POLITECNICO DI TORINO
Repository ISTITUZIONALE

Two continuous extensions of the Neural Approximated Virtual Element Method

Original
Two continuous extensions of the Neural Approximated Virtual Element Method / Berrone, S., Pintore, M., Teora, G.. - In: JOURNAL OF COMPUTATIONAL PHYSICS. - ISSN 0021-9991. - 568:(2027), pp. 1-25. [10.1016/j.jcp.2026.115392]

Availability:
This version is available at: 11583/3015948 since: 2026-09-26T09:12:11Z

Publisher:
Elsevier

Published
DOI:10.1016/j.jcp.2026.115392

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

20 November 2026

Journal of Computational Physics 568 (2027) 115392

Contents lists available at ScienceDirect

Journal of Computational Physics

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

Two continuous extensions of the neural approximated virtual element method

Stefano Berrone [a,1], Moreno Pintore [b], Gioana Teora [a,1,*]

[a] Dipartimento di Scienze Matematiche "G. L. Lagrange", Politecnico di Torino, Corso Duca degli Abruzzi 24, Torino, 10129, Italy
[b] Sorbonne Université, MEGAVOLT Team, Inria, 4 place Jussieu, Paris, 75005, France

ARTICLE INFO

Keywords:
NAVEM
VEM
Neural network
PINN
Basis functions
Polygonal meshes

ABSTRACT

We propose two globally continuous neural-based variants of the Neural Approximated Virtual Element Method (NAVEM), termed B-NAVEM and P-NAVEM. Both approaches construct local basis functions using pre-trained fully connected neural networks while ensuring exact continuity across adjacent mesh elements. B-NAVEM leverages a Physics-Informed Neural Network to approximately solve the local Laplace problem that defines the virtual element basis functions, whereas P-NAVEM directly enforces polynomial reproducibility via a tailored loss function, without requiring harmonicity within the element interior. Numerical experiments assess the methods in terms of computational cost and accuracy during both training and testing phases.

1. Introduction

The interest in Galerkin methods for the approximation of solutions to Partial Differential Equations (PDEs) based on polytopal (i.e. polygonal and polyhedral) meshes has significantly grown in the last decade, due to the high flexibility that such meshes allow. Indeed, the use of polytopal meshes inherently allows for non-convex elements and hanging nodes, thus providing a natural framework for the treatment of interface problems, mesh refinement strategies, and the efficient approximation of complex geometric features [1–3]. Examples of polytopal methods include Hybrid High-Order methods (HHO) [4], Hybrid Discontinuous Galerkin methods (HDGM) [5], and polygonal discontinuous Galerkin methods (DG-FEM) [6].

The Virtual Element Method (VEM) is an alternative approach that enables discretization on polytopal meshes [7,8]. It is based on globally continuous discretization spaces whose trial and test functions are implicitly defined as solutions of a local Laplace problem on each element. For the lowest-order, the elemental VEM space is described as

$$V_h^{\mathrm{V}}(E) = \Big\{ v \in H^1(E) : \quad (i)\ \Delta v = 0, \\ (ii)\ v \in C^0(\partial E) : v_{|e} \in \mathbb{P}_1(e),\ \forall e \subset \partial E \Big\}, \tag{1}$$

where we adopt standard conventions for functional spaces, while E and ∂E denote a polygonal element and its boundary, respectively. A remarkable property of this space is that $\mathbb{P}_1(E) \subseteq V_h^{\mathrm{V}}(E)$, i.e. it allows an exact reproduction of polynomials of degree up to the order of the method. Within the virtual element framework, these local PDE problems are never solved, neither exactly nor

* Corresponding author.
E-mail addresses: stefano.berrone@polito.it (S. Berrone), moreno.pintore@sorbonne-universite.fr (M. Pintore), gioana.teora@polito.it (G. Teora).
[1] The authors are members of the INdAM-GNCS.

https://doi.org/10.1016/j.jcp.2026.115392
Received 19 January 2026; Received in revised form 25 July 2026; Accepted 13 September 2026
Available online 21 September 2026
0021-9991/© 2026 The Author(s). Published by Elsevier Inc. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

approximately. As a consequence, the bilinear form, and thereby the entries of the stiffness matrix, are not directly computable. The computable version relies on an approximate discrete bilinear form consisting of two additive parts: a projection-based term that guarantees polynomial consistency and a computable stabilization term that restores coercivity. Despite VEM remarkable flexibility, the usage of stability and projector operators may lead to several issues, especially in the presence of anisotropic problems, eigenvalue problems, nonlinear problems, or in the post-processing phase of the quantities of interest [9,10]. For these reasons, new methods have been introduced to approximate VEM functions while retaining VEM advantages and mitigating its limitations. Examples include the Reduced-Basis Virtual Element Method (rbVEM) [9,11], the Lighting Virtual Element Methods (L-VEM) [12,13], and the Neural Approximated Virtual Element Method (NAVEM) [14,15]. The last two approaches describe virtual functions as a linear combination of proper harmonic functions: L-VEM determines these coefficients by using the Laplace Solver [16], while NAVEM leverages neural networks to exploit an efficient offline-online splitting strategy.

Since the elemental approximations in both L-VEM and NAVEM are computed independently on each mesh element, the resulting basis functions are not continuous across adjacent elements. Nevertheless, these methods maintain a C^0-conforming framework without doubling the degrees of freedom along element interfaces. Since virtual basis functions are known in closed form on the boundaries of elements, no approximation is required there, which prevents the duplication of boundary degrees of freedom. Nonetheless, the lack of inter-element continuity introduces an additional term in the error estimate that increases as the mesh size decreases [10].

In this paper, we propose two variants of NAVEM that restore the continuity of the basis functions across neighbouring elements. The involved neural networks retain the same kind of architecture as the standard NAVEM, namely a fully-connected feed-forward neural network [17], but differ in the definition of their inputs/outputs and loss functions.

In particular, the first variant, referred to as Boundary-NAVEM (B-NAVEM), employs a Physics-Informed Neural Network (PINN) with exact enforcement of Dirichlet boundary conditions [18,19] to approximate the local Laplace problem (1) that defines the VEM Lagrange basis functions. In this setting, the loss function aims to minimize the residual associated with the elemental PDE, following the standard PINN paradigm. The second variant, called Polynomial-NAVEM (P-NAVEM), adopts the same technique to enforce the Dirichlet boundary values as B-NAVEM, but employs a different loss function that aims at reducing the polynomial reproducibility error. The main goal here is to recover this key property, which is required to guarantee optimal polynomial convergence rates. We emphasize that, in both variants, the basis functions are exact at the boundary of mesh elements, unlike in the standard NAVEM method, but the functions belonging to these new spaces are no longer exactly harmonic functions in the interior of the elements, like VEM and NAVEM functions. More precisely, B-NAVEM basis functions are harmonic up to the neural network accuracy, whereas in P-NAVEM this property is neither enforced exactly nor approximately.

The proposed two variants are compared with the standard NAVEM in terms of memory usage, computational time, and accuracy in both the training and the testing phase, through a series of numerical experiments.

The paper is organized as follows. Section 2 introduces the neural network strategy that is shared by the three approaches to solve a simple Poisson Problem. In Section 3, we detail the main features of the standard NAVEM construction. Section 4 describes the strategy adopted to modify the neural network output in order to exactly enforce the Dirichlet boundary data, which underlies the B-NAVEM strategy, detailed in Section 5, and the P-NAVEM approach, which is described in Section 6. Sections 5 and 6 also provide the corresponding theoretical error estimates for the B-NAVEM and P-NAVEM methods, respectively. Section 7 reports a set of numerical experiments comparing the different NAVEM strategies and highlighting their advantages with respect to the classical VEM discretization. Finally, Section 8 summarizes the main findings and outlines future research directions.

The complete implementation of all the NAVEM methods is publicly available through the PolyNAVEM library, an open-source Python library built on top of the C++ PolyDiM library [20].

2. A neural network strategy to approximate basis functions

In the following, the superscript $\mathcal{NN}$ will refer to a generic neural-based method, whereas the superscripts H, B and P will denote the specific underlying method, namely the standard NAVEM (or H-NAVEM), B-NAVEM, and P-NAVEM, respectively.

Given a polygonal domain $\Omega \in \mathbb{R}^2$ with boundary $\partial\Omega$, let us consider the following Poisson problem

$$\begin{cases} -\Delta u = f & \text{in } \Omega, \\ u = 0 & \text{on } \partial\Omega, \end{cases} \tag{2}$$

where the source term $f \in L^2(\Omega)$.

Introducing the Hilbert space $V = H_0^1(\Omega)$, the variational formulation of Problem (2) reads as: *Find $u \in V$ such that*

$$a(u, v) = F(v) \quad \forall v \in V, \tag{3}$$

where the bilinear form $a : V \times V \to \mathbb{R}$ and the linear form $F : V \to \mathbb{R}$ are defined as:

$$a(v, w) = \int_\Omega \nabla v \cdot \nabla w, \quad \forall v, w \in V,$$

$$F(v) = \int_\Omega f\, v, \quad \forall v \in V.$$

Let us introduce a tessellation $\mathcal{T}_h$ of Ω made up of polygonal elements E. We denote by N_E^v the number of vertices of the polygon E and by h_E its diameter. Moreover, we define the mesh size as $h = \max_{E\in\mathcal{T}_h} h_E$. We assume $\mathcal{T}_h$ to satisfy the following mesh assumptions [21].

Assumption 1 (Mesh assumptions). There exists a positive real number $\rho \in (0,1)$, independent of h, such that

- for each element $E \in \mathcal{T}_h$ and each edge $e \subset \partial E$, it holds $|e| \geq \rho h_E$;
- each element $E \in \mathcal{T}_h$ is star-shaped with respect to a ball of radius $\geq \rho h_E$.

Since we consider the lowest-order version, the local virtual element degrees of freedom are the values of functions at the vertices of the element; thus, the number of local degrees of freedom is $N_E^{\mathrm{dof}} = N_E^v$. Moreover, the global degrees of freedom are associated with the vertices of the tessellation $\mathcal{T}_h$ that do not belong to the Dirichlet boundary $\partial\Omega$, and their number is denoted by

$$N^{\mathrm{dof}} = \dim V_h^{\mathrm{V}}. \tag{4}$$

Here, V_h^{V} is the global virtual element space and it is described as

$$V_h^{\mathrm{V}} = \{v \in V \cap C^0\left(\overline{\Omega}\right) : \ v_{|E} \in V_h^{\mathrm{V}}(E) \ \forall E \in \mathcal{T}_h\},$$

where the local virtual element space $V_h^{\mathrm{V}}(E)$ is defined in (1).

As in a standard Galerkin method, the basic idea of a NAVEM method is to define a finite-dimensional subspace $V_h^{\mathcal{NN}} \subset V$ as the span of some basis functions that are known in a closed form. In particular, such basis functions approximate the Lagrange virtual basis functions spanning V_h^{V}. Note that by satisfying

- Property (ii) in (1), we gain the C^0-conformity, i.e. continuity across adjacent elements of $\mathcal{T}_h$;
- Properties (i) and (ii) in (1), we obtain that $\mathbb{P}_k(E) \subseteq V_h^{\mathcal{NN}}$. This *polynomial reproducibility* property allows VEM to recover optimal polynomial error convergence estimates.

In particular, in the following, we approximate the VEM basis functions using neural networks that aim to learn the following nonlinear map

$$(j, E) \to \left(\varphi_{j,E}^{\mathcal{NN}}, \boldsymbol{q}_{j,E}^{\mathcal{NN}}\right), \quad \forall j = 1, \dots, N_E^{\mathrm{dof}}, \quad \forall E \in \mathcal{T}_h. \tag{5}$$

where

- the input (j, E) represents a pair made up by the index j of the elemental degree of freedom (physically represented by a vertex of E) and the element $E \in \mathcal{T}_h$ itself, identifying the elemental Lagrange basis functions we want to approximate.
- the output $\left(\varphi_{j,E}^{\mathcal{NN}}, \boldsymbol{q}_{j,E}^{\mathcal{NN}}\right)$ represents the approximation of the virtual basis function $\varphi_{j,E}$ related to the input pair (j, E) and of its gradient $\nabla\varphi_{j,E}$, respectively.

Given the local approximations $\left(\varphi_{j,E}^{\mathcal{NN}}, \boldsymbol{q}_{j,E}^{\mathcal{NN}}\right)$ for each input pair, we define the local NAVEM spaces as

$$V_h^{\mathcal{NN}}(E) = \mathrm{span}\{\varphi_{j,E}^{\mathcal{NN}} \ : \ j = 1, \dots, N_E^{\mathrm{dof}}\}, \quad \text{and} \quad \nabla V_h^{\mathcal{NN}}(E) = \mathrm{span}\{\boldsymbol{q}_{j,E}^{\mathcal{NN}} \ : \ j = 1, \dots, N_E^{\mathrm{dof}}\}. \tag{6}$$

We can now use standard gluing techniques to produce a full set of global basis functions and of their gradients, i.e. $\left(\varphi_i^{\mathcal{NN}}, \boldsymbol{q}_i^{\mathcal{NN}}\right)$, for each $i = 1, \dots, N^{\mathrm{dof}}$, and define the NAVEM global spaces as

$$V_h^{\mathcal{NN}} = \mathrm{span}\{\varphi_i^{\mathcal{NN}} \ : \ i = 1, \dots, N^{\mathrm{dof}}\}, \quad \text{and} \quad \nabla V_h^{\mathcal{NN}} = \mathrm{span}\{\boldsymbol{q}_i^{\mathcal{NN}} \ : \ i = 1, \dots, N^{\mathrm{dof}}\}.$$

Finally, we can proceed as in a standard finite element method, and solve the following discrete problem: *Find* $u_h \in V_h^{\mathcal{NN}}$ *such that*

$$\sum_{i=1}^{N^{\mathrm{dof}}} u_i \sum_{E \in \mathcal{T}_h} \int_E \boldsymbol{q}_i^{\mathcal{NN}} \cdot \boldsymbol{q}_j^{\mathcal{NN}} = \sum_{E \in \mathcal{T}_h} \int_E f \varphi_j^{\mathcal{NN}}, \quad \forall j = 1, \dots, N^{\mathrm{dof}}. \tag{7}$$

In the following, we describe three different strategies for approximating the VEM basis functions and their gradients, and thus the spaces $V_h^{\mathcal{NN}}$ and $\nabla V_h^{\mathcal{NN}}$, while discussing their properties and advantages. More precisely, we consider

1. the standard NAVEM or **Harmonic-NAVEM (H-NAVEM)** strategy, introduced in [14,15] and denoted here by the letter H to highlight the harmonic nature of its basis functions. Indeed, this approach approximates the VEM basis functions and their gradients as linear combinations of suitably chosen harmonic functions. As a result, Property (i) in (1) is exactly recovered, whereas Property (ii) is only satisfied in an approximate sense, i.e. the NAVEM basis functions $\varphi_i^{\mathrm{H}} \in C^0(\partial E)$, but they are only approximately polynomials of degree one on each edge of the polygon.
2. **Boundary-NAVEM strategy (B-NAVEM)**, denoted by B. In this case, the basis functions are constructed so that Property (ii) is exactly satisfied, while their harmonicity is only approximated. In particular, B-NAVEM basis functions φ_i^{B} attain the exact prescribed values on the boundary of each element $E \in \mathcal{T}_h$ and are such that $\varphi_i^{\mathrm{B}} \in C^0\left(\overline{\Omega}\right)$.
3. **Polynomial-NAVEM strategy (P-NAVEM)**, denoted by P. Since the two previous strategies do not guarantee the exact inclusion $\mathbb{P}_1(E) \subseteq V_h^{\mathcal{NN}}(E)$, we introduce a third approach in which the basis functions are constructed to exactly satisfy Property (ii) while being specifically designed to directly enforce polynomial reproducibility, at least approximately.

Each strategy relies on the same architecture for the underlying neural network, i.e. we only consider standard fully-connected feed-forward neural networks, also known as multi-layer perceptrons [17]. Given the encoding $\boldsymbol{x}_0 \in \mathbb{R}^{N_0}$ of a proper dimension N_0 for the pair (j, E), such architecture can be represented by the following formula:

$$\begin{aligned} &\boldsymbol{x}_\ell = \rho(A_\ell \boldsymbol{x}_{\ell-1} + b_\ell), && \ell = 1, \dots, L-1, \\ &\mathcal{N}(\boldsymbol{x}_0) = A_L \boldsymbol{x}_{L-1} + b_L. \end{aligned} \tag{8}$$

In this formula, the matrices $A_\ell \in \mathbb{R}^{N_\ell \times N_{\ell-1}}$ and the vectors $b_\ell \in \mathbb{R}^{N_\ell}$ contain the trainable weights of the neural network, which are optimized during the training phase, $L \in \mathbb{N}$ is the number of layers, $\rho : \mathbb{R} \to \mathbb{R}$ is a nonlinear scalar activation function which is applied entry-wise to the vector $A_\ell \boldsymbol{x}_{\ell-1} + b_\ell$, and $\mathcal{N}(\boldsymbol{x}_0) \in \mathbb{R}^{N_L}$ represents the neural network output.

We remark that $\mathcal{N}$ approximates a function mapping the vector $\boldsymbol{x}_0 \in \mathbb{R}^{N_0}$ to a target vector $\boldsymbol{x}_L = \mathcal{N}(\boldsymbol{x}_0) \in \mathbb{R}^{N_L}$. Therefore, even though L, ρ, and the intermediate values N_ℓ, $\ell = 1, \dots, L-1$ are model hyperparameters that can be tuned by the user, N_0 and N_L depend on the map that the neural network approximates and cannot be modified. Thus, the different NAVEM strategies may use different values for N_0 and N_L, since the role of the underlying neural network is different in each method. In the following, we always consider the hyperbolic tangent $\rho(x) = \tanh(x)$ as activation function, and we initialize the neural network weights using the Glorot normal initialization [22].

Remark 1. We recall that, to improve the neural network accuracy, we always map each polygon E to a reference polygon $\hat{E}$ through the affine mapping introduced in [15,23] and define all the local basis functions on $\hat{E}$. This approach reduces geometric variability while performing input reduction, thereby improving the neural network accuracy.

3. The H-NAVEM formulation

In this section, for the sake of completeness, we briefly summarize the standard H-NAVEM method. We refer the reader to [15] for further details.

Let us consider the input pair (j, E), we are interested in learning the map (5), i.e. in finding an approximation $\left(\varphi^{\mathrm{H}}_{j,E}, \boldsymbol{q}^{\mathrm{H}}_{j,E}\right)$ that is cheaply computable and accurately approximates $(\varphi_{j,E}, \nabla\varphi_{j,E})$.

Let $z = x_1 + ix_2$ be a complex scalar value and let $\mathfrak{R}(z)$ and $\mathfrak{I}(z)$ be its real and imaginary parts. Let us consider the reference square $S^{\mathrm{H}} = [-R^{\mathrm{H}}, R^{\mathrm{H}}]^2 \subset \mathbb{R}^2$ s.t. $E \subset S^{\mathrm{H}}$ and let $\mathbb{H}_{\ell^{\mathrm{H}}}(S^{\mathrm{H}})$ be the set of scaled harmonic polynomials defined over the squared region S^{H} (see Remark 1) up to order $\ell^{\mathrm{H}} \geq 0$:

$$\mathbb{H}_{\ell^{\mathrm{H}}}(S^{\mathrm{H}}) = \mathrm{span}\left\{1,\ \mathfrak{R}\left(\left(\frac{z}{R^{\mathrm{H}}}\right)^{\ell}\right), \mathfrak{I}\left(\left(\frac{z}{R^{\mathrm{H}}}\right)^{\ell}\right),\ \ell = 1, \dots, \ell^{\mathrm{H}}\right\}. \tag{9}$$

Let us consider the auxiliary problem:

$$\begin{cases} \Delta\tilde{\Phi} = 0 & \text{in } \Omega_\Phi = (-1,1)^2, \\ \tilde{\Phi} = 1 + x_2 & \text{on } \Gamma_{\Phi,1} = \{x_1 = 1 \text{ and } -1 \leq x_2 \leq 0\}, \\ \tilde{\Phi} = 1 - x_2 & \text{on } \Gamma_{\Phi,2} = \{x_1 = 1 \text{ and } 0 \leq x_2 \leq 1\}, \\ \tilde{\Phi} = 0 & \text{on } \partial\Omega_\Phi \setminus \{\Gamma_{\Phi,1} \cup \Gamma_{\Phi,2}\}. \end{cases} \tag{10}$$

By solving a linear least squares problem, we look for an approximate solution of Problem (10) of the form

$$\Phi(z) = \sum_{\alpha=1}^{N^{\Phi,1}} c_\alpha^{\Phi,1} \mathfrak{R}\left(\frac{d_\alpha}{z - z_\alpha}\right) + \sum_{\beta=0}^{N^{\Phi,2}} c_\beta^{\Phi,2} \mathfrak{R}\left(\left(\frac{z}{2}\right)^{\beta}\right), \tag{11}$$

where $z_\alpha = 1 + 2\exp\left(-4(\sqrt{N^{\Phi,1}} - \sqrt{\alpha})\right)$, for $\alpha = 1, \dots, N^{\Phi,1}$, are poles chosen to reproduce the singularity of Φ near the vertex $(1,0)$ of Ω_Φ.

We solve this problem just once on Ω_Φ and then, in order to adapt this representation to a generic element E, we introduce three different linear transformations that map the poles of the function Φ outside E and the point $(1,0)$ belonging to $\partial\Omega_\Phi$ in the $(j-1)$, j, and $(j+1)$th vertex of E, respectively. Applying these three transformations to Φ, we obtain three functions, denoted by $\Phi^{j-1}_{j,E}$, $\Phi^{j}_{j,E}$ and $\Phi^{j+1}_{j,E}$, that help to improve the neural network accuracy. See Remark 3 in [15] to better understand how these functions influence NAVEM accuracy.

We finally introduce the space $\mathcal{H}^{\mathrm{H}}_{j,E}$ of harmonic functions as the space:

$$\mathcal{H}^{\mathrm{H}}_{j,E} = \mathbb{H}_{\ell^{\mathrm{H}}}(S^{\mathrm{H}}) \bigcup \left\{\Phi^{j-1}_{j,E}, \Phi^{j}_{j,E}, \Phi^{j+1}_{j,E}\right\},$$

whose dimension is $\dim\mathcal{H}^{\mathrm{H}}_{j,E} = 2\ell^{\mathrm{H}} + 4$.

To simplify the notation, we denote by $\{\Psi_k\}_{k=1}^{2\ell^{\mathrm{H}}+4}$, the functions spanning $\mathcal{H}^{\mathrm{H}}_{j,E}$, namely the $2\ell^{\mathrm{H}}+1$ basis functions defined in (9) together with the three functions $\Phi^{j-1}_{j,E}$, $\Phi^{j}_{j,E}$, and $\Phi^{j+1}_{j,E}$. Given this approximation space $\mathcal{H}^{\mathrm{H}}_{j,E}$, the VEM basis functions and their gradients are approximated via the standard NAVEM approach as follows: for each input pair (j, E),

$$\varphi_{j,E} \approx \varphi^{\mathrm{H}}_{j,E} = \sum_{k=1}^{\dim\mathcal{H}^{\mathrm{H}}_{j,E}} c_k^{\mathrm{H},\varphi} \Psi_k, \tag{12}$$

$$\nabla\varphi_{j,E} \approx \boldsymbol{q}^{\mathrm{H}}_{j,E} = \sum_{k=1}^{\dim \nabla\mathcal{H}^{\mathrm{H}}_{j,E}} c^{\mathrm{H},\boldsymbol{q}}_{k} \nabla\Psi_{k+1}. \tag{13}$$

The coefficients of the above linear combinations correspond to the outputs $\boldsymbol{c}^{\mathrm{H},\varphi} = \mathcal{N}^{H,\varphi}(\boldsymbol{x}^{\mathrm{H}}_0)$ and $\boldsymbol{c}^{\mathrm{H},\boldsymbol{q}} = \mathcal{N}^{\mathrm{H},\boldsymbol{q}}(\boldsymbol{x}^{\mathrm{H}}_0)$ of two different neural networks for a given input $\boldsymbol{x}^{\mathrm{H}}_0$ that encodes the pair (j, E). The two neural networks share the same architecture, which is defined in (8). In particular, during the training phase, the weights of the neural network $\mathcal{N}^{\mathrm{H},\boldsymbol{q}}$ used to approximate the gradients are initialized with the optimized weights of the neural network $\mathcal{N}^{\mathrm{H},\varphi}$.

Remark 2. We remark that, given the output of the first neural network $\{c^{\mathrm{H},\varphi}_k\}_{k=1}^{\dim,\mathcal{H}^{\mathrm{H}}_{j,E}}$ in (12), then the gradients of the VEM functions might be trivially approximated as

$$\nabla\varphi_{j,E} \approx \sum_{k=1}^{\dim \mathcal{H}^{\mathrm{H}}_{j,E}} c^{\mathrm{H},\varphi}_{k} \nabla\Psi_{k}.$$

Nonetheless, we observed that the usage of a second neural network to approximate the gradients improves the accuracy of the method by producing less oscillating gradients [15].

These neural networks are trained to minimize the mean squared errors over all pairs (j, E) in a suitable training dataset of the following quantities:

$$\epsilon^{\mathrm{H},\varphi}_{j,E} = \|\varphi^{\mathrm{H}}_{j,E} - \varphi_{j,E}\|_{H^{1/2}(\partial E)}, \tag{14}$$

$$\epsilon^{\mathrm{H},\boldsymbol{q}}_{j,E} = \|\left(\boldsymbol{q}^{\mathrm{H}}_{j,E} - \nabla\varphi_{j,E}\right)\cdot \boldsymbol{t}\|_{L^2(\partial E)}, \tag{15}$$

where $\boldsymbol{t}$ denotes the unit tangent vector to the boundary ∂E.

As discussed in [15], these loss functions are computable since virtual basis functions are well known at the boundary of the elements. Moreover, this choice ensures good approximation properties over the entire element E (see [10] for further details). We also observe that Property (i) in (1) is exactly satisfied by any functions that can be expressed as (12), whereas the extent to which Property (ii) is satisfied depends on the accuracy of the trained neural networks.

Note that for these NAVEM neural networks, the following requirements hold:

- the output dimension must coincide with the dimension of $\mathcal{H}^{\mathrm{H}}_{j,E}$ and of $\nabla\mathcal{H}^{\mathrm{H}}_{j,E}$ for the two neural networks, respectively, i.e. $N^{\mathrm{H},\varphi}_L = \dim \mathcal{H}^{\mathrm{H}}_{j,E}$ and $N^{\mathrm{H},\boldsymbol{q}}_L = \dim \nabla\mathcal{H}^{\mathrm{H}}_{j,E}$. Note that $\dim \nabla\mathcal{H}^{\mathrm{H}}_{j,E} = \dim \mathcal{H}^{\mathrm{H}}_{j,E} - 1$, since the kernel of ∇ operator in $\mathcal{H}^{\mathrm{H}}_{j,E}$ contains only the constant polynomial;
- the input dimension N^{H}_0 is the same for both the neural networks and must be sufficient to properly encode the input pair (j, E) into the input vector $\boldsymbol{x}^{\mathrm{H}}_0$. In [15], an input reduction strategy is proposed that encodes the index of the basis function and the vertex coordinates representing the element E in a vector with dimension $N^{\mathrm{H}}_0 = 2(N^{v}_E - 1)$.

Remark 3 (Polygon classification)**.** Since the number of vertices of an element determines the neural network architecture, in particular its input layer, the elements are implicitly grouped into classes according to their number of vertices, and a distinct neural network must be trained for each class. Nonetheless, the number of vertices per element in most meshes is typically bounded, so this assumption is not restrictive and only a limited number of neural networks is required.

Remark 4 (Extension to the three-dimensional case)**.** The main idea behind the NAVEM methods remains the same in both two-dimensional and three-dimensional problems. However, the higher variability of polyhedra introduces some technical issues. In particular, the assumption made in Remark 3 that the number of vertices in each polyhedron is limited and small may become false and, as a consequence, a large number of neural networks should be trained to approximate all the discrete local spaces. To avoid this issue, we propose two strategies.

- Considering a single neural network for all the elements. For this purpose, the input encoding must be large enough to encode all the pairs (j, E), independently of the number of the vertices of the element E. This approach allows training a single neural network, but such a neural network has to learn a more complex pattern.
- On a polyhedral mesh, the restriction of a local VEM function to a face of an element is itself a two-dimensional virtual function, and is therefore not available in closed form: this is precisely the limit that prevents the face integrals from being evaluated exactly, and that forces the standard three-dimensional VEM to resort to face-wise polynomial projections and to a stabilizing term [21]. The techniques developed in this work can be applied to each face of the polyhedra, yielding closed-form approximations of the face traces of the basis functions. Once these are available, the boundary integrals entering the local operators become directly computable, which opens the way to build a stabilization-free discrete bilinear form on each polyhedron [24].

In the following, we denote by C an arbitrary constant independent of the mesh size h, with different meanings in different occurrences.

Let us introduce the following quantities

$$\epsilon^{\mathrm{H},\varphi} = \max_{E\in\mathcal{T}_h} \max_{j=1,\dots,N^{\mathrm{dof}}_E} \epsilon^{\mathrm{H},\varphi}_{j,E} \quad \text{and} \quad \epsilon^{\mathrm{H},\boldsymbol{q}} = \max_{E\in\mathcal{T}_h} \max_{j=1,\dots,N^{\mathrm{dof}}_E} \epsilon^{\mathrm{H},\boldsymbol{q}}_{j,E}, \tag{16}$$

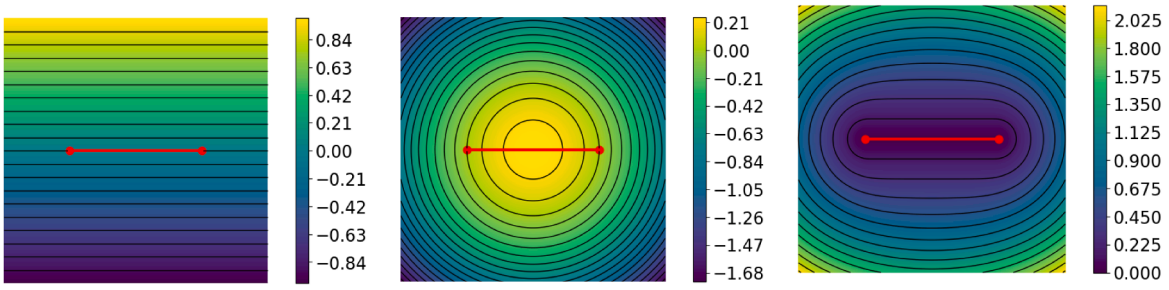


Fig. 1. Left: Distance function (20) to the line lying on the segment of extrema $(-0.5, 0.0)$ and $(0.5, 0.0)$. Center: trimming function defined in (22) related the edge of extrema $(-0.5, 0.0)$ and $(0.5, 0.0)$. Right: the ADF defined in (23) associated with the edge of extrema $(-0.5, 0.0)$ and $(0.5, 0.0)$.

that express the maximum values of the H-NAVEM test losses. In [10], the following a priori error estimate for the H-NAVEM method was proved.

Theorem 1 (Theorem 1 in [10]). *Let $u \in H^2(\Omega)$ be the solution of Problem (2) and let u_h^{H} be the solution of the discrete H-NAVEM problem. Then, the following a priori error bound holds*

$$\|\nabla u - \nabla u_h^{\mathrm{H}}\|_{L^2(\Omega)} \leq C\left(h + h^{-3/2}\epsilon^{\mathrm{H},q} + h^{-2}\epsilon^{\mathrm{H},\varphi}\right). \tag{17}$$

for a constant C independent of h.

We observe that the extra terms $h^{-3/2}\epsilon^{\mathrm{H},q}$ and $h^{-2}\epsilon^{\mathrm{H},\varphi}$ in (17) arise from two different sources of inconsistency. The former arises from errors in the gradients of the basis functions and would be absorbed into the last term if a single neural network was used to approximate both the basis functions and their derivatives, rather than employing two separate networks. The second source of inconsistency stems from the lack of continuity of the basis functions. In the following, we introduce two alternative approaches, B-NAVEM and P-NAVEM, designed to eliminate these inconsistencies.

4. Enforcing continuity across adjacent elements

As mentioned before, the standard H-NAVEM basis functions are no longer continuous across adjacent elements, since Property (ii) in (1) is enforced only approximately by minimizing the loss functions (14) and (15).

In this section, for each input pair (j, E), we show how to define an operator

$$\mathcal{B}_{j,E} : C^0\left(\overline{E}\right) \to C^0\left(\overline{E}\right), \tag{18}$$

which, when applied to the neural network output, can be used to exactly enforce Property (ii) when approximating $\varphi_{j,E}$. The definition of this operator is based on the definition of two auxiliary functions, namely the transfinite interpolant $\psi_{j,E}$, which coincides with $\varphi_{j,E}$ on the boundary of E, and the bubble function ψ_E^0, which vanishes on ∂E. The construction of these two functions will be presented in the following sections.

Given the bubble function ψ_E^0 and the transfinite interpolation $\psi_{j,E}$, we define the operator $\mathcal{B}_{j,E}$ as

$$\left(\mathcal{B}_{j,E}(v)\right)(\boldsymbol{x}) \coloneqq \psi_E^0(\boldsymbol{x})v(\boldsymbol{x}) + \psi_{j,E}(\boldsymbol{x}), \quad \forall \boldsymbol{x} \in E, \quad \forall v \in C^0\left(\overline{E}\right), \tag{19}$$

which satisfies by construction

$$\left(\mathcal{B}_{j,E}(v)\right)(\boldsymbol{x}) = \varphi_{j,E}(\boldsymbol{x}), \quad \forall \boldsymbol{x} \in \partial E, \quad \forall v \in C^0\left(\overline{E}\right).$$

Such an operator will be used in Sections 5 and 6 to devise two alternative approaches that allow us to obtain basis functions that are continuous all across the domain $\overline{\Omega}$.

In the following, the symbol $\|\cdot\|$ denotes the Euclidean norm in $\mathbb{R}^2$. Moreover, we denote by $\{e_1, \ldots, e_{N_E^v}\}$ the set of edges of E, where e_n connects the vertices $\boldsymbol{X}_n$ and $\boldsymbol{X}_{n+1}$ of E, with $\boldsymbol{X}_{N_E^v+1} = \boldsymbol{X}_1$, and we define $\boldsymbol{n}_n$ as the unit outward normal vector to the edge e_n with respect the polygon E.

4.1. A bubble function on a generic polygon

Our goal is to build a bubble function ψ_E^0 that vanishes on ∂E, is strictly positive inside E, and has non-vanishing inward normal derivative on ∂E [25].

For this purpose, we first aim to build a function $\psi_{n,E}^0$, for each $n = 1, \ldots, N_E^v$, that is computable and strictly positive on E and vanishes only on the edge e_n. Different strategies can be adopted to construct these functions, depending on the geometry and shape

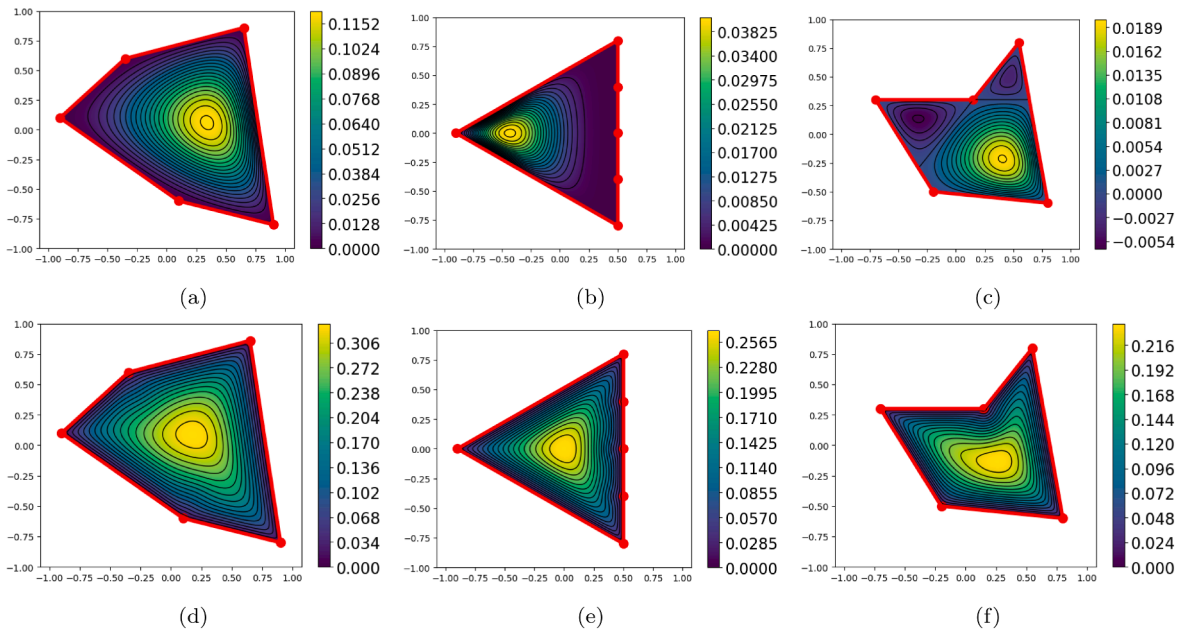


Fig. 2. Different alternatives to define the bubble function ψ_E^0 for a convex polygon (left), a triangle with hanging nodes (center), and a concave polygon (right). The top row refers to definition (21), whereas the bottom row refers to definition (23).

of the polygon E. If the element E is convex, we can define the function $\psi_{n,E}^0$ as the *signed distance function* d_n from the line l_n where the edge e_n lies, i.e.

$$\psi_{n,E}^0(\boldsymbol{x}) \coloneqq d_n(\boldsymbol{x}) = (\boldsymbol{x} - \boldsymbol{X}_n) \cdot \boldsymbol{n}_n, \quad \forall \boldsymbol{x} \in \overline{E} \quad \forall n = 1, \dots, N_E^v, \tag{20}$$

which is strictly positive in E and is zero on the whole l_n. For an illustration of this function, see Fig. 1a. Given these distance functions, we can finally define

$$\psi_E^0(\boldsymbol{x}) = \prod_{n=1}^{N_E^v} \psi_{n,E}^0(\boldsymbol{x}) \quad \forall \boldsymbol{x} \in \overline{E}, \quad \text{with} \quad \psi_{n,E}^0 \coloneqq d_n \quad \forall n = 1, \dots, N_E^v, \tag{21}$$

which vanishes on the boundary of ∂E and is strictly positive inside E. See Fig. 2a for a graphical illustration. This construction ensures that all the derivatives of ψ_E^0 exist and are bounded in $\overline{E}$, although ψ_E^0 may be very small in regions close to many edges of E or it may have zero normal derivative in the presence of aligned edges. For example, see Fig. 2b. If the element E is concave, the line l_n where the signed distance function d_n is zero may cross the polygon, making $\psi_{n,E}^0$ no longer strictly positive inside E as requested (see Fig. 2c).

In these cases, an example of a function that vanishes only on e_n is represented by the *Approximate Distance Function* (ADF) defined in [19]. More specifically, let us define the trimming function t_n^c as

$$t_n^c(\boldsymbol{x}) = \frac{1}{|e_n|}\left[\left(\frac{|e_n|}{2}\right)^2 - \|\boldsymbol{x} - \frac{\boldsymbol{X}_n + \boldsymbol{X}_{n+1}}{2}\|^2\right], \quad \forall \boldsymbol{x} \in \overline{E} \quad \forall n = 1, \dots, N_E^v. \tag{22}$$

Note that $t_n^c \geq 0$ defines a circle of center $\dfrac{\boldsymbol{X}_n + \boldsymbol{X}_{n+1}}{2}$ as shown in Fig. 1b. Now, we can define $\psi_{n,E}^0$ as the ADF related to e_n, i.e.

$$\psi_{n,E}^0 = \sqrt{d_n^2(\boldsymbol{x}) + \left(\frac{\sqrt{(t_n^c(\boldsymbol{x}))^2 + d_n^4(\boldsymbol{x})} - t_n^c(\boldsymbol{x})}{2}\right)^2}, \quad \forall \boldsymbol{x} \in \overline{E} \quad \forall n = 1, \dots, N_E^v. \tag{23}$$

Finally, the bubble function ψ_E^0 can be chosen as the ADF to ∂E, normalized up to order 2, i.e. as

$$\psi_E^0(\boldsymbol{x}) = \frac{1}{\sqrt{\sum_{n=1}^{N_E^v} \dfrac{1}{\left(\psi_{n,E}^0(\boldsymbol{x})\right)^2}}}. \tag{24}$$

With this definition, ψ_E^0 vanishes at the boundary, is strictly positive inside E, and has a unitary inward normal derivative at the boundary, independently of the shape of the polygon E, as requested. Moreover, it is C^2 at all points away from the boundary ∂E, whereas its Laplacian blows up at the vertices of E. See Fig. 2d, e, and f for an illustration of the ADF for different kinds of polygons.

In the following, we always adopt the combination (23)-(24), since it applies to all classes of polygons and it has been numerically observed to provide better accuracy.

Remark 5. In the case of convex polygonal elements, one could employ the standard polynomial bubble function defined in (21). However, since this bubble may attain very small values in large portions of the element, particularly in points $\boldsymbol{z}$ that are close to many edges, the contribution of the neural network output $\mathcal{N}(\boldsymbol{x}_0)$ is strongly damped. As a consequence, learning an effective interior correction becomes difficult, because the product $\psi_E^0(\boldsymbol{z})\mathcal{N}(\boldsymbol{x}_0)$ has only a marginal influence on the final neural approximation $\left(\mathcal{B}_{j,E}(\mathcal{N}(\boldsymbol{x}_0))\right)(\boldsymbol{z})$.

4.2. The transfinite interpolation of $\varphi_{j,E}$

Let us consider a point $\boldsymbol{x} \in \overline{E}$ and let $\Pi_{e_n}\boldsymbol{x}$ be the orthogonal projection of $\boldsymbol{x}$ onto the line l_n where the edge e_n lies. For any $\boldsymbol{z} \in l_n$, let $s_n(\boldsymbol{z})$ be the curvilinear coordinate of $\boldsymbol{z}$ with respect to the origin $\boldsymbol{X}_n$ and the direction $(\boldsymbol{X}_{n+1} - \boldsymbol{X}_n)$, i.e. s_n is a linear function such that $s_n(\boldsymbol{X}_n) = 0$ and $s_n(\boldsymbol{X}_{n+1}) = 1$, whose expression is given by:

$$s_n(\boldsymbol{z}) = \frac{(\boldsymbol{z} - \boldsymbol{X}_n)\cdot(\boldsymbol{X}_{n+1} - \boldsymbol{X}_n)}{\|\boldsymbol{X}_{n+1} - \boldsymbol{X}_n\|^2} \quad \forall \boldsymbol{z} \in l_n \quad \forall n = 1, \dots, N_E^v.$$

Let $\boldsymbol{x}_{n,\perp} := \boldsymbol{x} - \Pi_{e_n}\boldsymbol{x}$ be the component of $\boldsymbol{x}$ orthogonal to the vector $\boldsymbol{X}_{n+1} - \boldsymbol{X}_n$. The scalar product $(\boldsymbol{x} - \boldsymbol{X}_n)\cdot(\boldsymbol{X}_{n+1} - \boldsymbol{X}_n)$ can be written as:

$$\begin{aligned}(\boldsymbol{x} - \boldsymbol{X}_n)\cdot(\boldsymbol{X}_{n+1} - \boldsymbol{X}_n) &= (\boldsymbol{x}_{n,\perp} + \Pi_{e_n}\boldsymbol{x} - \boldsymbol{X}_n)\cdot(\boldsymbol{X}_{n+1} - \boldsymbol{X}_n)\\ &= (\Pi_{e_n}\boldsymbol{x} - \boldsymbol{X}_n)\cdot(\boldsymbol{X}_{n+1} - \boldsymbol{X}_n) + \boldsymbol{x}_{n,\perp}\cdot(\boldsymbol{X}_{n+1} - \boldsymbol{X}_n)\\ &= (\Pi_{e_n}\boldsymbol{x} - \boldsymbol{X}_n)\cdot(\boldsymbol{X}_{n+1} - \boldsymbol{X}_n).\end{aligned}$$

As a consequence, we can define the function $s_n : \overline{E} \to \mathbb{R}$ as

$$s_n(\Pi_{e_n}\boldsymbol{x}) = \frac{(\boldsymbol{x} - \boldsymbol{X}_n)\cdot(\boldsymbol{X}_{n+1} - \boldsymbol{X}_n)}{\|\boldsymbol{X}_{n+1} - \boldsymbol{X}_n\|^2} \quad \forall \boldsymbol{x} \in \overline{E} \quad \forall n = 1, \dots, N_E^v.$$

For each input pair (j, E), any $n = 1, \dots, N_E^v$ and for any $\boldsymbol{x} \in \overline{E}$, we introduce the functions

$$\psi_{n,j,E}(\boldsymbol{x}) = \begin{cases} s_{j-1}(\Pi_{e_{j-1}}\boldsymbol{x}) & \text{if } n = j-1,\\ 1 - s_j(\Pi_{e_j}\boldsymbol{x}) & \text{if } n = j,\\ 0 & \text{otherwise,}\end{cases}$$

which are defined everywhere in $\overline{E}$, are known in a closed form, and satisfy

$$\psi_{n,j,E} = \varphi_{j,E} \quad \text{on} \quad e_n \quad \forall n = 1, \dots, N_E^v.$$

Then, the transfinite interpolation $\psi_{j,E}$ of the function $\varphi_{j,E}$ is defined as [18]

$$\psi_{j,E}(\boldsymbol{x}) = \sum_{n=1}^{N_E^v} \omega_{n,E}(\boldsymbol{x})\psi_{n,j,E}(\boldsymbol{x}), \tag{25}$$

where

$$w_{n,E}(\boldsymbol{x}) = \frac{\prod_{m=1;m\neq n}^{N_E^v}\psi_{m,E}^0(\boldsymbol{x})}{\sum_{r=1}^{N_E^v}\prod_{m=1;m\neq r}^{N_E^v}\psi_{m,E}^0(\boldsymbol{x})},$$

and the functions $\psi_{n,E}^0$ are defined in (23). Given the properties of these functions, it is immediate to check that

$$\psi_{j,E} = \varphi_{j,E} \quad \text{on } \partial E.$$

5. The B-NAVEM formulation

The virtual basis functions $\{\varphi_{j,E}\}_{j=1}^{N_E^{\text{dof}}}$ are defined as the solutions of the elemental Laplace problems (1). As discussed in the previous sections, since the map associating each input pair (j, E) with the approximation function $\varphi_{j,E}^{\mathcal{NN}}$ is highly nonlinear, Problem (1) can be efficiently addressed using neural networks of the form (8). Among the most widely used neural network-based PDE solvers, Physics-Informed Neural Networks (PINN), originally introduced in [26], have recently gained a lot of attention.

Unlike the H-NAVEM approach, where the neural network output represents the vector of coefficients with respect to the harmonic basis for $\mathcal{H}^{\mathrm{H}}_{j,E}$, in PINNs the output directly represents the value of the function at a given point $\boldsymbol{z}$ inside the domain E. The core idea of the B-NAVEM method is to employ a PINN to learn the following nonlinear map:

$$(\boldsymbol{z}, j, E) \mapsto \varphi^{\mathcal{NN}}_{j,E}(\boldsymbol{z}). \tag{26}$$

Thus, the input and output dimensions in the B-NAVEM method are $N^{\mathrm{B}}_0 = 2N^v_E$ and $N^{\mathrm{B}}_L = 1$, respectively. In particular, the input vector $\boldsymbol{x}^{\mathrm{B}}_0$ consists of the concatenation of the evaluation point $\boldsymbol{z} \in \mathbb{R}^2$ and the H-NAVEM encoding $\boldsymbol{x}^{\mathrm{H}}_0 \in \mathbb{R}^{2(N^v_E-1)}$ of the pair (j, E). More precisely, we set

$$\boldsymbol{x}^{\mathrm{B}}_0 = [\boldsymbol{z};\, \boldsymbol{x}^{\mathrm{H}}_0] \in \mathbb{R}^{N^{\mathrm{B}}_0}.$$

Given the operator $\mathcal{B}_{j,E}$ defined in (19), for an input $\boldsymbol{x}^{\mathrm{B}}_0$ encoding the triplet $(\boldsymbol{z}, j, E)$, the B-NAVEM basis functions are defined as

$$\varphi^{\mathrm{B}}_{j,E}(\boldsymbol{z}) \coloneqq \mathcal{B}_{j,E}\big(\mathcal{N}^{\mathrm{B}}(\boldsymbol{x}^{\mathrm{B}}_0)\big)(\boldsymbol{z}) = \psi^0_E(\boldsymbol{z})\,\mathcal{N}^{\mathrm{B}}(\boldsymbol{x}^{\mathrm{B}}_0) + \psi_{j,E}(\boldsymbol{z}),$$

where $\mathcal{N}^{\mathrm{B}}(\boldsymbol{x}^{\mathrm{B}}_0) \in \mathbb{R}$ denotes the scalar output of the PINN. The function $\varphi^{\mathrm{B}}_{j,E}$ automatically satisfies Property (ii) in (1) for any choice of the trainable weights, thanks to the usage of the operator $\mathcal{B}_{j,E}$. Therefore, the training procedure only needs to enforce Property (i). Accordingly, the loss term is defined as the PDE residual, i.e.

$$\epsilon^{\mathrm{B}}_{j,E} = \|\Delta\varphi^{\mathrm{B}}_{j,E}\|_{L^2(E)}. \tag{27}$$

In the B-NAVEM approach, the Laplacian $\Delta\varphi^{\mathrm{B}}_{j,E}(\boldsymbol{z})$, as well as the gradient $\nabla\varphi^{\mathrm{B}}_{j,E}(\boldsymbol{z})$, can be computed via automatic differentiation [27]. This naturally leads to the definition of the discrete B-NAVEM gradient as

$$\boldsymbol{q}^{\mathrm{B}}_{j,E}(\boldsymbol{z}) = \nabla\varphi^{\mathrm{B}}_{j,E}(\boldsymbol{z}).$$

We now discuss the theoretical properties of the presented method.

Theorem 2. *For a given pair (j, E), let $\varphi_{j,E}$ and $\varphi^{\mathrm{B}}_{j,E}$ be the VEM basis function and its B-NAVEM approximation, respectively. Then*

$$\|\nabla\varphi^{\mathrm{B}}_{j,E} - \nabla\varphi_{j,E}\|_{L^2(E)} \le Ch_E\epsilon^{\mathrm{B}}_{j,E},$$

for a given constant C independent of h_E.

Proof. Since $\varphi_{j,E}$ and $\varphi^{\mathrm{B}}_{j,E}$ coincide on ∂E, while $\varphi_{j,E}$ is harmonic and $\varphi^{\mathrm{B}}_{j,E}$ is only approximately harmonic, the difference

$$w_{j,E} \coloneqq \varphi^{\mathrm{B}}_{j,E} - \varphi_{j,E} \in H^1_0(E),$$

satisfies the following problem:

$$\begin{cases} -\Delta w_{j,E} = -\Delta\varphi^{\mathrm{B}}_{j,E} \coloneqq f^{\mathrm{B}}_{j,E} & \text{in } E, \\ w_{j,E} = 0 & \text{on } \partial E. \end{cases}$$

Using the standard integration by parts formula, we observe that

$$\int_E f^{\mathrm{B}}_{j,E}\, w_{j,E} = -\int_E \Delta w_{j,E}\, w_{j,E} = \int_E \nabla w_{j,E}\cdot\nabla w_{j,E} = \|\nabla w_{j,E}\|^2_{L^2(E)}.$$

As a consequence, by using the first Poincaré inequality [28], we obtain

$$\begin{aligned} \|\nabla w_{j,E}\|^2_{L^2(E)} = \int_E f^{\mathrm{B}}_{j,E}\, w_{j,E} &\le \|f^{\mathrm{B}}_{j,E}\|_{L^2(E)}\|w_{j,E}\|_{L^2(E)} \\ &\le Ch_E\|f^{\mathrm{B}}_{j,E}\|_{L^2(E)}\|\nabla w_{j,E}\|_{L^2(E)} \\ &\le Ch_E\|\Delta\varphi^{\mathrm{B}}_{j,E}\|_{L^2(E)}\|\nabla w_{j,E}\|_{L^2(E)} \\ &\le Ch_E\epsilon^{\mathrm{B}}_{j,E}\|\nabla w_{j,E}\|_{L^2(E)}, \end{aligned}$$

where $\epsilon^{\mathrm{B}}_{j,E}$ is defined in (27). This concludes the proof. □

Let us introduce the global B-NAVEM test loss

$$\epsilon^{\mathrm{B}} = \max_{E\in\mathcal{T}_h}\max_{j=1,\dots,N^{\mathrm{dof}}_E}\epsilon^{\mathrm{B}}_{j,E},$$

the following a priori error estimate can be proved for the B-NAVEM method.

Theorem 3. *Let $u \in H^2(\Omega)\cap C^0\left(\overline{\Omega}\right)$ be the exact solution of Problem (2) and u^{B}_h be the B-NAVEM discrete solution. The following a priori error estimate holds*

$$\|\nabla u - \nabla u^{\mathrm{B}}_h\|_{L^2(\Omega)} \le C(h + h^{-1}\epsilon^{\mathrm{B}}),$$

for a given constant C independent of h.

Proof. Let us introduce the VEM and B-NAVEM interpolation operators I^{V} and I^{B} defined as usual as:

$$I^{\mathrm{V}}(u) = \sum_{i=1}^{N^{\mathrm{dof}}} u_i \, \varphi_i, \qquad I^{\mathrm{B}}(u) = \sum_{i=1}^{N^{\mathrm{dof}}} u_i \, \varphi_i^{\mathrm{B}}, \tag{28}$$

where $\boldsymbol{u} = (u_i)_{i=1}^{N^{\mathrm{dof}}}$ is the vector collecting the nodal values of the function u at the internal vertices of the tessellation.

Thanks to the coercivity and continuity of the involved linear and bilinear forms, the error $\|\nabla u - \nabla u_h^{\mathrm{B}}\|_{L^2(\Omega)}$ can be initially bounded by Cea's lemma [29]:

$$\|\nabla u - \nabla u_h^{\mathrm{B}}\|_{L^2(\Omega)} \leq C \inf_{v_h^{\mathrm{B}} \in V_h^{\mathrm{B}}} \|\nabla u - \nabla v_h^{\mathrm{B}}\|_{L^2(\Omega)}. \tag{29}$$

Now, let us bound the best approximation error as

$$\begin{aligned}
\inf_{v_h^{\mathrm{B}} \in V_h^{\mathrm{B}}} \|\nabla u - \nabla v_h^{\mathrm{B}}\|_{L^2(\Omega)} &\leq \|\nabla u - \nabla I^{\mathrm{B}} u\|_{L^2(\Omega)} \leq \|\nabla u - \nabla I^{\mathrm{V}} u\|_{L^2(\Omega)} + \|\nabla I^{\mathrm{V}} u - \nabla I^{\mathrm{B}} u\|_{L^2(\Omega)} \\
&\leq Ch|u|_{H^2(\Omega)} + \| \sum_{i=1}^{N^{\mathrm{dof}}} u_i \big(\nabla \varphi_i - \nabla \varphi_i^{\mathrm{B}}\big) \|_{L^2(\Omega)} \\
&\leq Ch|u|_{H^2(\Omega)} + \sum_{E \in \mathcal{T}_h} \sum_{j=1}^{N_E^{\mathrm{dof}}} |u_j| \|\nabla \varphi_{j,E} - \nabla \varphi_{j,E}^{\mathrm{B}}\|_{L^2(E)} \\
&\leq Ch|u|_{H^2(\Omega)} + \sum_{E \in \mathcal{T}_h} \sum_{j=1}^{N_E^{\mathrm{dof}}} |u_j| h_E \epsilon_{j,E}^{\mathrm{B}},
\end{aligned} \tag{30}$$

where we have exploited Theorem 2.

Exploiting the assumption of a quasi-uniform mesh, the maximum number of elements sharing a vertex is independent of h and the number of degrees of freedom scales as $N^{\mathrm{dof}} \approx h^{-2}$. Under this assumption, the following bound holds [30, Lemma 4]:

$$\|\boldsymbol{u}\|_2 \leq Ch^{-1} \|\nabla u\|_{L^2(\Omega)}. \tag{31}$$

As a consequence, the last term in (30) can be bounded as:

$$\sum_{E \in \mathcal{T}_h} \sum_{j=1}^{N_E^{\mathrm{dof}}} |u_j| h_E \epsilon_{j,E}^{\mathrm{B}} \leq Ch \, \epsilon^{\mathrm{B}} \sqrt{N^{\mathrm{dof}}} \|\boldsymbol{u}\|_2 \leq C \epsilon^{\mathrm{B}} \|\boldsymbol{u}\|_2 \leq C \epsilon^{\mathrm{B}} h^{-1} \|\nabla u\|_{L^2(\Omega)}. \tag{32}$$

Combining (29), (30), and (32) leads to the final estimate

$$\begin{aligned}
\|\nabla u - \nabla u_h^{\mathrm{B}}\|_{L^2(\Omega)} &\leq C\big(h|u|_{H^2(\Omega)} + \epsilon^{\mathrm{B}} h^{-1} \|\nabla u\|_{L^2(\Omega)}\big) \\
&\leq C(h + h^{-1}\epsilon^{\mathrm{B}}) \|u\|_{H^2(\Omega)}.
\end{aligned}$$

□

6. The P-NAVEM formulation

As shown in the previous sections, the NAVEM approach allows us to exactly enforce Property (i) in (1), while Property (ii) is satisfied only up to the NAVEM accuracy. Conversely, in the B-NAVEM approach, Property (ii) is exactly enforced, whereas Property (i) depends on the approximation properties of the neural network. Nonetheless, in both cases, we retrieve

$$\mathbb{P}_1(E) \not\subset V_h^*(E) \quad \forall * \in \{\mathrm{H}, \mathrm{B}\}, \tag{33}$$

where $V_h^{\mathrm{H}}(E)$ and $V_h^{\mathrm{B}}(E)$ represent the local H-NAVEM and B-NAVEM spaces, respectively, and are defined as dictated in (6). More precisely, the capability of H-NAVEM or B-NAVEM to reproduce polynomial functions depends on the accuracy of the underlying neural networks. The lack of exact polynomial inclusion may therefore limit the convergence properties of both methods.

The theoretical VEM analysis suggests introducing a linearly independent set of basis functions $\{\varphi_{j,E}^{\mathcal{NN}}\}_{j=1}^{N_E^v}$ satisfying Property (ii) in (1), which guarantees C^0-conformity, together with the two following additional properties: $\forall \boldsymbol{x} = (x_1, x_2) \in \overline{E}$,

P.1 *Partition of unity property*:

$$\sum_{j=1}^{N_E^{\mathrm{dof}}} \varphi_{j,E}^{\mathcal{NN}}(\boldsymbol{x}) = 1,$$

P.2 *Linear reproduction property*:

$$\sum_{j=1}^{N_E^{\mathrm{dof}}} (\boldsymbol{X}_j)_1 \; \varphi_{j,E}^{\mathcal{NN}}(\boldsymbol{x}) = x_1, \qquad \sum_{j=1}^{N_E^{\mathrm{dof}}} (\boldsymbol{X}_j)_2 \; \varphi_{j,E}^{\mathcal{NN}}(\boldsymbol{x}) = x_2,$$

where $((\boldsymbol{X}_j)_1, (\boldsymbol{X}_j)_2)$ are the coordinates of the jth vertex of E.

The two Properties 1 and 2 imply the exact linear polynomial reproducibility. This last condition is sufficient to ensure the desired convergence rates of the method, without requiring the basis functions to be harmonic as in the virtual element framework. Motivated by these considerations, we aim to construct a discrete space whose basis functions satisfy, as accurately as possible, Properties (ii) in (1), 1, and 2. We refer to this approach as P-NAVEM, emphasizing its focus on polynomial reproducibility.

Property (ii) in (1) can be imposed exactly by designing a neural network that learns the nonlinear mapping (26) and by applying the operator $\mathcal{B}_{j,E}$ to the P-NAVEM network output $\mathcal{N}^{\mathrm{P}}(\boldsymbol{x}_0^{\mathrm{P}})$, as done in the B-NAVEM approach. Since the P-NAVEM network input encodes the same geometric information as in B-NAVEM, we have $\boldsymbol{x}_0^{\mathrm{P}} = \boldsymbol{x}_0^{\mathrm{B}}$ and $N_0^{\mathrm{P}} = N_0^{\mathrm{B}}$. Moreover, as in B-NAVEM, we set $N_L^{\mathrm{P}} = 1$ and define

$$\varphi_{j,E}^{\mathrm{P}}(\boldsymbol{z}) \coloneqq \mathcal{B}_{j,E}\left(\mathcal{N}^{\mathrm{P}}(\boldsymbol{x}_0^{\mathrm{P}})\right)(\boldsymbol{z}) = \psi_E^0(\boldsymbol{z})\,\mathcal{N}^{\mathrm{P}}(\boldsymbol{x}_0^{\mathrm{P}}) + \psi_{j,E}(\boldsymbol{z}). \tag{34}$$

We observe that, independently of the method $*\in \{\mathrm{H},\mathrm{B},\mathrm{P}\}$, Property 1 can always be enforced exactly by learning (for instance) only the first $N_E^v - 1$ basis functions and then defining the last one as

$$\varphi_{N_E^v,E}^{*} = 1 - \sum_{j=1}^{N_E^v-1} \varphi_{j,E}^{*}.$$

Thus, in the P-NAVEM approach, to approximately enforce 2, we can thus the train neural network $\mathcal{N}^{\mathrm{P}}$ to minimize

$$\epsilon_E^{\mathrm{P},\boldsymbol{\varphi}} = \epsilon_E^{p_1} + \epsilon_E^{p_2}, \tag{35}$$

where

$$\epsilon_{j,E}^{p_i} = \| \sum_{j=1}^{N_E^v} p_i(\boldsymbol{X}_j)\varphi_{j,E}^{\mathrm{P}} - p_i \|_{L^2(E)}, \quad \forall i \in \{1,2\},$$

and $\{p_\alpha\}_{\alpha=0}^2$ are the set of scaled monomials. However, as already observed for the H-NAVEM method in [15], training the neural network by directly learning the basis functions $\varphi_{j,E}^{\mathrm{P}}$ through a loss contribution that depends exclusively on the basis functions and does not involve their gradients typically produces smooth basis functions but highly oscillatory gradients (see Remark 2). To mitigate this issue, rather than minimizing a loss functional that depends solely on the basis functions, we introduce a loss contribution that directly involves their gradients, thereby controlling their oscillatory behaviour:

$$\epsilon_E^{\mathrm{P},\boldsymbol{q}} = \epsilon_E^{\nabla p_1} + \epsilon_E^{\nabla p_2}, \tag{36}$$

where

$$\epsilon_E^{\nabla p_i} = \| \sum_{j=1}^{N_E^v} p_i(\boldsymbol{X}_j)\nabla\varphi_{j,E}^{\mathrm{P}} - \nabla p_i \|_{L^2(E)}, \quad \forall i \in \{1,2\}.$$

To compare these two training strategies, we consider two independent neural networks, each optimized with a different objective function. The first network is trained by minimizing the loss (35), which measures the approximation error of the basis functions, whereas the second one is trained by minimizing (36), which directly controls the error in their gradients. For both networks, we monitor the evolution of the two error quantities $\epsilon_E^{\mathrm{P},\boldsymbol{\varphi}}$ and $\epsilon_E^{\mathrm{P},\boldsymbol{q}}$ during the training process, and the results are reported in Fig. 3. We observe that the error $\epsilon_E^{\mathrm{P},\boldsymbol{\varphi}}$ exhibits a similar behaviour for the two training strategies. Conversely, $\epsilon_E^{\mathrm{P},\boldsymbol{q}}$ decreases significantly faster when the network is trained using the gradient-based loss (36). These results show that minimizing either (35) or (36) leads to a comparable approximation of the target polynomials. However, training with (36) provides a more accurate representation of their gradients.

We remark that, in the H-NAVEM method, two separate neural networks are employed to approximate the virtual basis functions and their gradients, respectively. This choice is necessary because minimizing only the gradient-based loss (15) does not preserve the correct VEM scaling of the basis functions [15]. In contrast, the P-NAVEM method allows us to employ a single neural network trained by minimizing the loss term (36), since the use of the $\mathcal{B}_{j,E}$ operator ensures that the basis functions satisfy the correct VEM scaling, thus allowing the gradient-based loss to be minimized without introducing an additional network.

In order to prove the convergence properties of the method, let us define the following quantity

$$\epsilon^{\mathrm{P},\boldsymbol{q}} = \max_{E\in\mathcal{T}_h} \epsilon_E^{\mathrm{P},\boldsymbol{q}}, \tag{37}$$

which defines the maximum P-NAVEM test loss.

Moreover, let us introduce the P-NAVEM interpolator

$$I^{\mathrm{P}}(u) = \sum_{i=1}^{N^{\mathrm{dof}}} u_i\,\varphi_i^{\mathrm{P}},$$

where $\boldsymbol{w} = (w_i)_{i=1}^{N^{\mathrm{dof}}}$ is the vector collecting the nodal values of the function $w \in C^0\left(\overline{\Omega}\right) \cap H^1(\Omega)$ at the internal vertices of the tessellation. We introduce the following preliminary lemmas.

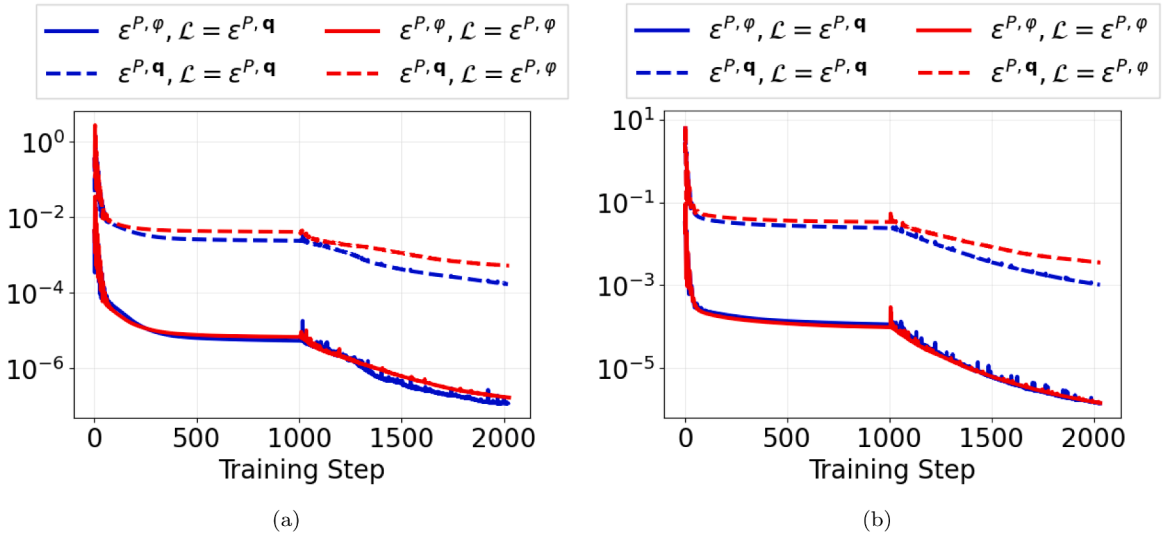


Fig. 3. Evolution of the quantities $\epsilon_E^{\mathrm{P},\varphi}$ (solid lines) and $\epsilon_E^{\mathrm{P},q}$ (dashed lines) during the training. In red, we report the behaviour of the neural network trained by minimizing the loss $\mathcal{L} = \epsilon_E^{\mathrm{P},\varphi}$. In blue, we report the behaviour of the neural network trained by minimizing the loss $\mathcal{L} = \epsilon_E^{\mathrm{P},q}$. In both cases, the ADAM optimizer is used for the first 1000 training steps, whereas the BFGS optimizer is adopted for the remaining steps. Left: The training dataset consists of 100 convex quadrilaterals. Right: The training dataset consists of 100 concave quadrilaterals. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Lemma 1. *For a given polynomial $q \in \mathbb{P}_1(E)$, it holds*

$$\|\nabla I^{\mathrm{P}} q - \nabla q\|_{L^2(E)} \leq C h_E^{-1} \epsilon_E^{\mathrm{P},q} \|q\|_{L^2(E)}, \tag{38}$$

for a given constant C independent of h.

Proof. For each polynomial $q = \sum_{\alpha=0}^{2} c_\alpha p_\alpha \in \mathbb{P}_1(E)$, we have

$$\begin{aligned}\nabla I^{\mathrm{P}} q - \nabla q &= c_0 \sum_{j=1}^{N_E^{\mathrm{dof}}} \nabla \varphi_{j,E}^{\mathrm{P}} + \sum_{\alpha=1}^{2} c_\alpha \left(\sum_{j=1}^{N_E^{\mathrm{dof}}} p_\alpha(\boldsymbol{X}_j) \nabla \varphi_{j,E}^{\mathrm{P}} - \nabla p_\alpha \right) \\ &= \sum_{\alpha=1}^{2} c_\alpha \left(\sum_{j=1}^{N_E^{\mathrm{dof}}} p_\alpha(\boldsymbol{X}_j) \nabla \varphi_{j,E}^{\mathrm{P}} - \nabla p_\alpha \right),\end{aligned}$$

due to the exact partition of unity property 1. Since $\{p_\alpha\}_{\alpha=0}^{2}$ are the scaled monomials, by using Lemma 4.1 in [31], we get

$$\|\nabla I^{\mathrm{P}} q - \nabla q\|_{L^2(E)} \leq C h_E^{-1} \epsilon_E^{\mathrm{P},q} \|q\|_{L^2(E)}, \qquad \forall q \in \mathbb{P}_1(E). \tag{39}$$

□

Lemma 2. *For a function $w \in C^0\left(\overline{E}\right) \cap H^2(E)$, it holds*

$$\|\nabla I^{\mathrm{P}} w\|_{L^2(E)} \leq C(h_E |w|_{H^2(E)} + h_E^{-1} \|w\|_{L^2(E)}), \tag{40}$$

for a constant C independent of the mesh size h.

Proof. Given a generic function $w \in C^0\left(\overline{E}\right) \cap H^2(E)$, whose vector of degrees of freedom is $\boldsymbol{w} = (w_j)_{j=1}^{N_E^{\mathrm{dof}}}$, we have

$$\begin{aligned}\|\nabla I^{\mathrm{P}} w\|_{L^2(E)} &= \| \sum_{j=1}^{N_E^{\mathrm{dof}}} w_j \nabla \varphi_{j,E}^{\mathrm{P}} \|_{L^2(E)} \leq \left(\max_j |w_j| \right) \sum_{j=1}^{N_E^{\mathrm{dof}}} \|\nabla \varphi_{j,E}^{\mathrm{P}}\|_{L^2(E)} \\ &\leq C \, \|w\|_{L^\infty(E)},\end{aligned} \tag{41}$$

where we use the stability bound $\sum_j \|\nabla \varphi_{j,E}^{\mathrm{P}}\|_{L^2(E)} \leq C$, with C independent of h. Such an assumption is reasonable because of the regularization term added to the loss function during training, which guides the optimization process toward functions with small oscillations.

Under mesh Assumption 1, the element is star-shaped with respect to a ball B. In the following, we proceed as in the proof of Lemma 4.3.4 in [32]. In particular, we observe that

$$\|w\|_{L^\infty(E)} \leq \|w - Q^2 w\|_{L^\infty(E)} + \|Q^2 w\|_{L^\infty(E)},$$

where $Q^2 w$ is the Taylor polynomial of degree 2 averaged over the ball B. Using Proposition 4.3.2 in [32], we have

$$\|w - Q^2 w\|_{L^\infty(E)} \leq C h_E |w|_{H^2(E)}.$$

Moreover, following standard scaling arguments (see Remark 1.37 in [33]), we obtain

$$\|Q^2 w\|_{L^\infty(E)} \leq C h_E^{-1} \|w\|_{L^2(E)}.$$

Thus, we get

$$\|w\|_{L^\infty(E)} \leq C(h_E |w|_{H^2(E)} + h_E^{-1} \|w\|_{L^2(E)}), \tag{42}$$

which concludes the proof. □

Theorem 4. *Let $u \in H^2(\Omega) \cap C^0\left(\overline{\Omega}\right)$ be the exact solution of Problem (2) and u_h^{P} be the P-NAVEM discrete solution. The following a priori error estimate holds*

$$\|\nabla u - \nabla u_h^{\mathrm{P}}\|_{L^2(\Omega)} \leq C(h + h^{-1}\epsilon^{\mathrm{P},q}),$$

for a constant C independent of h.

Proof. By Céa Lemma, choosing $v_h^{\mathrm{P}} = I^{\mathrm{P}} u \in V_h^{\mathrm{P}}$, we get

$$\|\nabla u - \nabla u_h^{\mathrm{P}}\|_{L^2(\Omega)} \leq C \inf_{v_h^{\mathrm{P}} \in V_h^{\mathrm{P}}} \|\nabla u - \nabla v_h^{\mathrm{P}}\|_{L^2(\Omega)} \leq C \|\nabla u - \nabla I^{\mathrm{P}} u\|_{L^2(\Omega)}, \tag{43}$$

and, since $\|\cdot\|^2_{L^2(\Omega)} = \sum_{E \in \mathcal{T}_h} \|\cdot\|^2_{L^2(E)}$, it suffices to estimate the local contributions. Introducing the L^2-projection Π_1^E onto $\mathbb{P}_1(E)$, the triangle inequality yields

$$\|\nabla u - \nabla I^{\mathrm{P}} u\|_{L^2(E)} \leq \|\nabla u - \nabla \Pi_1^E u\|_{L^2(E)} + \|\nabla \Pi_1^E u - \nabla I^{\mathrm{P}} \Pi_1^E u\|_{L^2(E)} + \|\nabla I^{\mathrm{P}}(\Pi_1^E u - u)\|_{L^2(E)}. \tag{44}$$

The first term is bounded by the Bramble-Hilbert Theorem [32]:

$$\|\nabla u - \nabla \Pi_1^E u\|_{L^2(E)} \leq C\, h_E\, |u|_{H^2(E)}. \tag{45}$$

Concerning the second term, by Lemma 1 and the L^2-stability of Π_1^E,

$$\|\nabla \Pi_1^E u - \nabla I^{\mathrm{P}} \Pi_1^E u\|_{L^2(E)} \leq C h_E^{-1} \epsilon_E^{\mathrm{P},q} \|\Pi_1^E u\|_{L^2(E)} \leq C h_E^{-1} \epsilon_E^{\mathrm{P},q} \|u\|_{L^2(E)} \leq C h_E^{-1} \epsilon_E^{\mathrm{P},q} \|u\|_{H^2(E)}. \tag{46}$$

Finally, concerning the last term in (44), by using Lemma 2 with $w = \Pi_1^E u - u$ and the Bramble-Hilbert theorem, we obtain

$$\|\nabla I^{\mathrm{P}}(\Pi_1^E u - u)\|_{L^2(E)} \leq C \left(h_E\, |\Pi_1^E u - u|_{H^2(E)} + h_E^{-1} \|\Pi_1^E u - u\|_{L^2(E)}\right) \leq C h_E |u|_{H^2(E)}. \tag{47}$$

Combining (45), (46) and (47),

$$\|\nabla u - \nabla I^{\mathrm{P}} u\|_{L^2(E)} \leq C\left(h_E + h_E^{-1} \epsilon_E^{\mathrm{P},q}\right) \|u\|_{H^2(E)}. \tag{48}$$

Finally, by summing the elemental contributions and by considering a quasi-uniform mesh, we obtain

$$\begin{aligned}
\|\nabla u - \nabla I^{\mathrm{P}} u\|^2_{L^2(\Omega)} &= \sum_{E \in \mathcal{T}_h} \|\nabla u - \nabla I^{\mathrm{P}} u\|^2_{L^2(E)} \\
&\leq C \sum_{E \in \mathcal{T}_h} \left(h_E + h_E^{-1} \epsilon_E^{\mathrm{P},q}\right)^2 \|u\|^2_{H^2(E)} \\
&\leq C \left(h + h^{-1} \epsilon^{\mathrm{P},q}\right)^2 \|u\|^2_{H^2(\Omega)}.
\end{aligned}$$

This concludes the proof. □

Remark 6 (Higher-order NAVEM basis functions)**.** Following [34], the local VEM space $V_{h,k}^V(E)$ of order $k \geq 1$ is given by

$$V_{h,k}^{\mathrm{V}}(E) := \Big\{ v \in H^1(E) : (i)\ \Delta v \in \mathbb{P}_{k-2}(E) \\ (ii)\ v_{|e} \in \mathbb{P}_k(e)\, \forall e \subset \partial E \text{ and } v \in C^0(\partial E) \Big\}.$$

A generic function $v \in V_{h,k}^{\mathrm{V}}(E)$ is therefore described by a Poisson problem with given Dirichlet boundary data and a given source term. This elemental PDE can be solved by a straightforward higher-order extension of the H-NAVEM and B-NAVEM approaches. See Remark 4 in [15] for further details. The P-NAVEM, instead, can directly minimize the distance from the polynomial space $\mathbb{P}_k(E)$ by adding the corresponding terms in (35).

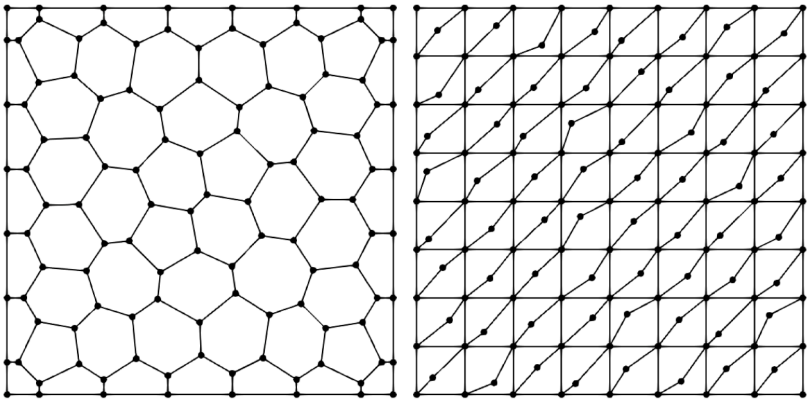


Fig. 4. Tests 1, 2, and 3: Second mesh in the two considered families of meshes used in the test phase. Left: Voronoi. Right: Convex-Concave.

7. Numerical results

In this section, we present a series of numerical experiments aimed at comparing the proposed techniques for approximating basis functions on general polygonal meshes.

Specifically, three numerical tests are conducted. The first experiment evaluates the training and generalization performance of the neural networks across different classes of polygonal elements. The second experiment compares the accuracy and computational efficiency of the corresponding numerical methods when applied to solve a diffusion–advection–reaction problem. Finally, the third experiment highlights the advantages of the proposed approaches over the standard virtual element method, particularly in terms of eliminating projection and stabilization operators. For details on the virtual element discretization employed in this manuscript, we refer the reader to [7,35].

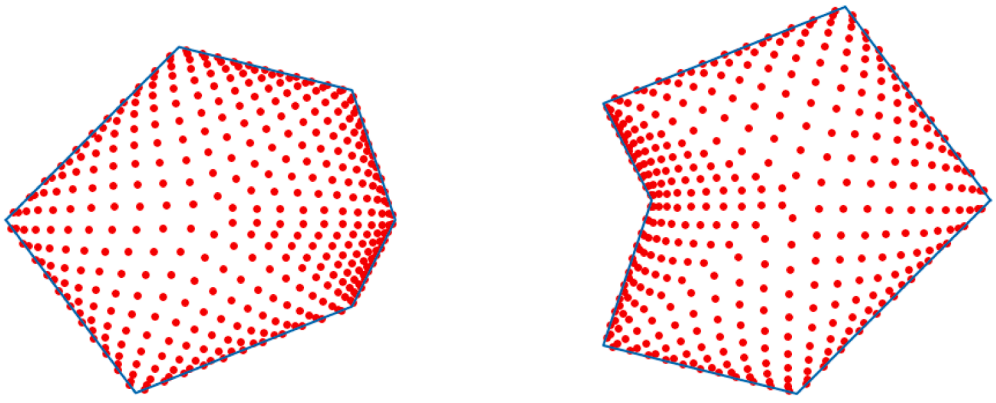
7.1. Test 1: The neural networks and the training phase

As highlighted in Remark 3, we have to train several neural networks to predict the basis functions, one for each class of polygons we consider. Since neural networks require a fixed input dimension and this dimension depends on the number of vertices, we have to train neural networks for each distinct class corresponding to a value of $N_E^v \geq 4$ appearing in our test mesh. The case $N_E^v = 3$ does not require a neural network, as the virtual element method for the lowest order reduces to the finite element method over triangles, whose basis functions are polynomials known in closed form. Therefore, the piecewise linear finite element basis functions are also used in the NAVEM framework in the case of triangles.

In all the numerical tests, we consider two sets of four meshes: a family of Voronoi meshes and a family of quadrilateral meshes that comprises both concave and convex quadrilaterals. The former is generated through the MATLAB library mVEM [36], while the latter is generated by randomly perturbing the vertices coordinates of a related family of structured Convex-Concave meshes. The second mesh for each family is shown in Fig. 4. As it can be observed from Fig. 4b, the strategy followed to build the Convex-Concave family $\{\mathcal{T}_{h,i}^{Q}\}_{i=1}^{4}$ can produce hanging nodes. In these meshes, the number of vertices is at most 7. To further improve the accuracy of the neural network approximations, we train separate networks for convex and concave quadrilaterals. In particular, we adopt the model trained for concave elements in the case of a quadrilateral that has the shape of a triangle and has one hanging node. Consequently, for each model, we train five neural networks corresponding to the following classes of polygons:

1. convex quadrilaterals;
2. concave quadrilaterals;
3. convex pentagons;
4. convex hexagons;
5. convex heptagons.

For the first two classes, thanks to the limited geometric variability of quadrilateral elements, we generate synthetic datasets consisting of 1000 randomly generated convex and concave quadrilaterals, respectively, by using the Python library polygenerator. For the remaining classes, instead, we sample elements with the prescribed number of vertices from training Voronoi meshes generated using the same algorithm adopted for the test meshes. Specifically, we obtain 1000 elements for convex pentagons and hexagons, and about 300 elements for heptagons, since they are rarer in Voronoi meshes. This strategy ensures that the training polygons closely resemble those encountered in the test phase, thereby enabling the networks to generalize effectively to unseen elements. We remark that these datasets (and the corresponding H-NAVEM networks) are the same employed to perform the simulation in [10] on a different test family of Voronoi meshes.



(a) (b)

Fig. 5. Points distribution generated by using the Algorithm 1 on general polygons.

In particular, we emphasize that, once the neural networks have been trained, the resulting approximation space can be reused to solve a broad class of differential problems, independently of the specific PDE, physical coefficients, source terms, or boundary conditions. This is demonstrated by the numerical experiments reported in the following sections, where the same networks are employed for solving different PDEs, and further confirmed by the fact that the H-NAVEM neural networks used therein are the same employed to solve the elasticity problems in [10]. Moreover, the same trained networks can be used on different meshes: the availability of networks for the different classes of polygonal elements allows the construction of basis functions on arbitrary meshes composed of elements belonging to the same classes (sharing the same number of vertices). The convergence properties of the resulting numerical method will naturally depend on the accuracy of the neural networks on the polygonal elements forming the mesh, according to the approximation estimates established for the corresponding spaces. Therefore, although the offline training may be computationally demanding, its cost is amortized, since it can be used to solve a large number of problems and on different meshes.

Algorithm 1: Algorithm designed to select points inside the triangle of vertices $A = (0,0)$, $B = (1,0)$, $C = (0,1)$ to approximate loss integrals.

Data: $N \geq 0$, $\varepsilon > 0$
Result: List of $(N+1)(N+2)/2$ points

points $\leftarrow$ [ ];
for $0 \leq i \leq N$ **do**
  for $0 \leq j \leq N - i$ **do**
    $k \leftarrow N - i - j$;
    $x_0 \leftarrow \dfrac{i + 0.5}{N + 1.5}$;
    $y_0 \leftarrow \dfrac{j + 0.5}{N + 1.5}$;
    $z_0 \leftarrow \dfrac{k + 0.5}{N + 1.5}$;
    $z \leftarrow 1 - \cos\left(\dfrac{\pi}{2} z_0\right)$; /* biased version of z to accumulate points close to edge $\overline{BC}$ */
    $s \leftarrow \dfrac{1 - z}{1 - z_0 + \varepsilon}$; /* scaling factor */
    new_point $\leftarrow (s\,x_0, s\,y_0)$; /* define the new point */
    points.insert(new_point) ; /* add the new point to the list */
  end
end
return points

For each class of polygons, we train our three models, namely H-NAVEM, B-NAVEM, and P-NAVEM, by minimizing the related losses on the training data.

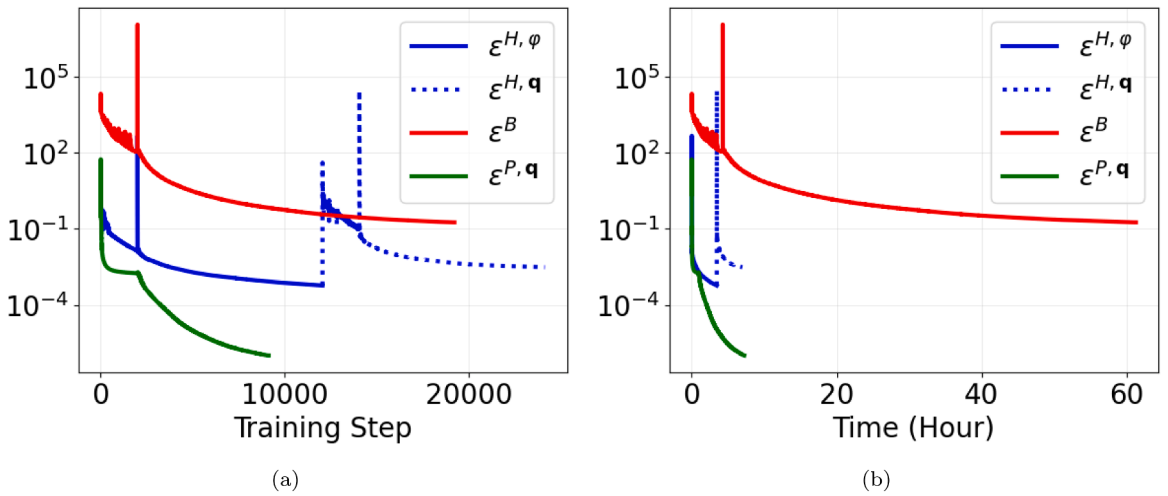


Fig. 6. Test 1: Behaviour of loss values as the number of training steps increases (left) and time increases (right) during the training phase. Dataset consisting of 1000 convex quadrilaterals.

The training procedure requires numerically evaluating the integrals that appear in the loss definitions. In the case of the H-NAVEM method, the integrals defining the loss functions (14) and (15) are approximated using a set of 50 quadrature points on each edge of the polygon, distributed exponentially toward the polygon vertices. For the B-NAVEM and P-NAVEM methods, the loss integrals (27) and (36) require an appropriate selection of points inside each element. To this end, we first construct a simple triangulation of the polygon by connecting all its vertices to a single interior point. The existence of such a point is guaranteed under the standard VEM assumption that elements are star-shaped [21,34]. On each resulting sub-triangle, we then apply Algorithm 1 with $N = 10$ to generate interior sampling points. Although the algorithm is formulated for the reference triangle with vertices $A = (0, 0)$, $B = (1, 0)$, and $C = (0, 1)$, points in any physical triangle are readily obtained via the standard finite element mapping from the reference configuration. The resulting point distribution within a generic polygon tends to cluster near the element edges, as illustrated in Fig. 5a. Moreover, the point density naturally increases close to the boundary and in sub-triangles associated with short edges or near re-entrant corners (see Fig. 5b). This feature is particularly advantageous, since the target functions typically exhibit larger gradients in these regions; a denser sampling therefore improves the accuracy of the neural network approximation in these areas.

To train each network, we first employ the Adam optimizer [37] for 2000 epochs, using an exponentially decaying learning rate from 10^{-2} to 10^{-3}, followed by up to 10,000 epochs with the BFGS optimizer [38]. The evolution of the loss functions, either with respect to the number of weight updates and the elapsed training time, is reported in Fig. 6 for the case of convex quadrilaterals. The corresponding plots for the remaining polygon classes are omitted, as they exhibit trends similar to those observed for convex quadrilaterals. The results indicate that H-NAVEM requires a larger number of optimization steps to reach convergence. However, each training step is computationally inexpensive, resulting in a competitive overall training time. In contrast, B-NAVEM incurs a substantially higher computational cost, with a total training time approximately eight times longer than that of the other two approaches.

To better explain the observed differences in computational time, we first observe that all matrices and vectors independent of the neural network parameters are precomputed in a pre-processing step. For H-NAVEM, this includes the evaluations of harmonic functions belonging to $\mathcal{H}^{\mathrm{H}}_{j,E}$ and their derivatives, whereas for B-NAVEM and P-NAVEM, it includes the evaluations of the ADFs (24), the transfinite interpolants (25), and their derivatives. These quantities are evaluated only once at the quadrature points and subsequently reused throughout the training process for the computation of the loss functions. Since the pre-processing time is negligible with respect to training time, the pre-processing time is not included in the reported time. During training, the only quantities that must be evaluated repeatedly are the neural network output and its derivatives. Consequently, the differences in computational cost among the three methods are mainly determined by the highest-order network derivatives required. In particular, H-NAVEM requires only the evaluation of the network output; P-NAVEM requires both the output and its gradient, whereas B-NAVEM additionally requires the computation of the network Laplacian at each epoch, which is considerably more expensive. Thus, the difference in the computational times reported in Fig. 6b strongly depends on the computation of the network Laplacian. It is worth emphasizing that this additional cost affects only the offline training phase. As shown in the following numerical experiments, during the test phase, when the trained networks are used to predict basis functions to solve the discrete problem (7), the Laplacian is no longer required for B-NAVEM. As a result, the computational costs of B-NAVEM and P-NAVEM become essentially identical during the online phase.

Remark 7. The computational cost of B-NAVEM is mainly due to the evaluation of the network Laplacian required by the PINN formulation. Although this cost could be reduced by adopting, for instance, a Variational PINN (V-PINN) approach [30,39,40], which requires only first-order derivatives, improving the training efficiency is not the primary focus of this work. Moreover, a PINN can

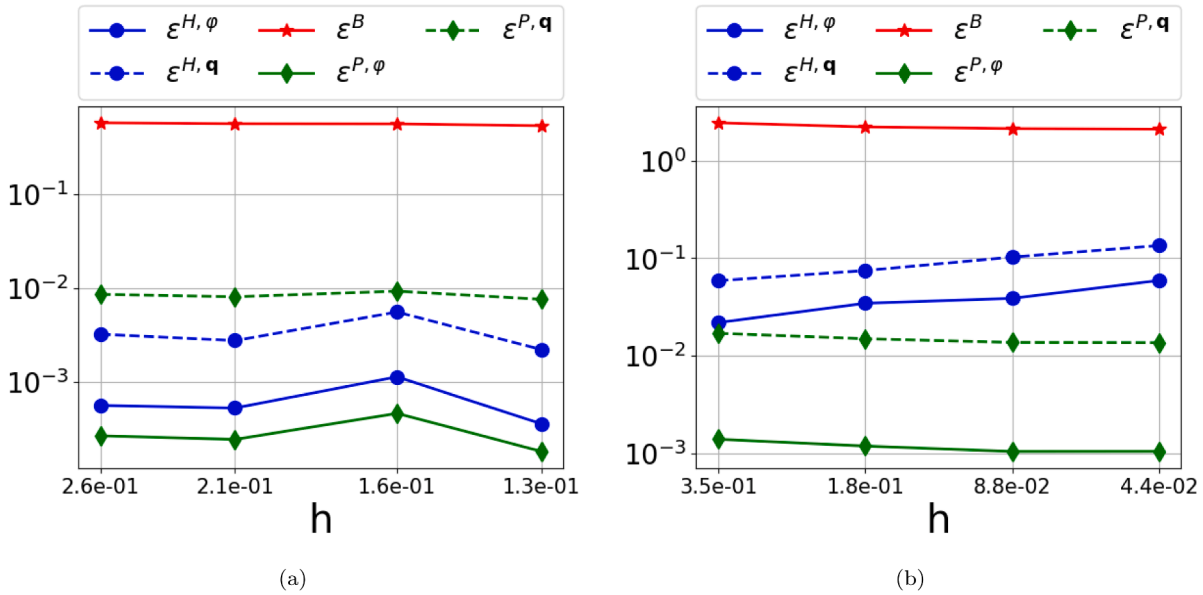


Fig. 7. Tests 1, 2, and 3: Test Loss values on each mesh refinement. Left: Voronoi. Right: Convex-Concave.

be trained by enforcing the Dirichlet boundary conditions through an extra penalization term in the loss function, rather than by means of the operator $\mathcal{B}_{j,E}$, defined in (19). This approach may simplify the optimization landscape, leading to a real unconstrained minimization problem. However, our objective is to construct basis functions that exactly satisfy the boundary conditions and preserve the continuity properties required to overcome the inconsistencies observed in H-NAVEM (see Theorem 1). For this reason, we enforce the Dirichlet conditions exactly through $\mathcal{B}_{j,E}$, following a strategy that is well established in the literature [18,19,41,42], which allows us to achieve very good performance and guarantees that the requested properties are satisfied.

Concerning memory usage, the computation of the neural network Laplacian represents the main computational bottleneck for B-NAVEM, resulting in a significantly higher memory demand than for H-NAVEM and P-NAVEM. Consequently, B-NAVEM requires a smaller network architecture to avoid out-of-memory issues. Specifically, H-NAVEM and P-NAVEM are trained on every polygon class using a network with 5 hidden layers and 50 neurons per layer. The same architecture is adopted for B-NAVEM when training on the convex and concave quadrilateral datasets (each consisting of 1000 samples) and on the convex heptagon dataset (approximately 300 samples), where the memory requirements remain manageable. However, for the classes of convex pentagon and convex hexagon datasets (each containing 1000 samples), the Laplacian computation becomes too memory-intensive. Therefore, in these last cases, B-NAVEM is trained using a reduced architecture with 4 hidden layers and 40 neurons per layer, which prevents out-of-memory errors.

Fig. 7 shows the behaviour of the test losses for each family of meshes and for each method. We note that, in this plot as well as in the following ones, the mesh size h decreases from left to right. In particular, for the case of the P-NAVEM method, we report both the cost functionals (35) and (36), even if we remember that we train the P-NAVEM neural network just by minimizing the loss (36) that involves gradients. We can observe that since the training datasets we built are very representative of the test datasets, training and test performance are very similar to each other.

Concerning the values of the various losses, we observe that the different nature of the loss functions associated with the different methods prevents a fair direct comparison. To compare the accuracy of the different neural-based methods during the online test phase, it is therefore necessary to introduce a meaningful and common evaluation metric. In particular, adopting the H-NAVEM or B-NAVEM loss functions as comparison metrics would be misleading, since these losses vanish identically for some networks by construction. More specifically, the H-NAVEM loss is zero for both the B-NAVEM and P-NAVEM networks, while the B-NAVEM loss is zero for the H-NAVEM network. As a result, these losses do not provide a meaningful metric for comparing the different methods. For this reason, to enable a fair and informative comparison on the test datasets, we evaluate all neural-based methods using the loss functions (35) and (36), introduced in the P-NAVEM framework, as evaluation metrics. These losses can be consistently computed for all methods and represent the most relevant indicators of accuracy when numerically solving the Problem (2).

The results are reported in Fig. 8 for each test mesh family. As expected, P-NAVEM consistently achieves the lowest values of the considered metrics, since its training objective directly minimizes a closely related loss function. We refer to the next sections for further comments.

Finally, we emphasize that both the test loss values reported in Fig. 7 and the metrics (35)–(36) shown in Fig. 8 are evaluated under conditions that differ from those used during training. Specifically, the evaluation is performed on polygons that are not included in the training datasets and on a different set of sampling points, thereby providing a more stringent assessment of the networks

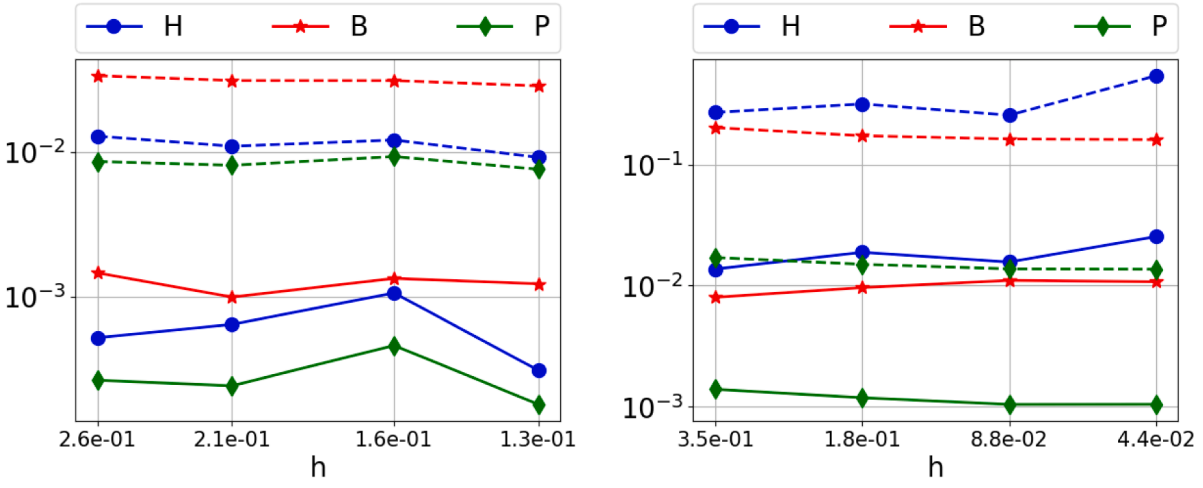


Fig. 8. Tests 1, 2, and 3: Behaviour of the metrics (35) and (36) computed on the test datasets for each method and for each test mesh refinement. Left: Voronoi. Right: Convex-Concave.

generalization capabilities. In particular, the sampling points are generated by Algorithm 1 using $N = 13$, whereas the training phase employs $N = 10$.

7.2. Test 2: A diffusion-advection-reaction problem

Let us consider the domain $\Omega = (0,1)^2$ with boundary $\partial\Omega$ and the following diffusion-advection-reaction problem:

$$\begin{cases} -\nabla\cdot(\boldsymbol{D}(\boldsymbol{x})\nabla u) + \boldsymbol{\beta}(\boldsymbol{x})\cdot\nabla u + \gamma(\boldsymbol{x})u = f & \text{in } \Omega, \\ u = g_D & \text{on } \partial\Omega, \end{cases} \tag{49}$$

where

$$\boldsymbol{D}(\boldsymbol{x}) = \begin{bmatrix} 1+x_2^2 & -x_1x_2 \\ -x_1x_2 & 1+x_1^2 \end{bmatrix}, \qquad \boldsymbol{\beta}(\boldsymbol{x}) = \begin{bmatrix} x_1 \\ -x_2 \end{bmatrix}, \qquad \gamma(\boldsymbol{x}) = x_1x_2,$$

and the forcing term f and the Dirichlet boundary condition g_D are chosen so that the exact solution, shown in Fig. 9, is

$$u(\boldsymbol{x}) = \sin\left[\pi^2\left(\left(x_1-\frac{1}{2}\right)^2+\left(x_2-\frac{1}{2}\right)^2\right)\right]\left(x_1-\frac{1}{2}\right)(1+\sin(\pi x_1)\sin(\pi x_2)). \tag{50}$$

For comparison purposes, Problem (49) is solved with the standard VEM and H-NAVEM, and with the new methods B-NAVEM and P-NAVEM on the two mesh families $\{\mathcal{T}^Q_{h,i}\}_{i=1}^4$ and $\{\mathcal{T}^V_{h,i}\}_{i=1}^4$ described in the previous test. Moreover, we remark that, for neural-based methods, the neural networks employed in this simulation are the same as those described in the previous test case, reported in Section 7.1.

Since the approximate solutions obtained with H-NAVEM, B-NAVEM, and P-NAVEM are known in closed form, we measure the performance of these methods by looking at the behaviour of the following errors

$$\mathrm{err}_0^* = \sqrt{\sum_{E\in\mathcal{T}_h}\|u-u_h^*\|^2_{0,E}}, \quad \mathrm{err}_\nabla^* = \sqrt{\sum_{E\in\mathcal{T}_h}\|\nabla u-\nabla u_h^*\|^2_{0,E}}, \quad *\in\{\mathrm{H},\mathrm{B},\mathrm{P}\}. \tag{51}$$

as the mesh size h decreases. For the neural-based methods, the discrete solution and its gradient are given by

$$u_h^* = \sum_{i=1}^{N^{\mathrm{dof}}} u_i^*\varphi_i^* \quad \nabla u_h^* = \sum_{i=1}^{N^{\mathrm{dof}}} u_i^*\boldsymbol{q}_i^* \quad \forall\, *\in\{\mathrm{H},\mathrm{B},\mathrm{P}\},$$

where the vector containing the degrees of freedom $\{u_i^*\}_{i=1}^{N^{\mathrm{dof}}}$ is the solution of the discrete Problem (7). On the other hand, since the VEM solution is not known in a closed form inside each element, we need to resort to polynomial projections of virtual functions to access their point-wise evaluation and compute the errors. Thus, we consider the usual definition of the VEM errors [7]:

$$\mathrm{err}_0^{\mathrm{V}} = \sqrt{\sum_{E\in\mathcal{T}_h}\|u-\Pi_1^{0,E}u_h^{\mathrm{V}}\|^2_{0,E}}, \quad \mathrm{err}_\nabla^{\mathrm{V}} = \sqrt{\sum_{E\in\mathcal{T}_h}\|\nabla u-\Pi_1^{\nabla,E}\nabla u_h^{\mathrm{V}}\|^2_{0,E}}, \tag{52}$$

where $\Pi_1^{0,E}$ denotes the L^2-projection onto $\mathbb{P}_1(E)$, while $\Pi_1^{\nabla,E}$ represents the elemental projection onto $\mathbb{P}_1(E)$ in the energy norm.

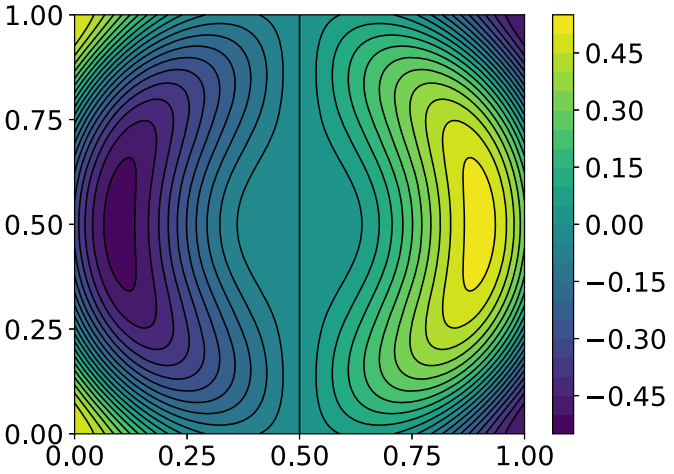


Fig. 9. Test 2: Contour plot of the exact solution u defined in (50).

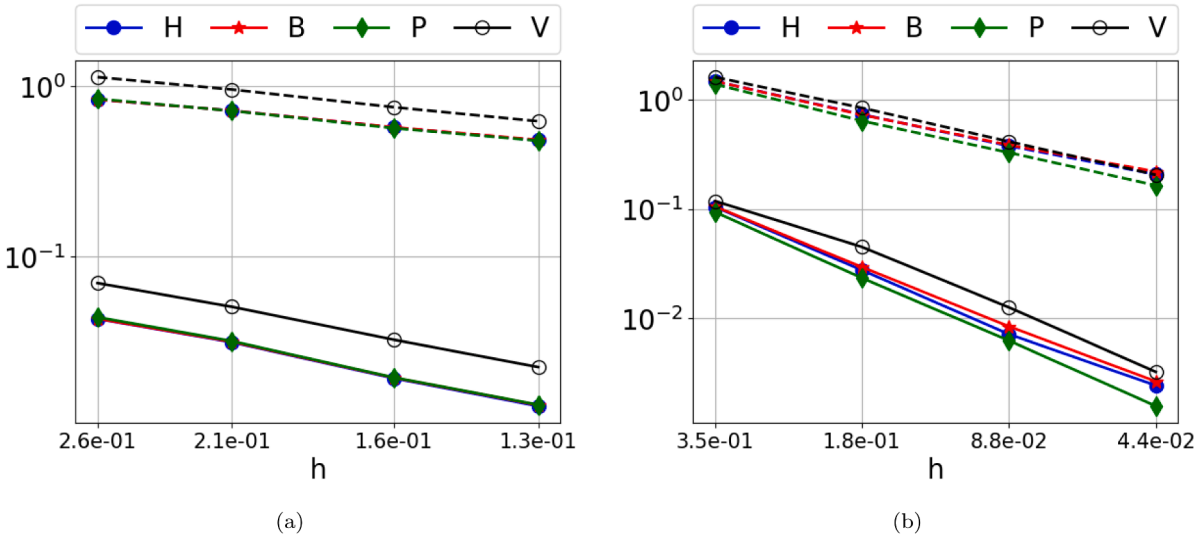


Fig. 10. Test 2: Behaviour of the errors (51) for the neural-based methods and of errors (52) for the VEM method as h decreases. Solid lines represent error err_0, whereas dashed lines denote err_V errors. Left: Voronoi. Right: Convex-Concave.

The errors decay related to the two mesh families is shown in Fig. 10 for each method. It can be observed that the convergence rates of neural-based methods are very close to the VEM ones in both cases. Moreover, we can notice that the error curves related to neural-based methods are downward shifted with respect to the VEM ones for both mesh families. We further note that all the neural-based methods perform very similarly on Voronoi meshes. On the other hand, P-NAVEM is the most accurate on meshes involving concave elements.

Fig. 11 reports the condition number of the global stiffness matrix on both families of meshes. The overall behaviour is essentially the same for all methods. P-NAVEM exhibits the larger condition numbers on Voronoi meshes, while the standard VEM attains slightly larger values on Convex-Concave meshes. Nevertheless, these differences are negligible and do not affect the stability of the methods.

To conclude this experiment, we compare the different methods by looking at the computational time T (in seconds) required to solve Problem (49) for each mesh refinement with each method. In particular, the computed time T represents an average time over 10 different executions of a Python code that exploits TensorFlow [43] for neural network operations, which runs on Ubuntu 24.04 LTS 64-bit, 12th Gen Intel(R) Core(TM) i7-1255U CPU (4.7 GHz) and 16 GB RAM.

More specifically, for each neural-based method, this time accounts for

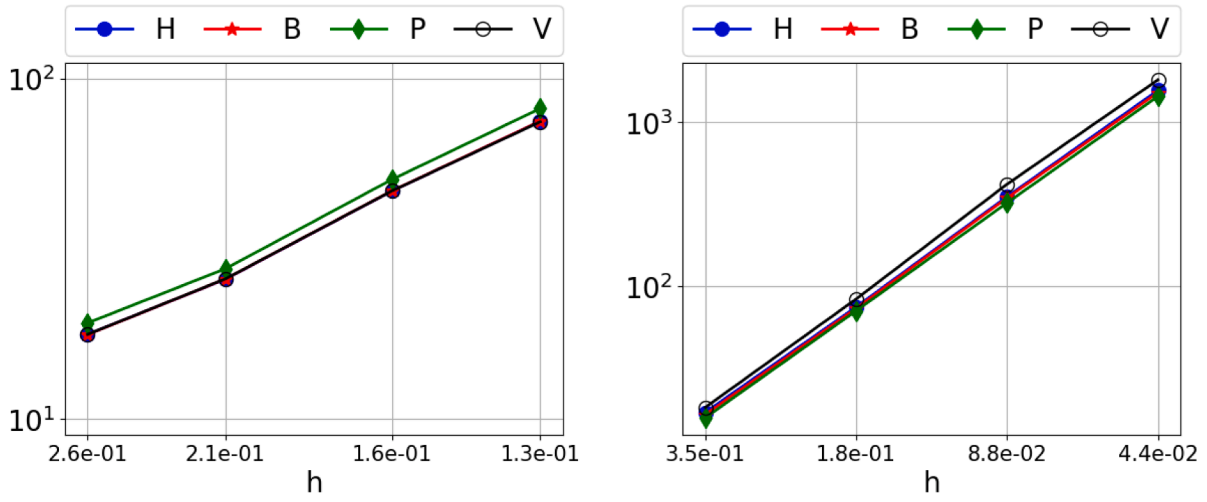


Fig. 11. Test 2: Condition number of the global system matrix associated with Problem (49). Left: Voronoi. Right: Convex-Concave.

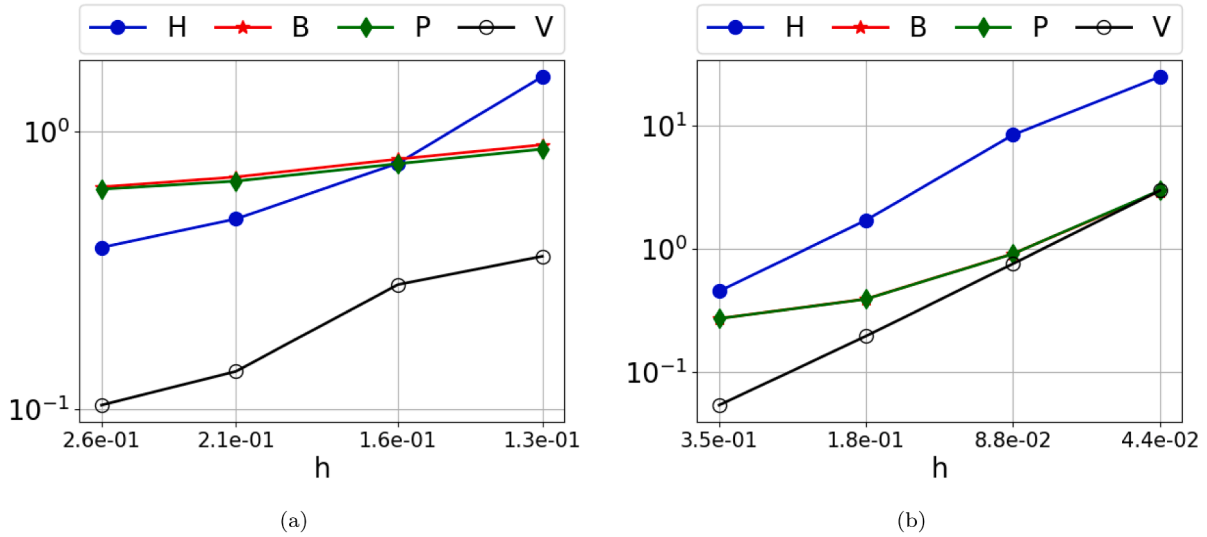


Fig. 12. Test 2: Time required (in seconds) to assemble the global system matrix and solve the discrete problem associated with (49). Left: Voronoi. Right: Convex-Concave.

- the time needed to evaluate auxiliary functions at quadrature points, like the harmonic functions in $\mathcal{H}^{\mathrm{H}}_{j,E}$ and their derivatives for H-NAVEM, the ADFs (24), the transfinite interpolant (25), and their derivatives for B-NAVEM and P-NAVEM;
- the time needed to encode the neural network input and to produce the output;
- the time to assemble and solve the associated discrete system.

For the standard VEM, instead, the computational time T accounts for the time needed to compute projection and stability operators as well as the time needed to assemble and solve the global discrete system.

The behaviour of the computational time T for each method and mesh family is shown in Fig. 12. From these log–log plots, we observe that neural-based methods generally require more computational time than the standard VEM. However, such a difference decreases when employing the B-NAVEM or the P-NAVEM method for finer meshes.

We remark that, to mitigate the cost associated with neural network evaluations, we minimize the number of forward passes through the neural network by aggregating the inputs associated with multiple basis functions, thus exploiting TensorFlow vectorization capabilities and reducing the overall computational burden. Nevertheless, the intrinsic cost per network call has a stronger impact on small datasets, corresponding to coarse meshes, and on mesh families characterized by a high variability in the number of polygon vertices (polygon classes), as already observed in [15].

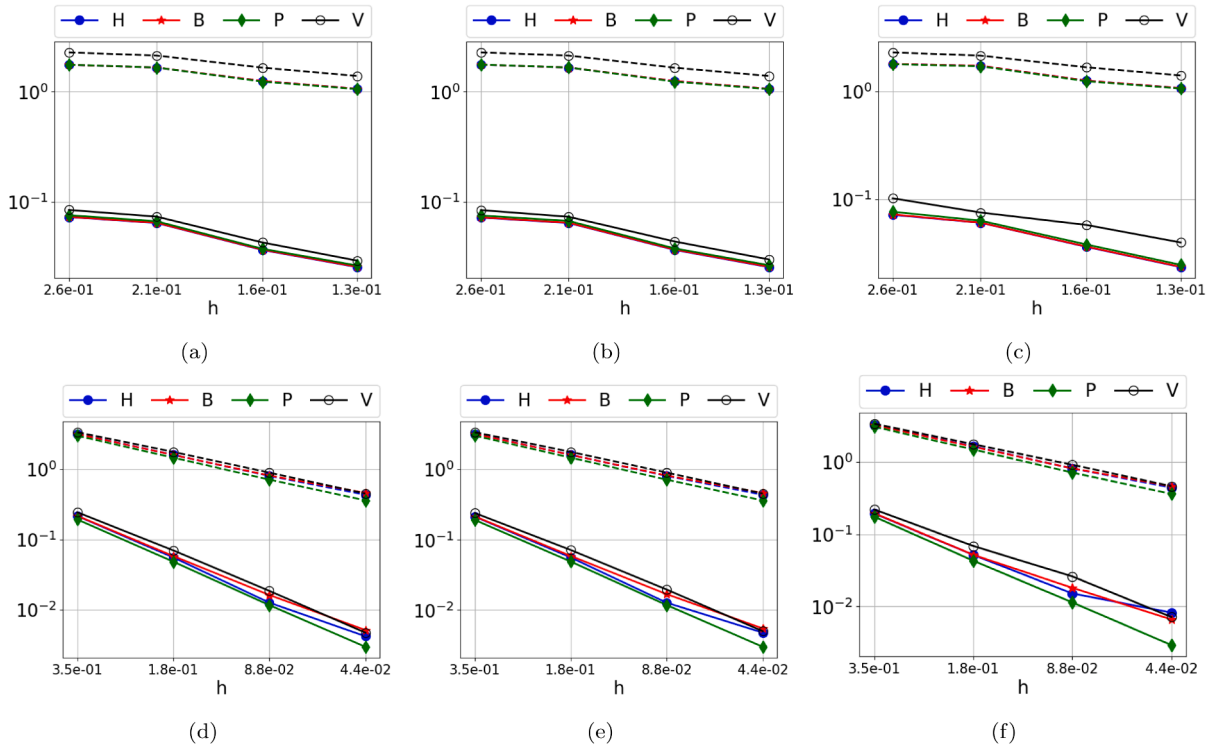


Fig. 13. Test 3: Behaviour of the errors (51) for the neural-based methods and of errors (52) for the VEM method as h decreases. Solid lines represent err_0, whereas dashed lines denote err_∇. Each column refers to a value of λ, namely 1.0, 0.5, and 0.1 from left to right. First row: Voronoi. Second row: Convex-Concave.

No significant difference is observed between the computational costs of the B-NAVEM and P-NAVEM methods. Indeed, both approaches require the same number of neural network evaluations and the same computational effort during the test phase to evaluate the auxiliary functions needed to enforce boundary conditions. Moreover, we observe that this overhead with respect to VEM becomes negligible as h decreases for P-NAVEM and B-NAVEM, since these auxiliary functions can be computed very efficiently using TensorFlow and NumPy vectorization [44].

On the other hand, the computation of the harmonic functions required to evaluate the H-NAVEM basis functions is performed element-wise. In this regard, we observe that we always adopt a value for the harmonic polynomial degree $\ell^{\mathcal{NN}} = 20$ for all the H-NAVEM neural networks we train, without fine-tuning this value, further increasing the cost associated with harmonic function evaluations. As a consequence, the associated computational cost increases linearly with the number of elements.

7.3. Test 3: A nonlinear diffusion problem

To show the benefits of avoiding projection and stability operators that are typical of the standard virtual element methods, in this section, we replicate Test Problem 3 in [15] on the new families of meshes $\{\mathcal{T}^V_{h,i}\}_{i=1}^4$ and $\{\mathcal{T}^Q_{h,i}\}_{i=1}^4$. More precisely, we set $\Omega = (0,1)^2$ and consider the following nonlinear diffusion problem

$$\begin{cases} -\nabla \cdot (\boldsymbol{D}(u,\lambda)\nabla u) = f & \text{in } \Omega, \\ u = g_D & \text{on } \partial\Omega, \end{cases} \tag{53}$$

where $\lambda \in \{1.0, 0.5, 0.1\}$ is a problem parameter, and the diffusion coefficient is defined as

$$\boldsymbol{D}(u,\lambda) = \frac{1}{\lambda + u^2}.$$

For computing the errors (51) and (52), we set the Dirichlet boundary condition and the forcing term in accordance with the following exact solution

$$u(\boldsymbol{x}) = \frac{1}{8}\left[\sin\left(3\pi((x_1-0.5)^2 + (x_2-0.5)^2)\right)\right]^3.$$

As in [15], to solve the nonlinear Problem (53), we adopt the Newton method and, by setting as the initial guess the all-zero vector, we adopt the following stopping criteria:

$$\|\boldsymbol{r}_h^{m,*}\| \le \alpha_{r,f}\|\boldsymbol{f}_h^{0,*}\| + \alpha_{a,f} \quad \text{and} \quad \|\boldsymbol{\delta}_h^{m,*}\| \le \alpha_{a,\delta} \quad \forall * \in \{\mathrm{V}, \mathrm{H}, \mathrm{B}, \mathrm{P}\},$$

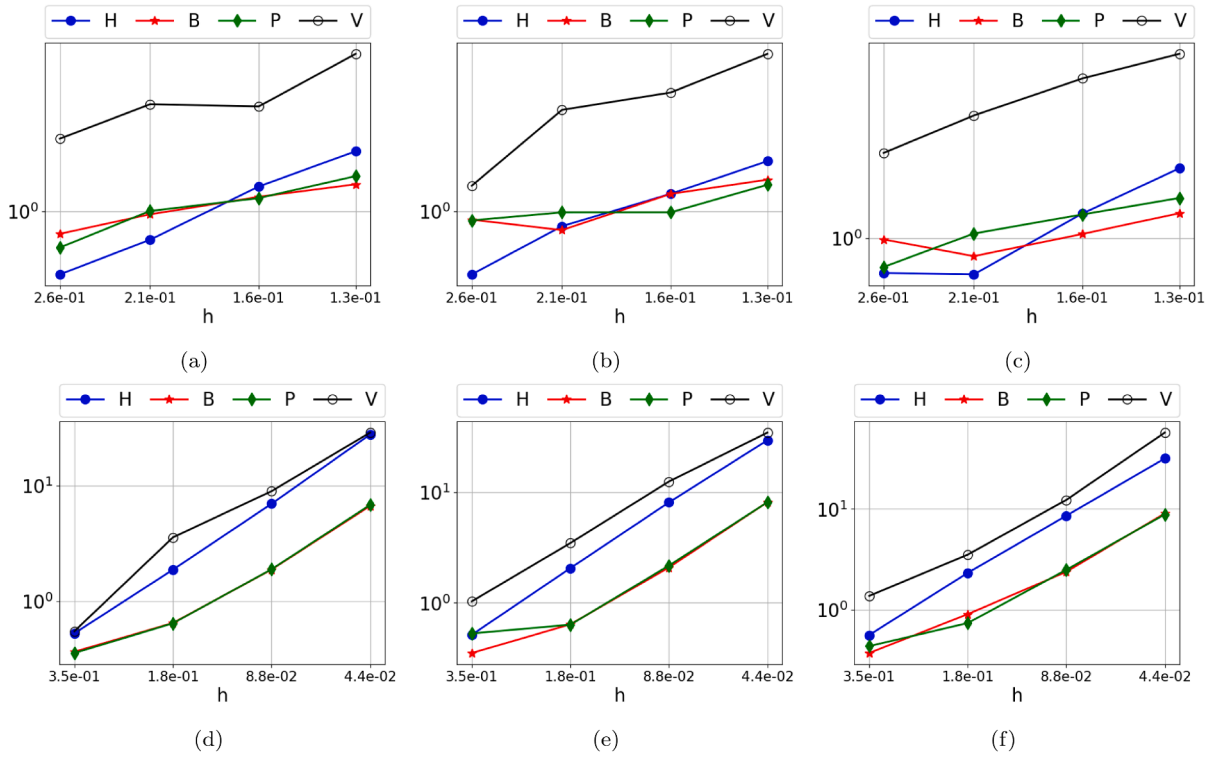


Fig. 14. Test 3: Time required (in seconds) to assemble the global system matrix and solve the discrete problem associated with (53). Each column refers to a value of λ, namely 1.0, 0.5, and 0.1 from left to right. First row: Voronoi. Second row: Convex-Concave.

where $\boldsymbol{r}_h^{m,*}$ is the residual associated with the method at the nonlinear iteration m, $\boldsymbol{f}_h^{0,*}$ is the initial residual, $\boldsymbol{\delta}_h^{m,*}$ is the difference between the discrete solutions at the iteration $m+1$ and m, and $\alpha_{r,f}$, $\alpha_{a,f}$, and $\alpha_{a,\delta}$ are tolerances chosen by the user.

In this test, we compute the errors defined in (51) and (52) to compare the performance and assess the accuracy of the different methods. In addition, to highlight the advantages of avoiding projection and stability operators when solving nonlinear problems, we introduce the following performance indicators. For each neural-based method $*\in \{\mathrm{H},\mathrm{B},\mathrm{P}\}$, we compute:

S.1 $r^*(T)$: the ratio between the time T (in seconds) required by VEM and by method $*$ to solve the nonlinear Problem (53);
S.2 $r^*(m)$: the ratio between the number of nonlinear iterations m required by VEM and by method $*$;
S.3 $r^*(\mathrm{ATI})$: the ratio between the Average Time per Iteration (ATI) of VEM and that of method $*$.

In Figs. 13 and 14, we report the error behaviour and the total computational time T, computed as described in Section 7.2, respectively, as the mesh size h decreases, for each method and mesh family.

As already observed in Test 2 (Section 7.2), avoiding projection and stability operators leads to smaller error constants compared to the standard VEM. Moreover, the P-NAVEM method, characterized by the smaller polynomial losses (35) and (36), is confirmed to be the most accurate among the neural-based methods for sufficiently small values of h, especially in the presence of concave elements. Furthermore, we observe that as the problem becomes more nonlinear (i.e., as λ decreases), the advantages of using neural-based methods over VEM become more pronounced.

Fig. 14 shows that the computational-time behaviour of neural-based methods is consistent with the one observed in the previous test case: for the coarser mesh and the Voronoi family, H-NAVEM is faster than B-NAVEM or P-NAVEM, but it is slower when simulating over finer meshes or over meshes belonging to the Convex-Concave family. However, in this nonlinear setting, neural-based methods require significantly less computational time than VEM in total to solve Problem (53). This behaviour can be explained by two main factors. First, the overhead associated with neural-network utilities is amortized over all nonlinear iterations, thus improving the overall efficiency of neural-based methods. Second, these methods typically require fewer nonlinear iterations than VEM to achieve the desired accuracy, resulting in a smaller number of total calls to the direct solver, which is used to solve the linearized discrete problem at each iteration.

These observations are summarized by the statistics 1–3 reported in Tables 1 and 2 for the Voronoi and Convex–Concave mesh families, respectively. From these tables, we highlight that neural-based methods employ the same number of nonlinear iterations to reach the desired accuracy, i.e. $r^{\mathrm{H}}(m) = r^{\mathrm{B}}(m) = r^{\mathrm{P}}(m)$. We further observe that, for all mesh families, the best performance of neural-based methods over VEM is obtained for the smallest value of $\lambda = 0.1$. This suggests that approaches that avoid stabilization and projection operators become increasingly efficient as the underlying PDE exhibits stronger nonlinearities.

Table 1
Test 3: Statistics about computational time required by each method to solve the nonlinear Problem (53) via the Newton method for each mesh refinement i. Voronoi family of meshes. For each row, the values for the average time T and the average time per iteration ATI related to the method with the best performance are highlighted in bold.

λ	i	$r^{\mathrm{H}}(T)$	$r^{\mathrm{H}}(m)$	$r^{\mathrm{H}}(\mathrm{ATI})$	$r^{\mathrm{B}}(T)$	$r^{\mathrm{B}}(m)$	$r^{\mathrm{B}}(\mathrm{ATI})$	$r^{\mathrm{P}}(T)$	$r^{\mathrm{P}}(m)$	$r^{\mathrm{P}}(\mathrm{ATI})$
1.0	0	**3.15**	1.67	**1.89**	2.23	1.67	1.34	2.51	1.67	1.51
0.5	0	**2.56**	2.00	**1.28**	1.43	2.00	0.72	1.45	2.00	0.72
0.1	0	**3.30**	1.75	**1.88**	2.36	1.75	1.35	3.11	1.75	1.77
1.0	1	**3.14**	1.67	**1.89**	2.54	1.67	1.52	2.46	1.67	1.48
0.5	1	3.44	1.83	1.87	**3.58**	1.83	**1.95**	2.97	1.83	1.62
0.1	1	**4.85**	1.75	**2.77**	4.05	1.75	2.31	3.23	1.75	1.85
1.0	2	1.97	1.67	1.18	**2.15**	1.67	**1.29**	2.17	1.67	1.30
0.5	2	2.93	1.83	1.60	2.94	1.83	1.60	**3.57**	1.83	**1.95**
0.1	2	3.83	1.50	2.55	**4.69**	1.50	**3.13**	3.86	1.50	2.58
1.0	3	2.28	1.50	1.52	**3.01**	1.50	**2.01**	2.81	1.50	1.87
0.5	3	3.12	1.67	1.87	3.80	1.67	2.28	**4.01**	1.67	**2.41**
0.1	3	3.12	1.50	2.08	**4.88**	1.50	**3.25**	4.19	1.50	2.79

Table 2
Test 3: Statistics about computational time required by each method to solve the nonlinear Problem (53) for each mesh refinement i. Convex-Concave family of meshes. For each row, the values for the average time T and the average time per iteration ATI related to the method with the best performance are highlighted in bold.

λ	i	$r^{\mathrm{H}}(T)$	$r^{\mathrm{H}}(m)$	$r^{\mathrm{H}}(\mathrm{ATI})$	$r^{\mathrm{B}}(T)$	$r^{\mathrm{B}}(m)$	$r^{\mathrm{B}}(\mathrm{ATI})$	$r^{\mathrm{P}}(T)$	$r^{\mathrm{P}}(m)$	$r^{\mathrm{P}}(\mathrm{ATI})$
1.0	0	1.04	1.50	0.70	1.51	1.50	1.00	**1.55**	1.50	**1.03**
0.5	0	2.01	1.83	1.10	**2.94**	1.83	**1.60**	1.95	1.83	1.06
0.1	0	2.45	2.57	0.95	**3.70**	2.57	**1.44**	3.15	2.57	1.23
1.0	1	1.90	1.67	1.14	5.52	1.67	3.31	**5.55**	1.67	**3.33**
0.5	1	1.69	1.83	0.92	5.44	1.83	2.97	**5.48**	1.83	**2.99**
0.1	1	1.52	1.75	0.87	3.90	1.75	2.23	**4.79**	1.75	**2.74**
1.0	2	1.29	1.50	0.86	**4.78**	1.50	**3.19**	4.76	1.50	3.17
0.5	2	1.54	1.43	1.08	**6.00**	1.67	**3.60**	5.76	1.67	3.46
0.1	2	1.44	1.50	0.96	**5.19**	1.50	**3.46**	4.96	1.50	3.30
1.0	3	1.04	1.33	0.78	**4.33**	1.33	**3.25**	4.24	1.33	3.18
0.5	3	1.17	1.29	0.91	4.26	1.29	3.32	**4.27**	1.29	**3.32**
0.1	3	1.81	1.25	1.45	6.36	1.25	5.08	**6.51**	1.25	**5.21**

8. Conclusion

In this manuscript, we propose two polygonal discretization methods, called B-NAVEM and P-NAVEM, as alternatives to the standard H-NAVEM, in which local basis functions are constructed over polygons using pre-trained neural networks.

The B-NAVEM basis functions are defined through a Physics-Informed Neural Network that minimizes the residual of the elemental Laplace problems characterizing the virtual element basis functions. As in the classical VEM and contrary to H-NAVEM, B-NAVEM functions are linear polynomials on the boundary. However, their Laplacian is only approximately zero, rather than identically zero as in the VEM or H-NAVEM setting.

The P-NAVEM approach, on the other hand, does not aim at approximating the VEM space itself. Instead, it directly constructs a local approximation space that accurately reproduces polynomials, a property that is essential to guarantee optimal polynomial convergence rates.

Contrary to standard H-NAVEM, where the lack of inter-element continuity introduces an additional inconsistency term in the error estimate that may deteriorate as the mesh size decreases [10], the proposed methods construct spaces that are exactly C^0-conforming. The absence of this inconsistency leads to improved convergence estimates, reducing the impact of the neural network accuracy, as demonstrated by the theoretical results established in this manuscript.

A series of numerical experiments is presented to compare the performance of the different methods on both linear and nonlinear benchmark problems. The results indicate that the P-NAVEM method provides the best trade-off between accuracy and computational cost, both in the training stage and in the testing phase.

CRediT authorship contribution statement

Stefano Berrone: Writing – review & editing, Supervision, Project administration, Funding acquisition, Formal analysis; **Moreno Pintore:** Writing – review & editing, Writing – original draft, Visualization, Software, Methodology, Investigation, Formal analy-

sis, Data curation, Conceptualization; **Gioana Teora:** Writing – review & editing, Writing – original draft, Visualization, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization.

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.

Acknowledgements

We thank Professor Claudio Canuto from Politecnico di Torino for the valuable suggestions.

The author S.B. kindly acknowledges partial financial support provided by European Union through project Next Generation EU, M4C2, PRIN 2022 PNRR project P2022BH5CB_001 “Polyhedral Galerkin methods for engineering applications to improve disaster risk forecast and management: stabilization-free operator-preserving methods and optimal stabilization methods”, and by PNRR M4C2 project of CN00000013 National Centre for HPC, Big Data and Quantum Computing (HPC) (CUP: E13C22000990001). The author M.P. kindly acknowledges financial support provided by PEPR/IA (https://www.pepr-ia.fr/). The author G.T. kindly acknowledges the financial support provided by project NODES which has received funding from the MUR-M4C2 1.5 of PNRR funded by the European Union - NextGenerationEU (Grant agreement no. ECS00000036), by the European Union through PRIN project 20227K44ME “Full and Reduced order modelling of coupled systems: focus on non-matching methods and automatic learning (FaReX)” (CUP: E53D23005510006), and by INdAM-GNCS Project “Metodi numerici politopali stabilization-free e neural-based per problemi accoppiati e non lineari” (CUP: E53C25002010001). The authors S.B. and G.T. kindly acknowledge the financial support provided by the European Union under the “BANDO AGGREGAZIONI R&S - TRANSIZIONE ECOLOGICA” project “OPTIMOLD, Ottimizzazione Integrata per lo Stampaggio Plastico Sostenibile” (CUP: B19J25000450007).

Data availability

Data will be made available on request.

References

[1] M.F. Benedetto, S. Berrone, S. Scialò, A globally conforming method for solving flow in discrete fracture networks using the virtual element method, Finite Elem. Anal. Des. 109 (2016) 23–36. https://doi.org/10.1016/j.finel.2015.10.003

[2] S. Berrone, F. Vicini, Effective polygonal mesh generation and refinement for VEM, Math. Comput. Simul. 231 (2025) 239–258. https://doi.org/10.1016/j.matcom.2024.12.007

[3] S. Berrone, S. Ferraris, D. Grappein, G. Teora, F. Vicini, A 3D-1D virtual element method for modeling root water uptake, Comput. Math. Appl. 213 (2026) 22–52. https://doi.org/10.1016/j.camwa.2026.03.027

[4] A. Ern, F. Hédin, G. Pichot, N. Pignet, Hybrid high-order methods for flow simulations in extremely large discrete fracture networks, SMAI J. Comput. Math. 8 (2022) 375–398. https://doi.org/10.5802/smai-jcm.92

[5] B. Cockburn, J. Gopalakrishnan, R. Lazarov, Unified hybridization of discontinuous Galerkin, mixed, and continuous Galerkin methods for second order elliptic problems, SIAM J. Numer. Anal. 47 (2) (2009) 1319–1365. https://doi.org/10.1137/070706616

[6] I. Fumagalli, M. Corti, N. Parolini, P.F. Antonietti, Polytopal discontinuous Galerkin discretization of brain multiphysics flow dynamics, J. Comput. Phys. 513 (2024) 113115. https://doi.org/10.1016/j.jcp.2024.113115

[7] L. Beirão da Veiga, F. Brezzi, L.D. Marini, A. Russo, Virtual element method for general second order elliptic problems on polygonal meshes, Math. Models Methods Appl. Sci. 26 (04) (2016) 729–750. https://doi.org/10.1142/S0218202516500160

[8] L. Beirão da Veiga, F. Dassi, A. Russo, High-order virtual element method on polyhedral meshes, Comput. Math. Appl. 74 (5) (2017) 1110–1122. https://doi.org/10.1016/j.camwa.2017.03.021

[9] F. Credali, S. Bertoluzza, D. Prada, Reduced basis stabilization and post-processing for the virtual element method, Comput. Methods Appl. Mech. Eng. 420 (2024) 116693. https://doi.org/10.1016/j.cma.2023.116693

[10] S. Berrone, M. Pintore, G. Teora, The neural approximated virtual element method for elasticity problems, Finite Elem. Anal. Des. 252 (2025) 104467. https://doi.org/10.1016/j.finel.2025.104467

[11] P.F. Antonietti, S. Bertoluzza, F. Credali, The reduced basis multigrid scheme for the virtual element method, in: P.F. Antonietti, M. Botti, S. Brenner, G. Ciaramella, H.H. Kim, U. Langer, C. Vergara (Eds.), Domain Decomposition Methods in Science and Engineering XXIX (DD29) 2025, Springer Nature, Cham, 2025, pp. 161–168.

[12] M. Trezzi, U. Zerbinati, When rational functions meet virtual elements: the lightning virtual element method, Calcolo 61 (3) (2024) 35. https://doi.org/10.1007/s10092-024-00585-1

[13] M. Trezzi, U. Zerbinati, The lightning virtual element method for self-adjoint eigenvalue problems, in: F. Marmo, S. Cuomo, A. Cutolo (Eds.), Computational Mechanics and Applied Mathematics: Perspectives from Young Scholars, Springer Nature Switzerland, Cham, 2025, pp. 247–257. https://doi.org/10.1007/978-3-031-76591-9_23

[14] S. Berrone, D. Oberto, M. Pintore, G. Teora, The lowest-order neural approximated virtual element method, in: A. Sequeira, A. Silvestre, S.S. Valtchev, J. Janela (Eds.), Numerical Mathematics and Advanced Applications ENUMATH 2023, Volume 1, Springer Nature Switzerland, Cham, 2025, pp. 129–138. https://doi.org/10.1007/978-3-031-86173-4_13

[15] S. Berrone, M. Pintore, G. Teora, The lowest-order neural approximated virtual element method on polygonal elements, Comput. Struct. 314 (2025) 107753. https://doi.org/10.1016/j.compstruc.2025.107753

[16] A. Gopal, L.N. Trefethen, Solving laplace problems with corner singularities via rational functions, SIAM J. Numer. Anal. 57 (5) (2019) 2074–2094. https://doi.org/10.1137/19M125947X

[17] I. Goodfellow, Y. Bengio, A. Courville, Y. Bengio, Deep Learning, 1, MIT press Cambridge, 2016.

[18] 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. https://doi.org/10.1016/j.cma.2021.114333

[19] S. Berrone, C. Canuto, M. Pintore, N. Sukumar, Enforcing Dirichlet boundary conditions in physics-informed neural networks and variational physics-informed neural networks, Heliyon 9 (8) (2023) e18820. https://doi.org/h10.1016/j.heliyon.2023.e18820

[20] S. Berrone, A. Borio, G. Teora, F. Vicini, POLYDIM: a C++ library for POLYtopal DIscretization methods, Comput. Phys. Commun. 320 (2026) 109937. https://doi.org/10.1016/j.cpc.2025.109937
[21] L. Beirão da Veiga, C. Lovadina, A. Russo, Stability analysis for the virtual element method, Math. Models Methods Appl. Sci. 27 (13) (2017) 2557–2594. https://doi.org/10.1142/S021820251750052X
[22] X. Glorot, Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, in: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, JMLR Workshop and Conference Proceedings, 2010, pp. 249–256.
[23] S. Berrone, G. Teora, F. Vicini, Improving high-order VEM stability on badly-shaped elements, Math. Comput. Simul. 216 (2024) 367–385. https://doi.org/10.1016/j.matcom.2023.10.003
[24] B.-B. Xu, F. Peng, P. Wriggers, Stabilization-free virtual element method for finite strain applications, Comput. Methods Appl. Mech. Eng. 417 (2023) 116555. https://doi.org/10.1016/j.cma.2023.116555
[25] V.L. Rvachev, T.I. Sheiko, R-functions in boundary value problems in mechanics, Appl. Mech. Rev. 48 (4) (1995) 151–188. https://doi.org/10.1115/1.3005099
[26] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, J. Comput. Phys. 378 (2019) 686–707. https://doi.org/10.1016/j.jcp.2018.10.045
[27] A.G. Baydin, B.A. Pearlmutter, A.A. Radul, J.M. Siskind, Automatic differentiation in machine learning: a survey, J. Mach. Learn. Res. 18 (2018) 1–43.
[28] A. Bonito, C. Canuto, R.H. Nochetto, A. Veeser, Adaptive finite element methods, Acta Numer. 33 (2024) 163–485. https://doi.org/10.1017/S0962492924000011
[29] P.G. Ciarlet, The Finite Element Method for Elliptic Problems, Society for Industrial and Applied Mathematics, 2002. https://doi.org/10.1137/1.9780898719208
[30] 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. https://doi.org/10.1007/s11565-022-00441-6
[31] L. Chen, J. Huang, Some error analysis on virtual element methods, Calcolo 55 (1) (2018) 1–23. https://doi.org/10.1007/s10092-018-0249-4
[32] S.C. Brenner, L.R. Scott, The Mathematical Theory of Finite Element Methods, 15 of *Texts in Applied Mathematics*, Springer, 2008. https://doi.org/0.1007/978-0-387-75934-0
[33] A. Ern, J.-L. Guermond, Finite Elements I, Springer Cham, 2022. https://doi.org/10.1007/978-3-030-56341-7
[34] L. Beirão da Veiga, F. Brezzi, A. Cangiani, G. Manzini, L.D. Marini, A. Russo, Basic principles of virtual element methods, Math. Models Methods Appl. Sci. 23 (01) (2013) 199–214. https://doi.org/10.1142/S0218202512500492
[35] L. Beirão da Veiga, F. Brezzi, L..D Marini, A. Russo, The Hitchhiker's guide to the virtual element method, Math. Models Methods Appl. Sci. 24 (08) (2014) 1541–1573. https://doi.org/10.1142/S021820251440003X
[36] Y. Yu, mVEM: A MATLAB software package for the virtual element methods, 2022, arXiv:2204.01339 [math.NA], https://doi.org/10.48550/arXiv.2204.01339
[37] D. Kingma, J. Ba, Adam: a method for stochastic optimization, Int. Conf. Learn. Represent. (2014). https://doi.org/10.48550/arXiv.1412.6980
[38] R. Battiti, F. Masulli, BFGS Optimization for Faster and Automated Supervised Learning, Springer Netherlands, Dordrecht, 1990, pp. 757–760. https://doi.org/10.1007/978-94-009-0643-3_68
[39] E. Kharazmi, Z. Zhang, G.E.M. Karniadakis, hp-VPINNs: variational physics-informed neural networks with domain decomposition, Comput. Methods Appl. Mech. Eng. 374 (2021) 113547. https://doi.org/10.1016/j.cma.2020.113547
[40] S. Berrone, C. Canuto, M. Pintore, Variational physics informed neural networks: the role of quadratures and test functions, J. Sci. Comput. 92 (3) (2022) 100. https://doi.org/10.1007/s10915-022-01950-4
[41] N. Sukumar, R. Roy, A Wachspress-based transfinite formulation for exactly enforcing Dirichlet boundary conditions on convex polygonal domains in physics-informed neural networks, Comput. Mech. (2026). https://doi.org/10.1007/s00466-026-02789-4
[42] K.A. Luong, T. Le-Duc, J. Lee, Automatically imposing boundary conditions for boundary value problems by unified physics-informed neural network, Eng. Comput. 40 (3) (2024) 1717–1739. https://doi.org/10.1007/s00366-023-01871-2
[43] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D.G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, X. Zheng, TensorFlow: a system for large-scale machine learning, in: Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation, OSDI'16, USENIX Association, USA, 2016, p. 265–283.
[44] C.R. Harris, K.J. Millman, S.J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N.J. Smith, R. Kern, M. Picus, S. Hoyer, M.H. van Kerkwijk, M. Brett, A. Haldane, J.F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, T.E. Oliphant, Array programming with NumPy, Nature 585 (7825) (2020) 357–362. https://doi.org/10.1038/s41586-020-2649-2