An introduction to the lowest-order Neural Approximated Virtual Element Method

COMinDS Workshop 2024 - Delft

Moreno Pintore (Inria, Sorbonne University).
Joint work with: Stefano Berrone, Davide Oberto, Gioana Teora (PoliTO).

Introduction

Given a polygonal/polyhedral domain $\Omega \in \mathbb{R}^d$, $d = 2, 3$, and a forcing term $f \in L^2(\Omega)$, let us consider the Poisson problem:

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

To solve problem (1) there exist several numerical methods, one of them is the **Virtual Element Method (VEM)**. It is a **polytopal** method particularly useful when **high-order** accuracy is required or the **geometry** of the domain is challenging. The VEM is characterized by the need of **projection** and problem-dependent and isotropic **stabilization** operators.

We now present the **Neural Approximated Virtual Element Method (NAVEM)**, a novel method that relies on neural networks ($\mathcal{NN}$) to eliminate the need of the limiting projection and stabilization operator in the VEM.

Note that we restrict ourself to problem (1) only for the sake of clarity, but more general equations can be considered, as in the *Numerical results* section.

VEM formulation

Let $\mathcal{T}_h$ be a decomposition of Ω into polygons E and let $\mathcal{E}_{h,E}$ be the set of edges of the element $E \in \mathcal{T}_h$. We define the lowest-order **local Virtual Element space** as:

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

where

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

We introduce two **computable local polynomial projectors** Π_0^0 and Π_1^∇ defined with respect to the L^2 and H_0^1 scalar products, respectively. Then, the **VEM variational formulation** of problem (1) reads as:

Find $u_h \in V_{h,1}$ *such that*

$$\sum_{E \in \mathcal{T}_h} a_h^E(u_h, v_h) = \sum_{E \in \mathcal{T}_h} (f, \Pi_0^0 v_h)_E \quad \forall v_h \in V_{h,1},$$

with

$$a_h^E(u_h, v_h) = \int_E \nabla \Pi_1^\nabla u_h \cdot \nabla \Pi_1^\nabla v_h + S^E((I - \Pi_1^\nabla)u_h, (I - \Pi_1^\nabla)v_h).$$

NAVEM formulation

Let us introduce the space $\mathbb{H}_\ell(E)$ of the **harmonic polynomials** of degree up to $\ell \geq 1$. Combining such functions by means of a **suitably trained** $\mathcal{NN}$, we aim at approximating the map

$$(v_j, E) \mapsto \varphi_{j,E}^{\mathcal{NN}} \in \mathbb{H}_\ell(E), \text{ for each vertex } v_j \text{ of } E \text{ and } \forall E \in \mathcal{T}_h.$$

This way, we obtain a **local approximation of each VEM basis function** $\varphi_{j,E}^{\mathcal{NN}} \in V_{h,1}(E)$ and we construct the linear space:

$$V_{h,1}^{\mathcal{NN}}(E) = \text{span}\{\varphi_{j,E}^{\mathcal{NN}},\ j = 1, \ldots, N_E^{\text{dof}}\} \subset \mathbb{H}_{\ell_E}(E),$$

Then, the **NAVEM variational formulation** of problem (1) reads as:

Find $u_h^{\mathcal{NN}} \in V_{h,1}^{\mathcal{NN}}$ *such that*

$$\sum_{E \in \mathcal{T}_h} a^E(u_h^{\mathcal{NN}}, v_h^{\mathcal{NN}}) = \sum_{E \in \mathcal{T}_h} (f, v_h^{\mathcal{NN}})_E \quad \forall v_h^{\mathcal{NN}} \in V_{h,1}^{\mathcal{NN}},$$

where

$$a^E(u_h^{\mathcal{NN}}, v_h^{\mathcal{NN}}) = \int_E \nabla u_h^{\mathcal{NN}} \cdot \nabla v_h^{\mathcal{NN}}.$$

The offline and online phases

The NAVEM **offline/training phase** can be described as follows:

- Given a number v of vertices, $v = 3, \ldots, V$, consider a set of polygons $\{G_k\}_{k=1}^{K_v}$ with v vertices and, map them to the elements $\{\widehat{G}_k\}_{k=1}^{K_v}$ through an **affine map based on the inertia tensor** to reduce the variability of elements seen by the $\mathcal{NN}$;
- For all k, evaluate the basis functions of $V_{h,1}(\widehat{G}_k)$ on enough points $\{x_{i,k}\}_{i,k}$, $i = 1, \ldots, I_k$ on the boundary $\partial\widehat{G}_k$ of $\widehat{G}_k$;
- Express the NAVEM basis functions as $\varphi_{j,\widehat{G}}^{\mathcal{NN}} = \sum_{q=1}^{\dim(\mathbb{H}_\ell(\widehat{G}))} c_{q,\widehat{G}}(\theta) p_q$, where the coefficient $c_{i,E}(\theta)$ is the i-th output of a $\mathcal{NN}$ evaluated on the input $\widehat{G}$ (suitably encoded). For each value v, train a $\mathcal{NN}$ to **minimize**:

$$\sum_{k=1}^{K_v} \sum_{j=1}^{V_k} \sum_{i=1}^{I_k} \left[\left(\varphi_{j,\widehat{G}}^{\mathcal{NN}} - \varphi_{j,\widehat{G}} \right)^2 + \left(\frac{\partial \varphi_{j,\widehat{G}}^{\mathcal{NN}}}{\partial \mathbf{t}} - \frac{\partial \varphi_{j,\widehat{G}}}{\partial \mathbf{t}} \right)^4 \right] (x_{i,k}).$$

The NAVEM **online phase** can be described as follows:

- Consider a PDE defined on a domain Ω and its decomposition $\mathcal{T}_h$. For each element $E \in \mathcal{T}_h$ with v_E vertices, use the related $\mathcal{NN}$ to **compute the corresponding basis functions**.
- Since all the basis functions are known, assemble and solve the linear system as in a **standard FEM solver**.

Examples of basis functions

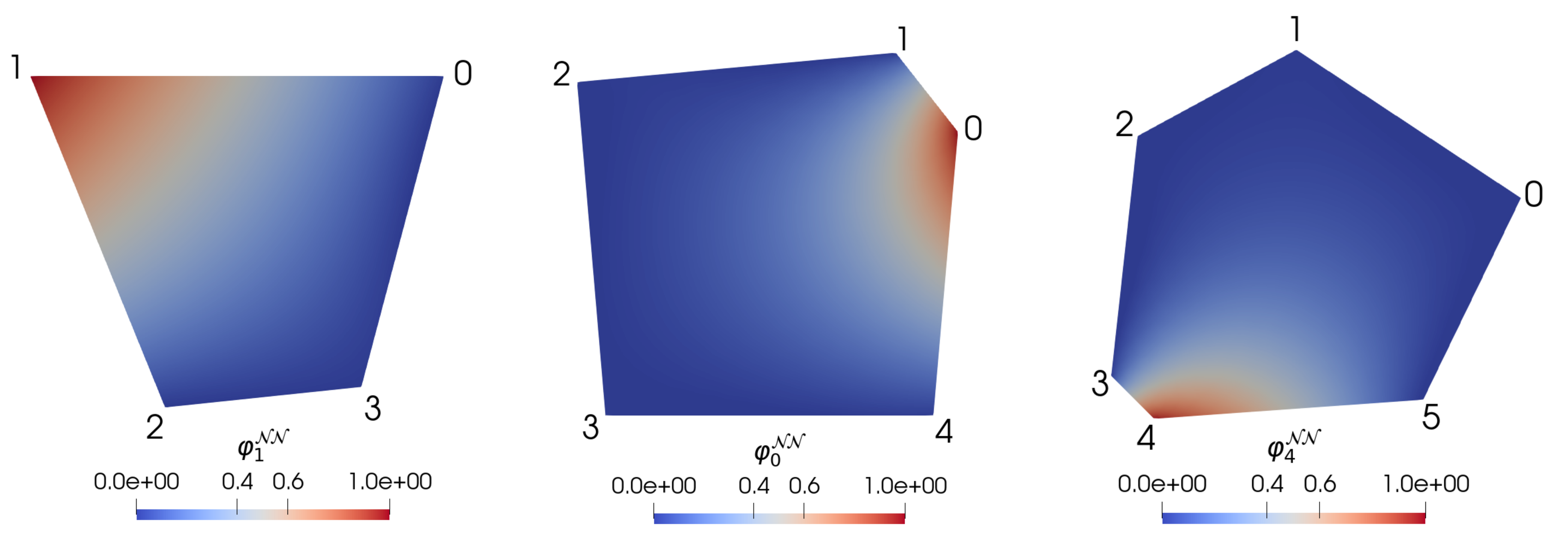

Figure 1: Three training polygons, coloured by the predicted basis functions.

Numerical results

Let us test the NAVEM on the following problem:

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

with:

$$\boldsymbol{D}(\boldsymbol{x}) = \begin{bmatrix} 1 + \boldsymbol{x}_2^2 & -\boldsymbol{x}_1\boldsymbol{x}_2 \\ -\boldsymbol{x}_1\boldsymbol{x}_2 & 1 + \boldsymbol{x}_1^2 \end{bmatrix}, \quad \boldsymbol{\beta}(\boldsymbol{x}) = \begin{bmatrix} \boldsymbol{x}_1 \\ -\boldsymbol{x}_2 \end{bmatrix}, \quad \gamma(\boldsymbol{x}) = \boldsymbol{x}_1\boldsymbol{x}_2.$$

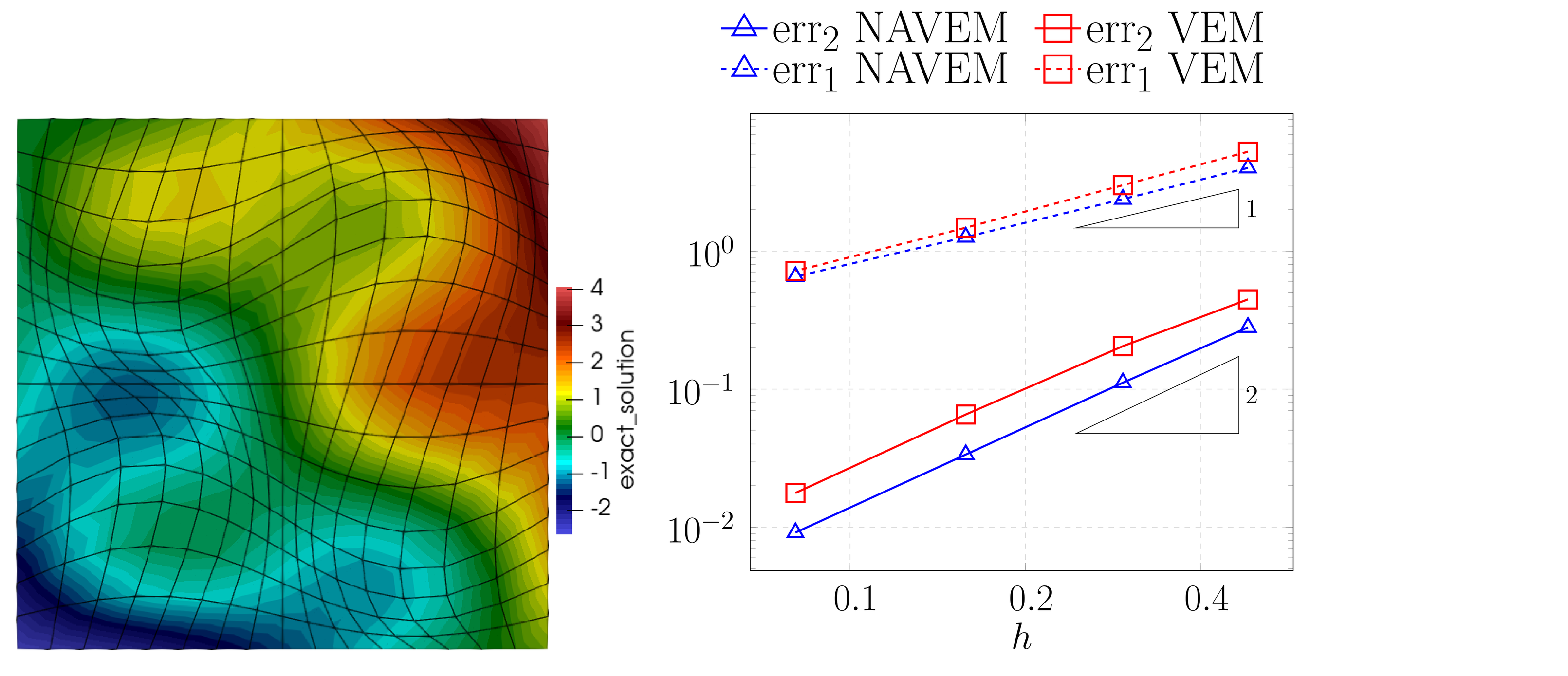

Figure 2: Mesh and exact solution (left), corresponding error decays (right).

Bibliography

[1] S. Berrone, D. Oberto, M. Pintore, and G. Teora. The lowest-order Neural Approximated Virtual Element Method. ArXiv preprint arXiv:2311.18534, (2023).