mVEM: A MATLAB Software Package for the Virtual Element Methods

Abstract

This paper summarizes the development of mVEM, a MATLAB software package containing efficient and easy-following codes for various virtual element methods (VEMs) published in the literature. We explain in detail the numerical implementation of the mixed VEMs for the Darcy problem and the three-dimensional linear VEMs for the reaction-diffusion problems. For other model problems, we present the construction of the discrete methods and only provide the implementation of the elliptic projection matrices. Some mesh related functions are also given in the package, including the mesh generation and refinement in two or three dimensions. mVEM is free and open source software.

Keywords. Virtual element method, Polygonal meshes, Three dimensions, MATLAB

1 Introduction

Developing the mimetic finite difference methods, Beirão and Brezzi et al. proposed the virtual element method (VEM) in 2013, and established an abstract framework for error analysis [7]. This method was further studied in [2] for an extension to reaction-diffusion problems, where a crucial enhancement technique is introduced to construct a computable L^2 projection. The computer implementation has been further studied in [11]. The proposed finite dimensional space in [7] has become the standard space for constructing conforming virtual element methods for second-order elliptic problems on polygonal meshes. The word *virtual* comes from the fact that no explicit knowledge of the basis functions is necessary since the shape functions are piecewise continuous polynomials on the boundary of the element and are extended to the interior by assuming basis functions as solutions of local Laplace equations. The construction of the VEMs for elliptic problems is very natural and standard, which can be derived based on an integration by parts formula for the underlying differential operator. As a matter of fact, this idea is used to devise conforming and nonconforming VEMs for arbitrary order elliptic problems though the resulting formulation and theoretical analysis are rather involved [27, 39].

VEMs have some advantages over standard finite element methods. For example, they are more convenient to handle partial differential equations on complex geometric domains or the ones associated with high-regularity admissible spaces. Until now, they have been successfully applied to solve various mathematical physical problems, such as the conforming and nonconforming VEMs for second-order elliptic equations [2, 6, 7, 25] and fourth-order elliptic equations [3, 23, 30, 56],

the time-dependent problems [1, 36, 43, 46, 47], the mixed formulation of the Darcy and Stokes problems [17, 22, 57] and the variational inequalities and hemivariational inequalities associated with the frictional contact problems [48–51].

For second-order problems with variable coefficients and convection terms, direct use of the elliptic projection approximation of the gradient operator does not ensure the optimal convergence, so the external projection approximation has been introduced in the literature, see [13, 14] for example. This approximation technique is also commonly used for the construction of virtual element methods for complex problems such as elastic or inelastic mechanics [4, 5, 10]. Since the virtual element space contains at least k-th order polynomials, the number of the degrees of freedom (d.o.f.s) in the virtual element space is generally higher than that in the classical finite element space when the polygonal element is degenerated into a triangle. These extra d.o.f.s are usually caused by internal moments and can be further reduced by exploiting the idea of building an incomplete finite element or the serendipity finite element [21]. In fact, Beirão et al. has proposed the serendipity nodal VEM spaces in [12].

In this paper, we are intended to develop a MATLAB software package for the VEMs in two or three dimensions, containing efficient and easy-following codes for various VEMs published in the literature. In particular, [11] provided a detailed explanation of the formulation of the terms in the matrix equations for the high order virtual element method applied to such a problem in two dimensions, and [44] presented a transparent MATLAB implementation of the conforming linear virtual element method for the Poisson equation in two dimensions. The construction of the VEMs for three-dimensional problems has been accomplished in many papers [9, 15, 16, 28, 31, 34, 35]. However, to the best of knowledge, no related implementation is publicly available in the literature. As an extension of [44] to three spatial dimensions, we have provided a clear and useable MATLAB implementation of the method for three-dimensional linear VEMs for the reaction-diffusion problems on general polyhedral meshes in the package with the detailed implementation given in Section 10. Although the current procedure is only for first-order virtual element spaces, the design idea can be directly generalized to higher-order cases.

The paper is organized as follows. In Section 2, we provide the complete and detailed implementation of the mixed VEMs for the Darcy problem as an example, which includes almost all of the programming techniques in mVEM, such as the construction of the data structure for polygonal meshes, the computation of elliptic and L^2 projection matrices, the treatment of boundary conditions, and examples that demonstrate the usage of running the codes, showing the solutions and meshes, computing the discrete errors and displaying the convergence rates. Section 3 summarizes the mesh related built-in functions, including the modified version of PolyMesher introduced in [45] and generation of some special meshes for VEM tests. We also provide the generation of polygonal meshes by establishing the dual mesh of a Delaunay triangulation and some basic functions to show the polygonal meshes as well as a boundary setting function to identify the Neumann and Dirichlet boundaries. Sections 4-9 discuss the construction of virtual element methods for various model problems and the computation of the corresponding elliptical projections, for example, the conforming and nonconforming VEMs for the Poisson equation and the linear elasticity problems,

the adaptive VEMs for the Poisson equation and the variational inequality for the simplified friction problem. The paper ends with some concluding remarks in Section 11.

2 Mixed VEMs for the Darcy problem

Considering that the implementation of conforming or nonconforming VEMs has been publicly available in the literature [42,44], we in this paper present the detailed implementation of the mixed VEMs for the Darcy problem to fill the gap in this regard. The mixed VEM is first proposed in [22] for the classical model problem of Darcy flow in a porous medium. Let's briefly review the method in this section.

2.1 Construction of the mixed VEMs

2.1.1 The model problem

Given the (polygonal) computational domain $\Omega \subset \mathbb{R}^2$, let $f \in L^2(\Omega)$ and $g \in H^{1/2}(\partial\Omega)$. The Darcy problem is to find $p \in H^1(\Omega)$ such that

$$\begin{cases} -\mathrm{div}(\mathbb{K}\nabla p) = f & \text{in } \Omega, \\ (\mathbb{K}\nabla p)\cdot \boldsymbol{n} = g & \text{on } \partial\Omega, \end{cases} \tag{1}$$

where $\mathbb{K}$ is a symmetric and positive definite tensor of size 2×2. For simplicity, we assume that $\mathbb{K}$ is constant. The given data f and g satisfy the compatibility condition

$$\int_\Omega f\mathrm{d}x = \int_{\partial\Omega} g\mathrm{d}s.$$

To remove an additive constant, we additionally require that

$$\int_\Omega p\mathrm{d}x = 0. \tag{2}$$

Introducing the velocity variable $\boldsymbol{u} = \mathbb{K}\nabla p$, the above problem can be rewritten in the mixed form

$$\begin{cases} \boldsymbol{u} = \mathbb{K}\nabla p & \text{in } \Omega, \\ \mathrm{div}\boldsymbol{u} = -f & \text{in } \Omega, \\ \boldsymbol{u}\cdot\boldsymbol{n} = g & \text{on } \partial\Omega. \end{cases}$$

Define

$$V_g = \{\boldsymbol{u} \in H(\mathrm{div};\Omega) : \boldsymbol{u}\cdot\boldsymbol{n} = g \ \text{ on } \partial\Omega\}, \qquad Q = L_0^2(\Omega), \qquad V = V_0.$$

The corresponding mixed variational problem is: Find $(\boldsymbol{u}, p) \in V_g \times Q$ such that

$$\begin{cases} a(\boldsymbol{u},\boldsymbol{v}) + b(\boldsymbol{v},p) = 0, & \boldsymbol{v}\in V, \\ b(\boldsymbol{u},q) = -(f,q), & q \in Q, \end{cases} \tag{3}$$

where

$$a(\boldsymbol{u},\boldsymbol{v}) = (\mathbb{K}^{-1}\boldsymbol{u},\boldsymbol{v}), \qquad b(\boldsymbol{v},q) = (\mathrm{div}\boldsymbol{v},q).$$

Note that the boundary condition is now related to the variable $\boldsymbol{u}$.

2.1.2 The virtual element space and the elliptic projection

The local virtual element space of V is

$$V_k(K) = \{\boldsymbol{v} \in H(\mathrm{div}; K) \cap H(\mathrm{rot}; K) : \boldsymbol{v} \cdot \boldsymbol{n}|_e \in \mathbb{P}_k(e), \\ \mathrm{div}\boldsymbol{v}|_K \in \mathbb{P}_{k-1}(K), \quad \mathrm{rot}\boldsymbol{v}|_K \in \mathbb{P}_{k-1}(K)\}, \tag{4}$$

where

$$\mathrm{div}\boldsymbol{v} = \partial_1 v_1 + \partial_2 v_2, \quad \mathrm{rot}\boldsymbol{v} = \partial_1 v_2 - \partial_2 v_1, \quad \boldsymbol{v} = (v_1, v_2)^T.$$

To present the degrees of freedom (d.o.f.s), we introduce a scaled monomial $\mathbb{M}_r(D)$ on a d-dimensional domain D

$$\mathbb{M}_r(D) := \Big\{\Big(\frac{\boldsymbol{x} - \boldsymbol{x}_D}{h_D}\Big)^{\boldsymbol{s}}, \quad |\boldsymbol{s}| \le r\Big\},$$

where h_D is the diameter of D, $\boldsymbol{x}_D$ the centroid of D, and r a non-negative integer. For the multi-index $\boldsymbol{s} \in \mathbb{N}^d$, we follow the usual notation

$$\boldsymbol{x}^{\boldsymbol{s}} = x_1^{s_1} \cdots x_d^{s_d}, \quad |\boldsymbol{s}| = s_1 + \cdots + s_d.$$

Conventionally, $\mathbb{M}_r(D) = \{0\}$ for $r \le -1$.

The d.o.f.s can be given by

$$\int_e \boldsymbol{v} \cdot \boldsymbol{n} q \mathrm{d}s, \quad q \in \mathbb{M}_k(e), \quad e \subset \partial K, \tag{5}$$

$$\int_K \boldsymbol{v} \cdot \nabla q \mathrm{d}x, \quad q \in \mathbb{M}_{k-1}(K) \backslash \{1\}, \tag{6}$$

$$\int_K \mathrm{rot}\boldsymbol{v} q \mathrm{d}x, \quad q \in \mathbb{M}_{k-1}(K). \tag{7}$$

We remark that the moments on edges or elements are not divided by $|e|$ or $|K|$ since $a^K(\cdot,\cdot)$ is associated with the L^2 norm rather than the H^1 semi-norm for the case of Poisson equation.

For the Poisson equation, the elliptic projection maps from the virtual element space $V_k(K)$ into the polynomial space $\mathbb{P}_k(K)$ [2, 7, 11]. For convenience, the image space is referred to as the elliptic projection space. For the Darcy problem, however, the elliptic projection space is now replaced by

$$\widehat{V}_k(K) = \{\widehat{\boldsymbol{v}} \in V_k(K) : \widehat{\boldsymbol{v}} = \mathbb{K}\nabla\widehat{q}_{k+1} \ \text{ for some } \ \widehat{q}_{k+1} \in \mathbb{P}_{k+1}(K)\}.$$

The elliptic projector $\widehat{\Pi}^K : V_k(K) \to \widehat{V}_k(K)$, $\boldsymbol{v} \mapsto \widehat{\Pi}^K\boldsymbol{v}$ is then defined by

$$a^K(\widehat{\Pi}^K\boldsymbol{v}, \widehat{\boldsymbol{w}}) = a^K(\boldsymbol{v}, \widehat{\boldsymbol{w}}), \quad \widehat{\boldsymbol{w}} \in \widehat{V}_k(K). \tag{8}$$

We now consider the computability of the elliptic projection. In view of the symmetry of $\mathbb{K}$, the integration by parts gives

$$\begin{aligned} a^K(\boldsymbol{v}, \widehat{\boldsymbol{w}}) &= \int_K \mathbb{K}^{-1}\boldsymbol{v} \cdot \widehat{\boldsymbol{w}} \mathrm{d}x = \int_K \boldsymbol{v} \cdot (\mathbb{K}^{-1}\widehat{\boldsymbol{w}}) \mathrm{d}x \\ &= \int_K \boldsymbol{v} \cdot \nabla\widehat{q}_{k+1} \mathrm{d}x = -\int_K \widehat{q}_{k+1} \mathrm{div}\boldsymbol{v} \mathrm{d}x + \int_{\partial K} \widehat{q}_{k+1}\boldsymbol{v} \cdot \boldsymbol{n} \mathrm{d}s. \end{aligned} \tag{9}$$

- For the second term, since $\boldsymbol{v}\cdot\boldsymbol{n}|_e \in \mathbb{P}_k(e)$ we expand it in the scaled monomials on e as

$$\boldsymbol{v}\cdot\boldsymbol{n}|_e(s) = c_1 m_1^e(s) + \cdots + c_n m_n^e(s).$$

  Clearly, the coefficients are uniquely determined by the d.o.f.s in (5).

- For the first term, noting that $\mathrm{div}\boldsymbol{v}|_K \in \mathbb{P}_{k-1}(K)$, we have

$$\mathrm{div}\boldsymbol{v}|_K(x) = c_1 m_1^K(x) + \cdots + c_n m_n^K(x).$$

  Let $q = m_i^K(x)$ and take inner product on both sides with respect to q. We obtain

$$\int_K \mathrm{div}\boldsymbol{v} q \mathrm{d}x = -\int_K \boldsymbol{v}\cdot\nabla q \mathrm{d}x + \int_{\partial K} \boldsymbol{v}\cdot\boldsymbol{n} q \mathrm{d}s, \quad q = m_i^K(x) \in \mathbb{P}_{k-1}(K).$$

  Hence the coefficients are determined by the d.o.f.s in (5) and (6).

In this paper, we only consider the implementation of the lowest order case $k = 1$. At this time, only the d.o.f.s of the first and third types exist. The local d.o.f.s will be arranged as

$$\chi_i(\boldsymbol{v}) = \int_{e_i} (\boldsymbol{v}\cdot\boldsymbol{n})\mathrm{d}s, \quad i = 1, \cdots, N_v, \tag{10}$$

$$\chi_{N_v+i}(\boldsymbol{v}) = \int_{e_i} (\boldsymbol{v}\cdot\boldsymbol{n})\frac{s - s_{e_i}}{h_{e_i}}\mathrm{d}s, \quad i = 1, \cdots, N_v, \tag{11}$$

$$\chi_{2N_v+1}(\boldsymbol{v}) = \int_K \mathrm{rot}\boldsymbol{v}\mathrm{d}x, \tag{12}$$

where N_v is the number of the vertices of K, h_e is the length of e, and s is the natural parameter of e with s_e being the midpoint in the parametrization.

2.1.3 The discrete problem

In what follows, we denote the global virtual element space by V_h associated with $V_k(K)$, and Q_h by the discretization of Q, given as

$$Q_h := \{q \in Q : q|_K \in \mathbb{P}_{k-1}(K), \quad K \in \mathcal{T}_h.\}$$

In particular, Q_h is piecewise constant for $k = 1$.

The VEM approximation of $a^K(\boldsymbol{u}, \boldsymbol{v})$ is

$$a_h^K(\boldsymbol{u}, \boldsymbol{v}) = a^K(\widehat{\Pi}^K\boldsymbol{u}, \widehat{\Pi}^K\boldsymbol{v}) + \|\mathbb{K}^{-1}\| S^K(\boldsymbol{u} - \widehat{\Pi}^K\boldsymbol{u}, \boldsymbol{v} - \widehat{\Pi}^K\boldsymbol{v}),$$

where $\|\cdot\|$ is the Frobenius norm and the stabilization term is

$$S^K(\boldsymbol{v}, \boldsymbol{w}) = \sum_{i=1}^{2N_v+1} \chi_i(\boldsymbol{v})\chi_i(\boldsymbol{w}).$$

The discrete mixed variational problem is: Find $(\boldsymbol{u}_h, p_h) \in V_h^g \times Q_h$ such that

$$\begin{cases} a_h(\boldsymbol{u}_h, \boldsymbol{v}_h) + b(\boldsymbol{v}_h, p_h) &= 0, \quad \boldsymbol{v}_h \in V_h, \\ b(\boldsymbol{u}_h, q_h) &= -(f, q_h), \quad q_h \in Q_h. \end{cases} \tag{13}$$

The constraint (2) is not naturally imposed in the above system. To this end, we introduce a Lagrange multiplier and consider the augmented variational formulation: Find $((\boldsymbol{u}_h, p_h), \lambda) \in V_h^g \times Q_h \times \mathbb{R}$ such that

$$\begin{cases} a_h(\boldsymbol{u}_h, \boldsymbol{v}_h) + b(\boldsymbol{v}_h, p_h) & = 0, \quad \boldsymbol{v}_h \in V_h, \\ b(\boldsymbol{u}_h, q_h) + \lambda \displaystyle\int_\Omega q_h \mathrm{d}x & = -(f, q_h), \quad q_h \in Q_h, \\ \mu \displaystyle\int_\Omega p_h \mathrm{d}x & = 0, \quad \mu \in \mathbb{R}. \end{cases} \tag{14}$$

It should be pointed out that the virtual element space V_h cannot be understood as a vector or tensor-product space, which is different from the conforming or nonconforming VEMs for the linear elasticity problem. In the computation, $\boldsymbol{u}_h$ should be viewed as a scalar at this time. Let $\boldsymbol{\varphi}_i$, $i = 1, \cdots, N$ be the nodal basis functions of V_h, where N is the dimension of V_h. We can write

$$\boldsymbol{u} = \sum_{i=1}^{N} \chi_i(\boldsymbol{u}) \boldsymbol{\varphi}_i =: \boldsymbol{\varphi}^T \boldsymbol{\chi}(\boldsymbol{u}).$$

The basis functions of Q_h are denoted by ψ_l, $l = 1, \cdots, M$:

$$p_h = \sum_{l=1}^{M} p_l \psi_l.$$

Plug above equations in (13), and take $\boldsymbol{v}_h = \boldsymbol{\varphi}_j$ and $q_h = \psi_l$. We have

$$\begin{cases} \displaystyle\sum_{i=1}^{N} a_h(\boldsymbol{\varphi}_i, \boldsymbol{\varphi}_j)\chi_i + \sum_{l=1}^{M} b(\boldsymbol{\varphi}_j, \psi_l) p_l & = 0, \quad j = 1, \cdots, N, \\ \displaystyle\sum_{i=1}^{N} b(\boldsymbol{\varphi}_i, \psi_l)\chi_i + \lambda \int_\Omega \psi_l \mathrm{d}x & = -(f, \psi_l), \quad l = 1, \cdots, M, \\ \displaystyle\sum_{l=1}^{M} \int_\Omega \psi_l \mathrm{d}x p_l & = 0. \end{cases}$$

Let

$$d_l = \int_\Omega \psi_l \mathrm{d}x, \quad \boldsymbol{d} = [d_1, \cdots, d_M]^T. \tag{15}$$

The linear system can be written in matrix form as

$$\begin{bmatrix} A & B & \mathbf{0} \\ B^T & O & \boldsymbol{d} \\ \mathbf{0}^T & \boldsymbol{d}^T & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{\chi} \\ \boldsymbol{p} \\ \lambda \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \boldsymbol{f} \\ 0 \end{bmatrix}, \tag{16}$$

where

$$A = (a_h(\boldsymbol{\varphi}_j, \boldsymbol{\varphi}_i))_{N\times N}, \quad B = (b(\boldsymbol{\varphi}_j, \psi_l))_{N\times M}, \quad \boldsymbol{f} = (-(f, \psi_l))_{M\times 1}.$$

For the global d.o.f.s in the unknown vector $\boldsymbol{\chi}$ in (16), we shall arrange the first type in (10), followed by the second and the third ones in (11) and (12).

2.2 Implementation

2.2.1 Overview of the code

We first provide an overview of the test script.

```
%% Parameters
nameV = [32, 64, 128, 256, 512];
maxIt = length(nameV);
h = zeros(maxIt,1);    N = zeros(maxIt,1);
ErruL2 = zeros(maxIt,1);
ErrpL2 = zeros(maxIt,1);
ErrI = zeros(maxIt,1);

%% PDE data
pde = Darcydata;

%% Virtual element method
for k = 1:maxIt
    % load mesh
    fprintf('Mesh %d: \n', k);
    load( ['meshdata', num2str(nameV(k)), '.mat'] );
    % get boundary information
    bdStruct = setboundary(node,elem);
    % solve the problem
    [uh,ph,info] = Darcy_mixedVEM(node,elem,pde,bdStruct);
    % record and plot
    N(k) = length(uh);  h(k) = 1/sqrt(size(elem,1));
    [uhI,phI,nodeI,elemI] = ProjectionDarcy(node,elem,uh,ph,info,pde);
    figure(1);
    showresult(nodeI,elemI,pde.uexact,uhI);
    %showresult(nodeI,elemI,pde.pexact,phI);
    drawnow; %pause(0.1);
    % compute errors in discrete L2 norm
    [ErruL2(k),ErrpL2(k)] = getL2error_Darcy(node,elem,uh,ph,info,pde);
end

%% Plot convergence rates and display error table
figure,
showrateh(h,ErruL2,'r-*','||u-u_h||',  ErrpL2, 'b-s','||p-p_h||')

fprintf('\n');
disp('Table: Error')
colname = {'#Dof','h','||u-u_h||','||p-p_h||'};
disptable(colname,N,[],h,'%0.3e',ErruL2,'%0.5e',ErrpL2,'%0.5e');
```

In the `for` loop, we first load or generate the mesh data, which immediately returns the matrix `node` and the cell array `elem` defined later to the MATLAB workspace. Then we set up the boundary conditions to get the structural information of the boundary edges. The subroutine `Darcy_mixedVEM.m` is the function file containing all source code to implement the VEM. When obtaining the numerical solutions, we can visualize the piecewise elliptic projection $\widehat{\Pi}^K \boldsymbol{u}_h$ by using the subroutines `ProjectionDarcy.m` and `showresult.m`. We then calculate the discrete L^2 error defined as

$$\mathrm{ErrL2} = \left(\sum_{K \in \mathcal{T}_h} \| \boldsymbol{u} - \widehat{\Pi}^K \boldsymbol{u}_h \|_{0,E}^2 \right)^{1/2}, \tag{17}$$

through the subroutine `getL2error_Darcy.m`. The procedure is completed by verifying the rate of convergence through `showrateh.m`.

The overall structure of a virtual element method implementation will be much the same as

for a standard finite element method, as outlined in Algorithm 1.

Algorithm 1 An overall structure of the implementation of a virtual element method

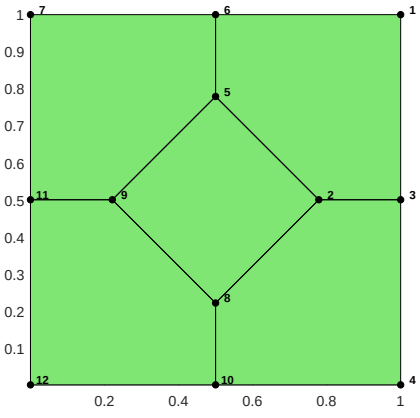
Input: Mesh data and PDE data

1. Get auxiliary data of the mesh, including some data structures and geometric quantities;
2. Derive elliptic projections;
3. Compute and assemble the linear system by looping over the elements;
4. Apply the boundary conditions;
5. Set solver and store information for computing errors.

Output: The numerical DoFs

2.2.2 Data structure

We first discuss the data structure to represent polygonal meshes so as to facilitate the implementation. There are two basic data structures `node` and `elem`, where `node` is a matrix with the first and second columns contain x- and y-coordinates of the nodes in the mesh, and `elem` is a cell array recording the vertex indices of each element in a counterclockwise order as shown in Fig. 1. The mesh can be displayed by using `showmesh.m`.



(a) mesh

	1	2
1	1.0000	1.0000
2	0.7789	0.5000
3	1.0000	0.5000
4	1.0000	1.1636e-10
5	0.5000	0.7790
6	0.5000	1.0000
7	1.1635e-10	1.0000
8	0.5000	0.2211
9	0.2210	0.5000
10	0.5000	1.1636e-10
11	1.1636e-10	0.5000
12	1.1636e-10	1.1636e-10

(b) node

	1
1	[8,10,4,3,2]
2	[2,3,1,6,5]
3	[11,9,5,6,7]
4	[9,8,2,5]
5	[12,10,8,9,11]

(c) elem

Fig. 1. Example of a polygonal mesh

Using the basic data structures, we can extract the topological or combinatorial structure of a polygonal mesh. These data are referred to as the auxiliary data structures as given in Tab. 1. The combinatorial structure will benefit the implementation of virtual element methods. The idea stems from the treatment of triangulation in *i*FEM for finite element methods [26], which is generalized to polygonal meshes with certain modifications.

Tab. 1. Auxiliary data structures

`edge`
`elem2edge`
`bdEdge`
`edge2elem`
`neighbor`
`node2elem`

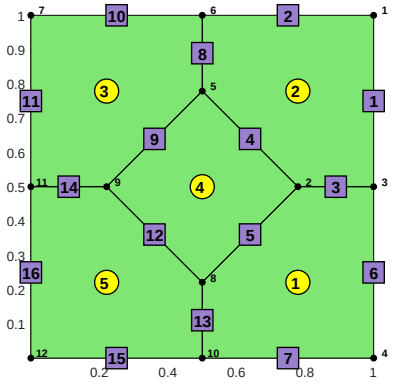


Fig. 2. Illustration of the auxiliary data structures.

edge. The d.o.f.s in (10) and (11) are edge-oriented. In the matrix `edge(1:NE,1:2)`, the first and second rows contain indices of the starting and ending points. The column is sorted in the way that for the k-th edge, `edge(k,1)<edge(k,2)`. The indices of these edges are are marked with purple boxes as shown in Fig. 2.

Following [26], we shall use the name convention a2b to represent the link form a to b. This link is usually the map from the local index set to the global index set. Throughout this paper, we use the symbols `N`, `NT`, and `NE` to represent the number of nodes, elements, and edges.

elem2edge. The cell array `elem2edge` establishes the map of local index of edges in each polygon to its global index in matrix `edge`. For instance, `elem2edge{1} = [13,7,6,3,5]` for the mesh in Fig. 2.

bdEdge. This matrix exacts the boundary edges from `edge`.

edge2elem. The matrix `edge(1:NE,1:2)` records the neighboring polygons for each edge. In Fig. 2, `edge(3,1:2)= [1,2]`. For a boundary edge, the outside is specified to the element sharing it as an edge.

neighbor. We use the cell array `neighbor` to record the neighboring polygons for each element. For example, `neighbor{4} = [5,1,2,3]`, where the i-th entry corresponds to the i-th edge of the current element. Note that for a boundary edge, the neighboring polygon is specified to the current element, for instance, `neighbor{1} = [5,1,1,2,4]`.

node2elem. This cell array finds the elements sharing a common nodes. For example, `node2elem{2} = [1,2,4]`.

In addition, we provide a subroutine `auxgeometry.m` to compute some useful geometric quantities, such as the barycenter `centroid`, the diameter `diameter` and the area `area` of each element.

2.2.3 Computation of the elliptic projection

Transition matrix

The shape functions of $V_k(K)$ are written in the following compact notation

$$\boldsymbol{\phi}^T = (\phi_1, \phi_2, \cdots, \phi_{N_k}),$$

where $N_k = 2N_v + 1$ is the cardinality of the local basis set. The basis of $\widehat{V}_k(K)$ is given by

$$\widehat{\boldsymbol{m}}^T = (\widehat{\boldsymbol{m}}_1, \widehat{\boldsymbol{m}}_2, \cdots, \widehat{\boldsymbol{m}}_{\widehat{N}_p}).$$

Noting that $\widehat{V}_k(K) \subset V_k(K)$, we set $\widehat{\boldsymbol{m}}^T = \boldsymbol{\phi}^T \boldsymbol{D}$, where $\boldsymbol{D}$ is referred to as the transition matrix from the elliptic projection space $\widehat{V}_k(K)$ to the virtual element space $V_k(K)$. By the definition of the d.o.f.s,

$$\widehat{\boldsymbol{m}}_\alpha = \sum_{i=1}^{N_k} \boldsymbol{\phi}_i \boldsymbol{D}_{i\alpha}, \quad \boldsymbol{D}_{i\alpha} = \chi_i(\widehat{\boldsymbol{m}}_\alpha).$$

For every $\boldsymbol{v} \in \widehat{V}_k(K)$, by definition, $\boldsymbol{v} \in \mathbb{K}\nabla\mathbb{P}_{k+1}(K)$. For $k = 1$, the scaled monomials of $\mathbb{P}_{k+1}(K) = \mathbb{P}_2(K)$ are

$$m^T = (m_1, m_2, \cdots, m_{N_m}), \quad N_m = 6,$$

with

$$m_1(x,y) = 1, \quad m_2(x,y) = \frac{x - x_K}{h_K}, \quad m_3(x,y) = \frac{y - y_K}{h_K},$$
$$m_4(x,y) = \frac{(x - x_K)^2}{h_K^2}, \quad m_5(x,y) = \frac{(x - x_K)(y - y_K)}{h_K^2}, \quad m_6(x,y) = \frac{(y - y_K)^2}{h_K^2},$$

where (x_K, y_K) and h_K are the barycenter and diameter of K, respectively. We then choose

$$\widehat{\boldsymbol{m}}_\alpha = \mathbb{K}\nabla(h_K m_{\alpha+1}), \quad \alpha = 1, \cdots, \widehat{N}_m = N_m - 1 = 5, \tag{18}$$

with the explicit expressions given by

$$\widehat{\boldsymbol{m}}_1 = \mathbb{K}\begin{bmatrix}1\\0\end{bmatrix}, \quad \widehat{\boldsymbol{m}}_2 = \mathbb{K}\begin{bmatrix}0\\1\end{bmatrix}, \quad \widehat{\boldsymbol{m}}_3 = \mathbb{K}\begin{bmatrix}2m_2\\0\end{bmatrix}, \quad \widehat{\boldsymbol{m}}_4 = \mathbb{K}\begin{bmatrix}m_3\\m_2\end{bmatrix}, \quad \widehat{\boldsymbol{m}}_5 = \mathbb{K}\begin{bmatrix}0\\2m_3\end{bmatrix}.$$

We now compute the transition matrix. For χ_i with $i = 1, \cdots, N_v$ in (10), noting that $\boldsymbol{v} \cdot \boldsymbol{n}|_e \in \mathbb{P}_1(e)$, the trapezoidal rule gives

$$\chi_i(\boldsymbol{v}) = \int_{e_i} (\boldsymbol{v} \cdot \boldsymbol{n}) \mathrm{d}s = \frac{1}{2}(\boldsymbol{v}(z_i) + \boldsymbol{v}(z_{i+1})) \cdot (h_{e_i} \boldsymbol{n}_{e_i}), \quad \boldsymbol{v} = \widehat{\boldsymbol{m}}_\alpha. \tag{19}$$

For χ_{N_v+i} with $i = 1, \cdots, N_v$ in (11), using the Simpson formula yields

$$\begin{aligned}
\chi_{N_v+i}(\boldsymbol{v}) &= \int_{e_i} (\boldsymbol{v} \cdot \boldsymbol{n}) \frac{s - s_{e_i}}{h_{e_i}} \mathrm{d}s = \frac{h_{e_i}}{6}(f(z_i) + 4f(s_{e_i}) + f(z_{i+1})) \\
&= \frac{h_{e_i}}{6}(f(z_i) + f(z_{i+1})), \quad f = (\boldsymbol{v} \cdot \boldsymbol{n}) \frac{s - s_{e_i}}{h_{e_i}} \\
&= \frac{1}{6}\Big(\boldsymbol{v}(z_i) \cdot \boldsymbol{n}_{e_i}(s_i - s_{e_i}) + \boldsymbol{v}(z_{i+1}) \cdot \boldsymbol{n}_{e_i}(s_{i+1} - s_{e_i})\Big) \\
&= \frac{1}{6}\Big(\boldsymbol{v}(z_i) \cdot \boldsymbol{n}_{e_i} \frac{-1}{2} h_{e_i} + \boldsymbol{v}(z_{i+1}) \cdot \boldsymbol{n}_{e_i} \frac{1}{2} h_{e_i}\Big) \\
&= \frac{1}{12}\Big(\boldsymbol{v}(z_{i+1}) - \boldsymbol{v}(z_i)\Big) \cdot (\boldsymbol{n}_{e_i} h_{e_i}), \quad \boldsymbol{v} = \widehat{\boldsymbol{m}}_\alpha.
\end{aligned} \tag{20}$$

For χ_{2N_v+1} in (12), the integration by parts gives

$$\begin{aligned}
\chi_{2N_v+1}(\boldsymbol{v}) &= \int_K \mathrm{rot}\boldsymbol{v} \mathrm{d}x = \int_K (\partial_1 v_2 - \partial_2 v_1) \mathrm{d}x \\
&= \int_{\partial K} (v_2 n_1 - v_1 n_2) \mathrm{d}s = \int_{\partial K} (\boldsymbol{v} \cdot \boldsymbol{t}) \mathrm{d}s \\
&= \sum_{i=1}^{N_v} \frac{1}{2}(\boldsymbol{v}(z_i) + \boldsymbol{v}(z_{i+1})) \cdot (h_{e_i} \boldsymbol{t}_{e_i}), \quad \boldsymbol{v} = \widehat{\boldsymbol{m}}_\alpha.
\end{aligned}$$

According to the above discussion, the transition matrix is calculated as follows.

```
K = pde.K; % coefficient matrix
    % ------- element information ----------
    index = elem{iel};     Nv = length(index);
    xK = centroid(iel,1); yK = centroid(iel,2); hK = diameter(iel);
    x = node(index,1); y = node(index,2);
    v1 = 1:Nv; v2 = [2:Nv,1]; % loop index for vertices or edges
    xe = (x(v1)+x(v2))/2;  ye = (y(v1)+y(v2))/2; % mid-edge points
    Ne = [y(v2)-y(v1), x(v1)-x(v2)]; % he*ne
    Te = [-Ne(:,2), Ne(:,1)]; % he*te

    % ------- scaled monomials --------
    m2 = @(x,y) (x-xK)./hK;
    m3 = @(x,y) (y-yK)./hK;
    m4 = @(x,y) (x-xK).^2./hK^2;
    m5 = @(x,y) (x-xK).*(y-yK)./hK^2;
    m6 = @(x,y) (y-yK).^2./hK^2;
    % \hat{m}_a = K*grad(hK*m_{a+1})
    mh1 = @(x,y) [K(1,1)+0*x; K(2,1)+0*x];
    mh2 = @(x,y) [K(1,2)+0*x; K(2,2)+0*x];
    mh3 = @(x,y) [2*K(1,1)*m2(x,y); 2*K(2,1)*m2(x,y)];
    mh4 = @(x,y) [K(1,1)*m3(x,y)+K(1,2)*m2(x,y); K(2,1)*m3(x,y)+K(2,2)*m2(x,y)];
    mh5 = @(x,y) [2*K(1,2)*m3(x,y); 2*K(2,2)*m3(x,y)];
    mh = @(x,y) [mh1(x,y), mh2(x,y), mh3(x,y), mh4(x,y), mh5(x,y)];

    % -------- transition matrix ----------
    NdofA = 2*Nv+1; Nmh = 5;
    D = zeros(NdofA,Nmh);
    for i = 1:Nv % loop of edges
        % v at z_i, z_{i+1} for v = [mK1,...,mK5]
        va = mh(x(v1(i)),y(v1(i))); vb = mh(x(v2(i)),y(v2(i)));
        % chi_i, i = 1,...,Nv
        D(i,:) =  1/2*Ne(i,:)*(va+vb);
        % chi_{Nv+i}, i = 1,...,Nv
        D(Nv+i,:) =  1/12*Ne(i,:)*(vb-va);
        % chi_{2Nv+1}
        D(end,:) = D(end,:) + 1/2*Te(i,:)*(va+vb);
    end
```

Here, `K` is for $\mathbb{K}$ and `mhat` is for $\widehat{\boldsymbol{m}}^T$.

Elliptic projection matrices

We denote the matrix representation of the $\widehat{\Pi}^K$-projection by $\widehat{\mathbf{\Pi}}^K$ in the sense that

$$\widehat{\Pi}^K(\boldsymbol{\phi}_1, \boldsymbol{\phi}_2, \cdots, \boldsymbol{\phi}_{N_k}) = (\boldsymbol{\phi}_1, \boldsymbol{\phi}_2, \cdots, \boldsymbol{\phi}_{N_k})\widehat{\mathbf{\Pi}}^K \quad \text{or} \quad \widehat{\Pi}^K\boldsymbol{\phi}^T = \boldsymbol{\phi}^T\widehat{\mathbf{\Pi}}^K.$$

By the definition of d.o.f.s, the j-th column of $\widehat{\mathbf{\Pi}}^K$ is the d.o.f vector of $\widehat{\Pi}^K\boldsymbol{\phi}_j$, i.e., $\widehat{\mathbf{\Pi}}^K = \left(\chi_i(\widehat{\Pi}^K\boldsymbol{\phi}_j)\right)$. The elliptic projection vector $\widehat{\Pi}^K\boldsymbol{\phi}^T$ can be expanded in the basis $\widehat{\boldsymbol{m}}^T$ of the elliptic projection space $\widehat{V}_k(K)$ as $\widehat{\Pi}^K\boldsymbol{\phi}^T = \widehat{\boldsymbol{m}}^T\widehat{\mathbf{\Pi}}_*^K$. It is easy to check that $\widehat{\mathbf{\Pi}}^K = \boldsymbol{D}\widehat{\mathbf{\Pi}}_*^K$.

The definition 8 is equivalent to

$$a^K(\widehat{\boldsymbol{m}}, \widehat{\Pi}^K\boldsymbol{\phi}^T) = a^K(\widehat{\boldsymbol{m}}, \boldsymbol{\phi}^T) \quad \text{or} \quad \widehat{\boldsymbol{G}}\widehat{\mathbf{\Pi}}_*^K = \widehat{\boldsymbol{B}},$$

where

$$\widehat{\boldsymbol{G}} = a^K(\widehat{\boldsymbol{m}}, \widehat{\boldsymbol{m}}^T), \quad \widehat{\boldsymbol{B}} = a^K(\widehat{\boldsymbol{m}}, \boldsymbol{\phi}^T).$$

We also have the consistency relation $\widehat{\boldsymbol{G}} = \widehat{\boldsymbol{B}}\boldsymbol{D}$.

We now compute $\widehat{\boldsymbol{B}}$. From (9) and (18), one has

$$\begin{aligned}\widehat{\boldsymbol{B}}_{\alpha i} = a^K(\widehat{\boldsymbol{m}}_\alpha, \boldsymbol{\phi}_i) &= -h_K \int_K m_{\alpha+1}\mathrm{div}\boldsymbol{\phi}_i \mathrm{d}x + h_K \int_{\partial K} m_{\alpha+1}\boldsymbol{\phi}_i \cdot \boldsymbol{n}\mathrm{d}s \\ &=: -h_K I_1(\alpha, i) + h_K I_2(\alpha, i),\end{aligned}$$

where

$$I_1(\alpha, i) = \int_K m_{\alpha+1}\mathrm{div}\boldsymbol{\phi}_i \mathrm{d}x, \quad I_2(\alpha, i) = \int_{\partial K} m_{\alpha+1}\boldsymbol{\phi}_i \cdot \boldsymbol{n}\mathrm{d}s.$$

For I_1, noting that $\mathrm{div}\boldsymbol{\phi}_i$ is constant when $k = 1$, set $\mathrm{div}\boldsymbol{\phi}_i = c_i m_1 = c_i$, which gives

$$c_i = |K|^{-1}\int_K \mathrm{div}\boldsymbol{\phi}_i \mathrm{d}x = |K|^{-1}\int_{\partial K} \boldsymbol{\phi}_i \cdot \boldsymbol{n}\mathrm{d}s = \begin{cases} |K|^{-1}, & i = 1, \cdots, N_v, \\ 0, & i > N_v, \end{cases}$$

and hence

$$I_1(\alpha, i) = \int_K m_{\alpha+1}\mathrm{div}\boldsymbol{\phi}_i \mathrm{d}x = c_i \int_K m_{\alpha+1}\mathrm{d}x.$$

The first term is now computed in MATLAB as follows.

```
% first term
nodeT = [node(index,:);centroid(iel,:)];
elemT = [(Nv+1)*ones(Nv,1),(1:Nv)',[2:Nv,1]'];
m = @(x,y) [m2(x,y), m3(x,y), m4(x,y), m5(x,y), m6(x,y)]; % m_{a+1},...
ci = zeros(1,NdofA); ci(1:Nv) = 1/area(iel);
Intm = integralTri(m,3,nodeT,elemT);
I1 = Intm'*ci;
```

The subroutine `integralTri.m` calculates the integral on a polygonal element which is triangulated with the basic data structures `nodeT` and `elemT`.

For I_2, since $\boldsymbol{\phi}_i \cdot \boldsymbol{n}|_{e_j} \in \mathbb{P}_1(e_j)$ $(i = 1, \cdots, 2N_v + 1)$, set

$$\boldsymbol{\phi}_i \cdot \boldsymbol{n}_{e_j}|_{e_j} = c_0^i + c_1^i \frac{s - s_{e_j}}{h_{e_j}},$$

where s it the natural parameter of the edge e_j and s_{e_j} is the mid-point in the local parametrization. Noting that

$$\int_e \Big(\frac{s - s_e}{h_e}\Big)^\alpha \Big(\frac{s - s_e}{h_e}\Big)^\beta \mathrm{d}s = \frac{h_e}{(\alpha + \beta + 1)}\Big(\frac{1}{2^{\alpha+\beta+1}} - \frac{1}{(-2)^{\alpha+\beta+1}}\Big),$$

we then obtain

$$\begin{cases} h_{e_j} c_0^i = \int_{e_j} \boldsymbol{\phi}_i \cdot \boldsymbol{n}\mathrm{d}s =: r_j^i, \\ \frac{1}{12} h_{e_j} c_1^i = \int_{e_j} \boldsymbol{\phi}_i \cdot \boldsymbol{n} \frac{s - s_{e_j}}{h_{e_j}}\mathrm{d}s =: s_j^i. \end{cases}$$

By definition, the nonzero elements of r_j^i and s_j^i are

$$r_j^j = s_j^{j+N_v} = 1, \quad j = 1, \cdots, N_v.$$

We have

$$c_0^i = \frac{1}{h_{e_j}} r_j^i, \quad c_1^i = \frac{12}{h_{e_j}} s_j^i,$$

and hence

$$\boldsymbol{r}_j^i(s) := \boldsymbol{\phi}_i \cdot (h_{e_j}\boldsymbol{n}_{e_j})|_{e_j} = r_j^i + 12 s_j^i \frac{s - s_{e_j}}{h_{e_j}}.$$

It is obvious that

$$\boldsymbol{r}_j^i(s_a) = r_j^i - 6s_j^i, \quad \boldsymbol{r}_j^i(s_e) = r_j^i, \quad \boldsymbol{r}_j^i(s_b) = r_j^i + 6s_j^i,$$

where, s_a, s_e and s_b correspond to the starting point, middle point and ending point in the local parameterization, respectively.

Observing that the Simpson formula is accurate for cubic polynomials, we have

$$\begin{aligned} I_2(\alpha, i) &= \sum_{j=1}^{N_v} \int_{e_j} m_{\alpha+1}\boldsymbol{\phi}_i \cdot \boldsymbol{n} \mathrm{d}s \\ &= \sum_{j=1}^{N_v} \frac{h_{e_j}}{6}(m_{\alpha+1}\boldsymbol{\phi}_i(z_j) + 4m_{\alpha+1}\boldsymbol{\phi}_i(z_{e_j}) + m_{\alpha+1}\boldsymbol{\phi}_i(z_{j+1}) \cdot \boldsymbol{n}_{e_j} \\ &= \frac{1}{6}\sum_{j=1}^{N_v} m_{\alpha+1}(z_j)\boldsymbol{r}_j^i(s_a) + 4m_{\alpha+1}(z_{e_j})\boldsymbol{r}_j^i(s_e) + m_{\alpha+1}(z_{j+1})\boldsymbol{r}_j^i(s_b) \end{aligned}$$

or

$$I_2(\alpha, :) = \frac{1}{6}\sum_{j=1}^{N_v} m_{\alpha+1}(z_j)\boldsymbol{r}_j(s_a) + 4m_{\alpha+1}(z_{e_j})\boldsymbol{r}_j(s_e) + m_{\alpha+1}(z_{j+1})\boldsymbol{r}_j(s_b).$$

Then the second term and the matrices $\widehat{\boldsymbol{B}}$ and $\widehat{\boldsymbol{G}}$ can be computed as follows.

```
% second term
rij = zeros(Ndof,Nv); rij(1:Nv,:) = eye(Nv);
sij = zeros(Ndof,Nv); sij(Nv+1:2*Nv,:) = eye(Nv);
rija = rij- 6*sij;
rije = rij;
rijb = rij + 6*sij;
I2 = zeros(np,Ndof);
for j = 1:Nv
    ma = m(x(v1(j)),y(v1(j)));
    me = m(xe(j),ye(j));
    mb = m(x(v2(j)),y(v2(j)));
    rja = rija(:,j)';  rje = rije(:,j)';  rjb = rijb(:,j)';
    I2 = I2 + 1/6*(ma'*rja + 4*me'*rje + mb'*rjb);
end
% \hat{B} and \hat{G}
Bs = hK*(-I1+I2); Gs = Bs*D
```

2.2.4 Computation and assembly of the stiffness matrix and load vector

The stiffness matrix

The local stiffness matrix of A in (16) is $A^K = A_1^K + A_2^K$, where

$$\begin{aligned} A_1^K(i,j) &= a^K(\widehat{\Pi}^K\boldsymbol{\phi}_j, \widehat{\Pi}^K\boldsymbol{\phi}_i), \\ A_2^K(i,j) &= \|\mathbb{K}^{-1}\| S^K(\boldsymbol{\phi}_j - \widehat{\Pi}^K\boldsymbol{\phi}_j, \boldsymbol{\phi}_i - \widehat{\Pi}^K\boldsymbol{\phi}_i) \\ &= \|\mathbb{K}^{-1}\| \sum_{i=1}^{N_k} \chi_r(\boldsymbol{\phi}_j - \widehat{\Pi}^K\boldsymbol{\phi}_j)\chi_r(\boldsymbol{\phi}_i - \widehat{\Pi}^K\boldsymbol{\phi}_i). \end{aligned}$$

The first term is written in matrix form as

$$A_K^1 = a^K(\widehat{\Pi}^K\boldsymbol{\phi}, \widehat{\Pi}^K\boldsymbol{\phi}^T) = (\widehat{\boldsymbol{\Pi}}_*^K)^T a^K(\widehat{\boldsymbol{m}}, \widehat{\boldsymbol{m}}^T)\widehat{\boldsymbol{\Pi}}_*^K = (\widehat{\boldsymbol{\Pi}}_*^K)^T\widehat{\boldsymbol{G}}\widehat{\boldsymbol{\Pi}}_*^K.$$

For the second term, from $\chi_r(\widehat{\Pi}^K\boldsymbol{\phi}_i) = (\widehat{\boldsymbol{\Pi}}^K)_{ri}$ one has

$$A_K^2 = \|\mathbb{K}^{-1}\|(\mathbf{I} - \widehat{\boldsymbol{\Pi}}^K)^T(\mathbf{I} - \widehat{\boldsymbol{\Pi}}^K).$$

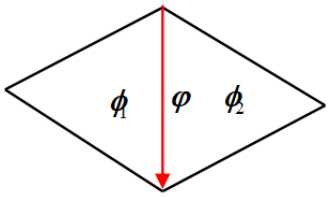


Fig. 3. Illustration of the global and local basis functions

Note that the signs of the d.o.f.s in (10) vary with the edge orientation, which means the global basis function $\boldsymbol{\varphi}_i$ restriction to the element may have the opposite sign with the local basis function $\boldsymbol{\phi}_i$. As shown in Fig. 3, e is an interior edge with the global orientation given by the arrow. Let $\boldsymbol{\varphi}$ be the global basis function, and let $\boldsymbol{\phi}_1$ and $\boldsymbol{\phi}_2$ be the local basis functions corresponding to the left and right elements, respectively. Then one has

$$\boldsymbol{\phi}_1 = -\boldsymbol{\varphi}|_{K_1}, \quad \boldsymbol{\phi}_2 = \boldsymbol{\varphi}|_{K_2}.$$

Obviously, the sign can be determined by computing the difference between the two indices of the end points of e. For this reason, one only needs to compute the elementwise signs of the edges of ∂K. The code is given as

```
%% Get elementwise signs of basis functions
bdEdgeIdx = bdStruct.bdEdgeIdx;  E = false(NE,1); E(bdEdgeIdx) = 1;
sgnBase = cell(NT,1);
for iel = 1:NT
    index = elem{iel};   Nv = length(index);   NdofA = 2*Nv+1;
    sgnedge = sign(diff(index([1:Nv,1])));
    id = elem2edge{iel}; sgnbd = E(id); sgnedge(sgnbd) = 1;
    sgnelem = ones(NdofA,1); sgnelem(1:Nv) = sgnedge;
    sgnBase{iel} = sgnelem;
end
```

Note that the positive signs of the boundary edges are recovered in the above code.

Since $a^K(\pm\boldsymbol{\phi}_i, \pm\boldsymbol{\phi}_j) = \pm\cdot\pm a^K(\boldsymbol{\phi}_i, \boldsymbol{\phi}_j)$, we can introduce a signed stiffness matrix `sgnK` to add the correct signs to the local stiffness matrix. The matrix A^K can be computed in the following way.

```
    % -------- sign matrix and sign vector -------
    sgnelem = sgnBase{iel};
    sgnK = sgnelem*sgnelem';

    % ------------- stiffness matrix -------------
    % Projection
    Pis = Gs\Bs;   Pi  = D*Pis;   I = eye(size(Pi));
```

```
    % Stiffness matrix A
    AK  = Pis'*Gs*Pis + norm(inv(K),'fro')*(I-Pi)'*(I-Pi);  % G = Gs
    AK = AK.*sgnK;
    AK = reshape(AK',1,[]); % straighten as row vector for easy assembly
```

By the definition of Q_h, on each element K, the basis function $\psi_l \in \mathbb{P}_{k-1}(K)$. Only one basis $\psi_l = m_1 = 1$ for $k = 1$. Hence, the element matrix B^K of B in (16) is a column vector of size $N_k \times 1$ (Note that B is of size `(2NE+NT) × NT`), and

$$B^K_{j1} = b^K(\boldsymbol{\phi}_j, m_1) = \int_K m_1 \mathrm{div}\boldsymbol{\phi}_j \mathrm{d}x = \int_K \mathrm{div}\boldsymbol{\phi}_j \mathrm{d}x$$
$$= \int_{\partial K} \boldsymbol{\phi}_j \cdot \boldsymbol{n} \mathrm{d}s = \begin{cases} 1, & 1 \le j \le N_v, \\ 0, & j > N_v. \end{cases}$$

The sign of each entry is adjusted by using $b^K(\pm\boldsymbol{\phi}_i, \psi_j) = \pm b^K(\boldsymbol{\phi}_i, \psi_j)$.

```
    % Stiffness matrix B
    BK = zeros(NdofA,1); BK(1:Nv) = 1;
    BK = BK.*sgnelem;
    BK = reshape(BK',1,[]); % straighten as row vector for easy assembly
```

For $k = 1$, Q_h is piecewise constant, and hence has `NT` basis functions given by

$$\psi_l(x) = \begin{cases} 1, & x \in K_l, \\ 0, & \text{otherwise.} \end{cases}$$

Obviously, the vector in (15) is

$$d_l = \int_{K_l} \mathrm{d}x = |K_l|, \quad l = 1, \cdots, M = \texttt{NT}.$$

We compute the elliptic projections and provide the assembly index by looping over the elements. The assembly index for the matrices A and B is given by

```
    % ------ assembly index for bilinear forms --------
    NdofA = 2*Nv+1; NdofB = 1;
    indexDofA = [elem2edge{iel},elem2edge{iel}+NE,iel+2*NE];
    indexDofB = iel;
    iiA(idA+1:idA+NdofA^2) = reshape(repmat(indexDofA, NdofA,1), [], 1);
    jjA(idA+1:idA+NdofA^2) = repmat(indexDofA(:), NdofA, 1);
    ssA(idA+1:idA+NdofA^2) = AK(:);
    idA = idA + NdofA^2;
    iiB(idB+1:idB+NdofA*NdofB) = reshape(repmat(indexDofA, NdofB,1), [], 1);
    jjB(idB+1:idB+NdofA*NdofB) = repmat(indexDofB(:), NdofA, 1);
    ssB(idB+1:idB+NdofA*NdofB) = BK(:);
    idB = idB + NdofA*NdofB;
```

Afterwards, we can assemble the matrices A and B using the MATLAB functions `sparse`.

```
A = sparse(iiA,jjA,ssA,NNdofA,NNdofA);
B = sparse(iiB,jjB,ssB,NNdofA,NNdofB);
d = area;  % for Lagrange multiplier
```

The load vector

For the right-hand side, in view of (16), we only need to compute $\boldsymbol{f} = (-(f, \psi_l))$. For $k = 1$, the local vector is

$$\boldsymbol{f}_K = -(f, \psi_l)_K = -\int_K f \mathrm{d}x,$$

with the realization reading

```
% -------- load vector f ----------
fxy = @(x,y) pde.f([x,y]); % f(p) = f([x,y])
rhs = integralTri(fxy,3,nodeT,elemT); rhs = rhs';
fK = -rhs;
```

The assembly index for the vector $\boldsymbol{f}$ is given by

```
% ------- assembly index for rhs -------
elemb(ib+1:ib+NdofB) = indexDofB(:);
Fb(ib+1:ib+NdofB) = fK(:);
ib = ib + NdofB;
```

Then $\boldsymbol{f}$ can be assembled using the MATLAB functions `accumarray`.

```
FB = accumarray(elemb,ffB,[NNdofB 1]);
```

To sum up, the linear system without boundary conditions imposed is given by

```
%% Get block linear system
kk = sparse(NNdof+1,NNdof+1);  ff = zeros(NNdof+1,1);
kk(1:NNdofA,1:NNdofA) = A;
kk(1:NNdofA, (1:NNdofB)+NNdofA) = B;
kk((1:NNdofB)+NNdofA, 1:NNdofA) = B';
kk((1:NNdofB)+NNdofA, end) = d;
kk(end, (1:NNdofB)+NNdofA) = d';
ff((1:NNdofB)+NNdofA) = FB;
```

Note that the extra variable λ leads to `NNdof+1` rows or columns.

2.2.5 Applying the boundary conditions

The boundary condition $\boldsymbol{u} \cdot \boldsymbol{n} = g$ is now viewed as a Dirichlet condition for $\boldsymbol{u}$, which provides the values of the first two types of d.o.f.s, i.e.,

$$\chi_i(\boldsymbol{u}) = \int_{e_i} (\boldsymbol{u} \cdot \boldsymbol{n}) \mathrm{d}s, \quad i = 1, \cdots, N_v,$$
$$\chi_{N_v+i}(\boldsymbol{u}) = \int_{e_i} (\boldsymbol{u} \cdot \boldsymbol{n}) \frac{s - s_{e_i}}{h_{e_i}} \mathrm{d}s, \quad i = 1, \cdots, N_v$$

The Simpson rule is used to approximate the exact ones as follows.

```
%% Apply Dirichlet boundary conditions
% bdDof, freeDof
bdEdge = bdStruct.bdEdge;  bdEdgeIdx = bdStruct.bdEdgeIdx;
id = [bdEdgeIdx; bdEdgeIdx+NE];
isBdDof = false(NNdof+1,1); isBdDof(id) = true;
bdDof = (isBdDof); freeDof = (¬isBdDof);
% bdval
u = pde.uexact;
z1 = node(bdEdge(:,1),:); z2 = node(bdEdge(:,2),:); ze = (z1+z2)./2;
```

```
e = z1-z2;  % e = z2-z1
Ne = [-e(:,2),e(:,1)];
chi1 = 1/6*sum((u(z1)+4*u(ze)+u(z2)).*Ne,2); % u*n = g
chi2 = 1/12*sum((u(z2)-u(z1)).*Ne,2);
bdval = [chi1;chi2];
% sol
sol = zeros(NNdof+1,1);
sol(bdDof) = bdval;
ff = ff - kk*sol;

%% Set solver
sol(freeDof) = kk(freeDof,freeDof)\ff(freeDof);
uh = sol(1:NNdofA); % u = [u1,u2]
ph = sol(NNdofA+1:end-1);

%% Store information for computing errors
info.Ph = Ph; info.elem2dof = elem2dof;
info.kk = kk; %info.freeDof = freeDof;
```

To compute the discrete errors, one can store the matrix representation $\widehat{\mathbf{\Pi}}_*^K$ and the assembly index `elem2dof` in the function file.

2.2.6 Numerical example

In this paper, all examples are implemented in MATLAB R2019b. Our code is available from GitHub (https://github.com/Terenceyuyue/mVEM). The subroutine `Darcy_mixedVEM.m` is used to compute the numerical solutions and the test script `main_Darcy_mixedVEM.m` verifies the convergence rates. The PDE data is generated by `Darcydata.m`.

Example 2.1. *Let $\mathbb{K}$ be the identity matrix. The right-hand side f and the boundary conditions are chosen in such a way that the exact solution of (1) is $p(x,y) = \sin(\pi x)\cos(\pi y)$.*

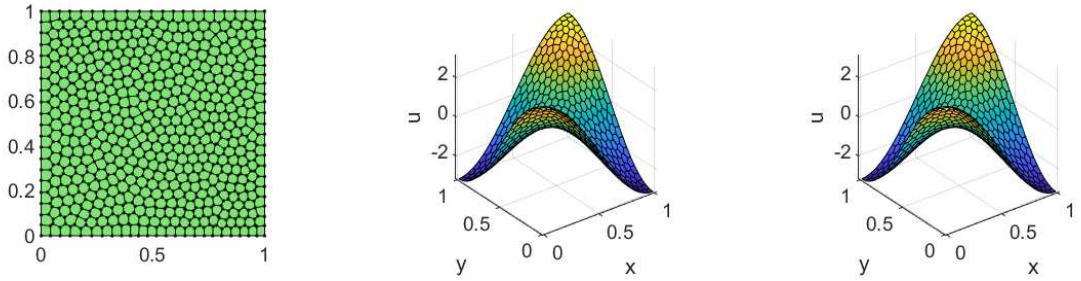


Fig. 4. Numerical and exact results for the Darcy problem

To test the accuracy of the proposed method we consider a sequence of polygonal meshes, which is a Centroidal Voronoi Tessellation of the unit square in 32, 64, 128, 256 and 512 polygons. These meshes are generated by the MATLAB toolbox - PolyMesher introduced in [45]. we report the nodal values of the exact solution and the piecewise elliptic projection $\widehat{\Pi}^K \boldsymbol{u}_h$ in Fig. 4. The convergence orders of the errors against the mesh size h are shown in Fig. 5. Generally speaking, h is proportional to $N^{-1/2}$, where N is the total number of elements in the mesh. The convergence rate with respect to h is estimated by assuming $\mathrm{ErrL2}(h) = ch^{\alpha}$, and by computing a least squares

fit to this log-linear relation. As observed from Fig. 5, the convergence rate of p is linear with respect to the L^2 norm, and the VEM ensures the quadratic convergence for $\boldsymbol{u}$ in the L^2 norm, which is consistent with the theoretical prediction in [8].

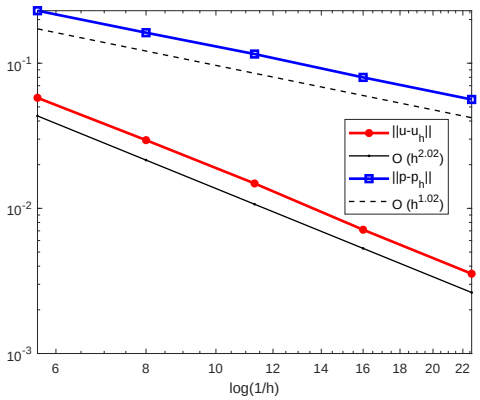


Fig. 5. The convergence rates for the Darcy problem

2.2.7 A lifting mixed virtual element method

We can obtain the second-order convergence of the variable p using the lifting technique. To do so, we modify the original virtual element space to the following lifting space

$$V_k(K) = \{\boldsymbol{v} \in H(\mathrm{div}; K) \cap H(\mathrm{rot}; K) : \boldsymbol{v} \cdot \boldsymbol{n}|_e \in \mathbb{P}_k(e), \\ \mathrm{div}\boldsymbol{v}|_K \in \mathbb{P}_k(K), \quad \mathrm{rot}\boldsymbol{v}|_K \in \mathbb{P}_{k-1}(K)\},$$

with the d.o.f.s given by

$$\int_e \boldsymbol{v} \cdot \boldsymbol{n} q \mathrm{d}s, \quad q \in \mathbb{M}_k(e), \quad e \subset \partial K,$$
$$\int_K \boldsymbol{v} \cdot \nabla q \mathrm{d}x, \quad q \in \mathbb{M}_k(K) \backslash \{1\},$$
$$\int_K \mathrm{rot}\boldsymbol{v} q \mathrm{d}x, \quad q \in \mathbb{M}_{k-1}(K).$$

In the lowest order case $k = 1$, the local d.o.f.s are arranged as

$$\chi_i(\boldsymbol{v}) = \int_{e_i} (\boldsymbol{v} \cdot \boldsymbol{n}) \mathrm{d}s, \quad i = 1, \cdots, N_v,$$
$$\chi_{N_v+i}(\boldsymbol{v}) = \int_{e_i} (\boldsymbol{v} \cdot \boldsymbol{n}) \frac{s - s_{e_i}}{h_{e_i}} \mathrm{d}s, \quad i = 1, \cdots, N_v,$$
$$\chi_{2N_v+1}(\boldsymbol{v}) = \int_K \boldsymbol{v} \cdot \nabla m_2 \mathrm{d}x = \frac{1}{h_K} \int_K \boldsymbol{v}_1 \mathrm{d}x,$$
$$\chi_{2N_v+2}(\boldsymbol{v}) = \int_K \boldsymbol{v} \cdot \nabla m_3 \mathrm{d}x = \frac{1}{h_K} \int_K \boldsymbol{v}_2 \mathrm{d}x,$$
$$\chi_{2N_v+3}(\boldsymbol{v}) = \int_K \mathrm{rot}\boldsymbol{v} \mathrm{d}x.$$

In this case, Q_h will be replaced by the piecewise linear space.

We repeat the test in Example 2.1. The subroutine `Darcy_LiftingmixedVEM.m` is used to compute the numerical solutions and the test script `main_Darcy_LiftingmixedVEM.m` verifies the

convergence rates. The exact and numerical solutions for the variable p are shown in Fig. 6. The corresponding convergence rates are displayed in Fig. 7, from which we observe the second-order convergence for both variables.

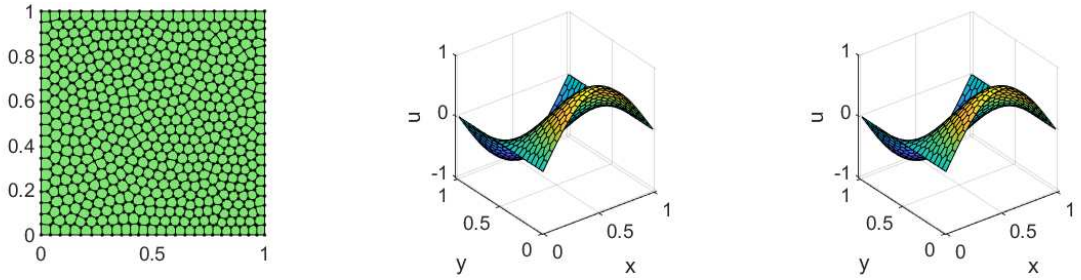


Fig. 6. p_h and p for the lifting VEM of the Darcy problem

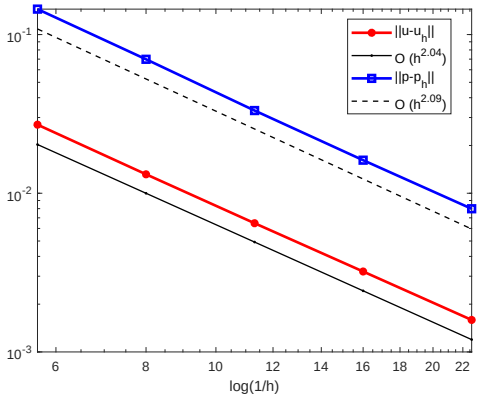


Fig. 7. The convergence rates for the lifting VEM of the Darcy problem

3 Polygonal mesh generation and refinement

The polygonal meshes can be generated by using the MATLAB toolbox - PolyMesher introduced in [45]. We provide a modified version in our package and give a very detailed description in the document. The test script is `meshfun.m` and we present a mesh for a complex geometry in Fig. 8.

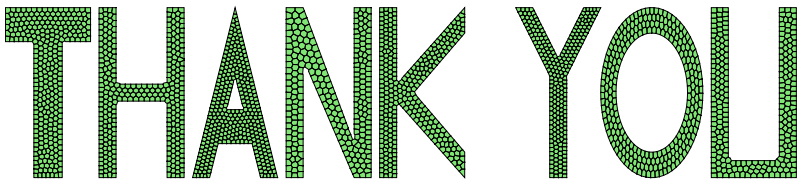
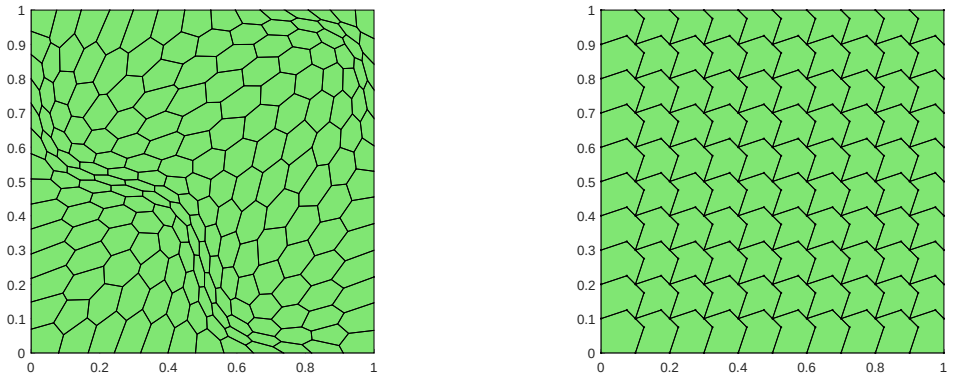


Fig. 8. A polygonal mesh generated by PolyMesher for a complex geometry.

We present some basic functions to show the polygonal meshes, including marking of the nodes, elements and (boundary) edges, see `showmesh.m`, `findnode.m`, `findelem.m` and `findedge.m`. For the convenience of the computation, some auxiliary mesh data are introduced (see Subsection 2.2.2).

The idea stems from the treatment of triangulation in *i*FEM, which is generalized to polygonal meshes with certain modifications. We also provide a boundary setting function to identify the Neumann and Dirichlet boundaries (See `setboundary.m`).



(a) (b)

Fig. 9. A distorted mesh and a non-convex octagonal mesh.

The routine `distortionmesh.m` is used to generate a distorted mesh. Let (ξ, η) be the coordinates on the original mesh. The nodes of the distorted mesh are obtained by the following transformation

$$x = \xi + t_c \sin(2\pi\xi)\sin(2\pi\eta), \quad y = \eta + t_c \sin(2\pi\xi)\sin(2\pi\eta),$$

where (x, y) is the coordinate of new nodal points; t_c, taken as 0.1 in the computation, is the distortion parameter. Such an example is displayed in Fig. 9(a). In the literature of VEMs (see [25] for example), one often finds the test for the non-convex mesh in Fig. 9(b), which is generated by `nonConvexMesh.m` in our package.

The polygonal meshes can also be obtained from a Delaunay triangulation by establishing its dual mesh. The implementation is given in `dualMesh.m`. Several examples are given in `main_dualMesh.m` as shown in Fig. 10.

Due to the large flexibility of the meshes, researchers have focused on the a posterior error analysis of the VEMs and made some progress in recent years [18–20,24,29]. We present an efficient implementation of the mesh refinement for polygonal meshes [52] (see `PolyMeshRefine.m`). To the best of our knowledge, this is the first publicly available implementation of the polygonal mesh refinement algorithms. We divide elements by connecting the midpoint of each edge to its barycenter, which may be the most natural partition frequently used in VEM papers. To remove small edges, some additional neighboring polygons of the marked elements are included in the refinement set by requiring the one-hanging-node rule: limit the mesh to have just one hanging node per edge. The current implementation requires that the barycenter of each element is an interior point.

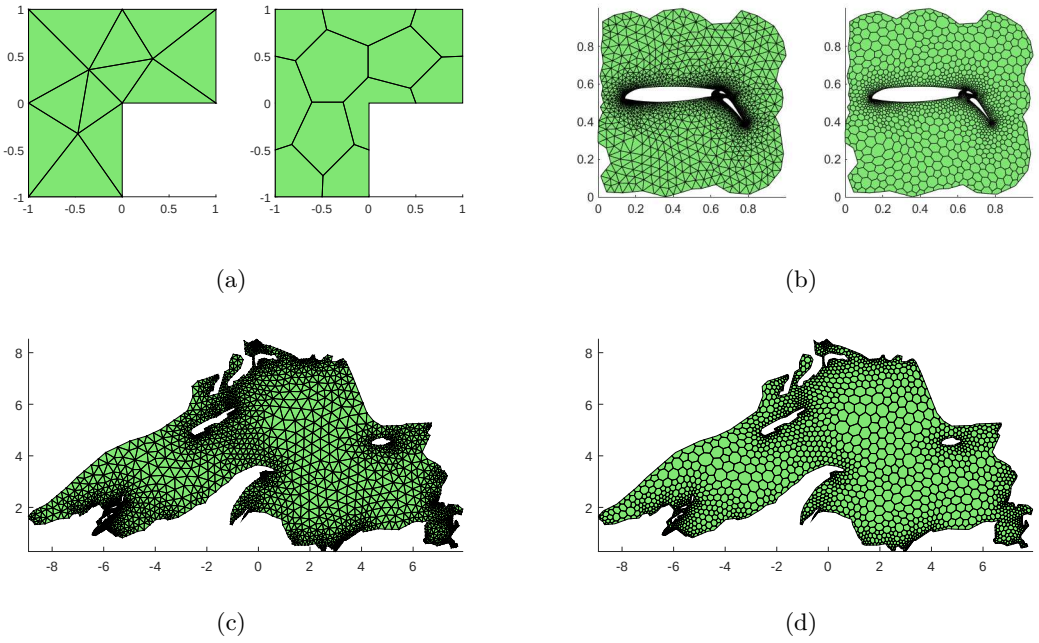


Fig. 10. Dual meshes generated by `dualMesh.m`.

4 Poisson equation

In this section we focus on the virtual element methods for the reaction-diffusion problem

$$\begin{cases} -\Delta u + \alpha u = f & \text{in } \Omega, \\ u = 0 & \text{on } \partial\Omega, \end{cases} \tag{21}$$

where α is a nonnegative constant and Ω is a polygonal domain.

4.1 Conforming VEMs

Consider the first virtual element space [7]

$$V_k(K) := \left\{ v \in H^1(K) : \Delta v \in \mathbb{P}_{k-2}(K) \ \text{ in } \ K, \quad v|_{\partial K} \in \mathbb{B}_k(\partial K) \right\}, \tag{22}$$

where

$$\mathbb{B}_k(\partial K) := \{ v \in C(\partial K) : v|_e \in \mathbb{P}_k(e), \quad e \subset \partial K \}.$$

Equipped with the d.o.f.s in [7], a computable approximate bilinear form can be constructed by using the elliptic projector Π_k^∇. To ensure the accuracy and the well-posedness of the discrete method, a natural candidate to approximate the reaction term $(u, v)_K$ in the local bilinear form is $(\Pi_k^0 u, \Pi_k^0 v)_K$, where Π_k^0 is the L^2 projection onto $\mathbb{P}_k(K)$ and $(\cdot, \cdot)_K$ the usual $L^2(K)$ inner product. Nevertheless, Π_k^0 can not be computed in terms of d.o.f.s attached to $V_k(K)$. Therefore, Ahmad *et al.* [2] modified the VEM space (22) to a local enhancement space

$$W_k(K) = \left\{ w \in \widetilde{V}_k(K) : (w - \Pi_k^\nabla w, q)_K = 0, \quad q \in \mathbb{M}_k(K) \backslash \mathbb{M}_{k-2}(K) \right\}, \tag{23}$$

where the lifting space is

$$\widetilde{V}_k(K) := \left\{ v \in H^1(K) : v|_{\partial K} \in \mathbb{B}_k(\partial K), \quad \Delta v \in \mathbb{P}_k(K) \ \text{ in } K \right\}.$$

In this space, the operator Π_k^0 can be easily computed using Π_k^∇ and the local d.o.f.s related to $W_k(K)$:

- χ_a: the values at the vertices of K,

$$\chi_i(v) = v(a_i), \quad a_i \text{ is a vertex of } K.$$

- χ_e^{k-2}: the moments on edges up to degree $k-2$,

$$\chi_e(v) = |e|^{-1}(m_e, v)_e, \quad m_e \in \mathbb{M}_{k-2}(e), \quad e \subset \partial K.$$

- χ_K^{k-2}: the moments on element K up to degree $k-2$,

$$\chi_K(v) = |K|^{-1}(m_K, v)_K, \quad m_K \in \mathbb{M}_{k-2}(K).$$

In what follows, denote by $W_{k,h}$ a finite dimensional subspace of V, produced by combining all $W_k(K)$ for $K \in \mathcal{T}_h$ in a standard way. We define a local H^1-elliptic projection operator $\Pi_k^\nabla : H^1(K) \to \mathbb{P}_k(K)$ by the relations

$$(\nabla \Pi_k^\nabla v, \nabla p)_K = (\nabla v, \nabla p)_K, \quad p \in \mathbb{P}_k(K). \tag{24}$$

Since $(\nabla\cdot, \nabla\cdot)$ is only semi-positive definite, the constraints

$$\int_{\partial K} v \mathrm{d}s = \int_{\partial K} \Pi_1^\nabla v \mathrm{d}s, \quad k = 1; \qquad \int_K v \mathrm{d}x = \int_K \Pi_k^\nabla v \mathrm{d}x, \quad k \geq 2$$

should be imposed. The VEM is to find $u_h \in W_{k,h}$ such that

$$a_h^c(u_h, v_h) = \langle f_h, v_h \rangle, \quad v_h \in W_{k,h}, \tag{25}$$

where

$$a_h^c(u_h, v_h) := \sum_{K \in \mathcal{T}_h} a_h^{c,K}(u_h, v_h)$$

and $\langle \cdot, \cdot \rangle$ is the duality pairing between $W'_{k,h}$ and $W_{k,h}$. The local approximate bilinear form is

$$a_h^{c,K}(v, w) = a^K(\Pi_k^\nabla v, \Pi_k^\nabla w) + \alpha(\Pi_k^0 v, \Pi_k^0 w)_K + S^K(v - \Pi_k^\nabla v, w - \Pi_k^\nabla w)$$

where $a^K(v, w) = (\nabla v, \nabla w)_K$ and the stabilization term is realized as

$$S^K(v - \Pi_k^\nabla v, w - \Pi_k^\nabla w) := (1 + \alpha h_K^2)\boldsymbol{\chi}(v - \Pi_k^\nabla v) \cdot \boldsymbol{\chi}(w - \Pi_k^\nabla w),$$

where $\boldsymbol{\chi}$ is referred to as the d.o.f vector of a VEM function on K.

Let $\phi^T = (\phi_1, \phi_2, \cdots, \phi_{N_k})$ be the basis of $W_k(K)$ and let $m^T = (m_1, m_2, \cdots, m_{N_m})$ be the scaled monomials. Then the transition matrix D can be defined through $m^T = \phi^T D$. The matrix expressions of $\Pi_k^\nabla \phi^T$ in the basis ϕ^T and m^T are defined by

$$\Pi_k^\nabla \phi^T = \phi^T \mathbf{\Pi}_k^\nabla, \quad \Pi_k^\nabla \phi^T = m^T \mathbf{\Pi}_{k*}^\nabla.$$

One easily finds that $\mathbf{\Pi}_k^\nabla = D\mathbf{\Pi}_{k*}^\nabla$. In vector form, the definition of (24) can be rewritten as

$$\begin{cases} a^K(m, \Pi_k^\nabla \phi^T) = a^K(m, \phi^T), \\ P_0(\Pi_k^\nabla \phi^T) = P_0(\phi^T), \end{cases}$$

where

$$P_0(v) = \int_{\partial K} v \mathrm{d}s, \quad k = 1; \qquad P_0(v) = \int_K v \mathrm{d}x, \quad k \geq 2.$$

Define

$$G = a^K(m, m^T), \quad B = a^K(m, \phi^T). \tag{26}$$

Then

$$\begin{cases} G\mathbf{\Pi}_{k*}^\nabla = B, \\ P_0(m^T)\mathbf{\Pi}_{k*}^\nabla = P_0(\phi^T) \end{cases} \quad \text{or} \quad \tilde{G}\mathbf{\Pi}_{k*}^\nabla = \tilde{B}.$$

Note that one always has the consistency relations $G = BD$ and $\tilde{G} = \tilde{B}D$.

We remark that in the implementation of the VEM, the most involved step is to compute the boundary part of B resulted from the integration by parts:

$$B = \int_K \nabla m \cdot \nabla \phi^T \mathrm{d}x = -\int_K \Delta m \cdot \phi^T \mathrm{d}x + \sum_{e \subset \partial K} \int_e (\nabla m \cdot \boldsymbol{n}_e) \phi^T \mathrm{d}s. \tag{27}$$

For the conforming VEMs, however, we can compute the second term of (27) by using the assembling technique for finite element methods since the VEM function restriction to the boundary ∂K is piecewise polynomial. For example, the computation for $k = 2$ reads

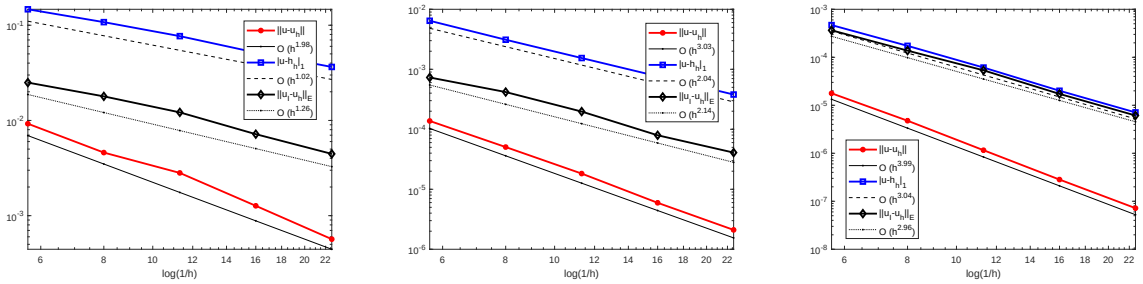
```
% second term
I2 = zeros(Nm, Ndof);
v1 = 1:Nv; v2 = [2:Nv,1];
elem1 = [v1(:), v2(:), v1(:)+Nv];
for im = 1:Nm
    gradmc = Gradmc{im};
    F1 = 1/6*sum(gradmc(x(v1), y(v1)).*Ne, 2);
    F2 = 1/6*sum(gradmc(x(v2), y(v2)).*Ne, 2);
    F3 = 4/6*sum(gradmc(xe, ye).*Ne, 2);
    F = [F1, F2, F3];
    I2(im, :) = accumarray(elem1(:), F(:), [Ndof 1]);
end
```

Example 4.1. *Let $\alpha = 1$ and $\Omega = (0,1)^2$. The Neumann boundary condition is imposed on $x = 0$ and $x = 1$ with the exact solution given by $u(x,y) = \sin(2x + 0.5)\cos(y + 0.3) + \log(1 + xy)$.*

The test script is `main_PoissonVEMk3.m` for $k = 3$. The optimal rate of convergence of the H^1-norm (3rd order), L^2-norm (4th order) and energy norm (3rd order) is observed for $k = 3$ from Fig. 11.

Example 4.2. *In this example, the exact solution is $u(x,y) = y^2 \sin(\pi x)$. The domain Ω is taken as a unit disk.*

The Neumann boundary condition is imposed on the boundary of the upper semicircle. The nodal values are displayed in Fig. 12.



(a) $k = 1$ (b) $k = 2$ (c) $k = 3$

Fig. 11. Convergence rates for Example 4.1.

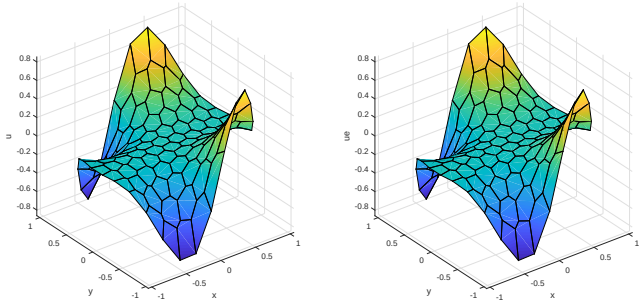


Fig. 12. Numerical and exact solutions for Example 4.2 ($k = 1$)

4.2 Nonconforming VEMs

4.2.1 The standard treatment of the domain boundary

Now we consider the nonconforming VEMs proposed in [6] for the problem (21) with $\alpha = 0$. The local nonconforming virtual element space is defined by

$$V_k^{nc}(K) = \left\{ v \in H^1(K) : \Delta v \in \mathbb{P}_{k-2}(K), \quad \partial_{\boldsymbol{n}} v|_e \in \mathbb{P}_{k-1}(e),\ e \subset \partial K \right\}, \quad k \geq 1,$$

and the d.o.f.s can be chosen as

- $\boldsymbol{\chi}_{\partial K}(v)$: the moments on edges up to degree $k-1$,

$$\chi_e(v) = |e|^{-1}(m_e, v)_e, \quad m_e \in \mathbb{M}_{k-1}(e), \quad e \subset \partial K.$$

- $\boldsymbol{\chi}_K(v)$: the moments on K up to degree $k-2$,

$$\chi_K(v) = |K|^{-1}(m_K, v)_K, \quad m_K \in \mathbb{M}_{k-2}(K).$$

The elliptic projection and the discrete problem can be constructed in the same way as before. We briefly discuss the implementation of $k = 1$. In this case, the first term for the matrix B in (27) vanishes and the second term is obtained from the Kronecher's property $\chi_i(\phi_j) = \sigma_{ij}$, which is realized as follows.

```
1 % B
```

```
B = zeros(Nm,Nv);
for i = 1:Nv % loop of edges
    gi = sum(Gradm.*repmat(Ne(i,:),3,1), 2); % gradm*Ne
    B(:,i) = gi;
end
```

For completeness, we also present the application of the Neumann boundary conditions. Let e be a boundary edge of the Neumann boundary Γ_N. Then the local boundary term is approximated as

$$F_e = \int_e \Pi^0_{0,e} g_N \phi_e \mathrm{d}s,$$

where $g_N = \partial_{\boldsymbol{n}} u = \nabla u \cdot \boldsymbol{n}_e$, $\Pi^0_{0,e}$ is the L^2 projection on e, and ϕ_e is the local basis function associated with the d.o.f $\chi_e(v) = |e|^{-1}(m_e, v)_e$. For problems with known explicit solutions, we provide the gradient $g_N = \nabla u$ in the PDE data instead and compute the true g_N in the M-file. Obviously, the L^2 projection $\Pi^0_{0,e} g_N$ can be computed by the mid-point formula or the trapezoidal rule. The computation reads

```
%% Assemble Neumann boundary conditions
bdEdgeN = bdStruct.bdEdgeN;  bdEdgeIdxN = bdStruct.bdEdgeIdxN;
if ~isempty(bdEdgeN)
    g_N = pde.Du;
    z1 = node(bdEdgeN(:,1),:); z2 = node(bdEdgeN(:,2),:);
    e = z1-z2;  % e = z2-z1
    Ne = [-e(:,2),e(:,1)]; % scaled ne
    F1 = sum(Ne.*g_N(z1),2);
    F2 = sum(Ne.*g_N(z2),2);
    FN = (F1+F2)/2;
    ff = ff + accumarray(bdEdgeIdxN(:), FN(:),[NNdof 1]);
end
```

Note that the structure data `bdStruct` stores all necessary information of boundary edges, which is obtained by using the subroutine `setboundary.m`. In the above code, `bdEdgeN` gives the data structure `bdEdge` in Subsection 2.2.2 for Neumann edges and `bdEdgeIdxN` provides their indices in the data structure `edge`.

We still consider Example 4.1 and display the convergence rates in the discrete L^2 and H^1 norms in Fig. 13. The optimal rate of convergence is observed for both norms and the test script is `main_PoissonVEM_NC.m`.

4.2.2 A continuous treatment of the domain boundary

In some cases it may be not convenient to deal with the moments on the domain boundary $\partial\Omega$. To do so, we can modify the corresponding local virtual element spaces to the following one

$$V_k^{ncb}(K) = \Big\{ v \in H^1(K) : \Delta v \in \mathbb{P}_{k-2}(K), \quad \partial_{\boldsymbol{n}} v|_e \in \mathbb{P}_{k-1}(e), \quad e \text{ is an interior edge}$$
$$v|_e \in \mathbb{P}_k(e), \quad e \text{ is a boudary edge}, \quad e \subset \partial K \Big\}.$$

We repeat the test in Fig. 13 and the discrete L^2 and H^1 errors are listed in Tab. 2. Please refer to `main_PoissonVEM_NCb.m` for the test script.

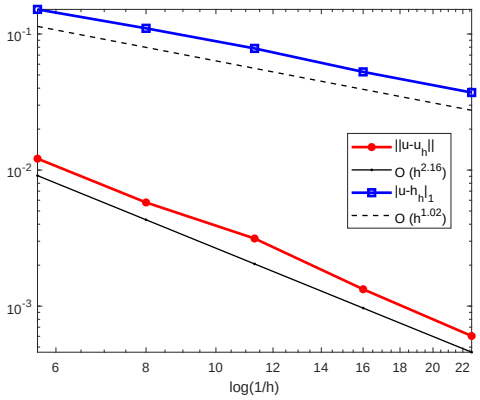


Fig. 13. Convergence rates of the Poisson equation for the nonconforming VEM ($k = 1$)

Tab. 2. The discrete errors for the modified nonconforming VEM

♯Dof	h	ErrL2	ErrH1
97	1.768e-01	1.17632e-02	1.51836e-01
193	1.250e-01	5.37635e-03	1.10077e-01
383	8.839e-02	3.14973e-03	7.82566e-02
760	6.250e-02	1.43956e-03	5.26535e-02
1522	4.419e-02	6.48454e-04	3.71405e-02

5 Linear elasticity problems

The linear elasticity problem is

$$\begin{cases} -\mathrm{div}\boldsymbol{\sigma} = \boldsymbol{f} & \text{in} \quad \Omega, \\ \boldsymbol{u} = \boldsymbol{0} & \text{on} \quad \Gamma_0, \\ \boldsymbol{\sigma}\boldsymbol{n} = \boldsymbol{g} & \text{on} \quad \Gamma_1, \end{cases} \tag{28}$$

where $\boldsymbol{n} = (n_1, n_2)^T$ denotes the outer unit vector normal to $\partial\Omega$. The constitutive relation for linear elasticity is

$$\boldsymbol{\sigma}(\boldsymbol{u}) = 2\mu\boldsymbol{\varepsilon}(\boldsymbol{u}) + \lambda(\mathrm{div}\boldsymbol{u})\boldsymbol{I},$$

where $\boldsymbol{\sigma} = (\sigma_{ij})$ and $\boldsymbol{\varepsilon} = (\varepsilon_{ij})$ are the second order stress and strain tensors, respectively, satisfying $\varepsilon_{ij} = \frac{1}{2}(\partial_i u_j + \partial_j u_i)$, λ and μ are the Lamé constants, $\boldsymbol{I}$ is the identity matrix, and $\mathrm{div}\boldsymbol{u} = \partial_1 u_1 + \partial_2 u_2$.

5.1 Conforming VEMs

5.1.1 The displacement type

The equilibrium equation in (28) can also be written in the form

$$-\mu\Delta\boldsymbol{u} - (\lambda + \mu)\mathrm{grad}(\mathrm{div}\boldsymbol{u}) = \boldsymbol{f} \quad \text{in} \quad \Omega, \tag{29}$$

which is referred to as the displacement type or the Navier type in what follows. In this case, we only consider $\Gamma_0 = \Gamma := \partial\Omega$. The first term $\Delta \boldsymbol{u}$ can be treated as the vector case of the Poisson equation, which is clearly observed in the implementation.

The continuous variational problem is to find $\boldsymbol{u} \in \boldsymbol{V} := \boldsymbol{H}_0^1(\Omega)$ such that

$$a(\boldsymbol{u}, \boldsymbol{v}) = (\boldsymbol{f}, \boldsymbol{v}), \quad \boldsymbol{v} \in \boldsymbol{V},$$

where the bilinear form restriction to K is

$$\begin{aligned} a^K(\boldsymbol{u}, \boldsymbol{v}) &= \mu \int_K \nabla \boldsymbol{u} \cdot \nabla \boldsymbol{v} \mathrm{d}x + (\lambda + \mu) \int_K (\mathrm{div}\boldsymbol{u})(\mathrm{div}\boldsymbol{v}) \mathrm{d}x \\ &=: \mu a_\nabla^K(\boldsymbol{u}, \boldsymbol{v}) + (\lambda + \mu) a_{\mathrm{div}}^K(\boldsymbol{u}, \boldsymbol{v}) \end{aligned}$$

with

$$a_\nabla^K(\boldsymbol{u}, \boldsymbol{v}) = \int_K \nabla \boldsymbol{u} \cdot \nabla \boldsymbol{v} \mathrm{d}x, \quad a_{\mathrm{div}}^K(\boldsymbol{u}, \boldsymbol{v}) = \int_K (\mathrm{div}\boldsymbol{u})(\mathrm{div}\boldsymbol{v}) \mathrm{d}x.$$

The local virtual element space $\boldsymbol{V}_k(K)$ can be simply taken as the tensor-product of $V_k(K)$ defined in (22), i.e., $\boldsymbol{V}_k(K) = (V_k(K))^2$. Let $\Pi^\nabla : \boldsymbol{V}_k(K) \to (\mathbb{P}_k(K))^2$ be the elliptic projector induced by a_∇^K. Then the approximate bilinear form can be split as

$$a_h^K(\boldsymbol{u}, \boldsymbol{v}) = \mu a_{h,\nabla}^K(\boldsymbol{u}, \boldsymbol{v}) + (\lambda + \mu) a_{h,\mathrm{div}}^K(\boldsymbol{u}, \boldsymbol{v}),$$

where

$$\begin{aligned} a_{h,\nabla}^K(\boldsymbol{u}, \boldsymbol{v}) &= a_\nabla^K(\Pi^\nabla \boldsymbol{u}, \Pi^\nabla \boldsymbol{v}) + S^K(\boldsymbol{u} - \Pi^\nabla \boldsymbol{u}, \boldsymbol{v} - \Pi^\nabla \boldsymbol{v}), \\ a_{h,\mathrm{div}}^K(\boldsymbol{u}, \boldsymbol{v}) &= (\Pi_{k-1}^0 \mathrm{div}\boldsymbol{u}, \Pi_{k-1}^0 \mathrm{div}\boldsymbol{v})_K, \end{aligned}$$

and the stabilization term for $k = 1$ is given by

$$S^K(\boldsymbol{u}, \boldsymbol{v}) = \sum_{i=1}^{N_v} \chi_i(\boldsymbol{u}) \cdot \chi_i(\boldsymbol{v}), \quad \chi_i(\boldsymbol{u}) = [\chi_i(u_1), \chi_i(u_2)]^T.$$

The discrete problem is: Find $\boldsymbol{u}_h \in \boldsymbol{V}_h$ such that

$$a_h(\boldsymbol{u}_h, \boldsymbol{v}_h) = \langle \boldsymbol{f}_h, \boldsymbol{v}_h \rangle, \quad \boldsymbol{v}_h \in \boldsymbol{V}_h, \tag{30}$$

where

$$a_h(\boldsymbol{u}_h, \boldsymbol{v}_h) = \sum_{K \in \mathcal{T}_h} a_h^K(\boldsymbol{u}_h, \boldsymbol{v}_h)$$

and the approximation of the right hand side is given by [54]

$$\langle \boldsymbol{f}_h, \boldsymbol{v}_h \rangle = (\boldsymbol{f}, P_h \boldsymbol{v}_h), \tag{31}$$

where

$$P_h \boldsymbol{v}_h|_K = \frac{1}{|\partial K|} \int_{\partial K} \boldsymbol{v}_h \mathrm{d}s.$$

Let $\phi_1, \cdots, \phi_N$ be the basis functions of the scalar space $V_k(K)$. Then the basis functions of the vector space $\boldsymbol{V}_k(K)$ can be defined by

$$\overline{\phi}_1, \cdots, \overline{\phi}_N, \underline{\phi}_1, \cdots, \underline{\phi}_N,$$

where

$$\overline{\phi}_i = \begin{bmatrix} \phi_i \\ 0 \end{bmatrix}, \qquad \underline{\phi}_i = \begin{bmatrix} 0 \\ \phi_i \end{bmatrix}, \quad i = 1, \cdots, N.$$

Similarly, we introduce the notation

$$\overline{m}_\alpha = \begin{bmatrix} m_\alpha \\ 0 \end{bmatrix}, \qquad \underline{m}_\alpha = \begin{bmatrix} 0 \\ m_\alpha \end{bmatrix}, \quad \alpha = 1, \cdots, N_m. \tag{32}$$

These vector functions will be written in a compact form as

$$\boldsymbol{m}^T = [\overline{m}_1, \cdots, \overline{m}_{N_m}, \underline{m}_1, \cdots, \underline{m}_{N_m}] =: [\overline{m}^T, \underline{m}^T],$$

$$\boldsymbol{\phi}^T = [\overline{\phi}_1, \cdots, \overline{\phi}_{N_k}, \underline{\phi}_1, \cdots, \underline{\phi}_{N_k}] =: [\overline{\phi}^T, \underline{\phi}^T].$$

We introduce the transition matrix $\boldsymbol{D}$ such that $\boldsymbol{m}^T = \boldsymbol{\phi}^T \boldsymbol{D}$. One easily finds that $\boldsymbol{D} = \mathrm{diag}(D, D)$, where D is the transition matrix for the scalar case, i.e., $m^T = \phi^T D$. This block structure is also valid for the elliptic projection matrices. That is,

$$\boldsymbol{G} = \mathrm{diag}(G, G), \quad \boldsymbol{B} = \mathrm{diag}(B, B), \quad \boldsymbol{G} = \boldsymbol{B}\boldsymbol{D},$$

where G and B are the same ones given in (26).

In the following, we consider the matrix expression of the L^2 projector $\Pi^0_{k-1}\mathrm{div}$ satisfying

$$\int_K \Pi^0_{k-1}(\mathrm{div}\boldsymbol{v}) p \mathrm{d}x = \int_K (\mathrm{div}\boldsymbol{v}) p \mathrm{d}x, \quad p \in \mathbb{P}_{k-1}(K).$$

The vector form can be written as

$$\int_K m^0 \Pi^0_{k-1}(\mathrm{div}\boldsymbol{\phi}^T)\mathrm{d}x = \int_K m^0(\mathrm{div}\boldsymbol{\phi}^T)\mathrm{d}x,$$

where m^0 is the vector consisting of the scaled monomials of order $\leq k-1$. Let $\boldsymbol{\Pi}_{0*}$ be the matrix expression of $\Pi^0_{k-1}(\mathrm{div}\boldsymbol{\phi}^T)$ in the basis $(m^0)^T$, i.e.,

$$\Pi^0_{k-1}(\mathrm{div}\boldsymbol{\phi}^T) = (m^0)^T \boldsymbol{\Pi}_{0*}.$$

Then $H_0 \boldsymbol{\Pi}_{0*} = C_0$, where

$$H_0 = \int_K m^0 (m^0)^T \mathrm{d}x, \qquad C_0 = \int_K m^0(\mathrm{div}\boldsymbol{\phi}^T)\mathrm{d}x.$$

For $k = 1$, one has $m^0 = m_1 = 1$ and

$$C_0 = \int_K m^0(\mathrm{div}\boldsymbol{\phi}^T)\mathrm{d}x = \int_K \mathrm{div}\boldsymbol{\phi}^T \mathrm{d}x = \int_{\partial K} \boldsymbol{\phi}^T \cdot \boldsymbol{n} \mathrm{d}s,$$

where

$$\int_{\partial K} \boldsymbol{\phi}^T \cdot \boldsymbol{n} \mathrm{d}s = \int_{\partial K} [\overline{\phi}^T \cdot \boldsymbol{n}, \underline{\phi}^T \cdot \boldsymbol{n}] \mathrm{d}s = \int_{\partial K} [\phi^T \cdot n_x, \phi^T \cdot n_y] \mathrm{d}s. \tag{33}$$

We remark that (33) can be assembled along the boundary ∂K using the technique in finite element methods.

```
v1 = 1:Nv;  v2 = [2:Nv,1];
H0 = area(iel);
C0 = zeros(1,2*Nv);
F = 1/2*[(1*Ne); (1*Ne)]; % [he*n1, he*n2]
C0(:) = accumarray([elem1(:);elem1(:)+Nv], F(:), [2*Nv 1]);
```

Example 5.1. *We consider a typical example to check the locking-free property of the proposed method. The right-hand side* $\boldsymbol{f}$ *and the boundary conditions are chosen in such a way that the exact solution is*

$$\boldsymbol{u}(x,y) = \begin{bmatrix} (-1+\cos 2\pi x)\sin 2\pi y \\ -(-1+\cos 2\pi y)\sin 2\pi x \end{bmatrix} + \frac{1}{1+\lambda}\sin \pi x \sin \pi y \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

One easily finds that the proposed VEM is exactly the $\mathbb{P}_1$-Lagrange element when the polygonal mesh degenerates into a triangulation, in which case the method cannot get a uniform convergence with respect to the Lamé constant λ. However, for general polygonal meshes, the method seems to be robust with respect to λ (see Fig. 14), and the optimal rates of convergence are achieved in the nearly incompressible case as shown in Fig. 15, although we have not justified it in a theoretical way or found a counter example. Please refer to `main_elasticityVEM_Navier.m` for the test script.

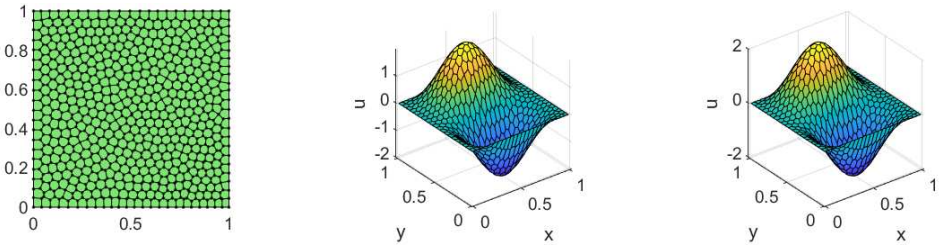


Fig. 14. The exact and numerical nodal values for the linear elasticity problem of displacement type with $\lambda = 10^8$ and $\mu = 1$.

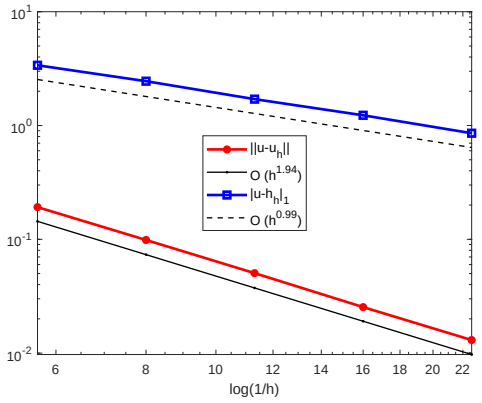


Fig. 15. The performance of the VEM for the linear elasticity problem of displacement type with $\lambda = 10^8$ and $\mu = 1$.

5.1.2 The tensor type

Let $\boldsymbol{f} \in \boldsymbol{L}^2(\Omega)$. For simplicity, we still consider $\Gamma_0 = \partial\Omega$. The tensor-type variational formulation of (28) is to find $\boldsymbol{u} \in \boldsymbol{V}$ such that

$$a(\boldsymbol{u},\boldsymbol{v}) = (\boldsymbol{f},\boldsymbol{v}), \quad \boldsymbol{v} \in \boldsymbol{V}, \tag{34}$$

where

$$a(\boldsymbol{u},\boldsymbol{v}) = 2\mu(\boldsymbol{\varepsilon}(\boldsymbol{u}),\boldsymbol{\varepsilon}(\boldsymbol{v})) + \lambda(\mathrm{div}\boldsymbol{u},\mathrm{div}\boldsymbol{v}), \tag{35}$$

and

$$(\boldsymbol{f},\boldsymbol{v})=\int_{\Omega}\boldsymbol{f}\boldsymbol{v}\mathrm{d}x.$$

For the ease of the presentation, we introduce the following notation

$$a_{\mu}(\boldsymbol{u},\boldsymbol{v})=(\boldsymbol{\varepsilon}(\boldsymbol{u}),\boldsymbol{\varepsilon}(\boldsymbol{v})),\qquad a_{\lambda}(\boldsymbol{u},\boldsymbol{v})=(\mathrm{div}\boldsymbol{u},\mathrm{div}\boldsymbol{v}).$$

The conforming VEM for (34) is proposed in [10], where a locking-free analysis is carried out for the virtual element spaces of order $k\geq 2$. The order requirement is to ensure the so-called discrete inf-sup condition and the optimal convergence.

Introduce the elliptic projection

$$\Pi^{a}:\ \boldsymbol{V}_k(K)\to(\mathbb{P}_k(K))^2,\quad \boldsymbol{v}\mapsto\Pi^{a}\boldsymbol{v},$$

satisfying

$$a_{\mu}^{K}(\Pi^{a}\boldsymbol{v},\boldsymbol{p})=a_{\mu}^{K}(\boldsymbol{v},\boldsymbol{p}),\quad \boldsymbol{p}\in(\mathbb{P}_k(K))^2. \tag{36}$$

Note that $\Pi^{a}\boldsymbol{v}$ is unique up to an additive vector in

$$K_0:=RM=\mathrm{span}\left\{\begin{bmatrix}1\\0\end{bmatrix},\quad\begin{bmatrix}0\\1\end{bmatrix},\quad\begin{bmatrix}-y\\x\end{bmatrix}\right\}.$$

For this reason, we can impose one of the following constraints:

Choice 1:

$$\sum_{i=1}^{N_v}(\Pi^{a}\boldsymbol{v}(z_i),\boldsymbol{p}(z_i))=\sum_{i=1}^{N_v}(\boldsymbol{v}(z_i),\boldsymbol{p}(z_i)),\quad \boldsymbol{p}\in K_0, \tag{37}$$

Choice 2:

$$\int_{\partial K}(\Pi^{a}\boldsymbol{v})\cdot\boldsymbol{p}\mathrm{d}s=\int_{\partial K}\boldsymbol{v}\cdot\boldsymbol{p}\mathrm{d}s,\quad \boldsymbol{p}\in K_0. \tag{38}$$

Choice 3:

$$\begin{aligned}\int_{K}\mathrm{rot}\ \Pi_K^1\boldsymbol{v}\mathrm{d}x&=\int_{K}\mathrm{rot}\boldsymbol{v}\mathrm{d}x,\\ \int_{\partial K}\Pi_K^1\boldsymbol{v}\mathrm{d}s&=\int_{\partial K}\boldsymbol{v}\mathrm{d}s,\end{aligned} \tag{39}$$

where $\mathrm{rot}\boldsymbol{v}=\nabla\times\boldsymbol{v}=\partial_1 v_2-\partial_2 v_1$. An integration by parts gives

$$\int_{K}\mathrm{rot}\boldsymbol{v}\mathrm{d}x=\int_{\partial K}\boldsymbol{v}\cdot\boldsymbol{t}_K\mathrm{d}s,$$

where $\boldsymbol{t}_K=(-n_2,n_1)^T$ is the anti-clockwise tangential along ∂K.

It is evident that all the three choices can be computed by using the given d.o.f.s.

The discrete problem is the same as the one in (30) with the local approximate bilinear form replaced by

$$a_h^K(\boldsymbol{v},\boldsymbol{w})=2\mu a_{\mu,h}^K(\boldsymbol{v},\boldsymbol{w})+\lambda a_{\lambda,h}^K(\boldsymbol{v},\boldsymbol{w}),$$

where

$$a_{\mu,h}^K(\boldsymbol{v},\boldsymbol{w}) = a_\mu^K(\Pi^a\boldsymbol{v},\Pi^a\boldsymbol{w}) + S^K(\boldsymbol{v}-\Pi^a\boldsymbol{v},\boldsymbol{w}-\Pi^a\boldsymbol{w}),$$
$$a_{\lambda,h}^K(\boldsymbol{v},\boldsymbol{w}) = (\Pi_{k-1}^0\mathrm{div}\boldsymbol{v},\Pi_{k-1}^0\mathrm{div}\boldsymbol{w})_K.$$

For the lowest-order case $k=1$, an integration by parts gives

$$\begin{aligned}\boldsymbol{B} = a_\mu^K(\boldsymbol{m},\boldsymbol{\phi}^T) &= \int_{\partial K}(\boldsymbol{\varepsilon}(\boldsymbol{m})\cdot\boldsymbol{n})\cdot\boldsymbol{\phi}^T\mathrm{d}s \\ &= \left[\varepsilon_{11}(\boldsymbol{m})\int_{\partial K}\phi^T n_x + \varepsilon_{12}(\boldsymbol{m})\int_{\partial K}\phi^T n_y,\ \ \varepsilon_{21}(\boldsymbol{m})\int_{\partial K}\phi^T n_x + \varepsilon_{22}(\boldsymbol{m})\int_{\partial K}\phi^T n_y\right]\end{aligned} \tag{40}$$

with

$$\varepsilon_{11}(\boldsymbol{m}) = [0,\frac{1}{h_K},0,0,0,0]^T, \qquad \varepsilon_{22}(\boldsymbol{m}) = [0,0,0,0,0,\frac{1}{h_K}]^T,$$
$$\varepsilon_{12}(\boldsymbol{m}) = \varepsilon_{21}(\boldsymbol{m}) = [0,0,\frac{1}{2h_K},0,\frac{1}{2h_K},0]^T.$$

Using the assembling technique for FEMs, the computation of $\boldsymbol{B}$ in MATLAB reads

```
elem1 = [v1(:), v2(:)];
C0 = zeros(1,2*Nv);
F = 1/2*[(1*Ne); (1*Ne)]; % [he*n1, he*n2]
C0(:) = accumarray([elem1(:);elem1(:)+Nv], F(:), [2*Nv 1]);
E = zeros(6,4);
E(2,1) = 1/hK;  E([3,5],[2,3]) = 1/(2*hK); E(6,4) = 1/hK;
B = [E(:,1)*C01x+E(:,2)*C01y, E(:,3)*C01x+E(:,4)*C01y];
```

Note that for the tensor type, we can consider the pure traction problem, i.e., $\Gamma_1 = \partial\Omega$. The numerical results are similar to that of the VEM of displacement type. The test script is `main_elasticityVEM.m`. It is worth pointing out that we proposed in [38] a novel conforming locking-free method with a rigorous proof, where several benchmarks are tested and the test script can be found in `main_elasticityVEM_reducedIntegration.m`.

5.2 Nonconforming VEMs

Nonconforming VEMs for the linear elasticity problems are first introduced in [54] for the pure displacement/traction formulation in two or three dimensions. The proposed method is robust with respect to the Lamé constant for $k \geq 2$.

For $k=1$, one easily finds that

$$\int_{\partial K}\phi^T n_x \mathrm{d}s = [h_{e_1}n_{1,x},\cdots,h_{e_{N_v}}n_{N_v,x}],$$

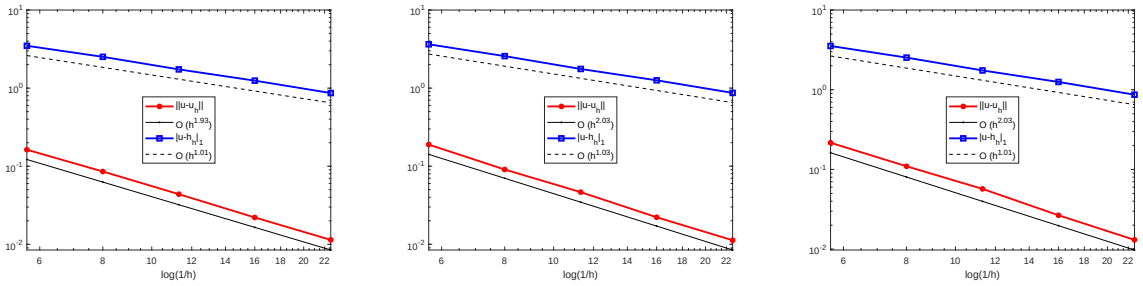
where h_{e_i} is the length of the edge e_i and $\boldsymbol{n}_{e_i} = (n_{i,x},n_{i,y})^T$. In this case, the computation of $\boldsymbol{B}$ in (40) for the tensor type reads

```
C01x = Ne(:,1)';  C01y = Ne(:,2)';
B = [E(:,1)*C01x+E(:,2)*C01y, E(:,3)*C01x+E(:,4)*C01y];
```

We provide the implementation of the displacement type and tensor type in the original nonconforming spaces with the results displayed in Fig. 16(a)(b). Similar to the Poisson equation, we

also present the realization of the tensor type in the modified nonconforming spaces and the convergence rate is shown in Fig. 16(c). Here, we still consider the test in Example 5.1 with $\lambda = 10^{10}$ and $\mu = 1$. As again observed in Fig. 16, the optimal rates of convergence are obtained for all the three methods with general polygonal meshes applied in the nearly incompressible case although we cannot justify it or provide a counter example.

The test scripts are `main_elasticityVEM_NavierNC.m`, `main_elasticityVEM_NC.m` and `main_elasticityVEM_NCb.m`, respectively.



(a) Displacement type (b) Tensor type (c) Tensor type with the continuous treatment

Fig. 16. Convergence rates of the nonconforming VEMs for the linear elasticity.

5.3 Several locking-free VEMs

The authors in [41] present two kinds of lowest-order VEMs with consistent convergence, in which the first one is achieved by introducing a special stabilization term to ensure the discrete Korn's inequality, and the second one can be seen as an extension of the idea of Kouhia and Stenberg suggested in [40] to the virtual element method. We provide the implementation of the second method in `elasticityVEM_KouhiaStenberg.m`. In this approach, the local space is taken as

$$\boldsymbol{V}(K) = V_1^{nc}(K) \times V_1(K),$$

where $V_1^{nc}(K)$ and $V_1(K)$ are the lowest-order nonconforming and conforming virtual element spaces, respectively. In this case, the computation for $k = 1$ reads

```
C01xe = Ne(:,1)';  C01ye = Ne(:,2)';
C01xv = 0.5*(Ne(p1,1)+Ne(p2,1))';
C01yv = 0.5*(Ne(p1,2)+Ne(p2,2))';
B = [E(:,1)*C01xe+E(:,2)*C01ye, E(:,3)*C01xv+E(:,4)*C01yv];
```

We also develop a lowest-order nonconforming virtual element method for planar linear elasticity in [53], which can be viewed as an extension of the idea in [32] to the virtual element method, with the family of polygonal meshes satisfying a very general geometric assumption. The method is shown to be uniformly convergent for the nearly incompressible case with optimal rates of convergence. In addition, we provide a unified locking-free scheme both for the conforming and nonconforming VEMs in the lowest order case. The implementation and the numerical test

can be found in [53]. The test scripts are `main_elasticityVEM_NCreducedIntegration.m` and `main_elasticityVEM_NCUniformReducedIntegration.m`, respectively.

6 Plate bending problems

In this section we focus on the plate bending problem in the form of

$$\begin{cases} -\partial_{ij} M_{ij}(w) = f & \text{in } \Omega \subset \mathbb{R}^2, \\ w = \partial_{\boldsymbol{n}} w = 0 & \text{on } \partial\Omega, \end{cases}$$

where

$$M_{ij} = D\left((1-\nu)K_{ij} + \nu K_{kk}\delta_{ij}\right), \quad i,j = 1,2, \quad k \in \{1,2\},$$
$$K_{ij} = -\partial_{ij} w, \quad i,j = 1,2, \quad D = \frac{Et^3}{12(1-\nu^2)}.$$

Note that we have used the summation convention whereby summation is implied when an index is repeated exactly two times. Please refer to [23, 30, 33] for details. The variational problem is to find $w \in V := H_0^2(\Omega)$ such that

$$a(w,v) = \ell(v), \quad v \in V,$$

where

$$a(w,v) = \int_\Omega M_{ij}(w) K_{ij}(v) \mathrm{d}x, \qquad \ell(v) = \int_\Omega f v \mathrm{d}x.$$

6.1 C^1-continuous VEMs

We first recall the H^2-conforming virtual element space $V_k^{2,c}(K)$ introduced in [23, 30]. For $k \geq 3$, define

$$V_k^{2,c}(K) = \{v \in H^2(K) : \Delta^2 v \in \mathbb{P}_{k-4}(K),\ v|_e \in \mathbb{P}_k(e),\ \partial_{\boldsymbol{n}} v|_e \in \mathbb{P}_{k-1}(e),\ e \subset \partial K\},$$

while for the lowest order $k = 2$, the space is modified as

$$V_2^{2,c}(K) = \left\{v \in H^2(K) : \Delta^2 v = 0,\ v|_e \in \mathbb{P}_3(e),\ \partial_{\boldsymbol{n}} v|_e \in \mathbb{P}_1(e),\ e \subset \partial K\right\}.$$

The d.o.f.s are:

- The values of $v(z)$ at the vertices of K.
- The values of $h_z \partial_1 v(z)$ and $h_z \partial_2 v(z)$ at the vertices of K, where h_z is a characteristic length attached to each vertex z, for instance, the average of the diameters of the elements having z as a vertex.
- The moments of v on edges up to degree $k-4$,

$$\chi_e(v) = |e|^{-1}(m_e, v)_e, \quad m_e \in \mathbb{M}_{k-4}(e).$$

- The moments of $\partial_{\boldsymbol{n}} v$ on edges up to degree $k-3$,

$$\chi_{n_e}(v) = (m_e, \partial_{\boldsymbol{n}} v)_e, \quad m_e \in \mathbb{M}_{k-3}(e).$$

- The moments on element K up to degree $k-4$,

$$\chi_K(v) = |K|^{-1}(m_K, v)_K, \quad m_K \in \mathbb{M}_{k-4}(K).$$

The elliptic projection can be defined in vector form as

$$\begin{cases} a^K(m, \Pi^K \phi^T) = a^K(m, \phi^T), \\ P_0^1(\Pi^K \phi^T) = P_0^1(\phi^T), \\ P_0^2(\nabla \Pi^K \phi^T) = P_0^2(\nabla \phi^T), \end{cases} \tag{41}$$

where

$$P_0^1(v) = \frac{1}{N_v} \sum_{i=1}^{N_v} v(a_i), \quad P_0^2(v) = \int_{\partial K} v \mathrm{d}s.$$

Introduce the following notation

$$Q_{3i} = M_{ij,j}, \quad Q_{3\boldsymbol{n}} = Q_{3i} n_i, \quad M_{\boldsymbol{nn}} = M_{ij} n_i n_j, \quad M_{\boldsymbol{tn}} = M_{ij} t_i n_j,$$

with $\boldsymbol{n} = (n_1, n_2)^T$ and $\boldsymbol{t} = (-n_2, n_1)^T$. We then have the integration by parts formula

$$\begin{aligned} a^K(p, v) = & -\int_K Q_{3i,i}(p) v \mathrm{d}x \\ & + \int_{\partial K} \left(Q_{3\boldsymbol{n}}(p) + \partial_{\boldsymbol{t}} M_{\boldsymbol{tn}}(p)\right) v \mathrm{d}s - \int_{\partial K} M_{\boldsymbol{nn}}(p) \partial_{\boldsymbol{n}} v \mathrm{d}s \\ & + \sum_{i=1}^{N_v} [M_{\boldsymbol{tn}}(p)](z_i) v(z_i), \end{aligned}$$

which implies the computability of the elliptic projection $\Pi^K v$ for any $v \in V_k^{2,c}(K)$, where the last term is a jump at z_i along the boundary ∂K, i.e.,

$$[M_{\boldsymbol{tn}}(p)](z_i) = M_{\boldsymbol{tn}}(p)\Big|_{z_i^-}^{z_i^+}.$$

We now consider the computation of the elliptic projection matrices. In the lowest order case, the d.o.f.s are

- The values of v at the vertices of K,

$$\chi_a(v) = v(z_i), \quad i = 1, \cdots, N_v.$$

- The values of ∇v at the vertices of K,

$$\chi_{a1}(v) = \partial_x v(z_i), \quad \chi_{a2}(v) = \partial_y v(z_i), \quad i = 1, \cdots, N_v.$$

In the implementation, they are arranged as

$$\begin{aligned} &\chi_i(v) = v(z_i), \quad i = 1, \cdots, N_v, \\ &\chi_{N_v+i}(v) = h_\xi \partial_x v(z_i), \quad i = 1, \cdots, N_v, \\ &\chi_{2N_v+i}(v) = h_\xi \partial_y v(z_i), \quad i = 1, \cdots, N_v, \end{aligned}$$

where h_ξ is the characteristic length attached to each vertex. To compute the transition matrix D, we first provide the characteristic lengths by using the data structure `node2elem`, given as

```
% characteristic length
hxi = cellfun(@(id) mean(diameter(id)), node2elem);
index = elem{iel};
hxiK = hxi(index);
```

Then the computation of the matrix D reads

```
% ------- scaled monomials ---------
% m'
m = @(x,y) [1+0*x, (x-xK)/hK, (y-yK)/hK, (x-xK).^2/hK^2, ...
            (x-xK).*(y-yK)/hK^2, (y-yK).^2/hK^2]; % m1,...,m6
% Dx(m'), Dy(m')
Dxm = @(x,y) [0*x, 1/hK+0*x, 0*x, 2*(x-xK)/hK^2, (y-yK)/hK^2, 0*x];
Dym = @(x,y) [0*x, 0*x, 1/hK+0*x, 0*x, (x-xK)/hK^2, 2*(y-yK)/hK^2];

% ------ transition matrix ---------
D = zeros(Ndof,Nm);
D(1:Nv,:) = m(x,y);
D(Nv+1:2*Nv,:) = repmat(hxiK,1,Nm).*Dxm(x,y);
D(2*Nv+1:end,:) = repmat(hxiK,1,Nm).*Dym(x,y);
```

For $k = 2$, one has

$$B_{\alpha j} = a^K(m_\alpha, \phi_j) = -\sum_{e\subset\partial K}\int_e M_{\boldsymbol{nn}}(m_\alpha)\partial_{\boldsymbol{n}}\phi_j \mathrm{d}s + \sum_{i=1}^{N_v}[M_{\boldsymbol{tn}}(m_\alpha)](z_i)\phi_j(z_i)$$
$$=: J_1(\alpha, j) + J_2(\alpha, j). \tag{42}$$

Since $\partial_{\boldsymbol{n}}\phi_j|_e \in \mathbb{P}_1(e)$, the trapezoidal rule gives

$$J_1(\alpha, j) = -\sum_{i=1}^{N_v} M_{\boldsymbol{nn}}(m_\alpha)|_{e_i}\frac{|e_i|}{2}\left(\partial_{\boldsymbol{n}}\phi_j(z_i) + \partial_{\boldsymbol{n}}\phi_j(z_{i+1})\right).$$

Noting that

$$\chi_{a1,i}(v) = h_\xi \partial_x v(z_i) \qquad \Rightarrow \qquad \partial_x v(z_i) = \frac{1}{h_\xi}\chi_{a1,i}(v), \quad i = 1, \cdots, Nv,$$

we then compute the values of $\partial_{\boldsymbol{n}}\phi_j$ at the vertices as follows.

```
% Dx(phi'), Dy(phi') at z1,...,zNv (each row)
Dxphi = zeros(Nv,Ndof); Dyphi = zeros(Nv,Ndof);
Dxphi(:,Nv+1:2*Nv) = eye(Nv)./repmat(hxiK,1,Nv);
Dyphi(:,2*Nv+1:end) = eye(Nv)./repmat(hxiK,1,Nv);
```

For J_2 with the entry given by

$$J_2(\alpha, j) = \sum_{i=1}^{N_v}[M_{\boldsymbol{tn}}(m_\alpha)](z_i)\phi_j(z_i),$$

by the definition of the jump,

$$[M_{\boldsymbol{tn}}(m_\alpha)](z_i) = M_{\boldsymbol{tn}}(m_\alpha)|_{z_i^-}^{z_i^+} = M_{\boldsymbol{tn}}(m_\alpha)(z_i^+) - M_{\boldsymbol{tn}}(m_\alpha)(z_i^-),$$

where $M_{\boldsymbol{tn}}(m_\alpha)(z_i^+)$ is the evaluation on the edge e_i to the right of z_i, $M_{\boldsymbol{tn}}(m_\alpha)(z_i^-)$ is the evaluation on the edge e_{i-1}, and

$$\phi^T(z_i) = [\boldsymbol{e}_i, \mathbf{0}, \mathbf{0}],$$

where $\boldsymbol{e}_i$ is a zero vector with i-th entry being 1.

The above discussion is summarized in the following code.

```
% --------- elliptic projection -----------
% \partial_ij (m)
D11 = zeros(Nm,1); D11(4) = 2/hK^2;
D12 = zeros(Nm,1); D12(5) = 1/hK^2;
D22 = zeros(Nm,1); D22(6) = 2/hK^2;
% Mij(m)
M11 = -para.D*((1-para.nu)*D11 + para.nu*(D11+D22));
M12 = -para.D*(1-para.nu)*D12;
M22 = -para.D*((1-para.nu)*D22 + para.nu*(D11+D22));
% Mnn(m) on e1,...,eNv
n1 = ne(:,1);  n2 = ne(:,2);
Mnn = M11*(n1.*n1)' + M12*(n1.*n2+n2.*n1)' + M22*(n2.*n2)';
% Mtn(m) on e1,...,eNv
t1 = te(:,1); t2 = te(:,2);
Mtn = M11*(t1.*n1)' + M12*(t1.*n2+t2.*n1)' + M22*(t2.*n2)';
% Dx(phi'), Dy(phi') at z1,...,zNv (each row)
Dxphi = zeros(Nv,Ndof); Dyphi = zeros(Nv,Ndof);
Dxphi(:,Nv+1:2*Nv) = eye(Nv)./repmat(hxiK,1,Nv);
Dyphi(:,2*Nv+1:end) = eye(Nv)./repmat(hxiK,1,Nv);
% B, Bs, G, Gs
B = zeros(Nm,Ndof);
p1 = [Nv,1:Nv-1]; p2 = 1:Nv;
for j = 1:Nv % loop of edges
    % int[\partial_n (phi')] on ej
    Dnphi1 = Dxphi(v1(j),:)*n1(j) + Dyphi(v1(j),:)*n2(j); % zj
    Dnphi2 = Dxphi(v2(j),:)*n1(j) + Dyphi(v2(j),:)*n2(j); % z_{j+1}
    nphi = 0.5*he(j)*(Dnphi1+Dnphi2);
    % Jump(m) at zj
    Jump = Mtn(:,p2(j))-Mtn(:,p1(j));
    % phi' at zj
    phi = zeros(1,Ndof);  phi(j) = 1;
    % B1 on e and at zj
    B = B - Mnn(:,j)*nphi + Jump*phi;
end
```

The first constraint in (41) is

$$\tilde{B}(1,:) = P_0^1(\phi^T) = \frac{1}{N_v}[\mathbf{1}, \mathbf{0}, \mathbf{0}].$$

The second constraint can be computed by using the integration by parts as

$$\begin{aligned}\int_{\partial K} \nabla v \mathrm{d}s &= \sum_{e\subset\partial K} \int_e \partial_{n_e} v n_e \mathrm{d}s + \int_e \partial_{t_e} v t_e \mathrm{d}s \\ &= \sum_{e\subset\partial K} n_e \int_e \partial_{\boldsymbol{n}} v \mathrm{d}s + \sum_{i=1}^{N_v} t_{e_i}(v(z_{i+1}) - v(z_i)),\end{aligned}$$

where

$$\int_{\partial K} \nabla \phi_j \mathrm{d}s = t_{e_{j-1}} - t_{e_j}.$$

The computation reads

```
Bs = B;
% first constraint
Bs(1,1:Nv) = 1/Nv;
% second constraint
for j = 1:Nv % loop of edges
```

```
    Bs(2:3,1:Nv) = te([Nv,1:Nv-1],:)' - te';
    Dnphi1 = Dxphi(v1(j),Nv+1:end)*n1(j) + Dyphi(v1(j),Nv+1:end)*n2(j); % zj
    Dnphi2 = Dxphi(v2(j),Nv+1:end)*n1(j) + Dyphi(v2(j),Nv+1:end)*n2(j); % z_{j+1}
    Nphi = 0.5*Ne(j,:)'*(Dnphi1+Dnphi2); % scaled
    Bs(2:3,Nv+1:end) = Bs(2:3,Nv+1:end) + Nphi;
end
```

Example 6.1. *The exact solution is chosen as* $u = \sin(2\pi x)\cos(2\pi y)$ *with the parameters* $t = 0.1$, $E = 10920$ *and* $\nu = 0.3$.

We consider a sequence of meshes, which is a Centroidal Voronoi Tessellation of the unit square in 32, 64, 128, 256 and 512 polygons. The results are shown in Fig. 17. The optimal rate of convergence of the discrete H^2-norm (1st order), H^1-norm (2nd order) and L^2-norm (2nd order) is observed for the lowest order $k = 2$ when meshes are fine enough. The test script is `main_PlateBending_C1VEM.m`.

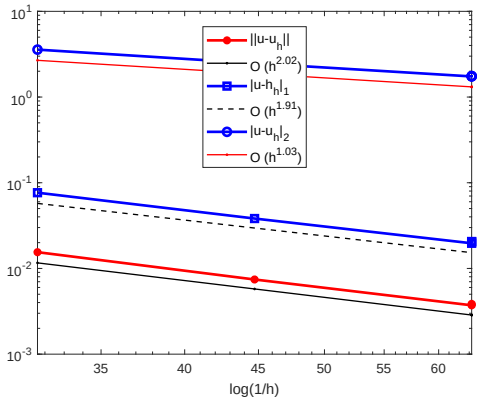


Fig. 17. Convergence rates of the plate bending problem for the C^1-continuous virtual elements

6.2 C^0-continuous VEMs

The Ref. [55] gives the C^0-continuous nonconforming virtual element method for the plate bending problem, with the local virtual element space defined as

$$V_k^{2,0}(K) = \{v \in H^2(K) : \Delta^2 v \in \mathbb{P}_{k-4}(K), \quad v|_{\partial K} \in \mathbb{B}_k(\partial K)\},$$

where K is a convex polygon and

$$\mathbb{B}_k(\partial K) = \{v \in C^0(\partial K) : v|_e \in \mathbb{P}_k(e), \quad \Delta v|_e \in \mathbb{P}_{k-2}(e), \quad e \subset \partial K\}.$$

A function in $V_k^{2,0}(K)$ is uniquely identified by the following degrees of freedom:

- The values at the vertices of K,

$$\chi_i(v) = v(z_i), \quad z_i \text{ is a vertex of } K.$$

- The moments of v on edges up to degree $k-2$,

$$\chi_e(v) = |e|^{-1}(m_e, v)_e, \quad v \in \mathbb{M}_{k-2}(e).$$

- The moments of $\partial_{\boldsymbol{n}} v$ on edges up to degree $k-2$,

$$\chi_{n_e}(v) = (m_e, \partial_{\boldsymbol{n}} v)_e, \quad m \in \mathbb{M}_{k-2}(e).$$

- The moments of v on K up to degree $k-4$,

$$\chi_K(v) = |K|^{-1}(m_K, v)_K, \quad v \in \mathbb{M}_{k-4}(K).$$

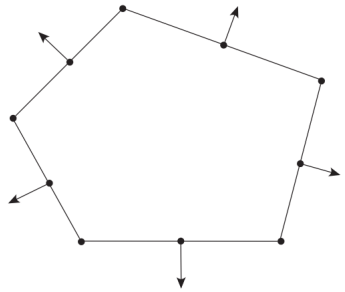


Fig. 18. Local degrees of freedom for the C^0-continuous virtual elements ($k = 2$)

We only consider the lowest order case $k = 2$. The d.o.f.s contain the values at the vertices and the following moments

$$\chi_e(v) = \frac{1}{|e|}\int_e v \mathrm{d}s, \quad \chi_{n_e}(v) = \int_e \partial_{\boldsymbol{n}} v \mathrm{d}s,$$

as shown in Fig. 18. Note that the first-type moments can also be replaced by the midpoint values on edges and the second-type moment has different signs when restricted to the left and right elements of an interior edge. Let N_v be the number of vertices. Then there are $N_k = 3N_v$ d.o.f.s on each element, arranged as

- The values at the vertices of K,

$$\chi_i(v) = v(z_i), \quad i = 1, \cdots, N_v.$$

- The mid-point values on each edge of K,

$$\chi_{i+N_v}(v) = v(m_i), \quad i = 1, \cdots, N_v.$$

- The moments

$$\chi_{2N_v+i}(v) = \int_{e_i} \partial_{\boldsymbol{n}} v \mathrm{d}s, \quad i = 1, \cdots, N_v.$$

Since

$$\int_e \partial_{\boldsymbol{n}} \phi^T \mathrm{d}s = [\mathbf{0}, \mathbf{0}, \boldsymbol{e}_j], \quad \phi^T(z_i) = [\boldsymbol{e}_i, \mathbf{0}, \mathbf{0}],$$

the matrix B for the elliptic projection can be realized as

```
B = zeros(Nm,Ndof);
p1 = [Nv,1:Nv-1]; p2 = 1:Nv;
for j = 1:Nv % loop of edges
    % nphi on ej
    nphi = zeros(1,Ndof); nphi(2*Nv+j) = 1;
```

```
    % Jump at zj
    Jump = Mtn(:,p2(j))-Mtn(:,p1(j));
    % phi at zj
    phi = zeros(1,Ndof); phi(j) = 1;
    % B1
    B = B - Mnn(:,j)*nphi + Jump*phi;
end
```

Noting that

$$\int_{\partial K} \nabla\phi_j \mathrm{d}s = \boldsymbol{t}_{e_{j-1}} - \boldsymbol{t}_{e_j}, \quad \int_{\partial K} \nabla\phi_{N_v+j} \mathrm{d}s = \boldsymbol{0}, \quad \int_{\partial K} \nabla\phi_{2N_v+j} \mathrm{d}s = \boldsymbol{n}_{e_j},$$

we can compute the constraints as follows.

```
Bs = B;
% first constraint
Bs(1,1:Nv) = 1/Nv;
% second constraint
Bs(2:3,1:Nv) = te([Nv,1:Nv-1],:)' - te';
Bs(2:3,2*Nv+1:end) = ne';
```

We repeat the test in Example 6.1 and display the result in Fig. 19. In this case, we still observe the optimal convergence rates. The test script is `main_PlateBending_C0VEM.m`.

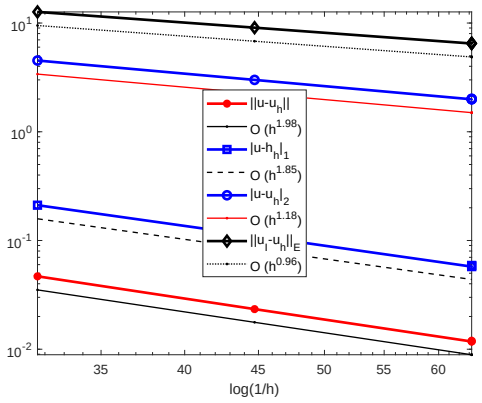


Fig. 19. Convergence rates of the plate bending problem for the C^0-continuous virtual elements

6.3 Morley-type VEMs

The fully H^2-nonconforming virtual element method was proposed in [3] for biharmonic problem. The local virtual element space is defined by

$$V_k^2(K) = \{v \in H^2(K) : \Delta^2 v \in \mathbb{P}_{k-4}(K), \\ M_{\boldsymbol{nn}}(v)|_e \in \mathbb{P}_{k-2}(e), \quad Q_{\boldsymbol{n}}(v)|_e \in \mathbb{P}_{k-3}(e), \quad e \subset \partial K\}.$$

The corresponding degrees of freedom can be chosen as:

- The values at the vertices of K.
- The moments of v on edges up to degree $k-3$,

$$\chi_e(v) = |e|^{-1}(m_e, v)_e, \quad m_e \in \mathbb{M}_{k-3}(e).$$

- The moments of $\partial_{\boldsymbol{n}} v$ on edges up to degree $k-2$,

$$\chi_{n_e}(v) = (m_e, \partial_{\boldsymbol{n}} v)_e, \quad m_e \in \mathbb{M}_{k-2}(e).$$

- The moments on element K up to degree $k-4$,

$$\chi_K(v) = |K|^{-1}(m_K, v)_K, \quad m_K \in \mathbb{M}_{k-4}(K).$$

If $k=2$ and K is a triangle, one can prove that $V_k^2(K)$ becomes the well-known Morley element. We remark that the above Morley-type virtual element is also proposed in [56] with the same d.o.f.s but different local space, where the enhancement technique in [2] is utilized to modify the C^0-continuous spaces.

The test script is `main_PlateBending_MorleyVEM.m`. We repeat the test in Example 6.1 and display the result in Fig. 20, from which we again observe the optimal rate of convergence in all the discrete norms.

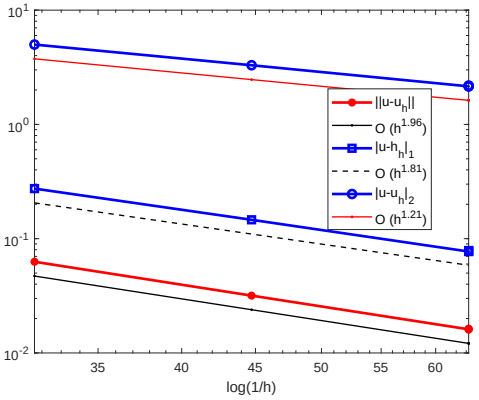


Fig. 20. Convergence rates of the plate bending problem for the fully nonconforming virtual elements

7 Stokes problem

The Stokes problem with homogeneous Dirichlet boundary conditions is to find $(\boldsymbol{u}, p)$ such that

$$\begin{cases} -\nu\Delta\boldsymbol{u} - \nabla p = \boldsymbol{f} & \text{in } \Omega, \\ \mathrm{div}\boldsymbol{u} = 0 & \text{in } \Omega, \\ \boldsymbol{u} = \boldsymbol{0} & \text{on } \partial\Omega. \end{cases}$$

Define

$$\boldsymbol{V} = \boldsymbol{H}_0^1(\Omega), \quad Q = L_0^2(\Omega).$$

The mixed variational problem is: Find $(\boldsymbol{u}, p) \in \boldsymbol{V} \times Q$ such that

$$\begin{cases} a(\boldsymbol{u}, \boldsymbol{v}) + b(\boldsymbol{v}, p) = (\boldsymbol{f}, \boldsymbol{v}), & \boldsymbol{v} \in \boldsymbol{V}, \\ b(\boldsymbol{u}, q) = 0, & q \in Q, \end{cases}$$

where

$$a(\boldsymbol{u}, \boldsymbol{v}) = (\nu\nabla\boldsymbol{u}, \nabla\boldsymbol{v}), \qquad b(\boldsymbol{v}, q) = (\mathrm{div}\boldsymbol{v}, q).$$

7.1 The mixed VEMs

In this subsection, we review the divergence-free virtual elements proposed in [17] for the Stokes problem.

7.1.1 Virtual element space and elliptic projection

The local virtual element space associated with $\boldsymbol{V}$ for $k \geq 2$ is

$$V_k(K) = \Big\{ \boldsymbol{v} \in \boldsymbol{H}^1(K) : \boldsymbol{v}|_{\partial K} \in [\mathbb{B}_k(\partial K)]^2, \\ \begin{cases} -\nu\Delta\boldsymbol{v} - \nabla s \in \mathcal{G}_{k-2}(K)^{\perp}, \\ \mathrm{div}\boldsymbol{v} \in \mathbb{P}_{k-1}(K), \end{cases} \quad \text{for some } s \in L^2(K) \Big\},$$

where

$$\mathcal{G}_{k-2}(K) = \nabla(\mathbb{P}_{k-1}(K)) \subset (\mathbb{P}_{k-2}(K))^2.$$

The finite dimensional space for Q is taken as $Q_k(K) = \mathbb{P}_{k-1}(K)$.

The d.o.f.s for $V_k(K)$ and $Q_k(K)$ can be chosen as

- $\mathbb{D}_{\mathbb{V}}1$: the values at the vertices of K.
- $\mathbb{D}_{\mathbb{V}}2$: $k-1$ interior points on each edge e.
- $\mathbb{D}_{\mathbb{V}}3$: the moments
$$\int_K \boldsymbol{v} \cdot \boldsymbol{g}_{k-2}^{\perp} \mathrm{d}x, \quad \boldsymbol{g}_{k-2}^{\perp} \in \mathcal{G}_{k-2}(K)^{\perp}.$$
- $\mathbb{D}_{\mathbb{V}}4$: the moments
$$\int_K (\mathrm{div}\boldsymbol{v}) q_{k-1} \mathrm{d}x, \quad q_{k-1} \in \mathbb{P}_{k-1}(K)/\mathbb{R}.$$
- $\mathbb{D}_{\mathbb{Q}}$: the moments
$$\int_K q p_{k-1} \mathrm{d}x, \quad p_{k-1} \in \mathbb{P}_{k-1}(K).$$

As usual, we define the elliptic projection $\Pi^K : V_k(K) \to (\mathbb{P}_k(K))^2$, $\boldsymbol{v} \mapsto \Pi^K\boldsymbol{v}$ satisfying

$$\begin{cases} a^K(\Pi^K\boldsymbol{v}, \boldsymbol{q}) = a^K(\boldsymbol{v}, \boldsymbol{q}), \quad \boldsymbol{q} \in (\mathbb{P}_k(K))^2, \\ P_0^K(\Pi^K\boldsymbol{v}) = P_0^K(\boldsymbol{v}), \end{cases} \tag{43}$$

where P_0^K is the constant projector on K. One can check that the elliptic projection $\Pi^K\boldsymbol{v}$ is computable by using the given d.o.f.s. In fact, the integration by parts gives

$$a^K(\boldsymbol{v}, \boldsymbol{q}) = \int_K \nu\nabla\boldsymbol{q} \cdot \nabla\boldsymbol{v} \mathrm{d}x = -\int_K \nu\Delta\boldsymbol{q} \cdot \boldsymbol{v} \mathrm{d}x + \int_{\partial K} (\nu\nabla\boldsymbol{q}\boldsymbol{n}) \cdot \boldsymbol{v} \mathrm{d}s,$$

where $\boldsymbol{n} = (n_1, n_2)^T$ and

$$\nabla\boldsymbol{q} = \begin{bmatrix} \nabla q_1 \\ \nabla q_2 \end{bmatrix} = \begin{bmatrix} q_{1,x} & q_{1,y} \\ q_{2,x} & q_{2,y} \end{bmatrix}.$$

Since $\nu\Delta\boldsymbol{q} \in (\mathbb{P}_{k-2}(K))^2$, there exists $q_{k-1} \in \mathbb{P}_{k-1}(K)/\mathbb{R}$ and $\boldsymbol{g}_{k-2}^{\perp} \in \mathcal{G}_{k-2}(K)^{\perp}$ such that

$$\nu\Delta\boldsymbol{q} = \nabla q_{k-1} + \boldsymbol{g}_{k-2}^{\perp}.$$

Then

$$a^K(\boldsymbol{v},\boldsymbol{q}) = -\int_K \nabla q_{k-1}\cdot\boldsymbol{v}\mathrm{d}x - \int_K \boldsymbol{g}_{k-2}^{\perp}\cdot\boldsymbol{v}\mathrm{d}x + \int_{\partial K}(\nu\nabla\boldsymbol{q}\boldsymbol{n})\cdot\boldsymbol{v}\mathrm{d}s$$
$$= \int_K q_{k-1}\mathrm{div}\boldsymbol{v}\mathrm{d}x - \int_K \boldsymbol{g}_{k-2}^{\perp}\cdot\boldsymbol{v}\mathrm{d}x + \int_{\partial K}(\nu\nabla\boldsymbol{q}\boldsymbol{n} - q_{k-1}\boldsymbol{n})\cdot\boldsymbol{v}\mathrm{d}s. \tag{44}$$

Obviously, the first term and the second term are determined by $\mathbb{D}_{\mathbb{V}}4$ and $\mathbb{D}_{\mathbb{V}}3$, respectively, while the boundary term can be computed by $\mathbb{D}_{\mathbb{V}}1$ and $\mathbb{D}_{\mathbb{V}}2$. Note that $\boldsymbol{g}_{k-2}^{\perp} = \boldsymbol{0}$ in the lowest order case $k = 2$.

7.1.2 The discrete problem

The bilinear form $a^K(\boldsymbol{u},\boldsymbol{v})$ is approximated by

$$a_h^K(\boldsymbol{u},\boldsymbol{v}) = a^K(\Pi^K\boldsymbol{u},\Pi^K\boldsymbol{v}) + S^K(\boldsymbol{u}-\Pi^K\boldsymbol{u},\boldsymbol{v}-\Pi^K\boldsymbol{v}),$$

with the stabilization term S^K given by the l^2-inner product of the d.o.f. vectors.

In what follows we use V_h to denote the global virtual element space of $V_k(K)$. The discrete space Q_h of Q is

$$Q_h := \{q\in Q : q|_K \in \mathbb{P}_{k-1}(K), \quad K\in\mathcal{T}_h.\}$$

The discrete mixed problem is: Find $(\boldsymbol{u}_h, p_h)\in V_h^g\times Q_h$ such that

$$\begin{cases} a_h(\boldsymbol{u}_h,\boldsymbol{v}_h) + b(\boldsymbol{v}_h,p_h) &= (\boldsymbol{f}_h,\boldsymbol{v}_h), \quad \boldsymbol{v}_h\in V_h,\\ b(\boldsymbol{u}_h,q_h) &= 0, \quad q_h\in Q_h. \end{cases} \tag{45}$$

The constraint $\int_\Omega p_h\mathrm{d}x$ is not naturally imposed in the above system. To this end, we introduce a Lagrange multiplier and consider the augmented variational formulation: Find $((\boldsymbol{u}_h,p_h),\lambda)\in V_h^g\times Q_h\times\mathbb{R}$ such that

$$\begin{cases} a_h(\boldsymbol{u}_h,\boldsymbol{v}_h) + b(\boldsymbol{v}_h,p_h) &= (\boldsymbol{f}_h,\boldsymbol{v}_h), \quad \boldsymbol{v}_h\in V_h,\\ b(\boldsymbol{u}_h,q_h) + \lambda\displaystyle\int_\Omega q_h\mathrm{d}x &= 0, \quad q_h\in Q_h,\\ \mu\displaystyle\int_\Omega p_h\mathrm{d}x &= 0, \quad \mu\in\mathbb{R}. \end{cases} \tag{46}$$

Let $\boldsymbol{\phi}_i$, $i = 1,\cdots,N$ be the basis functions of V_h. Then

$$\boldsymbol{u} = \sum_{i=1}^N \chi_i(\boldsymbol{u})\boldsymbol{\phi}_i =: \boldsymbol{\phi}^T\boldsymbol{\chi}(\boldsymbol{u}).$$

Similarly, the basis functions of Q_h are denote by ψ_l, $l = 1,\cdots,M$, with

$$p_h = \sum_{l=1}^M p_l\psi_l.$$

Plugging these expansions in (13) and taking $\boldsymbol{v}_h = \boldsymbol{\phi}_j$ and $q_h = \psi_l$, we obtain

$$\begin{cases} \sum\limits_{i=1}^N a_h(\boldsymbol{\phi}_i,\boldsymbol{\phi}_j)\chi_i + \sum\limits_{l=1}^M b(\boldsymbol{\phi}_j,\psi_l)p_l &= (\boldsymbol{f}_h,\boldsymbol{\phi}_j), \quad j = 1,\cdots,N,\\ \displaystyle\sum_{i=1}^N b(\boldsymbol{\phi}_i,\psi_l)\chi_i + \lambda\int_\Omega\psi_l\mathrm{d}x &= 0, \quad l = 1,\cdots,M,\\ \displaystyle\sum_{l=1}^M\int_\Omega\psi_l\mathrm{d}x p_l &= 0. \end{cases}$$

Let

$$d_l = \int_\Omega \psi_l \mathrm{d}x, \quad \boldsymbol{d} = [d_1, \cdots, d_M]^T.$$

The linear system can be written in matrix form as

$$\begin{bmatrix} A & B & \mathbf{0} \\ B^T & O & \boldsymbol{d} \\ \mathbf{0}^T & \boldsymbol{d}^T & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{\chi} \\ \boldsymbol{p} \\ \lambda \end{bmatrix} = \begin{bmatrix} \boldsymbol{f} \\ \mathbf{0} \\ 0 \end{bmatrix}, \tag{47}$$

where

$$A = (a_h(\boldsymbol{\phi}_j, \boldsymbol{\phi}_i)), \quad B = (b(\boldsymbol{\phi}_j, \psi_l)), \quad \boldsymbol{f} = ((\boldsymbol{f}_h, \boldsymbol{\phi}_j)).$$

We only consider the lowest order case $k = 2$. In this case, the third-type d.o.f.s vanish and the boundary part corresponds to a tensor-product space. We arrange the d.o.f.s in the following order:

$$\begin{aligned}
&\chi_i(\boldsymbol{v}) = \boldsymbol{v}_1(z_i), \quad i = 1, \cdots, N_v, \\
&\chi_{N_v+i}(\boldsymbol{v}) = \boldsymbol{v}_1(m_i), \quad i = 1, \cdots, N_v, \\
&\chi_{2*N_v+i}(\boldsymbol{v}) = \boldsymbol{v}_2(z_i), \quad i = 1, \cdots, N_v, \\
&\chi_{3N_v+i}(\boldsymbol{v}) = \boldsymbol{v}_2(m_i), \quad i = 1, \cdots, N_v, \\
&\chi_{4N_v+1}(\boldsymbol{v}) = \int_K \mathrm{div}\boldsymbol{v} m_2(x,y)\mathrm{d}x, \quad m_2(x,y) = \frac{x - x_K}{h_K}, \\
&\chi_{4N_v+2}(\boldsymbol{v}) = \int_K \mathrm{div}\boldsymbol{v} m_3(x,y)\mathrm{d}x, \quad m_3(x,y) = \frac{y - y_K}{h_K}.
\end{aligned}$$

Denote the basis functions of $\mathbb{B}_k(\partial K)$ by $\phi_1, \cdots, \phi_{N_v}; \phi_{N_v+1}, \cdots, \phi_{2N_v}$. Then the tensor-product space $(\mathbb{B}_k(\partial K))^2$ has the basis functions:

$$\overline{\phi}_1, \cdots, \overline{\phi}_{2N_v}, \underline{\phi}_1, \cdots, \underline{\phi}_{2N_v},$$

which correspond to the first $4N_v$ d.o.f.s. For convenience, we denote these functions by $\boldsymbol{\phi}_1, \boldsymbol{\phi}_2, \cdots, \boldsymbol{\phi}_{4N_v}$. The basis functions associated with the last two d.o.f.s are then denoted by $\boldsymbol{\varphi}_1 = \boldsymbol{\phi}_{4N_v+1}, \boldsymbol{\varphi}_2 = \boldsymbol{\phi}_{4N_v+2}$.

In the following, we only provide the details of computing the elliptic projection.

7.2 Computation of the elliptic projection

7.2.1 Transition matrix

We rewrite the basis of $V_k(K)$ in a compact form as

$$\boldsymbol{\phi}^T = (\boldsymbol{\phi}_1, \boldsymbol{\phi}_2, \cdots, \boldsymbol{\phi}_{N_k}),$$

where $N_k = 4N_v + 2$ is the number of the d.o.f.s. The basis of $(\mathbb{P}_k(K))^2$ can be denote by

$$\boldsymbol{m}^T = (\boldsymbol{m}_1, \boldsymbol{m}_2, \cdots, \boldsymbol{m}_{N_p}) = (\overline{m}_1, \cdots, \overline{m}_6, \underline{m}_1, \cdots, \underline{m}_6).$$

Since $(\mathbb{P}_k(K))^2 \subset V_k(K)$, one has $\boldsymbol{m}^T = \boldsymbol{\phi}^T \boldsymbol{D}$, where $\boldsymbol{D}$ is referred to as the transition matrix from $(\mathbb{P}_k(K))^2$ to $V_k(K)$. Let

$$\boldsymbol{m}^T = [\overline{m}_1, \overline{m}_2, \overline{m}_3, \underline{m}_1, \underline{m}_2, \underline{m}_3] =: [\overline{m}^T, \underline{m}^T],$$

$$\boldsymbol{\phi}^T = [\overline{\phi}_1, \cdots, \overline{\phi}_N, \underline{\phi}_1, \cdots, \underline{\phi}_N, \boldsymbol{\varphi}_1, \boldsymbol{\varphi}_2] =: [\overline{\phi}^T, \underline{\phi}^T, \boldsymbol{\varphi}_1, \boldsymbol{\varphi}_2].$$

Blocking the matrix $\boldsymbol{D}$ as $\boldsymbol{D} = \begin{bmatrix} \boldsymbol{D}_1 \\ \boldsymbol{D}_2 \end{bmatrix}$, one has

$$\begin{aligned} \boldsymbol{m} = [\overline{m}^T, \underline{m}^T] &= [\overline{\phi}^T, \underline{\phi}^T, 0, 0]\boldsymbol{D} + [\mathbf{0}^T, \mathbf{0}^T, \boldsymbol{\varphi}_1, \boldsymbol{\varphi}_2]\boldsymbol{D} \\ &= [\overline{\phi}^T, \underline{\phi}^T, 0, 0] \begin{bmatrix} \boldsymbol{D}_1 \\ \boldsymbol{O} \end{bmatrix} + [\mathbf{0}^T, \mathbf{0}^T, \boldsymbol{\varphi}_1, \boldsymbol{\varphi}_2] \begin{bmatrix} \boldsymbol{O} \\ \boldsymbol{D}_2 \end{bmatrix}. \end{aligned}$$

Let $m^T = \phi^T D$. One easily finds that

$$\boldsymbol{D}_1 = \begin{bmatrix} D & \\ & D \end{bmatrix}, \qquad \boldsymbol{D}_2 = \begin{bmatrix} \chi_{4N_v+1}(\overline{m}^T) & \chi_{4N_v+1}(\underline{m}^T) \\ \chi_{4N_v+2}(\overline{m}^T) & \chi_{4N_v+2}(\underline{m}^T) \end{bmatrix}$$

We first provide some necessary information.

```
% ------- element information ----------
index = elem{iel};      Nv = length(index);
xK = centroid(iel,1); yK = centroid(iel,2); hK = diameter(iel);
x = node(index,1); y = node(index,2);
v1 = 1:Nv; v2 = [2:Nv,1]; % loop index for vertices or edges
xe = (x(v1)+x(v2))/2;  ye = (y(v1)+y(v2))/2; % mid-edge points
Ne = [y(v2)-y(v1), x(v1)-x(v2)]; % he*ne
nodeT = [node(index,:);centroid(iel,:)];
elemT = [(Nv+1)*ones(Nv,1),(1:Nv)',[2:Nv,1]'];

% --------------- scaled monomials -----------------
m1 = @(x,y) 1+0*x;                    gradm1 = @(x,y) [0+0*x, 0+0*x];
m2 = @(x,y) (x-xK)./hK;               gradm2 = @(x,y) [1/hK+0*x, 0+0*x];
m3 = @(x,y) (y-yK)./hK;               gradm3 = @(x,y) [0+0*x, 1/hK+0*x];
m4 = @(x,y) (x-xK).^2/hK^2;           gradm4 = @(x,y) [2*(x-xK)./hK^2, 0+0*x];
m5 = @(x,y) (x-xK).*(y-yK)./hK^2;     gradm5 = @(x,y) [(y-yK)./hK^2, (x-xK)./hK^2];
m6 = @(x,y) (y-yK).^2./hK^2;          gradm6 = @(x,y) [0+0*x, 2*(y-yK)./hK^2];

m = @(x,y) [m1(x,y), m2(x,y), m3(x,y), m4(x,y), m5(x,y), m6(x,y)];
Gradm = {gradm1, gradm2, gradm3, gradm4, gradm5, gradm6};
divmm = @(x,y) [0+0*x, 1/hK+0*x, 0+0*x, 2*(x-xK)./hK^2, (y-yK)./hK^2, 0+0*x, ...
    0+0*x, 0+0*x, 1/hK+0*x, 0+0*x, (x-xK)./hK^2, 2*(y-yK)./hK^2];
```

Then the transition matrix can be realized as

```
% -------- transition matrix ----------
NdofBd = 2*Nv; NdofA = 2*NdofBd+2;
divmm2 = @(x,y) divmm(x,y).*repmat(m2(x,y),1,Nmm);
divmm3 = @(x,y) divmm(x,y).*repmat(m3(x,y),1,Nmm);
D = zeros(NdofA, Nmm);
Dbd = [m(x,y); m(xe,ye)];
D(1:4*Nv, :) = blkdiag(Dbd, Dbd);
D(end-1,:) = integralTri(divmm2,4,nodeT,elemT);
D(end,:) = integralTri(divmm3,4,nodeT,elemT);
```

7.2.2 Elliptic projection matrices

The elliptic projection satisfies

$$\begin{cases} \boldsymbol{G}\boldsymbol{\Pi}_*^K = \boldsymbol{B}, \\ P_0^K(\boldsymbol{m}^T)\boldsymbol{\Pi}_*^K = P_0^K(\boldsymbol{\phi}^T) \end{cases} \quad \text{or} \quad \tilde{\boldsymbol{G}}\boldsymbol{\Pi}_*^K = \tilde{\boldsymbol{B}},$$

where

$$\boldsymbol{G} = a^K(\boldsymbol{m}, \boldsymbol{m}^T), \quad \boldsymbol{B} = a^K(\boldsymbol{m}, \boldsymbol{\phi}^T).$$

For $k = 2$,

$$\begin{aligned} \boldsymbol{B}_{\alpha i} = a^K(\boldsymbol{m}_\alpha, \boldsymbol{\phi}_i) &= \int_K q_\alpha \mathrm{div}\boldsymbol{\phi}_i \mathrm{d}x + \int_{\partial K} (\nu \nabla \boldsymbol{m}_\alpha \boldsymbol{n} - q_\alpha \boldsymbol{n}) \cdot \boldsymbol{\phi}_i \mathrm{d}s \\ &=: I_1(\alpha, i) + I_2(\alpha, i), \end{aligned}$$

where

$$I_1(\alpha, i) = \int_K q_\alpha \mathrm{div}\boldsymbol{\phi}_i \mathrm{d}x, \quad I_2(\alpha, i) = \int_{\partial K} (\nu \nabla \boldsymbol{m}_\alpha \boldsymbol{n} - q_\alpha \boldsymbol{n}) \cdot \boldsymbol{\phi}_i \mathrm{d}s,$$

and

$$\nu \Delta \boldsymbol{m}_\alpha = \nabla q_\alpha + 0, \quad q_\alpha \in \mathbb{P}_1(K)/\mathbb{P}_0(K).$$

First consider I_1. By definition, let $q_\alpha = \nu h_K(c_{2,\alpha} m_2 + c_{3,\alpha} m_3)$. Then

$$\begin{bmatrix} c_{2,\alpha} \\ c_{3,\alpha} \end{bmatrix} \longleftarrow \begin{bmatrix} \Delta m^T & \boldsymbol{0}^T \\ \boldsymbol{0}^T & \Delta m^T \end{bmatrix}.$$

The Kronecher's property gives

$$\begin{aligned} I_1(\alpha, i) &= \nu h_K \Big(c_{2,\alpha} \int_K m_2 \mathrm{div}\boldsymbol{\phi}_i \mathrm{d}x + c_{3,\alpha} \int_K m_3 \mathrm{div}\boldsymbol{\phi}_i \mathrm{d}x \Big) \\ &= \nu h_K \Big(c_{2,\alpha} \delta_{i,(4N_v+1)} + c_{3,\alpha} \delta_{i,(4N_v+2)} \Big). \end{aligned}$$

The computation of I_1 reads

```
% --- first term ---
Lapm = [0, 0, 0, 2/hK^2, 0, 2/hK^2];
B = zeros(Nmm, NdofA);
B(1:Nm, end-1) = pde.nu*hK*Lapm;
B(Nm+1:end, end) = pde.nu*hK*Lapm;
```

For the second term I_2, let $(g_1^\alpha, g_2^\alpha)^T = \nu \nabla \boldsymbol{m}_\alpha \boldsymbol{n} - q_\alpha \boldsymbol{n}$, Then

$$I_2(\alpha, i) = \int_{\partial K} g_1^\alpha \boldsymbol{\phi}_{1,i} \mathrm{d}s + \int_{\partial K} g_2^\alpha \boldsymbol{\phi}_{2,i} \mathrm{d}s =: J_1(\alpha, i) + J_2(\alpha, i).$$

Let ϕ be the basis of $\mathbb{B}_k(\partial K)$. One has

$$J_1(\alpha, :)^T = \begin{bmatrix} (g_1^\alpha, \phi)_{\partial K} \\ (g_1^\alpha, \boldsymbol{0})_{\partial K} \\ (g_1^\alpha, \boldsymbol{\varphi}_{1,1})_{\partial K} \\ (g_1^\alpha, \boldsymbol{\varphi}_{2,1})_{\partial K} \end{bmatrix} = \begin{bmatrix} (g_1^\alpha, \phi)_{\partial K} \\ \boldsymbol{0} \\ 0 \\ 0 \end{bmatrix},$$

$$J_2(\alpha,:)^T = \begin{bmatrix} (g_2^\alpha, \mathbf{0})_{\partial K} \\ (g_2^\alpha, \phi)_{\partial K} \\ (g_2^\alpha, \boldsymbol{\varphi}_{1,2})_{\partial K} \\ (g_2^\alpha, \boldsymbol{\varphi}_{2,2})_{\partial K} \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ (g_2^\alpha, \phi)_{\partial K} \\ 0 \\ 0 \end{bmatrix},$$

where $(g_1^\alpha, \phi)_{\partial K}$ and $(g_2^\alpha, \phi)_{\partial K}$ can be computed using the assembling technique for FEMs. For example, for $(g_1^\alpha, \phi)_{\partial K}$ there exist three integrals on e:

$$F_i = \int_e g_1^\alpha \phi_i \mathrm{d}s, \quad i = 1, 2, 3,$$

where $e = [a_e, m_e, b_e]$ is an edge with a_e and b_e being the endpoints and m_e the midpoint. By the Simpson's formula,

$$F = \begin{bmatrix} F_1 \\ F_2 \\ F_3 \end{bmatrix} = \frac{h_e}{6} \begin{bmatrix} g_1^\alpha(a_e) \\ g_1^\alpha(b_e) \\ 4g_1^\alpha(m_e) \end{bmatrix}.$$

The above discussion can be realized as follows.

```
% --- second term ---
elem1 = [v1(:), v2(:), v1(:)+Nv]; % elem2dof for [ae, be, me]
Gradmm = cell(2,Nmm);
Gradmm(1,1:Nm) = Gradm;
Gradmm(2,Nm+1:end) = Gradm;
for im = 1:Nm
    Gradmm{1,im+Nm} = @(x,y) [0+0*x, 0+0*x];
    Gradmm{2,im} = @(x,y) [0+0*x, 0+0*x];
end
qmm = cell(1,Nmm);  % q = nu*hK(c2*m2 + c3*m3)
c2 = [Lapm, zeros(1,Nm)];
c3 = [zeros(1,Nm), Lapm];
for im = 1:Nmm
    qmm{im} = @(x,y) pde.nu*hK*(c2(im)*m2(x,y) + c3(im)*m3(x,y));
end
for s = 1:2
    id = (1:NdofBd) + (s-1)*NdofBd;
    for im = 1:Nmm
        pm = @(x,y) pde.nu*Gradmm{s,im}(x,y);
        qa = @(x,y) qmm{im}(x,y);
        F1 = 1/6*(sum(pm(x(v1),y(v1)).*Ne, 2) - qa(x(v1),y(v1)).*Ne(:,s));
        F2 = 1/6*(sum(pm(x(v2),y(v2)).*Ne, 2) - qa(x(v2),y(v2)).*Ne(:,s));
        F3 = 4/6*(sum(pm(xe,ye).*Ne, 2) - qa(xe,ye).*Ne(:,s));
        B(im, id) = accumarray(elem1(:), [F1; F2; F3], [NdofBd, 1]);
    end
end
```

We finally consider the implementation of the constraint. At first glance the L^2 projection is not computable since there is no zero-order moment on K. In fact, the computability can be obtained using the decomposition of polynomial spaces. Let $\boldsymbol{\phi}_i = [\boldsymbol{\phi}_{1,i}, \boldsymbol{\phi}_{1,i}]^T$. Then

$$\boldsymbol{\phi}_{1,i} = \boldsymbol{\phi}_i \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad \boldsymbol{\phi}_{2,i} = \boldsymbol{\phi}_i \cdot \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

It is easy to get

$$\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \nabla p_{k-1} + \boldsymbol{g}_{k-2}^\perp, \quad p_{k-1} = h_K m_2, \quad \boldsymbol{g}_{k-2}^\perp = \mathbf{0},$$

$$\begin{bmatrix}0\\1\end{bmatrix} = \nabla q_{k-1} + \boldsymbol{g}_{k-2}^{\perp}, \quad q_{k-1} = h_K m_3, \quad \boldsymbol{g}_{k-2}^{\perp} = \mathbf{0},$$

which yield

$$\begin{aligned} P_0^K(\boldsymbol{\phi}_{1,i}) &= |K|^{-1}\int_K \boldsymbol{\phi}_i \cdot \nabla p_{k-1}\mathrm{d}x = |K|^{-1}h_K\Big(-\int_K \mathrm{div}\boldsymbol{\phi}_i m_2 \mathrm{d}x + \int_{\partial K} m_2 \boldsymbol{\phi}_i \cdot \boldsymbol{n}\mathrm{d}s\Big),\\ P_0^K(\boldsymbol{\phi}_{2,i}) &= |K|^{-1}\int_K \boldsymbol{\phi}_i \cdot \nabla q_{k-1}\mathrm{d}x = |K|^{-1}h_K\Big(-\int_K \mathrm{div}\boldsymbol{\phi}_i m_3 \mathrm{d}x + \int_{\partial K} m_3 \boldsymbol{\phi}_i \cdot \boldsymbol{n}\mathrm{d}s\Big). \end{aligned} \tag{48}$$

As you can see, their computation is similar to the previous one for $B_{\alpha,i}$ with α fixed. The resulting two row vectors will replace the first and seventh rows of B.

```
% constraint
P0K = zeros(2,NdofA);
P0K(1, end-1) = -1; P0K(2, end) = -1;
m23 = {m2, m3};
for s = 1:2
    mc = m23{s};
    F1 = 1/6*(mc(x(v1),y(v1)).*Ne);  % [n1, n2]
    F2 = 1/6*(mc(x(v2),y(v2)).*Ne);
    F3 = 4/6*(mc(xe,ye).*Ne);
    F = [F1; F2; F3];
    P0K(s, 1:NdofBd) = accumarray(elem1(:), F(:,1), [NdofBd 1]);
    P0K(s, NdofBd+1:2*NdofBd) = accumarray(elem1(:), F(:,2), [NdofBd 1]);
end
P0K = 1/area(iel)*hK*P0K;
% Bs, G, Gs
Bs = B;  Bs([1,7], :) = P0K;
```

Example 7.1. *Let $\Omega = (0,1)^2$. We choose the load term $\boldsymbol{f}$ in such a way that the analytical solution is*

$$\boldsymbol{u}(x,y) = \begin{bmatrix} -\frac{1}{2}\cos(x)^2\cos(y)\sin(y) \\ \frac{1}{2}\cos(y)^2\cos(x)\sin(x) \end{bmatrix}, \quad p(x,y) = \sin(x) - \sin(y).$$

The results are displayed in Fig. 21 and Tab. 3, from which we observe the optimal rates of convergence both for u and p.

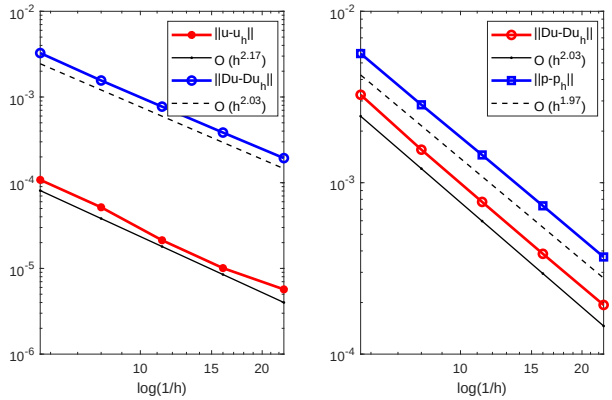


Fig. 21. Convergence rates of the divergence free VEM for the Stokes problem

Tab. 3. The discrete errors for the Stokes problem

♯Dof	h	$\|u-u_h\|$	$\|u-u_h\|_1$	$\|p-p_h\|$
390	1.768e-01	1.07945e-04	3.25802e-03	5.66765e-03
774	1.250e-01	5.16780e-05	1.55844e-03	2.84772e-03
1534	8.839e-02	2.13132e-05	7.72427e-04	1.45175e-03
3042	6.250e-02	1.00841e-05	3.84674e-04	7.33126e-04
6090	4.419e-02	5.69480e-06	1.93534e-04	3.68775e-04

8 Adaptive virtual element methods

Due to the large flexibility of the meshes, researchers have focused on the a posterior error analysis of the VEMs and made some progress in recent years [18–20, 24, 29, 37].

We consider the adaptive VEMs for the Poisson equation. The computable error estimator is taken from [20, 37], given as

$$\eta(u_h) = \Big(\sum_{K\in\mathcal{T}_h} \eta_K^2(u_h) \Big)^{1/2},$$

where

$$\eta_K^2(u_h) = \sum_{i=1}^{4} \eta_{i,K}^2(u_h),$$

with

$$\eta_{1,K}^2 = h_K^2 \|f - \Pi_0^K f\|_{0,K}^2, \quad \eta_{2,K}^2 = h_K^2 \|\Pi_0^K f\|_{0,K}^2, \quad \eta_{3,K}^2 = \|\boldsymbol{\chi}(u_h - \Pi_1^\nabla u_h)\|_{l^2}^2,$$

and

$$\eta_{4,K}^2 = \frac{1}{2} \sum_{e\subset\partial K} h_e \|[\partial_{\boldsymbol{n}} \Pi_1^\nabla u_h]\|_{0,e}^2.$$

Standard adaptive algorithms based on the local mesh refinement can be written as loops of the form

$$\textbf{SOLVE} \rightarrow \textbf{ESTIMATE} \rightarrow \textbf{MARK} \rightarrow \textbf{REFINE}.$$

Given an initial polygonal subdivision $\mathcal{T}_0$, to get $\mathcal{T}_{k+1}$ from $\mathcal{T}_k$ we first solve the VEM problem under consideration to get the numerical solution u_k on $\mathcal{T}_k$. The error is then estimated by using u_k, $\mathcal{T}_k$ and the a posteriori error bound. The local error bound is used to mark a subset of elements in $\mathcal{T}_k$ for refinement. The marked polygons and possible more neighboring elements are refined in such a way that the subdivision meets certain conditions, for example, the resulting polygonal mesh is still shape regular. In the implementation, it is usually time-consuming to write a mesh refinement function since we need to carefully design the rule for dividing the marked elements to get a refined mesh of high quality. We have present such an implementation of the mesh refinement for polygonal meshes in [52].

Consider the Poisson equation with Dirichlet boundary condition on the unit square. The exact solution is given by

$$u(x,y) = xy(1-x)(1-y)\exp\left(-1000((x-0.5)^2 + (y-0.117)^2)\right).$$

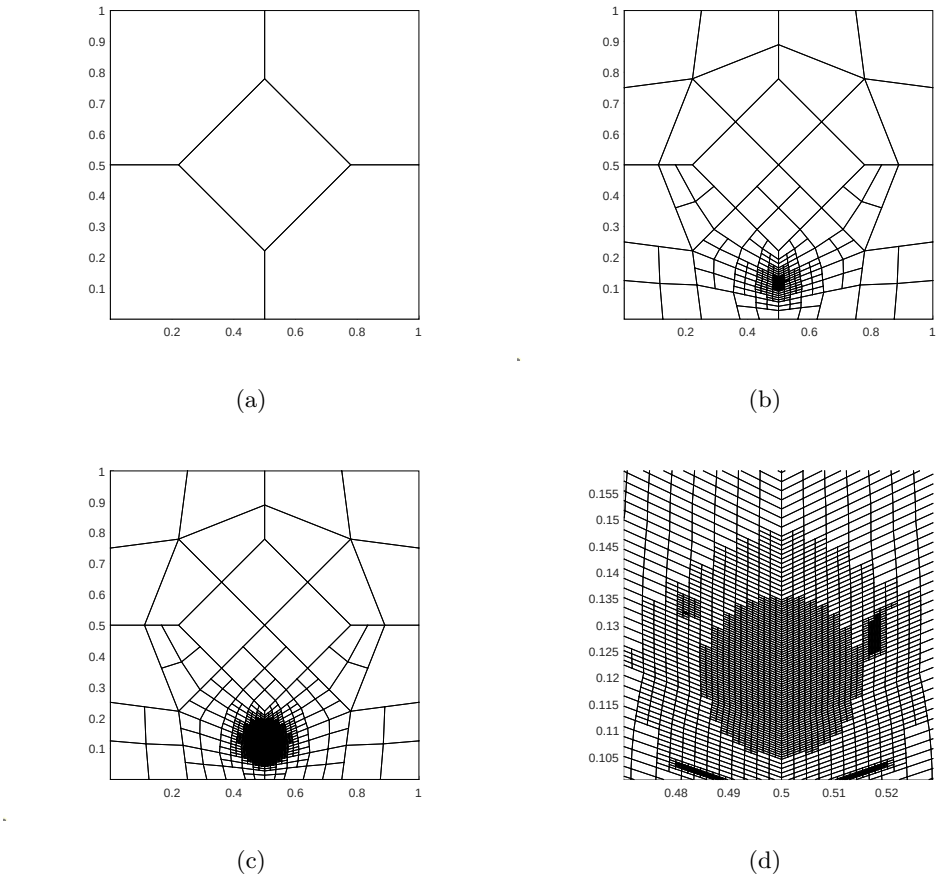
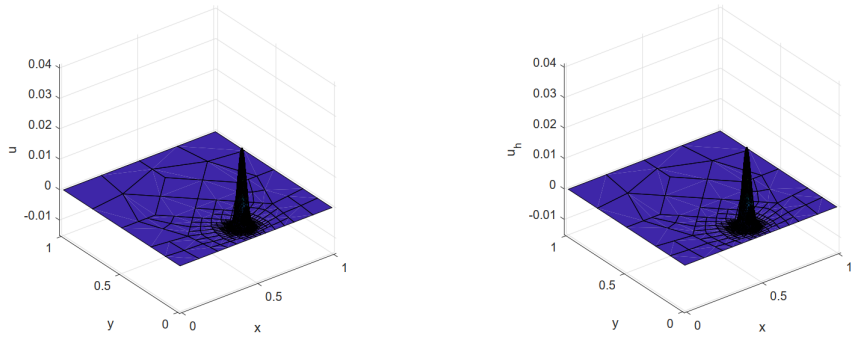


Fig. 22. The initial and the final adapted meshes. (a) The initial mesh; (b) After 20 refinement steps; (c) After 30 refinement steps; (d) The zoomed mesh in (c)



(a) Exact (b) Numerical

Fig. 23. The exact and numerical solutions

We employ the VEM in the lowest order case and use the Dörfler marking strategy with parameter $\theta = 0.4$ to select the subset of elements for refinement. The initial mesh and the final adapted meshes after 20 and 30 refinement steps are presented in Fig. 22 (a-c), respectively. The detail of the last mesh is shown in Fig. 22 (d). Clearly, no small edges are observed. We also plot the adaptive approximation in Fig. 23, which almost coincides with the exact solution. The full code is available from mVEM package. The subroutine `PoissonVEM_indicator.m` is used to compute

the local indicator and the test script is `main_Poisson_avem.m`. As shown in Fig. 24, we see the adaptive strategy correctly refines the mesh in a neighborhood of the singular point and there is a good level of agreement between the H^1 error and error estimator.

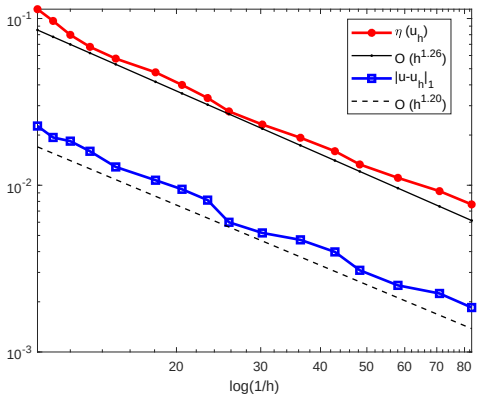


Fig. 24. Convergence rates of the error $|u - \Pi_1^{\nabla} u_h|_1$ and the error estimator $\eta(u_h)$

9 Variational inequalities

We now focus on the the virtual element method to solve a simplified friction problem, which is a typical elliptic variational inequality of the second kind.

9.1 The simplified friction problem

Let $\Omega \subset \mathbb{R}^2$ be a bounded domain with a Lipschitz boundary $\Gamma = \partial\Omega$ that is divided into two parts Γ_C and Γ_D. The problem is

$$\begin{cases} -\Delta u + \alpha u = f & \text{in } \Omega, \\ \partial_{\boldsymbol{n}} u \le g, \quad u\partial_{\boldsymbol{n}} u + g|u| = 0, & \text{on } \Gamma_C, \\ u = 0 \quad \text{on} & \Gamma_D, \end{cases} \tag{49}$$

where $\alpha > 0$ is a constant, $f \in L^2(\Omega)$, $g \in L^2(\Gamma_C)$, Γ_C is the frictional boundary part and Γ_D is the Dirichlet boundary part.

Define

$$V = \{v \in H^1(\Omega) : v|_{\Gamma_D} = 0\}.$$

The variational inequality is [48]: Find $u \in V$ such that

$$a(u, v-u) + j(v) - j(u) \ge \ell(v-u), \quad v \in V, \tag{50}$$

where

$$a(u,v) = \int_\Omega (\nabla u \cdot \nabla v + \alpha uv)\mathrm{d}x, \quad \ell(v) = \int_\Omega fv\mathrm{d}x, \quad j(v) = \int_{\Gamma_C} g|v|\mathrm{d}s.$$

9.2 The VEM discretization

We consider the lowest-order virtual element space. The local space is taken as the enhanced virtual element space $W_1(K)$ defined in (23). Let V_h be the global space. The virtual element method for solving the simplified friction problem is: Find $u_h \in V_h$ such that

$$a_h(u_h, v_h - u_h) + j(v_h) - j(u_h) \geq \ell_h(v_h - u_h), \quad v_h \in V_h, \tag{51}$$

where

$$a_h^K(v,w) = (\nabla\Pi_1^\nabla v, \nabla\Pi_1^\nabla w)_K + \alpha(\Pi_1^0 v, \Pi_1^0 w)_K + S^K(v - \Pi_1^\nabla v, w - \Pi_1^\nabla w),$$
$$S^K(v,w) := (1 + \alpha h_K^2)\boldsymbol{\chi}(v)\cdot\boldsymbol{\chi}(w), \quad \ell_h(v_h) = \sum_{K\in\mathcal{T}_h}(f, \Pi_0^0 v_h)_K.$$

By introducing a Lagrangian multiplier

$$\lambda \in \Lambda = \{\lambda \in L^\infty(\Gamma_C) : |\lambda| \leq 1 \quad \text{a.e. on} \quad \Gamma_C\},$$

the inequality problem (50) can be rewritten as

$$\begin{cases} a(u,v) + \int_{\Gamma_C} g\lambda v \mathrm{d}s = \ell(v), & v \in V, \\ \lambda u = |u| \quad \text{a.e. on} \quad \Gamma_C. \end{cases}$$

For this reason, the discrete problem (51) can be recast as

$$\begin{cases} a_h(u_h, v_h) + \int_{\Gamma_C} g\lambda_h v_h \mathrm{d}s = \ell_h(v_h), & v_h \in V_h, \\ \lambda_h u_h = |u_h| \quad \text{a.e. on} \quad \Gamma_C, \end{cases}$$

where $\lambda_h \in L^\infty(\Gamma_C)$ and $|\lambda_h| \leq 1$. Then the Uzawa algorithm for solving the above problem is [50,51]: given any $\lambda_h^{(0)} \in \Lambda$, for $n \geq 1$, find $u_h^{(n)}$ and $\lambda_h^{(n)}$ by solving

$$a_h(u_h^{(n)}, v_h) = \ell_h(v_h) - \int_{\Gamma_C} g\lambda_h^{(n-1)} v_h \mathrm{d}s \tag{52}$$

and

$$\lambda_h^{(n)} = P_\Lambda(\lambda_h^{(n-1)} + \rho g u_h^{(n)}),$$

where $P_\Lambda(\mu) = \sup\{-1, \inf\{1, \mu\}\}$ and ρ is a constant parameter.

9.3 Numerical example

Let $\Omega = (0,1)^2$ and suppose that the frictional boundary condition is imposed on $y = 0$. The function g can be simply chosen as $\sup_{\Gamma_C}|\partial_{\boldsymbol{n}} u|$. The right-hand function f is chosen such that the exact solution is $u = (\sin(x) - x\sin(1))\sin(2\pi y)$.

The Uzawa iteration stops when $\|\boldsymbol{\chi}(u_h^{n+1} - u_h^n)\|_{l^2} \leq \mathrm{tol}$ or $n \geq \mathrm{maxIt}$. It is evident that the problem (52) is exactly the VEM discretization for the reaction-diffusion problems, with the Neumann boundary data replaced by $g\lambda_h^{(n-1)}$. In addition, we only need to assemble the integral on Γ_C in each iteration. Because of this, we will not give the implementation details. We set $\mathrm{tol} = 10^{-8}$, $\mathrm{maxIt} = 500$ and $\rho = 10$. The results are shown in Figs. 25 and 26, from which we see that the lowest-order VEM achieves the linear convergence order in the discrete H^1 norm, which is optimal according to the a priori error estimate in [48]. The test script is `main_PoissonVEM_VI_Uzawa.m`.

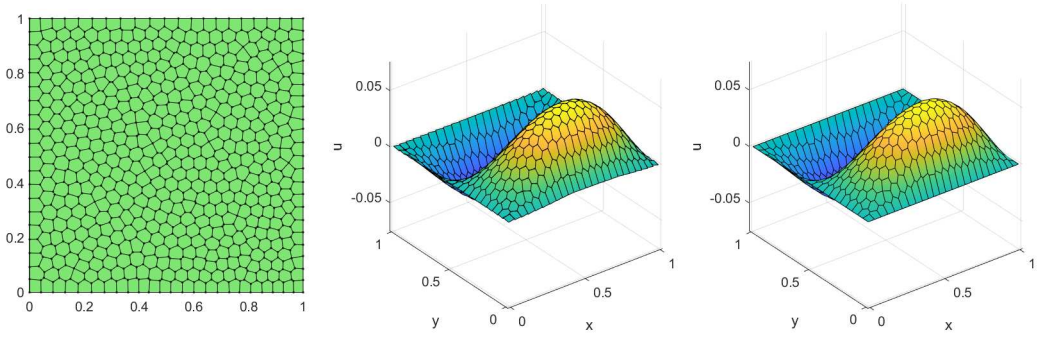


Fig. 25. Numerical and exact solutions for the simplified friction problem ($\alpha = 10^4$)

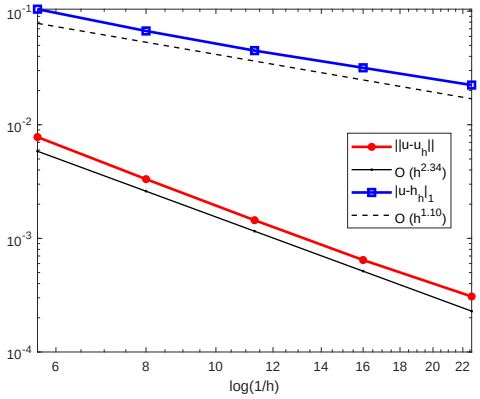


Fig. 26. Convergence rates of the VEM for the simplified friction problem ($\alpha = 10^4$)

10 Three-dimensional problems

In this section we are concerned with the implementation of 3-D VEMs for the Poisson equation.

10.1 Virtual element methods for the 3-D Poisson equation

Let $\Omega \subset \mathbb{R}^3$ be a polyhedral domain and let Γ denote a subset of its boundary consisting of some faces. We consider the following model problem

$$\begin{cases} -\Delta u = r & \text{in } \Omega, \\ u = g_D & \text{on } \Gamma, \\ \partial_n u = g_N & \text{on } \Gamma' = \partial\Omega \backslash \Gamma, \end{cases} \tag{53}$$

where $r \in L^2(\Omega)$ and $g_N \in L^2(\Gamma')$ are the applied load and Neumann boundary data, respectively, and $g_D \in H^{1/2}(\Gamma)$ is the Dirichlet boundary data function.

In what follows, we use K to represent the generic polyhedral element with $f \subset \partial K$ being its generic face. The vertices of a face f are in a counterclockwise order when viewed from the inside. The virtual element method proposed in [15] for (53) is to find $u_h \in V_\Gamma^k$ such that

$$a_h(u_h, v_h) = \ell_h(v_h), \quad v_h \in V_0^k,$$

where

$$a_h(u_h, v_h) = \sum_{K\in\mathcal{T}_h} a_h^K(u_h, v_h), \quad \ell_h(v_h) = \int_\Omega r_h v_h \mathrm{d}x + \int_{\Gamma'} g_h v_h \mathrm{d}s.$$

The local bilinear form is split into two parts:

$$a_h^K(v, w) = (\nabla v, \nabla w)_K + h_K S^K(v - \Pi_k^\nabla v, w - \Pi_k^\nabla w),$$

where $\Pi_k^\nabla : V^k(K) \to \mathbb{P}_k(K)$ is the standard elliptic projection, and S^K is the stabilization term given as

$$S^K(v, w) = \sum_{i=1}^{N^K} \chi_i(v)\chi_i(w),$$

where $\chi_i(v) = v(p_i)$ and p_i is the i-th vertex of K for $i = 1, 2, \cdots, N_K$. The local linear form of the right-hand side will be approximated as

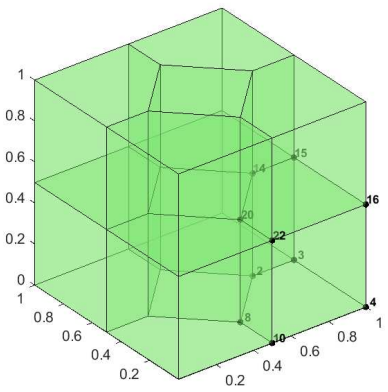
$$\ell_h^K(v_h) = \int_K r\Pi_k^\nabla v_h \mathrm{d}x + \sum_{f\subset\Gamma'\cap\partial K} \int_f g_N \Pi_{k,f}^\nabla v_h \mathrm{d}s,$$

where $\Pi_{k,f}^\nabla : V^k(f) \to \mathbb{P}_k(f)$ is the elliptic projector defined on the face f.

For the detailed introduction of the virtual element spaces, please refer to Section 2 in [15]. In this paper, we only consider the lowest order case $k = 1$, but note that the hidden ideas can be directly generalized to higher order cases.

10.2 Data structure and test script

We first discuss the data structure to represent polyhedral meshes. In the implementation, the mesh is represented by `node3` and `elem3`. The `N` × 3 matrix `node3` stores the coordinates of all vertices in the mesh. `elem3` is a cell array with each entry storing the face connectivity, for example, the first entry `elemf = elem3{1}` for the mesh given in Fig. 27(a) is shown in Fig. 27(b), which is still represented by a cell array since the faces may have different numbers of vertices.



(a) Polyhedral mesh

elem3{1, 1}

	1
1	[8,10,4,3,2]
2	[14,15,16,22,20]
3	[10,8,20,22]
4	[4,10,22,16]
5	[3,4,16,15]
6	[2,3,15,14]
7	[8,2,14,20]

(b) Representation of the first element

Fig. 27. Polyhedral mesh of a cube

All faces including the repeated internal ones can be gathered in a cell array as

```
allFace = vertcat(elem3{:}); %cell
```

By padding the vacancies and using the `sort` and `unique` functions to rows, we obtain the face set `face`. The cell array `elem2face` then establishes the map of local index of faces in each polyhedron to its global index in face set `face`. The above structural information is summarized in the subroutine `auxstructure3.m`. The geometric quantities such as the diameter `diameter3`, the barycenter `centroid3` and the volume `volume` are computed by `auxgeometry3.m`. We remark that these two subroutines may be needed to add more information when dealing with higher order VEMs.

The test script is `main_PoissonVEM3.m` listed as follows. In the `for` loop, we first load the pre-defined mesh data, which immediately returns the matrix `node3` and the cell array `elem3` to the MATLAB workspace. Then we set up the Neumann boundary conditions to get the structural information of the boundary faces. The subroutine `PoissonVEM3.m` is the function file containing all source code to implement the 3-D VEM. When obtaining the numerical solutions, we calculate the discrete L^2 errors and H^1 errors defined as

$$\mathrm{ErrL2} = \sum_{K\in\mathcal{T}_h} \|u - \Pi_1^\nabla u_h\|_{0,K}, \quad \mathrm{ErrH1} = \sum_{K\in\mathcal{T}_h} |u - \Pi_1^\nabla u_h|_{1,K}$$

by using respectively the subroutine `getError3.m`. The procedure is completed by verifying the rate of convergence through `showrateErr.m`.

```
%% Parameters
maxIt = 5;
h = zeros(maxIt,1);       N = zeros(maxIt,1);
ErrL2 = zeros(maxIt,1);   ErrH1 = zeros(maxIt,1);

%% PDE data
pde = Poisson3data();
bdNeumann = 'x==0'; % string for Neumann

%% Virtual element method
for k = 1:maxIt
    % load mesh
    fprintf('Mesh %d: \n', k);
    load( ['SimpleMesh3data', num2str(k), '.mat'] ); % polyhedral mesh
    %load( ['mesh3data', num2str(k), '.mat'] ); % polyhedral mesh
    %[node3,¬,elem3] = cubemesh([0 1 0 1 0 1], 1/(2*k)); % tetrahedral mesh
    % get boundary information
    bdStruct = setboundary3(node3,elem3,bdNeumann);
    % solve
    [uh,info] = PoissonVEM3(node3,elem3,pde,bdStruct);
    % record
    N(k) = length(uh);  h(k) = (1/size(elem3,1))^(1/3);
    % compute errors in discrete L2, H1 and energy norms
    kOrder = 1;
    [ErrH1(k),ErrL2(k)] = getError3(node3,elem3,uh,info,pde,kOrder);
end

%% Plot convergence rates and display error table
figure, showrateErr(h,ErrL2,ErrH1);

fprintf('\n');
```

```
32 disp('Table: Error')
33 colname = {'#Dof','h','||u-u_h||','|u-u_h|_1'};
34 disptable(colname,N,[],h,'%0.3e',ErrL2,'%0.5e',ErrH1,'%0.5e');
```

In the following subsections, we shall go into the details of the implementation of the 3-D VEM in `PoissonVEM3.m`.

10.3 Elliptic projection on polygonal faces

Let f be a face of K or a polygon embedded in $\mathbb{R}^3$. In the VEM computing, we have to get all elliptic projections $\Pi_{1,f}^{\nabla}\phi_f^T$ ready in advance, where ϕ_f is the nodal basis of the enhanced virtual element space $V^1(f)$ (see Subsection 2.3 in [15]). To this end, it may be necessary to establish local coordinates (s,t) on the face f.

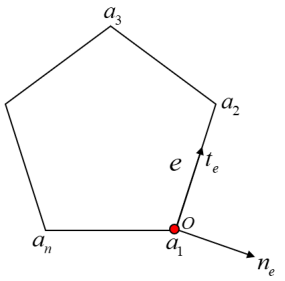


Fig. 28. Local coordinate system of a face or polygon embedded in $\mathbb{R}^3$

As shown in Fig. 28, the boundary of the polygon is oriented in a counterclockwise order as $a_1, \cdots, a_n$. Let $e = a_1a_2$ be the first edge, and n_e and t_e be the normal vector and tangential vector, respectively. Then we can define a local coordinate system with a_1 being the original point by using these two vectors. Let $n_e = (n_1, n_2, n_3)$ and $t_e = (t_1, t_2, t_3)$. For any $a = (x, y, z) \in f$, its local coordinate (s,t) is related by

$$\overrightarrow{Oa} = s \cdot n_e + t \cdot t_e, \quad \text{or} \quad (x - x_1, y - y_1, z - z_1) = s \cdot (n_1, n_2, n_3) + t \cdot (t_1, t_2, t_3),$$

which gives

$$(s,t) = (x - x_1, y - y_1, z - z_1) \begin{bmatrix} n_1 & n_2 & n_3 \\ t_1 & t_2 & t_2 \end{bmatrix}^{-1},$$

with the inverse understood in the least squares sense. When converting to the local coordinate system, we can compute all the matrices of elliptic projection in the same way for the Poisson equation in two-dimensional cases. For completeness, we briefly recall the implementation. In what follows, we use the subscript "f" to indicate the locally defined symbols.

Let $\phi_f^T = [\phi_{f,1}, \cdots, \phi_{f,n}]$ be the basis functions of $V^1(f)$ and $m_f^T = [m_{f,1}, m_{f,2}, m_{f,3}]$ the scaled monomials on f given as

$$m_{f,1} = 1, \quad m_{f,2} = \frac{s - s_f}{h_f}, \quad m_{f,3} = \frac{t - t_f}{h_f},$$

where (s_f, t_f) and h_f are the barycenter and the diameter of f, respectively. The vector form of

the elliptic projector $\Pi_{1,f}^{\nabla}$ can be represented as

$$\begin{cases} (\nabla_f m_f, \nabla_f \Pi_{1,f}^{\nabla} \phi_f^T)_f = (\nabla_f m_f, \nabla_f \phi_f^T)_f, \\ P_0(\Pi_{1,f}^{\nabla} \phi_f^T) = P_0(\phi_f^T). \end{cases} \tag{54}$$

where

$$P_0(v) = \frac{1}{n} \sum_{i=1}^{n} v(a_i).$$

Since $\mathbb{P}_1(f) \subset V^1(f)$, we can write

$$m_f^T = \phi_f^T \boldsymbol{D}_f, \qquad \boldsymbol{D}_f = (D_{i\alpha}), \quad D_{i\alpha} = \chi_{f,i}(m_{f,\alpha}),$$

where $\chi_{f,i}$ is the i-th d.o.f associated with a_i, and $\boldsymbol{D}_f$ is referred to as the transition matrix. We further introduce the following expansions

$$\Pi_{1,f}^{\nabla} \phi_f^T = \phi_f^T \boldsymbol{\Pi}_{1,f}^{\nabla}, \qquad \Pi_{1,f}^{\nabla} \phi_f^T = m_f^T \boldsymbol{\Pi}_{1*,f}^{\nabla}.$$

One easily finds that

$$\boldsymbol{\Pi}_{1,f}^{\nabla} = \boldsymbol{D}_f \boldsymbol{\Pi}_{1*,f}^{\nabla},$$

and (54) can be rewritten in matrix form as

$$\begin{cases} \boldsymbol{G}_f \boldsymbol{\Pi}_{1*,f}^{\nabla} = \boldsymbol{B}_f, \\ P_0(m_f^T) \boldsymbol{\Pi}_{1*,f}^{\nabla} = P_0(\phi_f^T) \end{cases}, \quad \text{or denoted by} \quad \tilde{\boldsymbol{G}}_f \boldsymbol{\Pi}_{1*,f}^{\nabla} = \tilde{\boldsymbol{B}}_f,$$

where

$$\boldsymbol{G}_f = (\nabla_f m_f, \nabla_f m_f^T)_f, \quad \boldsymbol{B}_f = (\nabla_f m_f, \nabla_f \phi_f^T)_f.$$

Note that the following consistency relation holds

$$\boldsymbol{G}_f = \boldsymbol{B}_f \boldsymbol{D}_f, \quad \tilde{\boldsymbol{G}}_f = \tilde{\boldsymbol{B}}_f \boldsymbol{D}_f.$$

Let `face` be the face set with internal faces repeated once. Then using the local coordinates we are able to derive all elliptic projections $\Pi_{1,f}^{\nabla} \phi_f^T$ as in 2-D cases. It is not recommended to carry out the calculation element by element in view of the repeated cost for the internal faces.

The above discussion is summarized in a subroutine with input and output as

```
Pifs = faceEllipticProjection(P),
```

where `P` is the coordinates of the face f and `Pifs` is the matrix representation $\boldsymbol{\Pi}_{1*,f}^{\nabla}$ of $\Pi_{1,f}^{\nabla} \phi_f^T$ in the basis m_f^T. One can derive all matrices by looping over the face set `face`:

```
%% Derive elliptic projections of all faces
faceProj = cell(NF,1);
for s = 1:NF
    % Pifs
    faces = face{s};  P = node3(faces,:);
    Pifs = faceEllipticProjection(P);
    % sort the columns
    [~,idx] = sort(faces);
    faceProj{s} = Pifs(:,idx);
end
```

Note that in the last step we sort the columns of $\mathbf{\Pi}^{\nabla}_{1*,f}$ in ascending order according to the numbers of the vertices. In this way we can easily find the correct correspondence on each element (see Lines 32-35 in the code of the next subsection).

The face integral is then given by

$$\int_f \Pi^{\nabla}_{1,f}\phi_f^T \mathrm{d}\sigma = \int_f m_f^T \mathrm{d}\sigma \mathbf{\Pi}^{\nabla}_{1*,f} = (|f|,0,0)\mathbf{\Pi}^{\nabla}_{1*,f}, \tag{55}$$

where $|f|$ is the area of f and the definition of the barycenter is used.

10.4 Elliptic projection on polyhedral elements

The 3-D scaled monomials $m^T = [m_1, m_2, m_3, m_4]$ are

$$m_1 = 1, \quad m_2 = \frac{x - x_K}{h_K}, \quad m_3 = \frac{y - y_K}{h_K}, \quad m_4 = \frac{z - z_K}{h_K},$$

where (x_K, y_K, z_K) is the centroid of K and h_K is the diameter, and the geometric quantities are computed by the subroutine `auxgeometry3.m`. Similar to the 2-D case, we have the symbols $\boldsymbol{D}, \boldsymbol{G}, \tilde{\boldsymbol{G}}, \boldsymbol{B}$ and $\tilde{\boldsymbol{B}}$. For example, the transition matrix is given by

$$\boldsymbol{D} = (D_{i\alpha}), \quad D_{i\alpha} = \chi_i(m_\alpha) = m_\alpha(p_i).$$

The most involved step is to compute the matrix

$$\begin{aligned}\boldsymbol{B} &= \int_K \nabla m \cdot \nabla \phi^T \mathrm{d}x = -\int_K \Delta m \cdot \phi^T \mathrm{d}x + \sum_{f\subset\partial K}\int_f (\nabla m \cdot \boldsymbol{n}_f)\phi^T \mathrm{d}\sigma \\ &= \sum_{f\subset\partial K}\int_f (\nabla m \cdot \boldsymbol{n}_f)\phi^T \mathrm{d}\sigma,\end{aligned}$$

where $\phi^T = [\phi_1, \phi_2, \cdots, \phi_{N_K}]$ are the basis functions with ϕ_i associated with the vertex p_i of K. According to the definition of $V^1(f)$, one has

$$\int_f (\nabla m \cdot \boldsymbol{n}_f)\phi^T \mathrm{d}\sigma = (\nabla m \cdot \boldsymbol{n}_f)\int_f \phi^T \mathrm{d}\sigma = (\nabla m \cdot \boldsymbol{n}_f)\int_f \Pi^{\nabla}_{1,f}\phi^T \mathrm{d}\sigma,$$

and the last term is available from (55). Obviously, for the vertex p_i away from the face f there holds $\Pi^{\nabla}_{1,f}\phi_i = 0$. In the following code, `indexEdge` gives the row index in the face set `face` for each face of `elemf`, and `iel` is the index for looping over the elements.

```
1   % ------- element information --------
2   % faces
3   elemf = elem3{iel};   indexFace = elem2face{iel};
4   % global index of vertices and local index of elemf
5   [¬,index3,¬,elemfLocal] = faceTriangulation(elemf);
6   % centroid and diameter
7   Nv = length(index3);  Ndof = Nv;
8   V = node3(index3,:);
9   xK = centroid3(iel,1); yK = centroid3(iel,2); zK = centroid3(iel,3);
10  hK = diameter3(iel);
11  x = V(:,1);  y = V(:,2);  z = V(:,3);
12
13  % ------- scaled monomials ----------
```

```
    m1 = @(x,y,z) 1+0*x;
    m2 = @(x,y,z) (x-xK)/hK;
    m3 = @(x,y,z) (y-yK)/hK;
    m4 = @(x,y,z) (z-zK)/hK;
    m = @(x,y,z) [m1(x,y,z),m2(x,y,z),m3(x,y,z),m4(x,y,z)]; % m1,m2,m3,m4
    mc = {m1,m2,m3,m4};
    gradmMat = [0 0 0; 1/hK 0 0; 0 1/hK 0; 0 0 1/hK];

    % -------- transition matrix ----------
    D = m(x,y,z);

    % ----------- elliptic projection -------------
    B = zeros(4,Ndof);
    for s = 1:size(elemf,1)
        % --- information of current face
        % vertices of face
        faces = elemf{s};  P = node3(faces,:);
        % elliptic projection on the face
        idFace = indexFace(s);
        Pifs = faceProj{idFace}; % the order may be not correct
        [¬,¬,idx] = unique(faces);
        Pifs = Pifs(:,idx);
        % normal vector
        e1 = P(2,:)-P(1,:);  en = P(1,:)-P(end,:);
        nf = cross(e1,en); nf = nf./norm(nf);
        % area
        areaf = polyarea3(P);
        % --- integral of Pifs
        intFace = [areaf,0,0]*Pifs;  % local
        intProj = zeros(1,Ndof);  % global adjustment
        faceLocal = elemfLocal{s};
        intProj(faceLocal) = intFace;
        % add grad(m)*nf
        Bf = dot(gradmMat, repmat(nf,4,1), 2)*intProj;
        B = B + Bf;
    end
    % constraint
    Bs = B;  Bs(1,:) = 1/Ndof;
    % consistency relation
    G = B*D;  Gs = Bs*D;
```

10.5 Computation of the right hand side and assembly of the linear system

The right-hand side is approximated as

$$F_K = \int_K f\Pi_1^\nabla \phi \mathrm{d}x = (\mathbf{\Pi}_{1*}^\nabla)^T \int_K f m \mathrm{d}x,$$

where Π_1^∇ is the elliptic projector on the element K and $\mathbf{\Pi}_{1*}^\nabla$ is the matrix representation in the basis m^T. The integral $\int_K f m \mathrm{d}x$ can be approximated by

$$\int_K f m \mathrm{d}x = |K| f(\boldsymbol{x}_K) m(\boldsymbol{x}_K) = |K| f(\boldsymbol{x}_K)[1,0,0,0]^T, \quad \boldsymbol{x}_K = (x_K, y_K, z_K).$$

One can also divide the element K as a union of some tetrahedrons and compute the integral using the Gaussian rule. Please refer to the subroutine `integralPolyhedron.m` for illustration.

One easily finds that the stiffness matrix for the bilinear form is

$$\boldsymbol{A}_K = (\boldsymbol{\Pi}_{1*}^{\nabla})^T \boldsymbol{G} \boldsymbol{\Pi}_{1*}^{\nabla} + h_K (\boldsymbol{I} - \boldsymbol{\Pi}_1^{\nabla})^T (\boldsymbol{I} - \boldsymbol{\Pi}_1^{\nabla}).$$

We compute the elliptic projections in the previous section and provide the assembly index element by element. Then the linear system can be assembled using the MATLAB function `sparse` as follows.

```
for iel = 1:NT

    ...

    % --------- local stiffness matrix ---------
    Pis = Gs\Bs;   Pi  = D*Pis;   I = eye(size(Pi));
    AK  = Pis'*G*Pis + hK*(I-Pi)'*(I-Pi);
    AK = reshape(AK,1,[]);  % straighten

    % --------- load vector -----------
    %fK = Pis'*[pde.f(centroid3(iel,:))*volume(iel);0;0;0];
    fun = @(x,y,z) repmat(pde.f([x,y,z]),1,4).*m(x,y,z);
    fK = integralPolyhedron(fun,3,node3,elemf);
    fK = Pis'*fK(:);

    % --------- assembly index for ellptic projection -----------
    indexDof = index3;
    ii(ia+1:ia+Ndof^2) = reshape(repmat(indexDof, Ndof, 1), [], 1);
    jj(ia+1:ia+Ndof^2) = repmat(indexDof(:), Ndof, 1);
    ss(ia+1:ia+Ndof^2) = AK(:);
    ia = ia + Ndof^2;

    % --------- assembly index for right hand side -----------
    elemb(ib+1:ib+Ndof) = indexDof(:);
    Fb(ib+1:ib+Ndof) = fK(:);
    ib = ib + Ndof;

    % --------- matrix for L2 and H1 error evaluation  ---------
    Ph{iel} = Pis;
    elem2dof{iel} = indexDof;
end
kk = sparse(ii,jj,ss,N,N);
ff = accumarray(elemb,Fb,[N 1]);
```

Note that we have stored the matrix representation $\boldsymbol{\Pi}_{1*}^{\nabla}$ and the assembly index `elem2dof` in the M-file so as to compute the discrete L^2 and H^1 errors.

10.6 Treatment of the boundary conditions

We first consider the Neumann boundary conditions. Let f be a boundary face with n vertices. The local load vector is

$$F_f = \int_f g_N \Pi_{1,f}^{\nabla} \phi_f \mathrm{d}\sigma = (\boldsymbol{\Pi}_{1*,f}^{\nabla})^T \int_f g_N m_f \mathrm{d}\sigma,$$

where $g_N = \partial_{\boldsymbol{n}_f} u = \nabla u \cdot \boldsymbol{n}_f$. For simplicity, we provide the gradient $g_N = \nabla u$ in the PDE data instead and compute the true g_N in the M-file. Note that the above integral can be transformed to a 2-D problem by using the local coordinate system as done in the following code,

where `localPolygon3.m` realizes the transformation and returns some useful information, and `integralPolygon.m` calculates the integral on a 2-D polygon.

```
%% Assemble Neumann boundary conditions
bdFaceN = bdStruct.bdFaceN;  bdFaceIdxN = bdStruct.bdFaceIdxN;
if ¬isempty(bdFaceN)
    Du = pde.Du;
    faceLen = cellfun('length',bdFaceN);
    nnz = sum(faceLen);
    elemb = zeros(nnz,1); FN = zeros(nnz,1);
    ib = 0;
    for s = 1:size(bdFaceN,1)
        % vertices of face
        faces = bdFaceN{s};   nv = length(faces);
        P = node3(faces,:);
        % elliptic projection on the face
        idFace = bdFaceIdxN(s);
        Pifs = faceProj{idFace}; % the order may be not correct
        [¬,idx] = sort(faces);
        Pifs = Pifs(:,idx);
        % 3-D polygon -> 2-D polygon
        poly = localPolygon3(P);
        nodef = poly.nodef;  % local coordinates
        nf = poly.nf;   % outer normal vector of face
        centroidf = poly.centroidf;
        sc = centroidf(1); tc = centroidf(2);
        hf = poly.diameterf;
        Coord = poly.Coord; % (s,t) ---> (x,y,z)
        % g_N
        g_N = @(s,t) dot(Du(Coord(s,t)), nf);
        fun = @(s,t) g_N(s,t)*[1+0*s, (s-sc)/hf, (t-tc)/hf];
        Ff = integralPolygon(fun,3,nodef);
        Ff = Pifs'*Ff(:);
        % assembly index
        elemb(ib+1:ib+nv) = faces(:);
        FN(ib+1:ib+nv) = Ff(:);
        ib = ib + nv;
    end
    ff = ff + accumarray(elemb(:), FN(:),[N 1]);
end
```

The Dirichlet boundary conditions are easy to handle. The code is given as follows.

```
%% Apply Dirichlet boundary conditions
g_D = pde.g_D;  bdNodeIdxD = bdStruct.bdNodeIdxD;
isBdNode = false(N,1); isBdNode(bdNodeIdxD) = true;
bdDof = (isBdNode); freeDof = (¬isBdNode);
nodeD = node3(bdDof,:);
u = zeros(N,1); uD = g_D(nodeD); u(bdDof) = uD(:);
ff = ff - kk*u;
```

In the above codes, `bdStruct` stores all necessary information of boundary faces. We finally derive the linear system `kk*uh = ff`, where `kk` is the resulting coefficient matrix and `ff` is the right-hand side. For small scale linear system, we directly solve it using the backslash command in MATLAB, while for large systems the algebraic multigrid method is used instead.

```
%% Set solver
solver = 'amg';
if N < 2e3, solver = 'direct'; end
% solve
switch solver
    case 'direct'
        u(freeDof) = kk(freeDof,freeDof)\ff(freeDof);
    case 'amg'
        option.solver = 'CG';
        u(freeDof) = amg(kk(freeDof,freeDof),ff(freeDof),option);
end
```

Here, the subroutine `amg.m` can be found in *i*FEM — a MATLAB software package for the finite element methods [26].

The complete M-file is `PoissonVEM3.m`. The overall structure of a virtual element method implementation will be much the same as for a standard finite element method, as outlined in Algorithm 2.

Algorithm 2 An overall structure of the implementation of a 3-D virtual element method

Input: Mesh data and PDE data

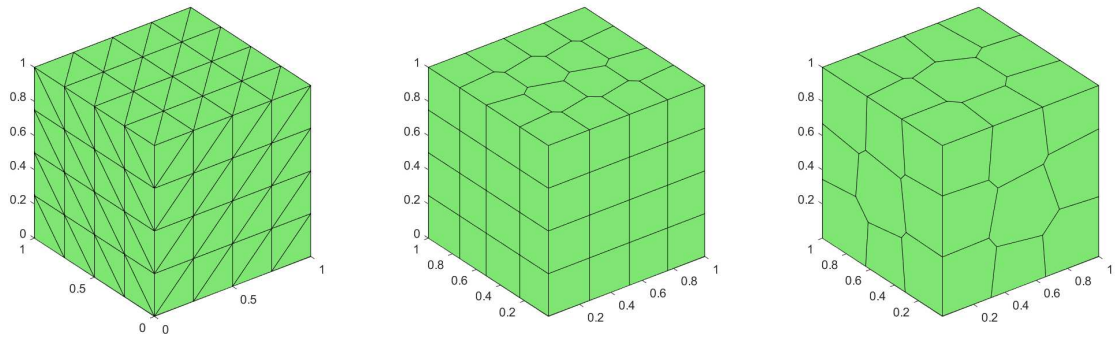
1. Get auxiliary data of the mesh, including some data structures and geometric quantities;
2. Derive elliptic projections of all faces;
3. Compute and assemble the linear system by looping over the elements;
4. Assemble Neumann boundary conditions and apply Dirichlet boundary conditions;
5. Set solver and store information for computing errors.

Output: The numerical DoFs

10.7 Numerical examples

It should be pointed out that all examples in this subsection are implemented in MATLAB R2019b. The domain Ω is always taken as the unit cube $(0,1)^3$ with the Neumann boundary condition imposed on $x = 0$.

We solve the problem on three different kinds of meshes. One is the uniform triangulation shown in Fig. 29(a) and the others are the polyhedral meshes displayed in Fig. 29(b) and Fig. 29(c). The structured polyhedral mesh in Fig. 29(b) is formed by translating a two-dimensional polygonal mesh along the z-axis and connecting the corresponding vertices, hence all sides are quadrilaterals. The CVT meshes refer to the Centroidal Voronoi Tessellations, which are obtained from a set of seeds that coincide with barycenters of the resulting Voronoi cells. We generate such meshes via a standard Lloyd algorithm by extending the idea in the MATLAB toolbox - PolyMesher introduced in [45] to a cuboid.



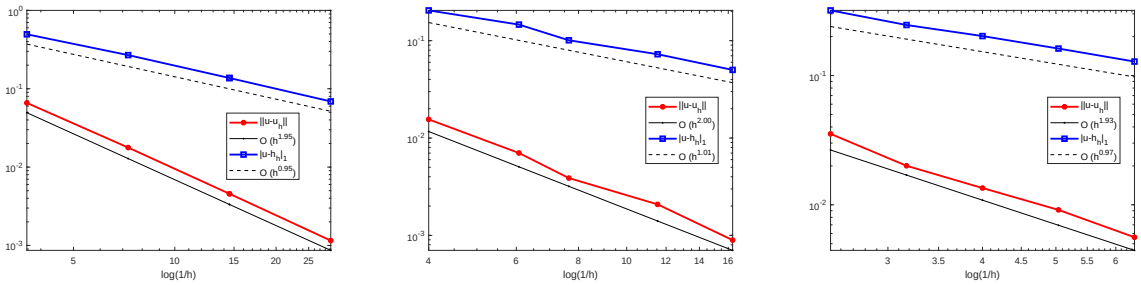
(a) Tetrahedral mesh (b) Structured mesh (c) CVT mesh

Fig. 29. Three types of discretizations

Example 10.1. *Let $\alpha = 1$. The right-hand side f and the boundary conditions are chosen in such a way that the exact solution is*

$$u(x, y, z) = \sin(2xy)\cos(z).$$

The results are displayed in Fig. 30, from which we observe that the optimal rates of convergence are achieved for all the three types of discretizations in the H^1 and L^2 norms. Note that the uniform triangulation is generated by `cubemesh.m` in *i*FEM.



(a) Tetrahedral mesh (b) Structured mesh (c) CVT mesh

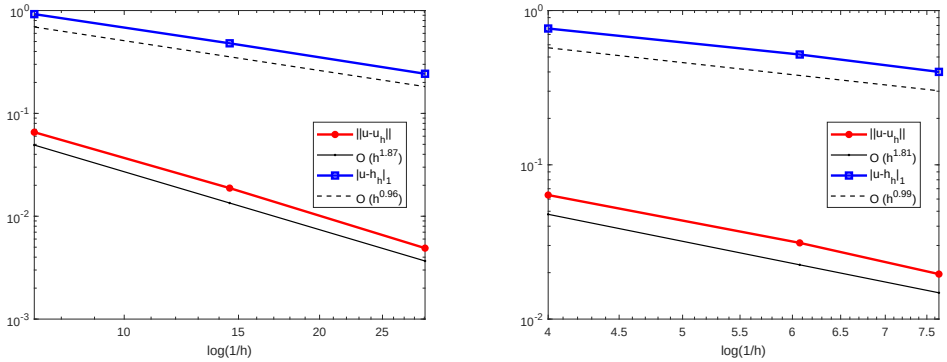
Fig. 30. Convergence rates in L^2 and H^1 norms for Example 10.1

Example 10.2. *In this example, the exact solution is chosen as*

$$u(x, y, z) = \sin(\pi x)\cos(\pi y)\cos(\pi z).$$

Tab. 4. Discrete L^2 and H^1 errors of Example 10.2 for polyhedral meshes

N	h	ErrL2	ErrH1
170	2.500e-01	6.32804e-02	7.63490e-01
504	1.647e-01	3.09424e-02	5.18111e-01
1024	1.307e-01	1.94670e-02	4.00026e-01



(a) Tetrahedral mesh (b) Structured mesh

Fig. 31. Convergence rates in L^2 and H^1 norms for Example 10.2

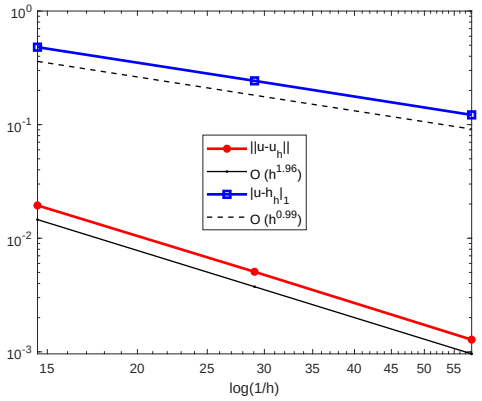


Fig. 32. Convergence rates in L^2 and H^1 norms of Example 10.2 for the triangulation with smaller mesh sizes

We still consider the problem with the Neumann boundary condition imposed on $x = 0$ and repeat the numerical simulation in Example 10.1. The uniform triangulation is generated by `cubemesh.m` and further refined by `uniformrefine3.m` in *i*FEM. By default, the initial mesh before refining has mesh size $h = 1$. We display the rate of convergence and the discrete errors in Fig. 31 and Tab. 4, respectively. It is evident that the code gives satisfactory accuracy and the optimal rate of convergence is achieved for the H^1 norm. However, the order of the error in the discrete L^2 norm is not close to 2. In fact, this phenomenon is also observed for the classical linear finite element methods under the same conditions. The reason lies in the coarse mesh. To get the optimal convergence rate, one can run the code on a sequence of meshes with much smaller sizes. For instance, Fig. 32 shows the result for the triangulation with initial size 0.25, in which case the optimal rates of convergence are obtained for both norms. It should be pointed out that the linear virtual element method on a triangulation is exactly the standard finite element method since in this case the virtual element space happens to be the set of polynomials of degree $k \leq 1$. Compared to the vectorized code in *i*FEM for the finite element methods, the current implementation is less efficient due to the large `for` loop. For example, three uniform refinements of the above initial triangulation will yield a mesh of 196608 triangular elements, and hence 196608 loops over the

elements.

11 Concluding remarks

In this paper, a MATLAB software package for the virtual element method was presented for various typical problems. The usage of the library, named mVEM, was demonstrated through several examples. Possible extensions of this library that are of interest include time-dependent problems, adaptive mixed VEMs, three-dimensional linear elasticity, polyhedral mesh generator, and nonlinear solid mechanics. Various applications such as the Cahn-Hilliard problem, Stokes-Darcy problem and Navier-Stokes are also very appealing.

For three-dimensional problems, the current code can be further vectorized to achieve comparable performance in MATLAB with respect to compiled languages, where the most time-consuming part lies in the evaluation of the large number of face integrals and element integrals, for example, the elementwise computation of the matrix $\boldsymbol{H}$ for the L^2 projections. To spare the computational cost, one may divide the polyhedral element as a union of some tetrahedrons and compute the integrals on those elements with the same number of tetrahedrons. Such a procedure can be vectorized in MATLAB with an additional effort of code design. In the current version, we still do the elementwise loop as in [44] to make the code more transparent.

mVEM is free and open source software.

References

[1] D. Adak and S. Natarajan. Virtual element method for semilinear sine-Gordon equation over polygonal mesh using product approximation technique. *Math. Comput. Simulation*, 172:224–243, 2020.

[2] B. Ahmad, A. Alsaedi, F. Brezzi, L. D. Marini, and A. Russo. Equivalent projectors for virtual element methods. *Comput. Math. Appl.*, 66(3):376–391, 2013.

[3] P. F. Antonietti, G. Manzini, and M. Verani. The fully nonconforming virtual element method for biharmonic problems. *Math. Models Methods Appl. Sci.*, 28(2):387–407, 2018.

[4] E. Artioli, L. Beirão da Veiga, C. Lovadina, and E. Sacco. Arbitrary order 2D virtual elements for polygonal meshes: part I, elastic problem. *Comput. Mech.*, 60(3):355–377, 2017.

[5] E. Artioli, L. Beirão da Veiga, C. Lovadina, and E. Sacco. Arbitrary order 2D virtual elements for polygonal meshes: part II, inelastic problem. *Comput. Mech.*, 60(4):643–657, 2017.

[6] B. Ayuso de Dios, K. Lipnikov, and G. Manzini. The nonconforming virtual element method. *ESAIM Math. Model. Numer. Anal.*, 50(3):879–904, 2016.

[7] L. Beirão da Veiga, F. Brezzi, A. Cangiani, G. Manzini, L. D. Marini, and A. Russo. Basic principles of virtual element methods. *Math. Models Meth. Appl. Sci.*, 23(1):199–214, 2013.

[8] L. Beirão da Veiga, F. Brezzi, A. Cangiani, G. Manzini, L. D. Marini, and A. Russo. Basic principles of mixed virtual element methods. *ESAIM Math. Model. Numer. Anal.*, 48(4):1227–1240, 2014.

[9] L. Beirão da Veiga, F. Brezzi, F. Dassi, L. D. Marini, and A. Russo. Serendipity virtual elements for general elliptic equations in three dimensions. *Chinese Ann. Math. Ser. B*, 39(2):315–334, 2018.

[10] L. Beirão da Veiga, F. Brezzi, and L. D. Marini. Virtual elements for linear elasticity problems. *SIAM J. Numer. Anal.*, 51(2):794–812, 2013.

[11] L. Beirão da Veiga, F. Brezzi, L. D. Marini, and A. Russo. The hitchhiker's guide to the virtual element method. *Math. Models Meth. Appl. Sci.*, 24(8):1541–1573, 2014.

[12] L. Beirão da Veiga, F. Brezzi, L. D. Marini, and A. Russo. Serendipity nodal VEM spaces. *Comput. & Fluids*, 141:2–12, 2016.

[13] L. Beirão da Veiga, F. Brezzi, L. D. Marini, and A. Russo. Virtual element implementation for general elliptic equations. In *Building bridges: connections and challenges in modern approaches to numerical partial differential equations*, volume 114 of *Lect. Notes Comput. Sci. Eng.*, pages 39–71. Springer, [Cham], 2016.

[14] L. Beirão da Veiga, F. Brezzi, L. D. Marini, and A. Russo. Virtual element method for general second-order elliptic problems on polygonal meshes. *Math. Models Methods Appl. Sci.*, 26(4):729–750, 2016.

[15] L. Beirão da Veiga, F. Dassi, and A. Russo. High-order virtual element method on polyhedral meshes. *Comput. Math. Appl.*, 74(5):1110–1122, 2017.

[16] L. Beirão Da Veiga, F. Dassi, and G. Vacca. The Stokes complex for virtual elements in three dimensions. *Math. Models Methods Appl. Sci.*, 30(3):477–512, 2020.

[17] L. Beirão da Veiga, C. Lovadina, and G. Vacca. Divergence free virtual elements for the Stokes problem on polygonal meshes. *ESAIM Math. Model. Numer. Anal.*, 51(2):509–535, 2017.

[18] L. Beirão da Veiga and G. Manzini. Residual a posteriori error estimation for the virtual element method for elliptic problems. *ESAIM Math. Model. Numer. Anal.*, 49(2):577–599, 2015.

[19] L. Beirão da Veiga, G. Manzini, and L. Mascotto. A posteriori error estimation and adaptivity in *hp* virtual elements. *Numer. Math.*, 143(1):139–175, 2019.

[20] S. Berrone and A. Borio. A residual a posteriori error estimate for the virtual element method. *Math. Models Methods Appl. Sci.*, 27(8):1423–1458, 2017.

[21] S. C. Brenner and L. R. Scott. *The Mathematical Theory of Finite Element Methods*. Springer, New York, 3rd edition, 2008.

[22] F. Brezzi, R. S. Falk, and L. D. Marini. Basic principles of mixed virtual element methods. *ESAIM Math. Model. Numer. Anal.*, 48(4):1227–1240, 2014.

[23] F. Brezzi and L. D. Marini. Virtual element methods for plate bending problems. *Comput. Methods Appl. Mech. Engrg.*, 253:455–462, 2013.

[24] A. Cangiani, E. H. Georgoulis, T. Pryer, and O. J. Sutton. A posteriori error estimates for the virtual element method. *Numer. Math.*, 137(4):857–893, 2017.

[25] A. Cangiani, G. Manzini, and O. J. Sutton. Conforming and nonconforming virtual element methods for elliptic problems. *IMA J. Numer. Anal.*, 37(3):1317–1354, 2017.

[26] L. Chen. iFEM: an integrated finite element method package in MATLAB. Technical report, University of California at Irvine, 2009.

[27] L. Chen and X. Huang. Nonconforming virtual element method for $2m$-th order partial differential equations in R^n. *Math. Comput.*, 89(324):1711–1744, 2020.

[28] H. Chi, L. Beirão da Veiga, and G. H. Paulino. Some basic formulations of the virtual element method (VEM) for finite deformations. *Comput. Methods Appl. Mech. Engrg.*, 318:148–192, 2017.

[29] H. Chi, L. Beirão da Veiga, and G. H. Paulino. A simple and effective gradient recovery scheme and a posteriori error estimator for the virtual element method (vem). *Comput. Methods Appl. Mech. Engrg.*, 347:21–58, 2019.

[30] C. Chinosi and L. D. Marini. Virtual element method for fourth order problems: L^2-estimates. *Comput. Math. Appl.*, 72(8):1959–1967, 2016.

[31] M. Cihan, B. Hudobivnik, F. Aldakheel, and P. Wriggers. 3D mixed virtual element formulation for dynamic elasto-plastic analysis. *Comput. Mech.*, 68(3):581–598, 2021.

[32] R. S. Falk. Nonconforming finite element methods for the equations of linear elasticity. *Math. Comp.*, 57(196):529–550, 1991.

[33] K. Feng and Z. Shi. *Mathematical Theory of Elastic Structures*. Springer-Verlag, Berlin, 1996.

[34] A. Gain, G. Paulino, S. Leonardo, and I. Menezes. Topology optimization using polytopes. *Comput. Methods Appl. Mech. Engrg.*, 293:411–430, 2015.

[35] A. L. Gain, C. Talischi, and G. H. Paulino. On the virtual element method for three-dimensional linear elasticity problems on arbitrary polyhedral meshes. *Comput. Methods Appl. Mech. Engrg.*, 282:132–160, 2014.

[36] J. Huang and S. Lin. A C^0P_2 time-stepping virtual element method for linear wave equations on polygonal meshes. *Electron. Res. Arch.*, 28(2):911–933, 2020.

[37] J. Huang and S. Lin. A posteriori error analysis of a non-consistent virtual element method for reaction diffusion equations. *Appl. Math. Lett.*, 122:Paper No. 107531, 10, 2021.

[38] J. Huang, S. Lin, and Y. Yu. A novel locking-free virtual element method for linear elasticity problems. *arXiv:2112.13848*, 2021.

[39] X. Huang. Nonconforming virtual element method for $2m$th order partial differential equations in $\mathbb{R}^n$ with $m > n$. *Calcolo*, 57(4):42, 2020.

[40] R. Kouhia and R. Stenberg. A linear nonconforming finite element method for nearly incompressible elasticity and Stokes flow. *Comput. Methods Appl. Mech. Engrg.*, 124(3):195–212, 1995.

[41] D. Y. Kwak and H. Park. Lowest-order virtual element methods for linear elasticity problems. *Comput. Methods Appl. Mech. Engrg.*, 390:Paper No. 114448, 2022.

[42] A. Ortiz-Bernardin, C. Alvarez, N. Hitschfeld-Kahler, A. Russo, R. Silva-Valenzuela, and E. Olate-Sanzana. Veamy: an extensible object-oriented C++ library for the virtual element method. *Numer. Algorithms*, 82:1189–1220, 2019.

[43] K. Park, H. Chi, and G. H. Paulino. Numerical recipes for elastodynamic virtual element methods with explicit time integration. *Internat. J. Numer. Methods Engrg.*, 121(1):1–31, 2020.

[44] O. J. Sutton. The virtual element method in 50 lines of MATLAB. *Numer. Algorithms*, 75(4):1141–1159, 2017.

[45] C. Talischi, G. H. Paulino, A. Pereira, and I. F. M. Menezes. Polymesher: a general-purpose mesh generator for polygonal elements written in Matlab. *Struct. Multidiscip. Optim.*, 45(3):309–328, 2012.

[46] G. Vacca. Virtual element methods for hyperbolic problems on polygonal meshes. *Comput. Math. Appl.*, 74(5):882–898, 2017.

[47] G. Vacca and L. Beirão da Veiga. Virtual element methods for parabolic problems on polygonal meshes. *Numer. Methods Partial Differential Equations*, 31(6):2110–2134, 2015.

[48] F. Wang and H. Wei. Virtual element method for simplified friction problem. *Appl. Math. Lett.*, 85:125–131, 2018.

[49] F. Wang, B. Wu, and W. Han. The virtual element method for general elliptic hemivariational inequalities. *J. Comput. Appl. Math.*, 389:Paper No. 113330, 19, 2021.

[50] F. Wang and J. Zhao. Conforming and nonconforming virtual element methods for a Kirchhoff plate contact problem. *IMA J. Numer. Anal.*, 41(2):1496–1521, 2021.

[51] B. Wu, F. Wang, and W. Han. Virtual element method for a frictional contact problem with normal compliance. *Commun. Nonlinear Sci. Numer. Simul.*, 107:Paper No. 106125, 13 pp., 2022.

[52] Y. Yu. Implementation of polygonal mesh refinement in MATLAB. *arXiv:2101.03456*, 2021.

[53] Y. Yu. A lowest-order locking-free nonconforming virtual element method based on the reduced integration technique for linear elasticity problems. *arXiv:2112.13378v2*, 2021.

[54] B. Zhang, J. Zhao, Y. Yang, and S. Chen. The nonconforming virtual element method for elasticity problems. *J. Comput. Phys.*, 378:394–410, 2019.

[55] J. Zhao, S. Chen, and B. Zhang. The nonconforming virtual element method for plate bending problems. *Math. Models Methods Appl. Sci.*, 26(9):1671–1687, 2016.

[56] J. Zhao, B. Zhang, S. Chen, and S. Mao. The Morley-type virtual element for plate bending problems. *J. Sci. Comput.*, 76(1):610–629, 2018.

[57] J. Zhao, B. Zhang, S. Mao, and S. Chen. The divergence-free nonconforming virtual element for the Stokes problem. *SIAM J. Numer. Anal.*, 57(6):2730–2759, 2019.