POLITECNICO DI TORINO
Repository ISTITUZIONALE

The lowest-order neural approximated virtual element method on polygonal elements

Original
The lowest-order neural approximated virtual element method on polygonal elements / Berrone, S., Pintore, M., Teora, G.. - In: COMPUTERS & STRUCTURES. - ISSN 1879-2243. - 314:(2025), pp. 1-17. [10.1016/j.compstruc.2025.107753]

Availability:
This version is available at: 11583/2999113 since: 2025-05-03T05:48:43Z

Publisher:
Elsevier

Published
DOI:10.1016/j.compstruc.2025.107753

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)

18 November 2026

Computers and Structures 314 (2025) 107753

Contents lists available at ScienceDirect

Computers and Structures

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

The lowest-order neural approximated virtual element method on polygonal elements

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] Laboratoire Jacques-Louis Lions, Sorbonne Université, Inria, 4 place Jussieu, Paris, 75005, France

HIGHLIGHTS

- An approximation of first-order VEM basis functions through neural networks.
- Good local approximation properties via custom neural networks are proved.
- A polygonal method without any projection and stabilization operators.
- Faster convergence in highly non-linear problems with respect to the standard VEM.

ARTICLE INFO

Keywords:
NAVEM
Virtual element method
Neural network
Basis functions
Polygonal meshes

ABSTRACT

The lowest-order neural approximated virtual element method on polygonal elements is proposed here. This method employs a neural network to locally approximate the virtual element basis functions, thereby eliminating issues concerning stabilization and projection operators, which are the key components of the standard virtual element method. By employing neural networks, the computational burden of approximating the virtual basis functions is shifted to the offline phase, aligning the novel method with the finite element method in the online assembling phase. We enhance the original approach, mainly designed for quadrilateral elements, by refining the local approximation space with additional harmonic functions to improve the neural network's accuracy on polygonal elements. Furthermore, we propose and analyze different training strategies, each offering varying levels of accuracy and supported by theoretical justifications. Several numerical experiments are conducted to validate our procedure on quite general polygonal meshes and demonstrate the advantages of the proposed method across different problem formulations, particularly in cases where the heavy usage of projection and stabilization terms may represent challenges for the standard version of the method. Particular attention is reserved for triangular meshes with hanging nodes which assume a central role in many virtual element applications.

1. Introduction

The virtual element method (VEM in short), introduced in [1] for the Laplace problem and then extended to general second-order elliptic problems in [2], can be considered a generalization of the finite element method (FEM) which introduces in the local space suitable non-polynomial functions, as well as standard polynomials. The introduction of these non-polynomial functions, which are not required in a closed form, allows to work with polytopal elements in a very simple way while preserving the polynomial accuracy. These virtual element functions are solutions to local PDE problems inside each element of the tessellation that are actually never solved explicitly, neither exactly nor approximately. Since these functions are not explicitly known inside the elements, the discrete bilinear form used in the VEM discretization of the problem is just an approximation of the continuous counterpart that exploits some computable polynomial projections of VEM functions to access their point-wise evaluation. Indeed, the core idea of the standard VEM method is to define suitable local spaces and degrees of freedom that allow to exactly compute the entries of the stiffness bilinear form when at least one of the two entries is a polynomial. The remaining entries, which account for the non-polynomial part, are replaced by

* Corresponding author.
Email 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.compstruc.2025.107753
Received 13 December 2024; Accepted 26 March 2025
Available online 8 April 2025
0045-7949/© 2025 The Author(s). Published by Elsevier Ltd. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

a stabilization term to produce results that are of the right order of magnitude and satisfy stability properties. However, a unique prescription of this stabilization term is not provided by the virtual element theory and its selection is mainly guided by numerical experiments, becoming highly problem-dependent [3]. Furthermore, the presence of the stabilization term can limit the accuracy of the method in cases of strongly anisotropic problems due to its intrinsic isotropic nature [4,5]. Finally, the need to introduce some polynomial projectors to access the point-wise evaluation of VEM functions can represent a limitation in the post-processing phase and may induce many issues also in complex non-linear problems [6,7].

Recently, various efforts have been made to address these limitations. In [4,8], the first stabilization-free methods have been proposed in which the discrete bilinear forms only involve polynomial projections on enhanced polynomial spaces, whose polynomial degrees depend on the geometry of the underlying polygons. In [9], a reduced basis method is proposed to cheaply reconstruct approximations of VEM basis functions which could be exploited to properly design stabilization terms or for post-processing of the solution. In [10], a lightning virtual element method is developed which actually computes the VEM basis functions by solving a PDE problem on each element with the iterative *Laplace solver* proposed in [11]. Lastly, in [12], the neural approximated virtual element method (NAVEM in short), which employs a neural network to approximate the VEM basis functions, has been briefly presented and tested for the case of quadrilateral elements. In the context of the last two methods, it is clear that the usage of the term "virtual" refers only to the underlying local space, since the local construction of VEM basis functions allows one to get rid of any stabilization terms or polynomial projectors that represent the main features of the virtual element method.

In the last few years, thanks to the availability of easily customizable machine learning libraries like TensorFlow [13], PyTorch [14] or JAX [15], numerous novel machine learning enhanced numerical methods have been proposed. In the context of this new research field, known as scientific machine learning (SciML) [16], we present the NAVEM method on polygonal meshes. Initially introduced by the authors for solving the Laplace problem on quadrilateral meshes in [12], this method is now extended to polygons with more than four vertices, allowing for the presence of hanging nodes. Inspired by Refs [4,11] and by the recent success of other SciML techniques, this method leverages the neural network to approximate VEM basis functions as a linear combination of harmonic functions, segregating the main computational effort needed to compute such approximations to the offline stage. Indeed, in the online phase, since the need for computing the local projection matrices and defining a stability operator is circumvented, NAVEM acts like a FEM method on polygonal meshes. Specifically, we propose an enhanced version of the original NAVEM method, modifying the local approximation spaces to include new harmonic functions in addition to the harmonic polynomials to better capture singularities near the vertices of the polygon, and refining the neural network architecture to reduce oscillations between interpolation points. We also explore several training strategies, each offering varying levels of accuracy and aimed at minimizing distinct loss functions, with theoretical justifications provided for each approach. Numerical experiments validate the viability of our procedure on different polygonal meshes and show the advantages of using this new procedure, especially when addressing highly non-linear problems.

We wish to emphasize that the method presented here is not intended as a modification of the virtual element method but rather as an alternative approach that builds upon the theoretical foundation of the latter. Once again, the use of the term "virtual" pertains solely to the underlying local space, as previously described. We are confident that this study paves the way for exploring innovative and advanced strategies for practical applications, effectively combining the strengths of neural networks with those of classical numerical methods.

The paper is structured as follows. Section 2 briefly introduces the VEM formulation, which is essential for developing an appropriate architecture and training strategy for the neural network. The neural approximated method is presented in Section 3, while Section 4 details the network architecture and training strategy. Finally, Section 5 presents various numerical experiments on polygonal meshes to demonstrate the method's performance, including its application to anisotropic and non-linear problems, which can pose challenges for the standard procedure.

2. The model problem and the virtual element method

Let us now introduce some notations used throughout the paper. Given $k \in \mathbb{N}$, we denote by $\|\cdot\|_{H^k(\omega)}$ the norm in the Sobolev space $H^k(\omega)$ on some open subset $\omega \subset \mathbb{R}^2$. Furthermore, we use the symbol $(\cdot,\cdot)_\omega$ to denote both the scalar product in $L^2(\omega)$ and in $L^2(\omega)\times L^2(\omega)$. We recall that given two vector functions $\boldsymbol{v} = \left[v_1, v_2\right]^T$ and $\boldsymbol{u} = \left[u_1, u_2\right]^T$, the scalar product in $L^2(\omega)\times L^2(\omega)$ is defined as

$$(\boldsymbol{v},\boldsymbol{u})_\omega = \int_\omega (v_1u_1 + v_2u_2), \quad \|\boldsymbol{v}\|_{L^2(\omega)} = \sqrt{(\boldsymbol{v},\boldsymbol{v})_\omega}.$$

Let us consider an open, bounded, convex polygonal domain $\Omega \subset \mathbb{R}^2$ with boundary Γ. Given $f \in L^2(\Omega)$, we consider the following Poisson problem:

$$\begin{cases} -\Delta u = f & \text{in } \Omega, \\ u = 0 & \text{on } \Gamma. \end{cases} \tag{1}$$

The variational formulation of problem (1) reads as: *Find* $u \in V = H^1_0(\Omega)$ *such that:*

$$a(u,v) = (f,v)_\Omega \quad \forall v \in V, \tag{2}$$

where the bilinear form $a(\cdot,\cdot) : V\times V \to \mathbb{R}$ is given by:

$$a(u,v) = (\nabla u, \nabla v)_\Omega \quad \forall u, v \in V. \tag{3}$$

2.1. The virtual element space

Let $\mathcal{T}_h$ be a decomposition of Ω into polygons E and let $\mathcal{E}_{h,}$ be the set of edges of the elements in $\mathcal{T}_h$. Furthermore, we denote by N_E^v the number of vertices (and of edges), by $\mathcal{E}_{h,E}$ the set of edges and by h_E the diameter of the element $E \in \mathcal{T}_h$. As usual, h denotes the maximum diameter of the polygons in $\mathcal{T}_h$. We assume that the following mesh assumptions hold true [2].

Assumption 1 (Mesh assumptions) *There exists a positive constant ρ, independent of E and h, such that*

- *each polygon $E \in \mathcal{T}_h$ is star-shaped with respect to a ball of radius $\geq \rho h_E$;*
- *for each edge $e \in \mathcal{E}_{h,E}$, it holds: $|e| \geq \rho h_E$.*

Given a polygon E, for each integer $k \geq 0$, we denote by $\mathbb{P}_k(E)$ the set of two-dimensional polynomials of degree up to k defined on E, of dimension $n_k = \dim \mathbb{P}_k(E) = \frac{(k+1)(k+2)}{2}$. Furthermore, we introduce the set

$$\mathbb{B}_1(\partial E) = \left\{ v \in C^0(\partial E) : v_{|e} \in \mathbb{P}_1(e)\ \forall e \in \mathcal{E}_{h,E} \right\},$$

whose dimension is $\dim \mathbb{B}_1(\partial E) = N_E^v$. For all $E \in \mathcal{T}_h$, we define the lowest-order local virtual element space [1] as the set

$$V_{h,1}(E) = \left\{ v \in H^1(E) : \quad (i)\ \Delta v = 0, \quad (ii)\ v_{|\partial E} \in \mathbb{B}_1(\partial E) \right\}, \tag{4}$$

with dimension $N_E^{\text{dof}} = \dim V_{h,1}(E) = N_E^v$, and we consider the value of $v_h \in V_{h,1}(E)$ at the vertices of E as local degrees of freedom.

The key property of the virtual element method is that, thanks to this definition of the degrees of freedom, we are able to exactly (up to machine precision) compute the local projection $\Pi_1^{E,\nabla} v_h$ of each VEM function $v_h \in V_{h,1}(E)$, where the *computable* local polynomial projector $\Pi_1^{E,\nabla} : H^1(E) \to \mathbb{P}_1(E)$ is defined such that, for each $E \in \mathcal{T}_h$,

$$\left(\nabla v_h - \nabla \Pi_1^{E,\nabla} v_h, \nabla p\right)_E = 0, \quad \forall p \in \mathbb{P}_1(E) \text{ and } \int_{\partial E} \Pi_1^{E,\nabla} v_h = \int_{\partial E} v_h.$$

Finally, the global virtual element space is obtained by gluing together the local spaces as

$$V_{h,1} = \left\{ v \in V \cap C^0\left(\overline{\Omega}\right) : \ v_{h|E} \in V_{h,1}(E) \quad \forall E \in \mathcal{T}_h \right\}.$$

2.2. The virtual element discretization

Initially, we can observe that the continuous bilinear form (3) can be split according to the tessellation $\mathcal{T}_h$ as

$$a(u,v) = \sum_{E \in \mathcal{T}_h} a^E(u,v), \quad a^E(u,v) = (\nabla u, \nabla v)_E \quad \forall u, v \in V.$$

Then, we note that, in general, we are not able to compute the quantity

$$a^E(u_h, v_h) = (\nabla u_h, \nabla v_h)_E \quad \forall u_h, \ v_h \in V_{h,1}(E),$$

since we do not know the virtual element functions in a closed-form in the interior of each element $E \in \mathcal{T}_h$. To overcome this issue, the main idea of the virtual element method is to substitute the continuous bilinear form with a *computable* discrete counterpart $a_h^E(\cdot,\cdot) : V_{h,1}(E) \times V_{h,1}(E) \to \mathbb{R}$ which satisfies the two following properties [1]:

- *Consistency*: For all $p \in \mathbb{P}_1(E)$ and for all $v_h \in V_{h,1}(E)$

  $$a_h^E(p, v_h) = a^E(p, v_h).$$

- *Stability*: There exist two positive constants α_*, α^* independent of h such that

  $$\alpha_* a^E(v,v) \le a_h^E(v,v) \le \alpha^* a^E(v,v), \quad \forall v \in V_{h,1}(E) : \ \Pi_1^{E,\nabla} v = 0. \tag{5}$$

To build a discrete bilinear form that satisfies the consistency and stability properties, the local continuous bilinear form is first split as

$$a^E(u_h, v_h) = a^E\left(\Pi_1^{E,\nabla} u_h, \Pi_1^{E,\nabla} v_h\right) + a^E\left(\left(I - \Pi_1^{E,\nabla}\right) u_h, \left(I - \Pi_1^{E,\nabla}\right) v_h\right), \tag{6}$$

where the equality is due to the orthogonality of $\Pi_1^{E,\nabla}$ with respect to the scalar product induced by $a^E(\cdot,\cdot)$. The first term in the right-hand side of (6) is computable thanks to the definition of the local degrees of freedom, whereas the second one could be approximated by any *computable* symmetric positive definite bilinear form $S^E(\cdot,\cdot)$ that satisfies the stability property (5).

Finally, it can be shown that the local discrete bilinear form

$$a_h^E(u_h, v_h) = a^E\left(\Pi_1^{E,\nabla} u_h, \Pi_1^{E,\nabla} v_h\right) + S^E\left(\left(I - \Pi_1^{E,\nabla}\right) u_h, \left(I - \Pi_1^{E,\nabla}\right) v_h\right)$$

is computable and satisfies the consistency and stability properties [1].

Now, let us define dof_i^E, for each $i = 1, \dots, N_E^{\mathrm{dof}}$ and each $E \in \mathcal{T}_h$, as the operator that associates with each sufficiently smooth function φ its ith local degree of freedom $\mathrm{dof}_i^E(\varphi)$. A standard choice for the stabilization term for the two-dimensional case is given by the *dofi-dofi* stabilization term

$$S^E(u_h, v_h) = \sum_{i=1}^{N_E^{\mathrm{dof}}} \mathrm{dof}_i^E(u_h) \mathrm{dof}_i^E(v_h). \tag{7}$$

We observe that, when dealing with more general elliptic equations, this stabilization is usually pre-multiplied by a constant C_s, which accounts for the magnitude of the diffusion coefficient. Other stabilization methods have been proposed in the literature, which may take integral forms [17] or be a variant of the dofi-dofi stabilization, such as the *D*-recipe version introduced in [18]. In particular, the *D*-recipe form aims to prevent the stabilization from becoming too small in magnitude with respect to the consistency term when high-order methods are considered.

Finally, the virtual element discretization of problem (2) reads as: *Find* $u_h \in V_{h,1}$ *such that:*

$$\sum_{E \in \mathcal{T}_h} a_h^E(u_h, v_h) = \sum_{E \in \mathcal{T}_h} \mathcal{F}_h^E(v_h) \quad \forall v_h \in V_{h,1}, \tag{8}$$

where $\mathcal{F}_h^E : V_{h,1}(E) \to \mathbb{R}$ is the discrete version of the local forcing term and it is given by:

$$\mathcal{F}_h^E(v_h) = \left(f, \Pi_0^{\partial E,0} v_h\right)_E, \quad \text{with} \quad \Pi_0^{\partial E,0} v_h = \int_{\partial E} v_h.$$

3. The neural approximated virtual element method

Let us introduce the set of the VEM Lagrange basis functions $\{\varphi_i\}_{i=1}^{N^{\mathrm{dof}}}$ corresponding to the aforementioned degrees of freedom, each of them associated with a different internal vertex v_i of the tessellation $\mathcal{T}_h$. We denote by $\mathbb{S}_i = \mathrm{supp}(\varphi_i) = \bigcup_{j=1}^{N_{v_i}} E_j$ the support of φ_i, i.e. the union of the N_{v_i} elements $E_j \in \mathcal{T}_h$ adjacent to the vertex v_i. Furthermore, given an element $E \in \mathcal{T}_h$, for the sake of brevity, we denote by $\{\varphi_{j,E}\}_{j=1}^{N_E^{\mathrm{dof}}}$ the set of the restrictions to E of the Lagrange basis functions related to the vertices of E. Clearly, the local and the global virtual element spaces can be written as

$$V_{h,1}(E) = \mathrm{span}\{\varphi_{j,E} : \ j = 1, \dots, N_E^{\mathrm{dof}}\}$$

and

$$V_{h,1} = \mathrm{span}\{\varphi_i : \ i = 1, \dots, N^{\mathrm{dof}}\}.$$

Let us denote by $\mathcal{H}_{j,E}^{\mathcal{NN}}$ a set of harmonic functions in which we approximate the VEM functions, which will be characterized in Section 4.3. Our goal is to approximate both the VEM basis functions $\varphi_{j,E}$ and their gradients $\nabla\varphi_{j,E}$ with a neural network-based approximation $\left(\varphi_{j,E}^{\mathcal{NN}}, \boldsymbol{q}_{j,E}^{\mathcal{NN}}\right)$, $\forall j = 1, \dots, N_E^{\mathrm{dof}}$ and $\forall E \in \mathcal{T}_h$. More specifically, the neural network aims to learn the following highly non-linear map for each vertex v_j of E and $\forall E \in \mathcal{T}_h$:

$$(v_j, E) \mapsto \left(\varphi_{j,E}^{\mathcal{NN}}, \boldsymbol{q}_{j,E}^{\mathcal{NN}}\right) \in \mathcal{H}_{j,E}^{\mathcal{NN}} \times \nabla\mathcal{H}_{j,E}^{\mathcal{NN}}, \tag{9}$$

finding the best linear combinations of some suitable harmonic functions in $\mathcal{H}_{j,E}^{\mathcal{NN}}$ and of their gradients in $\nabla\mathcal{H}_{j,E}^{\mathcal{NN}}$ which minimize the distance between the pair $\left(\varphi_{j,E}^{\mathcal{NN}}, \boldsymbol{q}_{j,E}^{\mathcal{NN}}\right)$ and the target $\left(\varphi_{j,E}, \nabla\varphi_{j,E}\right)$ at the boundary of the element E, where the virtual element functions are well-known.

3.1. The local neural approximated virtual element space

Here and in the sequel, C will denote a generic positive constant, with different meanings in different occurrences.

Given the approximations $\{\varphi_{j,E}^{\mathcal{NN}}\}_{j=1}^{N_E^{\mathrm{dof}}}$, we define the local lowest-order NAVEM space as the set

$$V_{h,1}^{\mathcal{NN}}(E) = \mathrm{span}\left\{\varphi_{j,E}^{\mathcal{NN}}, \ j = 1, \dots, N_E^{\mathrm{dof}}\right\}.$$

Firstly, we note that the functions $\varphi_{j,E}^{\mathcal{NN}}$ should belong to the VEM space $V_{h,1}(E)$ to represent a good approximation of $\varphi_{j,E}$, and, in particular, they should locally satisfy properties (i) and (ii) defined in Eq. 4. In this regard, we note that property (i) is trivially satisfied by functions $\varphi_{j,E}^{\mathcal{NN}}$ by construction, since the functions in $\mathcal{H}_{j,E}^{\mathcal{NN}}$ are harmonic. Instead,

property (ii) is, in general, not satisfied by functions belonging to $\mathcal{H}^{\mathcal{NN}}_{j,E}$. Nevertheless, we overcome this issue by training the neural network to learn functions $\varphi^{\mathcal{NN}}_{j,E}$ mimicking the VEM Lagrange basis functions $\varphi_{j,E}$ at the boundary ∂E of the element E, where all the virtual functions are known in a closed form. In particular, our goal is to minimize the distance between the traces of the functions $\varphi^{\mathcal{NN}}_{j,E}$ and $\varphi_{j,E}$ on ∂E, i.e.

$$\epsilon_{j,E} = \left\| \varphi^{\mathcal{NN}}_{j,E} - \varphi_{j,E} \right\|_{H^{1/2}(\partial E)}, \tag{10}$$

for all E and $j = 1, \dots, N^{\mathrm{dof}}_E$, to learn the non-linear relationship (9).

Thanks to the harmonicity of both the virtual element functions and the NAVEM basis functions, we can exploit the same steps performed in [10] to state the following proposition.

Proposition 1. *For all $E \in \mathcal{T}_h$ and for all $j = 1, \dots, N^{\mathrm{dof}}_E$, it holds*

$$\left\| \varphi_{j,E} - \varphi^{\mathcal{NN}}_{j,E} \right\|_{H^1(E)} \leq C_1 \epsilon_{j,E}, \qquad \left\| \varphi_{j,E} - \varphi^{\mathcal{NN}}_{j,E} \right\|_{L^\infty(\partial E)} \leq C_2 \epsilon_{j,E}, \tag{11}$$

where C_1 and C_2 depends on E and ∂E.

This proposition states that the NAVEM functions $\varphi^{\mathcal{NN}}_{j,E}$ could be a good approximation for the related VEM Lagrange basis functions on the entire element E in the H^1-norm, i.e.

$$\varphi^{\mathcal{NN}}_{j,E} \approx \varphi_{j,E} \text{ on } E, \forall j = 1, \dots, N^{\mathrm{dof}}_E, \ \forall E \in \mathcal{T}_h. \tag{12}$$

Concerning the vector of functions $\boldsymbol{q}^{\mathcal{NN}}_{j,E}$ in Eq. 9, we observe that we are able to compute it exactly as $\boldsymbol{q}^{\mathcal{NN}}_{j,E} = \nabla \varphi^{\mathcal{NN}}_{j,E}$. However, as we will describe in Section 4.5, sometimes better results are obtained approximating $\nabla \varphi^{\mathcal{NN}}_{j,E}$ independently from $\varphi^{\mathcal{NN}}_{j,E}$.

3.2. The NAVEM discretization

At this point, we observe that, since the approximation of each virtual element basis function φ_i, with $i = 1, \dots, N^{\mathrm{dof}}$, is computed locally, the corresponding global approximate function $\varphi^{\mathcal{NN}}_i$ is element-wise defined as

$$\varphi^{\mathcal{NN}}_i = \begin{cases} \varphi^{\mathcal{NN}}_{j,E} & \text{if } v_i \text{ is the } j\text{th vertex of } E \text{ and } E \in \mathbb{S}_i, \\ 0 & \text{otherwise,} \end{cases}$$

while the global lowest-order neural approximate virtual element space reads as

$$V^{\mathcal{NN}}_{h,1} = \mathrm{span}\{ \varphi^{\mathcal{NN}}_i \ : \ i = 1, \dots, N^{\mathrm{dof}} \}. \tag{13}$$

Thus, the NAVEM basis functions are not continuous functions across elements and they may have jumps at element interfaces. Nonetheless, we highlight that the degrees of freedom are not decoupled in our framework. Since NAVEM functions are no longer continuous across elements, we need to consider a broken version of the continuous bilinear form $a(\cdot,\cdot)$. Therefore, the NAVEM discretization of problem (3) reads as: *Find $u^{\mathcal{NN}}_h \in V^{\mathcal{NN}}_{h,1}$ such that:*

$$\begin{aligned} a_{h,\mathcal{NN}}\left(u^{\mathcal{NN}}_h, v^{\mathcal{NN}}_h\right) &= \sum_{E \in \mathcal{T}_h} a^E(u^{\mathcal{NN}}_h, v^{\mathcal{NN}}_h) \\ &= \sum_{E \in \mathcal{T}_h} (f, v^{\mathcal{NN}}_h)_E \quad \forall v^{\mathcal{NN}}_h \in V^{\mathcal{NN}}_{h,1}. \end{aligned} \tag{14}$$

Firstly, we note that also in the standard virtual element method we must consider a broken version of the global bilinear form due to the local definition of both the projection and stability operators.

Secondly, we observe that the lack of continuity of the functions in $V^{\mathcal{NN}}_{h,1} \not\subseteq V$ introduces a kind of consistency error in the approximation of the solution $u \in V$ [19]. Indeed, using integration by parts, we obtain

$$\begin{aligned} a_{h,\mathcal{NN}}\left(u, v^{\mathcal{NN}}_h\right) &= \sum_{E \in \mathcal{T}_h} \left(-\Delta u, v^{\mathcal{NN}}_h\right)_E + \sum_{E \in \mathcal{T}_h} \left(\nabla u \cdot \boldsymbol{n}, v^{\mathcal{NN}}_h\right)_{\partial E} \\ &= \sum_{E \in \mathcal{T}_h} \left(f, v^{\mathcal{NN}}_h\right)_E + \sum_{E \in \mathcal{T}_h} \left(\nabla u \cdot \boldsymbol{n}, v^{\mathcal{NN}}_h\right)_{\partial E} \\ &= \sum_{E \in \mathcal{T}_h} \left(f, v^{\mathcal{NN}}_h\right)_E + \sum_{e \in \mathcal{E}_h} \left(\nabla u, [\![v^{\mathcal{NN}}_h]\!]_e\right)_e, \end{aligned}$$

where $[\![v^{\mathcal{NN}}_h]\!]_e = v^{\mathcal{NN}}_{h|E_1} \boldsymbol{n}_{E_1} + v^{\mathcal{NN}}_{h|E_2} \boldsymbol{n}_{E_2}$ with E_1, E_2 being the elements sharing the edge e. This last term measures the extent to which the continuous solution u fails to satisfy the NAVEM formulation (14) [20].

Furthermore, we observe that the bilinear form $a_{h,\mathcal{NN}}(\cdot,\cdot)$ is still symmetric positive definite with respect to the broken norm and that $a_{h,\mathcal{NN}}(\cdot,\cdot)$ has the trivial kernel, i.e. the constant functions, and reduces to $a(\cdot,\cdot)$ on V. Thus, using the Strang's Lemma [21] and the same steps developed in [10], we can deduce the following error bound:

$$\left\| u - u^{\mathcal{NN}}_h \right\|_{\mathcal{NN}} \leq C \left(h + h^{-2} \varepsilon \right) \tag{15}$$

where $\|\cdot\|_{\mathcal{NN}} = \sqrt{a_{h,\mathcal{NN}}(\cdot,\cdot)}$ and $\varepsilon = \max_{j=1,\dots,N^{\mathrm{dof}}} \max_{E \in \mathcal{T}_h} \epsilon_{j,E}$.

4. The neural network

In this section, we focus on the role of the involved neural networks. In particular, we describe the encoding of the input data (v_j, E) in Sections 4.1 and 4.2, the approximation space $\mathcal{H}^{\mathcal{NN}}_{j,E}$ in Section 4.3, and the architectures of the neural networks and the related training procedures in Sections 4.4 and 4.5. Finally, in Section 4.6, we summarize the online phase of the NAVEM approach.

Given the encoding of the pair (v_j, E), which represents the input for our neural network (9), the corresponding output is represented by the set of coefficients which express $\varphi^{\mathcal{NN}}_{j,E}$ and $\nabla \varphi^{\mathcal{NN}}_{j,E}$ with respect to the basis functions of $\mathcal{H}^{\mathcal{NN}}_{j,E}$ and $\nabla \mathcal{H}^{\mathcal{NN}}_{j,E}$. Since a neural network assumes that its input and output have constant dimensions, changing these cardinalities implies using a different neural network. In particular, we subdivide the polygons into different classes such that, in each class, all the polygons can be encoded into vectors of the same size and the corresponding basis functions can be approximated by exploiting the same number of harmonic functions.

4.1. Input encoding

As mentioned before, in order to predict the VEM basis functions, the first step is the encoding of the pair (v_j, E) in a vector $\boldsymbol{x}^{\mathrm{coef}}_0$ of a given dimension N^{coef}_0. The encoding of the input is performed in three consecutive steps: *polygon classification*, *variability reduction* and *input reduction*. The first one is a mandatory step, whereas the other two steps are useful to enhance the performance of the neural network but they could be omitted as well.

Moreover, although Assumption 1 allows for both convex and concave elements, in the following, we devise an input encoding strategy that is mainly devised for elements with a convex shape. The consideration of concave polytopes introduces a vast array of scenarios that fall beyond the scope of our analysis and may necessitate specialized, case-specific strategies. We precise that, given a suitable input encoding strategy for concave elements, the NAVEM method can be applied to meshes characterized by concave elements without further modifications.

The polygon classification step consists of subdividing the polygons into different classes. Pairs (v_j, E) related to polygons E belonging to different classes are encoded into vectors $\boldsymbol{x}^{\mathrm{coef}}_0$ of different dimensions. The general rule for the classification is that, if two polygons have different numbers of vertices N^v, then they belong to different classes. This trivial classification is dictated by our encoding of the pair (v_j, E) into

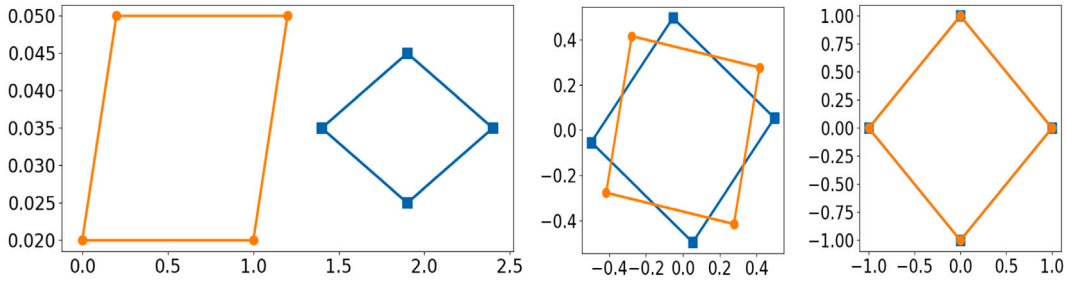


Fig. 1. Input encoding. Left: original elements. Center: variability reduction. Right: input reduction.

the vector $\boldsymbol{x}_0^{\text{coef}}$ whose dimension depends only on the number of the vertices N^v of the polygon.

Since the input dimension is fixed for a given neural network, we need to train a different neural network for each class of polygons and thus for each value of $N^v \geq 4$. We observe that, for $N^v = 3$, the lowest-order virtual element method coincides with the finite element method, eliminating the need for a neural network to access point-wise evaluation of the virtual basis functions.

In our framework, the only exception to the general rule of classification is represented by the case of triangles with hanging nodes, where we devise a different strategy to improve the method's accuracy given their importance in real-life applications [22,23]. We refer to Section 4.2 for the encoding of pairs (v_j, E) which are related to triangles with hanging nodes. However, we highlight that, as in the virtual element framework, a triangle with one hanging node could be classified as a quadrilateral, a triangle with two hanging nodes as well as a quadrilateral with one hanging node can be classified as a pentagon, and so on. This differentiation for the triangles with hanging nodes is only made to achieve very good accuracy with a very simple neural network architecture.

The variability reduction step is performed to reduce the variability of the elements in the datasets and enhance the neural networks' accuracy. For this purpose, we exploit the affine isomorphism defined in [24], mapping each element E in the dataset in a new polygon $\widehat{E} = F_E^{-1}(E)$ which is centred at the axes origin and has unit diameter and unit anisotropic ratio. In particular, the anisotropic ratio of an element E is here defined as the ratio between the maximum and the minimum eigenvalues of the inertia tensor of E. We recall that this map, in the absence of aligned or quasi-aligned edges, tends to uniform the elements within the same class in terms of their main geometric features, reducing the variability of the elements seen by the network.

The main role of the input reduction step is to shrink the dimension N_0^{coef} of $\boldsymbol{x}_0^{\text{coef}}$. For this purpose, assuming a polygon classification based on the number of vertices N^v, for each vertex v_j of $\widehat{E}$, we consider a second affine isomorphism $G_{j,\widehat{E}}^{-1}$ that scales and rotates the element $\widehat{E}$ into a polygon $\widetilde{E}_j$ such that the vertex v_j is mapped into the point $(1, 0)$. Since the coordinates of v_j are fixed, they can be excluded from the vector $\boldsymbol{x}_0^{\text{coef}}$. Thus, denoting by $(\widetilde{x}_1^r, \widetilde{x}_2^r)$ the coordinates of $G_{j,\widehat{E}}^{-1}(v_r)$, we define $\boldsymbol{x}_0^{\text{coef}} = \left[\widetilde{x}_1^{j+1} \quad \widetilde{x}_2^{j+1} \quad \dots \quad \widetilde{x}_1^{j+N^v-1} \quad \widetilde{x}_2^{j+N^v-1}\right] \in \mathbb{R}^{2(N^v-1)}$, where all indices j are intended up to module N^v. We observe that this map acts as a compression since the pair (v_j, E) is jointly encoded into a vector of size $N_0^{\text{coef}} = 2(N^v - 1)$, whereas a naive encoding would require a vector of size $2N^v + 1$, i.e. N^v x_1-coordinates and N^v x_2-coordinates plus the information about the index j.

We note that this type of input reduction also performs a variability reduction, even though the diameters of these elements are no longer exactly 1, but still scale as 1. Let us explain it with a very simple example. Let us consider two distinct parallelograms as shown in Fig. 1(a). The inertial mapping proposed in [5] maps these two parallelograms in the same square defined up to a rotation as noted in Fig. 1(b). Finally, the input reduction step fixes the rotation as highlighted in Fig. 1(c) transforming the two original elements into the same element.

4.2. Special case: triangles with hanging nodes

The ability to handle meshes with hanging nodes, especially triangular meshes characterized by a copious number of hanging nodes, is very important in many contexts such as discrete fracture networks [22] or adaptive strategies [23]. We highlight that, for such elements, the procedure described in Section 4.1 can be used but it may become very expensive or inaccurate when the number of hanging nodes grows. We thus decided to treat this case separately because of its importance in applications.

Let us consider an element E, which has the shape of a triangle and it is characterized by one or more hanging nodes. To encode the pair (v_j, E), let us now consider an element E' which is obtained from E by removing all the hanging nodes with the only exception of v_{j-1}, v_j and v_{j+1} if these vertices are hanging nodes. As usual, $v_{j-1} = v_{N^v-1}$ if $j = 0$ and $v_{j+1} = v_0$ if $j = N^v - 1$. We observe that in this way we obtain an element E' with at most six vertices: three vertices which define the shape of the triangle and at most three hanging nodes. Moreover, let us denote by φ_j' the VEM basis function associated with the vertex v_j but defined on E'. Since φ_j and φ_j' are the solution of the same Laplace problem, we can state that the removed hanging nodes do not contribute to define the shape of φ_j and they can thus be neglected.

In the NAVEM framework, the elimination of the hanging nodes that do not influence the function φ_j is very important to provide only useful information as the input of the neural network, limiting the input dimension and the number of possible configurations. Indeed, we observe that there exist only six different configurations, up to a reflection in the role of v_{j-1} and v_{j+1}, that is

1. v_{j-1} and v_j are vertices of the physical triangle and v_{j+1} is an hanging node;
2. v_{j-1} and v_{j+1} are vertices of the physical triangle and v_j is an hanging node;
3. v_{j-1} is a vertex of the physical triangle and v_j and v_{j+1} are hanging nodes;
4. v_{j-1}, v_j and v_{j+1} are hanging nodes;
5. v_j is a vertex of the physical triangle and v_{j-1} and v_{j+1} are hanging nodes;
6. v_{j-1}, v_j and v_{j+1} are vertices of the physical triangle.

Such configurations are summarized in Table 1. Furthermore, in order to perform variability and input reduction, we map the triangle E' into the equilateral triangle of vertices $\left\{(-1, 0), \left(0.5, -\frac{\sqrt{3}}{2}\right), \left(0.5, \frac{\sqrt{3}}{2}\right)\right\}$. For configurations 1, 2, 3, and 4 the triangle E' is rotated such that all the

Table 1
Existing configurations for triangles with hanging nodes. The letters V and H denote an actual vertex of the underlying physical triangle and a hanging node, respectively.

Configuration	v_{j-1}	v_j	v_{j+1}
1	V	V	H
2	V	H	V
3	V	H	H
4	H	H	H
5	H	V	H
6	V	V	V

hanging nodes are on the vertical edge, whereas in configuration 5 the vertex v_j is located at $\left(0.5, -\frac{\sqrt{3}}{2}\right)$.

To be as accurate as possible, for each configuration i, we train a neural network $\mathcal{NN}_{H,i}$, $i = 1, \dots, 5$, whereas for the sixth configuration, without loss of generality, we use the known finite element basis functions, i.e. we set

$$\varphi^{\mathcal{NN}}_{1,E}(x_1, x_2) = \frac{1}{3}\left(1 - 2x_1\right), \quad \varphi^{\mathcal{NN}}_{2,E}(x_1, x_2) = \frac{1}{3}(x_1 - \sqrt{3}x_2 + 1),$$

$$\varphi^{\mathcal{NN}}_{3,E}(x_1, x_2) = \frac{1}{3}\left(x_1 + \sqrt{3}x_2 + 1\right),$$

for the vertices $(-1, 0)$, $\left(0.5, -\frac{\sqrt{3}}{2}\right)$ and $\left(0.5, \frac{\sqrt{3}}{2}\right)$ of the equilateral triangle, respectively.

Furthermore, since the underlying physical triangle is always the same, we can avoid including the three vertices that define the shape of the triangle in the input for the network, leaving as the only inputs the curvilinear coordinates of the hanging nodes. We further note that the information about the index of the basis function is not included in the input since it is implicitly considered in the neural network configuration. Thus, the dimension of the input $\boldsymbol{x}_0^{\text{coef}}$ of the neural network $\mathcal{NN}_{H,i}$ is $N_0^{\text{coef}} = 1$ when $i = 1, 2$, $N_0^{\text{coef}} = 2$ when $i = 3, 5$ and $N_0^{\text{coef}} = 3$ when $i = 4$. We observe that these values are very small compared to $2(N_E^v - 1)$, which represents the input dimension when using the general encoding procedure shown in the previous section. It is important to highlight that N_E^v counts all the hanging nodes in E, which could be significantly much more than the ones in E'.

The possible configurations of elements E', up to rotation or reflection, and a more complex configuration are shown in Fig. 2, where we label each vertex with the different neural network that should be used to predict $\varphi^{\mathcal{NN}}_{j,E'}$.

4.3. The approximation spaces

Let us introduce a reference squared region $S^{\mathcal{NN}} = [-R^{\mathcal{NN}}, R^{\mathcal{NN}}]^2 \subset \mathbb{R}^2$, centred at the axes origin and with an edge length of $2R^{\mathcal{NN}}$. We define the harmonic polynomial space $\mathbb{H}_{\ell^{\mathcal{NN}}}(S^{\mathcal{NN}})$, consisting of all the harmonic polynomials of degree up to $\ell^{\mathcal{NN}} \geq 0$, as the span of the following scaled harmonic polynomial basis, i.e.

$$\left\{1,\ \Re\left(\left(\frac{z}{R^{\mathcal{NN}}}\right)^{\ell}\right), \Im\left(\left(\frac{z}{R^{\mathcal{NN}}}\right)^{\ell}\right),\ \ell = 1, \dots, \ell^{\mathcal{NN}}\right\} \tag{16}$$

where, for simplicity, we use the complex notation $z = x_1 + ix_2$ for each point $\boldsymbol{x} = \begin{bmatrix} x_1 & x_2 \end{bmatrix}^T \in \mathbb{R}^2$. We observe that the dimension of $\mathbb{H}_{\ell^{\mathcal{NN}}}(S^{\mathcal{NN}})$ is $\dim \mathbb{H}_{\ell^{\mathcal{NN}}}(S^{\mathcal{NN}}) = 2\ell^{\mathcal{NN}} + 1$. Furthermore, the harmonic scaled polynomials (16) and their gradients could be easily retrieved thanks to the recursive strategy presented in [25]. We then construct an orthonormal polynomial basis $\{\tilde{p}_\beta\}_{\beta=1}^{2\ell^{\mathcal{NN}}+1}$ for $\mathbb{H}_{\ell^{\mathcal{NN}}}(S^{\mathcal{NN}})$ by orthogonalizing the scaled polynomial basis (16) using the modified Gram-Schmidt algorithm applied twice to the Vandermonde matrix, whose columns contain the evaluations of the scaled polynomials at points forming a lattice built over $S^{\mathcal{NN}}$.

Next, we introduce a suitable harmonic function Φ which represents a least squares approximation of the solution $\tilde{\Phi}$ to the following Laplace problem

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

which is computed by exploiting a simplified version of the method presented in [11]. More precisely, we determine the set of coefficients

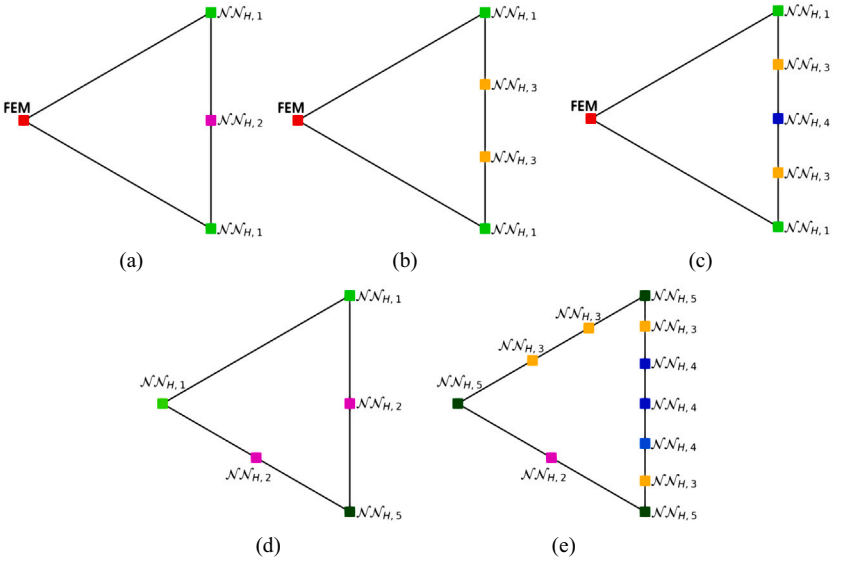
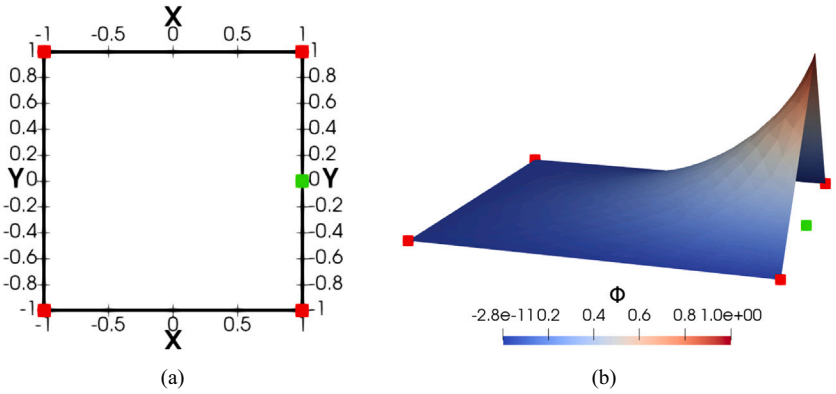


Fig. 2. Examples of triangles with hanging nodes. The colors and labels are associated with the different configurations. The required rotations and reflections are already taken into account.



(a) (b)

Fig. 3. Left: domain Ω_Φ of the function Φ. Right: shape of the function Φ.

$\left\{\{c_\alpha^1\}_{\alpha=1}^{N^1},\ \{c_\beta^2\}_{\beta=1}^{N^2}\right\}$ of the following linear combination of harmonic functions

$$\Phi(z) = \sum_{\alpha=1}^{N^1} c_\alpha^1 \Re\left(\frac{d_\alpha}{z - z_\alpha}\right) + \sum_{\beta=0}^{N^2} c_\beta^2 \Re\left(\left(\frac{z}{2}\right)^\beta\right), \tag{17}$$

which minimizes the distance between Φ and $\tilde{\Phi}$ at the boundary of the domain Ω_Φ, shown in Fig. 3(a). In Eq. (17), the points $z_\alpha = 1 + 2\exp\left(-4\left(\sqrt{N_1} - \sqrt{\alpha}\right)\right)$, $\alpha = 1, \dots, N^1$, represent N^1 poles exponentially distributed along the unit exterior angle bisector $\boldsymbol{b}_{\text{green}}$ at the green vertex $z_{\text{green}} = 1 + i0$ of the domain Ω_Φ, whereas $d_\alpha = |z_{\text{green}} - z_\alpha| = 2\exp\left(-4\left(\sqrt{N_1} - \sqrt{\alpha}\right)\right)$, for each $\alpha = 1, \dots, N^1$. Fig. 3(b) illustrates the shape of the function Φ obtained choosing $N^1 = 50$ and $N^2 = 25$.

Finally, for each polygon E and for each function $\varphi_{j,E}$, with $j = 1, \dots, N_E^{\text{dof}}$, we introduce the following approximation space:

$$\mathcal{H}_{j,E}^{\mathcal{NN}} = \text{span}\left\{\{\tilde{p}_\beta\}_{\beta=1}^{2\ell^{\mathcal{NN}}+1},\ \Phi_{j,E}^{j-1}, \Phi_{j,E}^{j}, \Phi_{j,E}^{j+1}\right\}, \tag{18}$$

where the three auxiliary functions $\Phi_{j,E}^{j-1}$, $\Phi_{j,E}^{j}$ and $\Phi_{j,E}^{j+1}$ are suitable mappings of the function Φ on the new domains $\Omega_{\Phi,j,E}^{j-1}$, $\Omega_{\Phi,j,E}^{j}$ and $\Omega_{\Phi,j,E}^{j+1}$, respectively. These new domains are obtained through an affine isomorphism which maps Ω_Φ into three corresponding squared regions $\Omega_{\Phi,j,E}^{i}$, for each $i = j-1, j, j+1$, defined such that $E \subset \Omega_{\Phi,j,E}^{i}$ and z_{green} is mapped into the ith vertex of E by aligning the exterior angle bisector $\boldsymbol{b}_{\text{green}}$ to the exterior angle bisector at the ith vertex of E. The whole procedure is outlined in Fig. 4.

We recall that, for each class of polygons, the number of outputs and thus the cardinality of the approximation space must be fixed. Nonetheless, the approximation space used to predict a basis function may differ from the approximation space used to predict another basis function, even if these basis functions are related to elements belonging to the same class. In particular, we note that all the approximation spaces related to the same class of polygons share the same set of harmonic polynomials $\{\tilde{p}_\beta\}_{\beta=1}^{2\ell^{\mathcal{NN}}+1}$, while the three auxiliary functions $\Phi_{j,E}^{j-1}$, $\Phi_{j,E}^{j}$ and $\Phi_{j,E}^{j+1}$ depend on the pair (v_j, E).

Since we fix the harmonic polynomial basis for a given neural network, $\ell^{\mathcal{NN}}$ and $R^{\mathcal{NN}}$ are constant across polygons belonging to the same class. In particular, $R^{\mathcal{NN}}$ is chosen in such a way these polynomials are well-scaled for each polygon E predicted with the same neural network.

We further highlight that this kind of construction allows us to orthogonalize the harmonic polynomials and compute the auxiliary function Φ just once.

Remark 1. Although the introduction of these functions $\Phi_{j,E}^{i}$, for $i = j-1, j, j+1$, could be tricky, it helps us to capture the singularities of the function $\varphi_{j,E}$ near the vertices of the polygon E, conducting the same task of the functions $\Re\left(\frac{d_\alpha}{z-z_\alpha}\right)$ in [11], $\alpha = 1, \dots, N^1$. In [11], these functions are introduced directly in the approximation space for all the vertices of E to approximate the single function $\varphi_{j,E}$, causing the number of the coefficients to predict to increase dramatically. For further details see Remark 3.

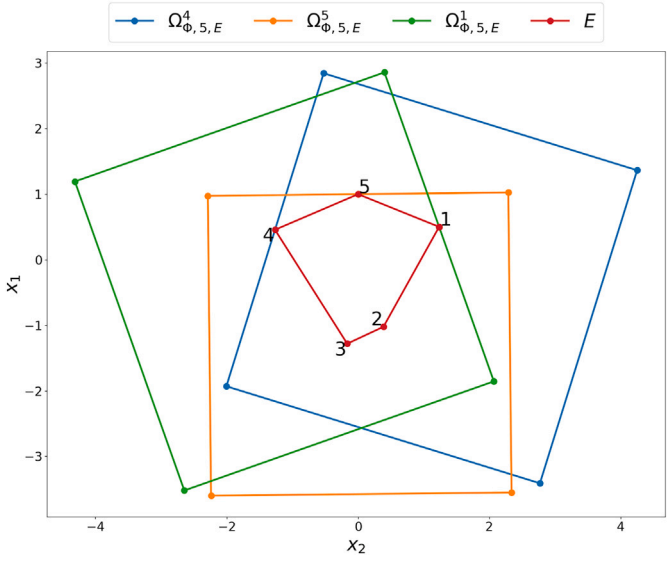


Fig. 4. Domain $\Omega_{\Phi,j,E}^{i}$ of the auxiliary function $\Phi_{j,E}^{i}$, for each $i = j-1, j, j+1$.

4.4. The neural network architecture and the training phase

As more deeply discussed in Section 4.1, the final input vector $\boldsymbol{x}_0^{\text{coef}}$ of the neural network represents the encoding of the polygon and the index of the VEM basis function that we are approximating.

Given an input pair (v_j, E) and its encoding $\boldsymbol{x}_0^{\text{coef}}$, let us consider the approximation space

$$\mathcal{H}_{j,E}^{\mathcal{NN}} = \text{span}\{h_k\}_{k=1}^{\dim \mathcal{H}_{j,E}^{\mathcal{NN}}} \tag{19}$$

which is built according to Section 4.3. To learn the non-linear map (9), we build a neural network that is a fully-connected feed-forward neural network implemented using standard routines available in TensorFlow [13]. It consists of multiple dense layers, where each neuron of a layer is connected to all neurons in the subsequent layer. More precisely, the architecture for the single input $\boldsymbol{x}_0^{\text{coef}}$ is:

$$
\begin{aligned}
&\boldsymbol{x}_0^{\varphi,\mathrm{coef}} := \boldsymbol{x}_0^{\mathrm{coef}},\\
&\boldsymbol{x}_\ell^{\varphi,\mathrm{coef}} = \rho(A_\ell^\varphi \boldsymbol{x}_{\ell-1}^{\varphi,\mathrm{coef}} + b_\ell^\varphi), \qquad \ell = 1,\dots,L-1,\\
&\mathbf{c}^\varphi(\boldsymbol{x}_0^{\mathrm{coef}}) = A_L^\varphi \boldsymbol{x}_{L-1}^{\varphi,\mathrm{coef}} + b_L^\varphi,
\end{aligned}
\tag{20}
$$

where

- L is the number of layers;
- the matrices and vectors $A_\ell^\varphi \in \mathbb{R}^{N_\ell \times N_{\ell-1}}$ and $b_\ell^\varphi \in \mathbb{R}^{N_\ell}$ store the network weights, for each $\ell = 1,\dots,L$. Several choices are available in the literature to initialize such weights. Here, we will employ the Glorot normal initialization [26];
- $\rho : \mathbb{R} \to \mathbb{R}$ is a non-linear activation function acting on its input vector component-wise, i.e. $\rho(\mathbf{y}) = \left[\rho(y_1),\dots,\rho(y_{n_y})\right]$ for any vector $\mathbf{y} \in \mathbb{R}^{n_y}$) [27]. Standard activation functions are, for example, $\rho(x) = \mathrm{ReLU}(x) = \max(0,x)$, $\rho(x) = \tanh(x)$ and $\rho(x) = 1/(1+e^{-x})$. In our numerical experiments, we always use the hyperbolic tangent $\rho(x) = \tanh(x)$ as the activation function;
- $\mathbf{c}^\varphi(\cdot)$ is the trainable multi-layer perceptron (MLP) (or fully-connected feed-forward neural network) [28,29]. Its value $\mathbf{c}^\varphi(\boldsymbol{x}_0^{\mathrm{coef}}) \in \mathbb{R}^{\dim \mathcal{H}_{j,E}^{\mathcal{NN}}}$ represents the vector of coefficients of the linear combination that expresses the function $\varphi_{j,E}^{\mathcal{NN}}$ with respect to the approximation basis functions belonging to $\mathcal{H}_{j,E}^{\mathcal{NN}}$.

Coherently with this structure, we approximate the value of the actual target function $\varphi_{j,E}$ in an evaluation point $\boldsymbol{x}_0^{\mathrm{fun}} \in E$ as

$$
\begin{aligned}
\varphi_{j,E}\left(\boldsymbol{x}_0^{\mathrm{fun}}\right) &\approx \varphi_{j,E}^{\mathcal{NN}}\left(\boldsymbol{x}_0^{\mathrm{fun}}\right) := \sum_{k=1}^{\dim \mathcal{H}_{j,E}^{\mathcal{NN}}} \mathbf{c}_k^\varphi\left(\boldsymbol{x}_0^{\mathrm{coef}}\right) h_k\left(\boldsymbol{x}_0^{\mathrm{fun}}\right),\\
\frac{\partial \varphi_{j,E}}{\partial x_1}\left(\boldsymbol{x}_0^{\mathrm{fun}}\right) &\approx \left(\boldsymbol{q}_{j,E}^{\mathcal{NN}}\right)_1\left(\boldsymbol{x}_0^{\mathrm{fun}}\right) := \sum_{k=1}^{\dim \mathcal{H}_{j,E}^{\mathcal{NN}}} \mathbf{c}_k^\varphi\left(\boldsymbol{x}_0^{\mathrm{coef}}\right) \frac{\partial h_k}{\partial x_1}\left(\boldsymbol{x}_0^{\mathrm{fun}}\right),\\
\frac{\partial \varphi_{j,E}}{\partial x_2}\left(\boldsymbol{x}_0^{\mathrm{fun}}\right) &\approx \left(\boldsymbol{q}_{j,E}^{\mathcal{NN}}\right)_2\left(\boldsymbol{x}_0^{\mathrm{fun}}\right) := \sum_{k=1}^{\dim \mathcal{H}_{j,E}^{\mathcal{NN}}} \mathbf{c}_k^\varphi\left(\boldsymbol{x}_0^{\mathrm{coef}}\right) \frac{\partial h_k}{\partial x_2}\left(\boldsymbol{x}_0^{\mathrm{fun}}\right).
\end{aligned}
\tag{21}
$$

We observe that, with this kind of construction, we are setting

$$
\boldsymbol{q}_{j,E}^{\mathcal{NN}}\left(\boldsymbol{x}_0^{\mathrm{fun}}\right) = \begin{bmatrix} \left(\boldsymbol{q}_{j,E}^{\mathcal{NN}}\right)_1\left(\boldsymbol{x}_0^{\mathrm{fun}}\right) \\ \left(\boldsymbol{q}_{j,E}^{\mathcal{NN}}\right)_2\left(\boldsymbol{x}_0^{\mathrm{fun}}\right) \end{bmatrix} = \nabla \varphi_{j,E}^{\mathcal{NN}}\left(\boldsymbol{x}_0^{\mathrm{fun}}\right).
$$

Finally, the weights characterizing the architecture (20) are optimized in the following way. For each class of polygons $\mathcal{C}$, we denote by $\mathcal{PC}^{\mathrm{train}}$ the training set consisting of pair (v_j, E) with elements E belonging to $\mathcal{C}$ that we are interested in. We train the related neural network to minimize the following L^2-loss function

$$
\begin{aligned}
\mathcal{L}_0 &= \frac{1}{\#\mathcal{PC}^{\mathrm{train}}} \sum_{(v_j,E)\in\mathcal{PC}^{\mathrm{train}}} \left\| \varphi_{j,E} - \varphi_{j,E}^{\mathcal{NN}} \right\|_{L^2(\partial E)}^2\\
&\approx \frac{1}{\#\mathcal{PC}^{\mathrm{train}}} \sum_{(v_j,E)\in\mathcal{PC}^{\mathrm{train}}} \sum_{q=1}^{N^q} \omega_q \left(\varphi_{j,E}(\boldsymbol{x}_q) - \varphi_{j,E}^{\mathcal{NN}}(\boldsymbol{x}_q) \right)^2
\end{aligned}
\tag{22}
$$

where $\{(\omega_q, \boldsymbol{x}_q)\}_{q=1}^{N^q}$ is an appropriate quadrature formula on ∂E. To improve generalization and prevent overfitting, we add a regularization term to the loss function (22) that penalizes the L^2-norm of the trainable coefficients. We highlight that such optimization procedure is performed through very efficient optimizers already available in the TensorFlow library, further details are provided in Section 5.

4.5. An effective variant for the training strategy

We must observe that the quadrature rule used to estimate the loss function (22) and the underlying non-linear optimization process may lead to small oscillations in the function $\varphi_{j,E}^{\mathcal{NN}}$, which may result in a poor approximation $\boldsymbol{q}_{j,E}^{\mathcal{NN}}$ for the gradient of the VEM basis functions $\nabla\varphi_{j,E}$. Thus, we decide to employ a second neural network with output $\boldsymbol{q}_{j,E}^{\mathcal{NN}}$ to approximate the gradient of $\varphi_{j,E}$, even if in this way we introduce a new consistency error because $\boldsymbol{q}_{j,E}^{\mathcal{NN}}$ is, in general, different from $\nabla\varphi_{j,E}^{\mathcal{NN}}$.

The whole architecture of the involved neural networks for a single input $\boldsymbol{x}_0^{\mathrm{coef}}$ is:

$$
\begin{aligned}
&\boldsymbol{x}_0^{\varphi,\mathrm{coef}} = \boldsymbol{x}_0^{\mathrm{coef}},\\
&\boldsymbol{x}_\ell^{\varphi,\mathrm{coef}} = \rho\left(A_\ell^\varphi \boldsymbol{x}_{\ell-1}^{\varphi,\mathrm{coef}} + b_\ell^\varphi\right), \qquad \ell = 1,\dots,L-1,\\
&\mathbf{c}^\varphi\left(\boldsymbol{x}_0^{\mathrm{coef}}\right) = A_L^\varphi \boldsymbol{x}_{L-1}^{\varphi,\mathrm{coef}} + b_L^\varphi,\\
&\boldsymbol{x}_0^{\boldsymbol{q},\mathrm{coef}} = \boldsymbol{x}_0^{\mathrm{coef}},\\
&\boldsymbol{x}_\ell^{\boldsymbol{q},\mathrm{coef}} = \rho\left(A_\ell^{\boldsymbol{q}} \boldsymbol{x}_{\ell-1}^{\boldsymbol{q},\mathrm{coef}} + b_\ell^{\boldsymbol{q}}\right), \qquad \ell = 1,\dots,L-1,\\
&\mathbf{c}^{\boldsymbol{q}}\left(\boldsymbol{x}_0^{\mathrm{coef}}\right) = A_L^{\boldsymbol{q}} \boldsymbol{x}_{L-1}^{\boldsymbol{q},\mathrm{coef}} + b_L^{\boldsymbol{q}}.
\end{aligned}
\tag{23}
$$

With this new architecture, given a pair (v_j, E) encoded in $\boldsymbol{x}_0^{\mathrm{coef}}$ and an evaluation point $\boldsymbol{x}_0^{\mathrm{fun}} \in E$, we approximate

$$
\begin{aligned}
\varphi_{j,E}\left(\boldsymbol{x}_0^{\mathrm{fun}}\right) &\approx \varphi_{j,E}^{\mathcal{NN}}\left(\boldsymbol{x}_0^{\mathrm{fun}}\right) := \sum_{k=1}^{\dim \mathcal{H}_{j,E}^{\mathcal{NN}}} \mathbf{c}_k^\varphi\left(\boldsymbol{x}_0^{\mathrm{coef}}\right) h_k\left(\boldsymbol{x}_0^{\mathrm{fun}}\right),\\
\frac{\partial \varphi_{j,E}}{\partial x_1}\left(\boldsymbol{x}_0^{\mathrm{fun}}\right) &\approx \left(\boldsymbol{q}_{j,E}^{\mathcal{NN}}\right)_1\left(\boldsymbol{x}_0^{\mathrm{fun}}\right) := \sum_{k=1}^{\dim \mathcal{H}_{j,E}^{\mathcal{NN}}} \mathbf{c}_k^{\boldsymbol{q}}\left(\boldsymbol{x}_0^{\mathrm{coef}}\right) \frac{\partial h_k}{\partial x_1}\left(\boldsymbol{x}_0^{\mathrm{fun}}\right),\\
\frac{\partial \varphi_{j,E}}{\partial x_2}\left(\boldsymbol{x}_0^{\mathrm{fun}}\right) &\approx \left(\boldsymbol{q}_{j,E}^{\mathcal{NN}}\right)_2\left(\boldsymbol{x}_0^{\mathrm{fun}}\right) := \sum_{k=1}^{\dim \mathcal{H}_{j,E}^{\mathcal{NN}}} \mathbf{c}_k^{\boldsymbol{q}}\left(\boldsymbol{x}_0^{\mathrm{coef}}\right) \frac{\partial h_k}{\partial x_2}\left(\boldsymbol{x}_0^{\mathrm{fun}}\right).
\end{aligned}
\tag{24}
$$

In this case, we initialize the weights of the first neural network $\mathbf{c}^\varphi$ as before and optimize them by minimizing the L^2-loss function (22), whereas the weights of the second neural network $\mathbf{c}^{\boldsymbol{q}}$ are initialized with the final ones of the first network and fine-tuned to minimize the following H^1-loss function

$$
\mathcal{L}_1 = \frac{1}{\#\mathcal{PC}^{\mathrm{train}}} \sum_{(v_j,E)\in\mathcal{PC}^{\mathrm{train}}} \left(\mathcal{L}_{j,E}^1\right)^2,
\tag{25}
$$

where

$$
\mathcal{L}_{j,E}^1 = \left\| \boldsymbol{q}_{j,E}^{\mathcal{NN}} \cdot \boldsymbol{t} - \nabla\varphi_{j,E} \cdot \boldsymbol{t} \right\|_{L^2(\partial E)}.
\tag{26}
$$

As in Section 4.4, $\mathcal{L}_1$ is computed using suitable quadrature formulas.

Remark 2. We remark that, to avoid loss in the accuracy of the approximation for $\nabla\varphi_{j,E}$, a different training procedure is proposed in [12], where a single neural network which minimizes a suitable combination of $\mathcal{L}_0$ and $\mathcal{L}_1$ at the same time is introduced. Such alternative formulation can be used without additional technical complexities, but we do not focus deeply on it since we observed that it is sub-optimal in the presence of very small edges.

Remark 3. To highlight the importance of introducing the auxiliary functions $\Phi_{j,E}^i$ into the approximation space, we minimize the loss functions (22) and (25), both with and without using these auxiliary functions. In the first case, we train the neural networks using an approximation space consisting solely of harmonic polynomials, i.e. we set $\mathcal{H}_{j,E}^{\mathcal{NN}} = \{\tilde{p}_\beta\}_{\beta=1}^{2\ell^{\mathcal{NN}}+1}$, with $\ell^{\mathcal{NN}} = 20$ and $R^{\mathcal{NN}} = 3$. In the second case, we extend this space by including the auxiliary functions as described in Eq. 18. In each case, we consider neural networks with 3 layers and 30 neurons per layer.

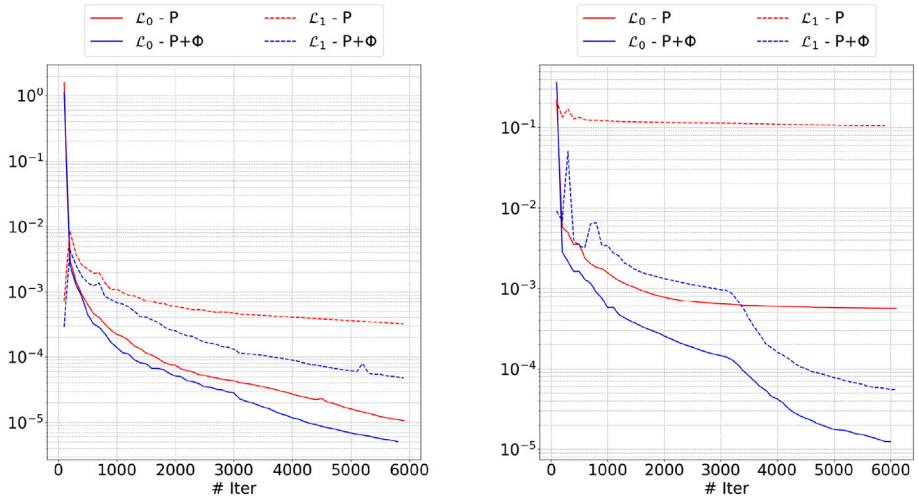


Fig. 5. Behaviour of the training loss functions (22) and (25) as the number of training iterations increases. The red lines are related to losses obtained by setting the approximation space $\mathcal{H}_{j,E}^{\mathcal{NN}} = \{\tilde{p}_\beta\}_{\beta=1}^{2\ell^{\mathcal{NN}}+1}$ (P), whereas the blue lines correspond to the approximation space defined in Eq. 18 (P + Φ). Left: Quadrilaterals. Right: Triangles with hanging nodes.

We also consider two different training datasets: the first one is made up of 10 random quadrilaterals, while the second one is designed to predict the Lagrange basis function $\varphi_{j,E}$ related to the configuration 2 of triangular elements with hanging nodes, presented in the previous section. Specifically, for the second test, we use a training dataset consisting of 10 copies of the reference triangle T_i with one hanging node H_i, whose position $\boldsymbol{x}(H_i)$ is given by

$$\boldsymbol{x}(H_i) = \left(0.5, -\frac{\sqrt{3}}{2}\right) + \frac{i}{11}\left(0, \sqrt{3}\right) \quad i = 1, \dots, 10.$$

In Fig. 5, we show the behaviours of loss functions (22) and (25) as the number of training iterations increases. Although the usage of only harmonic polynomials is sufficient (but suboptimal) to guarantee good results for quadrilaterals, such approximation space is not rich enough to approximate the basis functions on triangles with hanging nodes. Indeed, in this latter case, we are not able to retrieve the desired accuracy for the H^1-loss function (25). These results prove the importance of adding the auxiliary functions to the approximation space, enabling the accurate approximation of the virtual element Lagrange basis functions and their derivatives.

Note that the vector of functions $\boldsymbol{q}_{j,E}^{\mathcal{NN}}$ can be seen as the gradient of a function $\tilde{\varphi}_{j,E}^{\mathcal{NN}}$, which possibly differs from $\varphi_{j,E}^{\mathcal{NN}}$, defined through the minimization of the loss (22). A theoretical justification for the definition of the loss function (25) is offered by the following proposition, which shows that if two harmonic functions share the same tangential derivatives at the boundary ∂E, then they have the same gradient inside E.

Lemma 1. *Let us assume that u and v are two harmonic functions with the same tangential derivatives on ∂E. Then,*

$$\nabla u = \nabla v \quad \text{in } E. \tag{27}$$

Proof. Since u and v are harmonic and share the same tangential derivatives, the function $w = u - v$ is such that

$$\Delta w = 0 \quad \text{in } E, \quad \nabla_t w = \boldsymbol{0} \quad \text{on } \partial E,$$

where $\nabla_t w = \nabla w - (\nabla w \cdot \boldsymbol{n})\boldsymbol{n}$ is the tangential derivative of w. This implies that w is constant on the boundary ∂E and, since w is harmonic, we can state that w is constant everywhere on E. Thus,

$$\nabla w = 0 \quad \text{in } E \quad \Rightarrow \quad \nabla u = \nabla v \quad \text{in } E.$$

□

We now prove that minimizing a loss function of the form (25) ensures a good approximation of the gradient also inside the polygons. Note that this property is crucial since the integral forms involved in the PDEs are evaluated through quadrature rules with nodes inside the elements.

Proposition 2. *It holds*

$$\left\| \boldsymbol{q}_{j,E}^{\mathcal{NN}} - \nabla \varphi_{j,E} \right\|_{L^2(E)} \leq C \mathcal{L}_{j,E}^1, \tag{28}$$

where C depends on the polygon E.

Proof. We assume that $h_E = 1$, since the loss function (26) is computed on mapped elements which scale as 1. Let us set $\boldsymbol{q}_{j,E}^{\mathcal{NN}} = \nabla \tilde{\varphi}_{j,E}^{\mathcal{NN}}$ and $\tilde{\Psi}_{j,E} = \varphi_{j,E} - \tilde{\varphi}_{j,E}^{\mathcal{NN}}$. Since $\tilde{\varphi}_{j,E}^{\mathcal{NN}}$ is defined up to a constant, we define it in such a way $\tilde{\Psi}_{j,E}$ has zero mean value, so that the second Poincaré inequality holds [30]:

$$\left\| \tilde{\Psi}_{j,E} \right\|_{L^2(E)} \leq C \left\| \nabla \tilde{\Psi}_{j,E} \right\|_{L^2(E)}.$$

From the trace theorem, we have

$$\left\| \tilde{\Psi}_{j,E} \right\|_{L^2(\partial E)} \leq C \left\| \nabla \tilde{\Psi}_{j,E} \right\|_{L^2(E)}.$$

Furthermore, since $\tilde{\Psi}_{j,E}$ is harmonic, we obtain

$$\begin{aligned} \left\| \nabla \tilde{\Psi}_{j,E} \right\|_{L^2(E)}^2 &= \int_E \nabla \tilde{\Psi}_{j,E} \cdot \nabla \tilde{\Psi}_{j,E} = \int_{\partial E} \tilde{\Psi}_{j,E} \nabla \tilde{\Psi}_{j,E} \cdot \boldsymbol{n} \\ &\leq \left\| \tilde{\Psi}_{j,E} \right\|_{L^2(\partial E)} \left\| \nabla \tilde{\Psi}_{j,E} \cdot \boldsymbol{n} \right\|_{L^2(\partial E)} \\ &\leq C \left\| \nabla \tilde{\Psi}_{j,E} \right\|_{L^2(E)} \left\| \nabla \tilde{\Psi}_{j,E} \cdot \boldsymbol{n} \right\|_{L^2(\partial E)}. \end{aligned}$$

Thus,

$$\left\| \nabla \tilde{\Psi}_{j,E} \right\|_{L^2(E)} \leq C \left\| \nabla \tilde{\Psi}_{j,E} \cdot \boldsymbol{n} \right\|_{L^2(\partial E)}.$$

Now, we recall that the $L^2(\partial E)$-norm of the tangential derivatives is equivalent to the $L^2(\partial E)$-norm of the normal derivative for harmonic

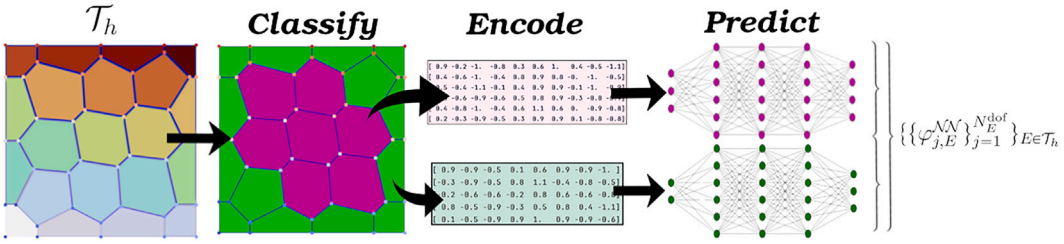


Fig. 6. A sketch of the NAVEM online phase.

functions, as a consequence of the Rellich's identity [31]. Thus, we can conclude that

$$\begin{aligned}\left\|\boldsymbol{q}_{j,E}^{\mathcal{NN}}-\nabla\varphi_{j,E}\right\|_{L^2(E)} &= \left\|\nabla\tilde{\Psi}_{j,E}\right\|_{L^2(E)}\\ &\le C\left\|\tilde{\Psi}_{j,E}\cdot\boldsymbol{n}\right\|_{L^2(\partial E)}\\ &\le C\left\|\tilde{\Psi}_{j,E}\cdot\boldsymbol{t}\right\|_{L^2(\partial E)} = C\left\|\boldsymbol{q}_{j,E}^{\mathcal{NN}}\cdot\boldsymbol{t}-\nabla\varphi_{j,E}\cdot\boldsymbol{t}\right\|_{L^2(\partial E)}\\ &= C\mathcal{L}_{j,E}^1.\end{aligned}$$

□

4.6. The NAVEM online phase

In this section, we outline the main steps of the NAVEM online phase, which are also summarized in Fig. 6.

In the online phase, given the tessellation $\mathcal{T}_h$, for each element $E\in\mathcal{T}_h$ and for each of $j=1,\dots,N_E^{\text{dof}}$, we

- **S.1** *classify* the pair (v_j,E) according to Section 4.1,
- **S.2** *encode* the information (v_j,E) to generate the input $\boldsymbol{x}_0^{\text{coef}}$ of the neural network,
- **S.3** *predict* the coefficients of the corresponding NAVEM basis function and its derivatives with respect to functions contained in the related approximation space $\mathcal{H}_{j,E}^{\mathcal{NN}}$, as reported in Eq. 23.

Given these coefficients, the NAVEM basis functions can be evaluated at each point of the domain Ω as in Eq. 24. Finally, to solve Eq. 14 we can proceed following a standard finite element solving strategy. That is, we

- **S.4** *assemble* the stiffness matrix and the right-hand side

$$\mathbf{A}^{\mathcal{NN}}\in\mathbb{R}^{N^{\text{dof}}\times N^{\text{dof}}}:\ \mathbf{A}_{ij}^{\mathcal{NN}}=a_{h,\mathcal{NN}}\left(\varphi_i^{\mathcal{NN}},\varphi_j^{\mathcal{NN}}\right)\ \forall i,j=1,\dots,N^{\text{dof}},$$

$$\mathbf{f}^{\mathcal{NN}}\in\mathbb{R}^{N^{\text{dof}}}:\quad \mathbf{f}_i^{\mathcal{NN}}=\left(f,\varphi_i^{\mathcal{NN}}\right)_\Omega\quad\forall i=1,\dots,N^{\text{dof}},$$

  by computing these integrals evaluating the NAVEM basis functions $\varphi_i^{\mathcal{NN}}$ and their gradients $\boldsymbol{q}_i^{\mathcal{NN}}$ at the quadrature points,
- **S.5** *solve* the system $\mathbf{A}^{\mathcal{NN}}\boldsymbol{u}^{\mathcal{NN}}=\mathbf{f}^{\mathcal{NN}}$, where $\boldsymbol{u}^{\mathcal{NN}}$ represents the vector containing the degrees of freedom related to the solution function $u_h^{\mathcal{NN}}\in V_{h,1}^{\mathcal{NN}}$.

We note that the output of the predict phase **S.3**, can be reused to post-process the solution.

Remark 4. We observe that, even if we are formally using rational functions as well as polynomials, we compute the integrals in Eq. (14) employing the standard Gauss quadrature formula used in the lowest-order virtual element framework, which is exact for integrating polynomial functions of degree up to 2. We further note that we are committing a second variational crime using this formula, but we decide to use it since we find out that its employment does not limit the convergence of the method. Other quadrature formulas could be employed to compute the integrals in Eq. (14), which are much more suitable for both rational functions and polynomials [32]. However, the discussion about the errors introduced by quadrature formulas is beyond the scope of this work.

Remark 5. Extension to the high-order For $k\ge 1$, the virtual element local space is given by Refs [1]

$$V_{h,k}(E)=\left\{v\in H^1(E):\quad (i)\ \Delta v\in\mathbb{P}_{k-2}(E),\quad (ii)\ v_{|\partial E}\in\mathbb{B}_k(\partial E)\right\}\tag{29}$$

whereas the local degrees of freedom are defined as

- the values of v at the vertices of E;
- the values of v at the $k-1$ Gauss-Lobatto quadrature points internal to each edge of E;
- the internal scaled moments: $\frac{1}{|E|}\int_E vp$ for all polynomial $p\in\mathbb{P}_{k-2}(E)$.

Given these definitions, it is immediate to see that each virtual function $v\in V_{h,k}(E)$ can be rewritten as the sum of two functions $v:=v_0+v_L$ which satisfy

$$\begin{cases}\Delta v_0=0 & \text{in } E,\\ v_0=v & \text{on } \partial E,\end{cases}\qquad\begin{cases}\Delta v_L=f & \text{in } E,\\ v_L=0 & \text{on } \partial E.\end{cases}\tag{30}$$

where $f:=\Delta v$ is a known polynomial of degree $k-2$. In particular, if φ^e is a Lagrange VEM basis function related to a boundary degree of freedom, then $\varphi^e\equiv\varphi_0^e$, whereas if we consider an internal basis function φ^I, then $\varphi^I\equiv\varphi_L^I$. We remark that the strategy proposed here for the lowest-order basis functions allows us also to approximate the basis functions related to boundary degrees of freedom for each order $k\ge 1$. Moreover, a similar strategy may be pursued to train a neural network that defines an approximation $\varphi^{I,\mathcal{NN}}$ of a basis function φ^I related to an internal degree of freedom, by introducing a further contribution in the loss function which aims to minimize the distance between $\Delta\varphi^{I,\mathcal{NN}}$ and the known value $\Delta\varphi^I$.

5. Numerical results

In this section, we perform three numerical experiments to validate our procedure on different families of polygonal meshes and for different kinds of partial differential equations. In particular, after evaluating the performance of our method on a general advection-diffusion-reaction problem, we investigate its behaviour in solving anisotropic problems and non-linear problems, where the use of a stabilizing form or polynomial projectors may limit the performance of the method.

Denoting by u the exact solution of the underlying problem, for each family of meshes, we test the performance of the NAVEM by looking at the behaviour of the following errors

$$\text{err}_0^{\mathcal{NN}}=\sqrt{\sum_{E\in\mathcal{T}_h}\left\|u-u_h^{\mathcal{NN}}\right\|_{0,E}^2},\quad \text{err}_1^{\mathcal{NN}}=\sqrt{\sum_{E\in\mathcal{T}_h}\left\|\nabla u-\nabla u_h^{\mathcal{NN}}\right\|_{0,E}^2},\tag{31}$$

as the family mesh parameter h decreases.

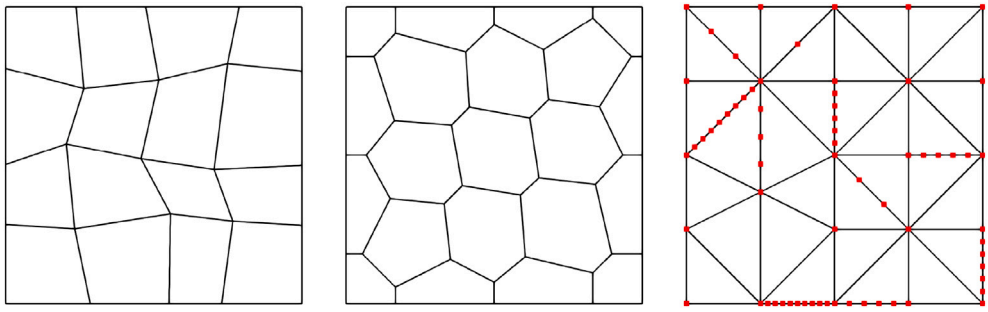


Fig. 7. Left: first mesh of RDQM family. Center: first mesh of VM family. Right: first mesh of HTM family. The red dots denote the hanging nodes.

We further compare the performance of our method with the standard VEM method which is available in the literature for the corresponding problem. Since we consider problems with variable coefficients, in the following numerical experiments, we will adopt the virtual element discretization introduced in [2] for the linear case, which is based on the definition of an *enhanced* space. This alternative formulation allows us to compute the L^2-projections of the virtual element functions on polynomial spaces of higher polynomial degrees in order to avoid loss of accuracy in the presence of variable coefficients. Thus, in the following, for each $k \geq 0$ and $E \in \mathcal{T}_h$, we will denote by $\Pi_k^{E,0} : V_{h,1}(E) \to \mathbb{P}_k(E)$ the L^2-polynomial projection of virtual element functions. Furthermore, without loss of generality, we use the same symbol also to denote the L^2 polynomial projection of vector-valued functions. In the following, we further employ the standard *dofi-dofi* stabilization term (7) as a stabilizing form for the virtual element method since we deal with the lowest-order discretization. Moreover, since we cannot access to the point-wise evaluation of virtual functions, we define the VEM errors as usual, that is

$$\mathrm{err}_0^{\mathrm{VEM}} = \sqrt{\sum_{E\in\mathcal{T}_h} \left\| u - \Pi_1^{E,0} u_h^{\mathrm{VEM}} \right\|_{0,E}^2},$$

$$\mathrm{err}_1^{\mathrm{VEM}} = \sqrt{\sum_{E\in\mathcal{T}_h} \left\| \nabla u - \Pi_0^{E,0} \nabla u_h^{\mathrm{VEM}} \right\|_{0,E}^2}. \tag{32}$$

To train the neural networks, we use a combination of the ADAM optimizer [33] with the BFGS optimizer [34] to optimize the weights of the neural networks. Moreover, to avoid problems related to overfitting, it is often advisable to add a regularization term to the loss function. We thus adopt a standard regularization technique penalizing the L^2-norm of the trainable coefficients and we set the regularization coefficients to 10^{-8}.

Finally, for simplicity, we choose $\ell^{\mathcal{NN}} = 20$ and $R^{\mathcal{NN}} = 3$ for all the neural networks. A better fine-tuning strategy of the parameter $\ell^{\mathcal{NN}}$ could be performed to further improve the efficiency of the method, reducing the number of function evaluations, but this is beyond the scope of the manuscript.

5.1. Meshes and training sets

In this section, we describe the three families of meshes used in the numerical experiments and the training sets that we use to train the related neural networks. Each family of meshes is made up of four meshes with decreasing mesh parameters h. The first mesh for each family is shown in Fig. 7.

Random distorted quadrilateral meshes (RDQM). The first family of meshes is obtained starting from a family of Cartesian meshes made up of identical squares. The vertices of such Cartesian meshes are then randomly perturbed to generate meshes with random quadrilaterals, as the one represented in Fig. 7(a). For this kind of mesh, a single neural network $\mathcal{NN}_4$ is needed since all the polygons have the same number of vertices. More precisely, two neural networks are trained to minimize the loss functions (22) and (25), respectively, over a training set of 1000 convex quadrilaterals which are randomly generated through the Python library polygenerator. The chosen neural network architectures comprise 4 hidden layers with 40 neurons in each layer.

Voronoi meshes (VM). The second family of meshes is a set of Voronoi meshes. The coarsest mesh is shown in Fig. 7(b). The elements included in these meshes are convex quadrilaterals, pentagons, hexagons and heptagons. For the quadrilaterals, we use the neural networks $\mathcal{NN}_4$ trained for the family RDQM, whereas for the other polygons, we train three networks $\mathcal{NN}_{V,i}$, with $i = 5, 6, 7$, all with 5 layers and 50 neurons in each layer. The training sets are obtained starting from different refinements of Voronoi meshes and then splitting the elements into different training sets according to the number of their vertices.

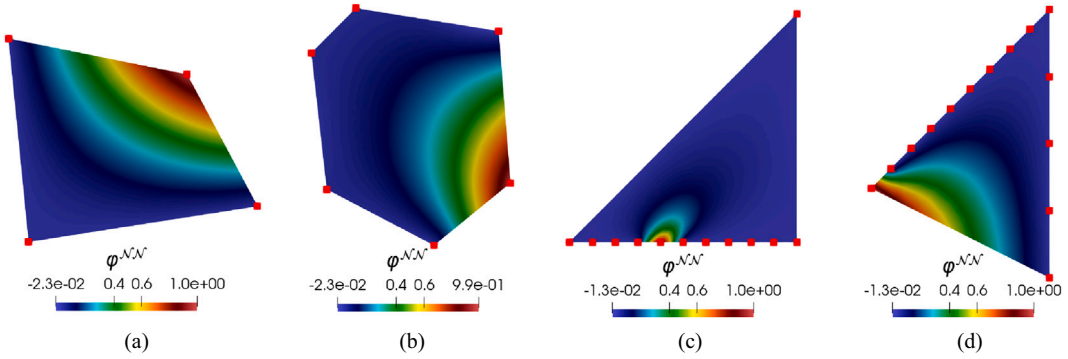
Triangular meshes with hanging nodes (HTM). The last family consists of a set of four triangular meshes with hanging nodes. These meshes are generated starting from standard triangular meshes and randomly selecting a subset of edges to which we add a random number between 1 and 10 of equispaced hanging nodes. The coarsest mesh is shown in Fig. 7(c), where the red dots represent the hanging nodes. As described in Section 4.2, we train 5 neural networks to approximate the basis functions related to this kind of mesh. Thus, we consider a set of identical neural networks comprising 5 hidden layers with 50 neurons in each layer. For each neural network, we construct the corresponding training dataset by adding 1, 2 or 3 hanging nodes (depending on the configuration) on the suitable edges of the reference equilateral triangle, as described in Section 4.2. We choose 10^{-2} as the minimum distance allowed between two consecutive vertices.

Table 2 summarizes the main geometric properties of the elements belonging to the training and test meshes and reports the square root of the related losses (22) and (25). In particular, we report statistics about the area, the diameter, the anisotropic ratio and the edge ratio, which is given by the ratio between the maximum and minimum lengths of the edges of the elements. These geometric properties are reported for both the original E and the mapped elements $\widehat{E}$. We recall that the mapped elements for the $RDQM$ and VM are the elements obtained through the inertial mapping introduced in [5], while all the physical triangles related to the HTM family are mapped to the same reference equilateral triangle.

Concerning the test and training meshes related to the families $RDQM$ and VM, which are grouped in $\mathcal{T}^{\mathrm{test}}_{RDQM}$-$\mathcal{T}^{\mathrm{train}}_{RDQM}$ and $\mathcal{T}^{\mathrm{test}}_{VM}$-$\mathcal{T}^{\mathrm{train}}_{VM}$, respectively, we can note that, thanks to the inertial mapping, we are able to strongly reduce the variability of the elements seen by the neural network obtaining elements with approximately the same area, with unit diameter and with unit anisotropic ratio. Concerning the test and the training sets for the HTM family, which are grouped in $\mathcal{T}^{\mathrm{test}}_{HTM}$-$\mathcal{T}^{\mathrm{train}}_{HTM}$,

Table 2
Geometric properties of training and test meshes used throughout the experiments.

	# {E}	N_v^E			Area		Diameter		Anisotropic ratio		Edge ratio		$\sqrt{\mathcal{L}_0}$	$\sqrt{\mathcal{L}_1}$
		min	max		min	max	min	max	min	max	min	max	avg	avg
$\mathcal{T}^{\text{test}}_{RDQM}$	736	4	4	E	1.20e−03	9.27e−02	5.32e−02	4.46e−01	1.02e+00	4.10e+00	1.05e+00	2.43e+00	2.00e−03	4.18e−03
				$\hat{E}$	4.69e−01	5.00e−01	1.00e+00	1.00e+00	1.00e+00	1.00e+00	1.01e+00	2.30e+00		
$\mathcal{T}^{\text{test}}_{VM}$	1054	5	7	E	3.09e−04	7.62e−02	2.84e−02	3.68e−01	1.00e+00	6.09e+00	1.00e+00	9.04e+00	6.91e−03	1.96e−02
				$\hat{E}$	4.97e−01	7.10e−01	1.00e+00	1.00e+00	1.00e+00	1.00e+00	1.00e+00	4.53e+00		
$\mathcal{T}^{\text{test}}_{HTM}$	542	3	20	E	1.90e−03	4.69e−02	7.11e−02	3.75e−01	1.00e+00	6.84e+00	1.00e+00	1.79e+01	1.52e−03	4.69e−03
				$\hat{E}$	1.30e+00	1.30e+00	1.73e+00	1.73e+00	1.00e+00	1.00e+00	1.00e+00	1.00e+01		
$\mathcal{T}^{\text{train}}_{RDQM}$	1000	4	4	E	4.03e−02	9.02e−01	1.00e+00	1.41e+00	1.01e+00	9.37e+02	1.04e+00	5.13e+01	4.62e−03	1.00e−02
				$\hat{E}$	4.34e−01	5.00e−01	1.00e+00	1.00e+00	1.00e+00	1.00e+00	1.01e+00	4.24e+01		
$\mathcal{T}^{\text{train}}_{VM}$	18,701	4	7	E	1.97e−05	9.02e−01	7.14e−03	1.41e+00	1.00e+00	9.37e+02	1.00e+00	5.13e+01	2.44e−03	5.79e−03
				$\hat{E}$	4.34e−01	7.14e−01	1.00e+00	1.00e+00	1.00e+00	1.00e+00	1.00e+00	4.24e+01		
$\mathcal{T}^{\text{train}}_{HTM}$	7670	4	6	E	1.30e+00	1.30e+00	1.73e+00	1.73e+00	1.00e+00	1.00e+00	2.02e+00	5.61e+02	4.19e−03	1.65e−02
				$\hat{E}$	1.30e+00	1.30e+00	1.73e+00	1.73e+00	1.00e+00	1.00e+00	2.02e+00	5.61e+02		



(a) (b) (c) (d)

Fig. 8. Contour plots of some basis functions predicted by the neural network on test elements. The red dots denote the vertices of the polygons.

we can see that the values for the area, the diameter, and the anisotropic ratio are equal for all the mapped elements since the latter always correspond with the same physical triangle. The only variability regards the value for the edge ratio, which takes into account the presence of the hanging nodes. Furthermore, we recall that the training set for the HTM family is made up of different copies of the same reference triangle, thus actually the elements $E \in \mathcal{T}^{\text{train}}_{HTM}$ are equal to the related mapped element $\hat{E}$. Finally, we can note that the losses related to the training set and the test set are very similar to each other and this means that the test elements are well represented by the chosen training sets. In Fig. 8, we report the contour plots of some basis functions predicted on test elements, which behave as expected.

5.2. Test problem 1: advection-diffusion-reaction problem

In this first experiment, we test the NAVEM method on a simple diffusion-advection-reaction problem. In particular, we consider the following boundary value problem on $\Omega = (0,1)^2$

$$\begin{cases} -\nabla \cdot (\boldsymbol{D}(\boldsymbol{x})\nabla u) + \boldsymbol{\beta}(\boldsymbol{x}) \cdot \nabla u + \gamma(\boldsymbol{x}) u = f & \text{in } \Omega, \\ u = g_D & \text{on } \Gamma, \end{cases} \tag{33}$$

where

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

while the Dirichlet boundary condition g_D and the forcing term f are chosen such that the exact solution is

$$u(\boldsymbol{x}) = 3\left((x_1 - 0.2) + \frac{x_2 - 0.3}{2}\right)^2 + 2\left(\frac{x_1 - 0.7}{2} + (x_2 - 0.8)\right)^3 + \sin(2\pi x_1)\sin(3\pi x_2), \tag{34}$$

which is shown in Fig. 9(a). We observe that this is the same test performed in [12], which is now extended to new polygonal meshes.

We solve problem (33) using both the NAVEM and VEM methods, and we plot the corresponding errors with respect to h in Fig. 10. Since the desired solution is regular enough, the VEM L^2-error $\text{err}_0^{\text{VEM}}$ and H^1-error $\text{err}_1^{\text{VEM}}$ decrease with expected rates of $O(h^2)$ and $O(h^1)$, respectively. Empirical observations indicate that the NAVEM solution converges at the same rate as the VEM in both the L^2 and H^1 norms. We further note that these numerical results are coherent with the expected ones, since the values of the loss functions are small enough on the elements of the meshes. Indeed, we recall that the elements of our meshes are well-represented by the polygons in the training sets, as discussed previously. Additionally, the absence of the projection and stabilization operator in the NAVEM is manifested as a downward shift in the convergence curves.

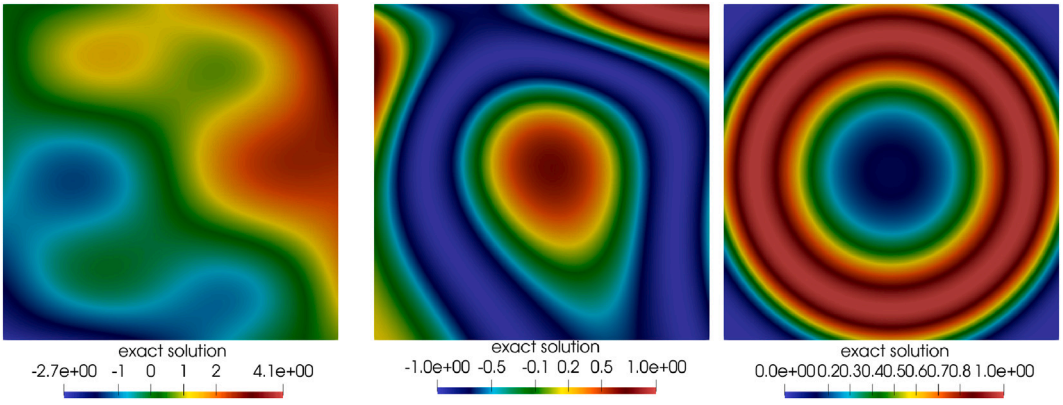
5.3. Test problem 2: anisotropic problem

Let us now consider a boundary-value problem with a strongly anisotropic tensor $\boldsymbol{D}$. In particular, we solve the problem (33) on $\Omega = (0,1)^2$ with

$$\boldsymbol{D} = \mathbf{G}\widetilde{\boldsymbol{D}}\mathbf{G}^T, \quad \widetilde{\boldsymbol{D}} = \begin{bmatrix} 1 & 0 \\ 0 & 1.0e{-}6 \end{bmatrix}, \quad \mathbf{G} = \begin{bmatrix} \cos\left(\frac{\pi}{6}\right) & -\sin\left(\frac{\pi}{6}\right) \\ \sin\left(\frac{\pi}{6}\right) & \cos\left(\frac{\pi}{6}\right) \end{bmatrix},$$

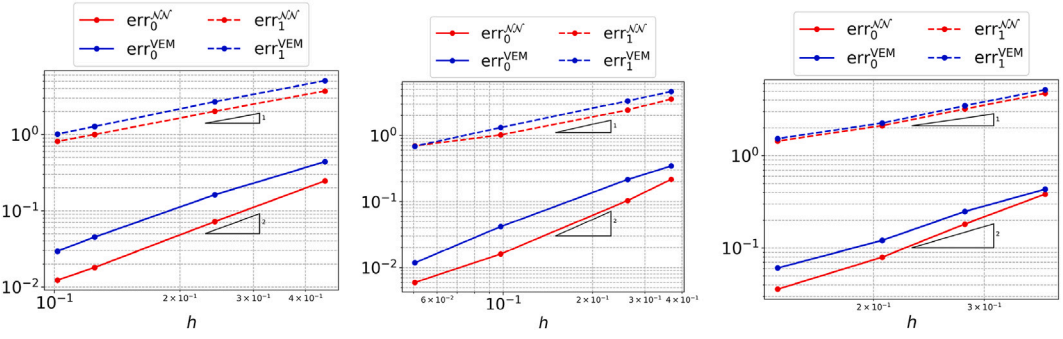
$$\boldsymbol{\beta} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad \gamma = 0,$$

where the matrix $\mathbf{G}$ is the Givens rotation matrix. As in the previous test cases, the forcing term and boundary data are chosen such that the exact solution is



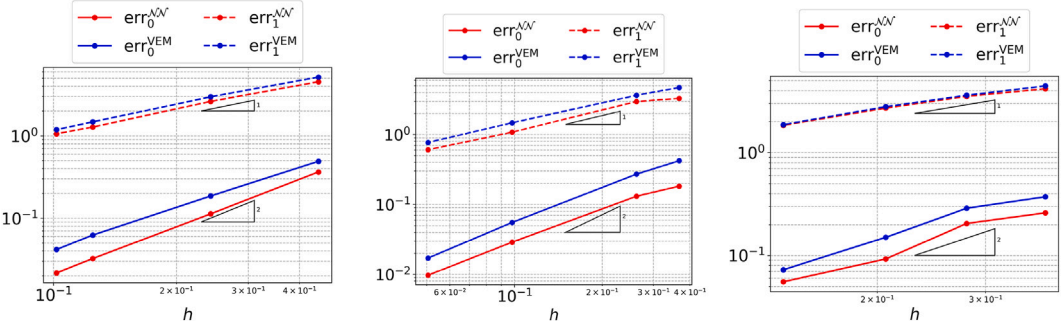
(a) (b) (c)

Fig. 9. Contour plots of the exact solutions (34), (35) and (38) from right to left. Left: Test 1. Center: Test 2. Right: Test 3.



(a) (b) (c)

Fig. 10. Test 1: NAVEM and VEM errors w.r.t h. Left: RDQM. Center: VM. Right: HTM.



(a) (b) (c)

Fig. 11. Test 2: NAVEM and VEM errors w.r.t h. Left: RDQM. Center: VM. Right: HTM.

$$u(\boldsymbol{x}) = \sin\left(3\cos\left(x_1 - 2x_2^2\right)^2 + 4\sin\left(x_2 + 2x_1\right)^2\right), \tag{35}$$

which is shown in Fig. 9(b).

The error convergence curves for both VEM and NAVEM are shown in Fig. 11. Again, we can observe that the knowledge of the virtual element basis functions is reflected in small error constants. We further highlight that, in this kind of problem, it is very difficult to design a proper stabilization term and a tuning strategy is advisable to choose a multiplicative stabilization coefficient [8]. The usage of a method like NAVEM removes this kind of issue.

5.4. Test problem 3: non-linear problem

Let us now consider the following non-linear problem

$$\begin{cases} -\nabla \cdot (D(u,\lambda)\nabla u) = f & \text{in } \Omega, \\ u = g_D & \text{on } \Gamma, \end{cases} \tag{36}$$

where the diffusion coefficient is given by

$$D(u,\lambda) = \frac{1}{\lambda + u^2}, \text{ with } \lambda \in \mathcal{P} := \{1.0, 0.5, 0.1\}, \tag{37}$$

whose graphical illustration is provided in Fig. 12 for $\lambda \in \mathcal{P}$. To compute the errors in Eqs. 31 and 32, we choose the Dirichlet boundary

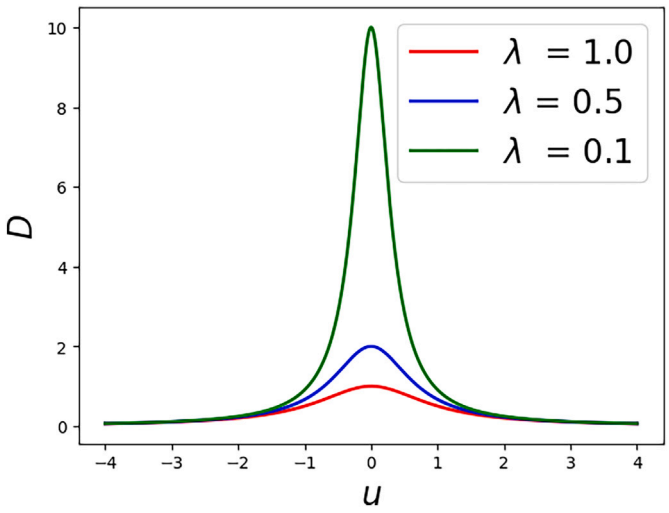


Fig. 12. Test 3: Diffusion coefficient $D(u,\lambda)$ as u varies. The different curves are associated with different values of the parameter λ.

condition and the forcing term in such a way that, for any λ, the exact solution is

$$u(\boldsymbol{x}) = \frac{1}{8}\left(\sin\left(3\pi((x_1-0.5)^2+(x_2-0.5)^2)\right)\right)^3. \tag{38}$$

The contour plot of the exact solution (38) is shown in Fig. 9(c).

For comparison purposes, we briefly report here the virtual element formulation that we use to solve the problem (36), which is introduced in [6]. In the case of quasi-linear elliptic problems, given $z_h \in V_{h,1}$ and $\lambda \in \mathcal{P}$, the local discrete virtual element bilinear form $a_h^E(\cdot,\cdot\,;z_h,\lambda)$ reads as

$$\begin{aligned} a_h^E(u_h,v_h;z_h,\lambda) = &\int_E D\left(\Pi_1^{E,0}z_h,\lambda\right)\Pi_0^{E,0}\nabla u_h\cdot\Pi_0^{E,0}\nabla v_h \\ &+ S^E\left(\left(I-\Pi_1^{E,\nabla}\right)u_h,\left(I-\Pi_1^{E,\nabla}\right)v_h;z_h,\lambda\right), \\ &\forall u_h,v_h\in V_{h,1}, \end{aligned}$$

where the VEM stabilizing form $S^E(\cdot,\cdot\,;z_h,\lambda)$ is given by

$$S^E\left(u_h,v_h;z_h,\lambda\right) = D\left(\Pi_0^{E,0}z_h,\lambda\right)\sum_{i=1}^{N_E^{\mathrm{dof}}}\mathrm{dof}_i^E(u_h)\,\mathrm{dof}_i^E(v_h),\quad \forall u_h,v_h\in V_{h,1}.$$

Now, we apply the Newton-Raphson method to deal with non-linearities. Thus, given an initial iterate $u_h^0\in V_{h,1}$, we define a sequence

$$u_h^{m+1} = u_h^m + \delta_h^m \quad \forall m\geq 0,$$

by solving, at each non-linear step m, the linearized problem: *Find* $\delta_h^m \in V_{h,1}$ *such that:*

$$\begin{aligned} &\sum_{E\in\mathcal{T}_h}\left[a_h^E\left(\delta_h^m,v_h;u_h^m,\lambda\right)+b_h^E\left(\delta_h^m,v_h;u_h^m,\lambda\right)\right] \\ &\quad=\sum_{E\in\mathcal{T}_h}\left[\mathcal{F}_h^E(v_h)-a_h^E\left(u_h^m,v_h;u_h^m,\lambda\right)\right]\quad \forall v_h\in V_{h,1}, \end{aligned}$$

where the extra term $b_h^E(\cdot,\cdot\,;u_h^m,\lambda)$ stems from the linearization of both the consistency and the stabilization terms and it is defined as

$$\begin{aligned} b_h^E\left(\delta_h^m,v_h;u_h^m,\lambda\right) = &\int_E \frac{\partial D\left(\Pi_1^{E,0}u_h^m,\lambda\right)}{\partial u}\Pi_1^{E,0}\delta_h^m\Pi_0^{E,0}\nabla u_h^m\cdot\nabla v_h \\ &+\frac{\partial D\left(\Pi_0^{E,0}u_h^m,\lambda\right)}{\partial u}\Pi_0^{E,0}\delta_h^m\sum_{i=1}^{N_E^{\mathrm{dof}}}\mathrm{dof}_i^E\left(\left(I-\Pi_1^{E,\nabla}\right)u_h\right) \\ &\times\mathrm{dof}_i^E\left(\left(I-\Pi_0^{E,\nabla}\right)v_h\right). \end{aligned}$$

Now, let us denote by $\boldsymbol{\delta}_h^m$ and $\boldsymbol{u}_h^m$ the vectors of coefficients of functions δ_h^m and u_h^m with respect to the virtual element basis functions, we introduce the following matrices

$$\mathbf{A}_h^m\in\mathbb{R}^{N^{\mathrm{dof}}\times N^{\mathrm{dof}}}\ :\ \left(\mathbf{A}_h^m\right)_{ji} = \sum_{E\in\mathcal{T}_h}\left[a_h^E\left(\varphi_i,\varphi_j;u_h^m,\lambda\right)\right],$$

$$\boldsymbol{f}_h^m\in\mathbb{R}^{N^{\mathrm{dof}}}\ :\ \left(\boldsymbol{f}_h^m\right)_j = \sum_{E\in\mathcal{T}_h}\left[\mathcal{F}_h^E(\varphi_j)-a_h^E\left(u_h^m,\varphi_j;u_h^m,\lambda\right)\right],$$

and impose the following stopping criteria

$$\left\|\boldsymbol{r}_h^{VEM}\right\|_2\leq\epsilon_{r,r}\left\|\boldsymbol{f}_h^0\right\|_2+\epsilon_{r,a}\ \text{and}\ \left\|\boldsymbol{\delta}_h^m\right\|_2\leq\epsilon_{\delta,r}\left\|\boldsymbol{u}_h^0\right\|_2+\epsilon_{\delta,a}, \tag{39}$$

where

$$\boldsymbol{r}_h^{VEM} = \boldsymbol{f}_h^m-\mathbf{A}_h^m\boldsymbol{u}_h^m\quad\forall m\geq 0.$$

We observe that, in this type of construction, the heavy usage of the polynomial projectors and stabilization terms can become a significant issue when the diffusion coefficient becomes highly non-linear. Indeed, the non-linearity may increase the discrepancy between the continuous and the approximated discrete bilinear form used in the virtual element formulation. The NAVEM method eliminates the need for stabilization terms or projection operators, simplifying the discrete bilinear form used in the Newton-Raphson method and further aligning NAVEM with standard FEM. For the sake of completeness, in the following, we outline the NAVEM discretization of the problem (36). Given an initial guess $u_h^{\mathcal{NN},0}\in V_{h,1}^{\mathcal{NN}}$, at each non-linear step $m\geq 0$, we solve the linearized problem: *Find* $\delta_h^{\mathcal{NN},m}\in V_{h,1}^{\mathcal{NN}}$ *such that:*

$$\begin{aligned} &\sum_{E\in\mathcal{T}_h}\left[a_{h,\mathcal{NN}}^E\left(\delta_h^{\mathcal{NN},m},v_h^{\mathcal{NN}};\,u_h^{\mathcal{NN},m},\lambda\right)+b_{h,\mathcal{NN}}^E\left(\delta_h^{\mathcal{NN},m},v_h^{\mathcal{NN}};\,u_h^{\mathcal{NN},m},\lambda\right)\right] \\ &\quad=\sum_{E\in\mathcal{T}_h}\left[\left(f,v_h^{\mathcal{NN}}\right)_E-a_{h,\mathcal{NN}}^E\left(u_h^{\mathcal{NN},m},v_h^{\mathcal{NN}};u_h^{\mathcal{NN},m},\lambda\right)\right]\ \forall v_h^{\mathcal{NN}}\in V_{h,1}^{\mathcal{NN}}, \end{aligned}$$

where

$$u_h^{\mathcal{NN},m+1} = \delta_h^{\mathcal{NN},m}+u_h^{\mathcal{NN},m}\quad m\geq 0,$$

and the involved bilinear forms are defined as

$$a_{h,\mathcal{NN}}^E\left(u_h^{\mathcal{NN}},v_h^{\mathcal{NN}};\,z_h^{\mathcal{NN}},\lambda\right) = \int_E D\left(z_h^{\mathcal{NN}},\lambda\right)\nabla u_h^{\mathcal{NN}}\cdot\nabla v_h^{\mathcal{NN}},$$

$$b_{h,\mathcal{NN}}^E\left(\delta_h^{\mathcal{NN}},v_h^{\mathcal{NN}};\,u_h^{\mathcal{NN}},\lambda\right) = \int_E\frac{\partial D\left(u_h^{\mathcal{NN}},\lambda\right)}{\partial u}\delta_h^{\mathcal{NN}}\ \nabla u_h^{\mathcal{NN}}\cdot\nabla v_h^{\mathcal{NN}}.$$

The stopping criteria are defined in the same fashion as what we have done with VEM. Thus, let us denote by $\boldsymbol{\delta}_h^{\mathcal{NN},m}$ and $\boldsymbol{u}_h^{\mathcal{NN},m}$ the vectors of degrees of freedom related to $\delta_h^{\mathcal{NN},m}$ and $u_h^{\mathcal{NN},m}$, we introduce the following matrices $\forall m\geq 0$

$$\mathbf{A}_h^{\mathcal{NN},m}\in\mathbb{R}^{N^{\mathrm{dof}}\times N^{\mathrm{dof}}}\ :\ \left(\mathbf{A}_h^{\mathcal{NN},m}\right)_{ji} = \sum_{E\in\mathcal{T}_h}\left[a_{h,\mathcal{NN}}^E\left(\varphi_i^{\mathcal{NN}},\varphi_j^{\mathcal{NN}};u_h^{\mathcal{NN},m},\lambda\right)\right],$$

$$\begin{aligned} \boldsymbol{f}_h^{\mathcal{NN},m}\in\mathbb{R}^{N^{\mathrm{dof}}}\ :\ &\left(\boldsymbol{f}_h^{\mathcal{NN},m}\right)_j \\ &=\sum_{E\in\mathcal{T}_h}\left[\left(f,\varphi_j^{\mathcal{NN}}\right)_E-a_{h,\mathcal{NN}}^E\left(u_h^{\mathcal{NN},m},\varphi_j^{\mathcal{NN}};u_h^{\mathcal{NN},m},\lambda\right)\right], \end{aligned}$$

and impose the following stopping criteria

$$\begin{aligned} &\left\|\boldsymbol{r}_h^{\mathcal{NN}}\right\|_2\leq\epsilon_{r,r}\left\|\boldsymbol{f}_h^{\mathcal{NN},0}\right\|_2+\epsilon_{r,a}\ \text{and} \\ &\left\|\boldsymbol{\delta}_h^{\mathcal{NN},m}\right\|_2\leq\epsilon_{\delta,r}\left\|\boldsymbol{u}_h^{\mathcal{NN},0}\right\|_2+\epsilon_{\delta,a}, \end{aligned} \tag{40}$$

where

$$\boldsymbol{r}_h^{\mathcal{NN}} = \boldsymbol{f}_h^{\mathcal{NN},m}-\mathbf{A}_h^{\mathcal{NN},m}\boldsymbol{u}_h^{\mathcal{NN},m}\quad\forall m\geq 0.$$

In this numerical experiment, we set the initial guess as the all-zeros vector and we choose $\epsilon_{r,r}=\epsilon_{r,a}=10^{-12}$ and $\epsilon_{\delta,r}=\epsilon_{\delta,a}=10^{-10}$.

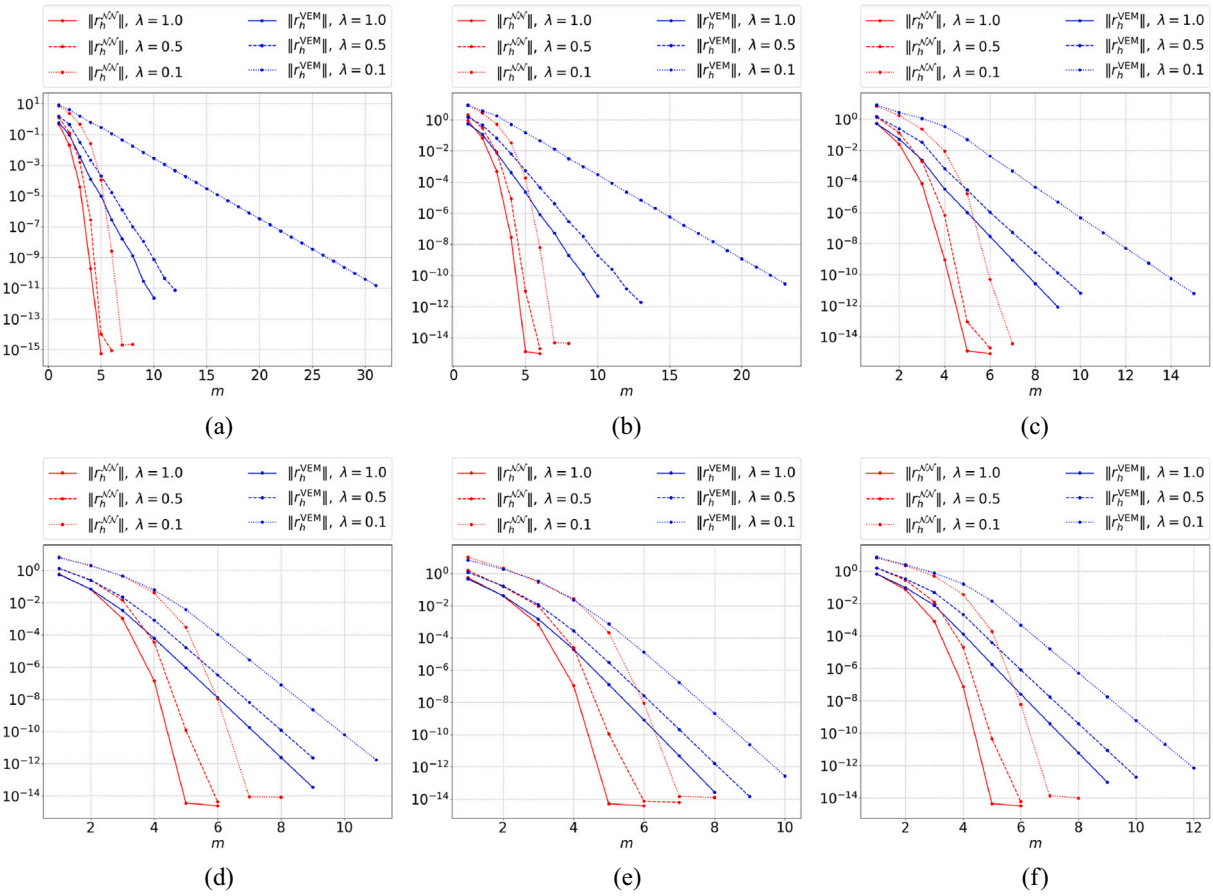


Fig. 13. Test 3: Residual norm as the number of non-linear iterations varies for both the VEM and the NAVEM. Different line styles represent different values of the parameter λ, while the two rows represent the first and the last mesh of each family, respectively. Left: RDQM. Center: VM. Right: HTM.

Table 3
Test 3: statistics 5.4 related to each mesh belonging to the family $\mathcal{T}_{\text{VM}}^{\text{test}}$ and each $\lambda \in \mathcal{P}$.

λ	Id. mesh	r(T)	r(m)	r(ATI)
1.00	0	1.41	1.67	0.85
0.50	0	1.77	2.17	0.82
0.10	0	2.95	2.88	1.03
1.00	1	2.09	1.67	1.26
0.50	1	2.37	1.83	1.29
0.10	1	2.48	1.75	1.42
1.00	2	2.45	1.50	1.64
0.50	2	2.55	1.43	1.78
0.10	2	3.20	1.50	2.13
1.00	3	2.43	1.33	1.82
0.50	3	2.40	1.29	1.87
0.10	3	2.54	1.25	2.03

Fig. 13 shows the behaviours of both $\left\|\boldsymbol{r}_h^{\mathcal{NN}}\right\|_2$ and $\left\|\boldsymbol{r}_h^{VEM}\right\|_2$ as the number of non-linear iterations m increases, for each $\lambda \in \mathcal{P}$ and for the first and the last mesh of each family. We observe that the coarser is the mesh and the smaller is λ, the larger is the number of iterations that the standard VEM employs to reach the desired tolerance. On the other hand, we observe that the number of iterations related to NAVEM is not strongly dependent on the parameter h and its variability with respect to the parameter λ is much weaker with respect to the VEM one. Moreover, we highlight the plateau of the residual is due to the double stopping criteria imposed (39).

To provide insight into the computational time of NAVEM with respect to VEM, Table 3 reports the time (in seconds) required to solve the non-linear problem (36) on the mesh family $\mathcal{T}_{\text{VM}}^{\text{test}}$. More precisely, we solve problem (36) 20 times and we average the solving times. These times refer to a Python code that exploits TensorFlow for neural network operations, running on a Ubuntu 24.04 LTS 64-bit, 12th Gen Intel(R) Core(TM) i7-1255U CPU (4.7 GHz) and 16 GB RAM memory. Specifically, we report

1. **r(T)**: the ratio between the time T required by VEM and and by NAVEM to solve problem (36);
2. **r(m)**: the ratio between the number of non-linear iteration m in the VEM and in the NAVEM approach;
3. **r(ATI)**: the ratio between the average time per iteration (ATI in short) in VEM and NAVEM.

We note that the NAVEM time specifically corresponds to the time required to apply the steps **S.1**–**S.5** for solving the problem (36). Additionally, to exploit the TensorFlow vectorization capabilities and reduce the computational burden, we minimize the number of neural network evaluations by aggregating the inputs associated with multiple basis functions. Indeed, as shown in Table 3, this cost per call has a significant impact on small datasets (coarser meshes) but a much smaller effect on larger datasets. We remark that, for finer meshes, these ratios are greater than 1, making NAVEM very advantageous for solving highly non-linear problems. Moreover, for each mesh, the best ratios are obtained when $\lambda = 0.1$. This suggests that using an approach without stabilization and projection operators is even more efficient when the PDE is very non-linear.

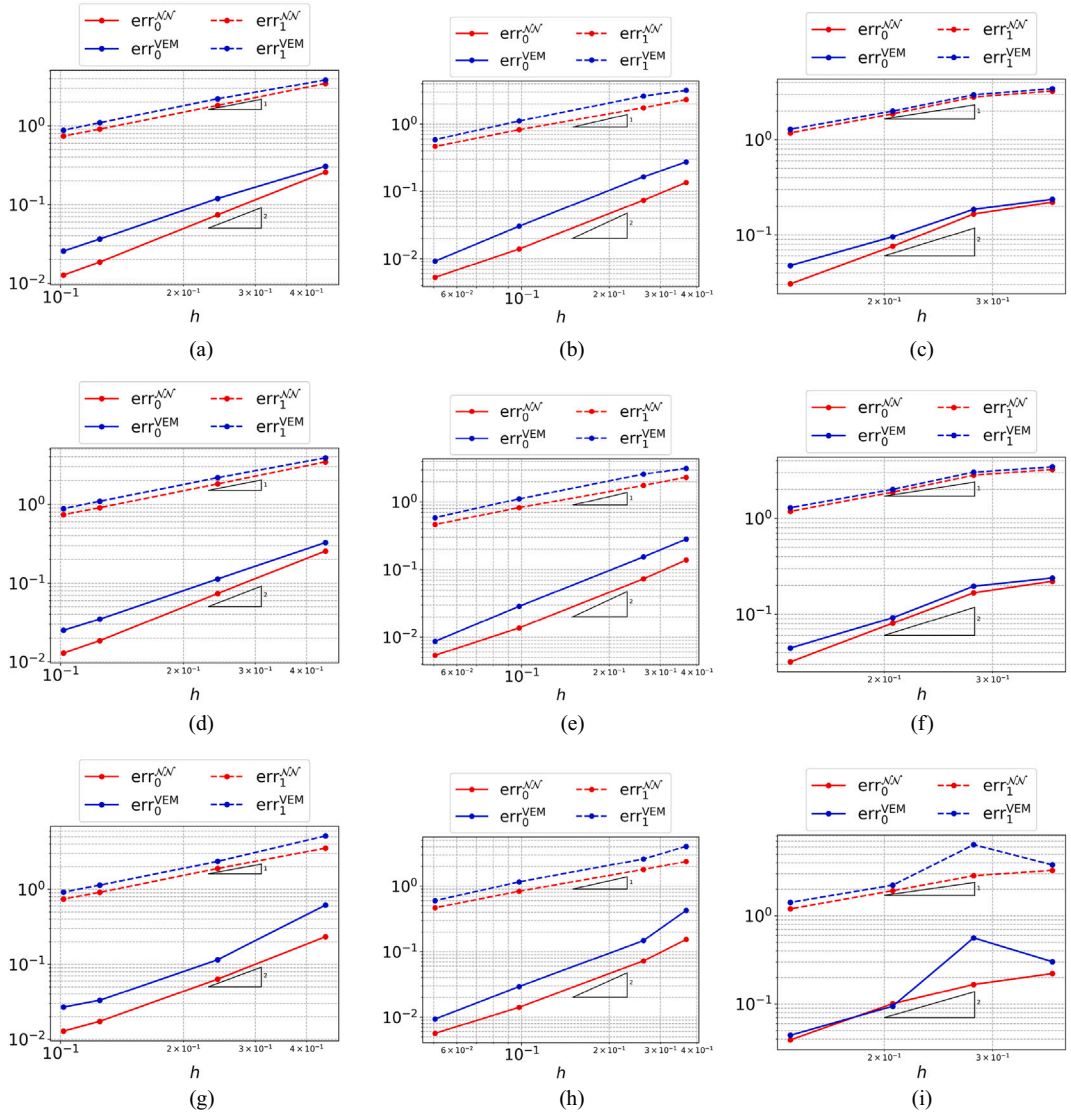


Fig. 14. Test 3: NAVEM and VEM errors w.r.t h. Each row corresponds to a different value of the parameter $\lambda = 1.0, 0.5, 0.1$ from top to bottom. Left: RDQM. Center: VM. Right: HTM.

Finally, Fig. 14 shows the convergence curves related to the errors (31) and (32) for NAVEM and VEM, respectively. Again, we observe a reduction in the error constants for each tested case with respect to VEM method, while significantly reducing the number of iterations needed to achieve the desired tolerance. These results suggest that NAVEM can provide competitive accuracy with respect to standard VEM while simplifying the formulation.

6. Conclusions

In this paper, we extend and describe the lowest-order neural approximated virtual element method on quite general polygonal elements. The NAVEM is a polygonal method used to solve partial differential equations that combines standard numerical techniques with neural networks, preserving the convergence rate of the standard numerical method, while exploiting the offline-online paradigm of neural networks to overcome the limitations of the standard procedure. Indeed, it modifies the original VEM formulation in [1] by explicitly approximating the virtual element basis functions through suitable harmonic functions parameterized by a neural network and deleting issues related to the introduction of polynomial projections and stabilization operators as in standard VEM.

Two different neural network architectures and related training strategies are described and theoretically justified. Few papers tackle theoretical discussions and results about the intersection between standard mesh-based solvers and neural networks are available; some examples include [35–37].

Numerical results confirm that the presented method helps to avoid issues concerning the choice of the stabilization term and accessing the point-wise evaluation of basis functions without using polynomial projectors, showing good performances, especially in the case of highly non-linear problems. Furthermore, particular attention is devoted to the analysis of triangular meshes with hanging nodes given their relevance in the numerical field.

We believe that this study could help exploring new advanced strategies for practical applications.

CRediT authorship contribution statement

Stefano Berrone: Writing – review & editing, Supervision, Resources, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Data curation. **Moreno Pintore:** Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Formal analysis, Data curation,

Conceptualization. **Gioana Teora:** Writing – review & editing, Writing – original draft, Visualization, Validation, 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

The author S.B. kindly acknowledges the partial financial support provided by PRIN project “Advanced polyhedral discretisations of heterogeneous PDEs for multiphysics problems” (No. 20204LN5N5_003), by PNRR M4C2 project of CN00000013 National Centre for HPC, Big Data and Quantum Computing (HPC) (CUP: E13C22000990001) and the funding by the 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.”. 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 the MIUR programme “Programma Operativo Nazionale Ricerca e Innovazione 2014–2020” (CUP: E11B21006490005) and by INdAM - GNCS Project CUP E53C23001670001.

Data availability

Data will be made available on request.

References

[1] Beirão da Veiga L, Brezzi F, Cangiani A, Manzini G, Russo A. Basic principles of virtual element methods. Math Models Methods Appl Sci 2013;23(1):199–214. doi:https://doi.org/10.1142/S0218202512500492.
[2] Beirão da Veiga L, Brezzi F, Marini LD, Russo A. Virtual element method for general second-order elliptic problems on polygonal meshes. Math Models Methods Appl Sci 2016;26(4):729–50. doi:https://doi.org/10.1142/S0218202516500160.
[3] Russo A, Sukumar N. Quantitative study of the stabilization parameter in the virtual element method. In da Veiga HB, Minhós F, Van Goethem N, Rodrigues LS, editors. Nonlinear differential equations and applications. Cham: Springer International Publishing; 2024. p. 259–78. doi:https://doi.org/10.1007/978-3-031-53740-0_14.
[4] Berrone S, Borio A, Marcon F, Teora G. A first-order stabilization-free virtual element method. Appl Math Lett 2023;142:108641. doi:https://doi.org/10.1016/j.aml.2023.108641.
[5] Berrone S, Scialò S, Teora G. The mixed virtual element discretization for highly-anisotropic problems: the role of the boundary degrees of freedom. Math Eng 2023;5(6):1–32. doi:https://doi.org/10.3934/mine.2023099.
[6] Cangiani A, Chatzipantelidis P, Diwan G, Georgoulis EH. Virtual element method for quasilinear elliptic problems. IMA J Numer Anal 2019;40(4):2450–72. doi:https://doi.org/10.1093/imanum/drz035.
[7] Adak D, Natarajan E, Kumar S. Convergence analysis of virtual element methods for semilinear parabolic problems on polygonal meshes. Numer Methods Partial Differ Equ 2019;35(1):222–45. doi:https://doi.org/10.1002/num.22298.
[8] Berrone S, Borio A, Marcon F. A stabilization-free virtual element method based on divergence-free projections. Comput Methods Appl Mech Eng 2024;424:116885. doi:https://doi.org/10.1016/j.cma.2024.116885.
[9] Credali F, Bertoluzza S, Prada D. Reduced basis stabilization and post-processing for the virtual element method. Comput Methods Appl Mech Eng 2024;420:116693. doi:https://doi.org/10.1016/j.cma.2023.116693.
[10] Trezzi M, Zerbinati U. When rational functions meet virtual elements: the lightning virtual element method. Calcolo 2024;61(3):35. doi:https://doi.org/10.1007/s10092-024-00585-1.
[11] Gopal A, Trefethen LN. Solving Laplace problems with corner singularities via rational functions. SIAM J Numer Anal 2019;57(5):2074–94. doi:https://doi.org/10.1137/19M125947X.
[12] Berrone S, Oberto D, Pintore M, Teora G. The lowest-order neural approximated virtual element method. In Sequeira A; Silvestre A, Valtchev SS, Janela J, editors. Numerical mathematics and advanced applications ENUMATH 2023, vol. 1. Cham: Springer Nature Switzerland; 2025.
[13] Abadi M, et al. TensorFlow: large-scale machine learning on heterogeneous systems, software available from tensorflow.org; 2015. http://tensorflow.org/.
[14] Paszke A, et al. Pytorch: an imperative style, high-performance deep learning library. In: Advances in neural information processing systems 32. Curran Associates, Inc; 2019. p. 8024–35. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.
[15] Bradbury J, Frostig R, Hawkins P, Johnson MJ, Leary C, Maclaurin D, et al. JAX: composable transformations of Python + NumPy programs; 2018. http://github.com/google/jax.
[16] Cuomo S, Di Cola VS, Giampaolo F, Rozza G, Raissi M, Piccialli F. Scientific machine learning through physics–informed neural networks: where we are and what’s next. J Sci Comput 2022;92(3):88. doi:https://doi.org/10.1007/s10915-022-01939-z.
[17] Beirão da Veiga L, Lovadina C, Russo A. Stability analysis for the virtual element method. Math Models Methods Appl Sci 2017;27(13):2557–94. doi:https://doi.org/10.1142/S021820251750052X.
[18] Beirão da Veiga L, Dassi F, Russo A. High-order virtual element method on polyhedral meshes. Comput Math Appl 2017;74(5):1110–22. sI: SDS2016 – Methods for PDEs. doi:https://doi.org/10.1016/j.camwa.2017.03.021.
[19] Boffi D, Brezzi F, Fortin M. Mixed finite element methods and applications. In: Springer series in computational mathematics. Berlin, Heidelberg: Springer; 2013. https://books.google.it/books?id=mRhAAAAAQBAJ.
[20] Ayuso de Dios B, Lipnikov K, Manzini G. The nonconforming virtual element method. ESAIM: M2AN 2016;50(3):879–904. doi:https://doi.org/10.1051/m2an/2015090.
[21] Brenner SC, Scott LR. The mathematical theory of finite element methods. In: Of texts in applied mathematics, vol. 15. Springer; 2008. doi:https://doi.org/10.1007/978-0-387-75934-0.
[22] Benedetto MF, Berrone S, Scialò S. A globally conforming method for solving flow in discrete fracture networks using the virtual element method. Finite Elem Anal Des 2016;109:23–36. doi:https://doi.org/10.1016/j.finel.2015.10.003.
[23] Canuto C, Fassino D. Higher-order adaptive virtual element methods with contraction properties. Math Eng 2023;5(6):1–33. doi:https://doi.org/10.3934/mine.2023101.
[24] Berrone S, Teora G, Vicini F. Improving high-order vem stability on badly-shaped elements. Math Comput Simul 2024; 216:367–85. doi:https://doi.org/10.1016/j.matcom.2023.10.003.
[25] Perot JB, Chartrand C. A mimetic method for polygons. J Comput Phys 2021;424:109853. doi:https://doi.org/10.1016/j.jcp.2020.109853.
[26] Glorot X, Bengio Y. Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the 13th international conference on artificial intelligence and statistics. JMLR workshop and conference proceedings; 2010. p. 249–56.
[27] Sharma S, Sharma S, Athaiya A. Activation functions in neural networks. Int J Eng Appl Sci Technol 2020;4(12):310–6. doi:https://doi.org/10.33564/ijeast.2020.v04i12.054.
[28] Popescu MC, Balas VE, Perescu-Popescu L, Mastorakis N. Multilayer perceptron and neural networks. WSEAS Trans Circuits Syst 2009;8(7):579–88.
[29] Delashmit WH, Manry MT, et al. Recent developments in multilayer perceptron neural networks. In: Proceedings of the 7th annual memphis area engineering and science conference. MAESC, vol. 7; 2005. p. 33.
[30] Bonito A, Canuto C, Nochetto RH, Veeser A. Adaptive finite element methods. Acta Numer 2024;33:163–485. doi:https://doi.org/10.1017/S0962492924000011.
[31] Ammari H, Kang H. Reconstruction of small inhomogeneities from boundary measurements, vol. 1846; 2004. doi:https://doi.org/10.1007/b98245.
[32] Gautschi W. The use of rational functions in numerical quadrature. J Comput Appl Math 2001;133(1):111–26. 5th int. symp. on orthogonal polynomials, special functions and their applications. doi:https://doi.org/10.1016/S0377-0427(00)00637-3.
[33] Kingma D, Ba J. Adam: a method for stochastic optimization. In: International conference on learning representations; 2014. doi:https://doi.org/10.48550/arXiv.1412.6980.
[34] Wright S, Nocedal J, et al. Numerical optimization, vol. 35. Springer; 1999.
[35] Berrone S, Canuto C, Pintore M. Variational physics informed neural networks: the role of quadratures and test functions. J Sci Comput 2022;92(3):100. doi:https://doi.org/10.1007/s10915-022-01950-4.
[36] Badia S, Li W, Martín AF. Finite element interpolated neural networks for solving forward and inverse problems. Comput Methods Appl Mech Eng 2024;418:116505. doi:https://doi.org/10.1016/j.cma.2023.116505.
[37] Berrone S, Canuto C, Pintore M. Solving PDEs by variational physics-informed neural networks: an a posteriori error analysis. Ann Univ Ferrara 2022;68:575–95. doi:https://doi.org/10.1007/s11565-022-00441-6.