Enforcing Dirichlet boundary conditions in physics-informed neural networks and variational physics-informed neural networks

Heliyon 9 (2023) e18820

Contents lists available at ScienceDirect

Heliyon

journal homepage: www.cell.com/heliyon

Research article

Enforcing Dirichlet boundary conditions in physics-informed neural networks and variational physics-informed neural networks

S. Berrone [a], C. Canuto [a], M. Pintore [a,*], N. Sukumar [b]

[a] Dipartimento di Scienze Matematiche, Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Torino, Italy
[b] Department of Civil and Environmental Engineering, University of California, Davis, CA 95616, USA

ARTICLE INFO

MSC:
35A15
65L10
65L20
65K10
68T05

Keywords:
Dirichlet boundary conditions
PINN
VPINN
Deep neural networks
Approximate distance function

ABSTRACT

In this paper, we present and compare four methods to enforce Dirichlet boundary conditions in Physics-Informed Neural Networks (PINNs) and Variational Physics-Informed Neural Networks (VPINNs). Such conditions are usually imposed by adding penalization terms in the loss function and properly choosing the corresponding scaling coefficients; however, in practice, this requires an expensive tuning phase. We show through several numerical tests that modifying the output of the neural network to exactly match the prescribed values leads to more efficient and accurate solvers. The best results are achieved by exactly enforcing the Dirichlet boundary conditions by means of an approximate distance function. We also show that variationally imposing the Dirichlet boundary conditions via Nitsche's method leads to suboptimal solvers.

1. Introduction

Physics-Informed Neural Networks (PINNs), proposed in [1] after the initial pioneering contributions of Lagaris et al. [2–4], are rapidly emerging computational methods to solve partial differential equations (PDEs). In its basic formulation, a PINN is a neural network that is trained to minimize the PDE residual on a given set of collocation points in order to compute a corresponding approximate solution. In particular, the fact that the PDE solution is sought in a nonlinear space via a nonlinear optimizer distinguishes PINNs from classical computational methods. This provides PINNs flexibility, since the same code can be used to solve completely different problems by adapting the neural network loss function that is used in the training phase. Moreover, due to the intrinsic nonlinearity and the adaptive architecture of the neural network, PINNs can efficiently solve inverse [5–7], parametric [8], high-dimensional [9,10] as well as nonlinear [11] problems. Another important feature characterizing PINNs is that it is possible to combine distinct types of information within the same loss function to readily modify the optimization process. This is useful, for instance, to effortlessly integrate (synthetic or experimental) external data into the training phase to obtain an approximate solution that is computed using both data and physics [12].

In order to improve the original PINN idea, several extensions have been developed. Some of these developments include the Deep Ritz method (DRM) [13], in which the energy functional of a variational problem is minimized; the conservative PINN (cPINN) [14],

* Corresponding author.
E-mail addresses: stefano.berrone@polito.it (S. Berrone), claudio.canuto@polito.it (C. Canuto), moreno.pintore@polito.it (M. Pintore), nsukumar@ucdavis.edu (N. Sukumar).

https://doi.org/10.1016/j.heliyon.2023.e18820
Received 13 July 2023; Received in revised form 29 July 2023; Accepted 29 July 2023

Available online 2 August 2023

where the approximate solution is computed by a domain-decomposition approach enforcing flux conservation at the interfaces, as well as its improvement in the extended PINN (XPINN) [15]; and the variational PINN (VPINN) [16,17], in which the loss function is defined by exploiting the variational structure of the underlying PDE.

Most of the existing PINN approaches enforce the essential (Dirichlet) boundary conditions by means of additional penalization terms that contribute to the loss function, these are each multiplied by constant weighting factors. See for instance [18–26]; note that this list is by no means exhaustive, therefore we also refer to [27–29] for more detailed overviews of the PINN literature. However, such an approach may lead to poor approximation, and therefore several techniques to improve it have been proposed. In [30] and [31], adaptive scaling parameters are proposed to balance the different terms in the loss functions. In particular, in [30] the parameters are updated during the minimization to maximize the loss function via backpropagation, whereas in [31] a fixed learning rate annealing procedure is adopted. Other alternatives are related to adaptive sampling strategies (e.g., [32–34]) or to specific techniques such as the Neural Tangent Kernel [35].

Note that although it is possible to automatically tune these scaling parameters during the training, such techniques require more involved implementations and in most cases lead to intrusive methods since the optimizer has to be modified. Instead, in this paper, we focus on three simple and non-intrusive approaches to impose Dirichlet boundary conditions and we compare their accuracy and efficiency. The proposed approaches are tested using standard PINN and interpolated VPINN which have been proven to be more stable than standard VPINNs [36].

The main contributions of this paper are as follows:

1. We present three non-standard approaches to enforce Dirichlet boundary conditions on PINNs and VPINNs, and discuss their mathematical formulation and their pros and cons. Two of them, based on the use of an approximate distance function, modify the output of the neural network to exactly impose such conditions, whereas the last one enforces them approximately by a weak formulation of the equation.
2. The performance of the distinct approaches to impose Dirichlet boundary conditions is assessed on various test cases. On average, we find that exactly imposing the boundary conditions leads to more efficient and accurate solvers. We also compare the interpolated VPINN to the standard PINN, and observe that the different approaches used to enforce the boundary conditions affect these two models in similar ways.

The structure of the remainder of this paper is as follows. In Section 2, the PINN and VPINN formulations are described: first, we describe the neural network architecture in Section 2.1 and then focus on the loss functions that characterize the two models in Section 2.2. Subsequently, in Section 3, we present the four approaches to enforce the imposition of Dirichlet boundary conditions; three of them can be used with both PINNs and VPINNs, whereas the last one is used to enforce the required boundary conditions only on VPINNs because it relies on the variational formulation. Numerical results are presented in Section 4. In Section 4.1, we first analyze for a second-order elliptic problem the convergence rate of the VPINN with respect to mesh refinement. In doing so, we demonstrate that when the neural network is properly trained, identical optimal convergence rates are realized by all approaches only if the PDE solution is simple enough. Otherwise, only enforcing the Dirichlet boundary conditions with Nitsche's method or by exactly imposing them via approximate distance functions ensure the theoretical convergence rate. In addition, we compare the behavior of the loss function and the H^1 error while increasing the number of epochs, as well as the behavior of the error when the network architecture is varied. In Section 4.2, we show that it is also possible to efficiently solve second-order parametric nonlinear elliptic problems. Furthermore, in Sections 4.3–4.5, we compare the performance of all approaches on PINNs and VPINNs by solving a linear elasticity problem and a stabilized Eikonal equation over an L-shaped domain, and a convection problem. Finally, in Section 5, we close with our main findings and present a few perspectives for future work.

2. PINNs and interpolated variational PINNs

In this section, we describe the PINN and VPINN that are used in Section 4. In particular, in Section 2.1 the neural network architecture is presented, and the construction of the loss functions is discussed in Section 2.2.

2.1. Neural network description

In this work we compare the efficiency of four approaches to enforce Dirichlet boundary conditions in PINN and VPINN. The main difference between these two numerical models is the training loss function; the architecture of the neural network is the same and is independent of the way the boundary conditions are imposed.

In our numerical experiments we only consider fully-connected feed forward neural networks with a fixed architecture. Such neural networks can be represented as nonlinear parametric functions $u^{\mathcal{NN}} : \mathbb{R}^{N_{\text{in}}} \to \mathbb{R}^{N_{\text{out}}}$ that can be evaluated via the following recursive formula:

$$\mathbf{x}_i^* = \sigma_i \left(A_i \mathbf{x}_{i-1}^* + b_i \right), \qquad i = 1, 2, \dots, L. \tag{2.1}$$

In particular, with the notation of (2.1), $\mathbf{x}_0^* \in \mathbb{R}^{N_{\text{in}}}$ is the neural network input vector, $\mathbf{x}_L^* \in \mathbb{R}^{N_{\text{out}}}$ is the neural network output vector, the neural network architecture consists of an input layer, $L-1$ hidden layers and one output layer, A_i and b_i are matrices and vectors containing the neural network weights, and $\sigma_i : \mathbb{R} \to \mathbb{R}$ is the activation function of the i-th layer and is element-wise applied to its input vector. We also remark that the i-th layer is said to contain $\dim(\mathbf{x}_i^*)$ neurons and that σ_i has to be nonlinear

for any $i = 1, 2, \dots, L-1$. Common nonlinear activation functions are the rectified linear unit ($\mathrm{ReLU}(x) := \max(0, x)$), the hyperbolic tangent and the sigmoid function. In this work, we take σ_L to be the identity function in order to avoid imposing any constraint on the neural network output.

The weights contained in A_i and b_i can be logically reorganized in a single vector $\mathbf{w}^{\mathcal{NN}}$. The goal of the training phase is to find a vector $\mathbf{w}^{\mathcal{NN}}$ that minimizes the loss function; however, since such a loss function is nonlinear with respect to $\mathbf{w}^{\mathcal{NN}}$ and the corresponding manifold is extremely complicated, we can at best find good local minima.

2.2. PINN and interpolated VPINN loss functions

For the sake of simplicity, the loss function for PINN and interpolated VPINN is stated for second-order elliptic boundary-value problems. However, the discussion can be directly generalized to different PDEs, and in Section 4, numerical results associated with other problems are also presented.

Let us consider the model problem:

$$\begin{cases} Lu := -\nabla \cdot (\mu \nabla u) + \boldsymbol{\beta} \cdot \nabla u + \sigma u = f & \text{in } \Omega, \\ u = g & \text{on } \Gamma_D, \\ \mu \frac{\partial u}{\partial n} = \psi & \text{on } \Gamma_N, \end{cases} \tag{2.2}$$

where $\Omega \subset \mathbb{R}^n$ is a bounded domain whose Lipschitz boundary $\partial\Omega$ is partitioned as $\partial\Omega = \Gamma_D \cup \Gamma_N$, with $\mathrm{meas}_{n-1}(\Gamma_D) > 0$. For the well-posedness of the boundary-value problem we require μ, $\sigma \in \mathrm{L}^\infty(\Omega)$ and $\boldsymbol{\beta} \in (\mathrm{W}^{1,\infty}(\Omega))^n$ satisfying, in the entire domain Ω, $\mu \geq \mu_0$ for some strictly positive constant μ_0 and $\sigma - \frac{1}{2}\nabla \cdot \boldsymbol{\beta} \geq 0$. Moreover, $f \in \mathrm{L}^2(\Omega)$, $\psi \in \mathrm{L}^2(\Gamma_N)$ and $g = \overline{u}_{|\Gamma_D}$ for some $\overline{u} \in \mathrm{H}^1(\Omega)$. We point out that even if these assumptions ensure the well-posedness of the problem, PINNs and VPINNs often struggle to compute low regularity solutions. We refer to [37] for a recent example of a neural network based model that overcomes this issue.

In order to train a PINN, one introduces a set of collocation points $\{x_1, \dots, x_{N_I}\}$ and evaluates the corresponding equation residuals $\{r_1^{\mathrm{PINN}}, \dots, r_{N_I}^{\mathrm{PINN}}\}$. Such residuals, for problem (2.2), are defined as:

$$r_i^{\mathrm{PINN}}(u) = -\nabla \cdot (\mu \nabla u)(x_i) + \boldsymbol{\beta} \cdot \nabla u(x_i) + \sigma u(x_i) - f(x_i) \quad \forall i = 1, 2, \dots, N_I. \tag{2.3}$$

Since we are interested in a neural network that satisfies the PDE in a discrete sense, the loss function minimized during the PINN training is:

$$R^2_{\mathrm{PINN}}(w) = \sum_{i=1}^{N_I} \left| r_i^{\mathrm{PINN}}(w) \right|^2. \tag{2.4}$$

In (2.4), when N_I is sufficiently large and $R^2_{\mathrm{PINN}}(u^{\mathcal{NN}})$ is close to zero, the function $u^{\mathcal{NN}}$ represented by the neural network output approximately satisfies the PDE and can thus be considered a good approximation of the exact solution. Other terms are often added to impose the boundary conditions or improve the training, which are discussed in Section 3.

Let us now focus on the interpolated VPINN proposed in [36]. We introduce the function spaces $U := H^1(\Omega)$ and $V := \{v \in H^1(\Omega) : v_{|\Gamma_D} = 0\}$, the bilinear form $a : U \times V \to \mathbb{R}$ and the linear form $F : V \to \mathbb{R}$,

$$a(w, v) = \int_\Omega \mu \nabla w \cdot \nabla v + \beta \nabla w v + \sigma w v, \qquad F(v) = \int_\Omega f v + \int_{\Gamma_N} \psi v.$$

The variational counterpart of problem (2.2) thus reads: Find $u \in U$ such that:

$$\begin{aligned} &a(u, v) = F(v) \quad \forall v \in V, \\ &\quad u = g \quad \text{on } \Gamma_D. \end{aligned} \tag{2.5}$$

In order to discretize problem (2.5), we use two discrete function spaces. Inspired by the Petrov-Galerkin framework, we denote the discrete trial space by $U_h \subset U$ and the discrete test space by $V_h \subset V$. The functions comprising such spaces are generated on two conforming, shape-regular and nested partitions $\mathcal{T}_H$ and $\mathcal{T}_h$ with compatible meshsizes H and h, respectively. Assuming that $\mathcal{T}_h$ is the finer mesh, one can claim that $H \lesssim h < H$ and that every element of $\mathcal{T}_h$ is strictly contained in an element of $\mathcal{T}_H$.

Denoting by $U_H := \mathrm{span}\{\varphi_i^u : i \in I_H\} \subset U$ the space of piecewise polynomial functions of order k_{int} over $\mathcal{T}_H$ and $V_h := \mathrm{span}\{\varphi_i^v : i \in I_h\} \subset V$ the space of piecewise polynomial functions of order k_{test} over $\mathcal{T}_h$ that vanish on Γ_D, we define the discrete variational problem as: Find $u \in U_H$ such that:

$$\begin{aligned} &a(u, v) = F(v) \quad \forall v \in V_h, \\ &\quad u = g_H \quad \text{on } \Gamma_D, \end{aligned} \tag{2.6}$$

where g_H is a suitable piecewise polynomial approximation of g. A representation of the spaces U_H and V_h in a one-dimensional domain is provided in Figs. 1a and 1b. Examples of pair of meshes $\mathcal{T}_H$ and $\mathcal{T}_h$ are shown in Fig. 1c.

In order to obtain computable forms a_h and F_h, we introduce elemental quadrature rules of order q and define $a_h(\cdot, \cdot)$ and $F_h(\cdot)$ as the approximations of $a(\cdot, \cdot)$ and $F(\cdot)$ computed with such quadrature rules. In [36], under suitable assumptions, an a priori error

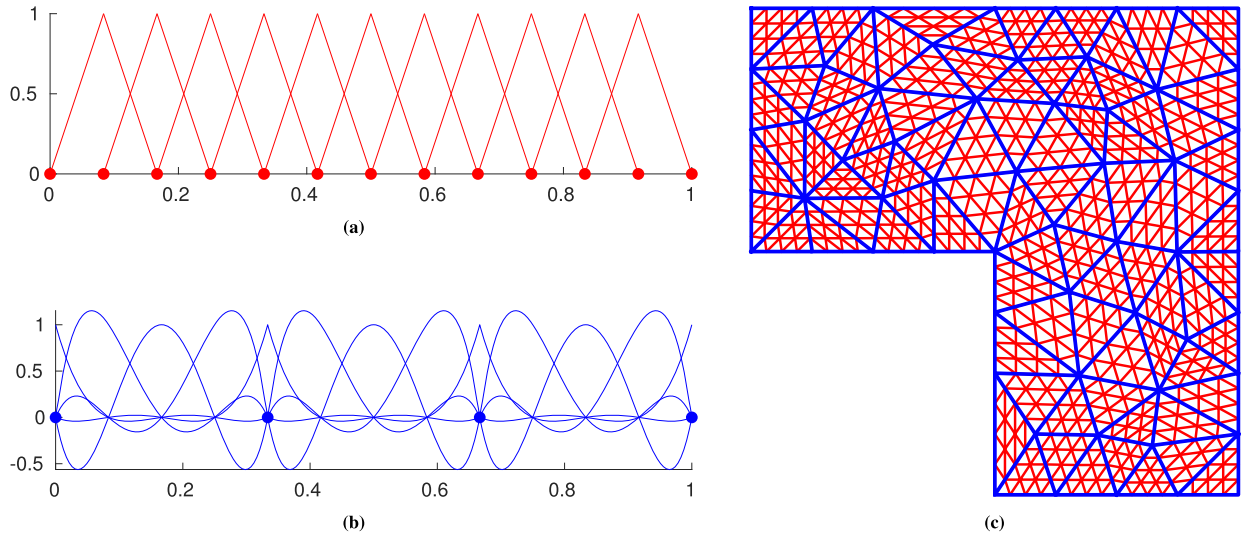


Fig. 1. Pair of meshes and corresponding basis functions of a one-dimensional discretization (left) and nested meshes $\mathcal{T}_H$ and $\mathcal{T}_h$ in a two-dimensional domain (right). (a) Basis functions of V_h. The filled circles (red) are the nodes of the corresponding mesh $\mathcal{T}_h$; (b) Basis functions of U_H. The filled circles (blue) are the vertex nodes that define the elements of the corresponding mesh $\mathcal{T}_H$; and (c) Meshes used in the numerical experiments of Sections 4.3 and 4.4. The blue mesh is $\mathcal{T}_H$, the red one is $\mathcal{T}_h$. All the figures are obtained with $q = 3$, $k_{\text{test}} = 1$, $k_{\text{int}} = 4$.

estimate with respect to mesh refinement has been proved when $q = k_{\text{int}} + k_{\text{test}} - 2$. It is then possible to define the computable variational residuals associated with the basis functions of V_h as:

$$r_{h,i}(w) = F_h(\varphi_i^v) - a_h(w, \varphi_i^v), \quad i \in I_h. \tag{2.7}$$

Consequently, in order to compute an approximate solution of problem (2.6), one seeks a function $w \in U_H$ that minimizes the quantity:

$$R_h^2(w) = \sum_{i \in I_h} r_{h,i}^2(w), \tag{2.8}$$

and satisfies the imposed boundary conditions. We refer to Section 3 for a detailed description of different approaches used to impose Dirichlet boundary conditions. It should be noted that, since in Sections 4.2–4.5 we consider problems other than (2.2), the residuals in (2.7) have to be suitably modified, while the loss function structure defined in (2.8) is maintained.

We are interested in using a neural network to find the minimizer of R_h^2. We thus denote by $\mathcal{I}_H : C^0(\overline{\Omega}) \to U_H$ an interpolation operator used to map the function $u^{\mathcal{NN}}$ associated with the neural network to its interpolating element in U_H, and train the neural network to minimize the quantity $R_h^2(\mathcal{I}_H u^{\mathcal{NN}})$. We highlight that in order to construct the function $\mathcal{I}_H u^{\mathcal{NN}}$, the neural network has to be evaluated only on $\dim(U_H)$ interpolation points $\{x_1^{\mathcal{I}}, \dots, x_{\dim(U_H)}^{\mathcal{I}}\} \subset \overline{\Omega}$. Then, assuming that $\{\varphi_i^u : i \in I_H\}$ is a Lagrange basis such that $\varphi_i^u(x_j^{\mathcal{I}}) = \delta_{ij}$ for every $i, j \in I_H$, it holds:

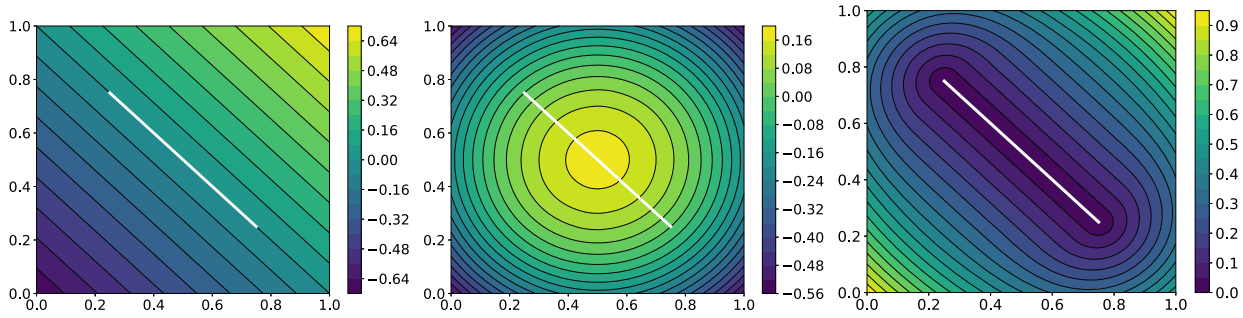
$$\mathcal{I}_H u^{\mathcal{NN}} = \sum_{i \in I_H} u^{\mathcal{NN}}(x_i^{\mathcal{I}}) \varphi_i^u. \tag{2.9}$$

We remark that the approaches proposed in Section 3 can also be used on non-interpolated VPINNs. However, we restrict our analysis to interpolated VPINNs because of their better stability properties (see Fig. 11 and the corresponding discussion).

3. Mathematical formulation

We compare four methods to impose Dirichlet boundary conditions on PINNs and VPINNs. We do not consider Neumann or Robin boundary conditions since they can be weakly enforced by the trained VPINN due to the chosen variational formulation (computations using PINNs is discussed in [38]). We also highlight that method $\mathbf{M_D}$ below can be used only with VPINNs because it relies on the variational formulation of the PDE. We analyze the following methods:

$\mathbf{M_A}$: Incorporation of an additional cost in the loss function that penalizes unsatisfied boundary conditions; this is the standard approach in PINNs and VPINNs because of its simplicity and effectiveness. In fact, it is possible to choose N_B control points $\{x_1^g, \dots, x_{N_B}^g\} \subset \Gamma_D$ and modify the loss functions defined in (2.4) or (2.8) as follows:



(a) $d(\mathbf{x})$. (b) $t(\mathbf{x})$. (c) $\phi(\mathbf{x})$.

Fig. 2. Representation of the signed distance function $d(\mathbf{x})$ to a straight line (left), the trimming function $t(\mathbf{x})$ (middle) and the approximate distance function $\phi(\mathbf{x})$ to a segment (right).

$$R^2_{\text{PINN}}(w) = \sum_{i=1}^{N_I} \left| r_i^{\text{PINN}}(w) \right|^2 + \lambda \sum_{i=1}^{N_B} \left(w(x_i^g) - g(x_i^g) \right)^2, \tag{3.1}$$

or

$$R^2_h(w) = \sum_{i \in I_h} r^2_{h,i}(w) + \lambda \sum_{i=1}^{N_B} \left(w(x_i^g) - g(x_i^g) \right)^2, \tag{3.2}$$

where $\lambda > 0$ is a model hyperparameter. Note that on considering the interpolated VPINN and exploiting the solution structure in (2.9), it is possible to ensure the uniqueness of the numerical solution by choosing the control points $\{x_1^g, \dots, x_{N_B}^g\}$ as the N_B interpolation points belonging to Γ_D.

We also highlight that such a method can be easily adapted to impose other types of boundary conditions just by adding suitable terms to (3.1) and (3.2). On the other hand, despite its simplicity, the main drawback of this approach is that it leads to a more complex multi-objective optimization problem.

$\mathbf{M_B}$: Exactly imposing the Dirichlet boundary conditions as described in [38] and [36]. In this method we add a non-trainable layer B at the end of the neural network to modify its output w according to the rule:

$$Bw = \overline{g} + \phi w, \tag{3.3}$$

where $\overline{g} \in C^0(\overline{\Omega})$ is an extension of the function g inside the domain Ω (i.e., $\overline{g}_{|\Gamma_D} = g$) and $\phi \in C^0(\overline{\Omega})$ is an approximate distance function (ADF) to the boundary Γ_D, i.e., $\phi(\boldsymbol{x}) = 0$ if and only if $\boldsymbol{x} \in \Gamma_D$, and it is positive elsewhere. During the training phase one minimizes the quantity $R^2_{\text{PINN}}(Bw)$ or $R^2_h(Bw)$.

For the sake of simplicity, we only consider ADFs for two-dimensional unions of segments, even though the approach generalizes to more complex geometries. Following the derivation of $\overline{g}$ and ϕ in [38], we start by defining d as the signed distance function from $\boldsymbol{x} := (x, y)$ to the line defined by the segment AB of length L with vertices $A = (x_A, y_A)$ and $B = (x_B, y_B)$:

$$d(\boldsymbol{x}) = \frac{(x - x_A)(y_B - y_A) - (y - y_A)(x_B - x_A)}{L}.$$

Then, we denote $(x_c, y_c) := \big((x_A + x_B)/2, (y_A + y_B)/2\big)$ to be the center of AB and define t as the following trimming function:

$$t(\boldsymbol{x}) = \frac{1}{L}\left[\left(\frac{L}{2} \right)^2 - \left\| (x, y) - (x_c, y_c) \right\|^2 \right].$$

Note that $t \geq 0$ defines a circle of center (x_c, y_c). Finally, the ADF to AB is defined as

$$\phi(\boldsymbol{x}) = \sqrt{d^2 + \left(\frac{\sqrt{t^2 + d^4} - t}{2} \right)^2}.$$

A graphical representation of $d(\boldsymbol{x})$, $t(\boldsymbol{x})$ and $\phi(\boldsymbol{x})$ for an inclined line segment is shown in Figs. 2a, 2b and 2c, respectively.

Assuming that Γ_D can be expressed as the union of n_s segments $\{s_1, \dots, s_{n_s}\}$, then the ADF to Γ_D, normalized up to order $m \geq 1$, is defined as:

$$\phi = \frac{1}{\sqrt[m]{\frac{1}{\phi_1^m} + \frac{1}{\phi_2^m} + \cdots + \frac{1}{\phi_{n_s}^m}}}, \tag{3.4}$$

where ϕ_i is the ADF to the segment s_i (see [39]). We remark that an ADF normalized up to order $m \geq 1$ is an ADF such that, for every regular point of Γ_D, the following holds:

$$\phi = 0, \qquad \frac{\partial \phi}{\partial n} = 1, \qquad \frac{\partial^k \phi}{\partial n^k} = 0 \quad (k = 2, 3, \dots, m).$$

Such a normalization is useful to impose constraints associated with the solution derivatives and to obtain ADFs with about the same order of magnitude in every region of the domain Ω. However, one of the main limitations of this approach with collocation-PINN is that $\Delta\phi$ tends to infinity near the vertices of Γ_D (see Appendix A for an example). This phenomenon produces oscillations in the numerical solutions, hence collocation points that are close to such vertices should not be selected. On the other hand, when only first derivatives are present in the weak formulation of second-order problems (as in the present study), then one can choose quadrature points that are very close to the vertices of Γ_D.

When a function $\overline{g}$ is not known, it is possible to construct it using transfinite interpolation. Let g_i be a function such that $g_{i|_{s_i}} = g_{|_{s_i}}$, then $\overline{g}$ can be defined as:

$$\overline{g} = \sum_{i=1}^{n_s} w_i g_i,$$

where w_i is defined as:

$$w_i = \frac{\prod_{j=1; j\neq i}^{n_s} \phi_j}{\sum_{k=1}^{n_s} \prod_{j=1; j\neq k}^{n_s} \phi_j}.$$

Note that since s_i is a segment, a function g_i can be readily defined at any arbitrary point (x, y) just by evaluating g at the orthogonal projection of (x, y) onto s_i.

$\mathbf{M_C}$: Exactly imposing the Dirichlet boundary conditions as in $\mathbf{M_B}$ but without normalizing the ADF. Therefore, we consider a different function ϕ in (3.3), namely

$$\phi = \prod_{i=1}^{n_s} \phi_i.$$

This ensures that ϕ and all its derivatives exist and are bounded in $\overline{\Omega}$, although ϕ may be very small in regions close to many segments s_i.

$\mathbf{M_D}$: Using Nitsche's method [40]. The goal of this method is to variationally impose the Dirichlet boundary conditions. In doing so, the network architecture is not modified with additional layers (as in $\mathbf{M_B}$ and $\mathbf{M_C}$) and a single objective function suffices for network training.

To do so, one enlarges the space V_h to contain all piecewise polynomials of order k_{test} defined on $\mathcal{T}_h$ and modifies the residuals defined in (2.7) in the following way:

$$\begin{aligned} r_{h,i}(w) = F_h(\varphi_i^v) - a_h(w, \varphi_i^v) + \int_{\Gamma_D} (w - g) \frac{\partial \phi_i^v}{\partial n} \\ + \gamma \int_{\Gamma_D} h^{-1} (g - w) \phi_i^v, \quad i \in I_h, \end{aligned} \tag{3.5}$$

where γ is a positive constant satisfying $\gamma \geq \gamma_0$ for a suitable $\gamma_0 > 0$ and I_h is now an enlarged index set corresponding to the enlarged basis $\{\phi_i^v : i \in I_H\}$. Thanks to the scaling term h^{-1} that magnifies the quantity $\int_{\Gamma_D}(g - w)\phi_i^v$ when fine meshes are used, the choice of γ is not as important as the one of λ in method $\mathbf{M_A}$. This property is confirmed by numerical results shown in Figs. 8 and 10. Since there is no ambiguity, we maintain the same symbols V_h, I_h and $\{\phi_i^v\}$ introduced in Section 2.2; they always represent the enlarged sets when method $\mathbf{M}_D$ is considered. Note that when w satisfies the Dirichlet boundary conditions, the terms added in (3.5) vanish.

We point out that method $\mathbf{M_A}$ is often referred to as *soft boundary condition imposition*, whereas $\mathbf{M_B}$ and $\mathbf{M_C}$ are known as *hard boundary condition impositions*. Hence, we can treat $\mathbf{M_D}$ as *weak boundary condition imposition*.

4. Numerical results

In this section, the methods $\mathbf{M_A}$, $\mathbf{M_B}$, $\mathbf{M_C}$ and $\mathbf{M_D}$ discussed in Section 3 are analyzed and compared. In each numerical experiment the neural network is a fully-connected feed-forward neural network as described in Section 2.1. The corresponding architecture is composed of 4 hidden layers with 50 neurons in each layer and with the hyperbolic tangent as the activation function, while the output layer is a linear layer with one or two neurons.

In order to properly minimize the loss function we use the first-order ADAM optimizer [41] with an exponentially decaying learning rate, and after a prescribed number of epochs, the second-order BFGS method [42] is used until a maximum number of iterations is reached, or it is not possible to further improve the objective function (i.e. when two consecutive iterates are identical, up to machine precision). When the interpolated VPINN is used, the training set consists of all the interpolation nodes $\{x_1^{\mathcal{I}}, \dots, x_{\dim(U_H)}^{\mathcal{I}}\}$ and no regularization is applied since the interpolation operator already filters the neural network high frequencies out. Instead, when the PINN is used, the training set contains a set of $\dim(U_H)$ control points inside the domain Ω, and when $\mathbf{M_A}$ is employed, a

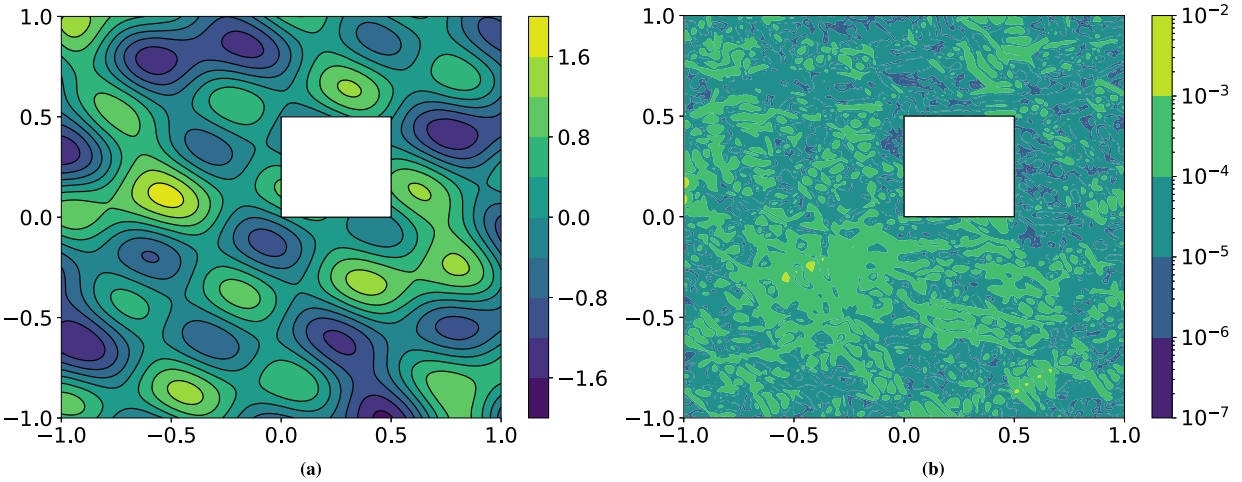


Fig. 3. Exact solution u (left) and a plot of the absolute error with VPINN and method $\mathbf{M_B}$ in which the Dirichlet boundary conditions are imposed on every edge of $\partial\Omega$ (right).

set of approximately $\sqrt{\dim(U_H)}$ control points on the boundary $\partial\Omega$. Moreover, in order to stabilize the PINN, the L^2 regularization term

$$\mathcal{L}_{\text{reg}}(\mathbf{w}^{\mathcal{NN}}) = \lambda_{\text{reg}} \|\mathbf{w}^{\mathcal{NN}}\|_2^2 \tag{4.1}$$

is added to the loss function, where $\mathbf{w}^{\mathcal{NN}}$ is the vector containing all the neural network weights defined in Section 2.1 and $\lambda_{\text{reg}} = 10^{-6}$. The value of this parameter has been chosen through several numerical experiments to minimize the H^1 norm of the error.

The computer code to perform the numerical experiments is written in Python, while the neural networks and the optimizers are implemented using the open-source Python package Tensorflow [43]. The loss function gradient with respect to the neural network weights and the PINN output gradient with respect to the spatial coordinates are always computed with automatic differentiation that is available in Tensorflow [44]. On the other hand, the VPINN output gradient with respect to its input is computed by means of suitable projection matrices as described in [36].

4.1. Rate of convergence for second-order elliptic problems

We focus on the VPINN model and show that the a priori error estimate proved in [36] for second-order elliptic problems holds even on varying the way in which the boundary conditions are imposed. On letting $\boldsymbol{x} := (x, y)$, we consider problem (2.2) in the domain $\Omega = (-1, 1)^2 \setminus (0, 0.5)^2$ with the physical parameters

$$\mu(\boldsymbol{x}) = 2 + \sin(x + 2y), \quad \beta(\boldsymbol{x}) = \left\{ \sqrt{x - y^2 + 5}, \sqrt{y - x^2 + 5} \right\}, \quad \sigma(\boldsymbol{x}) = e^{\frac{x}{2} - \frac{y}{3}} + 2.$$

We consider two test cases. In the first one the Dirichlet boundary conditions and forcing term are chosen so that the exact solution is

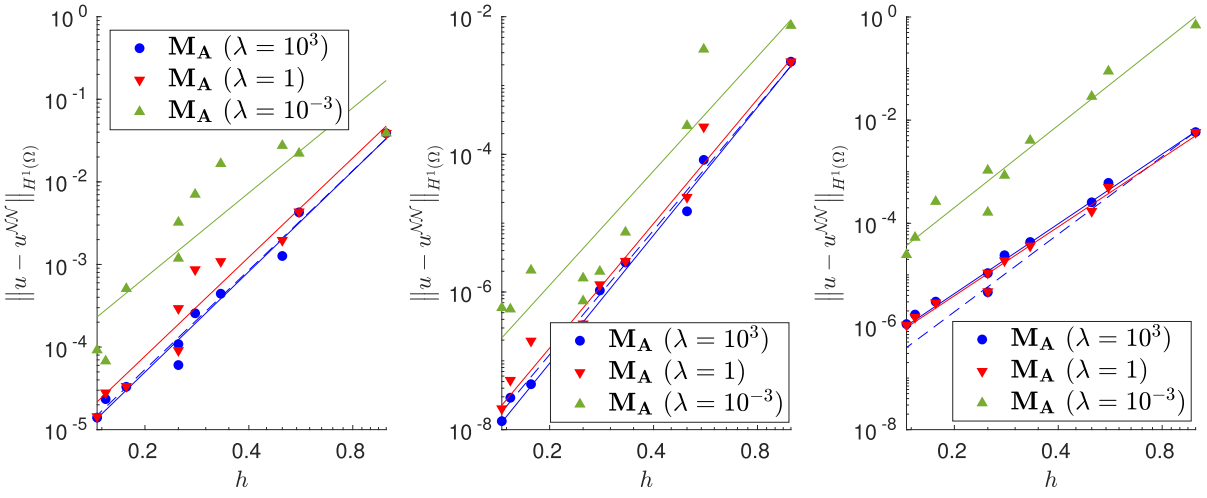
$$u(\boldsymbol{x}) = \cos(5(x + y/2)) + (x + y/2)^2, \tag{4.2}$$

whereas in the second one they are chosen such that the exact solution is more oscillatory. Its expression is:

$$u(\boldsymbol{x}) = \sin[3x(x - y)]\cos(4y + x) + \sin[5(x + 2y)]\cos[3(y - 2x)]. \tag{4.3}$$

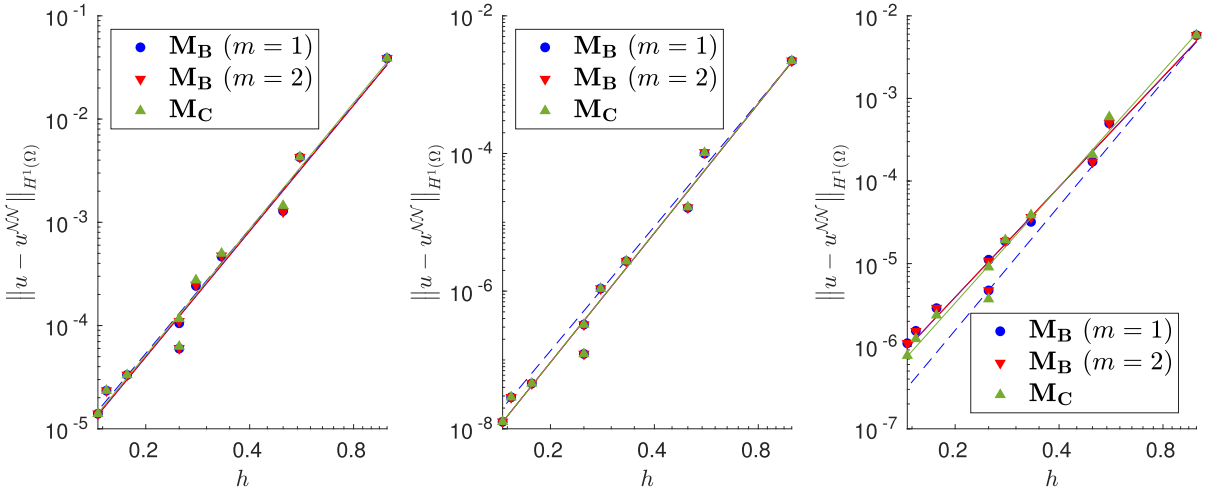
Such a solution is shown in Fig. 3a, whereas an example of numerical error corresponding to the VPINN in which Dirichlet boundary conditions are imposed using method $\mathbf{M_B}$ is shown in Fig. 3b; it exhibits a rather uniform distribution of the error, which is not localized near boundaries. We remark that in these numerical tests and in the subsequent ones, the function $\overline{g}$ used in $\mathbf{M_B}$ and $\mathbf{M_C}$ is computed via transfinite interpolation.

We vary both the order of the quadrature rule and the degree of the test functions, and train the same model with different meshes and impose the Dirichlet boundary conditions with the proposed approaches. In Figs. 4a–4c, 5a–5c and 6a–6c, in which the exact solution is the one in (4.2), we observe close agreement with the results shown in [36]. In fact, when the loss is properly minimized, all the approaches perform comparably and the corresponding empirical convergence rates are always close to the theoretical rate of $k_{\text{int}} = q + 2 - k_{\text{test}}$. We point out that in [36] we prove that, when the solution is regular enough and a method similar to $\mathbf{M_C}$ is used to enforce the boundary conditions, the convergence rate is $k_{\text{int}} = q + 2 - k_{\text{test}}$. Here, instead we show that the same behavior is



(a) $q = 3$, $k_{\text{test}} = 1$, $k_{\text{int}} = 4$. **(b)** $q = 5$, $k_{\text{test}} = 1$, $k_{\text{int}} = 6$. **(c)** $q = 5$, $k_{\text{test}} = 2$, $k_{\text{int}} = 5$.

Fig. 4. Error decay obtained with $\mathbf{M_A}$ and different values of λ. Forcing term and Dirichlet boundary conditions are set such that the exact solution is (4.2). The theoretical convergence rate is k_{int}. (a) Convergence rates: 4.04 ($\lambda = 10^3$), 3.99 ($\lambda = 1$), 3.42 ($\lambda = 10^{-3}$). (b) Convergence rates: 6.18 ($\lambda = 10^3$), 6.00 ($\lambda = 1$), 5.52 ($\lambda = 10^{-3}$). (c) Convergence rates: 4.50 ($\lambda = 10^3$), 4.44 ($\lambda = 1$), 5.29 ($\lambda = 10^{-3}$).



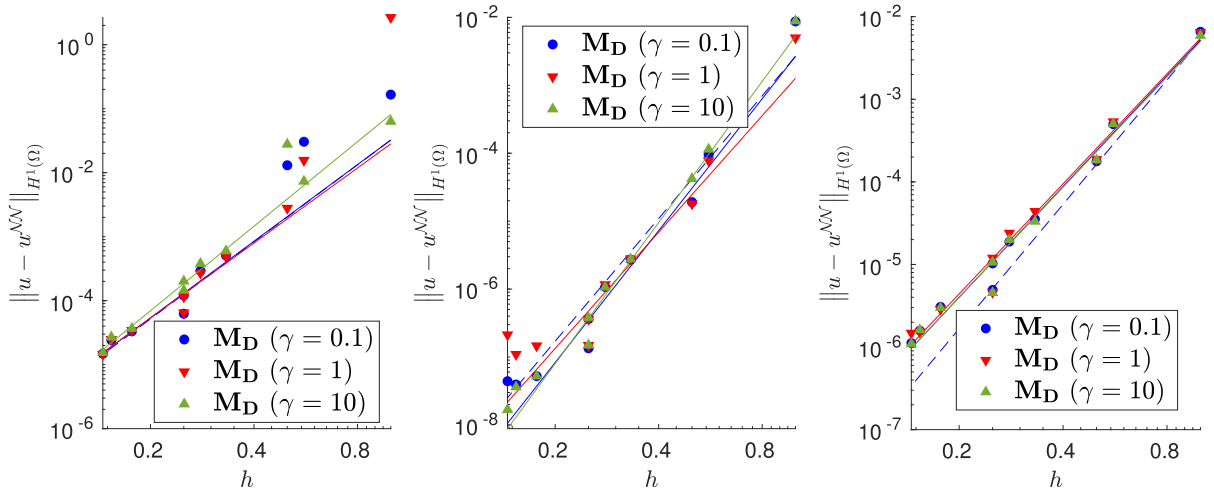
(a) $q = 3$, $k_{\text{test}} = 1$, $k_{\text{int}} = 4$. **(b)** $q = 5$, $k_{\text{test}} = 1$, $k_{\text{int}} = 6$. **(c)** $q = 5$, $k_{\text{test}} = 2$, $k_{\text{int}} = 5$.

Fig. 5. Error decay obtained with $\mathbf{M_B}$, with different values of m, and $\mathbf{M_C}$. Forcing term and Dirichlet boundary conditions are set such that the exact solution is (4.2). The theoretical convergence rate is k_{int}. (a) Convergence rates: 4.05 ($\mathbf{M_B}, m = 1$), 4.05 ($\mathbf{M_B}, m = 2$), 4.06 ($\mathbf{M_C}$). (b) Convergence rates: 6.24 ($\mathbf{M_B}, m = 1$), 6.25 ($\mathbf{M_B}, m = 2$), 6.25 ($\mathbf{M_C}$). (c) Convergence rates: 4.43 ($\mathbf{M_B}, m = 1$), 4.43 ($\mathbf{M_B}, m = 2$), 4.67 ($\mathbf{M_C}$).

observed even if the boundary conditions are enforced in different ways. Note in particular that the choice $m = 1$ or $m = 2$ in $\mathbf{M_B}$, and the choice $\gamma = 0.1$, $\gamma = 1$ or $\gamma = 10$ in $\mathbf{M_D}$ yields nearly identical results (see Figs. 5 and 6).

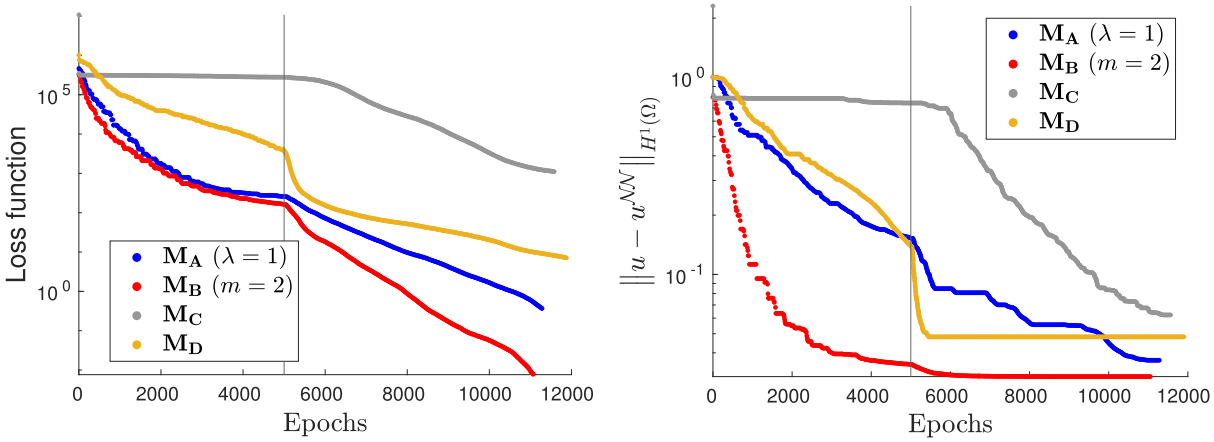
We highlight that the different methods, while delivering similar empirical convergence rates with respect to mesh refinement, exhibit very different performance during training. To observe this phenomenon, let us train multiple identical neural networks on the same mesh but impose the Dirichlet boundary conditions in different ways. Here we only consider quadrature rules of order $q = 3$ and piecewise linear test functions. The values of the loss function and of the H^1 error prediction during training are presented in Figs. 7a and 7b, respectively. A vertical line separates the epochs where the ADAM optimizer is used from the ones where the BFGS optimizer is used.

It can be noted that the most efficient method is $\mathbf{M_B}$, as it converges faster and to a more accurate solution, while method $\mathbf{M_D}$ is characterized by very fast convergence only when the BFGS optimizer is adopted. Such an optimizer is also crucial to train the VPINN with $\mathbf{M_C}$; in fact the corresponding error does not decrease when the ADAM optimizer is used. Instead, the convergence obtained using method $\mathbf{M_A}$ seems independent of the choice of the optimizer. It is important to remark that all the loss functions are decreasing even when the error is constant. This implies that there exist other sources of error that dominate and that a very small



(a) $q = 3$, $k_{\text{test}} = 1$, $k_{\text{int}} = 4$. **(b)** $q = 5$, $k_{\text{test}} = 1$, $k_{\text{int}} = 6$. **(c)** $q = 5$, $k_{\text{test}} = 2$, $k_{\text{int}} = 5$.

Fig. 6. Error decay obtained with $\mathbf{M_D}$ and different values of γ. Forcing term and Dirichlet boundary conditions are set such that the exact solution is (4.2). The theoretical convergence rate is k_{int}. (a) Convergence rates: 3.98 ($\gamma = 0.1$), 3.89 ($\gamma = 1$), 4.39 ($\gamma = 10$). (b) Convergence rates: 6.45 ($\gamma = 0.1$), 5.69 ($\gamma = 1$), 6.90 ($\gamma = 10$). (c) Convergence rates: 4.46 ($\gamma = 0.1$), 4.43 ($\gamma = 1$), 4.43 ($\gamma = 10$).



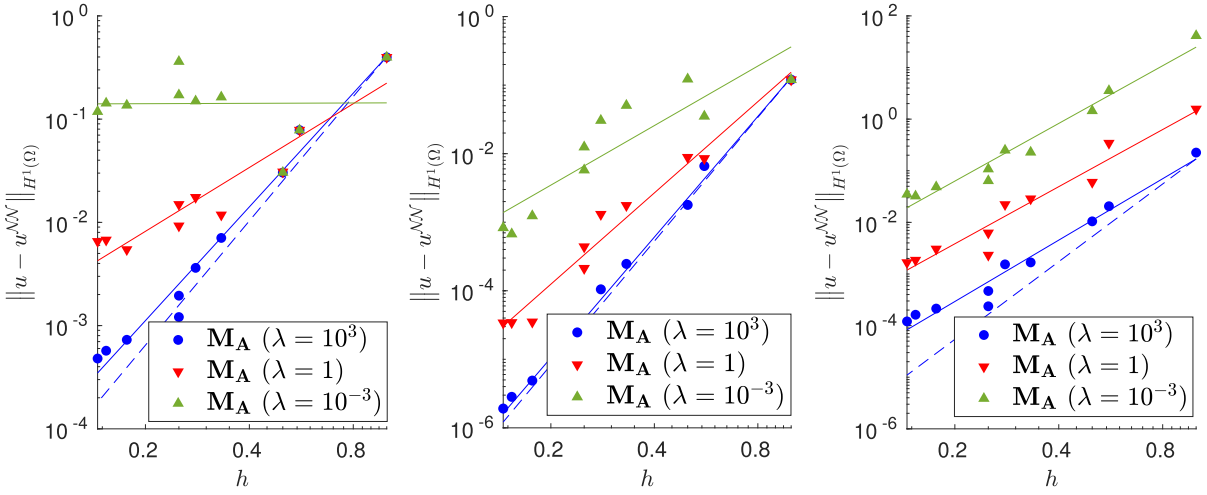
(a) Loss function behaviour during training. **(b)** H^1 error during training.

Fig. 7. Training loss (left) and H^1 error prediction (right) for the VPINN. The first 5000 epochs are performed with a standard ADAM optimizer, the remaining ones with the BFGS optimizer. The exact solution is given in (4.3).

loss function does not ensure a very accurate solution; this phenomenon is also observed in Fig. 3 of [45] and is discussed in greater detail therein.

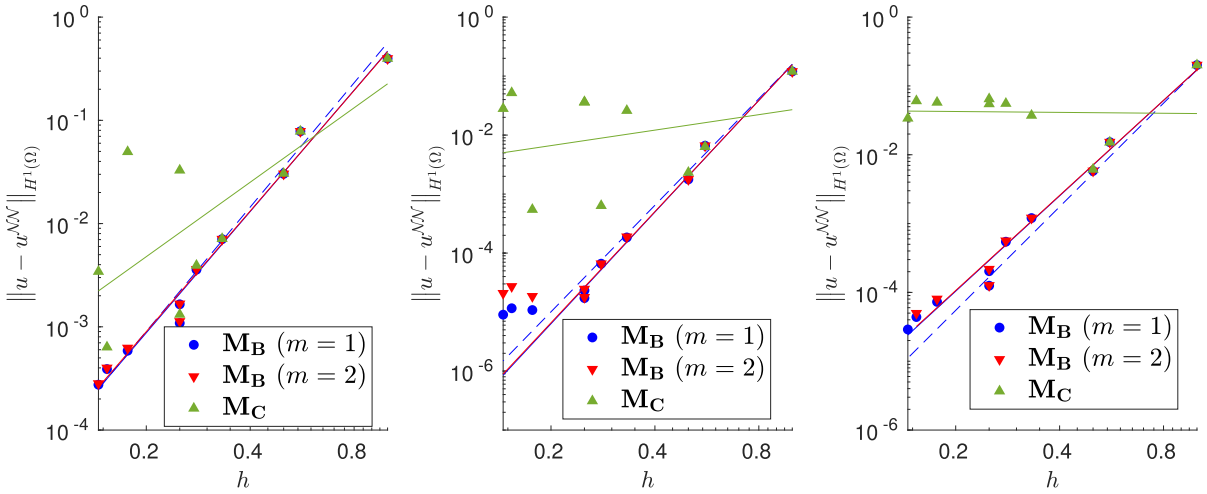
Note that, if we change the forcing term and Dirichlet boundary conditions to consider the more oscillatory exact solution in (4.3), some approaches do not ensure the theoretical convergence rate (see Figs. 8a–8c, 9a–9c and 10a–10c). In fact, in Fig. 8 it is evident that, in this case, large values of λ are required to properly enforce the Dirichlet boundary conditions. In Fig. 9, instead, we can observe that the VPINN trained with method $\mathbf{M_C}$ is often inaccurate and the corresponding error decay is very noisy. The performance of methods $\mathbf{M_B}$ and $\mathbf{M_D}$ seems independent of the complexity of the forcing term and boundary conditions.

In order to show that interpolation acts as a stabilization, we fix a mesh and vary the number of layers and neurons of the neural network. The boundary conditions are imposed using method $\mathbf{M_B}$ with $m = 2$ and the exact solution is the one in (4.2); the results are shown in Fig. 11. The number L of layers varies in $\{2, 3, 4, 5\}$, whereas the number of neurons in each hidden layer belongs to the set $\{1, 5, 10, 30, 50, 70, 100, 200, 500, 1000\}$. In Fig. 11a we show the performance of a non-interpolated VPINN trained with the L^2 regularization in (4.1), where $\lambda_{\text{reg}} = 10^{-6}$. It can be noted that the error is high when the neural network is small because of its poor approximation capability, and that it decreases with intermediate values of the two hyperparameters. However, when the neural networks contain more than 100 neurons in each layer the error increases because of uncontrolled spurious zero-energy modes and the fact that we are looking for good local minima in a very high-dimensional space. On the other hand, when the VPINN is interpolated and the neural network is sufficiently rich, the error is constant and independent of the network dimension (see Fig. 11b). In addition, note that the average accuracy of an interpolated VPINN is better than its non-interpolated counterpart.



(a) $q = 3$, $k_{\text{test}} = 1$, $k_{\text{int}} = 4$. **(b)** $q = 5$, $k_{\text{test}} = 1$, $k_{\text{int}} = 6$. **(c)** $q = 5$, $k_{\text{test}} = 2$, $k_{\text{int}} = 5$.

Fig. 8. Error decay obtained with $\mathbf{M_A}$ and different values of λ. Forcing term and Dirichlet boundary conditions are set such that the exact solution is (4.3). The theoretical convergence rate is k_{int}. (a) Convergence rates: 3.66 ($\lambda = 10^3$), 2.05 ($\lambda = 1$), 0.01 ($\lambda = 10^{-3}$). (b) Convergence rates: 5.85 ($\lambda = 10^3$), 4.42 ($\lambda = 1$), 2.89 ($\lambda = 10^{-3}$). (c) Convergence rates: 3.95 ($\lambda = 10^3$), 3.68 ($\lambda = 1$), 3.71 ($\lambda = 10^{-3}$).



(a) $q = 3$, $k_{\text{test}} = 1$, $k_{\text{int}} = 4$. **(b)** $q = 5$, $k_{\text{test}} = 1$, $k_{\text{int}} = 6$. **(c)** $q = 5$, $k_{\text{test}} = 2$, $k_{\text{int}} = 5$.

Fig. 9. Error decay obtained with $\mathbf{M_B}$, with different values of m, and $\mathbf{M_C}$. Forcing term and Dirichlet boundary conditions are set such that the exact solution is (4.3). The theoretical convergence rate is k_{int}. (a) Convergence rates: 3.90 ($\mathbf{M_B}, m = 1$), 3.88 ($\mathbf{M_B}, m = 2$), 2.36 ($\mathbf{M_C}$). (b) Convergence rates: 5.85 ($\mathbf{M_B}, m = 1$), 4.42 ($\mathbf{M_B}, m = 2$), 2.89 ($\mathbf{M_C}$). (c) Convergence rates: 4.60 ($\mathbf{M_B}, m = 1$), 4.59 ($\mathbf{M_B}, m = 2$), -0.43 ($\mathbf{M_C}$).

4.2. Application to nonlinear parametric problem

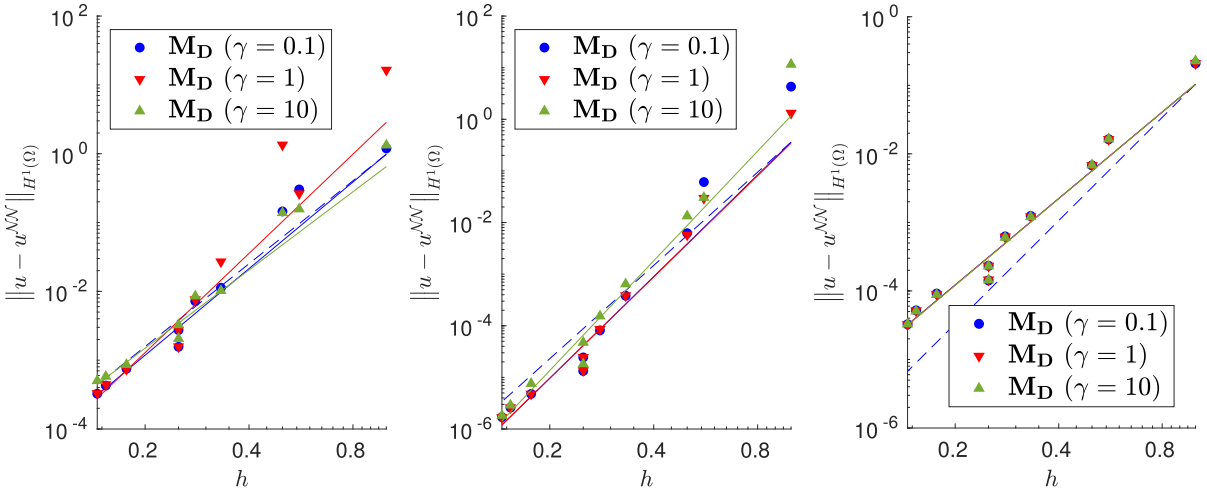
Let us now extend our analysis to nonlinear and parametric PDEs. Since in the previous section we observed that method $\mathbf{M_B}$ performs the best, in this example we do not consider $\mathbf{M_C}$ and $\mathbf{M_D}$. We focus on the following problem:

$$\begin{cases} N(u;p) := -\nabla \cdot (\mu \nabla u) + \boldsymbol{\beta} \cdot \nabla u + \sigma \sin(pu)u = f \quad \text{in } \Omega = (0,1)^2, \\ u = g \quad \text{on } \Gamma_D. \end{cases} \tag{4.4}$$

It has been observed in [36] that considering constant or variable coefficients does not influence VPINN convergence. Hence, we choose $\mu = 1$, $\boldsymbol{\beta} = [2,3]$, $\sigma = 4$ and assume that the exact solution is

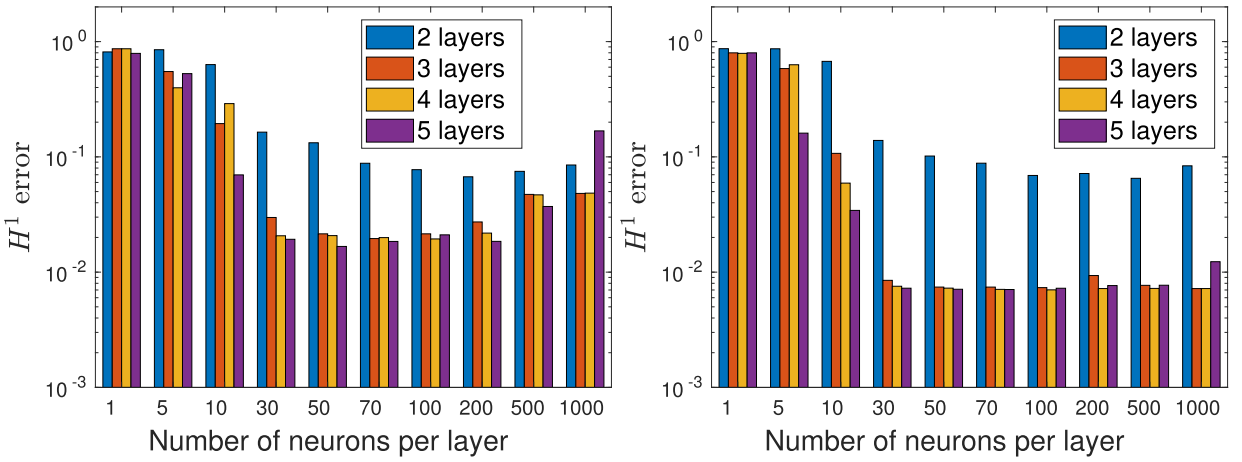
$$u(\boldsymbol{x};p) = \sin(p\pi x)\sin\left(\frac{1}{p}\pi y\right), \tag{4.5}$$

where $p \in \mathcal{I}_p = [0.5, 2]$ is a scalar parameter.



(a) $q = 3$, $k_{\text{test}} = 1$, $k_{\text{int}} = 4$. **(b)** $q = 5$, $k_{\text{test}} = 1$, $k_{\text{int}} = 6$. **(c)** $q = 5$, $k_{\text{test}} = 2$, $k_{\text{int}} = 5$.

Fig. 10. Error decay obtained with $\mathbf{M_D}$ and different values of γ. Forcing term and Dirichlet boundary conditions are set such that the exact solution is (4.3). The theoretical convergence rate is k_{int}. (a) Convergence rates: 4.18 ($\gamma = 0.1$), 4.79 ($\gamma = 1$), 3.78 ($\gamma = 10$). (b) Convergence rates: 6.54 ($\gamma = 0.1$), 6.51 ($\gamma = 1$), 7.06 ($\gamma = 10$). (c) Convergence rates: 4.19 ($\gamma = 0.1$), 4.19 ($\gamma = 1$), 4.20 ($\gamma = 10$).



(a) VPINN error. **(b)** Interpolated VPINN error.

Fig. 11. H^1 errors using method $\mathbf{M_B}$ for standard (left) and interpolated VPINNs (right) as a function of the hyperparameters.

In order to train the VPINN to solve problem (4.4), we minimize

$$R_h^2(w) = \sum_{p \in \mathcal{I}_p^{\#}} \left[\sum_{i \in I_h} r_{h,i;p}^2(w) + \lambda \sum_{i=1}^{N_B} \left(w(x_i^g) - g(x_i^g; p) \right)^2 \right]$$

when $\mathbf{M_A}$ is used, or

$$R_h^2(Bw) = \sum_{p \in \mathcal{I}_p^{\#}} \sum_{i \in I_h} r_{h,i;p}^2(Bw)$$

when $\mathbf{M_B}$ is used instead. Here $\mathcal{I}_p^{\#} = \{p_1, \dots, p_{N_p^{\text{train}}}\} \subset \mathcal{I}_p$ is a finite set of parameter values and $r_{h,i;p}$ is the residual obtained using the i-th test function and the parameter p. In the numerical computations, we use $N_p^{\text{train}} = 13$ and the VPINN is trained with $q = 3$ and $k_{\text{test}} = 1$.

In Fig. 12, we report the behavior of the loss function and the average H^1 error:

$$\frac{1}{N_p^{\text{test}}} \sum_{i=1}^{N_p^{\text{test}}} \|u(\cdot; p_i) - u^{\mathcal{NN}}(\cdot; p_i)\|_{H^1(\Omega)},$$

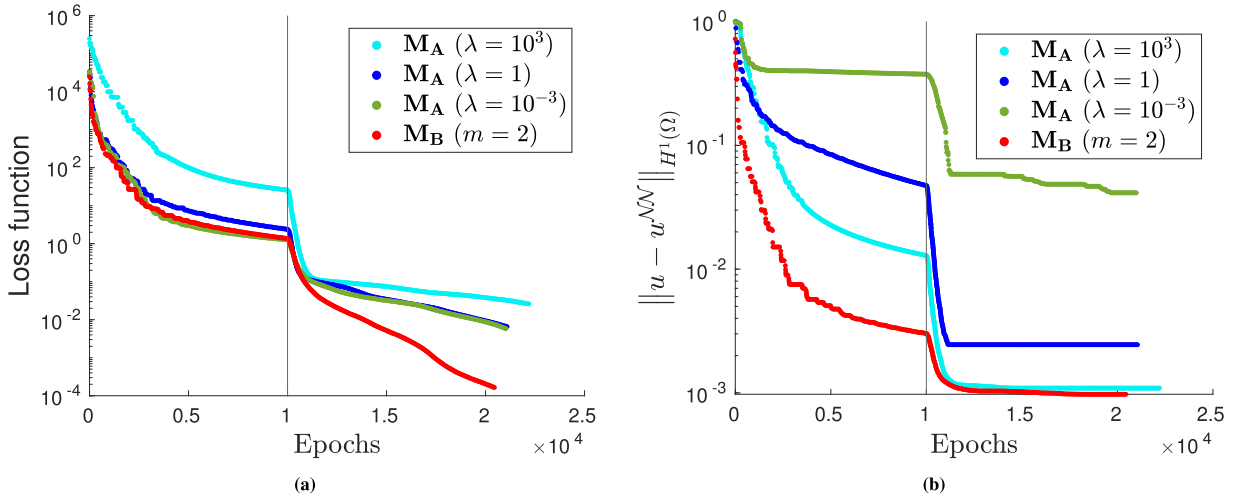


Fig. 12. (a) Training loss and (b) H^1 error prediction for the VPINN. The first 10000 epochs are performed with a standard ADAM optimizer, the remaining ones with the BFGS optimizer. The exact solution is given in (4.5).

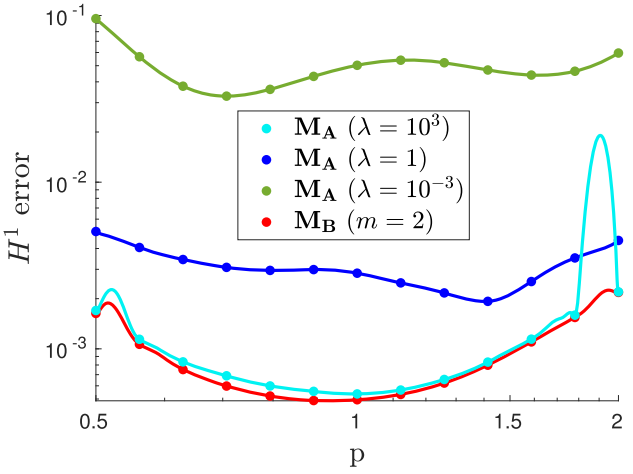


Fig. 13. H^1 error for different parameter values in problem (4.4).

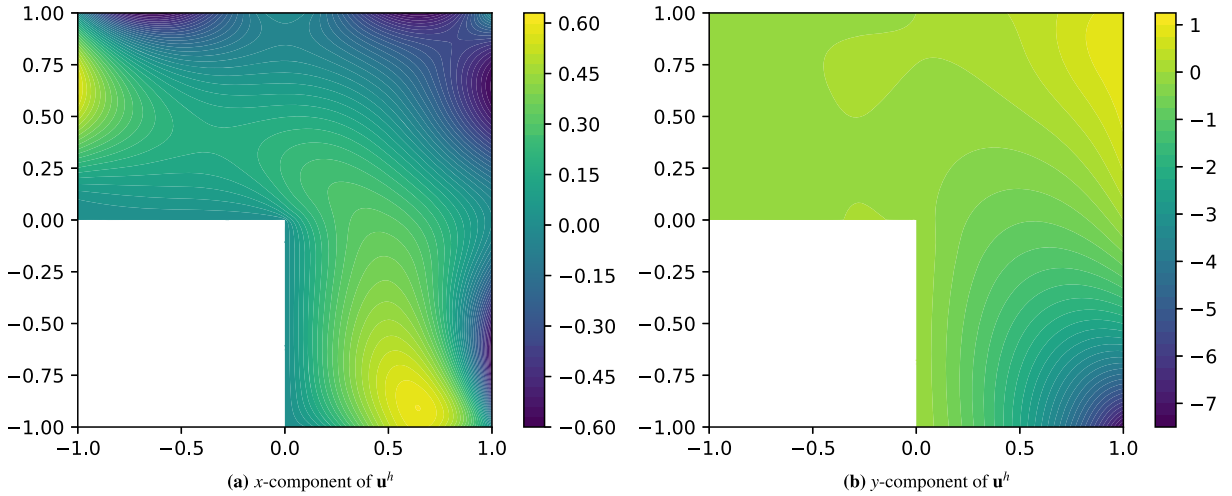
where $N_p^{\text{test}} = 100$. It is noted that the loss functions behave qualitatively similarly (see Fig. 12a). On the other hand, higher values of λ lead to lower errors when $\mathbf{M_A}$ is adopted, but the most stable and accurate approach remains $\mathbf{M_B}$.

Moreover, when the VPINN is trained, it can be evaluated at arbitrary locations in the parameter domain $\mathcal{I}_p$, yielding the error plot shown in Fig. 13. Therein, for each dot and for each point belonging to the solid lines, given the parameter value $\hat{p}$ represented on the horizontal axis, its value on the vertical axis represents the H^1 error between the VPINN solution and the exact solution $u(\cdot;\hat{p})$. Note that dots are associated with parameter values that are chosen in $\mathcal{I}_p^{\#}$ during the training, whereas solid lines are the predictions to assess the accuracy of the models for intermediate values of p. Such lines thus show the H^1 error for values of the parameter not used during the training.

4.3. Deformation of an elastic body

We consider the deformation of a linear elastic solid in the region $\Omega_L = (-1,1)^2 \setminus [-1,0]^2$, which is subjected to a body force field $\mathbf{f}$ and Dirichlet boundary conditions imposed on $\Gamma_D = \partial\Omega$. The elastostatic boundary-value problem is:

$$\begin{cases} -\nabla \cdot \boldsymbol{\sigma} = \mathbf{f} & \text{in } \Omega_L, \quad \text{(a)} \\ \boldsymbol{\varepsilon} = \frac{1}{2}\left(\nabla \mathbf{u} + \left(\nabla \mathbf{u}^T\right)\right) & \text{in } \Omega_L, \quad \text{(b)} \\ \boldsymbol{\sigma} = 2\mu\boldsymbol{\varepsilon} + \lambda\, \text{trace}(\boldsymbol{\varepsilon})\mathbf{I} & \text{in } \Omega_L, \quad \text{(c)} \\ \mathbf{u} = \mathbf{g} & \text{on } \Gamma_D. \quad \text{(d)} \end{cases} \tag{4.6}$$



(a) x-component of $\mathbf{u}^h$ **(b)** y-component of $\mathbf{u}^h$

Fig. 14. Reference finite element displacement field solution for problem (4.6). The x-component (left) and y-component (right) of $\mathbf{u}^h$ are shown.

In (4.6), $\boldsymbol{\sigma} := \boldsymbol{\sigma}(\mathbf{u})$ is the Cauchy stress tensor, $\boldsymbol{\varepsilon} := \boldsymbol{\varepsilon}(\mathbf{u})$ is the small strain tensor and (4.6c) is the isotropic linear elastic constitutive relation. The Lamé parameters λ and μ are related to the Young modulus E and the Poisson ratio ν via

$$\mu = \frac{E}{2(1+\nu)}, \quad \lambda = \frac{E\nu}{(1+\nu)(1-2\nu)}.$$

For the numerical experiments, we choose $E = 117$, $\nu = 1/3$ and the following body force field and boundary data:

$$\mathbf{f} = (\mu+\lambda)\left[xe^y,\ y\sqrt{x+2}\right], \qquad \mathbf{g} = \left[\sin(\pi(x+y)),\ e^{x-y}\right] xy.$$

The variational formulation of problem (4.6a) reads as: Find $\mathbf{u} \in \overline{\mathbf{u}} + \left(H_0^1(\Omega)\right)^2$ such that:

$$\int_\Omega \boldsymbol{\sigma}(\mathbf{u}) : \boldsymbol{\varepsilon}(\mathbf{v}) = \int_\Omega \mathbf{f}\mathbf{v} \quad \forall \mathbf{v} \in \left(H_0^1(\Omega)\right)^2,$$

where $\overline{\mathbf{u}} = \mathbf{g}$ is the natural lifting of the boundary data. Such a formulation is used to compute the quantity R_h^2 in (2.8), where the modified residuals

$$r_{h,i}(\mathbf{w}) = \int_\Omega \mathbf{f}\boldsymbol{\varphi}_i^v - \int_\Omega \boldsymbol{\sigma}(\mathbf{w}) : \boldsymbol{\varepsilon}(\boldsymbol{\varphi}_i^v), \quad i \in I_h,$$

replace the ones defined in (2.7). For this and the subsequent test cases, we will also provide a comparison with the results obtained by a PINN, in order to give a more complete view of the performance of the methods. The modified residuals required in the PINN loss function are defined as:

$$r_i^{\text{PINN}}(\mathbf{u}) = \nabla \cdot \boldsymbol{\sigma}(x_i) + \mathbf{f}(x_i) \quad \forall i = 1, 2, \dots, N_I.$$

Since the exact solution is not known, we produce a very accurate numerical solution for comparison (shown in Figs. 14a and 14b), using the open-source FEM solver FEniCS [46].

Problem (4.6) is solved by training a VPINN on the mesh shown in Fig. 1c with $q = 3$ and $k_{\text{test}} = 1$. Then, in order to compare the performance of PINN and VPINN, a standard PINN is trained to solve the same problem. In order to verify if the distribution of the collocation points affects the PINN accuracy, we firstly train it by choosing as collocation points the interpolation nodes used in the VPINN training, and then we train it with the same number of uniformly distributed collocation points.

For these three methods we analyze the H^1 error during the neural network training for a fixed training set dimension; we report the results in Figs. 15a–15c. Observing that Figs. 15b and 15c are very similar, we deduce that, in this case, the choice of control points in the PINN training is not strictly related to the efficacy of the different approaches.

It can be observed that method $\mathbf{M_B}$ is always the most efficient approach and leads to convergence to more accurate solutions. Exactly imposing the Dirichlet boundary conditions via $\mathbf{M_C}$ can be considered a good alternative since the solutions at convergence obtained with the VPINN and the PINN trained with random control points are very similar to the ones computed using $\mathbf{M_B}$, although the convergence is slower. The most commonly used approach, $\mathbf{M_A}$, is instead dependent on the choice of the non-trainable parameter λ. In this case, large values of λ ensure accurate solutions and acceptably efficient training phases, but the correct values are problem dependent and can be often found only after a potentially expensive tuning. Indeed, choosing the wrong values of λ can ruin the efficiency and the accuracy of the method, as it can be observed in Fig. 15 when $\lambda = 10^{-3}$ or $\lambda = 1$. We also highlight that the performance of method $\mathbf{M_D}$ is very similar to method $\mathbf{M_A}$ when reasonable values of λ are chosen.

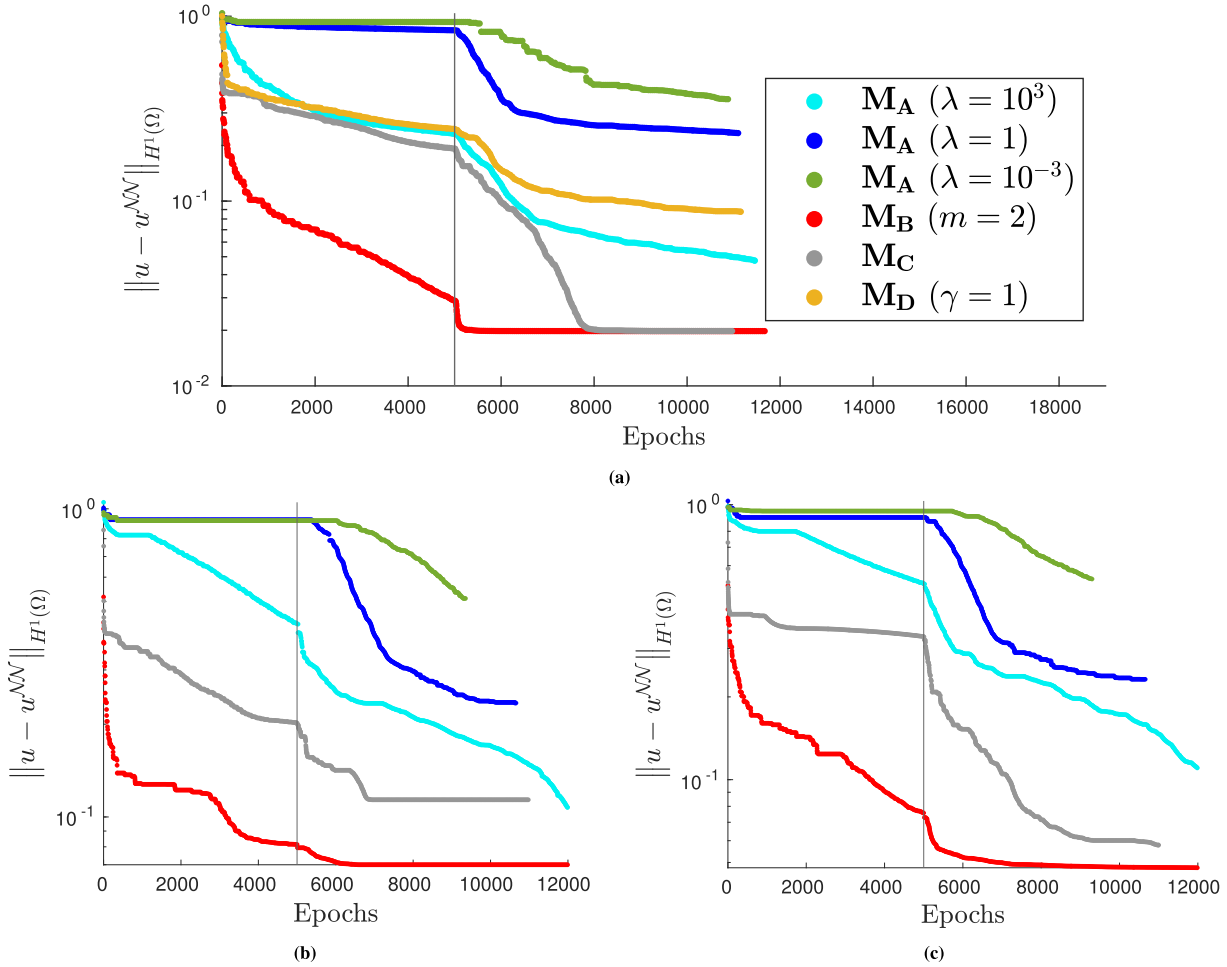


Fig. 15. H^1 error decay during the neural network training when solving problem (4.6). (a) VPINN error: H^1 error of the most accurate solution is 0.020; (b) PINN error: model is trained with collocation points distributed on a Delaunay mesh and the H^1 error of the most accurate solution is 0.070; and (c) PINN error: model is trained with collocation points from a uniform distribution and the H^1 error of the most accurate solution is 0.047. The legend in (a) also applies to (b) and (c).

4.4. Stabilized Eikonal equation

In this section we consider the stabilized Eikonal equation, which is a nonlinear second-order PDE and reads as:

$$\begin{cases} -\varepsilon \Delta u + \|\nabla u\|_2 = f & \text{in } \Omega_L, \\ u = g & \text{on } \Gamma_D, \end{cases} \tag{4.7}$$

where ε is a small positive constant. Note that when $\varepsilon = 0$, $f = 1$ and $g = 0$, the exact solution is the distance function to the boundary and the problem can be efficiently solved by the fast sweeping method [47] or by the fast marching method [48]. In our numerical computations we set $f = 1$ and $g = 0$ and we introduce a weak diffusivity with $\varepsilon = 0.1$ to guarantee uniqueness of the solution.

The PINN and VPINN residuals associated with problem (4.7) that extend the residuals in (2.3) and (2.7), respectively, are defined as:

$$r_i^{\text{PINN}}(w) = -\varepsilon \Delta w(x_i) + \|\nabla w(x_i)\|_2 - f(x_i) \quad \forall i = 1, \dots, N_I,$$

and

$$r_{h,i}(w) = \int_\Omega f \varphi_i^v - \int_\Omega \varepsilon \nabla w \nabla \varphi_i^v - \int_\Omega \|\nabla w\|_2 \varphi_i^v, \quad i \in I_h.$$

We compute the VPINN and PINN numerical solutions as described in Section 4.3 and compute the corresponding H^1 errors using a finite element reference solution that is computed on a much finer mesh (see Fig. 16).

As in Fig. 15, in Fig. 17 we show the decay of the H^1 error during the training for the different methods. Again, it can be noted that the most accurate method is always $\mathbf{M_B}$; $\mathbf{M_A}$ is a valid alternative provided λ is properly chosen. However, when the value of

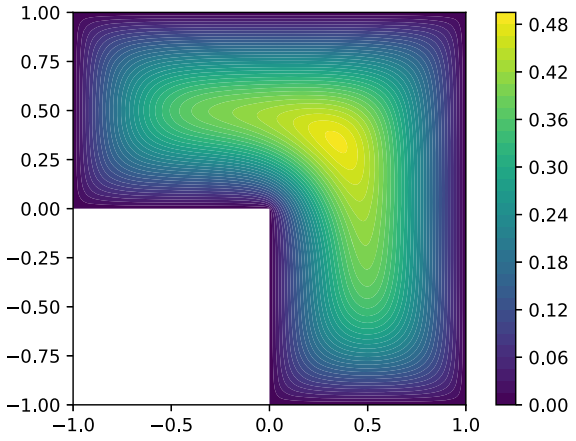


Fig. 16. Reference finite element solution for problem (4.7).

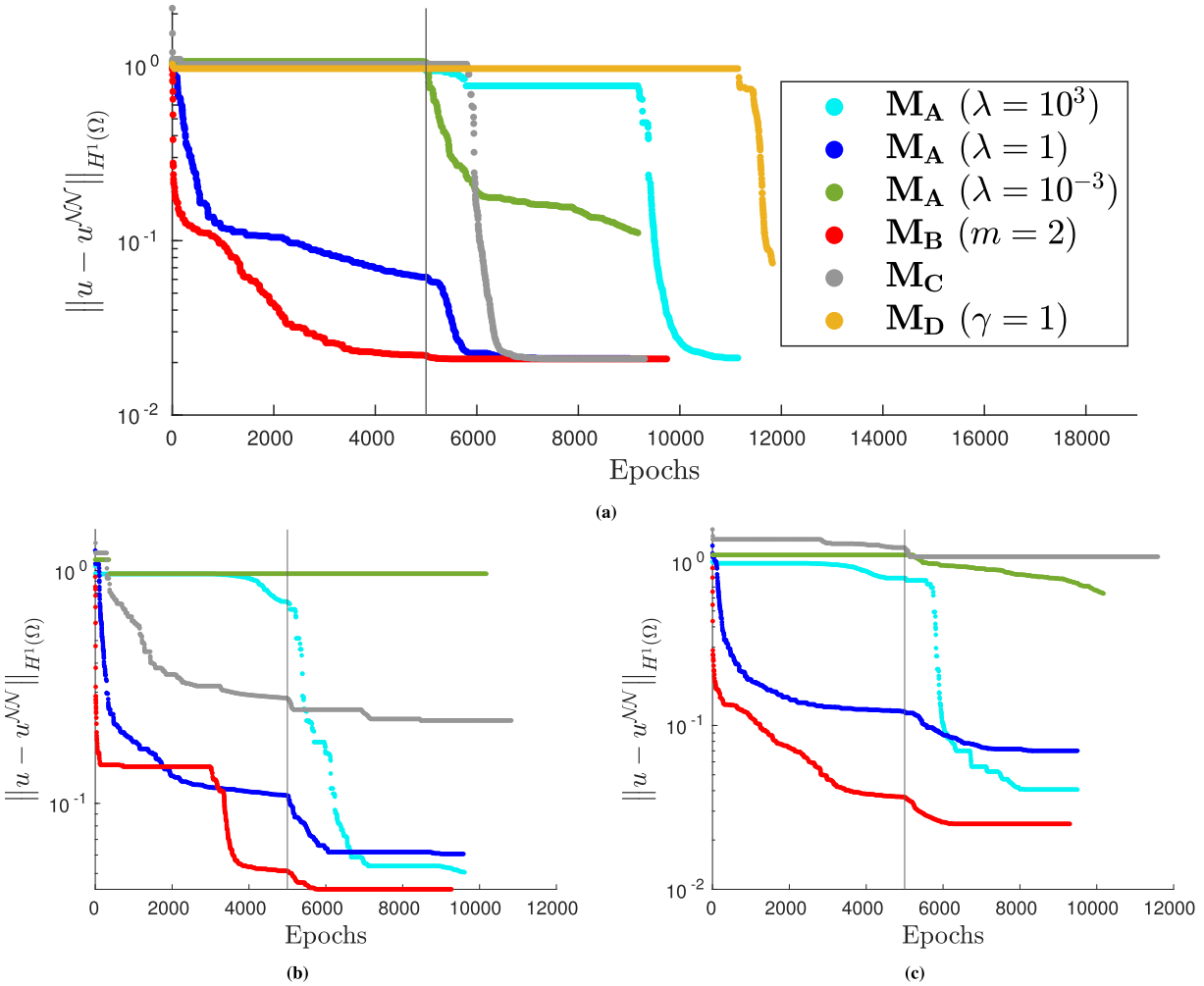


Fig. 17. H^1 error decay during the neural network training when solving problem (4.7). (a) VPINN error: H^1 error of the most accurate solution is 0.021; (b) PINN error: model is trained with collocation points distributed on a Delaunay mesh and the H^1 error of the most accurate solution is 0.085; and (c) PINN error: model is trained with collocation points from a uniform distribution and the H^1 error of the most accurate solution is 0.029. The legend in (a) also applies to (b) and (c).

λ is not suitably chosen, convergence can be completely ruined (see, for instance, the curves associated with $\lambda = 10^{-3}$ in Figs. 17b and 17c) or a second-order optimizer is required to retain convergence (see all the curves computed with $\lambda = 10^3$). Moreover, similar convergence issues are present when $\mathbf{M_C}$ or $\mathbf{M_D}$ are employed.

4.5. One-dimensional convection problem

As a final example, we consider a one-dimensional convection problem on the space-time domain $\Omega := \Omega_x \times \Omega_t = [0,1] \times [0,1]$. As discussed in [49], when solving such a hyperbolic PDE with PINN, possible failure modes may arise due to the very complex loss landscape. The model problem reads as:

$$\begin{cases} \dfrac{\partial u}{\partial t} + \beta \dfrac{\partial u}{\partial x} = 0, & \forall x \in \Omega_x = [0,1],\ t \in \Omega_t = [0,1], \\ u(0,t) = g(t) & \forall t \in \Omega_t, \\ u(x,0) = h(x) & \forall x \in \Omega_x. \end{cases} \tag{4.8}$$

Let us consider the boundary condition $g(t) = -\sin(\beta t)$ and the initial condition $h(x) = \sin(x)$. The corresponding exact solution is $u(x,t) = \sin(x - \beta t)$. We solve problem (4.8) with the convection coefficient $\beta = 30$.

Given a set of collocation points $(x_i, t_i) \in \Omega$, $i = 1, \dots, N_I$ and a suitable set of space-time test functions $V_h := \text{span}\{\varphi_i^v = \varphi_i^v(x,t) : i \in I_h\}$, the PINN and VPINN residuals that are used to train the models are given by

$$r_i^{\text{PINN}}(w) = \frac{\partial w}{\partial t}(x_i, t_i) + \beta \frac{\partial w}{\partial x}(x_i, t_i) \quad \forall i = 1, 2, \dots, N_I$$

and

$$r_{h,i}(w) = \int_\Omega \left[\frac{\partial w}{\partial t} + \beta \frac{\partial w}{\partial x} \right] \varphi_i^v, \quad i \in I_h,$$

respectively. When the boundary conditions are exactly imposed (i.e., when $\mathbf{M_B}$ or $\mathbf{M_C}$ are used), the function $\phi = \phi(x,t)$ is constructed as $\phi(x,t) := \phi_x(x)\phi_t(t)$, where $\phi_t(t) = t$ and $\phi_x(x)$ is a function that vanishes on the Dirichlet boundary of Ω_x. Note that, due to the simplicity of the spatial domain Ω_x, there is no reason to distinguish between $\mathbf{M_B}$ and $\mathbf{M_C}$. Therefore, we just consider the function $\phi_x(x) = x$ in both approaches.

The numerical results obtained using the different approaches are presented in Fig. 18. In Fig. 18a, problem (4.8) is solved with the VPINN method. In this case, $\mathbf{M_A}$ is slightly more accurate and efficient than $\mathbf{M_B}$ (or $\mathbf{M_C}$ since they coincide) if λ is chosen properly. However, when the value of λ is not optimal, the solution is significantly less accurate. Once more, method $\mathbf{M_D}$ is not competitive with the other approaches. On the other hand, when PINN is considered, exactly imposing the boundary conditions ensures better accuracy and efficiency than using $\mathbf{M_A}$, regardless of the value of λ (see Figs. 18b and 18c).

5. Conclusions

In this paper, we analyzed the formulation and the performance of four different approaches to enforce Dirichlet boundary conditions in PINNs and VPINNs on arbitrary polygonal domains. In the first approach, which is the most commonly used when training PINNs, the boundary conditions are imposed by means of additional terms in the loss function that penalize the discrepancy between the neural network output and the prescribed boundary conditions. The subsequent two approaches exactly enforce the boundary conditions and differ in the way they modify the model output in order to force it to satisfy the desired conditions. The last approach, which can be used only when the loss function is derived from the weak formulation of the PDE, is based on Nitsche's method and enforces the boundary conditions variationally.

We have shown that $\mathbf{M_B}$ and $\mathbf{M_D}$, in the considered second-order elliptic PDEs, always ensure the theoretically predicted convergence rate with respect to mesh refinement, regardless of the value of the involved parameter. Instead, method $\mathbf{M_A}$ and $\mathbf{M_C}$ ensure it only if the exact solution is not characterized by an intense oscillatory behavior.

In general, we observed that the most efficient and accurate approach is the one introduced in [38] (method $\mathbf{M_B}$), which is based on the use of a class of approximate distance functions. A variant of this approach (method $\mathbf{M_C}$) leads to suboptimal results and may even ruin the convergence of the method (as in Fig. 17c). Imposing the boundary conditions via additional cost (method $\mathbf{M_A}$) can be considered a valid alternative, but the choice of the additional penalization parameter is crucial because wrong values can prevent convergence to the correct solution or dramatically slow down the training. In the proposed numerical experiments we fixed the penalization parameter. As discussed in the Introduction, we highlight that it is possible to tune it during training, but we chose to fix it in order to compare non-intrusive methods with simple implementations. Finally, we observed that Nitsche's method (method $\mathbf{M_D}$) is in some cases similar to $\mathbf{M_A}$ with an acceptable value of λ, while in other cases requires a second-order optimizer to converge to the correct solution.

Among possible extensions of this work, we mention applications to high-dimensional PDEs over complex geometries, where we expect methods $\mathbf{M_B}$ and $\mathbf{M_C}$ to be even more efficient than their alternatives. In fact, such methods can enforce the correct conditions on each portion of the boundary, whereas methods $\mathbf{M_A}$ and $\mathbf{M_D}$ are likely to be less robust and efficient.

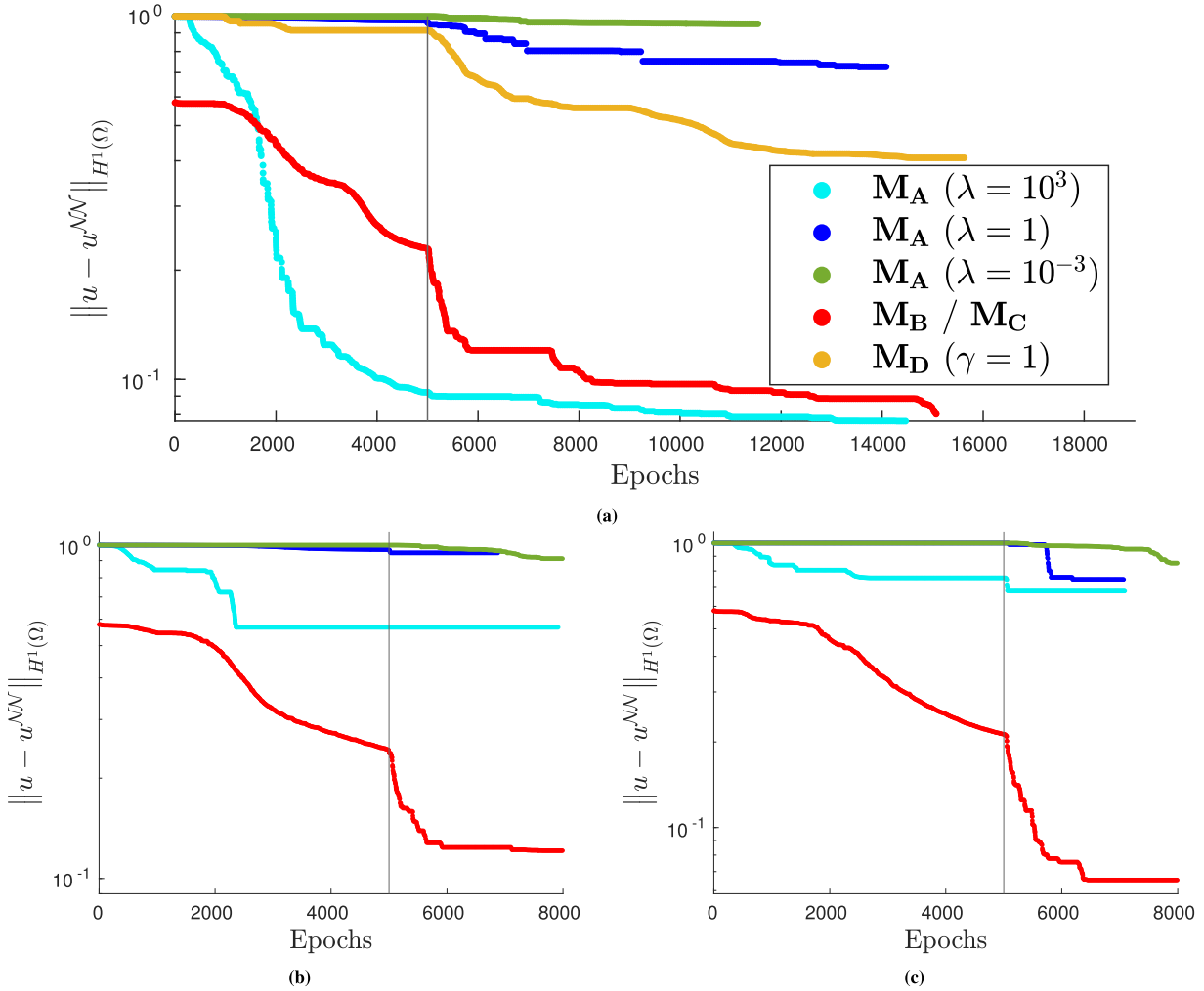


Fig. 18. H^1 error decay during the neural network training when solving problem (4.8). (a) VPINN error: H^1 error of the most accurate solution is 0.077; (b) PINN error: model is trained with collocation points distributed on a Delaunay mesh and the H^1 error of the most accurate solution is 0.125; and (c) PINN error: model is trained with collocation points from a uniform distribution and the H^1 error of the most accurate solution is 0.051. The legend in (a) also applies to (b) and (c).

CRediT authorship contribution statement

S. Berrone; C. Canuto; N. Sukumar: Conceived and designed the experiments; Analyzed and interpreted the data; Contributed reagents, materials, analysis tools or data.

M. Pintore: Conceived and designed the experiments; Performed the experiments; Analyzed and interpreted the data; Contributed reagents, materials, analysis tools or data; Wrote the paper.

Declaration of competing interest

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

Data availability statement

Data will be made available on request.

Acknowledgements

CC, SB and MP performed this research in the framework of the Italian MIUR Award “Dipartimenti di Eccellenza 2018-2022” granted to the Department of Mathematical Sciences, Politecnico di Torino (CUP: E11G18000350001). The research leading to this

paper has also been partially supported by the SmartData@PoliTO center for Big Data and Machine Learning technologies. SB was supported by the Italian MIUR PRIN Project 201744KLJL-004, CC was supported by the Italian MIUR PRIN Project 201752HKH8-003. CC, SB and MP are members of the Italian INdAM-GNCS research group.

Appendix A. On the Laplacian of the approximate distance function

In [38], the issue of the blowing-up of the Laplacian of ϕ in (3.4) is discussed. Herein, we illustrate the same for the simple setting in which Γ_D is composed of two edges that intersect.

Let Ω be the non-negative quadrant $\{(x,y) : x \geq 0, y \geq 0\}$ and let us consider the (semi-infinite) segments $s_1 = \{(x,0) : x \geq 0\}$ and $s_2 = \{(0,y) : y \geq 0\}$. For the sake of simplicity, we consider the exact distance functions $\phi_1(x,y) = x$ and $\phi_2(x,y) = y$. Let us compute the ADF of order $m = 1$ to the boundary $\Gamma_D = s_1 \cup s_2$. Substituting $m = 1$ in (3.4), ϕ can be written as:

$$\phi = \frac{\phi_1\phi_2}{\phi_1 + \phi_2} = \frac{xy}{x+y}.$$

The gradient of ϕ is:

$$\nabla\phi = \left[\frac{\partial\phi}{\partial x}, \frac{\partial\phi}{\partial y}\right]^T = \left[\frac{(x+y)y - xy}{(x+y)^2}, \frac{(x+y)x - xy}{(x+y)^2}\right]^T = \left[\frac{y^2}{(x+y)^2}, \frac{x^2}{(x+y)^2}\right]^T.$$

Note that the gradient is bounded on Γ_D, and in particular:

$$\left.\frac{\partial\phi}{\partial x}\right|_{x=0} = 1, \qquad \left.\frac{\partial\phi}{\partial y}\right|_{y=0} = 1, \qquad \nabla\phi|_{y=\alpha x} = \left[\frac{\alpha^2}{(1+\alpha)^2}, \frac{1}{(1+\alpha)^2}\right]^T,$$

where α is a strictly positive constant, i.e. ϕ is an ADF of order 1 and $\nabla\phi$ is bounded along any straight line intersecting the origin and entering inside the domain. The Laplacian of ϕ is:

$$\Delta\phi = \frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} = \frac{\partial}{\partial x}\left[\frac{y^2}{(x+y)^2}\right] + \frac{\partial}{\partial y}\left[\frac{x^2}{(x+y)^2}\right] = -2\frac{x^2+y^2}{(x+y)^3}.$$

Consider the limit at $(0,0)$ along the line $y = \alpha x$, for $\alpha \geq 0$:

$$\lim_{x,y\to 0}\Delta\phi = \lim_{x\to 0} -2\frac{x^2+(\alpha x)^2}{(x+(\alpha x))^3} = \lim_{x\to 0} -2\frac{(1+\alpha^2)x^2}{(1+\alpha)^3x^3} = \lim_{x\to 0} -2\frac{(1+\alpha^2)}{(1+\alpha)^3}\frac{1}{x} = -\infty.$$

Therefore, $\Delta\phi$ is unbounded at the origin.

For $m = 2$, the function ϕ is:

$$\phi = \frac{1}{\sqrt{\frac{1}{\phi_1^2} + \frac{1}{\phi_2^2}}} = \frac{\phi_1\phi_2}{\sqrt{\phi_1^2 + \phi_2^2}} = \frac{xy}{\sqrt{x^2+y^2}}.$$

Its gradient is:

$$\nabla\phi = \left[\frac{y}{\sqrt{x^2+y^2}} - \frac{x^2y}{(x^2+y^2)^{3/2}}, \frac{x}{\sqrt{x^2+y^2}} - \frac{xy^2}{(x^2+y^2)^{3/2}}\right]^T,$$

which in polar coordinates ($x = \rho\cos(\theta)$, $y = \rho\sin(\theta)$) is expressed as:

$$\begin{aligned}\nabla\phi &= \left[\frac{\rho\sin(\theta)}{\rho} - \frac{\rho^3\cos^2(\theta)\sin(\theta)}{\rho^3}, \frac{\rho\cos(\theta)}{\rho} - \frac{\rho^3\cos(\theta)\sin^2(\theta)}{\rho^3}\right]^T, \\ &= \left[\sin(\theta) - \cos^2(\theta)\sin(\theta), \cos(\theta) - \cos(\theta)\sin^2(\theta)\right]^T.\end{aligned}$$

In polar coordinates, we can write

$$\begin{aligned}\frac{\partial^2\phi}{\partial x^2} &= 3\frac{x^3y}{(x^2+y^2)^{5/2}} - 3\frac{xy}{(x^2+y^2)^{3/2}} = 3\frac{\rho^4\cos^3(\theta)\sin(\theta)}{\rho^5} - 3\frac{\rho^2\cos(\theta)\sin(\theta)}{\rho^3} \\ &= 3\frac{\cos(\theta)\sin(\theta)}{\rho}\left[\cos^2(\theta) - 1\right] = -3\frac{\cos(\theta)\sin^3(\theta)}{\rho}.\end{aligned}$$

Similarly,

$$\frac{\partial^2\phi}{\partial y^2} = -3\frac{\cos^3(\theta)\sin(\theta)}{\rho}$$

holds, which implies:

$$\Delta\phi = \frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} = -3\frac{\cos(\theta)\sin(\theta)}{\rho}.$$

Thus, as in the case $m = 1$, $\Delta\phi \to -\infty$ when $\rho \to 0$.

We point out that for $\theta = \pi/2$ and $\theta = 0$, respectively, we note that:

$$\left.\frac{\partial\phi}{\partial x}\right|_{x=0} = 1, \qquad \left.\frac{\partial\phi}{\partial y}\right|_{y=0} = 1, \qquad \left.\frac{\partial^2\phi}{\partial x^2}\right|_{x=0,y>0} = 0, \qquad \left.\frac{\partial^2\phi}{\partial y^2}\right|_{y=0,x>0} = 0.$$

Therefore, ϕ is an ADF that is normalized up to order 2.

References

[1] M. Raissi, P. Perdikaris, G. Karniadakis, Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, J. Comput. Phys. 378 (2019) 686–707.

[2] I.E. Lagaris, A. Likas, D.I. Fotiadis, Artificial neural network methods in quantum mechanics, Comput. Phys. Commun. 104 (1997) 1–14.

[3] I.E. Lagaris, A. Likas, D.I. Fotiadis, Artificial neural networks for solving ordinary and partial differential equations, IEEE Trans. Neural Netw. 9 (1998) 987–1000.

[4] I.E. Lagaris, A.C. Likas, D.G. Papageorgiou, Neural-network methods for boundary value problems with irregular boundaries, IEEE Trans. Neural Netw. 11 (2000) 1041–1049.

[5] Y. Chen, L. Lu, G.E. Karniadakis, L.D. Negro, Physics-informed neural networks for inverse problems in nano-optics and metamaterials, Opt. Express 28 (2020) 11618–11633.

[6] Q. Guo, Y. Zhao, C. Lu, J. Luo, High-dimensional inverse modeling of hydraulic tomography by physics informed neural network (HT-PINN), J. Hydrol. 616 (2023) 128828.

[7] S. Mishra, R. Molinaro, Estimates on the generalization error of physics-informed neural networks for approximating a class of inverse problems for PDEs, IMA J. Numer. Anal. (2021).

[8] H. Gao, L. Sun, J.-X. Wang, PhyGeoNet: physics-informed geometry-adaptive convolutional neural networks for solving parameterized steady-state PDEs on irregular domain, J. Comput. Phys. 428 (2021) 110079.

[9] J. Han, A. Jentzen, E. Weinan, Solving high-dimensional partial differential equations using deep learning, Proc. Natl. Acad. Sci. 115 (2018) 8505–8510.

[10] S. Lanthaler, S. Mishra, G.E. Karniadakis, Error estimates for DeepONets: a deep learning framework in infinite dimensions, Trans. Math. Appl. 6 (2022).

[11] X. Jiang, D. Wang, Q. Fan, M. Zhang, C. Lu, A.P. Tao Lau, Solving the nonlinear Schrödinger equation in optical fibers using physics-informed neural network, in: 2021 Optical Fiber Communications Conference and Exhibition (OFC), 2021, pp. 1–3.

[12] Z. Chen, Y. Liu, H. Sun, Physics-informed learning of governing equations from scarce data, Nat. Commun. 12 (2021) 1–13.

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

[14] A.D. Jagtap, E. Kharazmi, G.E. Karniadakis, Conservative physics-informed neural networks on discrete domains for conservation laws: applications to forward and inverse problems, Comput. Methods Appl. Mech. Eng. 365 (2020) 113028.

[15] A.D. Jagtap, G.E. Karniadakis, Extended physics-informed neural networks (XPINNs): a generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations, Commun. Comput. Phys. 28 (2020) 2002–2041.

[16] E. Kharazmi, Z. Zhang, G. Karniadakis, VPINNs: variational physics-informed neural networks for solving partial differential equations, arXiv preprint, arXiv: 1912.00873, 2019.

[17] E. Kharazmi, Z. Zhang, G. Karniadakis, *hp*-VPINNs: variational physics-informed neural networks with domain decomposition, Comput. Methods Appl. Mech. Eng. 374 (2021) 113547.

[18] T. De Ryck, A. Jagtap, S. Mishra, Error estimates for physics-informed neural networks approximating the Navier-Stokes equations, IMA J. Numer. Anal. (2023).

[19] T. De Ryck, S. Mishra, Error analysis for physics informed neural networks (PINNs) approximating Kolmogorov PDEs, Adv. Comput. Math. 48 (2022).

[20] N. Demo, M. Strazzullo, G. Rozza, An extended physics informed neural network for preliminary analysis of parametric optimal control problems, arXiv preprint, arXiv:2110.13530, 2021.

[21] R. Hu, Q. Lin, A. Raydan, S. Tang, Higher-order error estimates for physics-informed neural networks approximating the primitive equations, arXiv preprint, arXiv:2209.11929, 2022.

[22] J. Pu, J. Li, Y. Chen, Solving localized wave solutions of the derivative nonlinear Schrödinger equation using an improved PINN method, Nonlinear Dyn. 105 (2021) 1723–1739.

[23] J. Sirignano, K. Spiliopoulos, DGM: a deep learning algorithm for solving partial differential equations, J. Comput. Phys. 375 (2018) 1339–1364.

[24] A. Tartakovsky, C. Marrero, P. Perdikaris, G. Tartakovsky, D. Barajas-Solano, Learning parameters and constitutive relationships with physics informed deep neural networks, arXiv preprint, arXiv:1808.03398, 2018.

[25] L. Yang, X. Meng, G. Karniadakis, B-PINNs: Bayesian physics-informed neural networks for forward and inverse PDE problems with noisy data, J. Comput. Phys. 425 (2021) 109913.

[26] Y. Zhu, N. Zabaras, P. Koutsourelakis, P. Perdikaris, Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data, J. Comput. Phys. 394 (2019) 56–81.

[27] C. Beck, M. Hutzenthaler, A. Jentzen, B. Kuckuck, An overview on deep learning-based approximation methods for partial differential equations, Discrete Contin. Dyn. Syst., Ser. B (2022).

[28] S. Cuomo, V.S. Di Cola, F. Giampaolo, G. Rozza, M. Raissi, F. Piccialli, Scientific machine learning through physics-informed neural networks: where we are and what's next, J. Sci. Comput. 92 (2022).

[29] Z. Lawal, H. Yassin, D. Lai, A. Che Idris, Physics-informed neural network (PINN) evolution and beyond: a systematic literature review and bibliometric analysis, Big Data Cogn. Comput. 6 (2022).

[30] L.D. McClenny, U.M. Braga-Neto, Self-adaptive physics-informed neural networks, J. Comput. Phys. 474 (2023) 111722.

[31] S. Wang, Y. Teng, P. Perdikaris, Understanding and mitigating gradient flow pathologies in physics-informed neural networks, SIAM J. Sci. Comput. 43 (2021) A3055–A3081.

[32] C.L. Wight, J. Zhao, Solving Allen-Cahn and Cahn-Hilliard equations using the adaptive physics informed neural networks, Commun. Comput. Phys. 29 (2021) 930–954.

[33] K. Tang, X. Wan, Q. Liao, Adaptive deep density approximation for Fokker-Planck equations, J. Comput. Phys. 457 (2022) 111080.

[34] X. Feng, L. Zeng, T. Zhou, Solving time dependent Fokker-Planck equations via temporal normalizing flow, arXiv preprint, arXiv:2112.14012, 2021.

[35] S. Wang, X. Yu, P. Perdikaris, When and why pinns fail to train: a neural tangent kernel perspective, J. Comput. Phys. 449 (2022) 110768.

[36] S. Berrone, C. Canuto, M. Pintore, Variational physics informed neural networks: the role of quadratures and test functions, J. Sci. Comput. 92 (2022) 1–27.

[37] J.M. Taylor, D. Pardo, I. Muga, A deep Fourier residual method for solving PDEs using neural networks, Comput. Methods Appl. Mech. Eng. 405 (2023) 115850.

[38] N. Sukumar, A. Srivastava, Exact imposition of boundary conditions with distance functions in physics-informed deep neural networks, Comput. Methods Appl. Mech. Eng. 389 (2022) 114333.

[39] A. Biswas, V. Shapiro, Approximate distance fields with non-vanishing gradients, Graph. Models 66 (2004) 133–159.

[40] J.A. Nitsche, Uber ein Variationsprinzip zur Losung Dirichlet-Problemen bei Verwendung von Teilraumen, die keinen Randbedingungen unteworfen sind, Abh. Math. Semin. Univ. Hamb. 36 (1971) 9–15.

[41] D.P. Kingma, J. Ba, Adam: a method for stochastic optimization, arXiv preprint, arXiv:1412.6980, 2014.
[42] S. Wright, J. Nocedal, et al., Numerical Optimization, vol. 35, 1999.
[43] M. Abadi, et al., TensorFlow: large-scale machine learning on heterogeneous systems, http://tensorflow.org/, 2015, software available from tensorflow.org.
[44] A.G. Baydin, B.A. Pearlmutter, A.A. Radul, J.M. Siskind, Automatic differentiation in machine learning: a survey, J. Mach. Learn. Res. 18 (2018).
[45] S. Berrone, C. Canuto, M. Pintore, Solving PDEs by variational physics-informed neural networks: an a posteriori error analysis, Ann. Univ. Ferrara 68 (2022) 575–595.
[46] M.S. Alnaes, J. Blechta, J. Hake, A. Johansson, B. Kehlet, A. Logg, C. Richardson, J. Ring, M.E. Rognes, G.N. Wells, The FEniCS project version 1.5, Arch. Numer. Softw. 3 (2015).
[47] H. Zhao, A fast sweeping method for Eikonal equations, Math. Comput. 74 (2005) 603–627.
[48] J.A. Sethian, Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science, vol. 3, Cambridge University Press, 1999.
[49] A. Krishnapriyan, A. Gholami, S. Zhe, R. Kirby, M.W. Mahoney, Characterizing possible failure modes in physics-informed neural networks, Adv. Neural Inf. Process. Syst. 34 (2021) 26548–26560.