POLYDIM: A C++ library for POLYtopal DIscretization Methods

Computer Physics Communications 320 (2026) 109937

Contents lists available at ScienceDirect

Computer Physics Communications

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

Computational Physics

POLYDIM: A C++ library for POLYtopal DIscretization Methods

Stefano Berrone [1], Andrea Borio [1], Gioana Teora [1,*], Fabio Vicini [1]

Dipartimento di Scienze Matematiche "G. L. Lagrange", Politecnico di Torino, Corso Duca degli Abruzzi, 24, Turin, 10129, Italy

ARTICLE INFO

The review of this paper was arranged by Prof. Andrew Hazel

Keywords:
C++
Python
Implementation
Polytopal method
Virtual element method
Finite element method

ABSTRACT

This paper introduces PolyDiM, an open-source C++ library tailored for the development and implementation of polytopal discretization methods for partial differential equations. The library provides robust and modular tools to support advanced numerical techniques, with a focus on the Virtual Element Method in both 2D and 3D settings. PolyDiM is designed to address a wide range of challenging problems, including those involving non-convex geometries, domain decomposition and mixed-dimensional coupling applications. It is integrated with the geometry library GeDiM, and offers interfaces for MATLAB and Python to enhance accessibility. Distinguishing features include support for multiple polynomial bases, advanced stabilization strategies, and efficient local-to-global assembly procedures. PolyDiM aims to serve both as a research tool and a foundation for scalable scientific computing in complex geometrical settings.

1. Introduction

The numerical solution of partial differential equations (PDEs) on polytopal (polygonal or polyhedral) meshes has gained increasing attention in recent years due to its versatility and robustness in handling complex geometries. These meshes, which may include highly irregular or non-convex shapes, offer substantial advantages in engineering applications, since they can naturally represent domains with intricate structures and reduce the computational complexities of some advanced meshing strategies such as refinement and agglomeration processes. Many novel numerical methods have been developed in recent years to deal with polytopal elements, such as the Virtual Element Method (VEM) [1,2], the Hybrid High-Order method (HHO) [3], and Polytopal Discontinuous Galerkin methods (PDG) [4].

In this work, we present PolyDiM (POLYtopal DIscretization Method), an open-source C++ library tailored for the development and implementation of polytopal methods for PDEs. Built around the *local-to-global* philosophy typical of the Finite Element Method (FEM) [5,6], PolyDiM is designed to be modular, extensible, and researcher-friendly. Originally developed to support the authors's research, the library currently supports classical FEM in 1D, 2D, and 3D, alongside a robust implementation of VEM for a variety of boundary value problems in 2D and 3D.

In particular, the Virtual Element Method can be considered as a generalization of the FEM which introduces in the local space suitable non-polynomial functions that are not known in a closed form, as well as the standard polynomials, by introducing projection and stability operators that make the discrete operators computable and stable. Among the advantages of VEM, in addition to the employment of polytopal meshes, we recall the possibility of handling discrete spaces of arbitrary C^k global regularity [7], as well as spaces that satisfy exactly the divergence-free constraint [8].

Current applications of Virtual Elements in PolyDiM include the simulation of coupled problems on geometrically complex domains, such as Discrete Fracture Networks (DFNs) [9], and 3D-1D coupling [10]. Moreover, the library supports the research on polytopal mesh refinement and agglomeration strategies [11], alongside analysis of the robustness of VEM on badly-shaped elements [12–15].

To maximize interoperability and geometric flexibility, PolyDiM is fully integrated with GeDiM (GEometry for DIscretization MEthods) [16], a C++ library developed by the same authors to provide robust geometry handling for polytopal domains.

The authors extend the library including other tools and applications, which are summmarized in Table 2. Furthermore, PolyDiM is complemented by PyPolyDiM, its Python binding that facilitates integration with scientific workflows and enables rapid prototyping and post-processing. In addition, a MATLAB interface is also available upon request. By proceeding in this way, the authors intend for PolyDiM & GeDiM to serve as a valuable tool for the whole scientific community.

We recognize that several other software packages supporting polytopal discretization methods are currently available. Table 1 provides a summary of the most notable ones that share similar features with

* Corresponding author.
E-mail addresses: stefano.berrone@polito.it (S. Berrone), andrea.borio@polito.it (A. Borio), gioana.teora@polito.it (G. Teora), fabio.vicini@polito.it (F. Vicini).
[1] Authors are members of INDAM-GNCS group.

https://doi.org/10.1016/j.cpc.2025.109937
Received 6 June 2025; Received in revised form 22 October 2025; Accepted 2 November 2025
Available online 15 November 2025

Table 1
List of the main software available that implements virtual element method.

Software	Language	Availability	Problem	Website
PolyDiM	C++ / Python MATLAB	Open-Source Upon Request	2D - 3D	https://polydim.it/
VEMComp	MATLAB	Open-Source	2D - 3D	https://github.com/massimofrittelli/VEMcomp
mVEM	MATLAB	Open-Source	2D - 3D	https://github.com/Terenceyuyue/mVEM
VEMLAB	MATLAB	Open-Source	2D	https://camlab.cl/vemlab/
Veamy	C++	Open-Source	2D	https://camlab.cl/veamy
VEM++	C++	Upon Request	2D - 3D	https://sites.google.com/view/vembic/home
DUNE-VEM	C++ / Python	Open-Source	2D - 3D	https://dune-project.org

Table 2
Summary of problems and methods handled by PolyDiM & GeDiM.

PolyDiM & GeDiM features	
Linear and Non-Linear PDEs Sections 3, 5	Elliptic Parabolic Stokes/Navier–Stokes Elasticity Bulk-Surface
Mixed-dimensional coupled problems Section 5	3D-1D coupled problems Discrete Fracture Networks Discrete Fracture and Matrix
Discretization spaces Section 3	Finite Element Methods 1D/2D/3D: • Primal formulation • High-order methods Virtual Element Methods 2D/3D: • Primal/mixed/divergence-free formulations • High-order methods • Advanced stabilization terms • Robust polynomial bases • Robustness to mesh distortion
Mesh management/optimization Sections 4, 5	Different mesh import format (CSV, VTK, OVM, OFF) Homemade mesh generations Adaptivity-coarsening techniques Mesh quality improvement TetGen, Triangle and Voro++ interfaces Polygonal and polyhedral numerical integration
Other useful interfaces Sections 4, 5	Solver: Eigen, PETSc, LAPACK interfaces Export: VTK interface

PolyDiM, to the best of our knowledge. For MATLAB users, we highlight VEMComp [17], mVEM [18], and VEMLAB [19]. In the C++ domain, we mention the libraries Veamy [20], and VEM++. Finally, the recently developed DUNE-VEM module [21] is worth noting, as it is available for both C++ and Python.

The structure of the paper is as follows. After briefly introducing the main notations in Section 2, we present the VEM framework in Section 3 by focusing on the local spaces supported by the library. Section 4 discusses the implementation details, outlining the design of local and global discrete matrices as well as integration and solving techniques available in the library. Finally, Section 5 reviews selected numerical experiments performed by the authors, where PolyDiM is successfully applied, and presents novel examples showcasing the recently integrated features.

2. Notations

Throughout this paper, we adopt the following notations. Let $d \in \{1,2,3\}$ be the geometric dimension of the problem. We denote by $\boldsymbol{x}$ and $\boldsymbol{y}$ some generic points in $\mathbb{R}^d$ and by x_i, with $i \in \{1,\dots,d\}$, the i-the component of the generic point $\boldsymbol{x}$ with respect to Cartesian axes. Let us denote by $E \subset \mathbb{R}^d$ a generic polytope, i.e. a polygon if $d = 2$ or a polyhedron if $d = 3$. We denote the generic edge of a polytope E by e and, if $d = 3$, we use the symbol F to refer to a generic face of the polyhedron E. We further denote by N_E^v, N_E^e, and N_E^f the number of vertices, edges, and faces of E, respectively. Moreover, $|E|$, $\boldsymbol{x}_E$ and $h_E = \max_{\boldsymbol{x},\boldsymbol{y}\in E} \|\boldsymbol{x}-\boldsymbol{y}\|$ represent the measure, the centroid and the diameter of the polygon E, respectively.

Let $\Omega \subset \mathbb{R}^d$ be an open bounded convex polytopal domain and let $\mathcal{T}_h$ be a decomposition of Ω into star-shaped polytopes E that satisfies standard mesh assumptions [22], where we fix, as usual, $h = \max_{E\in\mathcal{T}_h} h_E$.

We define $\mathcal{E}_{h,E}$ and $\mathcal{F}_{h,E}$ as the set of edges and faces of $E \in \mathcal{T}_h$, respectively. Consequently, the set $\mathcal{E}_{h,F}$ incorporates all the edges of the face F. We use ∂ operator to denote the boundary of a polygon or polyhedron. More specifically, if E is a polyhedron, ∂E is the set of faces contained in $\mathcal{F}_{h,E}$, while ∂E is the union of edges belonging to $\mathcal{E}_{h,E}$ if E is a polygon. Moreover, we set $\mathcal{E}_h = \bigcup_{E\in\mathcal{T}_h} \mathcal{E}_{h,E}$ and $\mathcal{F}_h = \bigcup_{E\in\mathcal{T}_h} \mathcal{F}_{h,E}$. We adopt the symbols $\boldsymbol{n}_{\partial E}$, $\boldsymbol{n}_F$ and $\boldsymbol{n}_e$ to denote the outward unit normal vectors to the boundary of E, to the face F and the edge e, respectively.

We follow standard notations for Sobolev and Hilbert spaces. Let us consider a generic open subset $\omega \subset \mathbb{R}^d$. Given two scalar functions $p, q \in L^2(\omega)$, two vector fields $\boldsymbol{u}$, $\boldsymbol{v} \in \left[L^2(\omega)\right]^d$ and two tensor fields $\boldsymbol{T}$, $\boldsymbol{\sigma} \in \left[L^2(\omega)\right]^{d\times d}$, we denote by

$$(p,q)_\omega = \int_\omega pq \ d\omega, \quad (\boldsymbol{u},\boldsymbol{v})_\omega = \int_\omega \boldsymbol{u}\cdot\boldsymbol{v} \ d\omega, \quad (\boldsymbol{T},\boldsymbol{\sigma})_\omega = \int_\omega \boldsymbol{T} : \boldsymbol{\sigma} \ d\omega,$$

where $\boldsymbol{T} : \boldsymbol{\sigma} := \sum_{i,j=1}^n \boldsymbol{T}_{ij}\boldsymbol{\sigma}_{ij}$. Furthermore, given a generic Sobolev space $\mathcal{H}$, we use the symbol $\|\cdot\|_{\mathcal{H}}$ to indicate the norm in $\mathcal{H}$.

Let $\boldsymbol{n}_\Gamma$ be the outward unit normal vector to the boundary Γ of Ω, we define the following spaces

$$H_0^1(\Omega) = \{v \in H^1(\Omega) : v_{|\Gamma} = 0 \text{ on } \Gamma\},$$
$$H(\mathrm{div};\Omega) = \left\{\boldsymbol{v} \in \left[L^d(\Omega)\right]^d : \nabla\cdot\boldsymbol{v} \in L^d(\Omega)\right\},$$
$$H_0(\mathrm{div};\Omega) = \left\{\boldsymbol{v} \in H(\mathrm{div};\Omega) : \boldsymbol{v}\cdot\boldsymbol{n}_\Gamma = 0 \text{ on } \Gamma\right\}.$$

Additionally, for $d = 2$, we define the functional space

$$H(\mathrm{rot}\,;\Omega) = \{\boldsymbol{v} \in \left[L^2(\Omega)\right]^2 : \mathrm{rot}\,\boldsymbol{v} \in L^2(\Omega)\},$$

whereas, for $d = 3$, we introduce

$$H(\mathbf{curl}\,;\Omega) = \left\{\boldsymbol{v} \in \left[L^2(\Omega)\right]^3 : \mathbf{curl}\,\boldsymbol{v} \in \left[L^2(\Omega)\right]^3\right\}.$$

2.1. Polynomial spaces

Let us denote by $\mathbb{P}_k^d(E)$ the set of the d-dimensional polynomials defined on E of degree less or equal to $k \geq 0$ and by $n_k^d = \dim \mathbb{P}_k^d(E) = \frac{(k+1)\dots(k+d)}{d!}$. For the ease of notation, we further set $\mathbb{P}_{-1}^d(E) = \{0\}$ and $n_{-1}^d = 0$ and we use the two natural functions $\ell_2 : \mathbb{N} \to \mathbb{N}^2$ and $\ell_3 : \mathbb{N} \to \mathbb{N}^3$ that associate:

$$\ell_2(1) = (0,0), \quad \ell_2(2) = (1,0), \quad \ell_2(3) = (0,1), \quad \ell_2(4) = (2,0), \dots$$
$$\ell_3(1) = (0,0,0), \quad \ell_3(2) = (1,0,0), \quad \ell_3(3) = (0,1,0), \quad \ell_3(4) = (0,0,1),$$
$$\ell_3(5) = (2,0,0), \dots$$

On each polytope E, we define the set of scaled monomials of degree less or equal to $k \geq 0$ as the set

$$\mathcal{M}_k^d(E) = \left\{\mathbb{m}_\alpha^{k,d} = \left(\frac{\boldsymbol{x} - \boldsymbol{x}_E}{h_E}\right)^{\boldsymbol{\alpha}} : \boldsymbol{\alpha} = \ell_d(\alpha) \in \mathbb{N}^d, \alpha = 1, \dots, n_k^d\right\}, \tag{1}$$

which is a polynomial basis for $\mathbb{P}_k^d(E)$. Other polynomial bases can be used as well. Examples are offered by the so-called *MGS basis*, which is described in Mascotto [23], Dassi and Mascotto [24] and tested by the authors in Berrone et al. [12], and by Berrone and Borio [25], Cicuttin [26]. Therefore, for the sake of generality, we introduce the generic set of polynomial basis functions $\mathcal{P}_k^d(E) = \{\mathbb{p}_\alpha^{k,d}\}_{\alpha=1}^{n_k^d}$ for the polynomial space $\mathbb{P}_k^d(E)$ defined over a generic polytope E.

Starting from a generic polynomial basis for $\mathbb{P}_k^d(E)$, an easily computable basis $\{\boldsymbol{p}_I^{k,d}\}_{I=1}^{dn_k^d}$ for $\left[\mathbb{P}_k^d(E)\right]^d$ can be built as

$$\boldsymbol{p}_I^{k,d} = \begin{cases} \begin{bmatrix} \mathbb{p}_I^{k,d} \\ \vdots \\ 0 \end{bmatrix} & I = 1, \dots, n_k^d, \\ \dots & \\ \begin{bmatrix} 0 \\ \vdots \\ \mathbb{p}_{I-(d-1)n_k}^{k,d} \end{bmatrix} & I = (d-1)n_k^d + 1, \dots, dn_k. \end{cases} \tag{2}$$

Furthermore, we introduce the vector-polynomial space

$$\mathcal{G}_k^\nabla(E) = \nabla\mathbb{P}_{k+1}^d(E) := \mathrm{span}\left\{\boldsymbol{g}_\alpha^{\nabla,k}\right\}_{\alpha=1}^{n_k^\nabla} \tag{3}$$

and its orthogonal complement $\mathcal{G}_k^\perp(E) = \mathrm{span}\left\{\boldsymbol{g}_\alpha^{\perp,k}\right\}_{\alpha=1}^{n_k^\perp}$ in $\left[\mathbb{P}_k^d(E)\right]^d$, which satisfies

$$\left[\mathbb{P}_k^d(E)\right]^d = \mathcal{G}_k^\nabla(E) \bigoplus \mathcal{G}_k^\perp(E), \tag{4}$$

where $\bigoplus$ is the direct sum operator, and

$$\dim\left[\mathbb{P}_k^d(E)\right]^d = dn_k^d,$$

$$n_k^\nabla = \dim \mathcal{G}_k^\nabla(E) = n_{k+1}^d - 1,$$

$$n_k^\perp = \dim \mathcal{G}_k^\perp(E) = dn_k^d - n_k^\nabla.$$

Basis functions for the $\mathcal{G}_k^\nabla(E)$ space can be written as

$$\boldsymbol{g}_\alpha^{\nabla,k} = \nabla\mathbb{p}_{\alpha+1}^{k+1,d} = \sum_{I=1}^{dn_k} \mathbf{T}_{\alpha I}^{\nabla,k} \boldsymbol{p}_I^{k,d} \quad \forall\alpha = 1, \dots, n_k^\nabla,$$

where $\mathbf{T}^{\nabla,k} \in \mathbb{R}^{n_k^\nabla \times dn_k}$ is the coefficient matrix that expresses gradients of the polynomial functions $\{\mathbb{p}_\alpha^{k+1,d}\}_{\alpha=2}^{n_{k+1}^d}$ with respect to the polynomial basis $\{\boldsymbol{p}_I^{k,d}\}_{I=1}^{dn_k}$ of $\left[\mathbb{P}_k^d(E)\right]^d$. Moreover, each $\boldsymbol{g}_\alpha^{\perp,k}$ function can be written as

$$\boldsymbol{g}_\alpha^{\perp,k} = \sum_{I=1}^{dn_k} \mathbf{T}_{\alpha I}^{\perp,k} \boldsymbol{p}_I^k \quad \forall\alpha = 1, \dots, n_k^\perp,$$

where $\mathbf{T}^{\perp,k} \in \mathbb{R}^{n_k^\perp \times dn_k}$ is the matrix whose rows define an *Euclidean* orthonormal basis for the nullspace of the $\mathbf{T}^{\nabla,k}$ matrix. Thus, by considering the singular value decomposition of $\mathbf{T}^{\nabla,k} = \mathbf{U}\,\mathbf{\Sigma}(\mathbf{V})^T$, we can define $\mathbf{T}^{\perp,k}$ as

$$\mathbf{T}^{\perp,k} = \left[\mathbf{V}(:, n_k^\nabla + 1 : dn_k^d)\right]^T,$$

where $\mathbf{V}(:, n_k^\nabla + 1 : dn_k^d)$ is the sub-matrix of $\mathbf{V}$ made up of all its rows and of the columns running from the $(n_k^\nabla + 1)$th to the dn_k^dth. As a consequence,

$$\mathbf{T}^{\nabla,k}\left(\mathbf{T}^{\perp,k}\right)^T = \mathbf{O}.$$

Remark 1. We observe that the space $\mathcal{G}_k^\perp(E)$ could be replace by each complement $\mathcal{G}_k^\oplus(E)$ of $\mathcal{G}_k^\nabla(E)$ in $\left[\mathbb{P}_k^d(E)\right]^d$. For example, $\mathcal{G}_k^\oplus(E) = \boldsymbol{x}^\perp\mathbb{P}_{k-1}(E)$ with $\boldsymbol{x}^\perp := [x_2, -x_1]^T$ if $d = 2$, whereas it can be written as $\mathcal{G}_k^\oplus(E) = \boldsymbol{x} \wedge \left[\mathbb{P}_{k-1}^3(E)\right]^3$, if $d = 3$ [27].

To perform efficient matrix-based computations in PolyDiM, the evaluations of such polynomial basis functions at given evaluation points are stored in a Vandermonde matrix, whose (i, j)th entry represents the evaluation of the jth basis function at the ith evaluation point. These matrices are built through functions available in the namespace Polydim::Utilities .

3. The virtual element framework

In this section, we introduce the Virtual Element Method framework, focusing on the definition and structure of the local spaces implemented in the PolyDiM library. We also describe the main VEM projection operators and the stabilization terms, highlighting their role in ensuring the method consistency and stability properties.

In the following, we will refer to the *primal formulation* when the discretization involves only a single variable u (or $\boldsymbol{u}$ when considering vector-problems). On the other hand, the *mixed formulation* will refer to a discretization involving two variables $(\boldsymbol{u}, p)$, to which we refer as the velocity field and the pressure variable, respectively. Finally, *divergence-free formulation* refers to the discretization of a problem involving two variables $(\boldsymbol{u}, p)$, again called the velocity and the pressure fields, whose related velocity space is point-wise divergence-free [28].

For any $k \in \mathbb{N}$ and for each polytope or polytopal face $\mathcal{O}$, we introduce the following polynomial projectors:

- the L^2-projection $\Pi_k^{0,\mathcal{O}} : L^2(\mathcal{O}) \to \mathbb{P}_k^d(\mathcal{O})$, which is defined for all $v \in L^2(\mathcal{O})$ by

$$\left(v - \Pi_k^{0,\mathcal{O}} v, \mathbb{p}\right)_\mathcal{O} = 0, \quad \forall\mathbb{p} \in \mathbb{P}_k^d(\mathcal{O}), \tag{5}$$

  with natural extension $\Pi_k^{0,\mathcal{O}} : \left[L^2(\mathcal{O})\right]^d \to \left[\mathbb{P}_k^d(\mathcal{O})\right]^d$ for vector fields $\boldsymbol{v} \in \left[L^2(\mathcal{O})\right]^d$.
- the H^1-seminorm projection $\Pi_k^{\nabla,\mathcal{O}} : H^1(\mathcal{O}) \to \mathbb{P}_k^d(\mathcal{O})$, such that for any $v \in H^1(\mathcal{O})$

$$\begin{cases} \left(\nabla v - \nabla\Pi_k^{\nabla,\mathcal{O}} v, \nabla\mathbb{p}\right)_\mathcal{O} = 0 \quad \forall\mathbb{p} \in \mathbb{P}_k^d(\mathcal{O}), \\ P_0(\Pi_k^{\nabla,E} v - v) = 0, \end{cases} \text{where } P_0(v) = \begin{cases} \int_{\partial\mathcal{O}} v & \text{if } k = 1, \\ \int_\mathcal{O} v & \text{if } k \geq 2. \end{cases} \tag{6}$$

  with clear extension $\Pi_k^{\nabla,\mathcal{O}} : \left[H^1(\mathcal{O})\right]^d \to \left[\mathbb{P}_k^d(\mathcal{O})\right]^d$ for vector fields $\boldsymbol{v} \in \left[H^1(\mathcal{O})\right]^d$.

3.1. Primal $H^1(\Omega)$-conforming

Let us consider the following Poisson problem

$$\begin{cases} \nabla \cdot (-\boldsymbol{\kappa} \nabla u) = f & \text{in } \Omega, \\ u = 0 & \text{on } \Gamma, \end{cases} \tag{7}$$

where $\boldsymbol{\kappa} \in \mathbb{R}^{d\times d}$ is a symmetric positive-definite tensor and f is a sufficiently regular scalar field. Here, for the sake of simplicity, we impose homogeneous Dirichlet boundary conditions, but general boundary conditions as well as more general elliptic problems can be handled with PolyDiM.

Setting $\mathcal{V} := H^1_0(\Omega)$, the variational formulation of problem (7) reads as: *Find $u \in \mathcal{V}$, such that*

$$a(u, v) = (f, q)_\Omega \qquad \forall v \in \mathcal{V}, \tag{8}$$

where

$$a(u, v) := (\boldsymbol{\kappa} \nabla u, \nabla v)_\Omega \quad \forall u, v \in \mathcal{V}. \tag{9}$$

3.1.1. The local space and the Polydim::VEM::PCC classes

Let $k \geq 1$, for each polygon or polygonal face $\mathcal{O}$, we introduce the space

$$\mathbb{B}_k(\partial\mathcal{O}) = \left\{ v \in C^0(\partial\mathcal{O}) : v_{|e} \in \mathbb{P}^1_k(e) \ \forall e \in \mathcal{E}_{h,\mathcal{O}} \right\},$$

which allows us to define the two-dimensional local primal virtual element space on the element E as

$$\begin{aligned} \mathcal{V}^2_{h,k,\ell}(E) = \big\{ v \in H^1(E) : &(i)\ \ v_{|\partial E} \in \mathbb{B}_k(\partial E), \\ &(ii)\ \Delta v \in \mathbb{P}^2_{k+\ell}(E), \\ &(iii) \big(v - \Pi^{\nabla,E}_k v, \mathbb{p}\big)_E = 0 \ \forall \mathbb{p} \in \mathbb{P}^2_{k+\ell}(E) \setminus \mathbb{P}^2_{k-2}(E) \big\}. \end{aligned} \tag{10}$$

The set $\mathbb{P}^d_{k+\ell}(E) \setminus \mathbb{P}^d_{k-2}(F)$ denotes the union of set of d-dimensional homogeneous polynomials defined on E from degree $k-1$ to $k+\ell$. If $d = 3$, for each polyhedron $E \in \mathcal{T}_h$, we further consider the boundary space

$$\mathbb{U}_{k,\ell}(\partial E) = \left\{ v \in C^0(\partial E) : v_{|F} \in \mathcal{V}^2_{h,k,\ell}(F) \ \forall F \in \mathcal{F}_{h,E} \right\}, \tag{11}$$

and then we define three-dimensional local primal virtual element space as

$$\begin{aligned} \mathcal{V}^3_{h,k,\ell}(E) = \big\{ v \in H^1(E) : &(i)\ \ v_{|\partial E} \in \mathbb{U}_{k,\ell}(\partial E), \\ &(ii)\ \Delta v \in \mathbb{P}^3_{k+\ell}(E), \\ &(iii) \big(v - \Pi^{\nabla,E}_k v, \mathbb{p}\big)_E = 0 \ \forall \mathbb{p} \in \mathbb{P}^3_{k+\ell}(E) \setminus \mathbb{P}^3_{k-2}(E) \big\}. \end{aligned} \tag{12}$$

We want to remark that the Property (iii) which defines functions in the local virtual element spaces is usually called the *enhancement property*.

For each $v \in \mathcal{V}^d_{h,k,\ell}(E)$, a possible choice of the local Degrees of Freedom (DOFs) are:

- **Vertex DOFs**: the values of v at each vertex of E.
- **Edge DOFs**: the values of v at the $k-1$ internal Gauss–Lobatto quadrature points on each edge $e \in \mathcal{E}_{h,E}$.
- **Face DOFs**: if $d = 3$, the internal moments on each face $F \in \mathcal{F}_{h,E}$ defined as
$$\frac{1}{|F|} \int_F v\, \mathbb{p} \qquad \forall \mathbb{p} \in \mathcal{P}^2_{k-2}(F). \tag{13}$$
- **Internal DOFs**: the internal moments:
$$\frac{1}{|E|} \int_E v\, \mathbb{p} \qquad \forall \mathbb{p} \in \mathcal{P}^d_{k-2}(E). \tag{14}$$

The total number of local degrees of freedom is

$$N^{\text{dof}}_E := \begin{cases} N^v_E + N^e_E(k-1) + \frac{(k-1)k}{2} & \text{if } d = 2, \\ N^v_E + N^e_E(k-1) + N^f_E \frac{(k-1)k}{2} + \frac{(k-1)k(k+1)}{6} & \text{if } d = 3. \end{cases} \tag{15}$$

A graphical illustration of such degrees of freedom for $k = 3$ in two-dimensions is shown in Fig. 1a. It is important to note that the DOF positions shown are only illustrative. Indeed, many DOFs are not associated with specific points on the element.

Proposition 1 ([29–31]). *Let $\mathcal{V}^d_{h,k,\ell}(E)$ be the space defined in (10) for $d = 2$ and in (12) for $d = 3$. The local DOFs allow us to compute exactly (up to machine precision) the element projections*

$$\Pi^{0,E}_{k-2} : \mathcal{V}^d_{h,k,\ell}(E) \to \mathbb{P}^d_{k-2}(E), \quad \Pi^{\nabla,E}_k : \mathcal{V}^d_{h,k,\ell}(E) \to \mathbb{P}^d_k(E),$$
$$\Pi^{0,E}_{k-1} : \nabla\mathcal{V}^d_{h,k,\ell}(E) \to \left[\mathbb{P}^d_{k-1}(E)\right]^d$$

as defined in (5) and (6), i.e. we are able to compute the polynomials $\Pi^{0,E}_{k-2} v_h$, $\Pi^{\nabla,E}_k v_h$ and $\Pi^0_{k-1} \nabla v_h$, for any $v_h \in \mathcal{V}^d_{h,k}(E)$, using only the information given by the local degrees of freedom of v_h. Moreover, by exploiting the enhancement property, we are also able to compute the higher-order projections

$$\Pi^{0,E}_* : \mathcal{V}^d_{h,k,\ell}(E) \to \mathbb{P}^d_*(E) \quad \forall * = k-1, \dots, k+\ell.$$

PolyDiM offers the possibility to choose among different sets of degrees of freedom and different approaches to build the polynomial projections of virtual element functions. In particular, on the current release the reader can find:

- The monomial approach, that exploits the monomial basis defined in (1) in the definition of local degrees of freedom (13) and (14) as well as in the definition of local projections. The corresponding approach is implemented following [32] and the local space can be built exploiting classes Polydim::VEM::PCC::VEM_PCC_2D_LocalSpace if $d = 2$ and Polydim::VEM::PCC::VEM_PCC_3D_LocalSpace if $d = 3$.
- The orthonormal approach introduced in Mascotto [23], Dassi and Mascotto [24] and tested by the authors in Berrone et al. [12]. This approach guarantees well-conditioned local and global matrices for the higher-values of the local polynomial degree k. The related local space can be built exploiting Polydim::VEM::PCC::VEM_PCC_2D_Ortho_LocalSpace if $d = 2$ and Polydim::VEM::PCC::VEM_PCC_3D_Ortho_LocalSpace if $d = 3$.
- The novel (B-F) inertial approach introduced by the authors in Berrone et al. [12] that reveals to be robust in the presence of badly-shaped polytopes. The corresponding local spaces can be built exploiting the class Polydim::VEM::PCC::VEM_PCC_2D_Inertia_LocalSpace if $d = 2$ and Polydim::VEM::PCC::VEM_PCC_3D_Inertia_LocalSpace if $d = 3$.

Let us introduce the set $\{\varphi_{j,E}\}_{j=1}^{N^{\text{dof}}_E}$ of Lagrange basis functions related to the aforementioned local DOFs, where the numbering j counts in order the vertex, the edge, the face, and the internal DOFs.

In the virtual element framework, basis functions are not polynomial as for the FEM and are not known in a closed-form in the interior of an element $E \in \mathcal{T}_h$. Thus, to solve a partial differential equation using VEM, we must resort to the projection operators defined in Proposition 1. All the information needed to compute such projections is stored in the data structure Polydim::VEM::PCC::VEM_PCC_dD_LocalSpace_Data .

The above-mentioned classes offer user-friendly interfaces that allows us to retrieve the values of the projection of the local basis functions $\{\varphi_{j,E}\}_{j=1}^{N^{\text{dof}}_E}$ and their derivatives in a given set of points. The list of available projection operators is defined in the enumeration Polydim::VEM::PCC::ProjectionTypes and coincides with projectors defined in Proposition 1.

We highlight that, since virtual element functions are known polynomial on ∂E, we use the exact expression of the Lagrange basis functions to evaluate boundary contributions.

3.1.2. The discrete primal variational formulation

We define the global Virtual Element space as Da Veiga et al. [22]:

$$\mathcal{V}^d_{h,k} = \{ v \in \mathcal{V} \cap C^0(\overline{\Omega}) : v \in \mathcal{V}^d_{h,k,0}(E) \ \forall E \in \mathcal{T}_h \}. \tag{16}$$

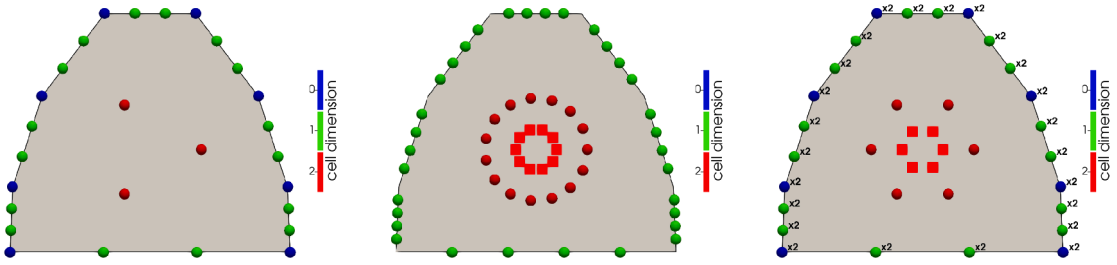


Fig. 1. Degrees of freedom for $k = 3$ and $d = 2$. Left: DOFs associated with the local primal space $\mathcal{V}^d_{h,k}$ defined in (10). Center: Degrees of freedom for the local mixed spaces. Circles refer to the velocity space $\boldsymbol{V}^d_{h,k}(E)$, defined in (30), whereas squares refer to the pressure space $Q^d_{h,k}(E)$. Right: Degrees of freedom for the local divergence-free spaces. Circles refer to the velocity space $\boldsymbol{V}^d_{h,k}(E)$, defined in (42), whereas squares refer to the pressure space $Q^d_{h,k}(E)$ defined in (44). The symbol $\times 2$ near a marker denotes the multiplicity of the related degree of freedom.

The continuous bilinear form (9) can be split according to the tessellation $\mathcal{T}_h$ as

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

In general, we are not able to compute the quantity

$$a^E(u_h, v_h) = \left(\boldsymbol{\kappa}\nabla u_h, \nabla v_h\right)_E \quad \forall u_h,\ v_h \in \mathcal{V}^d_{h,k}(E),$$

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 $a^E_h(\cdot,\cdot) : \mathcal{V}^d_{h,k}(E) \times \mathcal{V}^d_{h,k}(E) \to \mathbb{R}$ which satisfies the two following properties [30]:

- *Consistency*: For all $p \in \mathbb{P}^d_k(E)$ and for all $v_h \in \mathcal{V}^d_{h,k}(E)$

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

- *Stability*: There exist two positive constants $\alpha_*,\ \alpha^*$ independent of h such that

  $$\alpha_* a^E(v,v) \le a^E_h(v,v) \le \alpha^* a^E(v,v), \quad \forall v \in \mathcal{V}^d_{h,k}(E) : \ \Pi^{\nabla,E}_k v = 0. \tag{17}$$

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

$$a^E(u_h, v_h) = a^E(\Pi^{\nabla,E}_k u_h, \Pi^{\nabla,E}_k v_h) + a^E((I - \Pi^{\nabla,E}_k)u_h, (I - \Pi^{\nabla,E}_k)v_h), \tag{18}$$

where the equality is due to the orthogonality of $\Pi^{\nabla,E}_k$ with respect to the scalar product induced by $a^E(,)$. The first term in the right-hand side of (18) 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 (17). In [29], the local bilinear form $a^E(\Pi^{\nabla,E}_k u_h, \Pi^{\nabla,E}_k v_h)$ is replaced by

$$\left(\boldsymbol{\kappa}\Pi^{0,E}_{k-1}\nabla u_h, \Pi^{0,E}_{k-1}\nabla v_h\right)_E$$

to avoid loss of accuracy for the high orders of the method when dealing with variable coefficients. We observe that, when $k = 1$, the two choices coincide with each other. Finally, the local virtual discrete bilinear form is

$$a^E_h(u_h, v_h) := \left(\boldsymbol{\kappa}\Pi^{0,E}_{k-1}\nabla u_h, \Pi^{0,E}_{k-1}\nabla v_h\right)_E + S^E((I - \Pi^{\nabla,E}_k)u_h, (I - \Pi^{\nabla,E}_k)v_h),$$

which is computable and satisfies the consistency and the stability property [30]. Now, let us define dof^E_i, for each $i = 1, \dots, N^{\mathrm{dof}}_E$ and each $E \in \mathcal{T}_h$, as the operator that associates with each sufficiently smooth function φ its ith local degree of freedom $\mathrm{dof}^E_i(\varphi)$. A standard choice for the stabilization term is given by the *dofi-dofi* stabilization term

$$S^E(u_h, v_h) = h^{d-2} \sum_{i=1}^{N^{\mathrm{dof}}_E} \mathrm{dof}^E_i(u_h)\,\mathrm{dof}^E_i(v_h). \tag{19}$$

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 coefficients. Other stabilization methods have been proposed in the literature, which may take integral forms [33] or be a variant of the dofi-dofi stabilization, such as the *D*-recipe version introduced in Da Veiga et al. [22]. 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 (8) reads as: *Find* $u_h \in \mathcal{V}_{h,k}$ *such that:*

$$\sum_{E\in\mathcal{T}_h} a^E_h(u_h, v_h) = \sum_{E\in\mathcal{T}_h} \left(f, \Pi^{0,E}_{k-1} v_h\right)_E \quad \forall v_h \in \mathcal{V}_{h,k}. \tag{20}$$

The problem (20) has a unique solution $u_h \in \mathcal{V}_{h,k}$ and, for h sufficiently small, the following a priori error estimates hold true

$$\|u - u_h\|_{L^2(\Omega)} = O(h^{k+1}), \quad \|\nabla u - \nabla u_h\|_{[L^2(\Omega)]^d} = O(h^k). \tag{21}$$

3.1.3. A simple extension to deal with vector-problems

A simple way to deal with vector-problems, as the elasticity problems, is to build the local space for the primal vector-variable $\boldsymbol{u} \in \boldsymbol{V} := \left[H^1_0(\Omega)\right]^d$, with $\boldsymbol{u} = \begin{bmatrix} u_1 \\ \dots \\ u_d \end{bmatrix}$, as the cartesian product of the corresponding scalar local space [34]. Specifically, we define the virtual element space as $\boldsymbol{V}^d_{h,k} = [\mathcal{V}^d_{h,k}]^d$. By proceeding in this way, the number of local degrees of freedom for a vector-valued function $\boldsymbol{v} \in \boldsymbol{V}^d_{h,k}$ becomes $d\,N^{\mathrm{dof}}$.

Let us, for instance, consider the elasticity problem with homogeneous Dirichlet boundary conditions:

$$\begin{cases} -\nabla\cdot(2\mu\boldsymbol{\epsilon}(\boldsymbol{u}) + \lambda \mathrm{div}\boldsymbol{u}\boldsymbol{I}) = \boldsymbol{f} & \text{in } \Omega, \\ \boldsymbol{u} = \boldsymbol{0} & \text{on } \Gamma, \end{cases} \tag{22}$$

where $\boldsymbol{\epsilon}(\boldsymbol{u})$ denotes the symmetric gradient of the displacement $\boldsymbol{u}$, $\boldsymbol{I}$ is the identity tensor, λ and μ are positive coefficients (Lamé coefficients), while $\boldsymbol{f}$ is a vector-valued function belonging to $\left[L^2(\Omega)\right]^d$. Its virtual element discretization reads as: *Find* $\boldsymbol{u}_h \in \boldsymbol{V}^d_{h,k}$ *such that*

$$\sum_{E\in\mathcal{T}_h} \Big[\left(2\mu\Pi^{0,E}_{k-1}\boldsymbol{\epsilon}(\boldsymbol{u}_h), \Pi^{0,E}_{k-1}\boldsymbol{\epsilon}(\boldsymbol{v}_h)\right)_E + \mu S^E((I - \Pi^{\nabla,E}_k)\boldsymbol{u}_h, (I - \Pi^{\nabla,E}_k)\boldsymbol{v}_h) + \left(\lambda\Pi^{0,E}_{k-1}\mathrm{div}\boldsymbol{u}_h, \Pi^{0,E}_{k-1}\mathrm{div}\boldsymbol{v}_h\right)_E\Big] = \sum_{E\in\mathcal{T}_h} \left(\boldsymbol{f}, \Pi^{0,E}_{k-1}\boldsymbol{v}_h\right)_E \quad \forall \boldsymbol{v}_h \in \boldsymbol{V}^d_{h,k} \tag{23}$$

Moreover, $\forall \boldsymbol{v} \in \left[H^1(E)\right]^d$, we define [1]

$$\Pi^{0,E}_{k-1}\boldsymbol{\epsilon}(\boldsymbol{v}) = \frac{1}{2}\left(\Pi^{0,E}_{k-1}\nabla\boldsymbol{v} + (\Pi^{0,E}_{k-1}\nabla\boldsymbol{v})^T\right),$$
$$\Pi^{0,E}_{k-1}\mathrm{div}\boldsymbol{v} = \Pi^{0,E}_{k-1}\mathrm{tr}(\nabla\boldsymbol{v}) = \mathrm{tr}(\Pi^{0,E}_{k-1}\nabla\boldsymbol{v}),$$

where $\mathrm{tr}(\cdot)$ denotes the trace of a tensor-field.

3.2. Mixed $H(\mathrm{div};\Omega)$-conforming

The Poisson problem stated in its mixed-formulation reads as:

$$\begin{cases} \boldsymbol{K}\boldsymbol{u} = -\nabla p & \text{in } \Omega, \\ \nabla\cdot\boldsymbol{u} = f & \text{in } \Omega, \\ p = 0 & \text{on } \Gamma. \end{cases} \tag{24}$$

Setting

$$\boldsymbol{V} := H(\mathrm{div};\Omega), \quad Q := L^2(\Omega), \tag{25}$$

the variational formulation of (24) reads as: *Find* $\boldsymbol{u}\in\boldsymbol{V}$, *and* $p\in Q$ *such that*

$$\begin{cases} a(\boldsymbol{u},\boldsymbol{v}) - (p,\nabla\cdot\boldsymbol{v})_\Omega = 0 & \forall \boldsymbol{v}\in\boldsymbol{V}, \\ (\nabla\cdot\boldsymbol{u},q)_\Omega = (f,q)_\Omega & \forall q\in Q, \end{cases} \tag{26}$$

where

$$a(\boldsymbol{u},\boldsymbol{v}) := (\boldsymbol{K}\boldsymbol{u},\boldsymbol{v})_\Omega \quad \forall \boldsymbol{u},\boldsymbol{v}\in\boldsymbol{V}. \tag{27}$$

3.2.1. The local space and the Polydim::VEM::MCC classes

Following [35], for any integer $k\geq 0$, if $d=2$, we define the local mixed virtual element space for the velocity variable $\boldsymbol{u}$ as

$$\begin{aligned} \boldsymbol{V}^2_{h,k}(E) = \big\{ \boldsymbol{v}_h \in H(\mathrm{div};E)\cap H(\mathrm{rot};E) \text{ s.t. } & (i)\ \ \boldsymbol{v}_h\cdot\boldsymbol{n}_e \in \mathbb{P}^2_k(e)\ \forall e\in\mathcal{E}_{h,E}, \\ & (ii)\ \nabla\cdot\boldsymbol{v}_h\in\mathbb{P}^2_k(E), \\ & (iii)\ \mathrm{rot}\,\boldsymbol{v}_h\in\mathbb{P}^2_{k-1}(E)\big\}, \end{aligned} \tag{28}$$

whereas, for $d=3$, we define the space as Da Veiga et al. [36]

$$\begin{aligned} \boldsymbol{V}^3_{h,k}(E) = \big\{ \boldsymbol{v}_h \in H(\mathrm{div};E)\cap H(\mathbf{curl};E) \text{ s.t. } & (i)\ \ \boldsymbol{v}_h\cdot\boldsymbol{n}_f \in \mathbb{P}^3_k(f)\ \forall f\in\mathcal{F}_{h,E}, \\ & (ii)\ \nabla\cdot\boldsymbol{v}_h\in\mathbb{P}^3_k(E), \\ & (iii)\ \mathbf{curl}\,\boldsymbol{v}_h\in\mathbf{curl}\left[\mathbb{P}^3_k(E)\right]^3\big\}. \end{aligned} \tag{29}$$

The following set of local degrees of freedom is unisolvent for $\boldsymbol{V}^d_{h,k}(E)$ (see [32,36] for more details and Fig. 1b for a graphical illustration): for each $\boldsymbol{v}_h\in\boldsymbol{V}^d_{h,k}(E)$,

- **Boundary DOFs**: if $d=2$, the values of $\boldsymbol{v}_h\cdot\boldsymbol{n}_e$ at the $k+1$ Gauss quadrature points internal on each edge $e\in\mathcal{E}_{h,E}$, or, more generally,

$$\begin{cases} \frac{1}{|e|}\int_e \boldsymbol{v}_h\cdot\boldsymbol{n}_e\ p & \forall p\in\mathcal{P}^1_k(e) \quad \forall e\in\mathcal{E}_{h,E} \quad \text{if } d=2, \\ \frac{1}{|f|}\int_f \boldsymbol{v}_h\cdot\boldsymbol{n}_f\ p & \forall p\in\mathcal{P}^2_k(f) \quad \forall f\in\mathcal{F}_{h,E} \quad \text{if } d=3. \end{cases} \tag{30}$$

  We note that this choice automatically ensures the continuity of the flux $\boldsymbol{v}_h\cdot\boldsymbol{n}$ across two adjacent elements.
- **Internal ∇ DOFs**:

$$\frac{1}{|E|}\int_E \boldsymbol{v}_h\cdot\boldsymbol{g}^{\nabla,k-1}_\alpha \quad \forall\alpha=1,\dots,n^\nabla_{k-1}. \tag{31}$$

- **Internal $\perp$ DOFs**:

$$\frac{1}{|E|}\int_E \boldsymbol{v}_h\cdot\boldsymbol{g}^{\perp,k}_\alpha \quad \forall\alpha=1,\dots,n^\perp_k. \tag{32}$$

The dimension of the local mixed virtual element space is given by

$$N^{\mathrm{dof}}_E = \dim\boldsymbol{V}^d_{h,k}(E) = N^*_E n^{d-1}_k + n^\nabla_{k-1} + n^\perp_k, \quad \text{with} \quad *= \begin{cases} e & \text{if } d=2, \\ f & \text{if } d=3. \end{cases} \tag{33}$$

Proposition 2 ([35,36]). *Let* $\boldsymbol{V}^d_{h,k}(E)$ *be the space defined in* (28) *or in* (29). *The DOFs* (30), (31) *and* (32) *allow us to compute exactly (up to machine precision) the element projection*

$$\Pi^{0,E}_k : \boldsymbol{V}^d_{h,k}(E)\to\left[\mathbb{P}^d_k(E)\right]^d$$

as defined in (5), *i.e. we are able to compute the polynomial* $\Pi^{0,E}_k\boldsymbol{v}_h$, *for any* $\boldsymbol{v}_h\in\boldsymbol{V}^d_{h,k}(E)$, *using only the information given by the local degrees of freedom of* $\boldsymbol{v}_h$.

We further introduce the local mixed virtual element space $Q^d_{h,k}(E)$ for the pressure variable p as the space of polynomials $\mathbb{P}^d_k(E)$, i.e. we set $Q^d_{h,k}(E)=\mathbb{P}^d_k(E)$.

In particular, our library offers some alternatives to properly define the degrees of freedom related to both the pressure and the velocity variables and to compute the polynomial projections of the virtual velocity basis functions. This list includes

- The standard monomial approach detailed in Da Veiga et al. [32], where the sets $\mathcal{G}^\nabla_k(E)$ and $\mathcal{G}^\perp_k(E)$ are defined through the scalar monomials basis (1). The corresponding approach is implemented in the class Polydim::VEM::MCC::VEM_MCC_2D_Velocity_LocalSpace if $d=2$ and Polydim::VEM::MCC::VEM_MCC_3D_Velocity_LocalSpace if $d=3$.
- The partial orthonormal approaches introduced in Berrone et al. [13] by the authors. These approaches guarantee well-conditioned local and global matrices for the higher-values of the local polynomial degree k with respect to the monomial approach. The related local spaces can be built exploiting Polydim::VEM::MCC::VEM_MCC_2D_Partial_Velocity_LocalSpace and Polydim::VEM::PCC::VEM_MCC_2D_Ortho_Velocity_LocalSpace, respectively, for $d=2$.

The aforementioned classes for the two-dimensional spaces use the Gauss quadrature points to define the boundary degrees of freedom. Orthonormal edge degrees of freedom are further available for the monomial and the orthonormal approaches. The construction of virtual projections through these degrees of freedom is detailed by the authors in Berrone et al. [37] and implemented in the classes Polydim::VEM::MCC::VEM_MCC_2D_EdgeOrtho_Velocity_LocalSpace and Polydim::VEM::PCC::VEM_MCC_2D_Ortho_EdgeOrtho_Velocity_LocalSpace.

3.2.2. The discrete mixed variational formulation

We define the global mixed virtual element spaces for both velocity and pressure variables as

$$\boldsymbol{V}_{h,k} = \{\boldsymbol{v}_h\in\boldsymbol{V} \text{ s.t. } \boldsymbol{v}_{h|E}\in\boldsymbol{V}^d_{h,k}(E)\ \forall E\in\mathcal{T}_h\},$$
$$Q_{h,k} = \{q_h\in Q \text{ s.t. } q_{h|E}\in Q^d_{h,k}(E)\ \forall E\in\mathcal{T}_h\}.$$

As done for the primal formulation, we split the continuous bilinear form (27) according to the tessellation $\mathcal{T}_h$ as

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

Then, since we are not able to compute the quantity

$$a^E(\boldsymbol{u}_h,\boldsymbol{v}_h) = \left(\boldsymbol{K}\boldsymbol{u}_h,\boldsymbol{v}_h\right)_E \quad \forall\boldsymbol{u}_h,\ \boldsymbol{v}_h\in\boldsymbol{V}^d_{h,k}(E), \tag{34}$$

we define the local virtual discrete bilinear form as the computable bilinear form

$$a^E_h(\boldsymbol{u}_h,\boldsymbol{v}_h) := \left(\boldsymbol{K}\,\Pi^0_k\boldsymbol{u}_h,\Pi^0_k\boldsymbol{v}_h\right)_{0,E} + S^E((I-\Pi^0_k)\boldsymbol{u}_h,(I-\Pi^0_k)\boldsymbol{v}_h), \tag{35}$$

where the stabilization term $S^E(\cdot,\cdot)$ is any symmetric and positive definite bilinear form that scales as (34). As in Da Veiga et al. [32,35], a trivial choice is represented by the dofi-dofi stabilization term:

$$S^E(\boldsymbol{u}_h,\boldsymbol{v}_h) = |E|\sum_{r=1}^{N^{\mathrm{dof}}_E}\mathrm{dof}_r\left(\boldsymbol{u}_h\right)\mathrm{dof}_r\left(\boldsymbol{v}_h\right).$$

Finally, the mixed VEM approximation of (26) reads as: *Find* $(\boldsymbol{u}_h,p_h)\in\boldsymbol{V}_{h,k}\times Q_{h,k}$ *such that*:

$$\begin{cases} \sum_{E\in\mathcal{T}_h}\left[a^E_h\left(\boldsymbol{u}_h,\boldsymbol{v}_h\right) - \left(p_h,\nabla\cdot\boldsymbol{v}_h\right)_E\right] = 0 & \forall\boldsymbol{v}_h\in\boldsymbol{V}_{h,k}, \\ \sum_{E\in\mathcal{T}_h}\left(\nabla\cdot\boldsymbol{u}_h,q_h\right)_E = \sum_{E\in\mathcal{T}_h}\left(f,q_h\right)_E & \forall q_h\in Q_{h,k}. \end{cases} \tag{36}$$

The problem (36) has unique solution $(\boldsymbol{u}_h, p_h) \in \boldsymbol{V}_{h,k} \times Q_{h,k}$ and, for h sufficiently small, the following a priori error estimates hold true

$$\|p - p_h\|_{L^2(\Omega)} = O(h^{k+1}), \quad \|\boldsymbol{u} - \boldsymbol{u}_h\|_{[L^2(\Omega)]^d} = O(h^{k+1}). \tag{37}$$

Furthermore, the following superconvergence result holds true.

Theorem 1 (Superconvergence result [35]). *Let p_h be the solution to (36) and let $p_I \in Q_{h,k}$ be the interpolant of p. Then, for h sufficiently small,*

$$\|p_I - p_h\|_{L^2(\Omega)} = O(h^{k+2}). \tag{38}$$

3.3. The divergence-free formulation

Let us now consider the Stokes problem on $\Omega \subset \mathbb{R}^d$ with homogeneous Dirichlet boundary conditions:

$$\begin{cases} \text{Find } (\boldsymbol{u}, p) \text{ such that} & \\ -\nu\Delta\boldsymbol{u} - \nabla p = \boldsymbol{f} & \text{in } \Omega, \\ \operatorname{div}\boldsymbol{u} = 0 & \text{in } \Omega, \\ \boldsymbol{u} = \boldsymbol{0} & \text{on } \Gamma = \partial\Omega, \end{cases} \tag{39}$$

where $\boldsymbol{f} \in \left[L^2(\Omega)\right]^d$ represents the external source and $\nu \in L^\infty(\Omega)$ is the uniformly positive viscosity.

Let us consider the spaces

$$\boldsymbol{V} := \left[H_0^1(\Omega)\right]^d, \quad Q := L_0^2(\Omega) = \left\{ q \in L^2(\Omega) : \int_\Omega q = 0 \right\}. \tag{40}$$

The variational formulation of the Stokes problem (39) reads as

$$\begin{cases} \text{Find } (\boldsymbol{u}, p) \in \boldsymbol{V} \times Q \text{ such that} & \\ a(\boldsymbol{u}, \boldsymbol{v}) + (\operatorname{div}\boldsymbol{v}, p)_\Omega = (\boldsymbol{f}, \boldsymbol{v})_\Omega & \forall \boldsymbol{v} \in \boldsymbol{V} \\ (\operatorname{div}\boldsymbol{u}, q)_\Omega = 0 & \forall q \in Q \end{cases} \tag{41}$$

where

$$a(\boldsymbol{u}, \boldsymbol{v}) = \int_\Omega \nu \nabla\boldsymbol{u} : \nabla\boldsymbol{v} \quad \forall \boldsymbol{u}, \boldsymbol{v} \in \boldsymbol{V} \quad \text{and} \quad \nabla\boldsymbol{u} = \begin{bmatrix} \frac{\partial u_1}{\partial x_1} & \cdots & \frac{\partial u_1}{\partial x_d} \\ \vdots & \ddots & \vdots \\ \frac{\partial u_d}{\partial x_1} & \cdots & \frac{\partial u_d}{\partial x_d} \end{bmatrix}.$$

3.3.1. The local space and the Polydim::VEM::DF_PCC classes

Given $k \geq 2$, on each element $E \in \mathcal{T}_h$, we define the following finite two-dimensional divergence-free virtual element space for the velocity variable $\boldsymbol{u}$ [8]:

$$\begin{aligned} \boldsymbol{V}_{h,k}^2(E) := \Big\{ \boldsymbol{v} \in \left[H^1(E)\right]^2 : \ & (i) \quad \boldsymbol{v}_{|\partial E} \in \left[\mathbb{B}_k(\partial E)\right]^2, \\ & (ii) \quad -\Delta\boldsymbol{v} - \nabla s \in \mathcal{G}_k^\perp(E) \text{ for some } s \in L_0^2(E), \\ & (iii) \quad \operatorname{div}\boldsymbol{v} \in \mathbb{P}_{k-1}(E) \\ & (iv) \quad \left(\boldsymbol{v} - \Pi_k^{\nabla,E}\boldsymbol{v}, \boldsymbol{g}^{\perp,k}\right)_E = 0 \ \forall \boldsymbol{g}^{\perp,k} \in \mathcal{G}_k^\perp(E) \setminus \mathcal{G}_{k-2}^\perp(E) \Big\}. \end{aligned} \tag{42}$$

If $d = 3$, we define three-dimensional local primal divergence-free virtual element space for the velocity variable $\boldsymbol{u}$ as

$$\begin{aligned} \boldsymbol{V}_{h,k}^3(E) = \Big\{ \boldsymbol{v} \in \left[H^1(E)\right]^3 : & (i) \quad \boldsymbol{v}_{|\partial E} \in \left[\mathbb{U}_{k,1}(\partial E)\right]^3, \\ & (ii) \quad -\Delta\boldsymbol{v} - \nabla s \in \mathcal{G}_k^\perp(E) \text{ for some } s \in L_0^2(E), \\ & (iii) \quad \operatorname{div}\boldsymbol{v} \in \mathbb{P}_{k-1}(E) \\ & (iv) \quad \left(\boldsymbol{v} - \Pi_k^{\nabla,E}\boldsymbol{v}, \boldsymbol{g}^{\perp,k}\right)_E = 0 \ \forall \boldsymbol{g}^{\perp,k} \in \mathcal{G}_k^\perp(E) \setminus \mathcal{G}_{k-2}^\perp(E) \Big\}. \end{aligned} \tag{43}$$

where the space $\mathbb{U}_{k,1}(\partial E)$ is defined in (11). For the pressure variable p we set

$$Q_{h,k}^d(E) := \mathbb{P}_{k-1}^d(E). \tag{44}$$

Given a function $\boldsymbol{v} \in \boldsymbol{V}_{h,k}^d(E)$, we consider the following degrees of freedom (see Fig. 1c for a graphical illustration):

- **Vertex DOFs**: the values of $\boldsymbol{v}$ at the vertices of the polygon.
- **Edge DOFs**: the values of $\boldsymbol{v}$ at $k-1$ Gauss–Lobatto internal points of every edge $e \in \mathcal{E}_{h,E}$.
- **Face DOFs**: if $d = 3$, the internal moments on each face $F \in \mathcal{F}_{h,E}$, defined as

$$\frac{1}{|F|}\int_F \boldsymbol{v}\cdot\boldsymbol{n}_F \ \mathbb{p}_\alpha, \ \ \forall I = 1, \dots, 2n_{k-2}^2, \quad \frac{1}{|F|}\int_F \boldsymbol{v}_{\boldsymbol{\tau}}\cdot\boldsymbol{p}_I \ \ \forall I = 1, \dots, 2n_{k-2}^2, \tag{45}$$

  where $\boldsymbol{v}_{\boldsymbol{\tau}} = \boldsymbol{v} - \boldsymbol{v}\cdot\boldsymbol{n}_F$, $\{\boldsymbol{p}_I\}_{I=1}^{2n_{k-2}^2}$ is a vector polynomial basis for $\left[\mathbb{P}_{k-2}^2(F)\right]^2$ built as in (2) and $\{\mathbb{p}_\alpha\}_{\alpha=1}^{n_{k-2}^2}$ is a scalar polynomial basis for $\mathbb{P}_{k-2}^2(F)$.
- **Internal $\bigoplus$ DOFs**: the moments of $\boldsymbol{v}$

$$\int_E \boldsymbol{v}\cdot\boldsymbol{g}_\alpha^{\perp,k-2} \qquad \forall\alpha = 1, \dots, n_{k-2}^\perp. \tag{46}$$

- **Internal** div **DOFs**: the moments up to order $k-1$ and greater than zero of $\operatorname{div}\boldsymbol{v}$ in E, i.e.

$$\int_E \operatorname{div}\boldsymbol{v}\ \mathbb{p} \qquad \forall \mathbb{p} \in \mathcal{P}_{k-1}^d(E) \setminus \mathbb{R}. \tag{47}$$

It holds

$$N_E^{\text{dof}} := \dim \boldsymbol{V}_{h,k}^d(E) = \begin{cases} 2(N_E^v + N_E^e(k-1)) + n_{k-2}^\perp + n_{k-1}^2 - 1 & \text{if } d = 2, \\ 3\left(N_E^v + N_E^e(k-1) + N_E^f\frac{(k-1)k}{2}\right) + n_{k-2}^\perp + n_{k-1}^3 - 1 & \text{if } d = 3, \end{cases} \tag{48}$$

$$\dim Q_{h,k}^d(E) = \dim \mathbb{P}_{k-1}^d(E) = n_{k-1}^d. \tag{49}$$

Proposition 3 ([8,38]). *Let $\boldsymbol{V}_{h,k}^d(E)$ be the space defined in (42) for $d = 2$ and in (43) for $d = 3$. The local DOFs, along with the enhancement property, allow us to compute exactly (up to machine precision) the face projections*

$$\Pi_{k+1}^{0,F} : \left[\mathcal{V}_{h,k,1}^2(E)\right]^3 \to \left[\mathbb{P}_{k+1}^2(F)\right]^3, \quad \Pi_k^{\nabla,F} : \left[\mathcal{V}_{h,k,1}^2(E)\right]^3 \to \left[\mathbb{P}_k^2(F)\right]^3,$$

and the element projections

$$\Pi_k^{0,E} : \boldsymbol{V}_{h,k}^d(E) \to \left[\mathbb{P}_{k-2}^d(E)\right]^d, \quad \Pi_k^{\nabla,E} : \boldsymbol{V}_{h,k}^d(E) \to \left[\mathbb{P}_k^d(E)\right]^d,$$
$$\Pi_{k-1}^{0,E} : \nabla\boldsymbol{V}_{h,k}^d(E) \to \left[\mathbb{P}_{k-1}^d(E)\right]^{d\times d}$$

as defined in (5) and (6), i.e. we are able to compute the polynomials $\Pi_{k+1}^{0,F}\boldsymbol{v}_h$, $\Pi_k^{\nabla,F}\boldsymbol{v}_h$, $\Pi_k^{0,E}\boldsymbol{v}_h$, $\Pi_k^{\nabla,E}\boldsymbol{v}_h$ and $\Pi_{k-1}^0\nabla\boldsymbol{v}_h$, for any $\boldsymbol{v}_h \in \boldsymbol{V}_{h,k}^d(E)$, using only the information given by the local degrees of freedom of $\boldsymbol{v}_h$.

3.3.2. The discrete divergence-free formulation

We define the global virtual element spaces as

$$\boldsymbol{V}_{h,k} := \{\boldsymbol{v} \in \boldsymbol{V} : \boldsymbol{v}_{|E} \in \boldsymbol{V}_{h,k}^d(E) \ \forall E \in \mathcal{T}_h\},$$
$$Q_{h,k} := \{q \in Q : q_{|E} \in Q_{h,k}^d(E) \ \forall E \in \mathcal{T}_h\}.$$

The divergence-free virtual discretization of Stokes problem reads as: *Find $(\boldsymbol{u}_h, p_h) \in \boldsymbol{V}_{h,k} \times Q_{h,k}$ such that*

$$\begin{cases} \sum_{E\in\mathcal{T}_h}\left[a_h^E(\boldsymbol{u}_h, \boldsymbol{v}_h) + \left(\operatorname{div}\boldsymbol{v}_h, p_h\right)_E\right] = \sum_{E\in\mathcal{T}_h}\left(\boldsymbol{f}_h, \Pi_k^{0,E}\boldsymbol{v}_h\right)_E & \forall \boldsymbol{v}_h \in \boldsymbol{V}_{h,k}, \\ \sum_{E\in\mathcal{T}_h}\left(\operatorname{div}\boldsymbol{u}_h, q_h\right)_E = 0 & \forall q_h \in Q_{h,k}, \end{cases} \tag{50}$$

where we set

$$a_h^E(\boldsymbol{u}_h, \boldsymbol{v}_h) = \left(\nu\Pi_{k-1}^{0,E}\nabla\boldsymbol{u}_h, \Pi_{k-1}^{0,E}\nabla\boldsymbol{v}_h\right)_E + S^E((I - \Pi_k^{\nabla,E})\boldsymbol{u}_h, (I - \Pi_k^{\nabla,E})\boldsymbol{v}_h)$$

and

$$S^E(\cdot,\cdot)$$

is built following the construction detailed for the primal version of the method. If $(\boldsymbol{u}_h, p_h)$ are the solution of (50), it holds

$$\|\nabla\boldsymbol{u} - \nabla\boldsymbol{u}_h\|_{[L^2(E)]^{d\times d}} = O(h^k), \quad \|p - p_h\|_{L^2(\Omega)} = O(h^k).$$

We observe that this space offers the possibility to deal with more general problems, such as the Navier–Stokes equation and the Brinkman equation, being stable for both the Stokes and Darcy limit case [8,39,40].

Remark 2. Let us consider that the velocity field $\boldsymbol{u}$ should satisfy the continuous constraint $\mathrm{div}\boldsymbol{u} = f$ for a certain scalar field f. We observe that

$$\mathrm{div}\boldsymbol{V}_{h,k} = Q_{h,k}, \tag{51}$$

and that, in particular, the divergence-free discrete solution $\boldsymbol{u}_h$ satisfies [8]:

$$\mathrm{div}\boldsymbol{u}_h = \Pi^{0,E}_{k-1} f = \Pi^{0,E}_{k-1}\mathrm{div}\boldsymbol{u}. \tag{52}$$

Thus, if $f = 0$ as it happens for the Stokes problem, $\boldsymbol{u}_h$ is a "point-wise" divergence-free vector. On the other hand, employing the space detailed for the elastic problem in Section 3.1.3, the solution $\overline{\boldsymbol{u}}_h$ of the corresponding discretization satisfies the divergence-free property only in a "projected sense", i.e.

$$\Pi^{0,E}_{k-1}\mathrm{div}\overline{\boldsymbol{u}}_h = \Pi^{0,E}_{k-1} f = \Pi^{0,E}_{k-1}\mathrm{div}\boldsymbol{u}. \tag{53}$$

3.3.3. The reduced spaces

In PolyDiM, there is the possibility to deal with the reduced version of the above method, which is introduced in Da Veiga et al. [28]. In this reduced version, a large number of degrees of freedom can be automatically eliminated from the system, obtaining one pressure DOF per element and a few internal DOFs for the velocity field. Precisely, the new spaces are given by

$$\widehat{\boldsymbol{V}}^d_{h,k}(E) = \{\boldsymbol{v} \in \boldsymbol{V}^d_{h,k}(E) : \mathrm{div}\boldsymbol{v} \in \mathbb{P}^d_0(E)\}, \quad \widehat{Q}^d_{h,k}(E) = \mathbb{P}^d_0(E). \tag{54}$$

The subset given by the vertex, edge, (face) and the internal $\perp$ DOFs represents a unisolvent set of DOFs for the reduced velocity space $\widehat{\boldsymbol{V}}^d_{h,k}(E)$, thus leading to

$$\dim \widehat{\boldsymbol{V}}^d_{h,k}(E) = \dim \boldsymbol{V}^d_{h,k}(E) - (n^d_{k-1} - 1) < \dim \boldsymbol{V}^d_{h,k}(E),$$
$$\dim \widehat{Q}^d_{h,k}(E) = 1 < \dim Q^d_{h,k}(E).$$

Proposition 4 ([28]). *Let $(\boldsymbol{u}_h, p_h) \in \boldsymbol{V}_{h,k} \times Q_{h,k}$ be the solution of the discrete problem (50) and $(\widehat{\boldsymbol{u}}_h, \widehat{p}_h) \in \widehat{\boldsymbol{V}}_{h,k} \times \widehat{Q}_{h,k}$ be the solution of the related reduced discrete problem. Then*

$$\widehat{\boldsymbol{u}}_h = \boldsymbol{u}_h \qquad \widehat{p}_{h|E} = \Pi^{0,E}_0 p_h \quad \forall E \in \mathcal{T}_h. \tag{55}$$

Both the reduced and the full versions of the method are implemented in the library through classes belonging to the namespace Polydim::VEM::DF_PCC . We observe that, if $d = 3$, these classes use local spaces defined in Polydim::VEM::PCC to evaluate boundary contributions.

4. The implementation philosophy

Our implementation approach adheres to the standard Galerkin methodology: we design VEM classes that operate element-wise and then we glue together elemental contributions to assemble the global system associated with the PDE we aim to solve, as in a standard FEM solver.

Nonetheless, as in a standard polytopal element method, these VEM classes exploit element geometric properties, but operate independently of the values of these properties. Indeed, no element classification is performed based on element geometric properties, such as the number of vertices and edges (and faces), convexity or the presence of aligned edges and faces.

4.1. The polytope geometry

Polygons are represented in PolyDiM as an ordered list of vertices coordinates. Specifically, for each polygon, we require that the list of its vertices is ordered counterclockwise, by implicitly defining a connectivity matrix for them. The order of edges is induced by the order of the vertices, as illustrated in Fig. 2a. For a polyhedron, instead, the required data include the list of vertices, along with the edges and faces connectivity matrices. The ordering of vertices within these connectivity matrices can be arbitrary. It is important to note that only the vertex ordering in the two-dimensional rotated face matters, since a polyhedron face also represents a polygon. Consequently, for each three-dimensional face F, a map $\mathcal{M}_F$ is needed to project the face onto the two-dimensional $(x_1, x_2, 0)$-plane. After applying this transformation, the rotated face is represented as a two-dimensional polygon. Thus, we require that the map ensures the vertices of the resulting polygon are ordered counterclockwise. See Fig. 2 for an example.

Moreover, as in a standard Galerkin solver, the local discrete approximations may require other geometric properties. For example, in Virtual Element we require the polytope diameter, centroid, measure and unit outward normals on each edge or face. All these geometric properties can either be supplied by the user or automatically computed using GeDiM .

4.2. Integrals computation

Since PolyDiM is designed to deal with variational problems, we need to compute integrals. Moreover, to deal with polytopal methods, the integrals must be computed over generic polytopes. Integration of a generic polytope can be handled either by sub-triangulating the polytope and then resorting to a standard reduced Gaussian quadrature formula over triangles or tetrahedra, or by exploiting quadrature rules built ad-hoc for general polytope. For the sake of generality, we adopt the standard technique and proceed by sub-triangulating the polytope. To limit the computational cost due to the high density of the induced quadrature points, we make available in GeDiM many efficient algorithms to triangulate polytopes. Clearly, as with the geometric properties, both the sub-triangles and sub-tetrahedra can also be provided by the user.

Figs. 3 illustrate an example of such polygon and polyhedron subdivisions. The sub-triangulation shown in Fig. 2 is generated using the ear clipping algorithm provided by the GeDiM library. Note that the 2D triangles must preserve a counterclockwise ordering of their vertices. In contrast, the sub-tetrahedra depicted in Fig. 2b are constructed by connecting the triangulations of the polyhedron faces to its centroid. Alternatives to the methods described above are also available in the GeDiM library for both polygons and polyhedra.

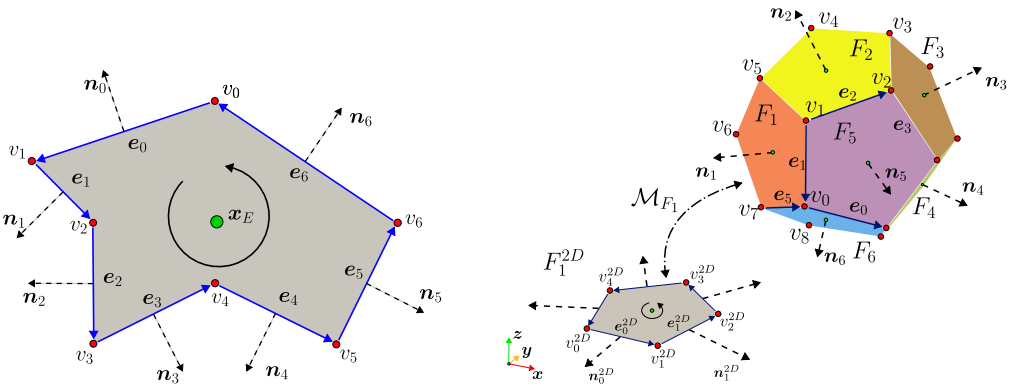
Finally, edge and face integrals must also be computed, both to enforce Neumann boundary conditions and to account for boundary contributions in the VEM projection computations. Face integrals are evaluated using aforementioned polygonal integration techniques, whereas, for edge integrals, both Gauss and Gauss–Lobatto quadrature rules are available.

4.3. Assembling

The assembly workflow for all the implemented discretization methods in PolyDiM is based on the *local-to-global* scheme typical of the Finite Element Method [6]. Thus, the assembly process of a generic problem is divided into two main steps:

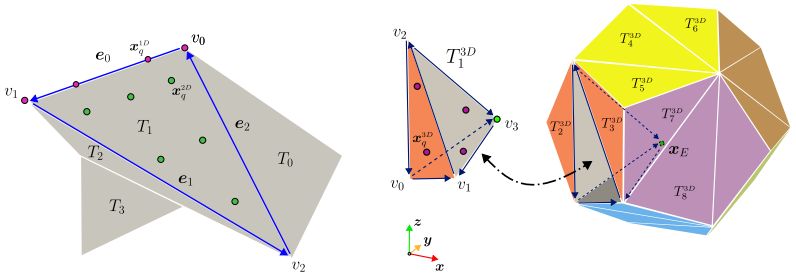
1. Compute local matrices that represent each bilinear or linear operators dictated by the PDE.
2. Assemble these local matrices into the global one.

Specifically, all the implemented local spaces, including the ones detailed in Sections 3.1, 3.2 and 3.3, are designed to evaluate the local (projected for the VEM) basis functions $\{\varphi_i\}_{i=1}^{N^{\mathrm{dof}}_E}$ and their derivatives at points belonging to the element E. These values are then used to compute local matrices. For instance, the VEM elemental stiffness matrix $\mathbf{A} \in \mathbb{R}^{N^{\mathrm{dof}}_E \times N^{\mathrm{dof}}_E}$ related to the primal formulation presented in Section 3.1.2 can be computed as:



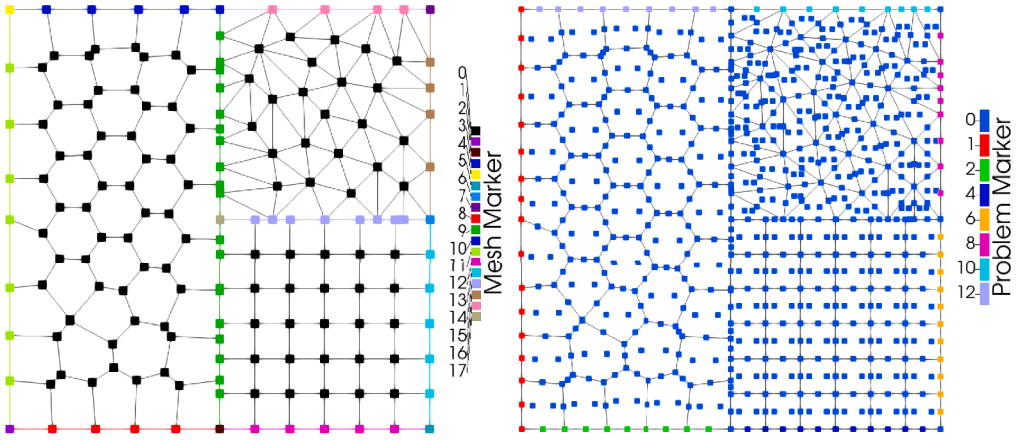
(a) $d = 2$. (b) $d = 3$.

Fig. 2. Representation of polygons and polyhedrons.



(a) $d = 2$. (b) $d = 3$.

Fig. 3. Sub-triangulation for a polygon (Left) and a polyhedron (right). green dots represent quadrature points related to a rule of order 4 over a triangle, whereas red dots represent points corresponding to a quadrature formula of order 2. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)



(a) (b)

Fig. 4. Left: Mesh cells coloured by mesh markers. Right: DOF cells coloured by problem markers.

```
std::vector<Eigen::MatrixXd> vandermonde_derivatives = ComputeBasisFunctionsDerivativeValues(
    local_space_data,
    Polydim::VEM::PCC::ProjectionTypes::Pi0km1Der
);
```

Fig. 5. Vandermonde matrices containing the projection of basis function derivatives evaluated at quadrature points.

```
Eigen::MatrixXd S = ComputeDofiDofiStabilizationMatrix(
    local_space_data,
    Polydim::VEM::PCC::ProjectionTypes::PiNabla
);
```

Fig. 6. Dofi-Dofi VEM stabilization matrix.

$$\mathbf{A}^E_{ij} = \int_E \boldsymbol{\kappa}\Pi^{0,E}_{k-1}\nabla\varphi_j \cdot \Pi^{0,E}_{k-1}\varphi_i + S^E((I-\Pi^{\nabla,E}_k)\varphi_i,(I-\Pi^{\nabla,E}_k)\varphi_j)$$
$$\approx \sum_{q=1}^{N^Q_E} \omega_q \boldsymbol{\kappa}(\boldsymbol{x}_q)\Pi^{0,E}_{k-1}\nabla\varphi_j(\boldsymbol{x}_q)\cdot\Pi^{0,E}_{k-1}\varphi_i(\boldsymbol{x}_q) + \mathbf{S}^E_{ij} \quad \forall i,j = 1,\dots,N^{\mathrm{dof}}_E,$$

where $\{(\omega_q,\boldsymbol{x}_q)\}_{q=1}^{N^Q_E}$ is a quadrature formula over E with N^Q_E points. In this case, the class method of Fig. 5 returns the projection values of the gradients of virtual basis functions at the quadrature points of the element E stored in Vandermonde matrices, whereas the Fig. 6 allows the user to retrieve the dofi-dofi stabilization matrix $\mathbf{S}^E$, defined in Eq. (19).

Concerning the second step, the global assembly phase inserts the entries of the local matrices into the global matrix associated with the discrete problem. This process is performed through a map that associates each local degree of freedom of the local spaces to the corresponding global degree of freedom. An example of the implementation of this *local-to-global* map is contained inside the library class Polydim::PDETools::DOFs::DOFsManager and it is used in all the examples contained in the library.

We observe that, based on the formulation employed, one or more instances of DOFsManager can be used. For instance, to solve the mixed differential problem (26) we employ two instances of DOFsManager. The first counts the degrees of freedom associated with the velocity variable, while the second accounts for pressure DOFs. Similarly, to deal with the elasticity problem (23) we employ again two objects of type DOFsManager . In this case, each is associated with a different component of the discrete (vector)-solution.

The local-to-global map is constructed based on the local discrete space information contained within the ReferenceElement classes. Since the VEM can handle meshes composed of arbitrarily shaped polygons and polyhedra, the concept of reference element in this context differs from the geometric interpretation typically used in FEM. Indeed, this ReferenceElement serves as a container for local information that is common across all elements in the tessellation: the number of degrees of freedom associated with vertices (called DOF cells 0D), edges (DOF cells 1D), faces (DOF cells 2D) if $d = 3$, and internal to the element E (DOF cells dD), the order of quadrature rules employed to compute projections and the information about objects related to the polynomial basis used to define both DOFs and projections.

Moreover, this map helps to store information about the test problem and, in particular, about the type of boundary conditions imposed on each considered elemental DOF cell. Indeed, to impose different conditions dictated by the problem (e.g. boundary and interface conditions) on different parts of the domain, each geometric cell of the mesh $\mathcal{T}_h$ is marked by a number. Fig. 4a shows an example of these *mesh markers* in the case of a domain divided into three sub-domains according to problem coefficients values: marker 9, 13, and 17 denote geometric cells belonging to the sub-domains interfaces, whereas all the other markers denotes different parts of the boundary domains. The only exception in this example is represented by the marker 0 that labels mesh cells that does not correspond to the particular condition. Particularly, all the dD geometric cells are always marked by 0.

Given a set of mesh markers, the test problem associates with each specific mesh marker a *problem marker*, which provides information about the boundary condition that must be applied to the DOF cell related to this geometric cell. Given this information, the DOFsManager tells us if a local DOF cell corresponds to

- a *strong* cell, i.e. an essential condition which is imposed strongly in the discrete formulation [6]. This is the case, for instance, of cells belonging to the Dirichlet boundary for the primal version of the method or to the Neumann boundary for the mixed formulation.
- a *weak* cell, i.e. a natural condition which is imposed weakly in the discrete formulation [6]. The weak cells correspond to DOFs (unknown) of the discrete problem. This is the case, for instance, of cells belonging to the Neumann boundary for the primal version of the method or to the Dirichlet boundary for the velocity variable in the mixed formulation.
- a cell with where no boundary condition is imposed that correspond to DOFs for the discrete problem. This can be the case of internal cells or $(d-2)$D cells in between two weak cells.

An example of this process is shown in Fig. 4b for the DOF cells related to the virtual divergence-free velocity space (42) for $k = 2$. This figure illustrates how the DOFsManager associates with each DOF cell a problem marker. In this example, the problem marker 0 denotes DOFs for which no boundary conditions are imposed. In contrast, odd problem markers greater than 0 correspond to strong DOF cells, whereas even problem markers greater than 0 correspond to weak DOF cells.

4.4. Solver

After the assembly phase, to find the discrete solution, i.e. the vector of DOFs, we must solve the global system of equations, that can be a linear or non-linear systems of equations. For this purpose, we offer through GeDiM an interface with the most common linear algebra C++ libraries, including `Eigen` [41], and `PETSc` [42].

All solver interfaces are designed to operate with the generic classes Gedim::ISparseArray and Gedim::IArray , which represent the sparse matrices and vectors of the linear system, respectively. Concrete implementations of these abstract classes are already available for both `Eigen` and the serial version of `PETSc` within the GeDiM library. This level of abstraction in the linear algebra layer, if used by the user, enables to switch between different solvers with ease and without altering the core components of the code, such as the assembly process.

5. Usage and examples

The numerical solution of partial differential equations on geometrically complex domains represents a fundamental task in scientific computing, owing to its central role in accurately modeling a broad spectrum of real-world phenomena, but it can be extremely challenging. Polytopal methods like the Virtual Element Method alleviate these meshing issues by allowing, for instance, Attene et al. [43]

- to achieve great flexibility in the treatment of complex geometries;
- automatic handling of hanging nodes;
- simplify refinement and agglomeration strategies.

In this section, we provide a brief overview of how to use PolyDiM and present a selection of numerical experiments that demonstrate the flexibility and robustness of our library in handling complex geometries. Most of the examples are drawn from the authors previous research, where PolyDiM has been effectively employed to solve challenging problems using advanced polytopal methods. We briefly summarize these applications to highlight the capabilities of the library, while referring the reader to the original papers for comprehensive details. Additionally, we include examples from external studies where PolyDiM could be successfully applied, further illustrating its versatility and broad applicability across various scientific computing contexts.

5.1. How to use PolyDiM

Our library is capable of handling and solving a wide range of PDEs, including both scalar and vector problems, as well as linear and non-linear ones. To define a test problem, several types of input data must be provided:

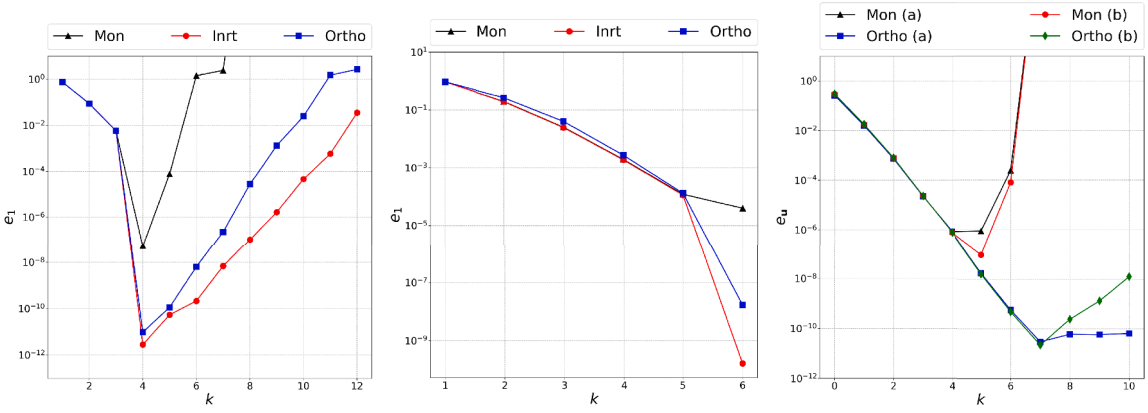
- *The PDE domain tessellation*. The mesh can either be supplied by the user or generated through GeDiM. GeDiM supports multiple mesh input file format (e.g. OVM, OFF, VTK) and interfaces with popular mesh generators as Triangle [44], TetGen [45], and Voro++ [46]. Additionally, GeDiM offers a wide range of custom mesh generators

```
struct I_Test
{
    virtual Polydim::PDETools::Mesh::PDE_Mesh_Utilities::PDE_Domain_2D domain() const = 0;
    virtual std::map<unsigned int, Polydim::PDETools::DOFs::DOFsManager::MeshDOFsInfo::BoundaryInfo>
        boundary_info() const = 0;
    virtual Eigen::VectorXd diffusion_term(const Eigen::MatrixXd &points) const = 0;
    virtual std::array<Eigen::VectorXd, 3> advection_term(const Eigen::MatrixXd &points) const = 0;
    virtual Eigen::VectorXd source_term(const Eigen::MatrixXd &points) const = 0;
    virtual Eigen::VectorXd strong_boundary_condition(const unsigned int marker, const Eigen::MatrixXd &points)
        const = 0;
    virtual Eigen::VectorXd weak_boundary_condition(const unsigned int marker, const Eigen::MatrixXd &points)
        const = 0;
    virtual Eigen::VectorXd exact_solution(const Eigen::MatrixXd &points) const = 0;
    virtual std::array<Eigen::VectorXd, 3> exact_derivative_solution(const Eigen::MatrixXd &points) const = 0;
};
```

Fig. 7. Test data methods needed to solve a second order elliptic problem.

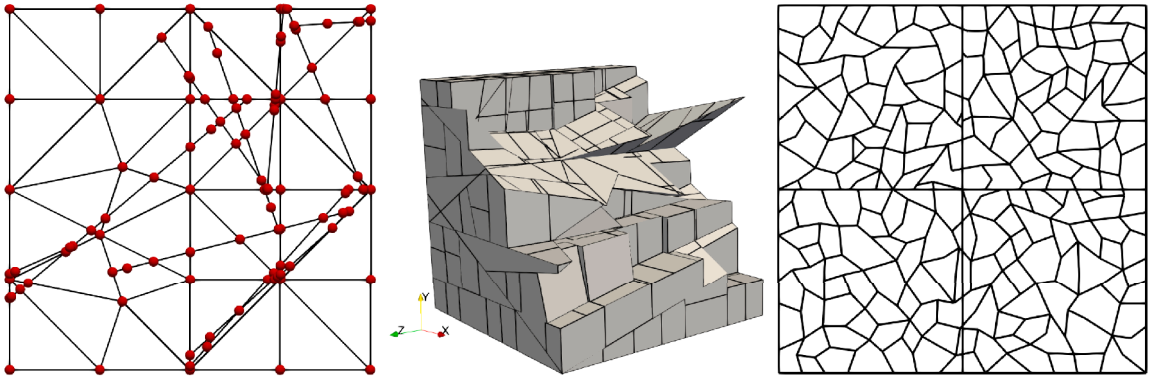
```
namespace Elliptic_PCC_2D = Polydim::examples::Elliptic_PCC_2D;
const auto test = Elliptic_PCC_2D::program_utilities::create_test(config);
const auto domain = test->domain();
const auto boundary_info = test->boundary_info();

// Create Domain Tessellation
Gedim::MeshMatrices meshData;
Gedim::MeshMatricesDAO mesh(meshData);
PCC_2D::program_utilities::create_domain_mesh(config, domain, mesh);

// Compute Geometric Properties
const auto meshGeometricData =
    Elliptic_PCC_2D::program_utilities::create_domain_mesh_geometric_properties(config, mesh);

// Create Discrete Space
const auto reference_element_data =
    Elliptic_PCC_2D::local_space::CreateReferenceElement(config.MethodType(), config.MethodOrder());
Polydim::PDETools::Mesh::MeshMatricesDAO mesh_connectivity_data mesh_connectivity_data = {mesh};
const auto reference_element_num_dofs =
    Elliptic_PCC_2D::local_space::ReferenceElementNumDOFs(reference_element_data);

// Define the DOFs manager
Polydim::PDETools::DOFs::DOFsManager dofManager;
const auto meshDOFsInfo =
    dofManager.Create_Constant_DOFsInfo_2D(mesh_connectivity_data, {reference_element_num_dofs,
        boundary_info});
const auto dofs_data = dofManager.CreateDOFs_2D(meshDOFsInfo, mesh_connectivity_data);

// Assemble and Solve
Elliptic_PCC_2D::Assembler assembler;
auto assembler_data =
    assembler.Assemble(config, mesh, meshGeometricData, meshDOFsInfo, dofs_data, reference_element_data, *test);
Gedim::Eigen_LUSolver solver;
solver.Initialize(assembler_data.globalMatrixA);
solver.Solve(assembler_data.rightHandSide, assembler_data.solution);

// Post process and export solution
auto post_process_data =
    assembler.PostProcessSolution(config, mesh, meshGeometricData, dofs_data, reference_element_data,
        assembler_data, *test);
Elliptic_PCC_2D::program_utilities::export_solution(config, mesh, dofs_data, assembler_data, post_process_data,
    exportSolutionFolder, exportVtuFolder);
Elliptic_PCC_2D::program_utilities::export_dofs(config, mesh, meshGeometricData,
    meshDOFsInfo, dofs_data, assembler_data, post_process_data, exportVtuFolder);

// Compute Method Performance
const auto performance = assembler.ComputePerformance(config, mesh, meshGeometricData, reference_element_data);
Elliptic_PCC_2D::program_utilities::export_performance(config, performance, exportSolutionFolder);
```

Fig. 8. Example of main to solve a two-dimensional elliptic problem by exploiting the PCC classes.



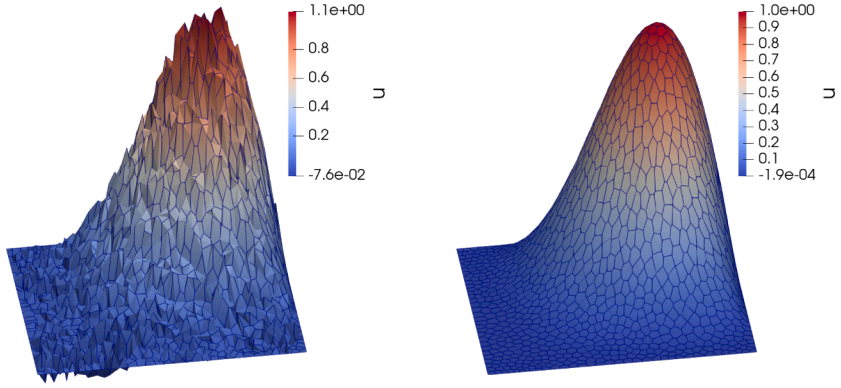
(a) Test 1 in [12]. (b) Test 3 in [12]. (c) Test 1 in [37].

Fig. 9. Left: H^1-error e_1 for the primal version of the method as the method order k varies in two-dimensions for the different approaches proposed in Section 3.1. Center: H^1-error for the primal version of the method as k varies in three-dimensions for the different approaches proposed in Section 3.1. Right: L^2-error for the velocity variable in the mixed version of the method as k varies in two-dimensions for different approaches proposed in Section 3.2.



(a) Test 1 in [12]. (b) Test 3 in [12]. (c) Test 1 in [37].

Fig. 10. Meshes characterized by concave elements and a copious number of hanging nodes taken from Discrete Matrix and Fractures applications.



(a) (b)

Fig. 11. A SUPG-stabilized VEM solution related to advection-dominated problem. Test described in Benedetto et al. [48].

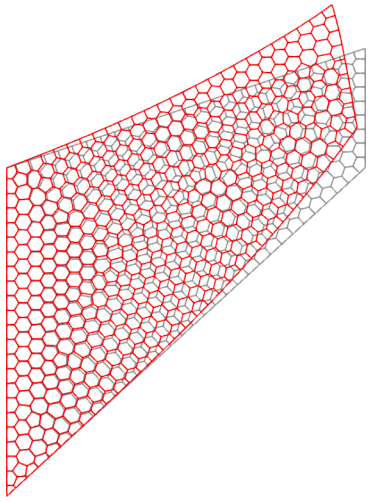


Fig. 12. Test 2: Cook's membrane. Undeformed (gray) and deformed (red) configurations. $k = 1$. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

that are commonly used in numerical experiments involving virtual element methods.

- *Problem coefficients*, such as diffusion, advection, reaction, viscosity coefficients, and source terms. These data are defined through methods in a test class that extends the C++ interface corresponding to the specific differential problem. Fig. 7 reports the I_Test interface for the case of a scalar elliptic problem. Each method in this interface takes as input a matrix points, where each column represents a three-dimensional point, and returns as an array the corresponding function evaluations in these points.
- *Boundary conditions*, both essential and natural, along with a standard map std::map < unsigned int, Polydim::PDETools::DOFs::DOFsManager::MeshDOFsInfo::BoundaryInfo > that associates with each mesh marker a strong or a weak boundary conditions identified by a problem marker as described in Section 4.3.

The test class may also optionally include methods to define the exact solution and its derivatives, allowing the computation of numerical errors. Indeed, the PolyDiM library provides several tools for post-processing the numerical solution, including residual evaluation, error estimation, and analysis of method performance metrics such as the conditioning of elemental matrices. In addition, PolyDiM offers VTK utilities [47] to realize graphical illustrations of quantities of interests.

An example of the program main used to solve a two-dimensional second order elliptic problems is provided in Fig. 8.

5.2. Elliptic problems and polynomial bases

PolyDiM enables the solution of general second-order elliptic problems on both two- and three-dimensional meshes, for a generic order k. Moreover, as discussed in the previous sections, PolyDiM provides methods that leverage novel polynomial bases to improve the conditioning of the discrete systems in the presence of badly-shaped polytope

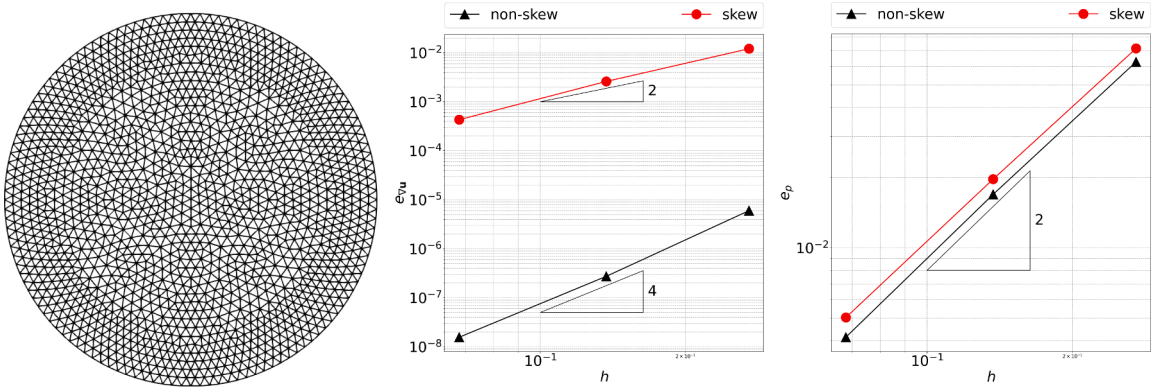


Fig. 13. Left: Domain and the finest mesh used to solve the Navier–Stokes problem. Center: Behaviour of errors related to the velocity variable as the mesh parameter h decreases for both the skew and the non-skew trilinear form. Right: Behaviour of errors related to pressure variable as the mesh as the mesh parameter h decreases for both the skew and the non-skew trilinear form $k = 2$.

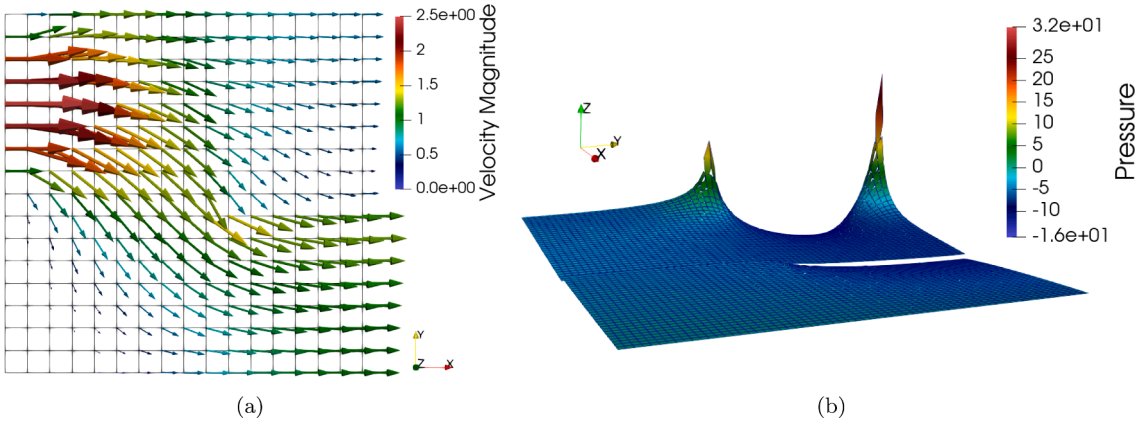


Fig. 14. Left: Velocity field computed by solving the Brinkman equation related to a coarse square mesh obtained with divergence-free formulation of virtual element for $k = 2$. Right: The pressure field obtained related to a fine square mesh obtained with divergence-free formulation of virtual element for $k = 2$.

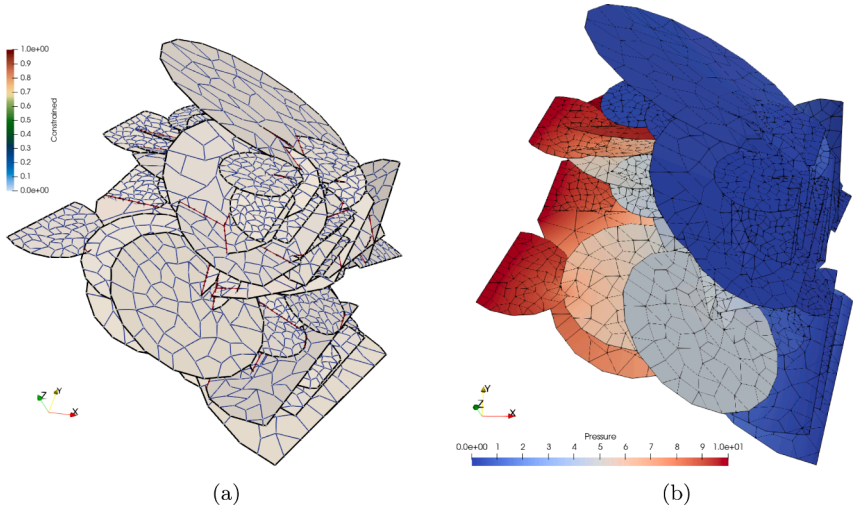


Fig. 15. Test described in Sorgente et al. [11]. Left: Mesh agglomeration in DFN. Right: DFN coloured by the numerical pressure.

and for high orders of the method. Fig. 9 illustrates the behaviour of VEM errors as the order k increases, when solving the general second-order elliptic problem on meshes shown in Fig. 10 for the different approaches detailed in Section 3.1. These meshes are characterized by concave elements and a copious number of hanging nodes, representing typical examples of meshes encountered in Discrete Matrix and Fractures applications. These examples are taken from Berrone et al. [12,13,37], where the authors have effectively used the PolyDiM library. Moreover, we recall that authors also contribute to introduce a consistent Streamline Upwind Petrov-Galerkin (SUPG) formulation for the virtual element

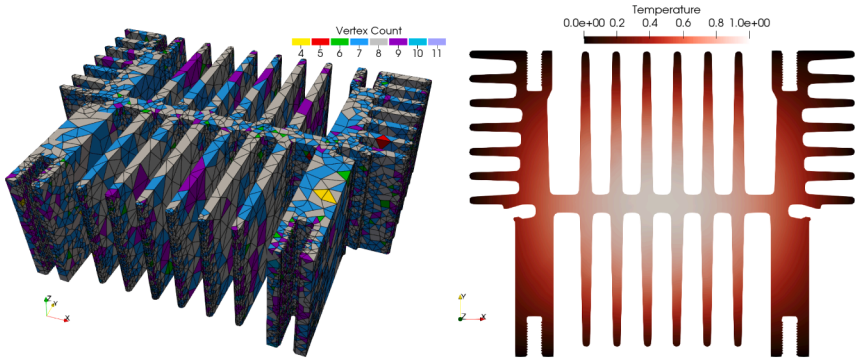


Fig. 16. Test described in Sorgente et al. [50]. Left: Mesh agglomeration performed in a mechanical component. Different colours represent polyhedra with a different number of faces. Right: Computational domain sliced and coloured by the numerical temperature.

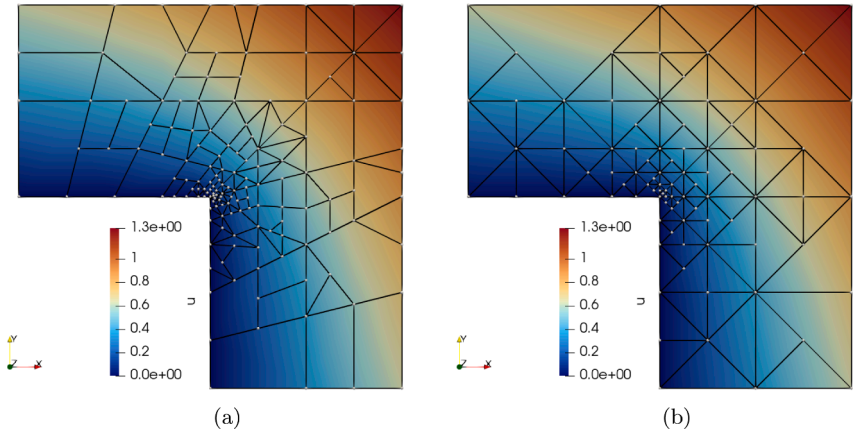


Fig. 17. Refinement of the L-shaped domain using two different polygonal meshes. The example is detailed in Berrone and Vicini [9].

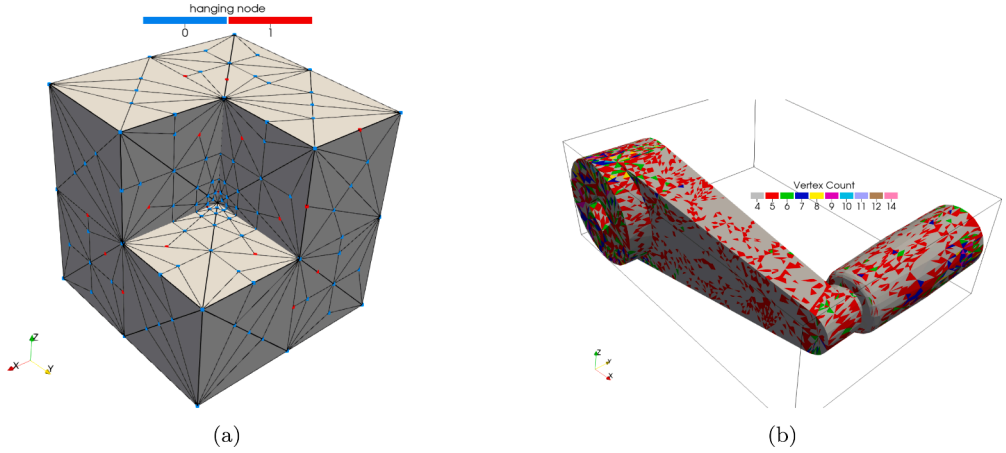


Fig. 18. These examples are detailed in Berrone et al. [51]. Left: Refinement of a three-dimensional L-shaped domain. Right: Refinement of a mechanical component. Different colours represent polyhedra with a different number of faces.

methods. Figure shows the results related to Test 1 performed in Benedetto et al. [48], where an advection-dominated problem is solved. All these examples are implemented and available to the reader on GitHub.

5.3. Elasticity, Brinkman and Navier–Stokes problems

PolyDiM supports the solution of various types of PDEs, including Brinkman, elasticity, and Navier–Stokes problems. In particular,

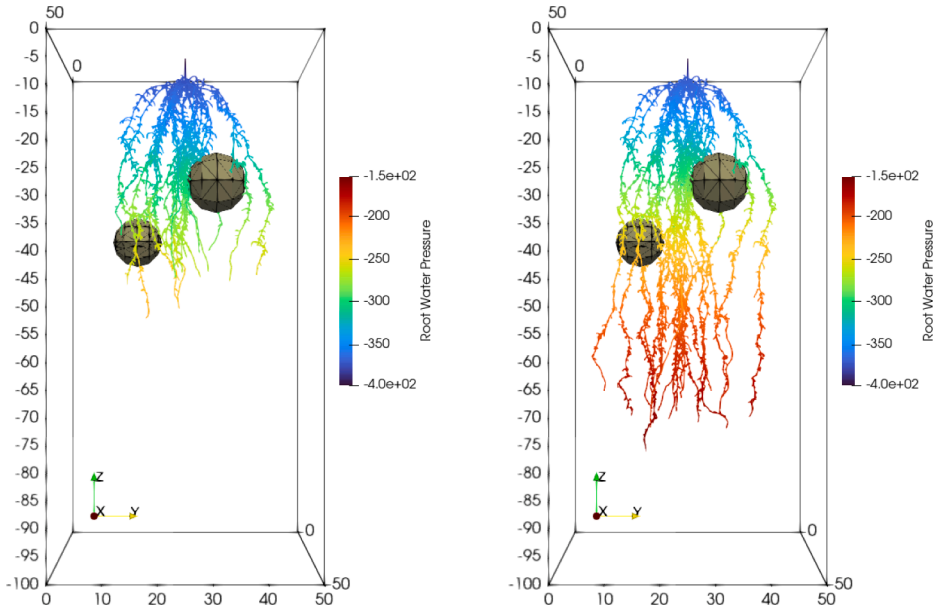


Fig. 19. The root water pressure across multiple phases of root growth. Left: After 80 days had elapsed. Right: 160 days had elapsed. This example is detailed in Berrone et al. [10].

divergence-free virtual elements can be effectively used to solve both the Navier–Stokes and Brinkman equations, offering stability in both the Darcy and Stokes limits (see Section 3.3), as well as in the incompressible limit for elasticity problems. Elasticity problems can also be addressed using standard primal formulations under compressibility assumptions, as detailed in Section 3.1.3. The library provides tools for solving all these problems in both two- and three-dimensional settings.

To showcase the capabilities of PolyDiM, we provide a representative numerical experiment for each problem type. All examples are fully implemented and publicly available on GitHub.

For the elasticity case, inspired by Test 4 in Artioli et al. [49], we solve the $2D$ Cook's membrane problem (23) with $k = 1$ on a Voronoi mesh. We highlight that standard Finite Element Methods on triangular meshes are also available in PolyDiM. Since the exact solution is not provided for this example, a reference solution can be computed through this standard method to test VEM accuracy.

Concerning fluid dynamic problems, Fig. 13 reports the VEM error corresponding to the resolution of the Navier–Stokes problem described in Test 5.2 of [49], whereas Fig. 14 illustrates the velocity and pressure fields obtained solving a Brinkman problem inspired by Test 5.6 in Artioli et al. [49]. Notably, in this latter test, the DOF cells related to the boundary degrees for the first component of the velocity are associated with the problem markers as shown in Fig. 4b. In addition, the domain subdivision highlighted in the figure allows us to set different coefficient values across subregions. This setup enables us the simultaneous resolution of a Stokes problem in the left part of the domain and a pure Darcy problem in the right part, effectively illustrating the versatility of PolyDiM in handling mixed regimes within a single simulation.

5.4. Mesh quality improvement

The capability of PolyDiM to generate numerical approximations on generic polytopes has been effectively exploited to enhance tessellation quality through mesh optimization techniques combined with agglomeration strategies [11,50]. In these manuscripts, quality-based optimization strategies to reduce the total number of degrees of freedom associated with a Virtual Element discretization of boundary problems defined over polytopal tessellations are proposed.

Figs. 15 and 16 illustrate two examples showcasing the application of these techniques to two- and three-dimensional problems, respectively. In particular, Fig. 15a shows the results of this agglomeration-based technique applied to a DFNs, composed by 86 rounded fractures and 159 interfaces, whereas Fig. 15b illustrates the hydraulic head distribution obtained by solving with PolyDiM a Darcy problem over this realistic DFN setting.

On the other hand, Fig. 16 presents the three-dimensional agglomeration-based optimization obtained starting from a tetrahedralization of a complex mechanical component and the corresponding time-dependent solution of a thermal problem defined over this domain obtained with PolyDiM. The reader may refer to Sorgente et al. [50] for detailed numerical results obtained using this 3D mesh enhancement.

5.5. Mesh refinement

It is well-known that using generic polytopal discrete spaces simplifies the refinement process, as aligned edges or faces can be managed without requiring global mesh updates to maintain conformity. The discrete spaces of PolyDiM, combined with the geometric functions provided by GeDiM, support general polygonal and polyhedral mesh refinement on complex geometries. Examples of two- and three-dimensional refinements using PolyDiM can be found in Berrone and Vicini [9], Canuto and Fassino [52] and [51], respectively. Specifically, the high order scheme for Adaptive Virtual Element Methods (AVEMs) on triangular meshes has been introduced in Canuto and Fassino [52], whereas work [9] details polygonal refinement algorithm to solve elliptic problems with VEM. Finally, the lowest order AVEM scheme on tetrahedral meshes is described by the authors in Berrone et al. [51]. Figs. 17 and 18 illustrate two cases demonstrating the refinement techniques described in those references. More precisely, Fig. 17 show the refinement of a two-dimensional L-shaped domain using two different polygonal meshes: the first mesh allows unrestricted aligned edges, while the second constrains the number of aligned edges to a maximum of two. Fig. 18, on the other hand, presents two refined 3D domains: the

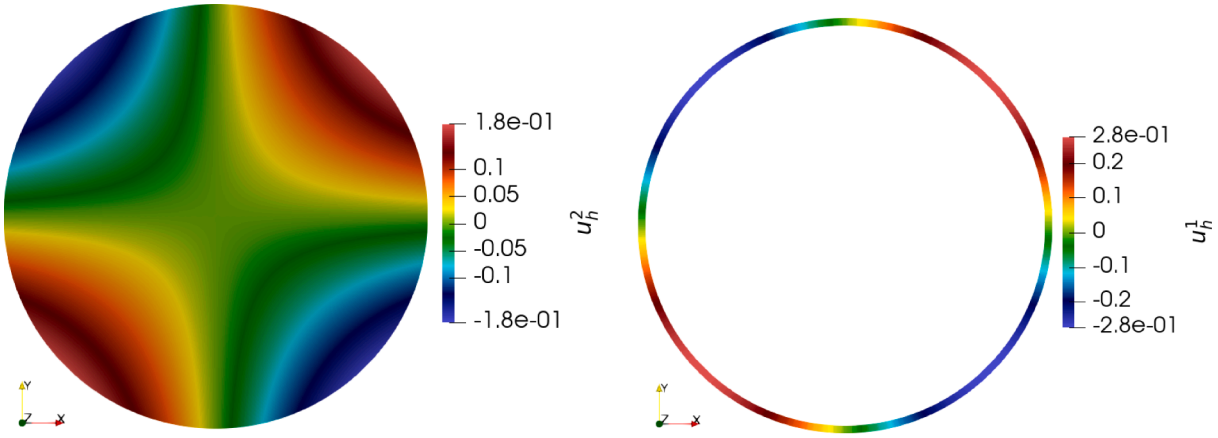


Fig. 20. Bulk and surface discrete solutions u_h^2 and u_h^1 of problem (56) over the finest mesh of disk of radius 1 centered at the origin.

first is the generalization of the 2D L-shaped problem to the third dimension, while the second is a complex geometry derived from a mechanical component.

5.6. Mixed-dimensional linear and non-linear coupled problems

PolyDiM provides a convenient framework for handling coupled systems of PDEs. Indeed, the namespace Polydim::PDETools::Assembler_Utilities together with the Polydim::PDETools::DOFs::DOFsManager offers different methods for managing multiple variables simultaneously and efficiently coupling them at interfaces.

As an example, the PolyDiM library was successfully applied by the authors in Berrone et al. [10] to solve 3D-1D coupled problems of root water uptake. In the original 3D-3D coupled problem, the soil moisture dynamics are governed by the nonlinear, time-dependent Richards equation in the soil domain, while the flow in the root xylem is described by the Brinkman equation. By reducing this problem to a well-posed 3D-1D coupled problem, the root-soil interactions are simulated on a dynamic root system architecture that develops in a stony soil. Fig. 19 illustrates the root water pressure during different stages of root growth. The flexibility of the Virtual Element Method in handling concave elements and hanging nodes is here exploited to easily mesh the soil domain around stones and effectively solve the Richards equation in the soil domain.

Other applications to mixed-dimensional problems include, but are not limited to, Discrete Fracture Networks models [13,53] and Discrete Fracture and Matrix problems [54].

In this work, we additionally demonstrate the use of PolyDiM for coupled bulk-surface problems in two spatial dimensions [55]. In particular, we solve the following parabolic coupled problem:

$$\begin{cases} \frac{\partial u^2}{\partial t} - d^2\Delta u^2 + \gamma^2 u^2 = f^2 & \boldsymbol{x} \in \Omega,\ t \in [0,T], \\ \frac{\partial u^1}{\partial t} - d^1\Delta_\Gamma u^1 + \gamma^1 u^1 + \nabla u^2 \cdot \boldsymbol{n} = f^1 & \boldsymbol{x} \in \Gamma,\ t \in [0,T], \\ \nabla u^2 \cdot \boldsymbol{n} = -\alpha u^2 + \beta u^1 & \boldsymbol{x} \in \Gamma,\ t \in [0,T], \end{cases} \tag{56}$$

where Δ_Γ is the Laplace-Beltrami operator and superscripts 2 and 1 refer to bulk and surface data, respectively. The problem data corresponds to those presented in *Experiment 2: linear parabolic problem* of the reference paper [55]. We solved the problem in PolyDiM on the same family of triangular meshes of the disk domain adopted in Section 5.3. The Finite Element Method is used for spatial discretization in both the 2D (bulk) and 1D (surface) domains, combined with an explicit Euler scheme for time integration. This example, together with an additional test case employing alternative time discretization techniques, is available in the PolyDiM repository. The bulk and surface numeric discrete counterparts of variables u^2 and u^1 obtained on the finest triangular mesh are shown in Fig. 20. Moreover, the maximum errors in time with respect to L^2-norm (e_0) and H^1-seminorm (e_∇) along with the empirical order of convergence (EOC) with respect to mesh size h are shown in Table 3. The obtained results are coherent with the theoretical estimates presented in Frittelli et al. [55].

Table 3
The maximum errors in time with respect to L^2-norm (e_0) and H^1-seminorm (e_∇) along with the empirical order of convergence (EOC) with respect to mesh size h related to the bulk-surface solution of the problem. The chosen uniform time step τ for each mesh size is also reported.

h	τ	e_0	EOC_0	e_∇	EOC_∇
2.71e−01	2.50e−03	1.89e−02	–	8.54e−02	–
1.37e−01	6.25e−04	4.85e−03	1.99e+00	4.04e−02	1.10e+00
6.79e−02	1.56e−04	1.24e−03	1.97e+00	1.99e−02	1.05e+00
3.81e−02	1.56e−04	2.68e−04	2.14e+00	9.66e−03	1.10e+00

CRediT authorship contribution statement

Stefano Berrone: Writing – review & editing, Supervision, Project administration, Methodology, Funding acquisition, Conceptualization; **Andrea Borio:** Writing – review & editing, Validation, Software, Methodology, Formal analysis, Conceptualization; **Gioana Teora:** Writing – review & editing, Writing – original draft, Validation, Software, Resources, Project administration, Methodology, Investigation, Formal analysis, Data curation, Conceptualization; **Fabio Vicini:** Writing – review & editing, Writing – original draft, Validation, Software, Resources, Project administration, Methodology, Investigation, Formal analysis, Data curation, Conceptualization.

Data availability

The software library presented in this article is available for download from the official POLYDIM website at www.polydim.it. The package includes source code, documentation, and test datasets.

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.

Acknowledgement

The author S.B. and A.B. kindly acknowledges 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 G.T. kindly acknowledges the financial support provided by the PRIN project 20227K44ME "Full and Reduced order modelling of coupled systems: focus on non-matching methods and automatic learning (FaReX)" (CUP:E53D23005510006).

References

[1] L.B. Da Veiga, F. Brezzi, L.D. Marini, A. Russo, The virtual element method, Acta Numer. 32 (2023) 123–202. https://doi.org/10.1017/S0962492922000095
[2] P. Antonietti, L.B. Da Veiga, G. Manzini, The virtual element method and its applications, in: SEMA SIMAI Springer Series, Springer International Publishing, 2022. https://doi.org/10.1007/978-3-030-95319-5
[3] D.A.D. Pietro, A. Ern, A hybrid high-order locking-free method for linear elasticity on general meshes, Comput. Methods Appl. Mech. Eng. 283 (2015) 1–21. https://doi.org/10.1016/j.cma.2014.09.009
[4] A. Cangiani, E.H. Georgoulis, P. Houston, Hp-version discontinuous Galerkin methods on polygonal and polyhedral meshes. Math. Models Methods Appl. Sci. 24 (2014) 2009–2041. https://doi.org/10.1142/S0218202514500146
[5] M. Larson, F. Bengzon, The Finite Element Method: Theory, Implementation, and Applications, Texts in Computational Science and Engineering, Berlin Heidelberg, Springer, 2013. https://doi.org/10.1007/978-3-642-33287-6
[6] S.C. Brenner, L.R. Scott, Texts in Applied Mathematics, 15, Springer, 2008. https://doi.org/10.1007/978-0-387-75934-0
[7] C. Chen, X. Huang, H. Wei, H(m)-conforming virtual elements in arbitrary dimension, SIAM J. Numer. Anal. 60 (6) (2022) 3099–3123. https://doi.org/10.1137/21M1440323
[8] G. Vacca, An H1-conforming virtual element for Darcy and Brinkman equations, Math. Models Methods Appl. Sci. 28 (01) (2018) 159–194. https://doi.org/10.1142/S0218202518500057
[9] S. Berrone, F. Vicini, Effective polygonal mesh generation and refinement for VEM, Math. Comput. Simul. 231 (2025) 239–258. https://doi.org/10.1016/j.matcom.2024.12.007
[10] S. Berrone, S. Ferraris, D. Grappein, G. Teora, F. Vicini, A 3D-1D virtual element method for modeling root water uptake, 2024. https://doi.org/10.48550/arXiv.2412.12884
[11] T. Sorgente, F. Vicini, S. Berrone, S. Biasotti, G. Manzini, M. Spagnuolo, Mesh quality agglomeration algorithm for the virtual element method applied to discrete fracture networks, Calcolo 60 (2) (2023). https://doi.org/10.1007/s10092-023-00517-5
[12] S. Berrone, G. Teora, F. Vicini, Improving high-order VEM stability on badly-shaped elements, Math. Comput. Simul. 216 (2024) 367–385. https://doi.org/10.1016/j.matcom.2023.10.003
[13] S. Berrone, S. Scialò, G. Teora, Orthogonal polynomial bases in the mixed virtual element method, Numer. Methods Partial Differ. Equ. 40 (6) (2024) 23144. https://doi.org/10.1002/num.23144
[14] T. Sorgente, S. Biasotti, G. Manzini, M. Spagnuolo, The Role of Mesh Quality and Mesh Quality Indicators in the Virtual Element Method, 48, 2022. https://doi.org/10.1007/s10444-021-09913-3
[15] T. Sorgente, S. Biasotti, G. Manzini, M. Spagnuolo, Polyhedral mesh quality indicator for the virtual element method, Comput. Math. Appl. 114 (2022) 151–160. https://doi.org/10.1016/j.camwa.2022.03.042
[16] S. Berrone, A. Borio, G. Teora, F. Vicini, Gedim, GEometry for DIscretization MEthod library, 2025. https://doi.org/10.5281/zenodo.15146658
[17] M. Frittelli, A. Madzvamuse, I. Sgura, VEMcomp: a virtual elements MATLAB package for bulk-surface PDEs in 2D and 3D, Numer. Algorithms, 2024. https://doi.org/10.1007/s11075-024-01919-4
[18] Y. Yu, mVEM: a MATLAB software package for the virtual element methods, 2022. https://doi.org/10.48550/arXiv.2204.01339
[19] A. Ortiz-Bernardin, VEMLAB: a MATLAB library for the virtual element method, 2025 https://camlab.cl/vemlab/.
[20] A. Ortiz-Bernardin, C. Alvarez, N. Hitschfeld-Kahler, A. Russo, R. Silva-Valenzuela, E. Olate-Sanzana, Veamy: an extensible object-oriented C library for the virtual element method, Numer. Algorithms 82 (2019) 1189–1220. https://doi.org/10.1007/s11075-018-00651-0
[21] A. Dedner, A. Hodson, A framework for implementing general virtual element spaces, SIAM J. Sci. Comput. 46 (3) (2024) 229–B253. https://doi.org/10.1137/23M1573653
[22] L.B. Da Veiga, F. Dassi, A. Russo, High-order virtual element method on polyhedral meshes, Comput. Math. Appl. sI: SDS2016 - Methods for PDEs 74 (2017) 1110–1122. https://doi.org/10.1016/j.camwa.2017.03.021
[23] L. Mascotto, Ill-conditioning in the virtual element method: stabilizations and bases, Numer. Methods Partial Differ. Equ. 34 (4) (2018) 1258–1281. https://doi.org/10.1002/num.22257
[24] F. Dassi, L. Mascotto, Exploring high-order three dimensional virtual elements: bases and stabilizations, Comput. Math. Appl. 75 (9) (2018) 3379–3401. https://doi.org/10.1016/j.camwa.2018.02.005
[25] S. Berrone, A. Borio, Orthogonal polynomials in badly shaped polygonal elements for the virtual element method, Finite Elem. Anal. Des. 129 (2017) 14–31. https://doi.org/10.1016/j.finel.2017.01.006
[26] M. Cicuttin, An implementation detail about the scaling of monomial bases in polytopal finite element methods, Appl. Math. Lett. 159 (2025) 109281. https://doi.org/10.1016/j.aml.2024.109281
[27] F. Dassi, S. Scacchi, Parallel solvers for virtual element discretizations of elliptic equations in mixed form, Comput. Math. Appl. 79 (7) (2020) 1972–1989. Computational methods for PDEs. https://doi.org/10.1016/j.camwa.2019.07.027
[28] L.B. da Veiga, C. Lovadina, G. Vacca, Divergence free virtual elements for the Stokes problem on polygonal meshes, ESAIM: M2AN, 51, 2017. https://doi.org/10.1051/m2an/2016032
[29] L.B. Da Veiga, F. Brezzi, L.D. Marini, A. Russo, Virtual element method for general second order elliptic problems on polygonal meshes, Math. Models Methods Appl. Sci. 26 (04) (2016) 729–750. https://doi.org/10.1142/S0218202516500160
[30] L.B. Da Veiga, F. Brezzi, A. Cangiani, G. Manzini, A. Russo, Basic principles of virtual element methods, Math. Models Methods Appl. Sci. 23 (01) (2013) 199–214. https://doi.org/10.1142/S0218202512500492
[31] B. Ahmad, A. Alsaedi, F. Brezzi, L. Marini, A. Russo, Equivalent projectors for virtual element methods, Comput. Math. Appl. 66 (3) (2013) 376–391. https://doi.org/10.1016/j.camwa.2013.05.015
[32] L.B. Da Veiga, F. Brezzi, L.D. Marini, A. Russo, Virtual Element Implementation for General Elliptic Equations, Springer International Publishing, 2016. https://doi.org/10.1007/978-3-319-41640-3_2
[33] L.B. Da Veiga, C. Lovadina, A. Russo, Stability analysis for the virtual element method, Math. Models Methods Appl. Sci. 27 (13) (2017) 2557–2594. https://doi.org/10.1142/S021820251750052X
[34] L.B. Da Veiga, F. Brezzi, L.D. Marini, Virtual elements for linear elasticity problems, SIAM J. Numer. Anal. 51 (2) (2013) 794–812. https://doi.org/10.1137/120874746
[35] L.B. Da Veiga, F. Brezzi, L.D. Marini, A. Russo, Mixed virtual element methods for general second order elliptic problems on polygonal meshes, ESAIM: M2AN 50 (2016) 727–747. https://doi.org/10.1051/m2an/2015067
[36] L.B. Da Veiga, F. Brezzi, L.D. Marini, A. Russo, H(div) and H(curl)-conforming VEM, Numer. Math. 133 (2016) 303–332. https://doi.org/10.1007/s00211-015-0746-1
[37] S. Berrone, S. Scialò, G. Teora, The mixed virtual element discretization for highly-anisotropic problems: the role of the boundary degrees of freedom, Math. Eng. 5 (6) (2023) 1–32. https://doi.org/10.3934/mine.2023099
[38] L.B. Da Veiga, F. Dassi, G. Vacca, The Stokes complex for virtual elements in three dimensions, Math. Models Methods Appl. Sci. 30 (03) (2020) 477–512. https://doi.org/10.1142/S0218202520500128
[39] L.B. Da Veiga, C. Lovadina, G. Vacca, Virtual elements for the Navier–Stokes problem on polygonal meshes, SIAM J. Numer. Anal. 56 (3) (2018) 1210–1242. https://doi.org/10.1137/17M1132811
[40] L.B. Da Veiga, D. Mora, G. Vacca, The Stokes complex for virtual elements with application to Navier–Stokes flows, J. Sci. Comput. 81 (2019). https://doi.org/10.1007/s10915-019-01049-3
[41] G. Guennebaud, B. Jacob, et al, 2010. http://eigen.tuxfamily.org.
[42] S. Balay, S. Abhyankar, M.F. Adams, S. Benson, J. Brown, P. Brune, K. Buschelman, E.M. Constantinescu, L. Dalcin, A. Dener, V. Eijkhout, J. Faibussowitsch, W.D. Gropp, V. Hapla, T. Isaac, P. Jolivet, D. Karpeev, D. Kaushik, M.G. Knepley, F. Kong, S. Kruger, D.A. May, L.C. Mcinnes, R.T. Mills, L. Mitchell, T. Munson, J.E. Roman, K. Rupp, P. Sanan, J. Sarich, B.F. Smith, S. Zampini, H. Zhang, H. Zhang, J. Zhang, 2025. PETSc Web page, https://petsc.org/.
[43] M. Attene, S. Biasotti, S. Bertoluzza, D. Cabiddu, M. Livesu, G. Patanè, M. Pennacchio, D. Prada, M. Spagnuolo, Benchmarking the geometrical robustness of a virtual element poisson solver, Math. Comput. Simul. 190 (2021) 1392–1414. https://doi.org/10.1016/j.matcom.2021.07.018
[44] J.R. Shewchuk, Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator, 1148, Springer-Verlag, 1996. Applied Computational Geometry: Towards Geometric Engineering.
[45] H. Si, TetGen, a Delaunay-based quality tetrahedral mesh generator, ACM Trans. Math. Softw. 41 (2) (2015). https://doi.org/10.1145/2629697
[46] C.H. Rycroft, VORO: a three-dimensional Voronoi cell library in C, Chaos 19 (4) (2009) 41111. https://doi.org/10.1063/1.3215722
[47] W. Schroeder, K. Martin, B. Lorensen, The Visualization Toolkit, Kitware, 2006. Th ed.
[48] M. Benedetto, S. Berrone, A. Borio, S. Pieraccini, S. Scialò, Order preserving SUPG stabilization for the virtual element formulation of advection-diffusion problems, Comput. Methods Appl. Mech. Eng. 311 (2016) 18–40. https://doi.org/10.1016/j.cma.2016.07.043
[49] E. Artioli, L.B. Da Veiga, C. Lovadina, E. Sacco, Arbitrary order 2D virtual elements for polygonal meshes: part I, elastic problem, Comput. Mech. 60 (2017) 355–377. https://doi.org/10.1007/s00466-017-1404-5
[50] T. Sorgente, F. Vicini, S. Berrone, S. Biasotti, G. Manzini, M. Spagnuolo, Mesh optimization for the virtual element method: how small can an agglomerated mesh become?, J. Comput. Phys. 521 (2025) 113552. https://doi.org/10.1016/j.jcp.2024.113552

[51] S. Berrone, D. Fassino, F. Vicini, 3D adaptive vem with stabilization-free a posteriori error bounds, J. Sci. Comput. 103 (2025) 35. https://doi.org/10.1007/s10915-025-02852-x

[52] C. Canuto, D. Fassino, Higher-order adaptive virtual element methods with contraction properties, Math. Eng. 5 (6) (2023) 1–33. https://doi.org/10.3934/mine.2023101

[53] S. Berrone, M. Busetto, F. Vicini, Virtual element simulation of two-phase flow of immiscible fluids in discrete fracture networks, J. Comput. Phys. 473 (2023) 111735. https://doi.org/10.1016/j.jcp.2022.111735

[54] S. Berrone, A. Borio, A. D'auria, S. Scialò, F. Vicini, A robust VEM-based approach for flow simulations in poro-fractured media, Math. Models Methods Appl. Sci. 31 (14) (2021) 2855–2885. https://doi.org/10.1142/S0218202521500639

[55] M. Frittelli, A. Madzvamuse, I. Sgura, Bulk-surface virtual element method for systems of PDEs in two-space dimensions, Numer. Math. 147 (2) (2021) 305–348. https://doi.org/10.1007/s00211-020-01167-3