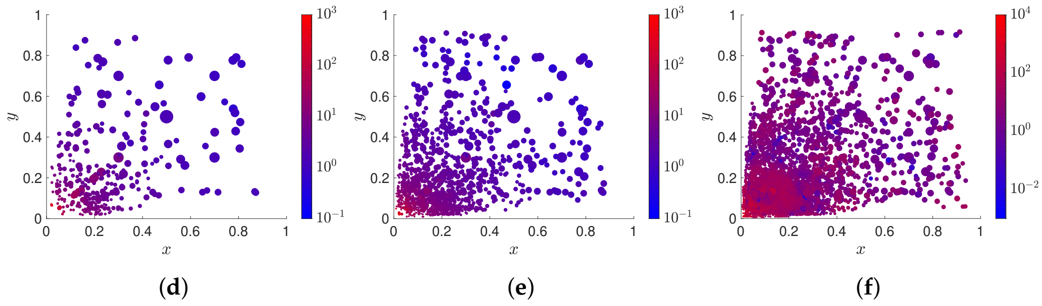


Figure 5. *Strategy #1*: Patches used to train the MF-VPINN with $C_M = 9$. Each dot represents a patch P_i, its position is the center $\mathbf{c}_{P_i}$ of the patch, its size is proportional to the patch size h_i^2, and its color is associated with the quantity η_i^γ. (**a**) Representation of $\mathcal{P}_1$; (**b**) Representation of $\mathcal{P}_2$; (**c**) Representation of $\mathcal{P}_3$; (**d**) Representation of $\mathcal{P}_4$; (**e**) Representation of $\mathcal{P}_5$; (**f**) Representation of $\mathcal{P}_6$.

Training an MF-VPINN with such a strategy leads to more accurate results. The error decays are shown in Figure 6, whereas a comparison with the previous one will be presented in Section 3.3. The patch distributions, for $C_M = 4$ and $C_M = 9$, are shown in Figures 7 and 8, respectively. Analyzing such distributions, it can be noted that the patches still accumulate near the origin as expected. However, it is possible to observe that there are regions that are only covered by the largest patches. This phenomenon is more evident when $C_M = 4$. To avoid such a phenomenon, we aim at inserting more patches far from the origin in order to train the MF-VPINN in the entire domain with a more balanced set of patches.

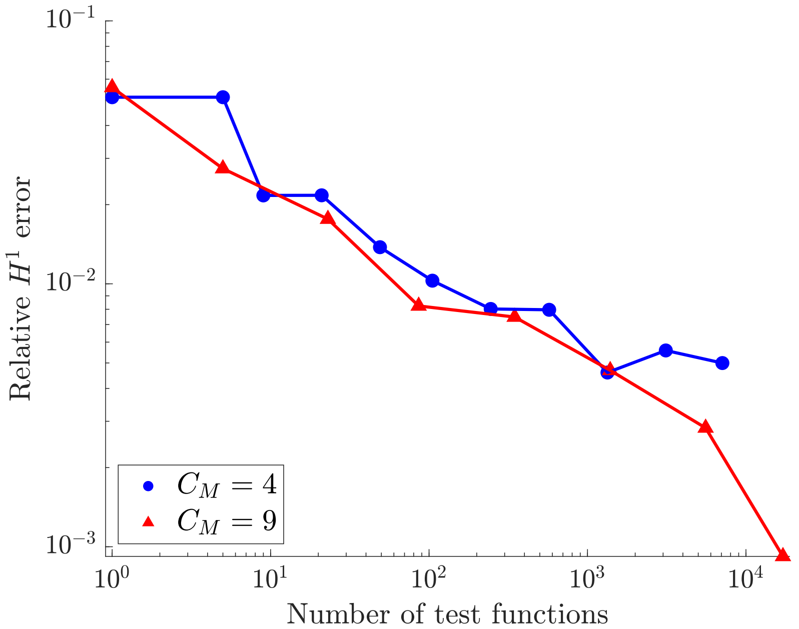


Figure 6. *Strategy #2*: Relative H^1 errors obtained at the end of each training iteration for $C_M = 4$ (blue circles) and $C_M = 9$ (red triangles).

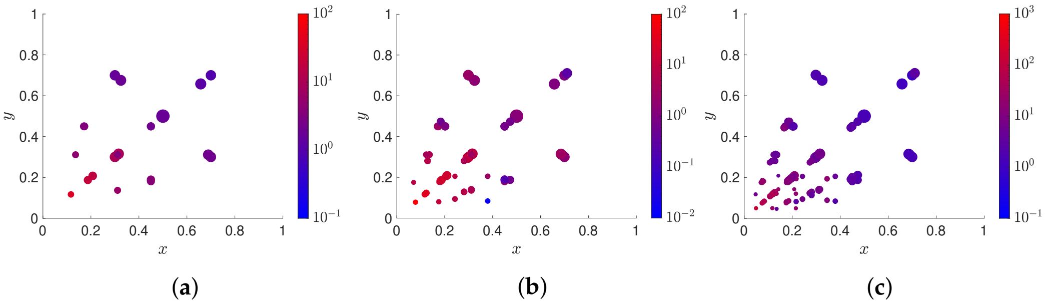


Figure 7. *Cont.*

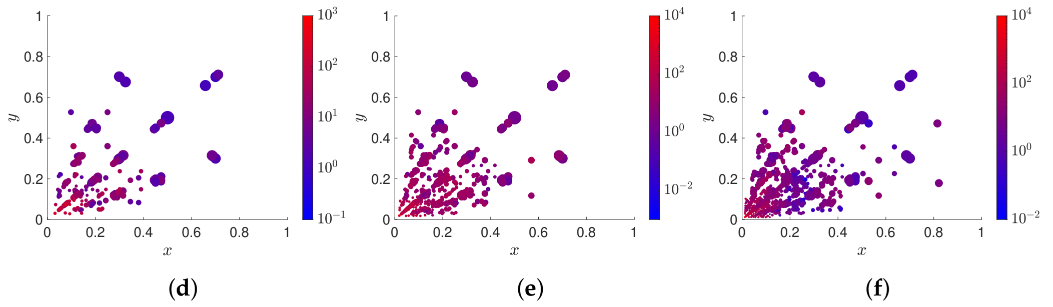


Figure 7. *Strategy #2*: Patches used to train the MF-VPINN with $C_M = 4$. Each dot represents a patch P_i, its position is the center $\boldsymbol{c}_{\boldsymbol{P}_i}$ of the patch, its size is proportional to the patch size h_i^2, and its color is associated with the quantity η_i^γ. (**a**) Representation of $\mathcal{P}_3$; (**b**) Representation of $\mathcal{P}_4$; (**c**) Representation of $\mathcal{P}_5$; (**d**) Representation of $\mathcal{P}_6$; (**e**) Representation of $\mathcal{P}_7$; (**f**) Representation of $\mathcal{P}_8$.

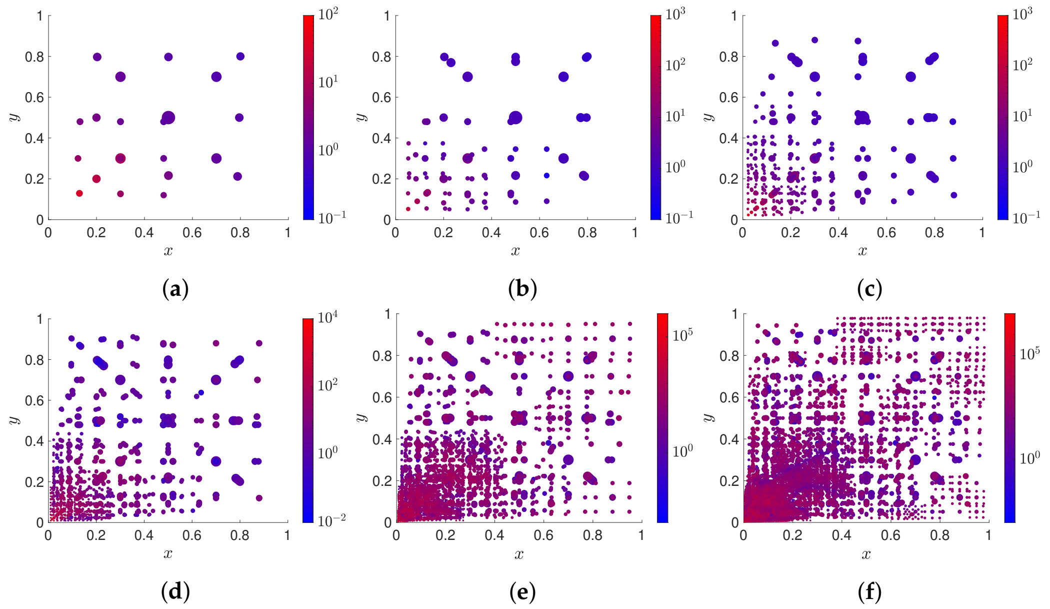


Figure 8. *Strategy #2*: Patches used to train the MF-VPINN with $C_M = 9$. Each dot represents a patch P_i, its position is the center $\boldsymbol{c}_{\boldsymbol{P}_i}$ of the patch, its size is proportional to the patch size h_i^2, and its color is associated with the quantity η_i^γ. (**a**) Representation of $\mathcal{P}_2$; (**b**) Representation of $\mathcal{P}_3$; (**c**) Representation of $\mathcal{P}_4$; (**d**) Representation of $\mathcal{P}_5$; (**e**) Representation of $\mathcal{P}_6$; (**f**) Representation of $\mathcal{P}_7$.

Strategy #3: Fixed patch centers and small level gap strategy

In order to ensure better patch distributions, let us consider a new criterion to choose the position and the size of the new patches. We name this strategy the *small-level gap strategy* because it penalizes patch distributions with large differences between the levels of the smallest patches and the ones of the largest patches.

We denote by k-th level patch any patch P_i such that $P_i \in \mathcal{P}_k$ and $P_i \notin \mathcal{P}_{k'}$ for any $k' < k$. With this notation, it is possible to group all the patches according to their level. To do so, we denote by L_ℓ the set of k-th level patches with $k \le \ell$. Let us consider the m-th training iteration. We define $\boldsymbol{\eta}_{\text{sort}}^\ell$ as the array containing the elements η_i^γ of $\boldsymbol{\eta}_{\text{sort}}$ (maintaining the same ordering) such that $P_i \in L_\ell$. We also denote by $\overline{\boldsymbol{\eta}}_{m,\ell}$ the array containing the first $\tau_m^\ell = \min\{\tau_m, \dim(L_\ell)\}$ elements of $\boldsymbol{\eta}_{\text{sort}}^\ell$. Note that $\overline{\boldsymbol{\eta}}_{m,\ell}$ is the equivalent of $\overline{\boldsymbol{\eta}}_m$ for patches in L_ℓ.

In order to generate the new patches in $\mathcal{P}_{m+1} \backslash \mathcal{P}_m$, let us add C_M new patches in any patch P_i such that $\eta_i^\gamma \in \overline{\boldsymbol{\eta}}_m \cup \overline{\boldsymbol{\eta}}_{m,\ell}$. The centers and sizes of the new patches are chosen as in *Strategy* #2. This allows us to exploit the fact that $P_i \subsetneq \cup_{k=1}^{C_M} P_i^k$ to remove the patches P_i such that $\eta_i^\gamma \in \overline{\boldsymbol{\eta}}_m \cup \overline{\boldsymbol{\eta}}_{m,\ell}$ from the new set of patches $\mathcal{P}_{m+1}$. We remark that such patches cannot be removed when the centers are randomly chosen as in *Strategy #1* because, in that case, $\mathcal{P}_{m+1}$ would not be a cover of Ω anymore.

We also highlight that, removing the patches P_i such that $\eta_i^\gamma \in \overline{\boldsymbol{\eta}}_m \cup \overline{\boldsymbol{\eta}}_{m,\ell}$ and choosing $A_{\text{ratio}} = 1$, it is possible to satisfy the inequality

$$\sum_{P_i \in \mathcal{P}_{m+1}} |P_i| \leq C|\Omega|,$$

for any $m \in \mathbb{N}$ and with $C > 0$ independent of m. Such a bound on the sum of the area of the patches is useful to ensure that there exists a number $N_{\text{patch\_per\_point}}$ such that any point inside Ω belongs to at most $N_{\text{patch\_per\_point}}$ patches. This property is useful to derive global error indicators. We choose to maintain $A_{\text{ratio}} = 1.25$ to compare the numerical results with the ones obtained using the previous strategies and to consider overlapping patches.

We train an MF-VPINN with $C_M = 4$ and $C_M = 9$ as in the previous tests. The corresponding error decays are shown in Figure 9. It can be observed that the error decreases in a smoother way and that, as in the previous tests, choosing $C_M = 4$ or $C_M = 9$ does not lead to significant differences in the error behavior. The patches used during the training are represented in Figures 10 and 11. We highlight that, when compared with the patch distributions in *Strategy #2*, there exist much more patches far from the origin, and, most importantly, the closer the center of a patch to the origin, the smaller its size. Even though the error decays with $C_M = 4$ and $C_M = 9$ are qualitatively similar, it should be noted that the patch distribution with $C_M = 9$ is more skewed. In fact. its patches can be clustered into two subgroups: the first one containing larger patches and covering most of the domain the second one containing only small patches with centers very close to the origin. A similar distribution is obtained with $C_M = 4$, even though it is characterized by a smoother transition between large and small patches.

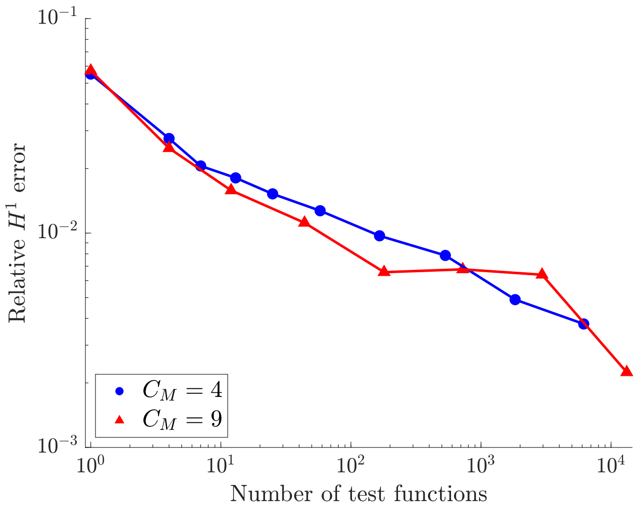


Figure 9. *Strategy #3*: Relative H^1 errors obtained at the end of each training iteration for $C_M = 4$ (blue circles) and $C_M = 9$ (red triangles).

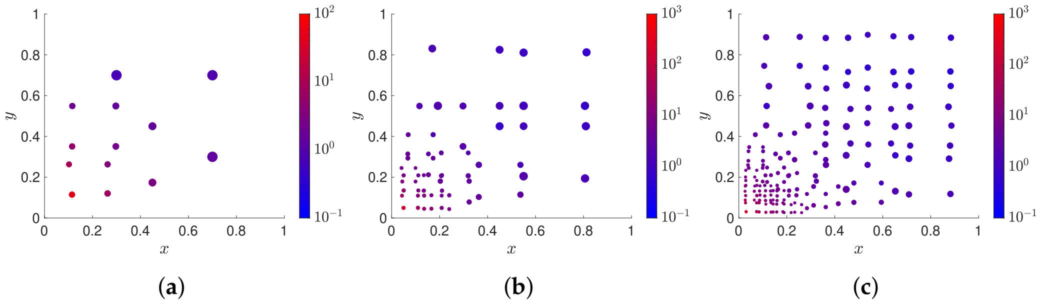


Figure 10. *Cont.*

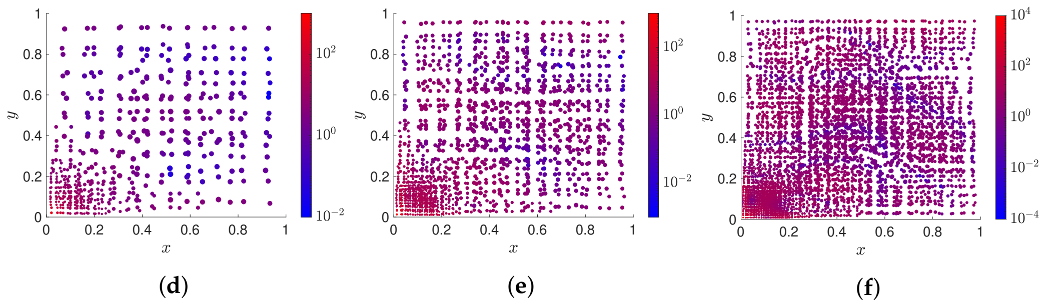


Figure 10. *Strategy* #3: Patches used to train the MF-VPINN with $C_M = 4$. Each dot represents a patch P_i, its position is the center $\boldsymbol{c}_{\boldsymbol{P}_i}$ of the patch, its size is proportional to the patch size h_i^2, and its color is associated with the quantity η_i^γ. (**a**) Representation of $\mathcal{P}_3$; (**b**) Representation of $\mathcal{P}_5$; (**c**) Representation of $\mathcal{P}_6$; (**d**) Representation of $\mathcal{P}_7$; (**e**) Representation of $\mathcal{P}_8$; (**f**) Representation of $\mathcal{P}_9$.

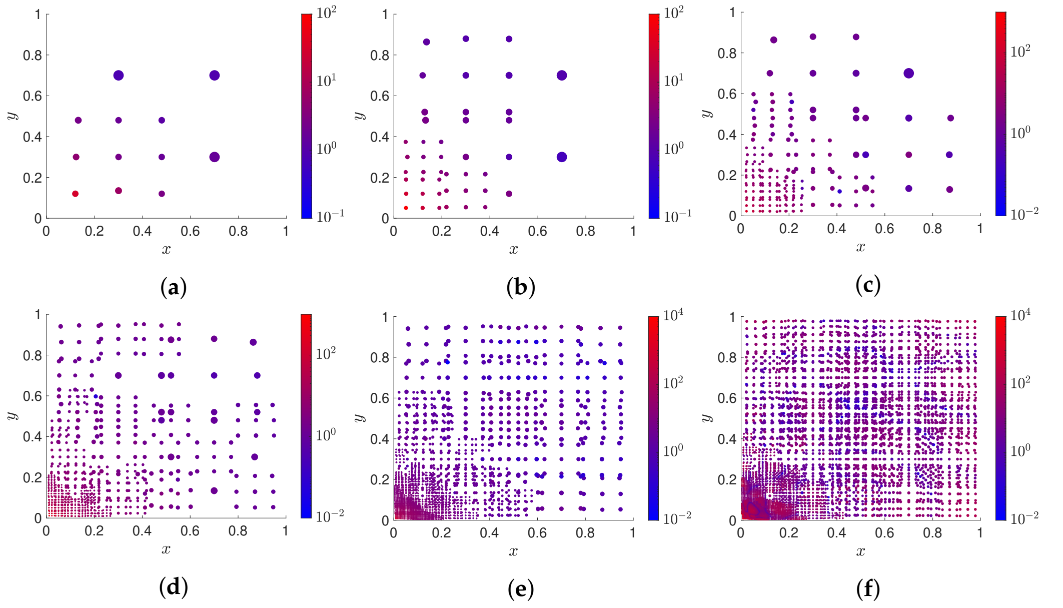


Figure 11. *Strategy* #3: Patches used to train the MF-VPINN with $C_M = 9$. Each dot represents a patch P_i, its position is the center $\boldsymbol{c}_{\boldsymbol{P}_i}$ of the patch, its size is proportional to the patch size h_i^2, and its color is associated with the quantity η_i^γ. (**a**) Representation of $\mathcal{P}_2$; (**b**) Representation of $\mathcal{P}_3$; (**c**) Representation of $\mathcal{P}_4$; (**d**) Representation of $\mathcal{P}_5$; (**e**) Representation of $\mathcal{P}_6$; (**f**) Representation of $\mathcal{P}_7$.

In both cases, it can be observed that there are no large patches very close to small ones. This is in contrast with the distributions obtained in *Strategy* #2 and leads to more stable solvers. Indeed, even though the test functions are not related to a global triangulation on the entire domain Ω, the current loss function is very similar to the one used in a standard VPINN with a good-quality mesh, i.e., a mesh in which neighboring elements are similar in size and shape. On the other hand, in *Strategy* #2, there exist large patches that are very close to small ones; this is equivalent to training a VPINN on a very poor-quality mesh. Such meshes, in the context of FEM, are strictly related to convergence and accuracy issues.

3.3. The Importance of the Error Indicator

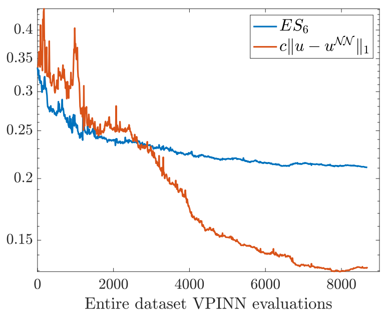
As discussed in the previous sections, we use the error indicator described in Section 2.3 to interrupt the training and to decide where the new patches have to be inserted to maximize the accuracy. In this section, the advantages of such a choice are described.

Since each set $\mathcal{P}_m$ is a cover of Ω, the quantity $ES_m = \sum_{i=1}^{\dim(\mathcal{P}_m)} \eta_i$ is an indicator of the global H^1 error $\|u - u^{\mathcal{NN}}\|_1$ on the entire domain Ω. Therefore, tracking its behavior during the training is equivalent to tracking that of the unknown H^1 error. Such information is used to implement an early stopping strategy to reduce the computational cost of the iterative training. At the beginning of the m-th training iteration, all the vectors and sparse matrices required to compute ES_m are computed in a preprocessing phase. When such data structures are available, the error indicator can be assembled suitably combining basic algebraic operations.

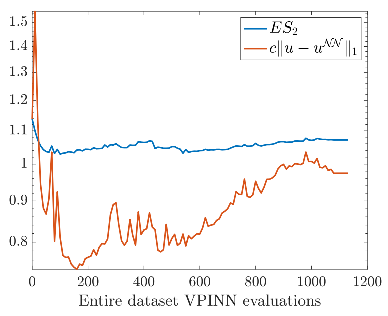
We assemble ES_m every N_{check} epochs and store the best value obtained during the training, together with the corresponding neural network trainable parameters. Then, if no improvements are obtained in $p \cdot N_{\text{check}}$ epochs, the training is interrupted and the neural network parameters associated with the best value of ES_m are restored. Here, p is a tunable parameter named *patience*. The first N^m_{negl} epochs are neglected because they are often characterized by strong oscillations due to the optimizer initialization and the different loss functions. In the numerical experiment, we use $N_{\text{check}} = 10$, $p = 10$, $N^m_{\text{negl}} = 100(m+1)$.

Two typical scenarios are shown in Figure 12. In the top row, the behaviors of ES_m and of $c\|u - u^{\mathcal{NN}}\|_1$ are shown. Here, c is a scaling parameter used for visualization purposes, chosen such that ES_m and $c\|u - u^{\mathcal{NN}}\|_1$ coincide at the beginning of the training. Indeed, $\|u - u^{\mathcal{NN}}\|_1$ is about two orders of magnitude smaller than ES_m. Nevertheless, it can be noted that these two quantities display very similar behaviors during the training. In the bottom row, instead, we represent the corresponding loss function decay. The left column is associated with the training performed using the patches in $\mathcal{P}_6$ shown in Figure 5f and the right column with the one performed using the patches in $\mathcal{P}_2$ in Figure 11a. We remark that the loss function, ES_m and $c\|u - u^{\mathcal{NN}}\|$ are evaluated in the same epochs and that, in real applications, it is not possible to explicitly compute $c\|u - u^{\mathcal{NN}}\|$ since u is not known. Moreover, since we use the L-BFGS optimizer, the neural network is evaluated multiple times on the entire training set in each epoch. Therefore, on the x-axis of Figure 12 we show the number of neural network evaluations instead of the number of epochs.

It can be noted that the behavior of the quantities shown in the left column is qualitatively different from the ones in the right column. In fact, when the MF-VPINN is trained with $\mathcal{P}_6$ of Figure 5f, the error, the error indicator, and the loss function decrease in similar ways. Therefore, there is no need to interrupt the training early since the accuracy is improving, minimizing the loss function. On the other hand, when the MF-VPINN is trained with the $\mathcal{P}_2$ of Figure 11a, the loss decreases even when the error and the error indicator increase or remain constant. In this case, it is convenient to interrupt the training, since minimizing the loss function further would lead to more severe overfitting phenomena and a loss in accuracy and efficiency. At the end of the training, the neural network's trainable parameters corresponding to the best value of ES_2 are restored. We highlight that such a phenomenon, observed in [46] too, highlights the fact that the minimization of the loss function generates spurious oscillations that cannot be controlled and ruin the model accuracy. The issue can be partially alleviated with the adopted regularization or completely removed using inf-sup stable models as in [47].

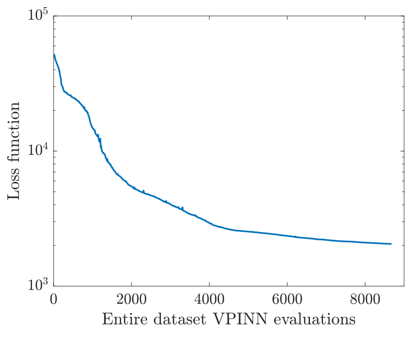


(**a**)

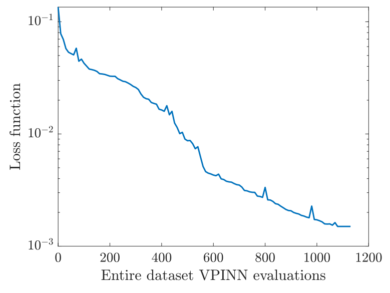


(**b**)

Figure 12. *Cont.*



(**c**)

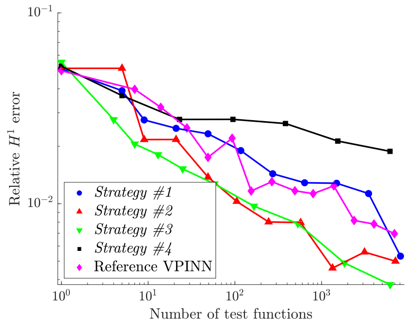


(**d**)

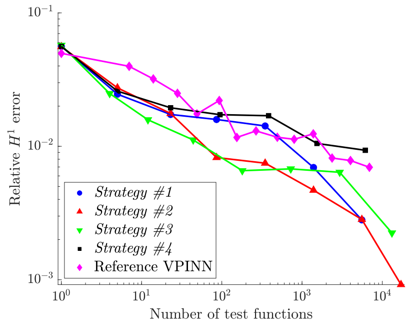
Figure 12. Top row: error indicator ES_m and rescaled H^1 error $c\|u - u^{\mathcal{NN}}\|$. Bottom row: loss function. Left column: curves for the training with patches in $\mathcal{P}_6$ shown in Figure 5f. Right column: curves for the training with patches in $\mathcal{P}_2$ in Figure 11a. (**a**) ES_6 and $c\|u - u^{\mathcal{NN}}\|$ for patches in Figure 5f; (**b**) ES_2 and $c\|u - u^{\mathcal{NN}}\|$ for patches in Figure 11a; (**c**) Loss function for patches in Figure 5f; (**d**) Loss function for patches in Figure 11a.

Strategy #4: Adaptive strategy without the error indicator

Let us now analyze the consequences of choosing the position of the new patches without using the error indicator. To do so, we consider *Strategy #1* but, instead of considering the new centers inside the patches P_i with the highest values of η_i^{γ}, we add them inside the patches with the highest values of $r^2_{h,i}(u^{\mathcal{NN}})$. Using the equation residuals is a common choice in PINN adaptivity because the residuals describe how accurately the neural network satisfies the PDE at that point. The obtained error decay is shown in Figure 13. It can be seen that the accuracy is worse than the ones obtained with the other strategies and that the convergence rate with respect to the number of patches is lower. In such a figure, we also compare the MF-VPINN with a standard VPINN trained with test functions defined on Delaunay meshes. Note that, when *Strategy #2* or *Strategy #3* is adopted, the MF-VPINN is more accurate than a simple VPINN, even though its main advantage resides in being a meshfree method.



(**a**)



(**b**)

Figure 13. Comparison between the relative H^1 errors obtained at the end of each training iteration with different strategies to choose the position of the new patches. (**a**) $C_M = 4$; (**b**) $C_M = 9$.

We highlight that, due to the low regularity of the solution, the expected convergence rate with respect to the number of test functions of an FEM solution computed on uniform refinements is $-1/3$. Note that the convergence rate of the proposed MF-VPINN method is still close to $-1/3$, even though it is a meshfree method (see Table 1). For completeness, we also remark that, if an adaptive FEM is used, the rate of convergence depends on the FEM order.

Table 1. Rates of convergence with respect to the number of test functions.

C_M	*Strategy #1*	*Strategy #2*	*Strategy #3*	*Strategy #4*	Reference VPINN
4	-0.213	-0.295	-0.283	-0.105	-0.232
9	-0.294	-0.376	-0.287	-0.182	-0.232

Coherently with Figure 13, the best strategies are *Strategy #2* and *Strategy #3*, whereas the worst one is *Strategy #4*, which does not exploit the error indicator. The poor performance of *Strategy #4* can also be explained by analyzing the corresponding patch distribution. Such distribution is shown in Figure 14 for $C_M = 4$ and in Figure 15 for $C_M = 9$. These plots highlight that the patches do not accumulate near the origin because the residuals of the patches closer to it are not significantly higher than the other ones. For example, note the different colors in Figures 4 and 14, since in both cases, we randomly choose the position of $C_M = 4$ centers inside the selected patches. Such a property is explained by the fact that, in order to minimize the loss function, the optimizer does not focus on specific regions of the domain. Therefore, the orders of magnitude of all the residuals with similar sizes are very close to each other regardless of the position of the corresponding patches. As discussed regarding Figure 12, we can conclude that the value of the residuals is not a good indicator of the actual error.

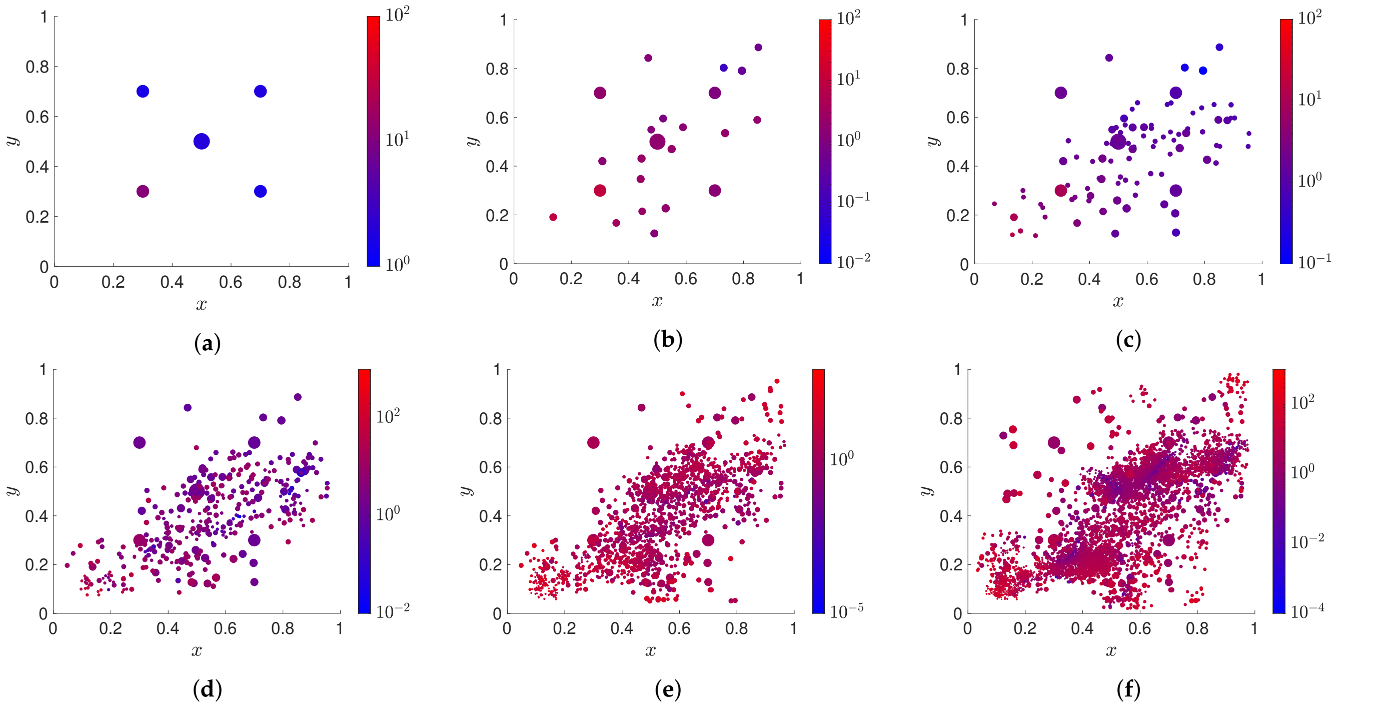


Figure 14. *Strategy #4*: Patches used to train the MF-VPINN with $C_M = 4$. Each dot represents a patch P_i, its position is the center $\boldsymbol{c}_{P_i}$ of the patch, its size is proportional to the patch size h_i^2, and its color is associated with the quantity $r_{h,i}^2(u^{\mathcal{NN}})$. (**a**) Representation of $\mathcal{P}_1$; (**b**) Representation of $\mathcal{P}_2$; (**c**) Representation of $\mathcal{P}_3$; (**d**) Representation of $\mathcal{P}_4$; (**e**) Representation of $\mathcal{P}_5$; (**f**) Representation of $\mathcal{P}_6$.

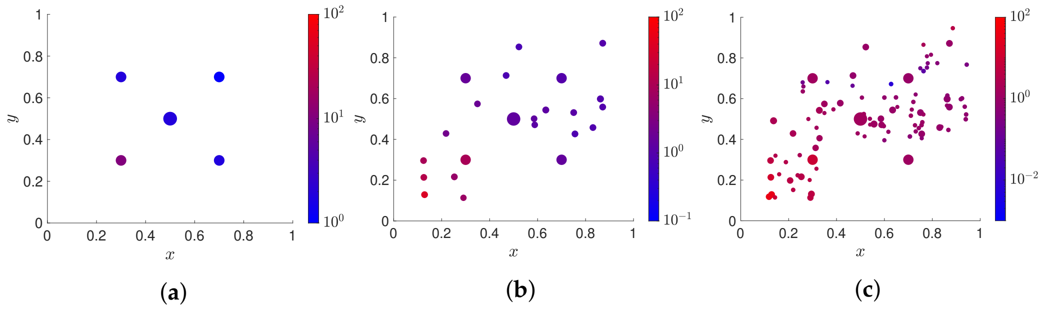


Figure 15. *Cont.*

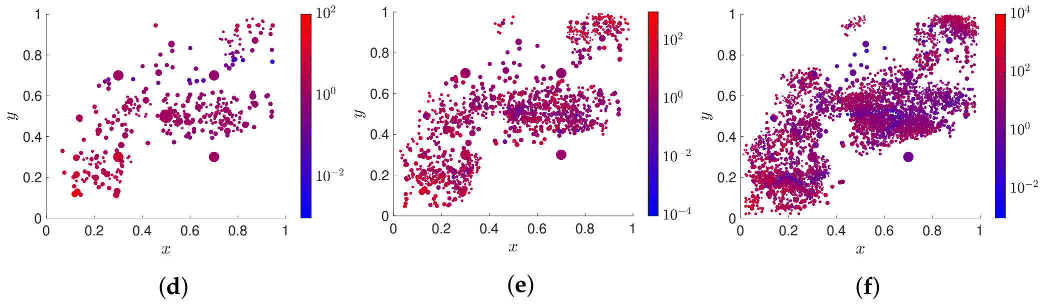


Figure 15. *Strategy #4*: Patches used to train the MF-VPINN with $C_M = 9$. Each dot represents a patch P_i, its position is the center $\boldsymbol{c}_{P_i}$ of the patch, its size is proportional to the patch size h_i^2, and its color is associated with the quantity $r_{h,i}^2(u^{\mathcal{NN}})$. (**a**) Representation of $\mathcal{P}_1$; (**b**) Representation of $\mathcal{P}_2$; (**c**) Representation of $\mathcal{P}_3$; (**d**) Representation of $\mathcal{P}_4$; (**e**) Representation of $\mathcal{P}_5$; (**f**) Representation of $\mathcal{P}_6$.

3.4. Extension to More a Complex Domain

In this section, we present some ideas that can be used to apply the method to more complex domains.

Let us consider a domain Ω_2 with some internal holes and boundary $\partial\Omega_2 = \Gamma_2$. In particular, $\Omega_2 = (0,1)^2 \backslash (\cup_{i=1}^4 H_i)$, where H_i, $i = 1,2,3,4$ are rectangular holes with centers $\mathbf{c}_{H_i}$ defined as

$$\mathbf{c}_{H_1} = \left(\frac{9}{26}, \frac{9}{34}\right), \qquad \mathbf{c}_{H_2} = \left(\frac{17}{26}, \frac{9}{34}\right),$$

$$\mathbf{c}_{H_3} = \left(\frac{9}{26}, \frac{25}{34}\right), \qquad \mathbf{c}_{H_4} = \left(\frac{17}{26}, \frac{25}{34}\right),$$

and basis and height equal $\frac{1}{26}$ and $\frac{1}{34}$, respectively.

In this domain, we consider the Poisson problem:

$$\begin{cases} -\Delta u = f & \text{in } \Omega_2, \\ u = g & \text{on } \Gamma_2, \end{cases} \tag{35}$$

with f and g such that the exact solution is

$$u(x,y) = \frac{1}{C_u}\left[x(x-1)\left(x-\frac{4}{13}\right)\left(x-\frac{5}{13}\right)\left(x-\frac{8}{13}\right)\left(x-\frac{9}{13}\right)\cdot \right. \\ \left. y(y-1)\left(y-\frac{4}{17}\right)\left(y-\frac{5}{17}\right)\left(y-\frac{12}{17}\right)\left(y-\frac{13}{17}\right)\right], \tag{36}$$

normalized through the constant $\frac{1}{C_u}$ to assume value 1 in $\left(\frac{2}{13}, \frac{2}{17}\right)$. This function is represented in Figure 16.

We extend the approaches proposed in Section 3.2 by adding a cutting procedure after the generation of the new patches. Note that, in particular, all the patches are already completely inside the square $[0,1]^2$ when we apply the cutting procedure, and we can thus focus only on the holes. When a patch intersects more than one hole, we recursively remove it from $\mathcal{P}_m$, we subdivide the corresponding region in 4 overlapping patches, and we add them to $\mathcal{P}_m$ until all the generated patches intersect at most one hole. Moreover, we observe that the region $P_i \backslash H_j$ inside the patch $P_i \in \mathcal{P}_m$ and outside the hole H_j, $j = 1,2,3,4$, can always be covered by the union of at most four rectangles. When a generated patch intersects a hole, we thus remove the patch and generate the minimum number of patches (at most four) that are as large possible and whose union is the region $P_i \backslash H_j$.

To avoid numerical instabilities, when this cutting procedure generates a patch with an aspect ratio larger than 100 or with an area more than 100 times smaller than the original uncutted patch, the new patches are removed from $\mathcal{P}_m$. This implies that it is not possible to remove the patches associated with the highest error indicators as in *Strategy #3* because otherwise, the union of all the patches would not cover the entire domain. We thus present numerical results only for *Strategy #1* and *Strategy #2*.

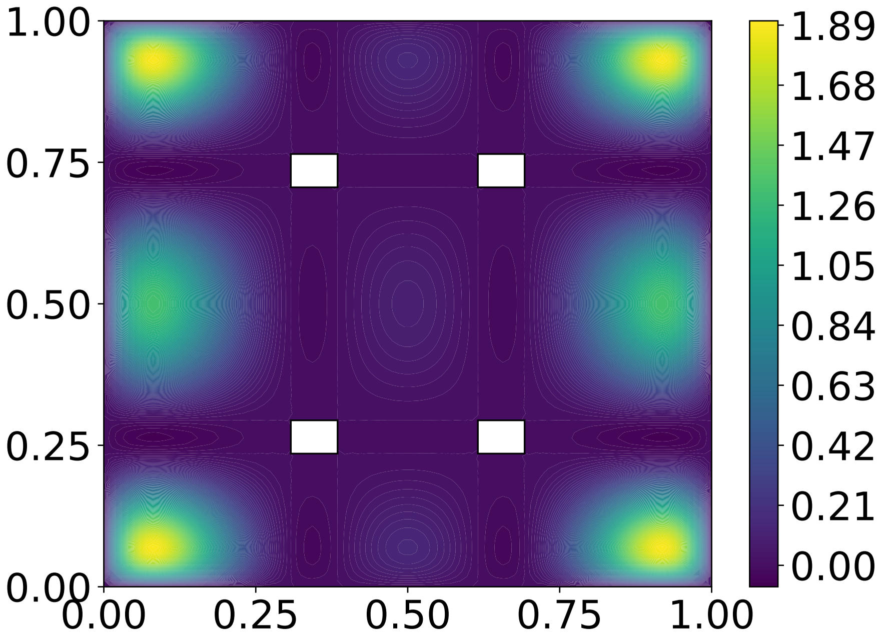


Figure 16. Graphical representation of the solution u in (36).

The obtained error decays are shown in Figure 17 for *Strategy #1* and *Strategy #2* with $C_M = 4$ and $C_M = 9$. The first and second errors are computed with the patches generated by cutting the patches in $\mathcal{P}_0$ and $\mathcal{P}_1$, respectively, whereas the third and fourth errors are obtained by refining the previous patches with the error indicator as previously described. Note that the first and second errors are very close for all the curves since the strategy and the value of C_M does not influence the training and that both strategies converge better with $C_M = 4$. The final patch distributions are displayed in Figure 18. Here, we can see that the inner part of the domain is covered by a few large patches, whereas the distribution is denser closer to the external boundary of Ω_2, where the solution is more oscillating.

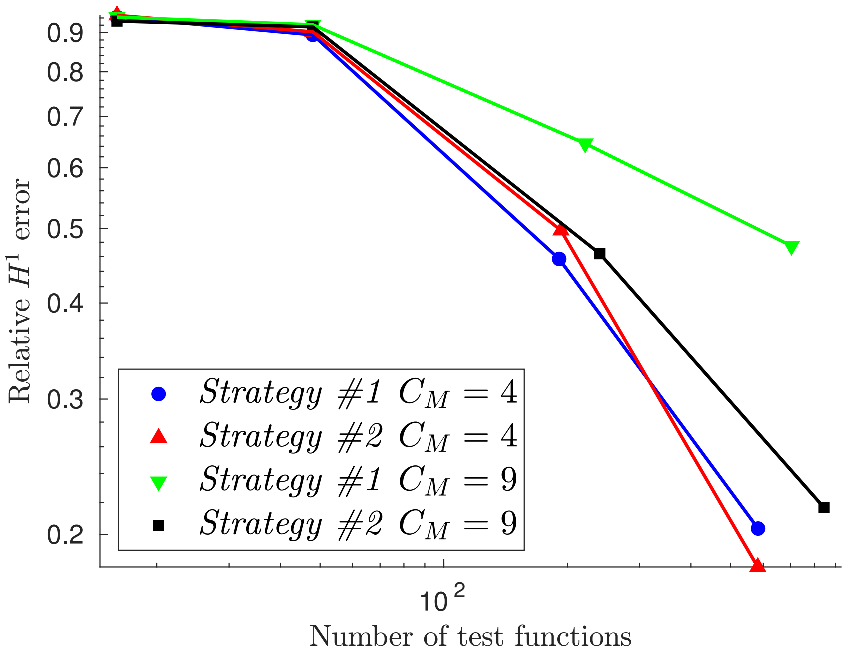


Figure 17. Relative H^1 errors obtained by solving problem (35).

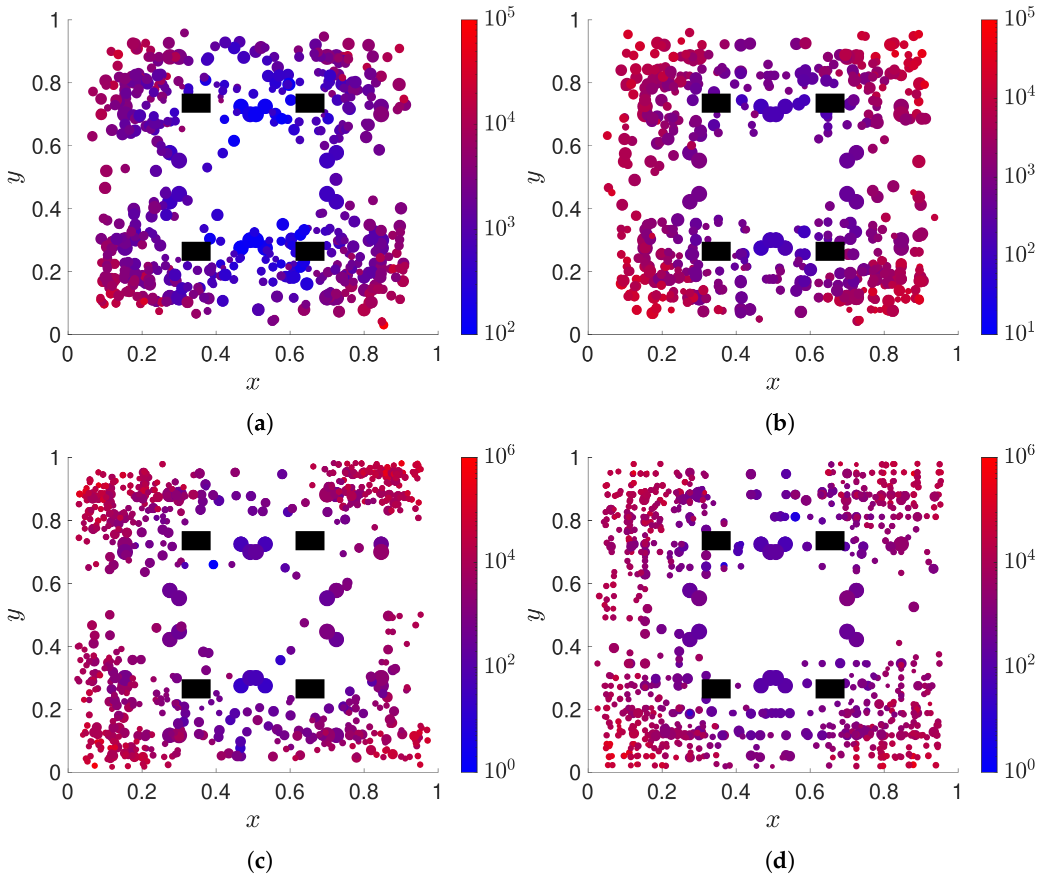


Figure 18. Problem (35): Representation of the last set of patches obtained with the different strategies. Each dot represents a patch P_i, its position is the center $\boldsymbol{c}_{P_i}$ of the patch, its size is proportional to the patch size h_i^2, and its color is associated with the quantity $r_{h,i}^2(u^{\mathcal{NN}})$. The black rectangles represent the holes H_i, $i = 1, 2, 3, 4$. (**a**) *Strategy #1*, $C_M = 4$; (**b**) *Strategy #2*, $C_M = 4$; (**c**) *Strategy #1*, $C_M = 9$; (**d**) *Strategy #2*, $C_M = 9$.

4. Conclusions and Discussion

In this work, we presented a Meshfree Variational-Physics-Informed Neural Network (MF-VPINN). It is a PINN trained using the PDE variational formulation that does not require the generation of a global triangulation of the entire domain. In order to generate the test functions involved in the loss computation, we use an a posteriori error estimator based on the one discussed in [46]. Using such an error estimator, it is possible to add test functions only in regions in which the error is higher, thus increasing the efficiency of the method.

We highlight that the main advantages of the method are that it is meshfree, as it requires only a covering of the domain with patches that can be of different shapes and that it automatically improves the solution with the application of local patches without requiring a global mesh manipulation. It can be therefore used in domains where it is expensive or impossible to generate a mesh. On the other hand, if a mesh suitable to describe the solution can be generated, a standard VPINN is preferable since the implementation is simpler and the convergence rate with respect to the number of test functions is higher.

We discuss several strategies to generate the set of test functions. We observe that adding a few test functions inside the patches associated with higher errors while ensuring a smooth transition between regions with large patches and regions with small patches is the best way to obtain accurate solutions. We also show that, if the a posteriori error indicator is not used, the model's accuracy decreases and the training is slower.

In this paper, we only focus on second-order elliptic problem even though VPINNs can be used to solve more complex problems. In a forthcoming paper, we will adapt the a posteriori error estimator and analyze the MF-VPINN performance on other PDEs. Moreover, we are interested in the analysis of the approach in more complex domains (in which the patches have to be suitably deformed) and in high-dimensional problems, where using a standard VPINN is not practical.

Author Contributions: Conceptualization, S.B. and M.P.; methodology, S.B. and M.P.; software, M.P.; validation, M.P.; formal analysis, S.B. and M.P.; investigation, S.B. and M.P.; resources, S.B.; data curation, M.P.; writing—original draft preparation, M.P.; visualization, M.P.; supervision, S.B.; project administration, S.B.; funding acquisition, S.B. All authors have read and agreed to the published version of the manuscript.

Funding: The author S.B. kindly acknowledges partial financial support provided by PRIN project "Advanced polyhedral discretisations of heterogeneous PDEs for multiphysics problems" (No. 20204LN5N5_003) and by PNRR M4C2 project of CN00000013 National center for HPC, Big Data and Quantum Computing (HPC) (CUP: E13C22000990001). The author M.P. kindly acknowledges the financial support provided by the Politecnico di Torino where the research was carried out.

Data Availability Statement: The data are available upon reasonable request.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Lagaris, I.; Likas, A.; Fotiadis, D. Artificial neural network methods in quantum mechanics. *Comput. Phys. Commun.* **1997**, *104*, 1–14. [CrossRef]
2. Lagaris, I.; Likas, A.; Fotiadis, D. Artificial neural networks for solving ordinary and partial differential equations. *IEEE Trans. Neural Netw.* **1998**, *9*, 987–1000. [CrossRef] [PubMed]
3. Lagaris, I.; Likas, A.; Papageorgiou, D. Neural-network methods for boundary value problems with irregular boundaries. *IEEE Trans. Neural Netw.* **2000**, *11*, 1041–1049. [CrossRef] [PubMed]
4. Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G.S.; Davis, A.; Dean, J.; Devin, M .; et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. 2015. Available online : https://www.tensorflow.org (accessed on 15 September 2024).
5. Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*; Curran Associates, Inc.: Red Hook, NY, USA, 2019; pp. 8024–8035.
6. Bradbury, J.; Frostig, R.; Hawkins, P.; Johnson, M.J.; Leary, C.; Maclaurin, D.; Necula, G.; Paszke, A.; VanderPlas, J.; Wanderman-Milne, S.; et al. JAX: Composable Transformations of Python+NumPy Programs. 2018. Available online: http://github.com/google/jax (accessed on 15 September 2024).
7. Raissi, M.; Perdikaris, P.; Karniadakis, G. Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations. *arXiv* **2017**, arXiv:1711.10561.
8. Raissi, M.; Perdikaris, P.; Karniadakis, G. Physics informed deep learning (part ii): Data-driven solutions of nonlinear partial differential equations. *arXiv* **2017**, arXiv:1711.10566.
9. Raissi, M.; Perdikaris, P.; Karniadakis, G. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *J. Comput. Phys.* **2019**, *378*, 686–707. [CrossRef]
10. Pu, J.; Li, J.; Chen, Y. Solving localized wave solutions of the derivative nonlinear Schrödinger equation using an improved PINN method. *Nonlinear Dyn.* **2021**, *105*, 1723–1739. [CrossRef]
11. Yuan, L.; Ni, Y.; Deng, X.; Hao, S. A-PINN: Auxiliary physics informed neural networks for forward and inverse problems of nonlinear integro-differential equations. *J. Comput. Phys.* **2022**, *462*, 111260. [CrossRef]
12. Guo, Q.; Zhao, Y.; Lu, C.; Luo, J. High-dimensional inverse modeling of hydraulic tomography by physics informed neural network (HT-PINN). *J. Hydrol.* **2023**, *616*, 128828. [CrossRef]
13. Demo, N.; Strazzullo, M.; Rozza, G. An extended physics informed neural network for preliminary analysis of parametric optimal control problems. *Comput. Math. Appl.* **2023**, *143*, 383–396. [CrossRef]
14. Gao, H.; Sun, L.; Wang, J. PhyGeoNet: Physics-informed geometry-adaptive convolutional neural networks for solving parameterized steady-state PDEs on irregular domain. *J. Comput. Phys.* **2021**, *428*, 110079. [CrossRef]
15. Yuyao, C.; Lu, L.; Karniadakis, G.; Dal Negro, L. Physics-informed neural networks for inverse problems in nano-optics and metamaterials. *Opt. Express* **2020**, *28*, 11618–11633. [CrossRef]
16. Tartakovsky, A.; Marrero, C.; Perdikaris, P.; Tartakovsky, G.; Barajas-Solano, D. Learning parameters and constitutive relationships with physics informed deep neural networks. *arXiv* **2018**, arXiv:1808.03398.
17. Chen, Z.; Liu, Y.; Sun, H. Physics-informed learning of governing equations from scarce data. *Nat. Commun.* **2021**, *12*, 6136. [CrossRef] [PubMed]

18. Weinan, E.; Yu, B. The Deep Ritz method: A deep learning-based numerical algorithm for solving variational problems. *Commun. Math. Stat.* **2018**, *6*, 1–12.
19. Müller, J.; Zeinhofer, M. Error estimates for the deep Ritz method with boundary penalty. In Proceedings of the Mathematical and Scientific Machine Learning, PMLR, Beijing, China, 15–17 August 2022; pp. 215–230.
20. Lu, Y.; Lu, J.; Wang, M. A priori generalization analysis of the deep Ritz method for solving high dimensional elliptic partial differential equations. In Proceedings of the Conference on Learning Theory. PMLR, Boulder, CO, USA, 15–19 August 2021; pp. 3196–3241.
21. Sirignano, J.; Spiliopoulos, K. DGM: A deep learning algorithm for solving partial differential equations. *J. Comput. Phys.* **2018**, *375*, 1339–1364. [CrossRef]
22. Al-Aradi, A.; Correia, A.; Jardim, G.; de Freitas Naiff, D.; Saporito, Y. Extensions of the deep Galerkin method. *Appl. Math. Comput.* **2022**, *430*, 127287. [CrossRef]
23. Li, J.; Zhang, W.; Yue, J. A deep learning Galerkin method for the second-order linear elliptic equations. *Int. J. Numer. Anal. Model.* **2021**, *18*, 427–441.
24. Smith, B.F. Domain decomposition methods for partial differential equations. In *Parallel Numerical Algorithms*; Springer: Berlin/Heidelberg, Germany, 1997; pp. 225–243.
25. Toselli, A.; Widlund, O. *Domain Decomposition Methods-Algorithms and Theory*; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2006; Volume 34.
26. Jagtap, A.; Kharazmi, E.; Karniadakis, G. Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems. *Comput. Methods Appl. Mech. Eng.* **2020**, *365*, 113028. [CrossRef]
27. Shukla, K.; Jagtap, A.D.; Karniadakis, G.E. Parallel physics-informed neural networks via domain decomposition. *J. Comput. Phys.* **2021**, *447*, 110683. [CrossRef]
28. Jagtap, A.; Karniadakis, G. Extended physics-informed neural networks (XPINNs): A generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations. *Commun. Comput. Phys.* **2020**, *28*, 2002–2041.
29. Moseley, B.; Markham, A.; Nissen-Meyer, T. Finite Basis Physics-Informed Neural Networks (FBPINNs): A scalable domain decomposition approach for solving differential equations. *Adv. Comput. Math.* **2023**, *49*, 62. [CrossRef]
30. Viana, F.; Nascimento, R.; Dourado, A.; Yucesan, Y. Estimating model inadequacy in ordinary differential equations with physics-informed neural networks. *Comput. Struct.* **2021**, *245*, 106458. . [CrossRef]
31. Yang, L.; Meng, X.; Karniadakis, G. B-PINNs: Bayesian physics-informed neural networks for forward and inverse PDE problems with noisy data. *J. Comput. Phys.* **2021**, *425*, 109913 . [CrossRef]
32. Yang, L.; Zhang, D.; Karniadakis, G. Physics-Informed Generative Adversarial Networks for Stochastic Differential Equations. *SIAM J. Sci. Comput.* **2020**, *42*, A292–A317. [CrossRef]
33. Yucesan, Y.; Viana, F. Hybrid physics-informed neural networks for main bearing fatigue prognosis with visual grease inspection. *Comput. Ind.* **2021**, *125*, 103386. . [CrossRef]
34. Zhu, Y.; Zabaras, N.; Koutsourelakis, P.; Perdikaris, P. Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. *J. Comput. Phys.* **2019**, *394*, 56–81. [CrossRef]
35. Pang, G.; Lu, L.; Karniadakis, G.E. fPINNs: Fractional physics-informed neural networks. *SIAM J. Sci. Comput.* **2019**, *41*, A2603–A2626. [CrossRef]
36. Liu, Z.; Wang, Y.; Vaidya, S.; Ruehle, F.; Halverson, J.; Soljačić, M.; Hou, T.Y.; Tegmark, M. Kan: Kolmogorov-arnold networks. *arXiv* **2024**, arXiv:2404.19756.
37. Koenig, B.C.; Kim, S.; Deng, S. KAN-ODEs: Kolmogorov-Arnold Network Ordinary Differential Equations for Learning Dynamical Systems and Hidden Physics. *arXiv* **2024**, arXiv:2407.04192.
38. Qian, K.; Kheir, M. Investigating KAN-Based Physics-Informed Neural Networks for EMI/EMC Simulations. *arXiv* **2024**, arXiv:2405.11383.
39. Kumar, V.; Gleyzer, L.; Kahana, A.; Shukla, K.; Karniadakis, G.E. Mycrunchgpt: A llm assisted framework for scientific machine learning. *J. Mach. Learn. Model. Comput.* **2023**, *4*, 41–72. [CrossRef]
40. Beck, C.; Hutzenthaler, M.; Jentzen, A.; Kuckuck, B. An overview on deep learning-based approximation methods for partial differential equations. *Discret. Contin. Dyn. Syst. B* **2022**, *28*, 3697–3746. [CrossRef]
41. Cuomo, S.; Di Cola, V.S.; Giampaolo, F.; Rozza, G.; Raissi, M.; Piccialli, F. Scientific Machine Learning Through Physics-Informed Neural Networks: Where we are and What's Next. *J. Sci. Comput.* **2022**, *92*, 88. [CrossRef]
42. Lawal, Z.; Yassin, H.; Lai, D.; Che Idris, A. Physics-Informed Neural Network (PINN) Evolution and Beyond: A Systematic Literature Review and Bibliometric Analysis. *Big Data Cogn. Comput.* **2022**, *6*, 140. [CrossRef]
43. Viana, F.A.; Subramaniyan, A.K. A survey of Bayesian calibration and physics-informed neural networks in scientific modeling. *Arch. Comput. Methods Eng.* **2021**, *28*, 3801–3830. [CrossRef]
44. Kharazmi, E.; Zhang, Z.; Karniadakis, G. VPINNs: Variational physics-informed neural networks for solving partial differential equations. *arXiv* **2019**, arXiv:1912.00873.
45. Kharazmi, E.; Zhang, Z.; Karniadakis, G. *hp*-VPINNs: Variational physics-informed neural networks with domain decomposition. *Comput. Methods Appl. Mech. Eng.* **2021**, *374*, 113547. [CrossRef]

46. Berrone, S.; Canuto, C.; Pintore, M. Solving PDEs by variational physics-informed neural networks: An a posteriori error analysis. *Ann. Univ. Ferrara* **2022**, *68*, 575–595. [CrossRef]
47. Berrone, S.; Canuto, C.; Pintore, M. Variational-Physics-Informed Neural Networks: The role of quadratures and test functions. *J. Sci. Comput.* **2022**, *92*, 100. [CrossRef]
48. Berrone, S.; Pieraccini, S.; Scialò, S. Towards effective flow simulations in realistic discrete fracture networks. *J. Comput. Phys.* **2016**, *310*, 181–201. [CrossRef]
49. Sukumar, N.; Srivastava, A. Exact imposition of boundary conditions with distance functions in physics-informed deep neural networks. *Comput. Methods Appl. Mech. Eng.* **2022**, *389*, 114333. [CrossRef]
50. Berrone, S.; Canuto, C.; Pintore, M.; Sukumar, N. Enforcing Dirichlet boundary conditions in physics-informed neural networks and variational physics-informed neural networks. *Heliyon* **2023**, *9*, e18820. [CrossRef] [PubMed]
51. Kingma, D.; Ba, J. Adam: A method for stochastic optimization. *arXiv* **2014**, arXiv:1412.6980 .
52. Wright, S.; Nocedal, J. *Numerical Optimization*; Springer: Berlin/Heidelberg, Germany, 1999; Volume 35, p. 7.
53. Baydin, A.; Pearlmutter, B.; Radul, A.; Siskind, J. Automatic differentiation in machine learning: A survey. *J. Mach. Learn. Res.* **2018**, *18*, 5595–5637.
54. Prechelt, L. Early stopping-but when? In *Neural Networks: Tricks of the Trade*; Springer: Berlin/Heidelberg, Germany, 1998; pp. 55–69.